

**DOCUMENTATION : DLL**

Table des matières

[***I.*** ***Introduction :*** 3](#_Toc189926847)

[**II.** **Définition :** 3](#_Toc189926848)

[**III.** **Cas d’utilisation :** 4](#_Toc189926849)

[Cas 1 : DLL pour la gestion des logs (EasySave.Logging.dll) : 4](#_Toc189926850)

[Cas 2 : DLL pour le chiffrement des fichiers (EasySave.Crypto.dll) : 7](#_Toc189926851)

[Cas 3 : DLL pour une API de gestion des utilisateurs (EasySave.Users.dll) : 10](#_Toc189926852)

[**IV.** **Mise en place d’une DLL des logs journaliers pour le projet EasySave :** 12](#_Toc189926853)

[- Création de notre dll sur Visual Studio 2022 : 12](#_Toc189926854)

[- Construction de notre dll : 12](#_Toc189926855)

[- Arborescence et mise en place de la dll : 13](#_Toc189926856)

[ Explication de chaque fichier : 14](#_Toc189926857)

[- Compilation en DLL : 19](#_Toc189926858)

[- Comment utiliser la DLL dans un projet ? 20](#_Toc189926859)

[- Exemple d’utilisation de la DLL dans un projet : 22](#_Toc189926860)

[**V.** **Dépannage et erreurs fréquentes :** 22](#_Toc189926861)

[**VI.** **Conclusion :** 22](#_Toc189926862)

1. ***Introduction :***

Dans le cadre du développement de EasySave, une Dynamic Link Library (DLL) a été intégrée afin d’optimiser l’architecture de l’application et de garantir une meilleure modularité.  
Cette DLL permet notamment de gérer les logs des sauvegardes de manière indépendante et efficace, tout en respectant l'architecture MVC du projet.

L’objectif principal est de centraliser la gestion des logs dans un module réutilisable, garantissant ainsi :

* Une séparation claire des responsabilités (principe du MVC),
* Une évolutivité accrue, permettant l'ajout de nouvelles fonctionnalités sans impacter l’ensemble du code,
* Une compatibilité future avec les évolutions d’EasySave.

1. **Définition :**

Une Dynamic Link Library (en français, bibliothèque de liens dynamiques) est une [bibliothèque logicielle](https://fr.wikipedia.org/wiki/Biblioth%C3%A8que_logicielle) dont les fonctions sont chargées en mémoire par un programme, au besoin, lors de son exécution, par opposition aux bibliothèques logicielles statiques ou partagées dont les fonctions sont chargées en mémoire avant le début de l'exécution du programme.

*.dll* est une [extension de nom de fichier](https://fr.wikipedia.org/wiki/Extension_de_nom_de_fichier) utilisée par des fichiers contenant une Dynamic Link Library.

1. **Cas d’utilisation :**

Les DLLs (Dynamic Link Libraries) sont utilisées pour regrouper du code réutilisable dans un fichier séparé. Cela permet de modulariser l'application, de faciliter la maintenance et de réduire la redondance du code.

## Cas 1 : DLL pour la gestion des logs (EasySave.Logging.dll) :

*Exemple :* Une application de sauvegarde utilise une DLL pour stocker tous les événements dans un fichier log.

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; }

}

}

Utilisation de la DLL dans un projet principal :

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 !");

}

}

## Cas 2 : DLL pour le chiffrement des fichiers (EasySave.Crypto.dll) :

*Exemple :* Une application qui doit chiffrer et déchiffrer des fichiers avant de les sauvegarder.

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);

}

}

}

}

}

Utilisation de la DLL dans un projet principal :

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 !");

}

}

## Cas 3 : DLL pour une API de gestion des utilisateurs (EasySave.Users.dll) :

*Exemple :* Une DLL qui gère les utilisateurs d'une application.

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);

}

}

}

Utilisation de la DLL dans un projet principal :

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 :

Les DLL permettent de séparer la logique métier d'un projet en modules indépendants, réutilisables et faciles à maintenir.  
Elles sont particulièrement utiles pour :

* **Gérer les logs** (EasySave.Logging)
* **Chiffrer les fichiers** (EasySave.Crypto)
* **Gérer les utilisateurs** (EasySave.Users)

1. **Mise en place d’une DLL des logs journaliers pour le projet EasySave :**

* Création de notre DLL sur Visual Studio 2022 :

Tout d’abord depuis l’interface Visual Studio 2022, nous devons mettre en place un nouveau projet. Ce nouveau projet doit être une « Bibliothèque de classe » langage .NET 8.0 (le même que le projet dans lequel nous devons l’utiliser). Elle sera placée là où l’on souhaite, ainsi que le nom aussi dépendra de nous.

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.

* Construction de notre DLL :

Dans le cas de l’application EasySave de notre projet, il nous a été demandé de pouvoir gérer la mise en place d’un fichier log journalier de toutes les exécutions des sauvegardes de travails réalisées ce jour-là, depuis un fichier « backup\_log\_yyyy-mm-dd.JSON ».

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.

* Arborescence et mise en place de la DLL :

Pour une DLL nous avons besoin d’une classe avec plusieurs méthodes qui le construit, celle-ci serons notre base de la DLL à cela s’ajoute les attribue dynamique qui nous permettrons de faire le lien avec le projet.

Une image contenant texte, capture d’écran, logiciel, Police

Le contenu généré par l’IA peut être incorrect.

* Explication de chaque fichier :

**LogEntry.cs : Définition d'un enregistrement de log**

* Ce fichier définit la structure d’une entrée de log.

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)

}

}

**Explication :**

Cette classe sert de modèle pour stocker les informations d’un log.  
Elle sera utilisée pour sérialiser/désérialiser les logs en JSON.

**Logger.cs : Gestion des logs**

**1. Présentation générale**

Le **namespace EasySave.Logger** contient une classe Logger, qui :

* Génère un fichier de log JSON **par jour** dans un répertoire défini.
* Écrit chaque opération de sauvegarde dans ce fichier.
* Assure la **sécurité des accès concurrents** avec un verrou (lockObj).

Il utilise **Newtonsoft.Json** pour manipuler les fichiers JSON.

**2. Explication des éléments du code**

**a) Importations**

Le fichier commence par **importer des bibliothèques** essentielles :

using System;

using System.IO;

using Newtonsoft.Json;

* System.IO : Gère-la lecture/écriture des fichiers de logs.
* Newtonsoft.Json : Sérialise les objets en JSON.
* System.Threading.Tasks : Permettrait d'ajouter du multitâche (non utilisé ici).

**b) Classe Logger**

public class Logger

La classe Logger contient :

* Un **chemin de fichier de log** (logFilePath).
* Un **verrou (lockObj)** pour garantir la **sécurité d'accès en écriture**.

**c) Génération dynamique du fichier de log**

Le fichier de log **change chaque jour** et est stocké dans un dossier spécifique.

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");

}

* Vérifie si la variable d’environnement **EASYSAVE\_LOG\_PATH** est définie.
* Si non, **utilise le dossier commun C:\ProgramData\CESI\EasySave\Logs**.
* **Crée le dossier si nécessaire**.
* Génère un fichier JSON journalier sous le format **backup\_log\_DD-MM-YYYY.json**.

**d) Écriture dans le fichier de log**

La méthode **WriteLog()** ajoute un **enregistrement JSON** pour chaque fichier sauvegardé.

**1. Création d'un objet log**

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 : Nom de la sauvegarde.
* FileSource : Chemin du fichier source.
* FileTarget : Chemin du fichier destination.
* FileSize : Taille du fichier.
* FileTransferTime : Temps de transfert (si erreur : -1).
* Date : Date et heure de l’entrée.

**2. Écriture dans le fichier JSON**

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);

}

}

* **Si le fichier n’existe pas ou est vide**, il est initialisé avec une **liste JSON contenant un seul élément**.
* **Sinon**, le JSON existant est **modifié pour ajouter une nouvelle entrée** :
  + Il **retire le] final** du tableau.
  + Il **ajoute la nouvelle entrée avec une virgule**.
  + Il **referme le tableau avec]**.

Cette approche permet de **conserver un format JSON valide** contenant **une liste d’entrées**.

**3. Ajout de la DLL au projet**

La DLL doit être ajoutée au projet avec :

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.

* Compilation en DLL :

Après avoir écrit le code :

* **Compiler** le projet depuis Visual Studio
* **Vérifier** dans « bin/Debug/net8.0/ »

Vous devriez voir un fichier dans le répertoire du projet « bin/Debug/net8.0/ » EasySave.Logger.dll

* Alors votre compilation est réussie !

**Erreur possible :**

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Si cette erreur survie lors de la compilation, pas de panique cela prouve qu’elle est utilisable que dans des projets « application de console (.Net) » cela permet de voir que votre DLL fonctionne comme il le faut !

* Comment utiliser la DLL dans un projet ?

Dans un projet **(ex : EasySave)**, ajouter la DLL :  
➜ **Clic droit sur "Références ou le projet.sln"**

➜ **Ajouter une « Référence de projet »**

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

Le contenu généré par l’IA peut être incorrect.

➜ **Sélectionner "EasySave.Logger.dll"**

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

Le contenu généré par l’IA peut être incorrect.

**Vous auriez alors la DLL qui s’affiche dans le projet :**

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

Le contenu généré par l’IA peut être incorrect.

En retour, celle-ci devrait apparaître dans « bin/Debug/net8.0 » du projet dans laquelle vous travailler, sinon alors vous pourrais avoir ce genre d’erreur :

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

Dans ce cas, ajouter le manuellement dans ce répertoire et refaite l’ajout de la DLL !

* Exemple d’utilisation de la DLL dans un projet :

using EasySave.Logging; **🡪 Le DLL ajouté en tant que bibliothèque**

class Program

{

static void Main()

{

LogManager.WriteLog("Backup1", "C:\\Source", "D:\\Backup", 1000, 2.5); 🡪 **Appelle d’une des méthodes de la DLL**

var logs = LogManager.GetLogs();

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

}

}

1. **Dépannage et erreurs fréquentes :**

* **Problèmes et solutions**

**Erreur : "Impossible de trouver la DLL"**  
🡪 Solution : Vérifier que la DLL est bien ajoutée en référence et présente dans bin/Debug/net8.0/.

**Erreur : "Méthode introuvable"**  
🡪 Solution : Vérifier la signature de la méthode appelée dans Logger.

**Erreur : "Format JSON invalide"**  
🡪 Solution : Vérifier que le fichier JSON des logs (backup\_log\_yyyy-MM-dd.json) n'est pas corrompu.

* 1. **Conclusion :**

L'utilisation d'une Dynamic Link Library (DLL) permet de :

- Optimiser la modularité et la réutilisation du code.  
- Faciliter la maintenance et les mises à jour.  
- Améliorer la gestion des logs dans EasySave.

Grâce à cette DLL, le projet est plus structuré, évolutif et facile à maintenir.