# SizingServers.Log

## Intro

This is my take on an application logger for any 64 bit .Net 4.5 (and up) Windows desktop (maybe other app types, untested) app. (Yes, I know there is Log4Net, this was a fun little project, and well suited for my needs).

## Contents and usage

The Visual Studio 2015 solution holds the source code of the logger and and example WinForm. All (64 bit) binaries are compiled to the Build folder. The lib is fairly well commented. XML documentation is available.

There are two types of loggers available:

- FileLogger: logs to one plain text file per day, one entry per line, in a folder called logs next to the executing binary of you app
- MailLogger: to mail log entries to a given e-mail address

Both inherrit from the abstract class **Logger**. Logger provides all functionality doing the heavy lifting.

The log functions in the static class **Loggers** should be called to log using all loggers you enabled. FileLogger is enabled by default.
The library is coded this way so you could easily make your own logger. e.g. to log to a database or Logstash. Use the existing loggers as an example to do this.

Log entries are JSON serialized. Fatal, app crash, exceptions are logged automatically using al enabled loggers. Logging, except for app crashes, happens always on another thread, so the world does not stop when you do log.

Log entries are composed using a modern approach. Take a look at this function in the Loggers class for instance:

```
Log(Level level, string description, Exception exception, o
bject[] parameters,
[CallerMemberName] string member = "", [CallerFilePath] str
ing sourceFile = "",
[CallerLineNumber] int line = -1)
```

You can provide the log level, a description, the thrown exception and parameters. Parameters can be anything, but should be used to log the parameters you provided to the function wherein you call this Log fx.
All info regarding from where this function was called is always included in a log entry, if there are up-to-date .pdb files available for your binaries.

Do not forget to call Loggers.Flush() when the application exits.

Furthermore, this lib includes a simple WinForms file logger panel that you can use to list all log entries.

## Examples

There are 3 examples available in a WinForms app:

- FileLoggerExample
- MailLoggerExample
- FileLoggerPanelExample

We'll take a quick look at the FileLogger example.
Clicking button1 raises an exception, this exception is handled and a log entry is written to file. CLicking button2 raises an unhandled exception.

```
public FileLoggerExample() {

  InitializeComponent();
  Loggers.GetLogger<FileLogger>().LogEntryWritten += FileLo
ggerExample_LogEntryWritten;

  this.HandleCreated += FileLoggerExample_HandleCreated;

  //See Program.cs for an example of flushing the logger on
 application exit.

```

```
}



private void FileLoggerExample_HandleCreated(object sender,
 EventArgs e) {

  this.HandleCreated -= FileLoggerExample_HandleCreated;

  SynchronizationContextWrapper.SynchronizationContext = Sy
nchronizationContext.Current;

}



private void FileLoggerExample_LogEntryWritten(object sende
r, WriteLogEntryEventArgs e) {

  SynchronizationContextWrapper.SynchronizationContext.Send
(o => {

    linkLabel1.Visible = true;

    linkLabel1.Text = Loggers.GetLogger<FileLogger>().Curre
ntLogFile;

  }, null);

}



private void button1_Click(object sender, EventArgs e) {

  try {

    throw new Exception("Whoopsie");

  } catch (Exception ex) {

    Loggers.Log(Level.Error, "Caught whoopsie", ex, new obj
ect[] { sender, e, null });
```

```
  }

}



private void button2_Click(object sender, EventArgs e) {

  throw new Exception("Oh noze!");

}



private void linkLabel1_LinkClicked(object sender, LinkLabe
lLinkClickedEventArgs e) {

  Process.Start(linkLabel1.Text);

}
```