

Документация

изготвена от

Велин Новков Яворски 36 група, фак. ном. 121217099

по дисциплина Програмни Среди

на тема

„Предложение за реализация на Logger в отделен проект, който да се компилира до библиотека и да се преизползва в другите проекти“

Описание на реализацията

За реализация на библиотеката на C# е използван проект от тип Class Library (.NET Framework), чрез който накрая се генерира .dll файл, съдържащ програмния код, данните и ресурсите, нужни, за да се използва библиотеката в други проекти. Използването на библиотеката става като се добави препратка към .dll файла.

Описание на кода

Проектът на библиотеката съдържа следните файлове:

* Logger.cs – главния файл, в който се съдържа цялта логика на библиотеката
* LogFileCreateException.cs – грешка при създаване на лог файл
* LogFileReadException.cs – грешка при четене от лог файл
* LogFileWriteException.cs – грешка при запис в лог файл
* App.config – файл съдържащ шаблон за настройките на библиотеката. Оттук могат да се видят и добавят към App.config-а на приложението.

**Logger.cs:**

Този клас е главният за библиотеката и в него се съдържа цялата й логика.

Класът е статичен, защото няма смисъл да се инстанцира всеки път, когато някой иска да логне някаква дейност в кода си. За да се използва библиотеката е нужно само да се добави using static Logger.Logger;

static public class Logger

{

// По подразбиране имената на лог файловете ще са във този формат: trace\_<day>-<month>-<year>.log и ще се запазват в текущата директория.

static private readonly String DefaultFileName = String.Concat("trace\_", DateTime.Now.ToString("dd-MM-yyyy"), ".log");

private static readonly String DefaultFileDir = ".";

static private String currLogFileName = GetDefaultLogFileName();

static private String logFileName = GetDefaultLogFileName();

static private String logFileDir = GetDefaultLogFileDir();

// За логване на грешки може да се използва отделен файл.

static private String logFileNameErrors = GetDefaultLogFileNameErrors();

static private String logFilePath;

// В лист се пазят и логовете за текущата сесия, ако на някой случайно му трябват.

static private List<String> currentSessionActivities = new List<String>();

static private Boolean logCurrentSessionActivities = false;

private static readonly Int32 MaxCharsInActivity = 10\_000;

// Тези свойства пазят локациите за записване на логовете. Също така чрез тях потребителите могат да сменят къде да се записват логовете по време на изпълнение на приложението.

public static string LogFileDir

{

get => logFileDir;

set

{

CheckIfDirExists(value);

logFileDir = value;

}

}

public static string LogFileName

{

get => logFileName;

set

{

CreateLogFileIfNotExists(value);

logFileName = value;

}

}

public static string LogFileNameErrors

{

get => logFileNameErrors;

set

{

CreateLogFileIfNotExists(value);

logFileNameErrors = value;

}

}

// Това свойство съдържа името на лог файла, във който ще се пише. Може да се възприеме като pointer към името на текущия лог файл, в който ще се пише.

private static string CurrLogFileName

{

get => currLogFileName;

set => currLogFileName = value;

}

// Дали да се записват логове за текущата сесия

public static bool LogCurrentSessionActivities

{

get => logCurrentSessionActivities;

set => logCurrentSessionActivities = value;

}

// Това свойство съдържа файловата пътека за записване на логовете. Използва се всеки път, когато се записва някакъв лог. Може да се възприеме като pointer към текущата локация на записване на лога.

public static string LogFilePath

{

get => logFilePath;

private set { logFilePath = value; }

}

// Това са настройките по подразбиране

static public String GetDefaultLogFileName()

{

String logFileName = String.IsNullOrEmpty(GetAppSetting("LogFileName"))

? DefaultFileName

: GetAppSetting("LogFileName");

return logFileName;

}

static public String GetDefaultLogFileNameErrors()

{

String logFileName = String.IsNullOrEmpty(GetAppSetting("LogFileNameErrors"))

? DefaultFileName

: GetAppSetting("LogFileNameErrors");

return logFileName;

}

static public String GetDefaultLogFileDir()

{

String logFileName = String.IsNullOrEmpty(GetAppSetting("LogFileDir"))

? DefaultFileDir

: GetAppSetting("LogFileDir");

return logFileName;

}

// Това е метод, който връща настройките по подразбиране на библиотеката (имена и локации на лог файловете)

static public void ResetLogFilePaths()

{

LogFileDir = GetDefaultLogFileDir();

LogFileName = GetDefaultLogFileName();

LogFileNameErrors = GetDefaultLogFileNameErrors();

}

// Връща логовете за текущата сесия като string

static public String GetCurrentSessionActivitiesAsString()

{

StringBuilder stringBuilder = new StringBuilder();

foreach (String activity in CurrentSessionActivities)

{

stringBuilder.Append(activity);

}

return stringBuilder.ToString();

}

// Този метод създава лог файл по подадено име на файл (example: test\_log.log). Ако името е невалидно или файлът не успее да се създаде успешно, се връща съответно ArgumentNullException или LogFileCreateException.

static private void CreateLogFileIfNotExists(String fileName)

{

if (fileName == null)

{

throw new ArgumentNullException();

}

CheckIfDirExists(LogFileDir);

String filePath = Path.Combine(LogFileDir, Path.GetFileName(fileName));

LogFilePath = filePath;

if (!File.Exists(filePath))

{

try

{

File.Create(filePath).Close();

}

catch (Exception e)

{

throw new LogFileCreateException(e.ToString());

}

}

}

// Метод за проверка дали дадена директория съществува. Ако не съществува се хвърля DirectoryNotFoundException.

static private void CheckIfDirExists(String dir)

{

if (!Directory.Exists(dir))

{

throw new DirectoryNotFoundException("The directory '" + dir + "' does not exist!");

}

}

// Този метод се извиква от потребителите, за да се създаде нов лог. Ако дължината на подадения текст на лога е по-голям от разрешения се хвърля ArgumentOutOfRangeException. Тук става и записването в листа за текущата сесия.

static public void LogActivity(String activity)

{

if (activity == null)

{

activity = "";

}

if (activity.Length > MaxCharsInActivity)

{

throw new ArgumentOutOfRangeException("The activity is too long!");

}

String activityLine = DateTime.Now.ToString() + ": " + activity + "\n";

if (LogCurrentSessionActivities)

{

CurrentSessionActivities.Add(activityLine);

}

CurrLogFileName = logFileName;

AppendToFile(activityLine);

}

// Този метод се извиква от потребителите, за да се създаде нов лог за грешка. Ако дължината на подадената грешка е по-голяма от разрешената се хвърля ArgumentOutOfRangeException.

static public void LogError(String errMsg)

{

if (errMsg == null)

{

errMsg = "";

}

String errLine = DateTime.Now.ToString() + ": Error: " + errMsg + "\n";

if (errLine.Length > MaxCharsInActivity)

{

throw new ArgumentOutOfRangeException("The error is too long!");

}

CurrLogFileName = logFileNameErrors;

AppendToFile(errLine);

}

// Чрез този метод става записването на лога във файл. Първо се прави проверка дали лог файла съществува и, ако не, се създава. Ако стане грешка при записването се хвърля LogFileWriteException.

static private void AppendToFile(String line)

{

CreateLogFileIfNotExists(CurrLogFileName);

try

{

File.AppendAllText(LogFilePath, line);

} catch (Exception e)

{

throw new LogFileWriteException("Failed to write to log file '" + LogFilePath + "': " + e.ToString());

}

}

// Метод за принтиране на съдръжанието на лог файла. Използва се StreamReader, понеже обемът на лога може да е твърде голям. Ако стане грешка при четенето от лог файла се връща хвърля LogFileReadException.

static public void PrintLogFileToConsole()

{

StreamReader sr = null;

try

{

sr = new StreamReader(LogFilePath);

String line;

while (!sr.EndOfStream)

{

line = sr.ReadLine();

Console.WriteLine(line);

}

}

catch (Exception e)

{

throw new LogFileReadException("Failed to read from log file '" + LogFilePath + "': " + e.ToString());

}

finally

{

sr.Close();

}

}

// Чрез този метод се взимат зададените настройки от App.config файла (key = LogFileName или LogFileNameErrors или LogFileDir)

static private String GetAppSetting(String key)

{

String value = ConfigurationManager.AppSettings[key];

if (!String.IsNullOrEmpty(value))

{

return value;

}

return String.Empty;

}

}