**DotNetStarter.Abstractions.RegistrationAttribute**

using the DotNetStarter.Abstractions.RegistrationAttribute on the implementation class. This works for registering type implementations but doesn’t support delegate or instance-based registrations.

internal interface IService { }

[Registration(typeof(IService), Lifecycle.Transient)]

internal class IImplemetation : IService { }

## DotNetStarter.Abstractions.ILocatorConfigure

## Requires a class that implements DotNetStarter.Abstractions.ILocatorConfigure with a StartupModuleAttribute decoration. The types that implement this interface also need empty constructors as locator with its underlying container have not been configured, there will be nothing to inject.

## configurating locator/container services which requires empty constructors for implementations

[StartupModule(typeof(RegistrationConfiguration))]

public class ExampleModule : ILocatorConfigure

{

internal static bool SupportsServiceRemoval = false;

internal static bool RegisterException = false;

public void Configure(ILocatorRegistry container, ILocatorConfigureEngine engine)

{

try

{

container.Add(typeof(IRemove), typeof(Remove), null, Lifecycle.Singleton);

}

catch(Exception ex)

{

RegisterException = true;

}

//if (container is ILocatorRegistryWithRemove removable)

//{

// removable.Remove(typeof(IRemove));

// SupportsServiceRemoval = true;

//}

container.Add<BaseTest, BaseImpl>(lifecycle: Lifecycle.Singleton);

container.Add(typeof(IFooTwo), locator => FooTwoFactory.CreateFoo(), Lifecycle.Transient);

container.Add<ITest, Test>(lifecycle: Lifecycle.Transient);

}

}

## DotNetStarter.Abstractions.ILocatorConfigure

## During the application startup configure the startup builder, that registers the types.

Assembly scanning occurs to discover service registrations (RegistrationAttribute) and IStartupModule/ILocatorConfigure modules.

var builder = DotNetStarter.Configure.StartupBuilder.Create();

builder

// ability to customize environment object, which can be used make registration decisions based on environment

.UseEnvironment(new StartupEnvironment("Development", ""))

// configure the assemblies to scan

.ConfigureAssemblies(assemblies =>

{

assemblies

// Filters assemblies for ones using the [assembly: DotNetStarter.Abstractions.DiscoverableAssembly]

.WithDiscoverableAssemblies() // for netstandard 1.0 projects, an initial list of assemblies must be provided

.WithAssemblyFromType<RegistrationConfiguration>()

.WithAssembliesFromTypes(typeof(StartupBuilder));

})

.ConfigureStartupModules(modules =>

{

modules

// ability to manually add ILocatorConfigure modules after the scanned ones

.ConfigureLocatorModuleCollection(configureModules =>

{

configureModules.Add(new ExampleModule ());

});

})

// override default objects

.OverrideDefaults(d => d.UseLocatorRegistryFactory(new DotNetStarter.Locators.DryIocLocatorFactory())) //uses DryIoc

.Build() // configures the ILocator

.Run(); // Runs IStartupModule registrations;

**Resolve registration**

var loc = DotNetStarter.ApplicationContext.Default.Locator.Get<IService>();