

Présentation du logiciel Optimizeit Profiler pour le framework .Net

par [MORAND Louis-Guillaume](#)

Date de publication : 10/07/2005

Dernière mise à jour :

Tutoriel sur l'utilisation de Borland Optimizeit Profiler for .net



- 1 - Présentation d'un profiler
 - 2 - Présentation d'Optimizeit Profiler for .net
 - 2.1 - Lancement d'Optimizeit Profiler
 - 2.2 - Utilisation du Memory Profiler
 - 2.3 - Utilisation du CPU Profiler
 - 2.4 - Informations sur le CLR
 - 3 - Fonctionnalités complémentaires
 - 3.1 - L'export des données
 - 3.2 - L'intégration à Visual Studio .Net
 - 3.3 - Le visualiseur de code
- Conclusion
Téléchargements

1 - Présentation d'un profiler

Depuis que le développement logiciel existe, les développeurs se sont penchés sur les performances des logiciels, tout d'abord en développant des méthodes plus fonctionnelles, plus rapides puis en optimisant les accès à des ressources extérieures. Rapidement, cette technique a éprouvé des limites: en effet, l'utilisation récurrente et excessive de méthodes même optimisées (boucle infinie), peut entraîner une baisse de performance lors de l'exécution d'un programme et c'est là que le profiler entre en jeu.

Le profiler est un logiciel chargé d'étudier les performances d'un second programme en collectant des informations pendant que le code est exécuté. Comme vous le verrez par la suite, un profiler est capable d'étudier le temps utilisé par une portion d'un autre programme, pour ainsi détecter les méthodes qui prennent le plus de temps/ressources et qui seront alors l'objet même des optimisations à venir.

Enfin, le profiler est un logiciel exclusivement dédié à la période de test avant la mise en production, et non pas à la mise en production, car son utilisation et la collecte de d'informations d'un second programme divisent parfois par 10 (ou bien plus), les performances normales de ce dernier.

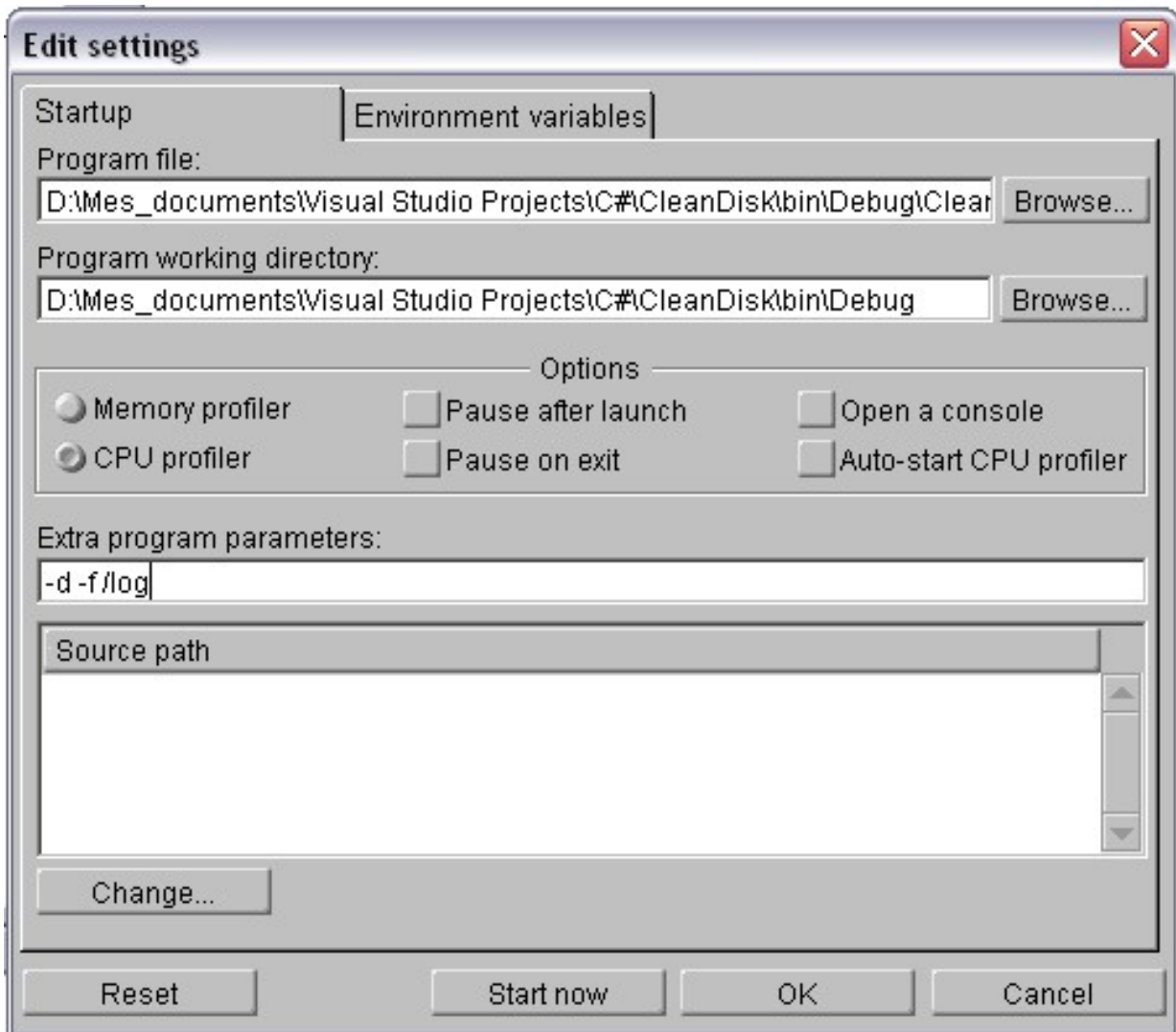
2 - Présentation d'Optimizeit Profiler for .net

Avec des fonctionnalités de gestion des performances complètes et détaillées, comme l'analyse du taux d'utilisation de la CPU et de la mémoire, Optimizeit Profiler aide les développeurs à identifier les obstacles aux performances, en accélérant la résolution des problèmes et en améliorant la productivité du programme. En effet grâce à différents outils, Optimizeit permet aux développeurs de contrôler leur code, d'écrire des applications plus stables et surtout plus rapides.

2.1 - Lancement d'Optimizeit Profiler

Comme vous l'avez devinez, le *profiling* un logiciel nécessite de spécifier à notre outil quel exécutable étudier. Heureusement pour nous, Optimizeit Profiler nous fourni un formulaire permettant de spécifier cette exécutable, son chemin de fonctionnement ainsi que d'éventuels paramètres. Ce formulaire va également vous demander quel type de profiling vous souhaitez effectuer.

C'est avant de commencer votre *profiling* que vous devez définir l'objectif de cette étude. Souhaitez-vous optimiser l'utilisation du CPU ou de la mémoire? Ou les deux? Quoi que vous choisissiez, Optimizeit Profiler va vous permettre de le réaliser en vous donnant la capacité d'aller au sein même (behind the scenes) du CLR (Common Language Runtime) pour identifier le code allouant trop de mémoire ou utilisant le processeur de manière inefficace.

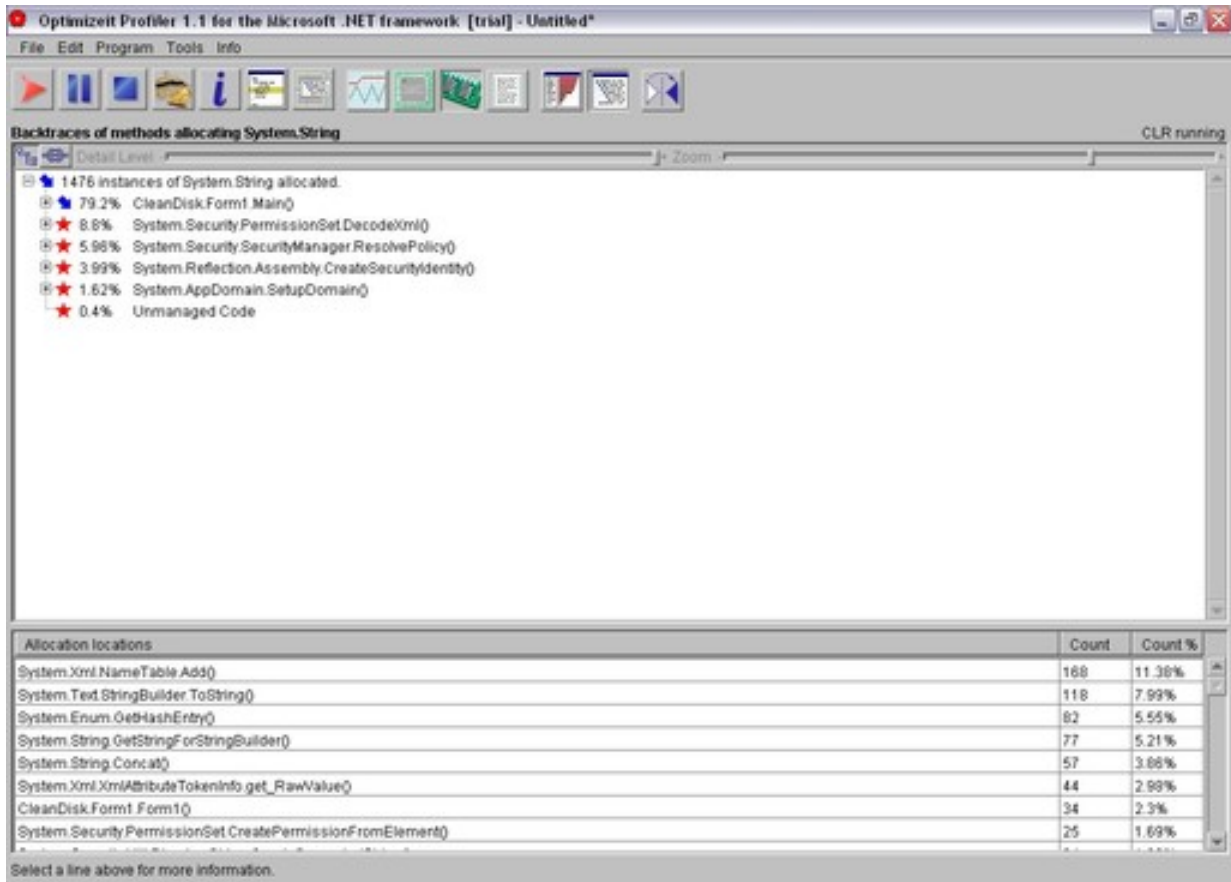


2.2 - Utilisation du Memory Profiler

Le Memory Profiler fournit un affichage en temps réel de toutes les classes utilisées par votre programme ainsi que le nombre d'instances de celles-ci.

Une fois ce mode de *profiling* lancé vous pouvez alors choisir entre l'affichage hiérarchique et agrégé. Le premier vous affiche les méthodes de façon hiérarchique ainsi que le nombre d'appels de chacune et le pourcentage mémoire utilisé à ce moment là. Cette fonctionnalité est alors extrêmement utile pour détecter les fuites mémoires.

L'autre possibilité est l'affichage agrégé. Cette vue dessine un diagramme des méthodes ainsi que le chemin utilisé par celles-ci, on peut alors repérer la suite logique des méthodes ou tout simplement détecter une boucle d'une méthode avec une autre. Cet affichage permet donc de détecter très rapidement les boucles infinies ou simplement les méthodes appelées un grand nombre de fois.

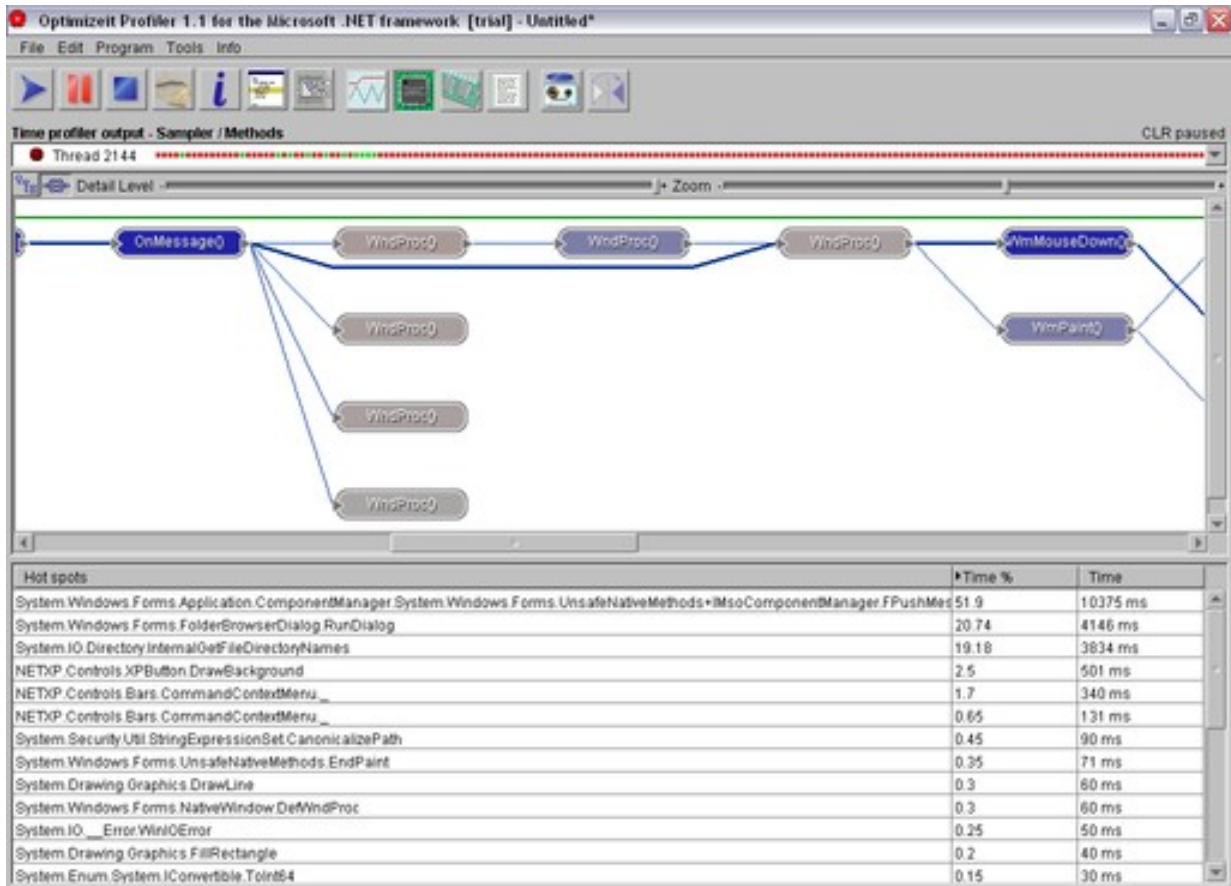


Avec le Memory Profiler, vous avez également la possibilité de voir le "heap"(tas) du CLR en temps réel. Vous pouvez ainsi voir les classes chargées et leur instances, mais également le mémoire utilisée par celles-ci.

Class name	Instance count	Diff.	Size	Size diff.
System.String	1834	None	123 K	None
System.Windows.Forms.MouseEventHandler	622	+ 31	17 K	+ 868 b
System.Collections.Hashtable	571	+ 36	20 K	+ 1 296 b
System.Byte []	442	None	3495 K	None
System.Drawing.Drawing2D.GraphicsS	248	None	3968 b	None
System.EventHandler	237	None	6636 b	None
System.Windows.Forms.PaintEventArg	232	None	9 K	None
System.Int32	221	None	2652 b	None
System.ComponentModel.EventHandle	182	None	2184 b	None
System.Windows.Forms.PropertyStore	176	None	9272 b	None
System.Windows.Forms.PropertyStore	175	None	2800 b	None
System.Windows.Forms.CreateParams	175	None	9100 b	None
System.Windows.Forms.Control.Contro	175	None	9100 b	None
System.RuntimeType	164	None	2624 b	None
System.Windows.Forms.LayoutEventAr	162	None	2592 b	None
System.Text.StringBuilder	151	None	3020 b	None
System.Object	127	None	1524 b	None

2.3 - Utilisation du CPU Profiler

Si vous avez décidé d'optimiser l'utilisation CPU, et que vous avez donc lancé le CPU Profiler, ce dernier vous permet d'afficher des informations sur les méthodes utilisant du temps processus, et ce, en fonction du thread auquel elles appartiennent.



L'affichage se fait en sur panels distincts. Le premier (celui du bas) présente les *hot spots*, c'est-à-dire les méthodes qui utilisent le plus de temps processus. Vous pouvez d'un seul coup d'œil trouver la ou les méthodes qui sont les points faibles de votre application. Vous pouvez alors les optimiser ou les mettre dans un thread séparé selon le cas, pour améliorer la vitesse générale de votre application.

Le second panel vous permet d'afficher les méthodes soit de façon hiérarchique, soit de façon agrégée. Le premier panel lui, à l'aide d'icônes, vous permet de voir si l'utilisation processus d'une méthode provient de la méthode en elle-même ou d'une méthode enfant. La seconde vue, vous permet elle, de contempler le parcours logique tracé par vos méthodes lors de l'utilisation courante de votre application.

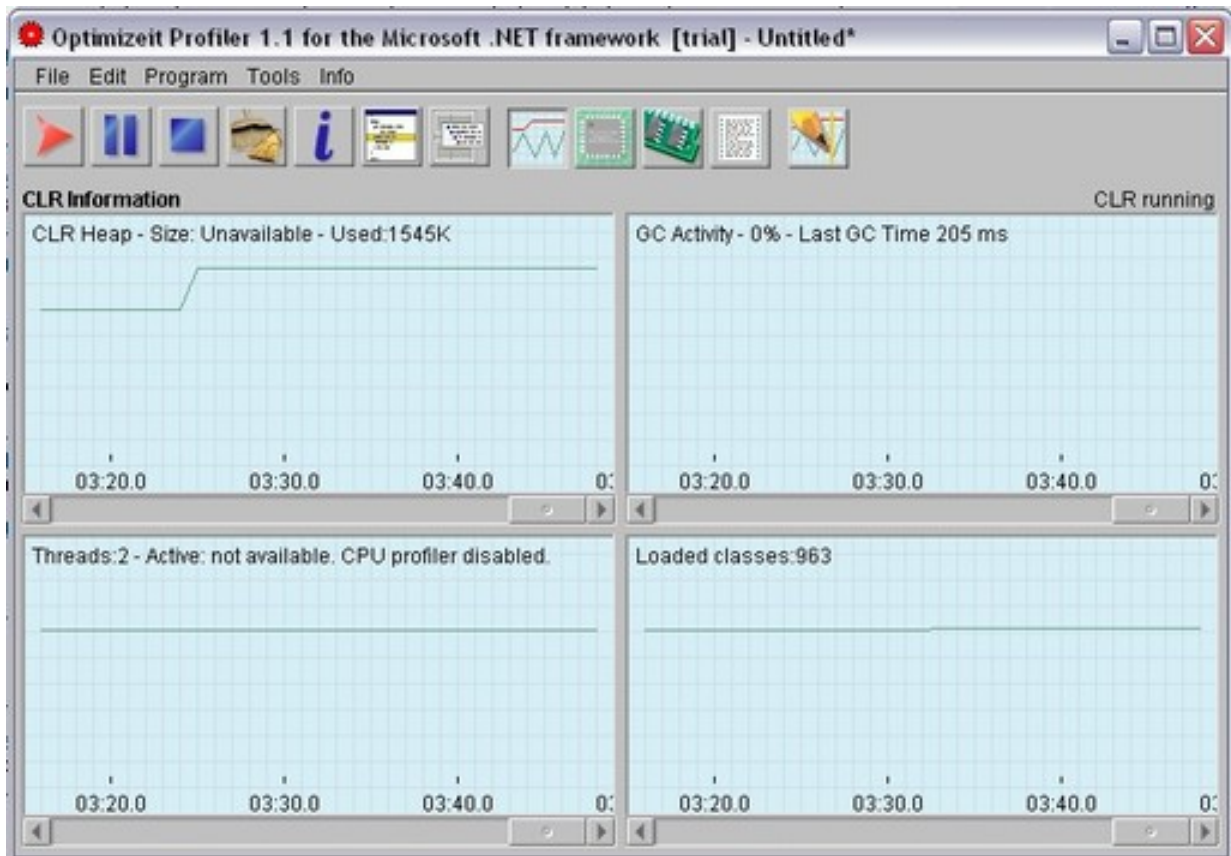
2.4 - Informations sur le CLR

Le troisième et dernier mode que vous propose Optimizeit est le mode CLR. Ce dernier mode vous permet de savoir si un problème est lié à la mémoire, au CPU ou au deux. Celui-ci vous indique en effet, les objets instanciés, leur nombre d'occurrence et les ressources qu'il demande. On peut ainsi voir rapidement si l'utilisation de tel ou tel objet n'est pas adapté à l'application courante.

Il est composé de 4 graphiques:

- le graphe "heap": celui présente la "pile" du CLR et la quantité de mémoire utilisée: cela comprend les classes, les objets, les piles des threads et la mémoire allouée par le CLR ou le code non managé.

- le graphe du garbage collector: celui-ci présente l'utilisation du garbage collector en temps réel. Bien utilisé il aide à définir si vos objets sont bien détruits au moment où vous le souhaitez.
- le graphe des threads: il fournit des informations sur les threads. Ceux qui existent et ceux qui utilisent réellement du temps processeur.
- le graphe des classes: celui-ci représente le nombre de classes actuellement chargées dans la CLR

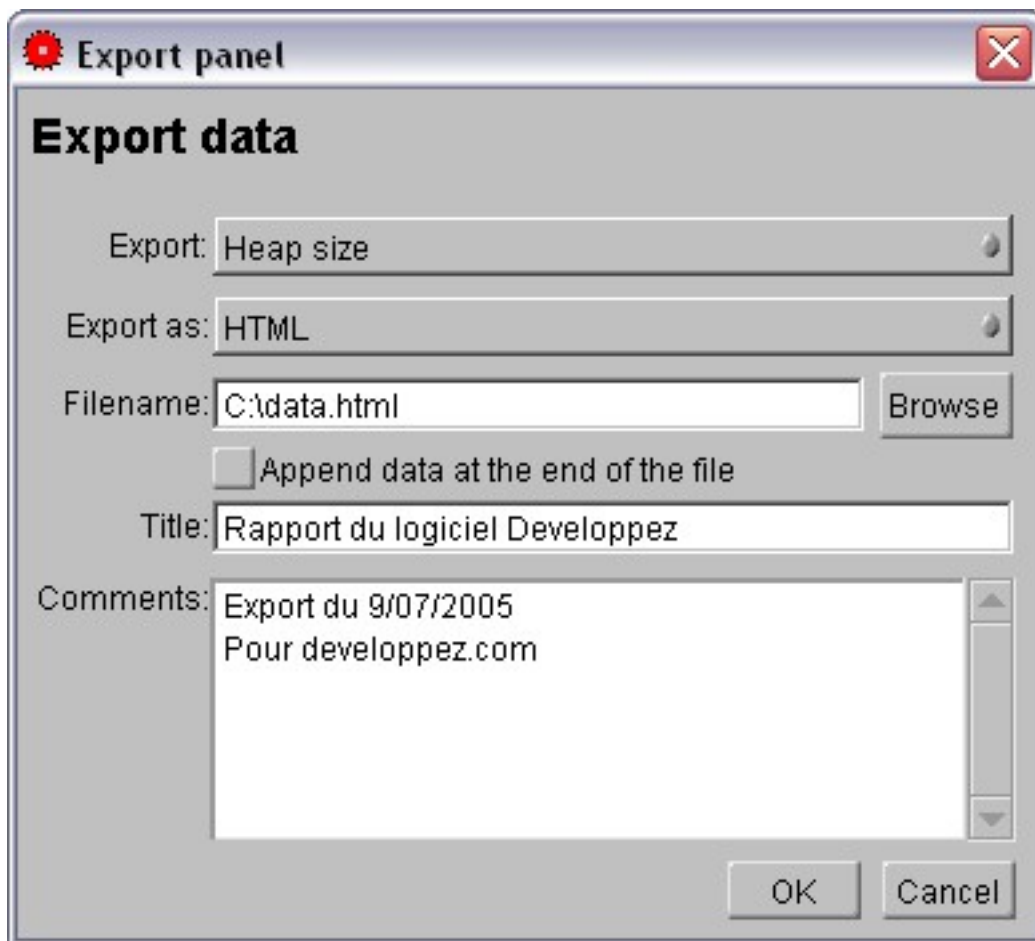


3 - Fonctionnalités complémentaires

3.1 - L'export des données

L'une des fonctionnalités utiles d'Optimizeit est l'export des résultats. En effet, même si le document exporté reste très simplifié, il représente une copie des résultats récupérés lors du "profiling" et permet ainsi de garder une trace pour une étude comparative si vous le souhaitez.

L'export se fait via le menu **Save > Export Data** ou Ctrl+E qui ouvre alors une fenêtre permettant de choisir les informations à exporter (filtrer ce que l'on a précédemment récupérer), ainsi que le format souhaité (ASCII, html ou Importable ASCII)



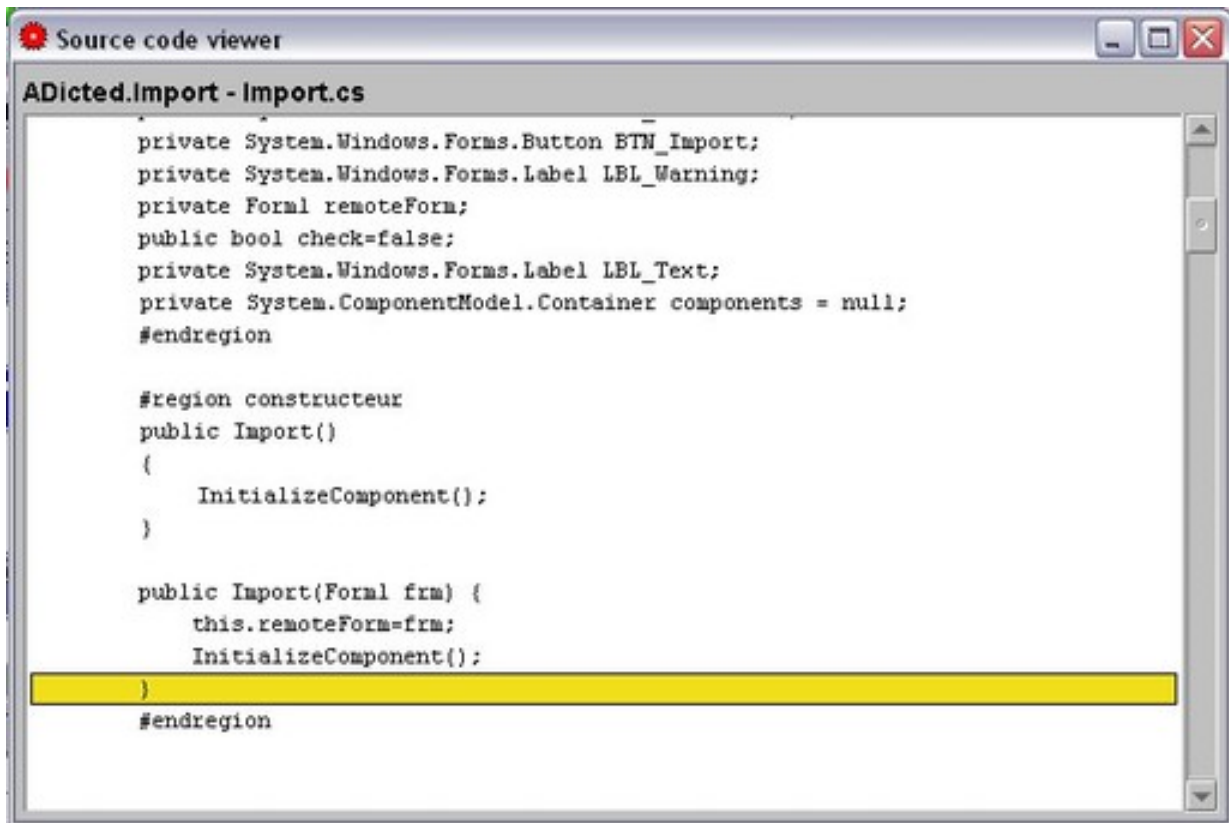
Fenêtre d'export

3.2 - L'intégration à Visual Studio .Net

Deuxième fonctionnalité bien utile d'Optimizeit, c'est la possibilité d'intégration dans l'ide de développement Microsoft Visual Studio .Net. L'"installation" se fait via le menu **Tool > IDE intégration > Visual Studio .Net**. Un assistant s'ouvre alors pour vous permettre de préciser quelle version de Visual Studio vous utilisez. Vous pouvez ensuite lancer Visual Studio .Net

3.3 - Le visualiseur de code

Cette petite extension vous permet de cliquer sur l'objet incriminé dans vos résultats de profiling et d'afficher le code auquel il appartient. Ainsi, si vous repérez une méthode utilisée excessivement et que vous ne savez où elle se trouve, double-cliquez simplement sur sa ligne pour afficher le code de sa classe.



```
Source code viewer
ADicted.Import - Import.cs

private System.Windows.Forms.Button BTN_Import;
private System.Windows.Forms.Label LBL_Warning;
private Form1 remoteForm;
public bool check=false;
private System.Windows.Forms.Label LBL_Text;
private System.ComponentModel.Container components = null;
#endregion

#region constructeur
public Import()
{
    InitializeComponent();
}

public Import(Form1 frm) {
    this.remoteForm=frm;
    InitializeComponent();
}
#endregion
```

Pour que le visualiseur de code fonctionne, il est impératif que votre exécutable ait été compilé en mode debug.

Conclusion

Optimizeit Profiler est l'outil adapté, le couteau suisse du développeur souhaitant optimiser son application, que ce soit pour trouver les fuites mémoires, les surcharges processeurs, la gestion des threads ou tout autre problème pouvant avoir lieu entre un code fonctionnel (sans erreur) et le CLR.

Téléchargements

[Article au format PDF](#)