2.4. Logging

2.4.1. Server Side

StudioX uses Castle Windsor's logging facility. It can work with different logging libraries**: Log4Net, NLog, Serilog**... etc. Castle provides a common interface for all logger libraries. So, you're independent from specific logging library and can easily change it later if you need.

Log4Net is one of the most popular logging libraries for .NET. StudioX templates come with Log4Net properly configured and working. But, there is just a single-line of dependency to log4net (as seen in configuration section), so you can change it to your favourite library.

Getting The Logger

No matter which logging library you choice, the code to write log is same (thanks to Castle's common ILogger interface).

First, we should get a Logger object to write logs. Since StudioX strongly uses dependency injection, we can easily inject a Logger object using property injection (or constructor injection) pattern. See a sample class that writes a log line:

```
using Castle.Core.Logging; //1: Import Logging namespace

public class TaskAppService : ITaskAppService
{
    //2: Getting a logger using property injection
    public ILogger Logger { get; set; }

    public TaskAppService()
    {
        //3: Do not write logs if no Logger supplied.
        Logger = NullLogger.Instance;
    }

    public void CreateTask(CreateTaskInput input)
    {
        //4: Write logs
        Logger.Info("Creating a new task with description: " + input.Description);

        //TODO: save task to database...
    }
}
```

- **First**, we imported namespace of Castle's ILogger interface.

- **Second**, we defined a public ILogger object named Logger. This is the object we will write logs. Dependency injection system will set (inject) this property after creating TaskAppService object. This is known as property injection pattern.
- **Third**, we set Logger to NullLogger.Instance. System will work fine without this line. But this is best practice on property injection pattern. If no one sets the Logger, it will be null and we get an "object reference..." exception when we want to use it. This guaranties that it's not null. So, if no one sets the Logger, it will be NullLogger. This is known as null object pattern. NullLogger actually does nothing, does not write any logs. Thus, our class can work with and without an actual logger.
- **Fourth** and the last, we're writing a log text with info level. There are different levels (see configuration section).

If we call the CreateTask method and check the log file, we see a log line as like as shown below:

```
INFO  2014-07-13 13:40:23,360 [8    ] SimpleTaskSystem.Tasks.TaskAppService - Creating a
new task with description: Remember to drink milk before sleeping!
```

Base Classes With Logger

StudioX provides base classes for MVC controllers, Web API controllers, Application service classes and more. They declare a Logger property. So, you can directly use this Logger to write logs, no injection needed. Example:

```
public class HomeController : SimpleTaskSystemControllerBase
{
    public ActionResult Index()
    {
        Logger.Debug("A sample log message...");
        return View();
    }
}
```

Note that SimpleTaskSystemControllerBase is our application specific base controller that inherits **StudioXController**. Thus, it can directly use Logger. You can also write your own common base class for other classes. Then, you will not have to inject logger every time.

Configuration

All configuration is done for Log4Net when you create your application from StudioX templates.

Default configuration's log format is as show below (for each line):

- **Log level**: DEBUG, INFO, WARN, ERROR or FATAL.
- **Date and time**: The time when the log line is written.
- **Thread number**: The thread number that writes the log line.
- **Logger name**: This is generally the class name which writes the log line.
- **Log text**: Actual log text that you write.

It's defined in the log4net.config file of the application as shown below:

```
<?xml version="1.0" encoding="utf-8" ?>
<log4net>
  <appender name="RollingFileAppender" type="log4net.Appender.RollingFileAppender" >
    <file value="Logs/Logs.txt" />
    <appendToFile value="true" />
    <rollingStyle value="Size" />
    <maxSizeRollBackups value="10" />
    <maximumFileSize value="10000KB" />
    <staticLogFileName value="true" />
    <layout type="log4net.Layout.PatternLayout">
       <conversionPattern value="%-5level %date [%-5.5thread] %-40.40logger -
%message%newline" />
    </layout>
  </appender>
  <root>
    <appender-ref ref="RollingFileAppender" />
    <level value="DEBUG" />
  </root>
  <logger name="NHibernate"><level value="WARN" /></logger>
</log4net>
```

Log4Net is highly configurable and strong logging library. You can write logs in different formats and to different targets (text file, database...). You can set minimum log levels (as set for NHibernate in this configuration). You can write diffent loggers to different log files. It can automatically backup and create new log file when it reaches to a specific size (Rolling file adapter with 10000 KB per file in this configuration) and so on... Read it's own confuguration documentation for more.

Finally, in the Global.asax file, we declare that we use Log4Net with log4net.config file:

```
public class MvcApplication : StudioXWebApplication
{
    protected override void Application_Start(object sender, EventArgs e)
    {
        IocManager.Instance.IocContainer.AddFacility<LoggingFacility>(
                f => f.UseLog4Net().WithConfig("log4net.config"));
        base.Application_Start(sender, e);
    }
}
```

This is the only code line we directly depend on log4net. Also, only the web project depends on log4net library's nuget package. So, you can easily change to another library without changing your logging code.

StudioX.Castle.Log4Net Package

StudioX uses Castle Logging Facility for logging and it does not directly depend on log4net, as declared above. But there is a problem with Castle's Log4Net integration. It does not support the latest log4net. We created a nuget package, StudioX.Castle.Log4Net, to solve this issue. After adding this package to our solution, all we should do is to change the code in application start like that:

```
public class MvcApplication : StudioXWebApplication
{
    protected override void Application_Start(object sender, EventArgs e)
    {
        IocManager.Instance.IocContainer.AddFacility<LoggingFacility>(
                f => f.UseStudioXLog4Net().WithConfig("log4net.config"));
        base.Application_Start(sender, e);
    }
}
```

The only difference is that we used "UseStudioXLog4Net()" method (defined in StudioX.Castle.Logging.-Log4Net namespace) instead of "UseLog4Net()".

2.4.2. Client Side

StudioX defines a simple javascript logging API for client side. It logs to browser's console as default. Sample javascript code to write logs:

```
studiox.log.warn('a sample log message...');
```

For more information, see logging API documentation.