FREE eBook

LEARNING autofac

Free unaffiliated eBook created from
Stack Overflow contributors.

#autofac

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: autofac

It is an unofficial and free autofac ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official autofac.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with autofac

Remarks

Autofac is an IoC container for Microsoft .NET 4.5, Silverlight 5, Windows Store apps, and Windows Phone 8 apps. It manages the dependencies between classes so that applications stay easy to change as they grow in size and complexity. This is achieved by treating regular .NET classes as components.

From Wikipedia:
In software engineering, inversion of control (IoC) is a design principle in which custom-written portions of a computer program receive the flow of control from a generic framework. A software architecture with this design inverts control as compared to traditional procedural programming: in traditional programming, the custom code that expresses the purpose of the program calls into reusable libraries to take care of generic tasks, but with inversion of control, it is the framework that calls into the custom, or task-specific, code.

- Inversion of Control

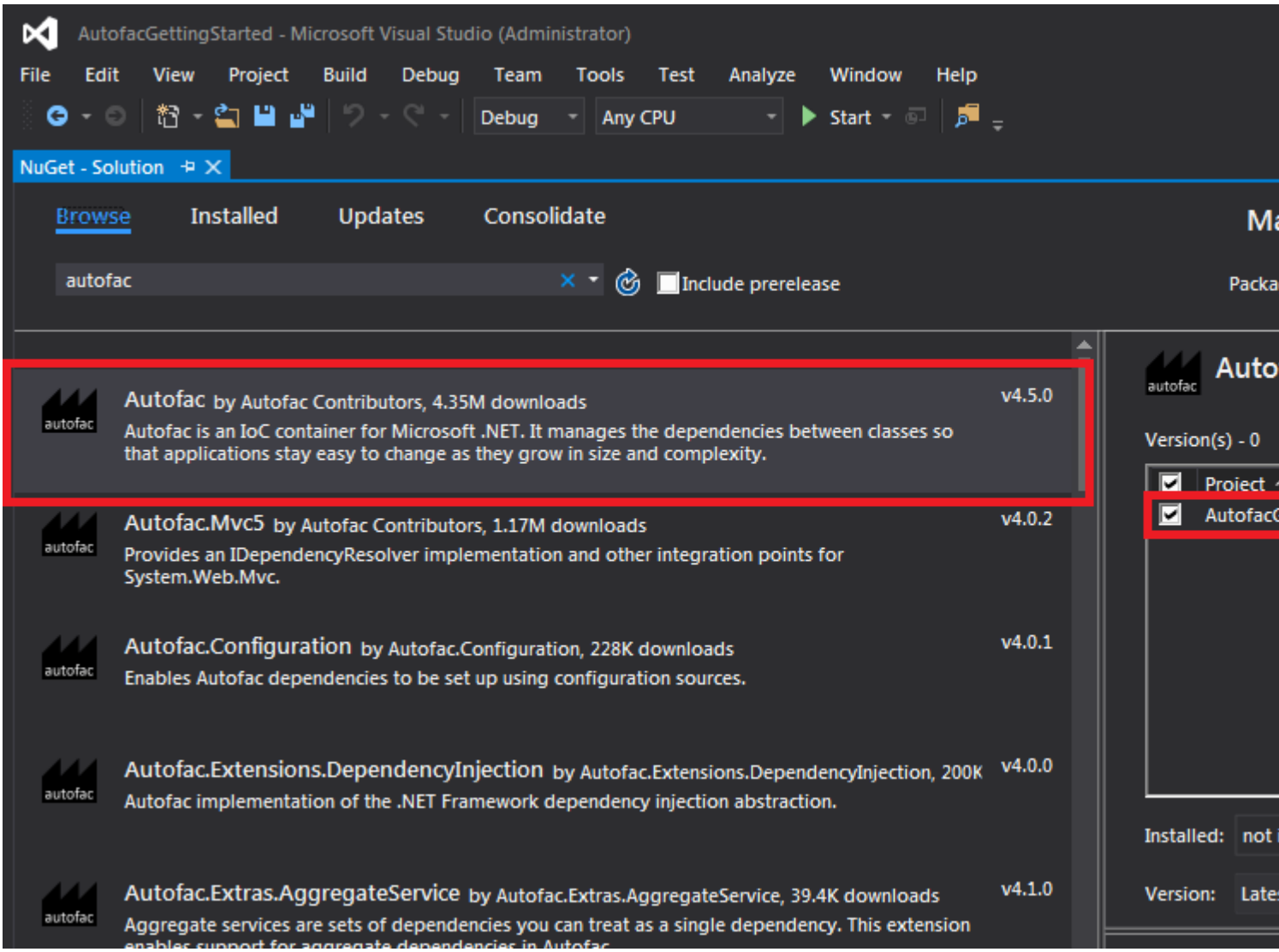
Examples

Installing Autofac

To use Autofac in your project, all you have to do is install Autofac from NuGet Package Manager. Open the solution that want to use Autofac in, then select `Manager NuGet Packages for Solution...` by going to:

```
Tools -> NuGet Package Manager -> Manager NuGet Packages for Solution...
```

In the `NuGet-Solution` tab, type in "Autofac" in the search box. Make sure you are in the "Browse" section. Install the first option as shown in the image below (take note of the marked regions in the image):



Installing through NuGet will automatically add Autofac in the References of the projects which were selected during installation.

Take a look at the official documentation.

Setting up Autofac

This example will show how get started with *Inverion of Control* using Autofac with a relatively simple project, closely following the official getting started docs.

1. Create a console application from `File -> New -> Project -> Console Application`
2. Install Autofac for this project. You can take a look here Installing Autofac
3. Add 2 interfaces and 2 classes, with the following names:

```
  Interfaces   |  Classes
---------------------------
IOutput        | ConsoleOutput (implementing IOutput)
IDateWriter    | TodayWriter (implementing IDateWriter)
```

For simplicity, the using statements and namespaces are not shown.

IOuput.cs

```
public interface IOutput
{
    void Write(string content);
}
```

ConsoleOutput.cs

```
public class ConsoleOutput : IOutput
{
    public void Write(string content)
    {
        Console.WriteLine(content);
    }
}
```

IDateWriter.cs

```
public interface IDateWriter
{
    void WriteDate();
}
```

TodayWriter.cs

```
public class TodayWriter : IDateWriter
{
    private IOutput _output;

    public TodayWriter(IOutput output)
    {
        _output = output;
    }

    public void WriteDate()
    {
        _output.Write(DateTime.Today.ToShortDateString());
    }
}
```

So far the code has been plain and simple. Lets get to the part where automatic dependency injection takes place, which of course is being done by Autofac!

Replace the `Program` class in Program.cs file with this code (`Program` class is automatically created by Visual Studio at project creation. If it doesn't exist, go ahead and create one):

```
class Program
{
    private static IContainer Container { get; set; }

    static void Main(string[] args)
    {
        var builder = new ContainerBuilder();
        builder.RegisterType<ConsoleOutput>().As<IOutput>();
        builder.RegisterType<TodayWriter>().As<IDateWriter>();
        Container = builder.Build();

        WriteDate();
    }

    public static void WriteDate()
    {
        using (var scope = Container.BeginLifetimeScope())
        {
            var writer = scope.Resolve<IDateWriter>();
            writer.WriteDate();
        }
    }
}
```

When run, the output should be the current date in the console. You have successfully used Autofac in your project to inject dependencies automatically.

Here is what's going on under the hood:

1. At application startup, we are creating a `ContainerBuilder` and registering our *Components* with it. A component in simple terms is a .NET type that implements an interface, and thus exposes some *services*. Read Services vs. Components.

2. We then *register* our components (classes) with the services (interfaces) they expose. When registered, Autofac knows which instance of a class to create when an interface is to be *resolved*.

3. Finally, when we run the program:

    - The `WriteDate()` method (in `Main()`) asks Autofac for an `IDateWriter`.
    - Autofac sees that `IDateWriter` maps to `TodayWriter` so starts creating a `TodayWriter`.
    - Autofac sees that the `TodayWriter` needs an `IOutput` in its constructor.
    - Autofac sees that `IOutput` maps to `ConsoleOutput` so creates a new `ConsoleOutput` instance.
    - Autofac uses the new `ConsoleOutput` instance to finish constructing the `TodayWriter`.
    - Autofac returns the fully-constructed `TodayWriter` for `WriteDate()` to consume.

Read Getting started with autofac online: https://riptutorial.com/autofac/topic/9595/getting-started-with-autofac

Credits

S. No	Chapters	Contributors
1	Getting started with autofac	Community, Snađoшfa