

Développer une console MMC 3.0 avec .NET

par Lainé Vincent ([autres articles](#))

Date de publication : 14/02/2007

Dernière mise à jour : 14/02/2007

Dans cet article vous découvrirez comment développer une console d'administration à l'aide de .NET et de MMC 3.0

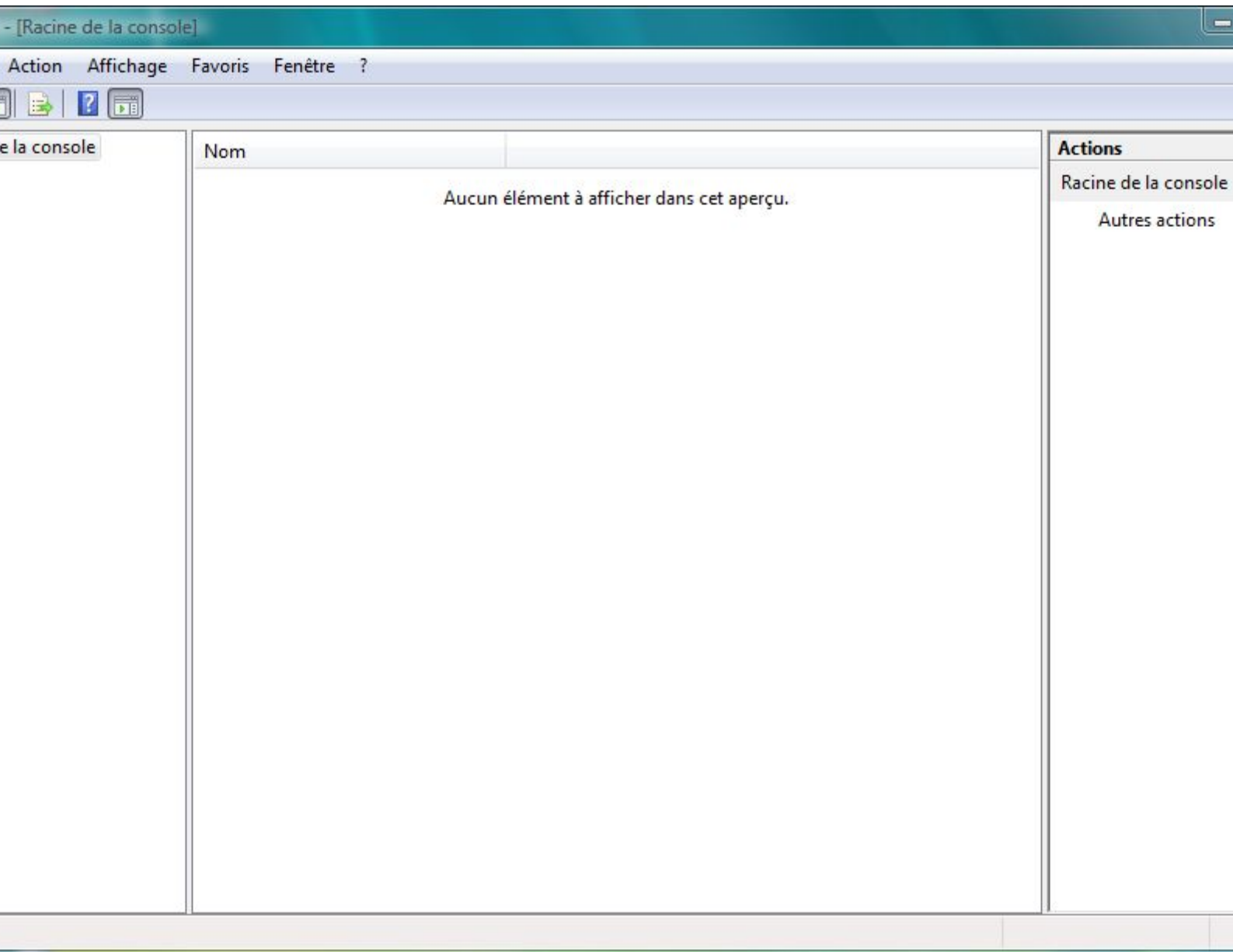
- 1 - Une MMC ? C'est quoi docteur ?
 - 1-1 - Pourquoi développer une MMC ?
- 2 - Pré-requis
- 3 - Création du projet et premier ajout de référence
- 4 - Développement de notre snap-in
 - 4-1 - La base du snap-in
 - 4-2 - Notre première vue ...
 - 4-3 - ... Et les autres vues simples
 - 4-4 - La vue FormView et les raccourcis
 - 4-5 - Ajouter une page de propriété sur un noeud
 - 4-6 - Conclusion sur le développement de snap-in
- 5 - Déploiement d'une console MMC
- 6 - Débogage d'une console MMC avec Visual Studio
- 7 - Conclusion
- 8 - Remerciements
- 9 - Téléchargements

1 - Une MMC ? C'est quoi docteur ?

Avant de rentrer dans la partie technique de la chose, il faut savoir ce qu'est une MMC. D'abord que veut dire MMC ? MMC c'est Microsoft Management Console, c'est-à-dire que ce sont des consoles permettant l'administration d'une partie du système d'exploitation, d'un serveur ou même d'une application. Sans le savoir vous manipulez sûrement déjà des MMC. En effet que ce soit l'event log, ou le gestionnaire des services locaux, ce sont deux exemples concrets de MMC.

Mais au delà des exemples de chez Microsoft, la plupart des logiciels ayant une partie serveur ou simplement étant un peu complexe possèdent une MMC.

En fait le terme MMC est assez mal utilisé. En effet il sert aussi bien à désigner le container MMC que le composant logiciel en fichage (terminologie Microsoft ;)). Dans cet article je m'efforcerai de faire la différence entre la MMC et les snap-in (le terme anglais en plus court) mais il peut rester des erreurs, je vous prie de bien vouloir m'en excuser.



La mmc sans snap-in de lancé

1-1 - Pourquoi développer une MMC ?

Assurément développer un snap-in n'est pas pour tout le monde. En effet si votre application ne possède pas de partie à configurer, ce n'est pas la peine de partir dans le développement d'un tel composant.

Avoir un snap-in pour son logiciel peut être intéressant dans les cas suivants :

- C'est une application client/serveur avec des paramètres communs à tous les utilisateurs
- C'est une application qui génère des informations de sécurité qui ne doivent être vus que par les administrateurs
- L'application est elle-même un snap-in et sert à configurer une autre application.

Ce sont là des cas d'utilisation mais ils résument assez bien l'usage des MMC

2 - Pré-requis

Afin de se lancer dans le développement, il vous faut quelques outils préalables. Passons sur l'incontournable Visual Studio, n'importe quelle version à partir de la 2005 fera l'affaire. D'ailleurs si le coeur vous en dit vous pouvez parfaitement utiliser le bloc-notes et le compilateur en ligne de commande.

Plus sérieusement la console MMC ne prend pas en charge directement les assemblies .NET, il lui faut wrapper natif, c'est à dire un morceau de code .NET permettant de faire la liaison entre votre code .NET et le code natif de la MMC. Heureusement pour nous ce wrapper à été développé par Microsoft et est disponible dans le SDK de Windows à l'adresse suivante : <http://www.microsoft.com/downloads/details.aspx?FamilyId=C2B1E300-F358-4523-B479-F53D234CDCCF&displaylang=en>

Toutefois si vous ne souhaitez télécharger tout le SDK de Windows pour simplement deux fichiers, je les mets à votre disposition à la fin de l'article.

Pour finir il vous faut évidemment la version 3.0 de la console MMC.

Celle-ci est installée par défaut sous Vista mais si vous êtes sous XP vous la trouverez encore une fois sur le site de Microsoft à [cette adresse](#).

3 - Création du projet et premier ajout de référence

Maintenant que nous savons ce qu'est une MMC et un snap-in et que nous possédons tout le matériel requis pour en développer une, nous pouvons nous mettre au travail.

Pour commencer il faut créer un projet de type Windows -> Librairie de classe. Le choix du langage n'a pas d'importance du moment que c'est un langage .NET. Vous pouvez donc parfaitement écrire votre MMC avec C++.NET.

Une fois ce nouveau projet créé, il ne reste plus qu'à ajouter la référence à la librairie microsoft.managementconsole.dll. A partir de là nous avons tout ce qu'il faut pour créer notre MMC.

4 - Développement de notre snap-in

4-1 - La base du snap-in

La première chose à faire pour créer un snap-in c'est de la définir. C'est à dire que nous devons lui donner un nom et un GUID (Global Unique Identifier).

Nous allons donc créer un fichier contenant ces informations ainsi que le point d'entrée du programme

Le code de base de notre snap-in

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.ManagementConsole;
using System.Threading;

namespace MMCStabilities
{
    /// <summary>
    ///
    /// </summary>
    [SnapInSettings(" {A0AD21F3-EABA-47ad-A8A9-F6E2806F7045} ",
        DisplayName = "NovaLabs SnapIn",
        Description = "NovaLabs Management SnapIn")]
    [SnapInAbout("MMCNovaLabs")]
    public class StabilitiesSnapIn : SnapIn
    {
    }
}
```

Voyons dans le détail ce code.

Premièrement nous avons les using qui vont bien et plus particulièrement celui-ci : *using Microsoft.ManagementConsole*; Ceci est possible grâce au précédement ajout de la référence.

Plus intéressant nous avons une classe qui hérite de SnapIn. Un petit tour dans la Msdn nous apprendra que c'est effectivement la classe de base pour toute extension à la MMC. Nous avons donc notre point d'entrée pour la MMC. Mais cet héritage ne suffit pas à définir un snap-in complet. En effet il manque des informations qui sont fournies par les attributs de classe. L'attribut SnapInSettings permet de définir le GUID de notre snap-in ainsi que le nom qui sera affiché dans la boîte de dialogue d'ajout de snap-in et sa description. D'autres paramètres sont disponibles mais je vous laisse le soin d'aller explorer la Msdn pour avoir les détails.

A partir de là nous avons bien un snap-in tout à fait correct sur le plan de la déclaration mais qui ne fait rien.

Avant de nous lancer dans la suite, prenons le temps de nous arrêter sur l'architecture visuelle des MMC. Les MMC sont divisées en 3 panneaux comme nous l'avons vu avant. Le panneau de gauche permet de sélectionner les éléments à explorer, le panneau central permet d'afficher les informations requises et le panneau de droite permet d'avoir des raccourcis vers les actions les plus fréquemment effectuées. Les MMC ne possèdent pas de menu personnalisable dans le sens où vous ne pouvez pas influencer directement sur la barre de menu en haut. Il faut donc trouver un autre moyen pour permettre les actions sur les différentes parties de notre application. La solution nous vient de la philosophie même des MMC. Tout se fait grâce au clic droit. En effet les menus permettant les actions sur les différentes parties des snap-in sont disponibles dans le menu contextuel. Dans notre développement ce menu nous est également accessible et surtout il est directement constitué des éléments que nous définissons dans le panneau d'action.

Une fois l'architecture « visible » explorée il faut maintenant comprendre comment fonctionne la « tuyauterie » interne des MMC 3.0 en .NET. L'organisation interne des MMC est entièrement basée sur la notion d'arbre. En effet toutes les parties d'une MMC sont rattachées à un élément de l'arbre affiché à gauche de la console. Un noeud de cet arbre permet de stocker le nom du noeud, la description qui est affichée en bas de la console, le texte à afficher ainsi que les éléments du panneau d'action et surtout la vue à afficher dans le panneau central. Cette vue définit la façon dont les items sont affichés et les items eux même. La « version .NET » des MMC permet de définir 4 formes d'affichages (Vues) :

- **MmcListView** : Les items sont de simples items de listview avec les options que nous connaissons de ce contrôle
- **HtmlView** : Cette vue permet d'afficher directement du contenu Html depuis n'importe quelle url
- **MessageView** : Permet d'afficher un texte simple avec un titre et une icone
- **FormView** : Cette vue permet de créer des usercontrol qui seront intégrés dans la MMC.

Chaque vue est accompagnée d'une classe « d'aide » nommée *TypeDeLaVueDescription* et qui permet de faire le lien entre la vue elle-même et la MMC. Elle permet de définir le nom interne de la vue, le type de la vue et si besoin, le contrôle rattaché à cette vue.

4-2 - Notre première vue ...

L'architecture des vues des MMC est un peu complexe mais un morceau de code facilitera votre compréhension :

Notre première vue

```
MessageViewDescription mvd = new MessageViewDescription();
    mvd.Title = "Bienvenue";
    mvd.BodyText = "Pour commencer veuillez sélectionner un des éléments dans le panneau de
gauche";
    mvd.IconId = MessageViewIcon.Information;
    mvd.DisplayName = "Accueil";
    mvd.ViewType = typeof(MessageView);
```

Dans cet exemple nous avons défini une vue de type *MessageView*, et instancié la classe de support *MessageViewDescription*. Notez que cet exemple est un peu particulier car c'est la classe de support qui définit les informations de la classe d'affichage.

Nous avons maintenant notre première vue mais elle n'est rattachée à aucun noeud de notre snap-in et donc elle ne s'affiche pas dans notre snap-in. Comme indiqué plus haut toutes les informations sont portées par les éléments de l'arbre à gauche de la MMC. Il est donc temps de voir comment peupler cet arbre.

En premier lieu il faut revenir sur notre classe qui hérite de *SnapIn*. En effet en dérivant de *SnapIn* nous avons accès à la propriété **RootNode** de type *ScopeNode*. Cette propriété nous permet de définir le noeud principal.

Définition d'un noeud

```
this.RootNode = new ScopeNode();
    this.RootNode.DisplayName = "NovaLabs";
    this.RootNode.ViewDescriptions.Add(mvd);
    this.RootNode.ViewDescriptions.DefaultIndex = 0;
```

La classe *ScopeNode* définit un certain nombre de propriétés intéressantes dans notre cas :

La propriété *DisplayName* permet comme son nom l'indique de spécifier le nom à afficher. La propriété *ViewDescriptions* permet de spécifier les différentes vues disponibles à ce niveau de l'arbre. Ainsi vous pouvez parfaitement avoir un noeud qui possède une vue de type *MmcListView* et une autre de type *FormView*. L'utilisateur

pourra faire le changement grâce aux raccourcis qui sont générés automatiquement par le système (elle est pas belle la vie ?). Nous avons donc maintenant notre première vue entièrement fonctionnelle.

4-3 - ... Et les autres vues simples

Les vues HtmlView et MmListView sont assez faciles à utiliser également, je vous laisse le soin d'étudier le code ci-dessous :

Les autres vues possible pour un snap-in

```
MmcListViewDescription mld = new MmcListViewDescription();
    mld.DisplayName = "Items d'accueil";
    mld.Options = MmcListViewOptions.SingleSelect;
    mld.ViewType = typeof(MmcListView);

    HtmlViewDescription hvd = new HtmlViewDescription();
    hvd.DisplayName = "Page d'accueil de dvp";
    hvd.ViewType = typeof(HtmlView);
    hvd.Url = new Uri("http://dotnet.developpez.com/");
```

4-4 - La vue FormView et les raccourcis

La dernière vue est un peu plus complexe car elle requiert plus d'intervention de notre part. En effet la vue basée sur un control winform s'initialise en plusieurs temps.

Voici déjà le code de base pour la création de notre vue FormView :

La vue FormView

```
FormViewDescription configurationViewDesc = new FormViewDescription();
    configurationViewDesc.ControlType = typeof(ConfigurationControl);
    configurationViewDesc.DisplayName = "conf";
    configurationViewDesc.ViewType = typeof(ConfigurationView);
```

A travers ce code nous voyons que nous utilisons uniquement des classes personnelles pour renseigner le ControlType et le ViewType. Cela est nécessaire afin que nous puissions prendre en charge l'initialisation du composant de manière assez fine. Une autre chose est à remarquer ici : Nous n'utilisons pas d'instance de nos classes mais seulement les types. En effet le runtime .NET se charge de créer les objets qui vont bien à partir de nos classes et cela grâce à la réflexion. Afin que ce mécanisme puisse fonctionner nous devons respecter une règle simple : Il faut absolument définir un constructeur public et sans arguments. Une fois ceci clairement expliqué nous pouvons nous pencher sur la première des classes qui nous intéresse : *ConfigurationView*.

Voici le code de cette classe :

Définition de notre propre vue héritant de FormView

```
public class ConfigurationView: FormView
{
    private ConfigurationControl configuration;

    protected override void OnInitialize(AsyncStatus status)
    {
        base.OnInitialize(status);
        configuration = (ConfigurationControl)this.Control;
    }
}
```

La première chose à noter est que notre classe hérite de FormView. En fait vous pouvez très bien vous passer de cette classe si vous ne souhaitez pas faire d'initialisation particulière lors de l'affichage.

Ici nous nous contentons de récupérer l'instance du contrôle qui est créé par le système afin de pouvoir si besoin est interagir dessus. Nous allons également en profiter pour agir sur la console MMC en ajoutant des éléments dans le panneau d'action. Voici à quoi ressemble notre initialisation maintenant :

La méthode d'initialisation

```
protected override void OnInitialize(AsyncStatus status)
{
    base.OnInitialize(status);

    configuration = (ConfigurationControl)this.Control;

    //Chargement du fichier de config
    Microsoft.ManagementConsole.Action loadConfigAction =
    new Microsoft.ManagementConsole.Action("Charger un fichier de configuration",
    "Charge un fichier de configuration au format Xml");
    loadConfigAction.Triggered += new
Microsoft.ManagementConsole.Action.ActionEventHandler(loadConfigAction_Triggered);

    //Sauvegarde du fichier de config
    Microsoft.ManagementConsole.Action saveConfigAction =
    new Microsoft.ManagementConsole.Action("Sauvegarder les paramètres",
    "Sauvegarde la configuration dans le fichier spécifié");
    saveConfigAction.Triggered += new
Microsoft.ManagementConsole.Action.ActionEventHandler(saveConfigAction_Triggered);

    ActionGroup ag = new ActionGroup("Configuration", "Gère la configuration");
    ag.Items.Add(loadConfigAction);
    ag.Items.Add(saveConfigAction);

    this.ActionsPaneItems.Add(ag);
}

private void loadConfigAction_Triggered(object sender, ActionEventArgs e)
{
    using (OpenFileDialog ofd = new OpenFileDialog())
    {
        ofd.Filter = "Fichier exécutable|*.exe";
        if (ofd.ShowDialog() == DialogResult.Cancel)
            return;

        this.configuration.LoadConfigurationFile(ofd.FileName);
    }
}

private void saveConfigAction_Triggered(object sender, ActionEventArgs e)
{
    this.configuration.SaveConfig();
}
```

Nous voyons bien pourquoi il est intéressant de garder une référence sur le control stocké dans notre vue. De plus si vous avez fait attention au code et à la logique des MMC vous remarquerez que toute l'architecture respecte le model MVC (Modèle Vue Contrôleur). C'est pour cela que nous avons autant de classe et d'étape pour un petit exemple mais c'est grandement appréciable lors du développement de console imposante.

4-5 - Ajouter une page de propriété sur un noeud

Une dernière chose reste à faire. Si vous avez l'habitude d'utiliser les MMC vous savez qu'il existe une entrée sous le clic droit nommé Propriétés dans laquelle nous retrouvons des informations généralement très utiles. Afin de faire apparaître cette option il faut hériter de la classe ScopeNode, surcharger la méthode « OnAddPropertyPages » et accéder à la liste des pages de propriétés. Cela nous donne le code suivant :

Définition de notre propre type de noeud

```
public class NovaLabsNode : ScopeNode
{
    // . . . Constructeurs
    protected override void OnAddPropertyPages(PropertyPageCollection propertyPageCollection)
    {
        base.OnAddPropertyPages(propertyPageCollection);

        PropertyPage page = new PropertyPage();
        page.Title = "Aide";
        page.Control = new UserControl();

        propertyPageCollection.Add(page);
    }

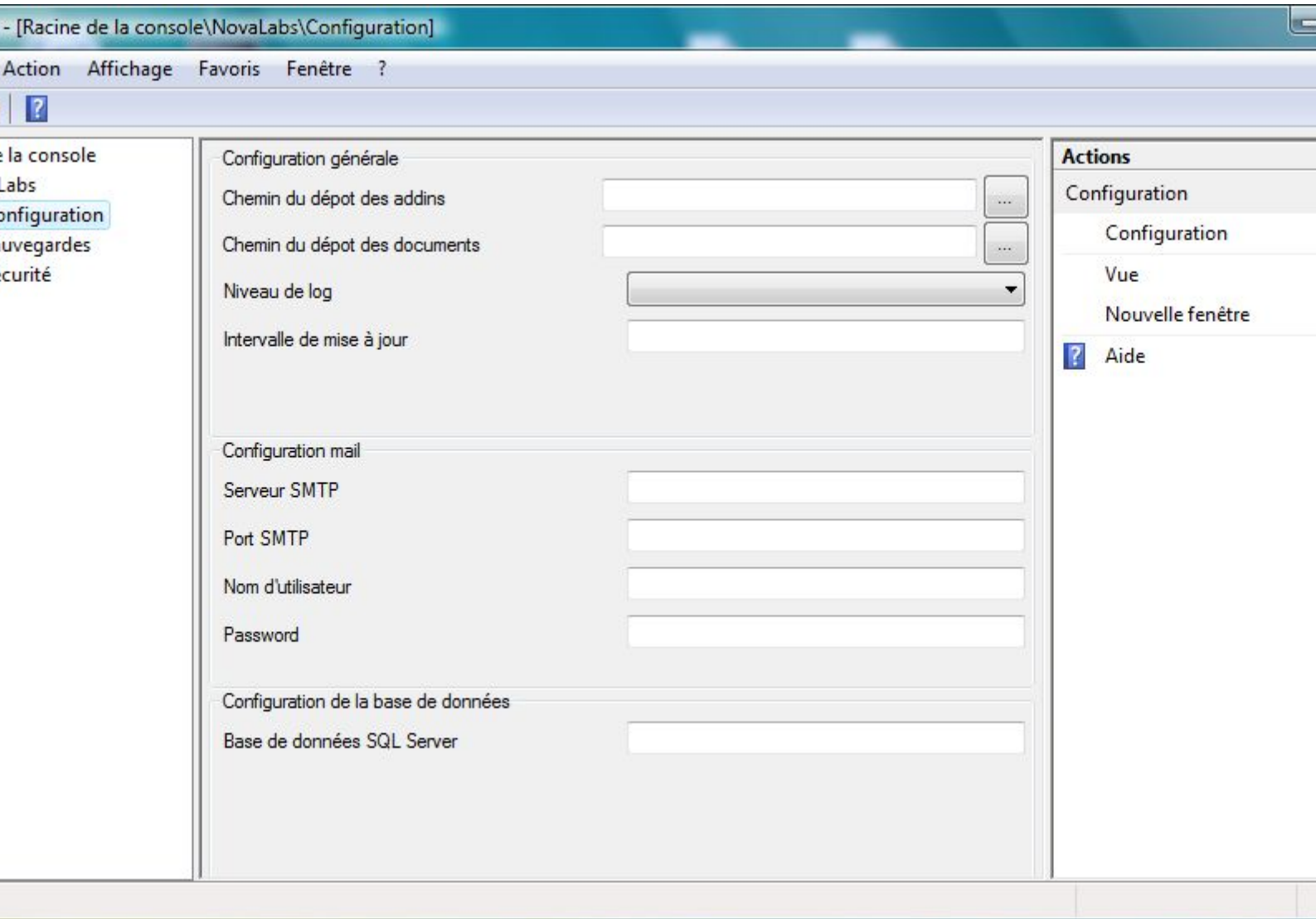
    private void Initialize()
    {
        MessageViewDescription mvd = new MessageViewDescription();
        mvd.Title = "Bienvenue";
        mvd.BodyText = "Pour commencer veuillez sélectionner un des éléments dans le panneau de
gauche";
        mvd.IconId = MessageViewIcon.Information;
        mvd.DisplayName = "Accueil";
        mvd.ViewType = typeof(MessageView);

        this.DisplayName = "NovaLabs";
        this.ViewDescriptions.Add(mvd);
        this.ViewDescriptions.DefaultIndex = 0;
        this.EnabledStandardVerbs = StandardVerbs.Properties;
    }
}
```

Les pages de propriétés sont assez simples d'utilisation dans le sens où leur seule demande est un UserControl tout bête et un titre. Par contre au niveau de la redéfinition du noeud il faut bien faire attention à renseigner la propriété **EnabledStandardVerbs** avec **StandardVerbs.Properties** sinon l'entrée du menu ne sera pas affichée et la méthode OnAddPropertyPages ne sera pas appelée.

4-6 - Conclusion sur le développement de snap-in

A travers cet article nous avons vu que la philosophie des MMC est basée sur le modèle MVC et sur l'héritage massif. En effet nombre d'éléments ne sont accessibles que lors de la redéfinition d'une classe comme par exemple les pages de propriétés.



Notre snap-in finalisé et installé dans la MMC

5 - Déploiement d'une console MMC

Maintenant que le code de notre snap-in est prêt, que nous avons tout compilé et que nous avons notre librairie .dll, encore faut-il pouvoir charger le tout dans une MMC. Cela est possible en passant par un projet d'installation que nous allons détailler rapidement.

Pour commencer voyons le code :

La classe d'installation

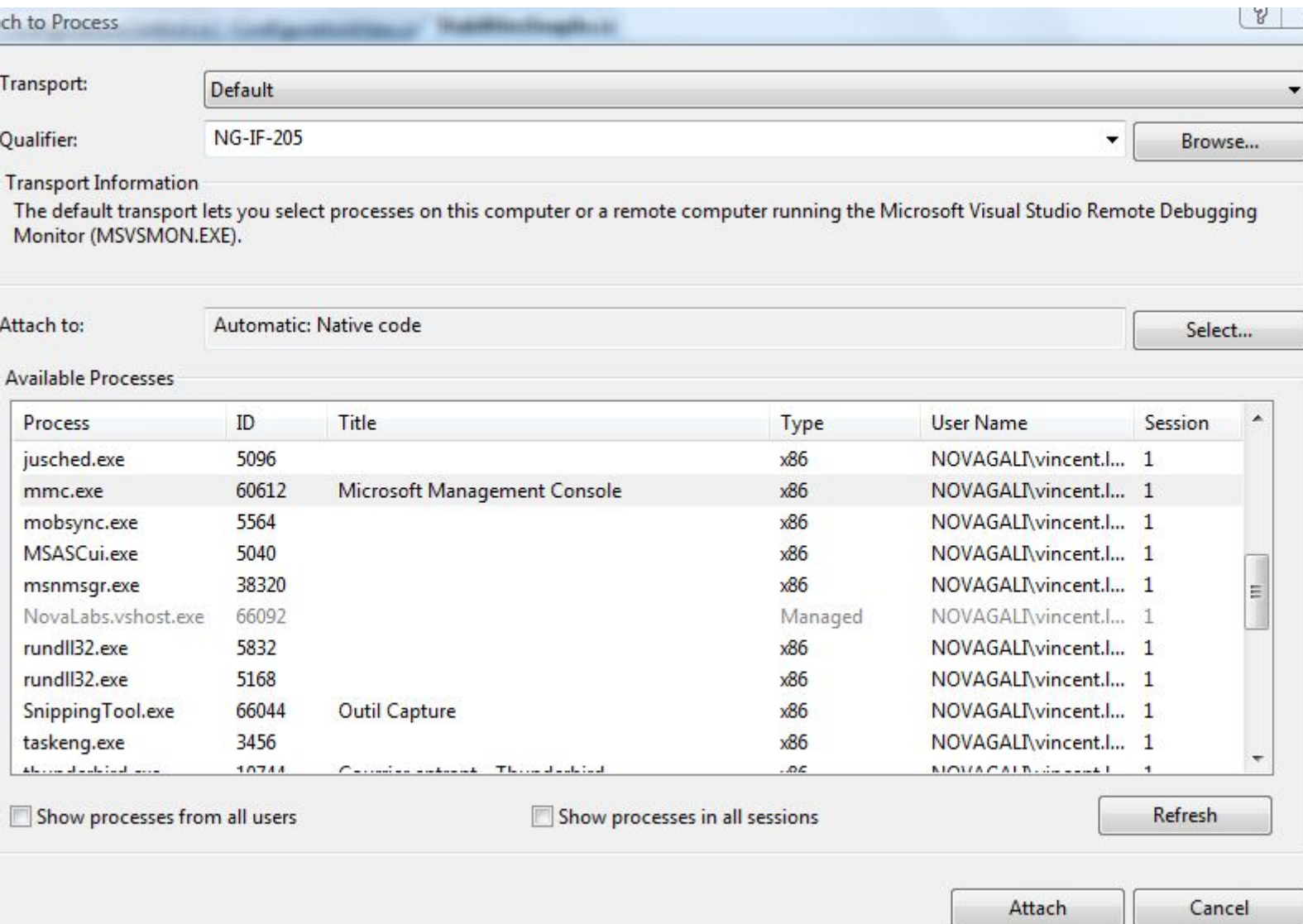
```
[RunInstaller(true)]  
public class MMCInstaller : SnapInInstaller  
{  
}
```

Et voilà nous avons fini :D. En effet la classe SnapInInstaller fait tout le travail pour nous en allant chercher les classes qui possèdent l'attribut SnapInSettings et qui héritent de SnapIn et en les inscrivant dans la console. Une fois l'installation effectuée (grâce à InstallUtil.exe qui se trouve dans le répertoire d'installation du SDK du Framework 2.0) notre snap-in est disponible dans une MMC.

Vous pouvez tout à fait le charger et enregistrer la console afin de pouvoir la lancer directement avec notre snap-in chargé.

6 - Débogage d'une console MMC avec Visual Studio

Il est complètement utopiste de penser que l'écriture d'une MMC est simple et ne souffrira d'aucun bug. Ainsi il est important de pouvoir effectuer un débogage comme si notre snap-in était une application winform par exemple. La technique consistant à créer un projet tierce et à y charger la librairie est assez compliquée à mettre en oeuvre et surtout ne permet pas de vérifier la totalité du code. Heureusement Visual Studio est là pour nous sauver la mise encore une fois. En effet le débogueur de Visual Studio est capable de s'attacher à n'importe quel processus et d'en surveiller les actions. Dès lors il devient intéressant d'attacher le débogueur au processus mmc.exe afin de surveiller le chargement des assemblies dans la console. Ainsi lors du chargement de notre snap-in managé, Visual Studio charge le fichier contenant les symboles de débogage correspondant et nous permet d'effectuer un débogage comme si toute l'application était managée.



Attach to Process

Transport:

Qualifier:

Transport Information
 The default transport lets you select processes on this computer or a remote computer running the Microsoft Visual Studio Remote Debugging Monitor (MSVSMON.EXE).

Attach to:

Available Processes

Process	ID	Title	Type	User Name	Session
jusched.exe	5096		x86	NOVAGALI\vincent.l...	1
mmc.exe	60612	Microsoft Management Console	x86	NOVAGALI\vincent.l...	1
mobsync.exe	5564		x86	NOVAGALI\vincent.l...	1
MSASCui.exe	5040		x86	NOVAGALI\vincent.l...	1
msnmsgr.exe	38320		x86	NOVAGALI\vincent.l...	1
NovaLabs.vshost.exe	66092		Managed	NOVAGALI\vincent.l...	1
rundll32.exe	5832		x86	NOVAGALI\vincent.l...	1
rundll32.exe	5168		x86	NOVAGALI\vincent.l...	1
SnippingTool.exe	66044	Outil Capture	x86	NOVAGALI\vincent.l...	1
taskeng.exe	3456		x86	NOVAGALI\vincent.l...	1
thunderbird.exe	10744	Thunderbird	x86	NOVAGALI\vincent.l...	1

Show processes from all users Show processes in all sessions

Attachement du debugger au processus mmc

7 - Conclusion

A travers cet article nous avons vu que Dotnet nous permet d'interagir avec la partie d'administration des OS Windows.

Cela montre bien la place de plus en plus grande que prend Dotnet dans l'écosystème de Microsoft.

8 - Remerciements

Je remercie toute l'équipe dotnet et plus particulièrement Stéphane Eyskens pour sa correction orthographique et Laurent Dardenne pour ses conseils.

9 - Téléchargements

Le zip contenant les deux bibliothèques nécessaires au développement (et uniquement au développement) d'une MMC 3.0 .NET : **[c'est par ici](#)**

Le zip contenant les sources du projet : **[Source](#) c'est par ici (Attention c'est un projet VS 2008)**

