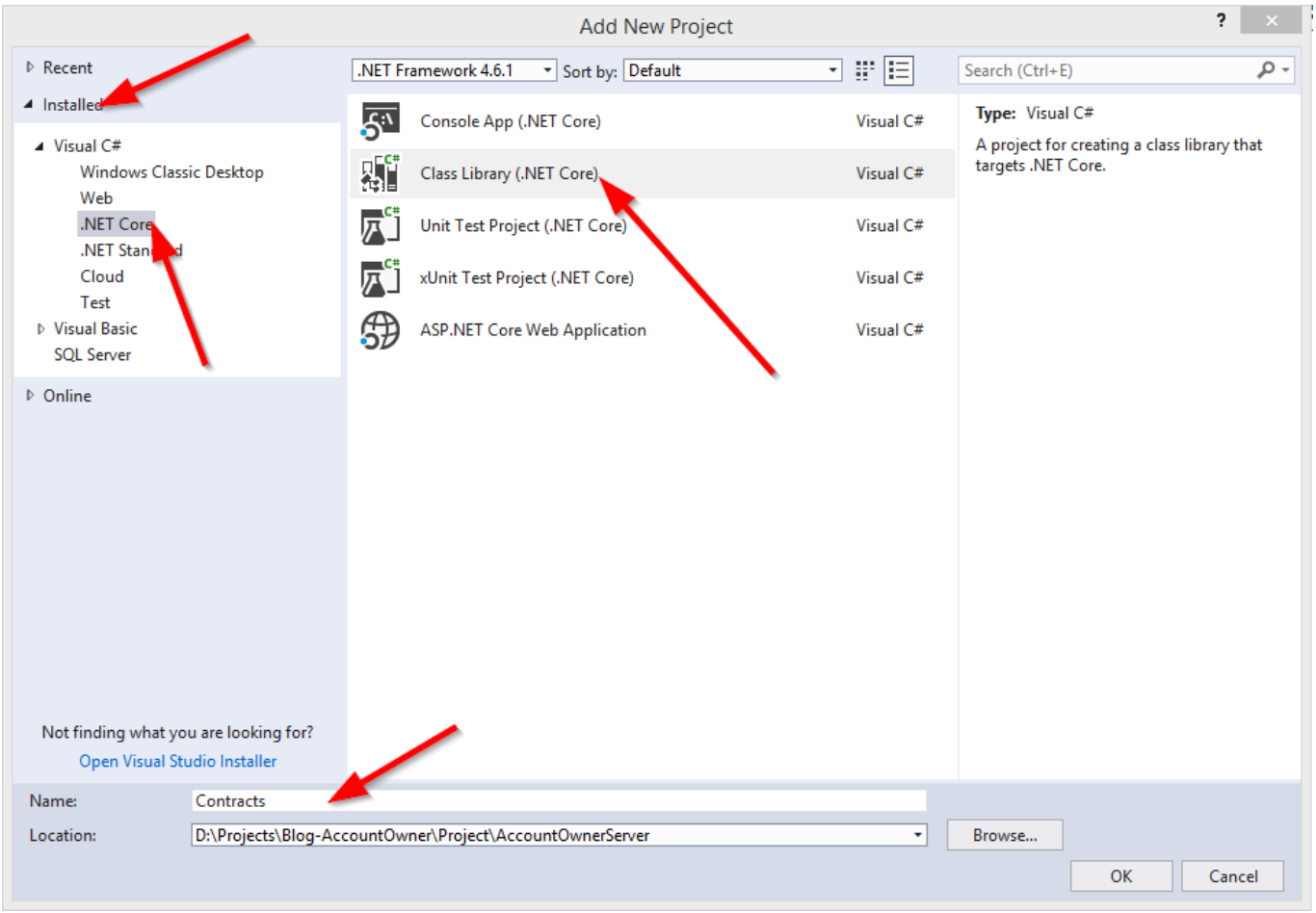
Logging With NLog

Creating Required Projects

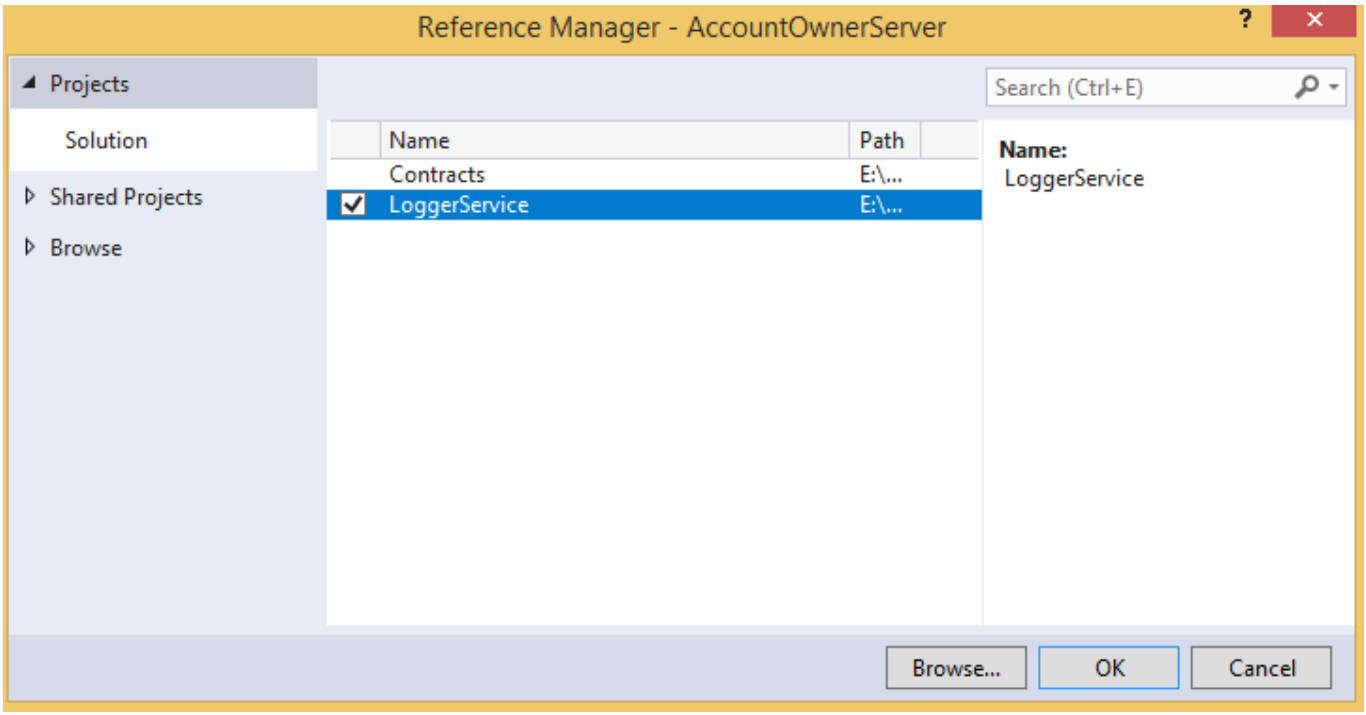
Let's create two new projects. Name the first one `Contracts`. You are going to store interfaces inside this project. Name the second one `LoggerService`. You are going to use it for the logger logic.

To create a new project, right click on the solution window, choose Add and then NewProject. Under the .NET Core, choose the Class Library (.NET Core) and name it `Contracts`.



Do the same thing for the second project, just name it `LoggerService`.

With these two projects in place, we need to reference the `LoggerService` project to the main project. In the main project inside solution explorer, right click on the Dependences and choose the AddReference. Under the Projects click the Solution and check this project:



Before we proceed to the implementation of the `LoggerService`, let's do one more thing.

In the `LoggerService` project, right-click on the Dependencies and then click on Add Reference. Inside check the `Contracts` checkbox to import its reference inside the `LoggerService` project. This would automatically add the `Contracts` reference to the main project because we already have a `LoggerService` referenced inside.

Creating the Interface and Installing NLog

Our logger service will contain four methods for logging:

- Info messages
- Debug messages
- Warning messages
- Error messages

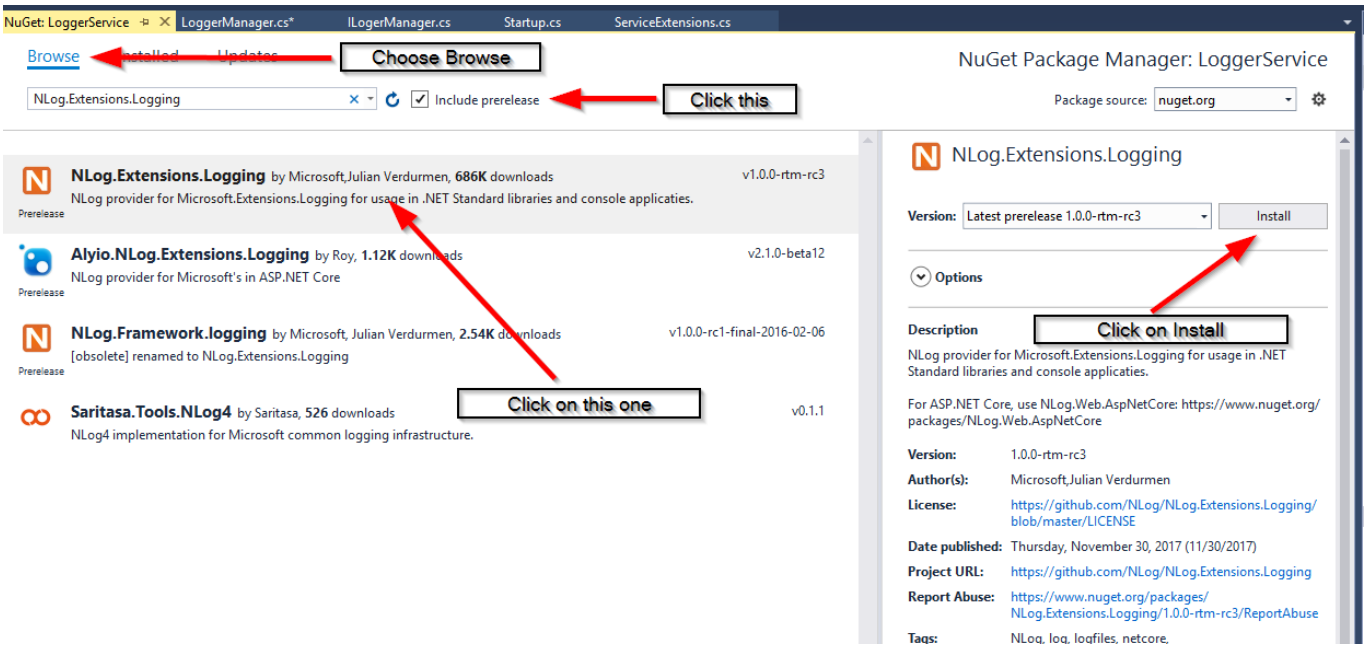
Consequently, we will create the interface `ILoggerManager` inside the `Contracts` project containing those four method definitions.

Add the following code to the `ILoggerManager` interface:

```
namespace Contracts
{
    public interface ILoggerManager
    {
        void LogInfo(string message);
        void LogWarn(string message);
        void LogDebug(string message);
        void LogError(string message);
    }
}
```

Before you implement this interface inside the `LoggerService` project, you need to install the `NLog` library in the `LoggerService` project. `NLog` is a logging platform for .NET.

In the `LoggerService` project, right click on Dependences and choose Manage NuGet Packages. After the NuGet window appears, just follow the steps:



Implementing Interface and Nlog.Config File

In the `LoggerService` project, create the new class `LoggerManager`.

Let's modify the `LoggerManager` class:

```
using Contracts;
using NLog;
using System;

namespace LoggerService
{
    public class LoggerManager : ILoggerManager
    {
        private static ILogger logger = LogManager.GetCurrentClassLogger();

        public LoggerManager()
        {
        }

        public void LogDebug(string message)
        {
            logger.Debug(message);
        }

        public void LogError(string message)
        {
            logger.Error(message);
```

```
        }

        public void LogInfo(string message)
        {
            logger.Info(message);
        }

        public void LogWarn(string message)
        {
            logger.Warn(message);
        }
    }
}
```

Now, we need to configure it and inject it into the `Startup` class in the `ConfigureServices` method.

`NLog` needs to have information about the folder to create log files in, what the name of these files will be and what a minimum level of logging is. Therefore, you need to create one text file in the main project with the name `nlog.config` and to populate it as in the example below. You need to change the path of the internal log and filename parameters to your paths.

```
<?xml version="1.0" encoding="utf-8" ?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      autoReload="true"
      internalLogLevel="Trace"

internalLogFile="C:\Users\paul\source\repos\Project\internal_logs\internallog.txt"
>

  <targets>
    <target name="logfile" xsi:type="File"

fileName="C:/Users/paul/source/repos/Project/logs/${shortdate}_logfile.txt"
            layout="${longdate} ${level:uppercase=true} ${message}"/>
  </targets>

  <rules>
    <logger name="*" minlevel="Debug" writeTo="logfile" />
  </rules>
</nlog>
```

Configuring Logger Service for Logging Messages

Setting up the configuration for a logger service is quite easy.

First, you must update the constructor in the `Startup` class:

```
public Startup(IConfiguration configuration)
{
    LogManager.LoadConfiguration(String.Concat(Directory.GetCurrentDirectory(),
"/nlog.config"));
    Configuration = configuration;
}
```

Secondly, you need to add the logger service inside the .NET Core's IOC container.

```
public static void ConfigureLoggerService(this IServiceCollection services)
{
    services.AddSingleton<ILoggerManager, LoggerManager>();
}
```

Lastly, in the `ConfigureServices` method, invoke this extension method, right above `services.AddMvc()`:

```
services.ConfigureLoggerService();
```

Every time you want to use a logger service, all you need to do is to inject it into the constructor of the class that is going to use that service. .NET Core will serve you that service from the IOC container and all of its features will be available to use. This type of injecting objects is called Dependency Injection.

DI, IOC and Logger Service Testing

What is Dependency Injection (DI) and what is IOC (Inversion of Control)?

Dependency injection is a technique for achieving loose coupling between objects and their dependencies. It means that rather than instantiating an object every time it is needed, we can instantiate it once and then serve it in a class, most often through a constructor method. This approach is also known as Constructor Injection.

In a system that is designed to use DI, you may find many classes requesting their dependencies via their constructor. In that case, it is helpful to have a class which will provide all instances to classes through the constructor. These classes are referred to as containers or more specific *Inversion of Control containers*. An IOC container is essentially a factory that is responsible for providing instances of types that are requested from it.

For logger service testing, we can use `ValuesController`, in the main project in the `Controllers` folder.

In the Solution Explorer, open the `Controllers` folder and open the `ValuesController` class. Let's modify it:

```
[Route("api/[controller]")]
[ApiController]
public class ValuesController : ControllerBase
{
    private ILoggerManager _logger;
```

```
        public ValuesController(ILoggerManager logger)
        {
            _logger = logger;
        }
        // GET api/values
        [HttpGet]
        public IEnumerable<string> Get()
        {
            _logger.LogInfo("Here is info message from our values controller.");
            _logger.LogDebug("Here is debug message from our values controller.");
            _logger.LogWarn("Here is warn message from our values controller.");
            _logger.LogError("Here is error message from our values controller.");

            return new string[] { "value1", "value2" };
        }
    }
```

Start the application and browse to http://localhost:5000/api/values. You should see an array of two strings. Now go to the folder that you specified in the `nlog.config` file and check out the result. You should see four lines logged inside that file:

```
2017-12-01 21:09:44.7578 INFO Here is info message from our values controller.
2017-12-01 21:09:44.8015 DEBUG Here is debug message from our values controller.
2017-12-01 21:09:44.8015 WARN Here is warn message from our values controller.
2017-12-01 21:09:44.8015 ERROR Here is error message from our values controller.
```

Conclusion

We now have a working logger service we can use throughout our application development.

Lessons learned:

- How to create an external service
- How to configure and use NLog library
- What are DI and IOC
- How to use them for service injection