**Dependency Injection Lifetime Management in Sitecore**

Sitecore 8.2 and above implements dependency injection based on [ASP.NET Core DI](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-3.0). Service registration is handled by using the IServiceConfigurator interface. When we opt for third-party DI modules like Simple Injector, you can have your own customised scoped lifetime. This post is a quick reference for service lifetimes when you implement DI to Sitecore by using [the out-of-box Sitecore DI](https://doc.sitecore.com/developers/90/sitecore-experience-manager/en/dependency-injection.html). If you are interested in implementing DI in this way, you can check out my post [here](https://www.mydatahack.com/implementing-dependency-injection-in-sitecore/).

Sitecore dependency injection implements the same lifetimes as the ASP.NET Core, Transient, Scoped and Singleton.

**Transient**

Transient services can be registered by [AddTransient](https://docs.microsoft.com/en-gb/dotnet/api/microsoft.extensions.dependencyinjection.servicecollectionserviceextensions.addtransient?view=aspnetcore-3.0). They are created each time they are requested from the service container.

This is typically used for stateless services such as controllers.

**Scoped**

Scoped services can be registered by AddScoped. These are created per request (connection) and get disposed when the request ends. Classes belong to foundation projects are good candidate for this one.

To support scoped per request lifetime, we need to make sure SitecorePerRequestScopeModule is enabled as per [the documentation](https://doc.sitecore.com/developers/90/sitecore-experience-manager/en/dependency-injection.html). This can be found in web.config. When you install Sitecore, this option is included by default. It is, however, always good to check. In the config file, it will look like this:

<script src="https://gist.github.com/mydatahack/9f4802bfe4caf5130f653e46f94b7eb3.js"></script>

Good candidates for Scoped lifetimes are account manager classes where you authenticate user and bring user information. In fact, a majority of classes in Foundation project can have Scoped lifetimes.

**Singleton**

Singleton services can be registered by [AddSingleton](https://docs.microsoft.com/en-gb/dotnet/api/microsoft.extensions.dependencyinjection.servicecollectionserviceextensions.addsingleton?view=aspnetcore-3.0). They are created the first time they are requested and every subsequent request uses the same instance. It gets disposed when the parent container gets disposed, which usually means the application shut down.

When your Sitecore solutions have a class instantiated by .Instance() and injected into other classes, this is where you can use Singleton. For example, if you are using feature flags for your solutions, the class to get feature flag state can be a singleton.

You definitely cannot use Singleton for, say, account manager classes that handle user authentication. If you are running an e-commerce site, all the sales can potentially be attributed to a single user!

**Golden Rule**

The single most important rule is that **services of shorter scopes can only depend on services of longer or equivalent scope**. Violation of this rule will make the application go nuts!

When we implement lifetimes, we need to be extra careful about the dependencies. For example, you cannot resolve a scoped service from a singleton as the state will be incorrect from the subsequent request. If you have a scoped with dependency to a transient, the scoped class will loose its transient class when the request ends and result in error.

If you are not sure about which lifetime to use, I recommend to read this [fantastic blog post](https://www.coreysmith.co/sitecore-dependency-injection-scoped-services/) by Corey Smith, describing services lifetimes.