Named Async Locks

As application complexity grows, you will eventually need a scalable way to serialize (or "singulate" to not get confused with object serialization) queries and updates to resources.

For Example

You might have a document platform, that provides collaborative editing.
In which case, there may be multiple clients submitting changes, simultaneously.
And, these changes all have to be incorporated as a serialized/singulated list of individual changes to a document.

Since multiple such documents may be in-flux, the easiest way to perform changes, one at a time, to each is with asynchronous semaphore.

Below is how to manage changes to these documents, using a process-wide set of named, asynchronous locks.

Implementation

For our document editing use case, above, we will use this library:

https://github.com/LeeWhite187/AsyncKeyedLock

> NOTE: The above repo is actually a fork of the original, here:
> https://github.com/MarkCiliaVincenti/AsyncKeyedLock

Install AsyncKeyedLock from Nuget.

It's currently published as .NET Standard 2.0, so it's quite compatible across .NET versions.

Best way to use this library is to register it with DI, on startup, with this:

```
// Setup the global session update serializer, here...
{
    // We want the session serializer to allow one and only one thread/task to update
    //    a particular document (or other top-level entity) at a time...
    services.AddSingleton(sp =>
```

```
    {
        // Create an instance that allows only one thread/task in at a time...
        var asyncKeyedLocker1 = new AsyncKeyedLocker<string>(new AsyncKeyedLockOptions(maxCount: 1));
        // Return it as the singleton...
        return asyncKeyedLocker1;
    });
}
```

The above will register the key locker as a singleton, so it can maintain a list of named locks, across the process.

> NOTE: We set the key type to 'string', so that we can name locks by hash string, Guid string, document name, or whatever unique entity identifier we want.

You can now inject it into services, like this:

```
public class DocumentServices
{
    /// <summary>
    /// This is the local reference to the process's session lock, that we will use to serialize updates to each
document.
    /// </summary>
    private AsyncKeyedLock.AsyncKeyedLocker<string> _doclock;

    public DocumentServices(AsyncKeyedLocker<string> doclock)
    {
        this._doclock = doclock;
    }

    ...
}
```

The above will inject the locker singleton into our document service instance.

Now. Wherever your service makes changes to a particular document, you wrap that block of code with a using statement that retrieves the particular document's lock instance, like this:

```
public async Task<(int res, DocumentDTO_v1? data)> Update_Document(DocumentDTO_v1 dto)
{
    if (dto == null)
    {
```

```
        // Nothing given.
        return (-1, null);
    }
    if (dto.id == Guid.Empty)
    {
        // Empty Id.
        return (-1, null);
    }
    // From here down, we have a valid guid for an id.

    // Changes to the specific document, are serialized in the below async lock.
    using (await this._doclock.LockAsync(dto.id.ToString()))
    {
      ... DO ANY CHANGES TO THE DOCUMENT, SAFELY, HERE...
    }
    // Lock is released at end of the using statement, via the implicit Dispose()

    return (1, dto);
}
```

Revision #3
Created 19 March 2025 15:58:23 by glwhite
Updated 6 May 2026 15:39:20 by glwhite