‹ Previous Next ›

Unity Container: Method Injection

In the previous chapter, we learned about property injection. Here, we will learn about method injection using Unity container.

In the method injection, dependencies are provided through method parameters. Visit Dependency Injection chapter to learn more about the method injection.

Let's understand how we can perform method injection using Unity container. Consider the following example classes.

Example: C# Copy

```
public interface ICar
{
    int Run();
}

public class BMW : ICar
{
    private int _miles = 0;

    public int Run()
    {
                return ++_miles;
    }
}

public class Ford : ICar
{
    private int _miles = 0;

    public int Run()
    {
        return ++_miles;
    }
}

public class Audi : ICar
{
    private int _miles = 0;

    public int Run()
    {
        return ++_miles;
    }

}
public class Driver
{
    private ICar _car = null;

    public Driver()
    {
    }

    public void UseCar(ICar car) {
        _car = car;
```

```
    }

    public void RunCar()
    {
        Console.WriteLine("Running {0} - {1} mile ", _car.GetType().Name, _car.Run());
    }
}
```

As you can see in the above sample classes, the `Driver` class includes the method `UseCar()` to set the object of type `ICar`. Here, we have taken a simple method example. However, you can also use interface-based method injection, explained in the Dependency Injection chapter.

Method injection in Unity can be implemented in two ways:

1. Using the [InjectionMethod] attribute
2. Using run-time configuration

InjectionMethod Attribute

For the method injection, we need to tell the Unity container which method should be used for dependency injection. So, we need to decorate a method with the [InjectionMethod] attribute as shown in the following `Driver` class.

Example: Method Injection - C# Copy

```
public class Driver
{
    private ICar _car = null;

    public Driver()
    {
    }
    [InjectionMethod]
    public void UseCar(ICar car) {
        _car = car;
    }

    public void RunCar()
    {
        Console.WriteLine("Running {0} - {1} mile ", _car.GetType().Name, _car.Run());
    }
}
```

We can implement method injection in Unity container as shown below.

Example: Method Injection - C# Copy

```
var container = new UnityContainer();
container.RegisterType<ICar, BMW>();

var driver = container.Resolve<Driver>();
driver.RunCar();
```

Output:

```
Running BMW - 1 mile
```

Run-time Configuration

Unity container allows us to configure method injection with the `RegisterType()` method if a method is not marked with the `[InjectionMethod]` attribute. Pass an object of the InjectionMethod class in the RegisterType() method to specify a method name and a parameter value.

Note:

The InjectionMethod is derived from the InjectionMember Class. The `InjectionMember` is an abstract class which can be used to configure the injection type. There are three subclasses of InjectionMembers: InjectionConstruction to configure construction injection, InjectionProperty to configure property injection and InjectionMethod to configure method injection.

```
var container = new UnityContainer();

//run-time configuration
container.RegisterType<Driver>(new InjectionMethod("UseCar", new Audi()));

//to specify multiple parameters values
container.RegisterType<Driver>(new InjectionMethod("UseCar", new object[] { new Audi() }));

var driver = container.Resolve<Driver>();
driver.RunCar();
```

Output:

```
Running Audi - 1 Mile
```

As you can see in the above example, `container.RegisterType<driver>(new InjectionMethod("UseCar", new Audi()))` registers the `Driver` class by passing an object of the `InjectionMethod` that specifies the method name and the parameter value. So, Unity container will inject an object of Audi when we resolve it using `container.Resolve<Driver>()`.