

**DOCUMENTATION : DLL**

Table des matières

[***I.*** ***Introduction :*** 3](#_Toc190612564)

[**II.** **Définition :** 3](#_Toc190612565)

[**III.** **Mise en place d’une DLL des logs journaliers pour le projet EasySave :** 4](#_Toc190612566)

[- Création de notre DLL sur Visual Studio 2022 : 4](#_Toc190612567)

[- Construction de notre DLL : 4](#_Toc190612568)

[- Arborescence et mise en place de la DLL : 5](#_Toc190612569)

[ Explication de chaque fichier **Logger.cs** : 5](#_Toc190612570)

[- Compilation en DLL : 9](#_Toc190612571)

[- Comment utiliser la DLL dans un projet ? 10](#_Toc190612572)

[- Exemple d’utilisation de la DLL dans un projet : 12](#_Toc190612573)

[**IV.** **Dépannage et erreurs fréquentes :** 12](#_Toc190612574)

[**VI.** **Conclusion :** 12](#_Toc190612575)

1. ***Introduction :***

Dans le cadre du développement de EasySave, une Dynamic Link Library (DLL) a été intégrée afin d’optimiser la gestion des logs et assurer un suivi détaillé des sauvegardes.

Cette DLL permet :  
Un suivi précis des sauvegardes et du cryptage  
Un stockage structuré des logs (JSON ou XML)  
Une gestion centralisée des logs  
Une indépendance vis-à-vis du reste du code

Depuis la version 2.0, un compteur de temps de cryptage a été ajouté pour suivre la performance des opérations.

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. **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 JSON ou XML « C:\ProgramData\CESI\EasySave\Logs\backup\_log\_yyyy-mm-dd.JSON /XML».

Une image contenant texte, Police, capture d’écran

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, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

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

Le contenu généré par l’IA peut être incorrect.

* Explication de chaque fichier **Logger.cs** :

**1. Présentation générale**

Le namespace EasySave.Logger contient une classe Logger qui :

* Génère un fichier de log JSON ou XML par jour dans un répertoire défini.
* Écrit chaque opération de sauvegarde dans ce fichier.
* Prend en charge l’enregistrement du temps de cryptage en plus du temps de transfert.
* Assure la sécurité des accès concurrents avec un verrou lockObj.
* Permet de choisir entre JSON et XML en fonction des préférences de l’utilisateur.

Bibliothèque utilisée :

* Newtonsoft.Json pour manipuler les fichiers JSON.
* System.Xml.Linq pour gérer les fichiers XML.
* System.IO pour la lecture et l’écriture des fichiers.

**2. Explication des éléments du code**

**a) Importations**

Le fichier commence par importer des bibliothèques essentielles :

* System.IO : Gère la lecture et l’écriture des fichiers de logs.
* Newtonsoft.Json : Sérialise les objets en JSON.
* System.Xml.Linq : Manipule les fichiers XML.
* System.Threading.Tasks : Sécurise l’accès aux fichiers en multitâche.

**b) Classe Logger**

La classe Logger contient :

* logFilePath : Chemin d’accès du fichier de logs.
* logFormat : Format des logs (json ou xml).
* lockObj : Verrou garantissant la sécurité d’accès en écriture.
* Méthodes WriteLog() : Permet d’enregistrer les opérations de sauvegarde.
* Gestion du temps de cryptage dans les logs pour un meilleur suivi des performances.

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

**Processus de création du fichier log :**

1. **Le fichier de log est créé chaque jour avec un format dynamique :**

backup\_log\_YYYY-MM-DD.json

backup\_log\_YYYY-MM-DD.xml

1. **Il est stocké dans un répertoire défini par une variable d’environnement EASYSAVE\_LOG\_PATH ou, par défaut :**

C:\ProgramData\CESI\EasySave\Logs\

1. **Si le répertoire n’existe pas, il est automatiquement créé.**
2. **Le fichier est écrit en JSON ou XML en fonction du format choisi.**

**3. Écriture des logs**

**a) Création d’un objet log**

Chaque entrée de log contient :

* Name : Nom de la sauvegarde.
* FileSource : Chemin du fichier source.
* FileTarget : Chemin du fichier destination.
* FileSize : Taille du fichier.
* FileTransferTime : Temps de transfert (en millisecondes).
* EncryptionTime : Temps de cryptage (en millisecondes).
* Date : Date et heure de l’entrée.

**b) Écriture dans un fichier JSON**

**Processus d’enregistrement JSON :**

1. Vérification de l'existence du fichier :
   * S’il n’existe pas, un tableau JSON vide est créé [ ].
   * Sinon, le fichier existant est ouvert.
2. Ajout d'une nouvelle entrée JSON :
   * L'ancien fichier est ouvert et modifié sans casser le format JSON.
   * La nouvelle entrée est ajoutée à la liste avec une virgule de séparation.
   * Le fichier est mis à jour avec la nouvelle entrée.

**Exemple d’une entrée JSON générée :**

{

"Name": "Backup1",

"FileSource": "C:\\Documents\\File.txt",

"FileTarget": "D:\\Backup\\File.txt",

"FileSize": 1024,

"FileTransferTime": 1.5,

"EncryptionTime": 0.8,

"Date": "15/03/2025 14:32:10"

}

**c) Écriture dans un fichier XML**

**Processus d’enregistrement XML :**

1. Vérification de l'existence du fichier :
   * S’il n’existe pas, il est créé avec un élément <Logs> contenant le premier <LogEntry>.
   * Sinon, il est chargé et mis à jour.
2. Ajout d’un nouvel élément <LogEntry> :
   * Ajoute un élément enfant sous <Logs>.
   * Le fichier est mis à jour.

**Exemple d’une entrée XML générée :**

<LogEntry>

<Name>Backup1</Name>

<FileSource>C:\Documents\File.txt</FileSource>

<FileTarget>D:\Backup\File.txt</FileTarget>

<FileSize>1024</FileSize>

<FileTransferTime>1.5</FileTransferTime>

<EncryptionTime>0.8</EncryptionTime>

<Date>15/03/2025 14:32:10</Date>

</LogEntry>

* 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 :

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

Le contenu généré par l’IA peut être incorrect.

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/XML invalide"**  
🡪 Solution : Vérifier que le fichier JSON des logs (backup\_log\_yyyy-MM-dd.json/xml) 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.