

TD Objets distribués n°3 : Windows XP et Visual Studio .NET

Introduction à .NET Remoting

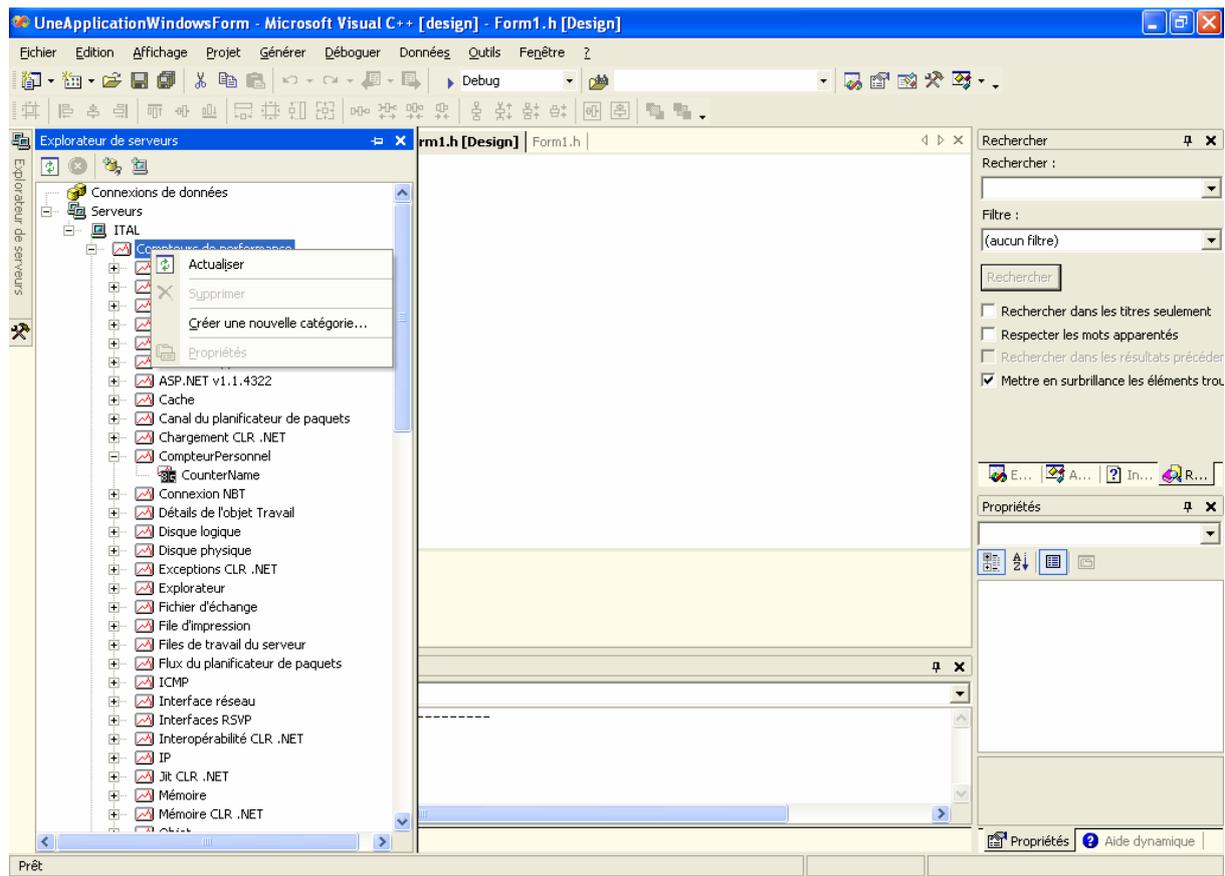
Partie 1 : l'analyseur de performances et compteurs

L'analyseur de performances de Windows (Démarrer -> Outils d'administration -> Performances sous Windows XP) prend en charge l'analyse détaillée de l'utilisation des ressources du système d'exploitation.

Une application VC++ avec une ressource à surveiller

Développons une application utilisant une certaine ressource dont son activité peut être quantifiée par un compteur (par exemple un interrupteur ON/OFF, un canal FIFO avec des caractères) et créons une application mesurant l'utilisation de cette ressource : nous exploiterons l'analyseur de performance de Windows pour tester notre application.

Notre application devra gérer une instance de compteur de performances : elle incrémentera ou décrémentera les valeurs d'un compteur 'personnalisé' (les compteurs 'non personnalisés' présents sur une machine n'autorisent qu'un accès en lecture seule à leurs informations). Utiliser l'explorateur de serveur pour accéder à la liste des compteurs sur de votre machine. Pour créer de nouveaux compteurs sur votre machine, utiliser le nœud '**compteurs de performance**'.

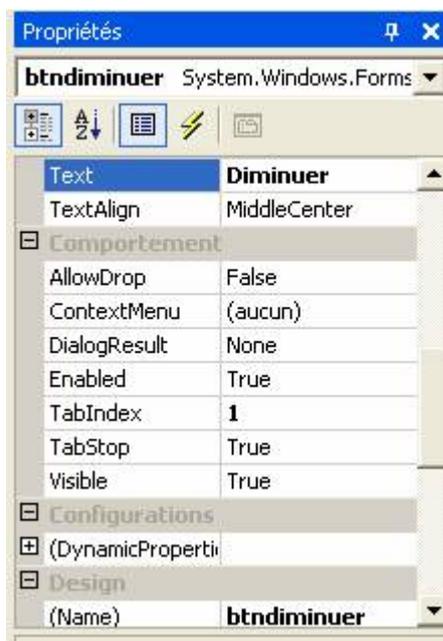
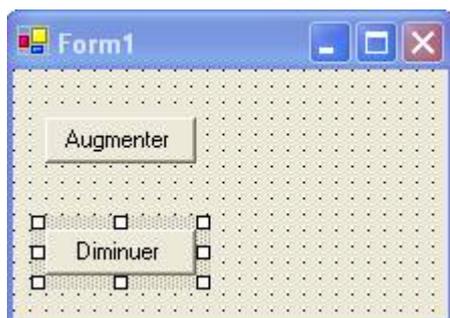


Pour un nouveau compteur personnalisé, il faut créer une nouvelle catégorie de compteurs de performance. Pour cela, **Bouton Droit -> Nouvelle catégorie**. Il suffit ensuite de renseigner les champs « **Nom** » et « **description** » de la catégorie depuis la boîte de dialogue « **Générateur de compteurs de performances** », nous allons ajouter un **nouveau compteur** de performance à la machine locale.

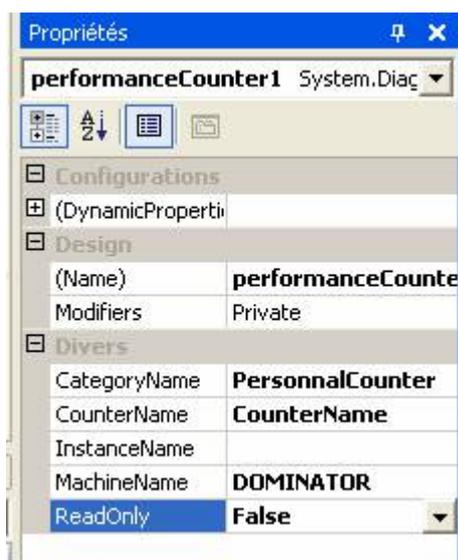
Après avoir renseigné les informations sur le nouveau compteur, il ne reste plus qu'à valider l'opération et notre nouveau compteur apparaît dans la catégorie correspondante dans le nœud « **compteurs de performances** » de notre machine locale.



Créons maintenant notre 'Application Windows Form' (Menu Fichier/nouveau projet/Projet Visual C++/.NET : choisir "application Windows Forms") possédant 2 boutons 'Augmenter' et 'Diminuer' (dans notre cas les noms et le texte sont respectivement « **btnAugmenter** », « **Augmenter** » et « **btnDiminuer** », « **Diminuer** »).



Nous allons maintenant associer à ces boutons les fonctions d'incrémentation et de décrémentation de notre compteur. La première chose à faire consiste à sélectionner le nouveau compteur créé dans notre nouvelle catégorie et à **glisser déposer** notre compteur dans la forme principale de notre application. Cela aura pour effet de rajouter un composant « **non visuelle** » appelé « **performanceCounter1** » dans la barre des composants de VS.NET. Pour être sûr de pouvoir modifier les valeurs de notre compteur, nous allons changer la propriété « **ReadOnly** » de notre nouvel objet en la positionnant à « **False** ».



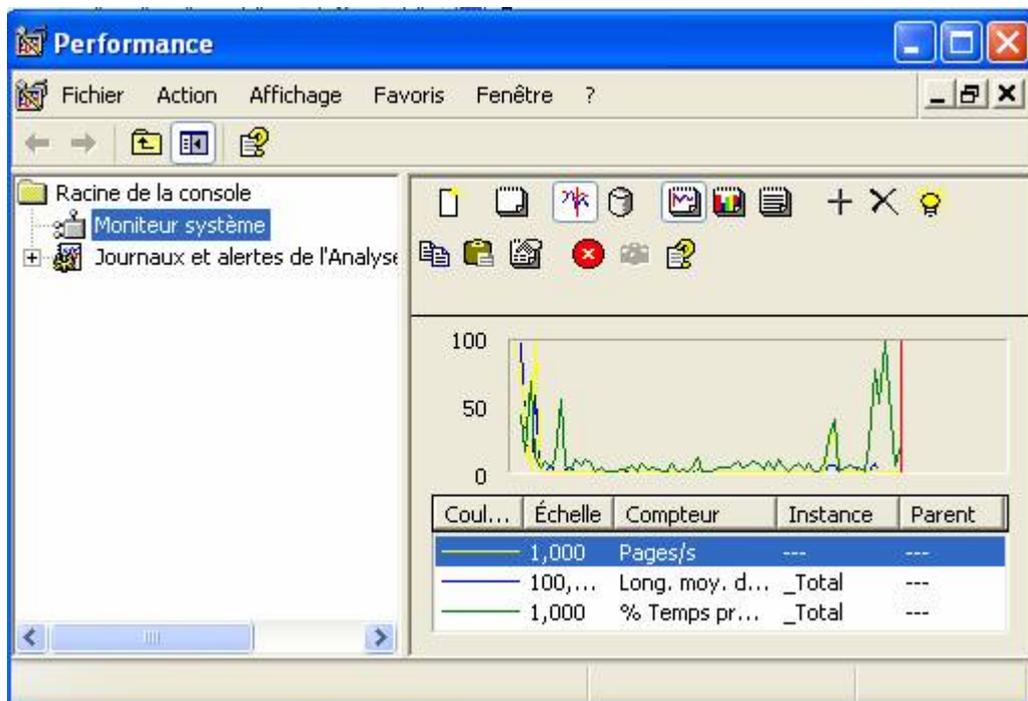
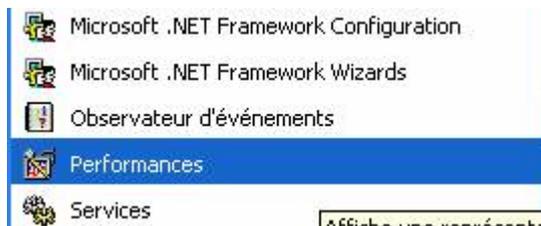
Il est question maintenant d'incrémenter ou de décrémentation notre compteur en gérant les évènement **Click** des boutons « Augmenter » et « Diminuer ». Pour cela, il suffit de créer les gestionnaires de l'évènement « **Click** » de chaque bouton en double-cliquant sur ses boutons

dans le Designer. Une fois cela fait, il suffit ensuite d'appeler les méthodes « **Increment** » et « **Decrement** » de la classe « **PerformanceCounter** ». Voici le code correspondant :

```
private void btnAugmenter_Click(object sender, System.EventArgs e)
{
    performanceCounter1.Increment();
}

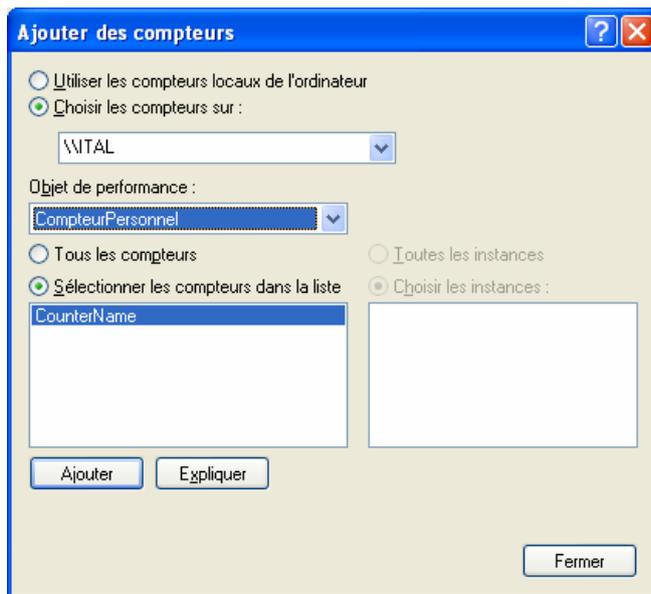
private void btndiminuer_Click(object sender, System.EventArgs e)
{
    performanceCounter1.Decrement();
}
```

Il est désormais possible d'utiliser le « **moniteur de performance** » de Windows (Démarrer -> Outils d'administration -> Performances sous Windows XP) pour effectuer les tests de notre application et utiliser notre nouveau compteur.

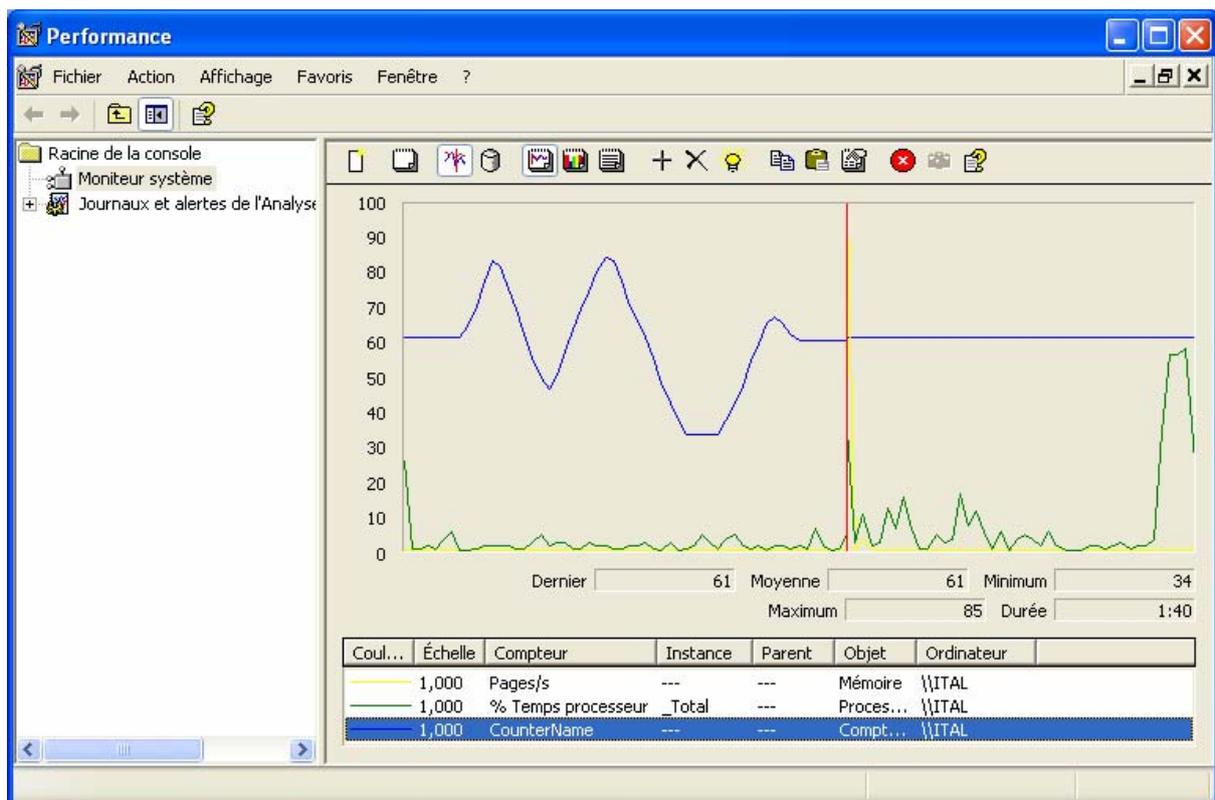


Nous allons ajouter notre nouveau compteur personnalisé dans le moniteur système en utilisant le bouton + (  ) dans la barre d'outils du moniteur système.

En tant que « **objet de performance** », ajoutons la catégorie précédemment créée (CompteurPersonnel) et choisissons notre compteur dans cette catégorie (ici CounterName).



Il suffit ensuite de cliquer sur le bouton « **Ajouter** » pour ajouter notre nouveau compteur. Maintenant, exécutons notre application : cliquons sur chacun des boutons « **Augmenter** » ou « **Diminuer** » pour se rendre compte que cela fait varier les relevés instantanés du moniteur de performance sur notre compteur.



Partie 2 : Introduction au .NET Remoting et compteurs...

Dans cette partie, vous mettrez en œuvre votre première application en .NET remoting. Les sources sont disponibles sur :

<http://dept-info.labri.fr/~felix/Annee2005-06/LicencePro/Remoting/>

Lancer Visual Studio .NET 2003 et créez une nouvelle solution "PremierRemoting". Nos clients et serveurs seront des applications consoles que nous programmerons en C# :

1. Ajoutez dans votre solution une nouvelle Application Console que vous nommerez "Serveur".
2. Ajoutez une nouvelle Application Console que vous nommerez "Client".
3. Ajoutez enfin un nouveau projet de type "Class library" que vous nommerez "IRemote". Il s'agit là de notre interface.

Objet Compteur 'Côté serveur'

Nous allons créer un Objet "Compteur" comme ceci :

```
public class Compteur
{
    public int iID;
    public string strNom;
    public double dNiveau;
    public void Augmenter(double dValeur)
    {
        this.dNiveau+=dValeur;
        Console.WriteLine(" - Compteur Augmenté, nouveau niveau : " +
this.dNiveau.ToString() + " -");
    }
}
```

Une fonction permet d'augmenter la valeur du compteur.

Ajouter ensuite cette fonction dans l'interface afin que le client puisse l'utiliser.

```
public interface ICompteur
{
    void Augmenter(double dValeur);
}
```

Le serveur.

Ajouter au projet IRemote une référence à la dll .NET qui s'appelle System.Runtime.Remoting. Cette dll contient toutes les classes nécessaires au remoting.

Création de l'objet qui sera servi : on crée une classe que l'on appelle CompteurManager. Cette classe implémente l'interface que l'on a définie plus haut. Elle doit donc implémenter les méthodes qui sont listées dans l'interface.

```
public class CompteurManager:MarshalByRefObject,IRemote.ICompteur
{
    DemarreServeur myclass = new DemarreServeur();
    //C'est cette classe que l'on distribue sur le reseau
    public override object InitializeLifetimeService()
    {
        //Durée de vie de l'objet : pour "toujours"
        return null;
    }
    public void Augmenter(double dMontant)
    {
        myclass.monCompteur.Augmenter(dMontant);
    }
}
```

Créons une classe RemotingServer, avec une méthode qui "démarré" le serveur. Il faut spécifier plusieurs choses dans cette méthode :

- le canal de transmission, et quel protocole il utilise (HTTP, TCP etc...)
- le port du canal (éviter certains ports...)
- le mode de transmission des données (il y a deux choix possible : Singleton ou SingleCall)
 - Singleton : Il n'y a uniquement qu'une seule instance de l'objet au même moment. On peut leur associer une durée de vie (quelques secondes ou éternel). Après la durée de vie écoulée, le serveur recrée une instance de l'objet à nouveau et la servira aux clients jusqu'à la fin de durée de vie.
 - SingleCall : une nouvelle instance de l'objet sera créée à chaque appel et détruite juste après.

```
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

public class RemotingServer
{
    public void Start()
    {
        int canal;
        //Ouvre un canal en mode TCP sur le port '1236'
        canal=1236;
        TcpChannel chnl_1 = new TcpChannel(canal);
        ChannelServices.RegisterChannel(chnl_1);

        //Publier notre objet CompteurManager

        RemotingConfiguration.RegisterWellKnownServiceType(typeof(CompteurManager),
"CompteurManager", WellKnownObjectMode.Singleton);
    }
}
```

On spécifie dans la méthode RegisterWellKnownServiceType que c'est la classe CompteurManager que l'on sert. Si vous voulez que votre objet dure pour "toujours", il vous suffit de surcharger une méthode qui s'appelle InitializeLifetimeService. En retournant "null" dans cette méthode, la durée de vie est ainsi infinie. Mais vous pouvez aussi mettre une durée spécifique.

Finalement, écrivons un code qui démarre le serveur et attend des instructions dans un nouveau module :

```
class DemarreServeur
{
    public Compteur monCompteur = new Compteur();
    [STAThread]
    static void Main(string[] args)
    {
        Serveur.RemotingServer myServer = new Serveur.RemotingServer();
        try
        {
            Console.WriteLine(" Serveur : Demarrage ...");
            myServer.Start();
            Console.WriteLine("Serveur démarré...");
            Console.ReadLine();
        }
        catch (Exception ex)
        {
            Console.WriteLine("Serveur : Erreur d'initialisation !!!
" + ex.Message);
        }
    }
}
```

Dans le sub main, appelez la méthode `RemotingServer.Start`. Compiler et exécuter le serveur.

Le client qui utilise le service :

Ouvrez le même canal que le serveur, et prenez une référence à l'objet servi. Ensuite il vous suffit d'appeler les méthodes par cette référence :

```
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
namespace Client
{
    class Client
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            try
            {
                //Enregistrer le canal TCP port 1234
                TcpChannel chnl = new TcpChannel();
                ChannelServices.RegisterChannel(chnl);
                //Prendre la référence sur le serveur (tcp://nommachine:port/CompteurManager)
                IRemote.ICompteur CompteurDistant =
                    (IRemote.ICompteur)Activator.
                    GetObject(typeof(IRemote.ICompteur),
                    "tcp://localhost:1236/CompteurManager");
                //Verifier que la référence au serveur a été acquise
                if (CompteurDistant == null)
                {
                    Console.WriteLine("Référence non acquise");
                }
                else
                {
                    Console.WriteLine("Référence au serveur acquise ");
                }
                //On peut à présent appeler les méthodes CompteurDistant
                CompteurDistant.Augmenter(1);
                CompteurDistant.Augmenter(2);
                CompteurDistant.Augmenter(3);
                //Attendre les instructions sur la console...
                Console.ReadLine();
            }
            catch (Exception ex)
            {
                Console.WriteLine("ERREUR :" + ex.Message);
            }
        }
    }
}
```

Lancer le client et vérifier que :

- le serveur crée une fois l'instance `Compteur`,
- les appels aux méthodes ont été reçus par l'objet `Compteur` (sur la console du serveur),
- si vous arrêtez le client en appuyant sur une touche et le relancez sans arrêter le serveur, vous verrez qu'il s'agit du même objet `Compteur` sur le serveur qui est utilisé depuis le début et qui continue d'être augmenté (effet du mode Singleton),

Exercice :

Tester votre application avec client et serveur sur machines séparées.

Rajouter et utiliser des méthodes pour :

1. diminuer la valeur du compteur,
2. afficher la valeur du compteur chez le client,

Ecrire la classique application 'Hello World' en version Distribuée.