

[< Previous](#)[Next >](#)

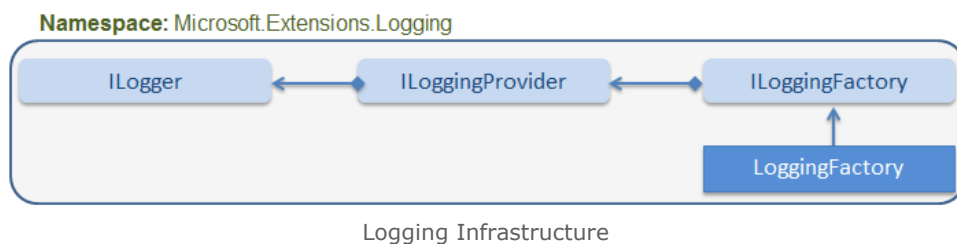
## ASP.NET Core - Logging

ASP.NET Core framework provides built-in supports for logging. However, we can also use third party logging provider easily in ASP.NET Core application.

Before we see how to implement logging in ASP.NET Core application, let's understand the important logging interfaces and classes available in ASP.NET Core framework. The following are built-in interfaces and classes available for logging under `Microsoft.Extensions.Logging` namespace .

1. ILoggerFactory
2. ILoggerProvider
3. ILogger
4. LoggerFactory

The following figure shows the relationship between logging classes.



Let's have an overview on each of them.

### ILoggerFactory

The `ILoggerFactory` is the factory interface for creating an appropriate `ILogger` type instance and also to add `ILoggerProvider` instance.

ILoggerFactory:

```
public interface ILoggerFactory : IDisposable
{
    ILogger CreateLogger(string categoryName);
    void AddProvider(ILoggerProvider provider);
}
```

ASP.NET Core framework includes built-in `LoggerFactory` class that implements `ILoggerFactory` interface. We can use it to add an instance of type `ILoggerProvider` and to retrieve `ILogger` instance for the specified category.

ASP.NET Core runtime creates an instance of `LoggerFactory` class and registers it for `ILoggerFactory` with the built-in IoC container when the application starts. Thus, we can use `ILoggerFactory` interface anywhere in your application. The IoC container will pass an instance of `LoggerFactory` to your application wherever it encounters `ILoggerFactory` type.

### ILoggerProvider

The `ILoggerProvider` manages and creates appropriate logger specified by logging category.

ILoggerProvider:



```
public interface ILoggerProvider : IDisposable
{
    ILogger CreateLogger(string categoryName);
}
```

We can create our own logging provider by implementing `ILoggerProvider` interface.

## ILogger

ILogger interface includes methods for logging to the underlying storage.

ILogger:

```
public interface ILogger
{
    void Log<TState>(LogLevel logLevel, EventId eventId, TState state, Exception exception, Func<TState, Exception, string> formatter);
    bool IsEnabled(LogLevel logLevel);
    IDisposable BeginScope<TState>(TState state);
}
```

## Built-in Logging Providers

There are different logging providers available as NuGet packages which we can use to send log output to the different medium such as console, debug window, EventSource etc. ASP.NET Core ships with the following providers:

1. [Console](#)
2. Debug
3. EventSource
4. EventLog
5. TraceSource
6. Azure App Service

Let's have an overview of Console logger.

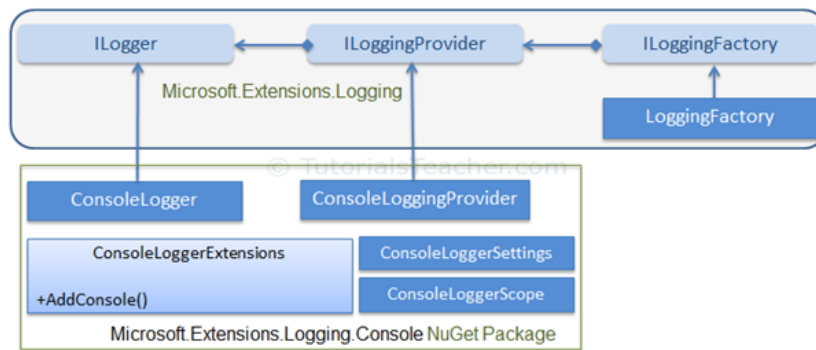
## Console Logger

The `Microsoft.Extensions.Logging.Console` package includes logging classes which sends log output to the console. The following are important classes for console logging.

1. ConsoleLoggingProvider
2. ConsoleLogger
3. ConsoleLoggerExtension
4. ConsoleLoggerSettings
5. ConsoleLoggerScope

The following figure illustrates the console logger classes.





Logging Infrastructure

As you can see in the above figure, the `ConsoleLogger` implements `ILogger` and `ConsoleLoggerProvider` implements `ILoggerProvider`. The `ConsoleLoggerExtensions` class includes extension method `AddConsole()` which adds console logger to `LoggerFactory`.

Now, let's use console logger to display log output to the console in both ASP.NET Core 1.x and 2.x application.

### Step 1 - Install NuGet Package

#### ASP.NET Core 1.x:

Include `Microsoft.Extensions.Logging.Console` dependencies in `project.json` if it is not added already.

```

"dependencies": {
  "Microsoft.NETCore.App": {
    "version": "1.0.1",
    "type": "platform"
  },
  "Microsoft.AspNetCore.Diagnostics": "1.0.0",
  "Microsoft.AspNetCore.Server.IISIntegration": "1.0.0",
  "Microsoft.AspNetCore.Server.Kestrel": "1.0.1",
  "Microsoft.Extensions.Logging.Console": "1.0.0"
}

```

Visual Studio will restore the packages automatically as soon as `project.json` is saved.

#### ASP.NET Core 2.x:

By default, `Microsoft.Extensions.Logging.Console` is included in the meta package `Microsoft.AspNetCore.All`, so we don't need to install it separately in ASP.NET Core 2.x application.

### Step 2 - Use Provider

#### ASP.NET Core 1.x:

Now, in order to use console logger, we first need to configure it. First, we need to add `ConsoleLoggerProvider` to the providers list using `LoggerFactory`. As you can see in the `LoggerFactory` class above, it provides `AddProvider()` method which adds our custom `ILoggerProvider` type instance to the list. The `Microsoft.Extensions.Logging.Console` package includes all necessary classes. So, we can add `ConsoleLoggerProvider` instance as shown below.

Example: Add `ConsoleLoggerProvider`

```

public class Startup
{
    public Startup()
    {
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
    {

```



```

        loggerFactory.AddProvider(new ConsoleLoggerProvider((category, logLevel) => logLevel >= LogLevel.Information,
false));

        //code removed for clarity
    }
}

```

As you can see above, we add an instance of `ConsoleLoggerProvider` to the factory. Now, we can start logging. The console will display all the logs whose `LogLevel` is Information or above.

We can also use `AddConsole()` extension method instead of configuring console logger manually as shown below.

#### Example: Add Console Logger

```

public class Startup
{
    public Startup()
    {
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
    {
        loggerFactory.AddConsole();

        //code removed for clarity
    }
}

```

#### ASP.NET Core 2.x:

The `CreateDefaultBuilder()` already includes console and debug logging providers. So there is no need to add it again in the `Configure()` method in ASP.NET Core 2.x application.

#### Step 3 - Create Logs

This step is applicable for both ASP.NET Core 1.x and 2.x application.

Now, we can create logs and see it on the console by getting an instance of `ILogger` using `LoggerFactory` and start logging as shown below.

#### Example: Add Console Logger

```

public class Startup
{
    public Startup()
    {
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
    {
        loggerFactory.AddConsole();

        //start logging to the console
        var logger = loggerFactory.CreateLogger<ConsoleLogger>();
        logger.LogInformation("Executing Configure()");

        //code removed for clarity
    }
}

```

We can also use `ILogger` anywhere in our application. IoC container will inject `ConsoleLogger` instance wherever it sees `ILogger`. For example, we can use `ILogger` in the MVC-controller as shown below.

### Example: Logging in MVC Controller

```
public class HomeController : Controller
{
    ILogger _logger;

    public HomeController(ILogger<HomeController> logger)
    {
        _logger = logger;
    }
    public IActionResult Index()
    {
        _logger.LogInformation("Executing Home/Index");

        return View();
    }
}
```

You will see the above log on the console when you browse <http://localhost:5000/home/index>.

We can use any of the above mentioned built-in logging providers by following the same process.

### Log Levels

Log levels indicate the importance or severity of the log messages. Built-in log providers include extension methods to indicate log levels.

ASP.NET Core defines following log levels.

Log Level	Severity	Extension Method	Description
Trace	0	LogTrace()	Log messages only for tracing purpose for the developers
Debug	1	LogDebug()	Log messages for short-term debugging purpose
Information	2	LogInformation()	Log messages for the flow of the application.
Warning	3	LogWarning()	Log messages for abnormal or unexpected events in the application flow.
Error	4	LogError()	Log error messages.
Critical	5	LogCritical()	Log messages for the failures that require immediate attention

We can use extension method to indicate level of the log messages as shown below.

### Example: Log Level

```
public class HomeController : Controller
{
    ILogger _logger;

    public HomeController(ILogger<HomeController> logger)
    {
        _logger = logger;
    }

    public IActionResult Index(string id)
    {
        _logger.LogInformation("Home/Index Executing..");

        if (String.IsNullOrEmpty(id))
        {
```



```
        _logger.LogWarning(LoggingEvents.GET_ITEM_NOTFOUND, "Index({ID}) NOT FOUND", id);

        return NotFound();
    }


    return View();
}
```

Third-party Logging Providers

The following are some logging providers that work with ASP.NET Core:

Logging Provider	Description
<a href="#">elmah.io</a>	Provider for the Elmah.Io service
<a href="#">Loggr</a>	Provider for the Logger service
<a href="#">NLog</a>	Provider for the NLog library
<a href="#">Serilog</a>	Provider for the Serilog library

Learn about ASP.NET Core environment variables in the next chapter.



### Native Xamarin Chart Control - Powerful & Flexible Charts

Ad Realtime, Streaming & Big Data Xamarin Chart Controls. Download free trial now!

SciChart

[Learn more](#)

[< Previous](#)

[Next >](#)

TUTORIALSTEACHER.COM

TutorialsTeacher.com is optimized for learning web technologies step by step. Examples might be simplified to improve reading and basic understanding. While using this site, you agree to have read and accepted our terms of use and privacy policy.

✉ [feedback@tutorialsteacher.com](mailto:feedback@tutorialsteacher.com)

TUTORIALS

- > ASP.NET Core
- > ASP.NET MVC
- > IoC
- > Web API
- > C#
- > LINQ
- > Entity Framework
- > AngularJS
- > Node.js
- > D3.js



[JavaScript](#)[jQuery](#)[Sass](#)[Https](#)

## 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.

We respect your privacy.

---

[HOME](#) [PRIVACY POLICY](#) [TERMS OF USE](#) [ADVERTISE WITH US](#)

© 2018 TutorialsTeacher.com. All Rights Reserved.

