

# NLog Documentation

This documentation concerns just the issue surrounding the writing of the logs to the incorrect targets and it's NOT a complete documentation of NLog, since there are already enough resources for the general use of it on the internet:

1. <https://blog.elmah.io/nlog-tutorial-the-essential-guide-for-logging-from-csharp/>
2. <https://www.codeproject.com/Articles/4051307/NLog-Rules-and-filters>
3. [https://nlog-project.org/documentation/v4.3.0/html/R\\_Project\\_Documentation.htm](https://nlog-project.org/documentation/v4.3.0/html/R_Project_Documentation.htm)

NLog can create very detailed and organized logs having different targets for different log levels (debug, error, critical, etc.).

This is a simple example for configuring NLog to write all log levels' messages to a single target (a file), making sure that the logs reach just the file in case and that we don't overwrite the already existing configurations for NLog in case the app is run in another's app context like in our case (DeepL TP plugin).

```
var config = LogManager.Configuration ?? new LoggingConfiguration();

var logDirectoryPath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Applicati
Directory.CreateDirectory(logDirectoryPath);

var target = new FileTarget
{
    Name = "DeepL",
    FileName = Path.Combine(logDirectoryPath, "DeepLLogs.txt"),
    Layout = "${logger}: ${longdate} ${level} ${message} ${exception}"
};

config.AddTarget(target);
config.AddRuleForAllLevels(target, "*DeepL*");

LogManager.ReconfigExistingLoggers();
```

In the first line of code, we are making sure that we're not creating a new configuration for our app but checking to see if there isn't one available already and if there is, we'll add to that and not replace it.

```
var config = LogManager.Configuration ?? new LoggingConfiguration();
```

Then we create a target for our logs, which in this case will be of type FileTarget for which we have to have a name and a file name.

```
var target = new FileTarget
{
    Name = "DeepL",
    FileName = Path.Combine(logDirectoryPath, "DeepLLogs.txt"),
    Layout = "${logger}: ${longdate} ${level} ${message} ${exception}"
};
```

Then we add the target to the configuration (as you can see below, you can also add a name for the target when adding it to the config; of course, you don't have to do it in both places).

```
config.AddTarget("DeepL", target);
```

Then we map our chosen rules to this target. This is where the problem that resulted in our messages being routed to the incorrect targets resided. We have to be careful to specify the optional parameter "loggerNamePattern" (see docs [here](#)), this being the parameter that is used for mapping the messages to the target. This pattern has to match the name of the logger (not to be confused with the name of the target), logger obtained by calling `LogManager.GetLogger("NameOfLogger")` or `LogManager.GetCurrentClassLogger()`, explained later on this page.

Below you can see an example of such a mapping which directs every log levels' messages to the target created earlier.

```
config.AddRuleForAllLevels(target, "*DeepL*");
```

- `"*DeepL*"` translates to every string that contains "DeepL", so if we have a logger with the name `Sdl.Community.DeepLMTProvider.DeepLTranslationProviderConnector` it'll get matched by this pattern (more about this in the last section of the page).

**!!! Pay extra attention to the pattern, it is case sensitive.** For example, if the class name is `Sdl.Community.SdlDataProtectionSuite.SdlProjectAnonymizer.Batch_Task.RestOfFilesParser`, a logger in this class obtained by using `LogManager.GetCurrentClassLogger()` will return a logger by the EXACT same name. If you specify your pattern like this:

```
config.AddRuleForAllLevels(target, "*SDLDataProtectionSuite*");
```

it won't get picked up and you'll end up with no logs because of the letters' case.

As mentioned at the beginning of this page, we can add just the messages of the log levels we want just to the targets we want (in this example we only have one target, but we could have more, of more than just one type; eg. `"ColoredConsoleTarget"`):

```
config.AddRuleForOneLevel(LogLevel.Fatal, target, "*DeepL*");
```

Or, we can add the messages of a range of log levels:

```
config.AddRule(LogLevel.Info, LogLevel.Debug, target, "*DeepL*");
```

And finally, in the last line of code:

```
LogManager.ReconfigExistingLoggers();
```

we make sure that everything was updated (see docs [here](#)).

## What is the logger name used for?

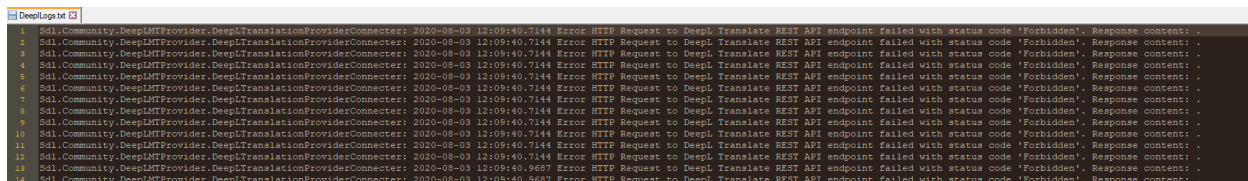
We obtain the logger either by using:

```
LogManager.GetCurrentClassLogger()
```

or

```
LogManager.GetLogger("NameOfLogger")
```

In the first case, we'll get one having the name of the class whence the call has been made. This is useful for more specific logs, having the class where the error occurred in its content:



```
1 Sdl.Community.DeepLMTProvider.DeepLTranslationProviderConnector: 2020-08-03 12:09:40.7144 Error HTTP Request to DeepL Translate REST API endpoint failed with status code 'Forbidden'. Response content: ...
2 Sdl.Community.DeepLMTProvider.DeepLTranslationProviderConnector: 2020-08-03 12:09:40.7144 Error HTTP Request to DeepL Translate REST API endpoint failed with status code 'Forbidden'. Response content: ...
3 Sdl.Community.DeepLMTProvider.DeepLTranslationProviderConnector: 2020-08-03 12:09:40.7144 Error HTTP Request to DeepL Translate REST API endpoint failed with status code 'Forbidden'. Response content: ...
4 Sdl.Community.DeepLMTProvider.DeepLTranslationProviderConnector: 2020-08-03 12:09:40.7144 Error HTTP Request to DeepL Translate REST API endpoint failed with status code 'Forbidden'. Response content: ...
5 Sdl.Community.DeepLMTProvider.DeepLTranslationProviderConnector: 2020-08-03 12:09:40.7144 Error HTTP Request to DeepL Translate REST API endpoint failed with status code 'Forbidden'. Response content: ...
6 Sdl.Community.DeepLMTProvider.DeepLTranslationProviderConnector: 2020-08-03 12:09:40.7144 Error HTTP Request to DeepL Translate REST API endpoint failed with status code 'Forbidden'. Response content: ...
7 Sdl.Community.DeepLMTProvider.DeepLTranslationProviderConnector: 2020-08-03 12:09:40.7144 Error HTTP Request to DeepL Translate REST API endpoint failed with status code 'Forbidden'. Response content: ...
8 Sdl.Community.DeepLMTProvider.DeepLTranslationProviderConnector: 2020-08-03 12:09:40.7144 Error HTTP Request to DeepL Translate REST API endpoint failed with status code 'Forbidden'. Response content: ...
9 Sdl.Community.DeepLMTProvider.DeepLTranslationProviderConnector: 2020-08-03 12:09:40.7144 Error HTTP Request to DeepL Translate REST API endpoint failed with status code 'Forbidden'. Response content: ...
10 Sdl.Community.DeepLMTProvider.DeepLTranslationProviderConnector: 2020-08-03 12:09:40.7144 Error HTTP Request to DeepL Translate REST API endpoint failed with status code 'Forbidden'. Response content: ...
11 Sdl.Community.DeepLMTProvider.DeepLTranslationProviderConnector: 2020-08-03 12:09:40.7144 Error HTTP Request to DeepL Translate REST API endpoint failed with status code 'Forbidden'. Response content: ...
12 Sdl.Community.DeepLMTProvider.DeepLTranslationProviderConnector: 2020-08-03 12:09:40.7144 Error HTTP Request to DeepL Translate REST API endpoint failed with status code 'Forbidden'. Response content: ...
13 Sdl.Community.DeepLMTProvider.DeepLTranslationProviderConnector: 2020-08-03 12:09:40.9657 Error HTTP Request to DeepL Translate REST API endpoint failed with status code 'Forbidden'. Response content: ...
14 Sdl.Community.DeepLMTProvider.DeepLTranslationProviderConnector: 2020-08-03 12:09:40.9657 Error HTTP Request to DeepL Translate REST API endpoint failed with status code 'Forbidden'. Response content: ...
```

But to actually put the name of the logger into the log we need to specify a layout for our target like this:

```
Layout = "${logger}: ${longdate} ${level} ${message} ${exception}"
```

In the code above, you can see that there are other layout variables available. You can find all the common ones explained [here](#).

# Working with NLog from Windows Service

The following article will help on setting up the NLog from a Windows Service: <http://programmerintoronto.blogspot.com/2014/08/nlog-using-nlog-with-a-windows-service.html>

**! Important: The NLog reference from Studio installed folder should be used and not the NLog installed through Nuget package (which will overwrite the Studio's NLog.dll and it might get into versioning/other configuration problems within Studio).**

1. It is needed to add a config file called NLog.config with the following setup that should be done (Example from SDL Analyse):

## NLog.config

```
<?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="${specialfolder:folder=CommonApplicationData}/SDL Community/SDL Analyse/SDL
      throwExceptions="false">

  <variable name="appName" value="SDL Analyse" />

  <targets async="true">
    <target xsi:type="File"
            name="logfile"
            layout="${longdate} - ${level:uppercase=true}:${logger} ${message}"
            fileName="${specialfolder:folder=CommonApplicationData}/SDL Community/SDL Analyse/SDL
            keepFileOpen="false"
            archiveFileName="${basedir}\logs\SDLANalyseLogs_${shortdate}.{##}.txt"
            archiveNumbering="Sequence"
            archiveEvery="Day"
            maxArchiveFiles="10"/>
  </targets>
  <rules>
    <logger name="*" writeTo="logfile" minlevel="Info" />
  </rules>
</nlog>
```

2. The NLog.config file needs to be defined within the Product.vxw file, under the following areas:

- ComponentGroup tag

### ComponentGroup define files

```
<ComponentRef Id="NLog.config" />
```

- DirectoryRef tag where the rest of the project components are defined as below:

### Define Nlog within Product.vxs file

```
<Component>
  <File Name="NLog.config"/>
</Component>
```

Note: The NLog.config file is installed and uninstalled together with the service (in SDL Analyse app case, within SDL Analyse service)

The fileName="{specialfolder:folder=CommonApplicationData}/SDL Community/SDL Analyse/SDLAnalyseLogs.txt" path is set up to CommonApplicationData folder, because WindowsService doesn't recognized correctly the ApplicationData ({specialfolder: folder=ApplicationData}) special folder path, which normally is recognized by .Net as "C:\Users\{UserName}\AppData\Roaming".

The ProgramData folder will be recognized correctly when running the logger from a WindowsService, see the below comment (and the related Stackoverflow topic is: <https://stackoverflow.com/questions/51750682/writing-to-the-applicationdata-folder-with-nlog-configuration-api>)

this?

▲

2

▼

✓

I did some more digging and it turns out that my issue is actually occurring because `Environment.SpecialFolder.ApplicationData` refers to a different location when referenced by a Windows Service running under the LocalSystem account and not due to a bug in NLog. The workaround for me was to choose to log to the `Environment.SpecialFolder.CommonApplicationData` folder as that location does not change when referenced by a Windows Service.

I used [this reference](#) to come up with my solution.