**Введение**

Многие начинающие разработчики при создании своих проектов не задумываются о такой вещи, как создание журнала события. Мол, проект у меня нормальный, и так сойдет. Но не забываем, что наше приложение мы пишем не для себя самих, а для клиента.

Всем нужна статистика и слежение за проектами. Итак, что же насчет логирования, так это процесс записи всех сведений о проекте, а именно: информации о работе тех или иных элементах приложения, предупреждение о критической нагрузке, всяческие ошибки и т.д. Для .NET приложений был разработан очень удобный фреймворк под название NLog, с его помощью можно вести учет о состоянии всего приложения. Есть поддержка записи в файл, в базу данных.

Настройка данной платформы очень удобна и легка, есть два способа:

1. через конфигурационный файл;
2. через конфигурационный объект LoggingConfiguration;

Первый способ самый простой, так как зондирование проекта уже встроено в саму библиотеку NLog. Вся работа основа на объекте Logger – парне, который занимается ведением учета состояния нашего проекта.

Для того чтобы продемонстрировать работу NLog, создадим проект по шаблону консольного приложения и назовем его NLogUnderstanding. Изначально наш проект выглядит следующим образом:

using System;

namespace NLogUnderstanding

{

    class Program

    {

        static void Main(string[] args)

        {

        }

    }

}

Чтобы начать работу данного фреймворка в нашем проекте, нужно установить следующие библиотеки через Package Manager Console (или же через сам менеджер расширений):

Установка бибиотеки

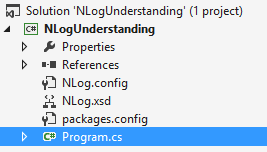
После установки NLog выбираем подход, по которому будем строить процесс слежения за состоянием приложение.

1. **Настройка через конфигурационный файл:**

Первое, что нужно сделать- это установить данный пакет:

Установка пакета

После это у нас в проекте появится указанный файлик NLog.config:



Начальное содержимое файла выглядеть будет примерно так:

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>">

  <targets>

  targets>

  <rules>

  rules>

nlog>

Все, после того как мы подготовили данную библиотеку, начинаем настройку объекта Logger. Первое, что мы должны сделать, это указать ему, куда мы будем писать те или иные сообщения. Все эти файлы указываются в разделе <targets>. Первое, что мы добавим, так это все возможные записи, которые мы сможем проводить:

  <targets>

    <target xsi:type="File" name="file" fileName="${basedir}/logs/${shortdate}.log"

            layout="${longdate} | ${uppercase:${level}} | ${logger} | ${message}" />

  targets>

Шесть возможных вариантов ведения учета:

* Информация о состоянии элементов;
* Информация, запущенная в режиме debug для отладки проекта (можно применять в тестах);
* Всяческие предупреждения (например, связанные с нагрузкой);
* Информация об исключениях;
* Информация об ошибках, которые привели к критическому завершению приложения.

Для каждого сообщения присутствует свой метод все в том же объекте Logger, который мы чуть позже будем разбирать. Основные атрибуты, которые нужно заполнить, это:

* name – название файла, нам оно понадобиться для организации правил, по которым мы будем писать именно в этот файл;
* fileName – указываем файл и путь к файлу, в который будем писать наши логи;
* layout – шаблон, по которому будет заполнятся наш файл.

Как Вы уже заметили,  заполнение значений атрибутов ведется в характерной манере регулярных выражений. То есть, мы используем заранее подготовленные в библиотеке маркеры подстановки для ведения учета наших сообщений в разные файлы. Основные, которые мы использовали, это:

* ${basedir} – вернет базовую директорию вашего приложения. При компиляции этот маркер вернет изначальный путь (папку bin);
* ${shortdate} / ${longdate} – маркеры подстановки устанавливают текущую дату и время в зависимости от маркера (полную дату и время или же только дату);
* ${uppercase:${level}} – интересное использование вложения маркеров. Как Вы поняли, маркер ${level} будет указываться уровень сообщения (мы их перечислили ранее), приводим в верхний регистр;
* ${message} – под данный маркер подставляется сообщение, указанное в аргументных скобках методов (об этом далее);
* ${logger} – название класса, от которого поступило сообщение.

После настройки целей для записи наших сообщений мы приступаем к организации правил, по которым будем заполнять наши файлы:

  <rules>

    <logger name="\*" minlevel="Trace" writeTo="file" />

  rules>

Тут все намного проще, единственное, что нужно заполнить - это основные атрибуты, т.к. minlevel (минимальный уровень заполнения файла, имя которого указанного в атрибуте writeTo). После того как настроили конфигурационный файл, приступаем к работе с проектом и нашим Logger. Первое, что нужно - это создать экземпляр Logger. Это можно сделать двумя способами:

* Создать через первый фабричный метод LogManager.GetLogger("Example"), в аргументах указываем название логгера, менее эффективный способ, т.к. всегда нужно указывать название класса, в котором происходит запись в журнал;
* Создание через второй фабричный метод LogManager.GetCurrentClassLogger(), пользуясь данным методом, мы предоставляем возможность экземпляру логгера самому узнать полное квалификационное название класса, в котором произошла запись в журнал.

Теперь привнесем изменения в наш созданный проект:

using System;

using NLog;

namespace NLogUnderstanding

{

    class Program

    {

        static void Main()

        {

            Logger logger = LogManager.GetCurrentClassLogger();

            log.Trace("trace message");

            log.Debug("debug message");

            log.Info("info message");

            log.Warn("warn message");

            log.Error("error message");

            log.Fatal("fatal message");

        }

    }

}

После компиляции проекта у нас создается файл с текущей датой и в него внесутся следующий записи:

2015-05-06 14:33:46.0911 | TRACE | NLogUnderstanding.Program | trace message

2015-05-06 14:33:46.1380 | DEBUG | NLogUnderstanding.Program | debug message

2015-05-06 14:33:46.1380 | INFO | NLogUnderstanding.Program | info message

2015-05-06 14:33:46.1536 | WARN | NLogUnderstanding.Program | warn message

2015-05-06 14:33:46.1536 | ERROR | NLogUnderstanding.Program | error message

2015-05-06 14:33:46.1693 | FATAL | NLogUnderstanding.Program | fatal message

Теперь можно приступать к внедрению NLog в Ваш проект, и отслеживать состояние ваших объектов.

**P.S.** Если вы планируете применять слежение за состоянием вашего проекта, то экземпляр логгера нужно будет создавать в нужных для отладки классах. В следующей части я опишу, как применять логгирование в веб проектах на основе ASP.NET MVC.

<https://itvdn.com/ru/blog/article/logging-project-with-nlog-framework>

<https://github.com/NLog/NLog/wiki/Tutorial>