

**DOCUMENTATION : DLL**

Table des matières

[***I.*** ***Introduction:*** 3](#_Toc190160562)

[**II.** **Definition:** 3](#_Toc190160563)

[**III.** **Use cases:** 4](#_Toc190160564)

[Case 1: DLL for log management (EasySave.Logging.dll): 4](#_Toc190160565)

[Case 2: DLL for file encryption (EasySave.Crypto.dll): 7](#_Toc190160566)

[Case 3: DLL for a user management API (EasySave.Users.dll): 10](#_Toc190160567)

[**IV.** **Setting up a daily log DLL for the EasySave project:** 12](#_Toc190160568)

[- Creating our DLL on Visual Studio 2022: 12](#_Toc190160569)

[- Building our DLL: 12](#_Toc190160570)

[- Tree structure and DLL setup: 13](#_Toc190160571)

[ Explanation of each file: 14](#_Toc190160572)

[- DLL compilation: 18](#_Toc190160573)

[- How to use the DLL in a Project ? 19](#_Toc190160574)

[- Example of DLL use in a project: 21](#_Toc190160575)

[**V.** **Troubleshooting and common errors:** 22](#_Toc190160576)

[**VI.** **Conclusion:** 22](#_Toc190160577)

1. ***Introduction:***

As part of EasySave's development, a Dynamic Link Library (DLL) was integrated to optimize the application's architecture and guarantee greater modularity.

This DLL enables back-up logs to be managed independently and efficiently, while respecting the project's MVC architecture.

The main objective is to centralize log management in a reusable module, thus guaranteeing:

- Clear separation of responsibilities (MVC principle),

- Enhanced scalability, enabling new functionalities to be added without impacting the code as a whole,

- Future compatibility with EasySave evolutions.

1. **Definition:**

A Dynamic Link Library is a software library whose functions are loaded into memory by a program, as and when required, during execution, as opposed to static or shared software libraries whose functions are loaded into memory before program execution begins.

.dll is a filename extension used by files containing a Dynamic Link Library.

1. **Use cases:**

DLLs (Dynamic Link Libraries) are used to bundle reusable code in a separate file. This modularizes the application, facilitates maintenance and reduces code redundancy.

## Case 1: DLL for log management (EasySave.Logging.dll):

*Example:* A backup application uses a DLL to store all events in a log file.

using System;

using System.IO;

using System.Text.Json;

using System.Collections.Generic;

namespace EasySave.Logging

{

public static class LogManager

{

private static readonly string \_logDirectory = @"C:\Users\Public\Documents\EasySave\Logs";

static LogManager()

{

if (!Directory.Exists(\_logDirectory))

Directory.CreateDirectory(\_logDirectory);

}

public static void WriteLog(string backupName, string sourceFile, string targetFile, long fileSize, double transferTime)

{

string logFilePath = Path.Combine(\_logDirectory, $"backup\_log\_{DateTime.Now:yyyy-MM-dd}.json");

List<LogEntry> logs = File.Exists(logFilePath)

? JsonSerializer.Deserialize<List<LogEntry>>(File.ReadAllText(logFilePath)) ?? new List<LogEntry>()

: new List<LogEntry>();

logs.Add(new LogEntry

{

Time = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"),

BackupName = backupName,

FileSource = Path.GetFullPath(sourceFile),

FileTarget = Path.GetFullPath(targetFile),

FileSize = fileSize,

TransferTime = transferTime

});

File.WriteAllText(logFilePath, JsonSerializer.Serialize(logs, new JsonSerializerOptions { WriteIndented = true }));

}

}

public class LogEntry

{

public string Time { get; set; }

public string BackupName { get; set; }

public string FileSource { get; set; }

public string FileTarget { get; set; }

public long FileSize { get; set; }

public double TransferTime { get; set; }

}

}

Using the DLL in a main project:

using System;

using EasySave.Logging;

class Program

{

static void Main()

{

Console.WriteLine("Démarrage du test de la DLL...");

LogManager.WriteLog("Sauvegarde1", @"C:\DossierSource\Fichier.txt", @"D:\Backup\Fichier.txt", 1024, 500);

Console.WriteLine("Log enregistré avec succès !");

}

}

## Case 2: DLL for file encryption (EasySave.Crypto.dll):

*Example:* An application that encrypts and decrypts files before saving them.

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

namespace EasySave.Crypto

{

public static class CryptoManager

{

private static readonly string key = "MaCléSecrète1234";

public static void EncryptFile(string inputFile, string outputFile)

{

using (Aes aes = Aes.Create())

{

aes.Key = Encoding.UTF8.GetBytes(key);

aes.IV = new byte[16];

using (FileStream fsInput = new FileStream(inputFile, FileMode.Open))

using (FileStream fsOutput = new FileStream(outputFile, FileMode.Create))

using (CryptoStream cryptoStream = new CryptoStream(fsOutput, aes.CreateEncryptor(), CryptoStreamMode.Write))

{

fsInput.CopyTo(cryptoStream);

}

}

}

public static void DecryptFile(string inputFile, string outputFile)

{

using (Aes aes = Aes.Create())

{

aes.Key = Encoding.UTF8.GetBytes(key);

aes.IV = new byte[16];

using (FileStream fsInput = new FileStream(inputFile, FileMode.Open))

using (FileStream fsOutput = new FileStream(outputFile, FileMode.Create))

using (CryptoStream cryptoStream = new CryptoStream(fsInput, aes.CreateDecryptor(), CryptoStreamMode.Read))

{

cryptoStream.CopyTo(fsOutput);

}

}

}

}

}

Using the DLL in a main project:

using System;

using EasySave.Crypto;

class Program

{

static void Main()

{

Console.WriteLine("Démarrage du test de chiffrement...");

CryptoManager.EncryptFile(@"C:\DossierSource\Fichier.txt", @"C:\DossierChiffre\Fichier.enc");

Console.WriteLine("Fichier chiffré avec succès !");

CryptoManager.DecryptFile(@"C:\DossierChiffre\Fichier.enc", @"C:\DossierDechiffre\Fichier.txt");

Console.WriteLine("Fichier déchiffré avec succès !");

}

}

## Case 3: DLL for a user management API (EasySave.Users.dll):

*Example:* A DLL that manages an application's users.

using System;

using System.Collections.Generic;

namespace EasySave.Users

{

public static class UserManager

{

private static List<string> users = new List<string>();

public static void AddUser(string username)

{

users.Add(username);

Console.WriteLine($"Utilisateur {username} ajouté !");

}

public static bool UserExists(string username)

{

return users.Contains(username);

}

}

}

Using the DLL in a main project:

using System;

using EasySave.Users;

class Program

{

static void Main()

{

Console.WriteLine("Ajout d'un utilisateur...");

UserManager.AddUser("Alice");

if (UserManager.UserExists("Alice"))

{

Console.WriteLine("Utilisateur Alice trouvé !");

}

}

}

Conclusion:

DLLs make it possible to separate a project's business logic into independent, reusable and easy-to-maintain modules.

They are particularly useful for:

- Log management (EasySave.Logging)

- Encrypt files (EasySave.Crypto)

- Manage users (EasySave.Users)

1. **Setting up a daily log DLL for the EasySave project:**

* Creating our DLL on Visual Studio 2022:

First, from the Visual Studio 2022 interface, we need to set up a new project. This new project must be a .NET 8.0 language “Class Library” (the same as the project in which we need to use it). It will be placed wherever we like, and the name will also be up to us.Une image contenant texte, Police, Logiciel multimédia, logiciel

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.

* Building our DLL:

In the case of our project's EasySave application, we were asked to be able to manage the creation of a daily log file of all job backup executions carried out that day, from a “backup\_log\_yyyy-mm-dd.JSON” file. Une image contenant texte, Police, nombre, ligne

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, affichage

Le contenu généré par l’IA peut être incorrect.

* Tree structure and DLL setup:

For a DLL, we need a class with several methods to build it. This will be the basis of the DLL, plus dynamic attributes that will enable us to link it to the project. Une image contenant texte, capture d’écran, logiciel, Police

Le contenu généré par l’IA peut être incorrect.

* Explanation of each file:

**LogEntry.cs: Definition of a log entry**

* This file defines the structure of a log entry.

namespace EasySave.Logging

{

public class LogEntry

{

public string Time { get; set; } // Horodatage (yyyy-MM-dd HH:mm:ss)

public string BackupName { get; set; } // Nom de la sauvegarde

public string FileSource { get; set; } // Adresse complète du fichier source

public string FileTarget { get; set; } // Adresse complète du fichier destination

public long FileSize { get; set; } // Taille du fichier (en octets)

public double TransferTime { get; set; } // Temps de transfert en ms (négatif si erreur)

}

}

**Explanation:**

This class serves as a template for storing log information.

It will be used to serialize/deserialize logs in JSON.

**Logger.cs: Log management**

* 1. **General presentation**

The EasySave.Logger namespace contains a Logger class, which:

- Generates a daily JSON log file in a defined directory.

- Writes each save operation to this file.

- Ensures concurrent access security with a lock Obj.

Uses Newtonsoft.Json to manipulate JSON files.

* 1. **Explanation of code elements**

**a) Imports**

The file begins by importing the essential libraries:

using System;

using System.IO;

using Newtonsoft.Json;

- System.IO: Manages read/write of log files.

- Newtonsoft.Json: Serializes objects in JSON.

- System.Threading.Tasks: Adds multitasking (not used here).

**b) Classe Logger**

The Logger class contains:

public class Logger

- A log file path (logFilePath).

- A lock (lockObj) to guarantee write access security.

**c) Dynamic log file generation**

The log file changes every day and is stored in a specific folder.

public static string GetLogFileName()

{

string basePath = Environment.GetEnvironmentVariable("EASYSAVE\_LOG\_PATH") ??

Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData), "CESI", "EasySave", "Logs");

if (!Directory.Exists(basePath))

{

Directory.CreateDirectory(basePath);

}

return Path.Combine(basePath, $"backup\_log\_{DateTime.Today:dd-MM-yyyy}.json");

}

- Check that the EASYSAVE\_LOG\_PATH environment variable is set.

- If not, use common folder C:\ProgramData\CESI\EasySave\Logs.

- Create folder if necessary.

- Generates a daily JSON file in backup\_log\_DD-MM-YYYY.json format.

**d) Writing to the log file**

The WriteLog() method adds a JSON record for each saved file.

1. **Creating a Log Object**

var LogEntry = new

{

Name = name,

FileSource = fileSource,

FileTarget = fileTarget,

FileSize = fileSize,

FileTransferTime = isError ? -1 : fileTransferTime,

Date = DateTime.Now.ToString("dd/MM/yyyy HH:mm:ss")

};

- Name: Backup name.

- FileSource: Source file path.

- FileTarget: Destination file path.

- FileSize: File size.

- FileTransferTime: Transfer time (if error: -1).

- Date: Date and time of input.

1. **Writing to JSON file**

lock (lockObj)

{

string jsonData = JsonConvert.SerializeObject(LogEntry, Formatting.Indented);

if (!File.Exists(logFilePath) || new FileInfo(logFilePath).Length == 0)

{

File.WriteAllText(logFilePath, "[" + jsonData + "]" + Environment.NewLine);

}

else

{

string existingData = File.ReadAllText(logFilePath);

existingData = existingData.TrimEnd(']', '\r', '\n', ' ');

existingData += "," + Environment.NewLine + jsonData + "]";

File.WriteAllText(logFilePath, existingData);

}

}

- If the file doesn't exist or is empty, it is initialized with a JSON list containing a single element.

- Otherwise, the existing JSON is modified to add a new entry:

o It removes the final] from the array.

o Adds the new entry with a comma.

o Closes the array with].

This approach preserves a valid JSON format containing a list of entries.

1. **Add DLL to project**

The DLL must be added to the project with:

dotnet add reference C:\Users\salem\source\repos\Aysee2\FISA\_A3\_GL\_AYSE2\EasySave.Logger/EasySave.Logger.csproj

Cela permet au projet **EasySave** d’utiliser cette bibliothèque pour générer des logs.

* DLL compilation:

After writing the code:

* **Compile** the project from Visual Studio
* **Check** in “bin/Debug/net8.0/”.

You should see a file in the project directory “bin/Debug/net8.0/”

EasySave.Logger.dll

* Then you’ve compiled successfully!

**Possible error:**

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

If this error occurs during compilation, don't panic, it proves that the DLL can only be used in “console application (.Net)” projects. This shows that your DLL is working as it should!

* How to use the DLL in a Project ?

In a project **(e.g. EasySave),** add the DLL:  
**➜ Right-click on “References or the project.sln”**

**➜ Add a “Project reference”.**

**Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.**

➜ **Select "EasySave.Logger.dll"**

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

**You would then have the DLL displayed in the project:**Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

In return, it should appear in “bin/Debug/net8.0” of the project you're working on, otherwise you could get this kind of error: Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

In this case, add it manually in this directory and add the DLL again!

* Example of DLL use in a project:

using EasySave.Logging; 🡪 **the DLL added as a library**

class Program

{

static void Main()

{

LogManager.WriteLog("Backup1", "C:\\Source", "D:\\Backup", 1000, 2.5); 🡪 **calls one of the DLL's methods**

var logs = LogManager.GetLogs();

Console.WriteLine($"Nombre de logs trouvés : {logs.Count}");

}

}

1. **Troubleshooting and common errors:**

* **Problems and solutions**

**Error: "** **DLL cannot be found"**  
🡪 Solution: Check that the DLL has been added as a reference and is present in bin/Debug/net8.0/.

**Error: "** **Method not found"**  
🡪 Solution: Check the signature of the method called in LogManager.

**Error: "** **JSON format invalid"**  
🡪 Solution: Check that the log JSON file (backup\_log\_yyyy-MM-dd.json) is not corrupted.

* 1. **Conclusion:**

Using a Dynamic Link Library (DLL) allows you to:

- Optimize modularity and code reuse.

- Facilitate maintenance and updates.

- Improve EasySave log management.

Thanks to this DLL, the project is more structured, scalable and easy to maintain.