# Unity Container: Constructor Injection

In the previous chapter, we learned about registering and resolving types using Unity container. Here, you will learn how Unity container performs constructor injection.

Construction injection is a type of Dependency Injection where dependencies are provided through a constructor. Visit the <u>Dependency Injection</u> chapter to learn more about it.

We learned about the Resolve() method in the previous chapter. By default, Resolve<T>() performs construction injection to inject dependencies and returns an object of the specified type. Let's take the same examples from the previous chapter.

```
Example: C#
                                                                                              企 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(ICar car)
        _car = car;
    public void RunCar()
```

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

As you can see above, the Driver class accepts an object of type ICar in the constructor. So, Unity container will inject dependencies via the constructor as shown below.

```
Example: Construction Injection using Unity Container - C#

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

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

```
Output:

Running BMW - 1 mile
```

In the above example, container.RegisterType<ICar, BMW>() maps ICar to BMW. It means that whenever Unity container needs to inject an object of type ICar, it will create and inject an object of the BMW class. The container.Resolve<driver>() method will create and return an object of the Driver class by passing an object of ICar into the constructor. As we have mapped ICar to BMW, it will create and inject a BMW object to a constructor of the Driver class and return an object of the Driver class.

Thus, by default, the Resolve() method performs constructor injection while resolving types.

### Multiple Parameters

You can also inject multiple parameters in the constructor. Consider the following example.

```
public interface ICarKey {
}

public class BMWKey : ICarKey
{
}

public class AudiKey : ICarKey
{
}

public class FordKey : ICarKey
{
}

public class FordKey : ICarKey
{

private ICar _car = null;
 private ICarKey _key = null;

public Driver(ICar car, ICarKey key)
{
    __car = car;
}
```

Thus, you can now register ICar and ICarKey with Unity container and inject both the parameters as shown below.

```
Example: Constructor Injection for Multiple Parameters - C#

var container = new UnityContainer();

container.RegisterType<ICar, Audi>();

container.RegisterType<ICarKey, AudiKey>();

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

driver.RunCar();
```

```
Output:

Running Audi with AudiKey - 1 mile
```

## **Multiple Constructors**

If a class includes multiple constructors, then use the [InjectionConstructor] attribute to indicate which constructor to use for construction injection.

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

    [InjectionConstructor]
    public Driver(ICar car)
    {
        _car = car;
    }

    public Driver(string name)
    {
     }

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

As you can see, the Driver class includes two constructors. So, we have used the [InjectionConstructor] attribute to indicate which constructor to call when resolving the Driver class.

You can configure the same thing as above at run time instead of applying the [InjectionConstructor] attribute by passing an object of the InjectionConstructor in the RegisterType() method, as shown below.

```
container.RegisterType<Driver>(new InjectionConstructor(new Ford()));

//or

container.RegisterType<ICar, Ford>();
container.RegisterType<Driver>(new InjectionConstructor(container.Resolve<ICar>()));
```

## Primitive Type Parameter

Unity also injects primitive type parameters in the constructor. Consider the following Driver class with primitive type parameters in the constructor.

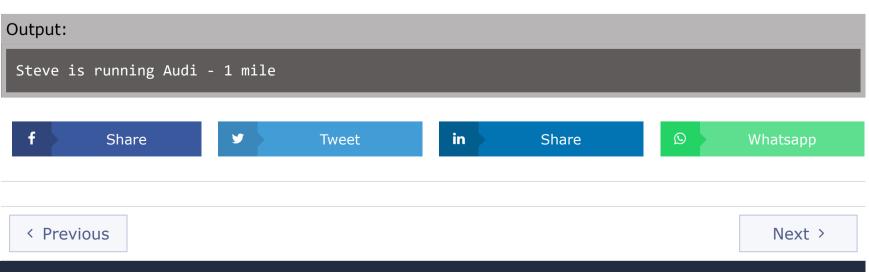
Use the <u>InjectionConstructor</u> class to configure the constructor's parameter values. Pass an object of the <u>InjectionConstructor</u> class in the RegisterType() method to specify multiple parameters values.

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

```
var container = new UnityContainer();

container.RegisterType<Driver>(new InjectionConstructor(new object[] { new Audi(), "Steve"
}));

var driver = container.Resolve<Driver>(); // Injects Audi and Steve
driver.RunCar();
```



### TUTORIALSTEACHER.COM **TUTORIALS** > AngularJS 1 > ASP.NET Core TutorialsTeacher.com is optimized for learning web technologies step by step. Examples might > ASP.NET MVC > <u>Node.js</u> be simplified to improve reading and basic > <u>D3.js</u> > <u>IoC</u> understanding. While using this site, you agree to have read and accepted our terms of use > Web API > <u>JavaScript</u> and <u>privacy policy</u>. > <u>C#</u> > <u>jQuery</u> > <u>LINQ</u> > <u>Sass</u> > Entity Framework > <u>Https</u> **E-MAIL LIST** Subscribe to TutorialsTeacher email list and get latest updates, tips & tricks on C#, .Net, JavaScript, jQuery, AngularJS, Node.js to your inbox. Email address GO We respect your privacy. HOME PRIVACY POLICY TERMS OF USE ADVERTISE WITH US © 2020 TutorialsTeacher.com. All Rights Reserved.