

Serveur

Les nouvelles API de Windows Server : API Application Recover et Restart

Quel développeur n'a jamais essayé de mettre en place sur une application qu'il développe un système efficace de sauvegarde du travail effectué par l'utilisateur en cas de plantage ? La principale difficulté dans ce scénario est de pouvoir détecter que l'application ne répond plus, et de lui demander malgré tout d'exécuter une routine de sauvegarde/récupération puis d'éventuellement redémarrer en restaurant automatiquement les données.

Microsoft Windows Server 2008 propose de nouvelles API qui permettent de déléguer au système d'exploitation une partie de ce processus de récupération d'état de façon très efficace. Nous nous attarderons ici sur l'API Application Recover & Restart (ARR).

Application de test

Afin d'expliquer le fonctionnement de cette API, nous allons imaginer une simple application de traitement de texte créée avec le Framework .NET. Nous allons également ajouter un bouton qui provoquera un gel de l'application afin de simuler un plantage.

API Application Recovery & Restart

Une application peut utiliser l'API Application Recovery & Restart pour sauvegarder les données ainsi que son état avant qu'elle ne soit définitivement fermée suite à une exception non gérée ou un gel de l'interface. L'application peut alors être redémarrée et son état restauré si l'utilisateur le désire. Pour cela, il convient d'utiliser un certain nombre de fonctions :

- `RegisterApplicationRestart` : Inscrit l'application à redémarrer
- `RegisterApplicationRecoveryCallback` : Précise la méthode à appeler pour sauvegarder les données et l'état de l'application au cas où l'application plante ou ne répond plus
- `ApplicationRecoveryInProgress` : Indique à l'OS que l'application est en cours de sauvegarde
- `ApplicationRecoveryFinished` : Indique que la sauvegarde est terminée

Ces fonctions n'étant accessibles que dans la DLL `kernel32.dll`, il est nécessaire de faire un peu d'interop pour y accéder depuis une application .NET :

```
[DllImport("kernel32.dll", CharSet = CharSet.Auto)]
public static extern uint RegisterApplicationRestart(string pwzCommand
line, int dwFlags);
[DllImport("kernel32.dll")]
public static extern uint RegisterApplicationRecoveryCallback(APPLICATION
_RECOVERY_CALLBACK pRecoveryCallback, object pvParameter, int dw
PingInterval, int dwFlags);
[DllImport("kernel32.dll")]
public static extern uint ApplicationRecoveryInProgress(out bool pb
Cancelled);
[DllImport("kernel32.dll")]
public static extern uint ApplicationRecoveryFinished(bool bSuccess);
public delegate int APPLICATION_RECOVERY_CALLBACK(object pvParameter);
```

Pour plus d'informations sur les signatures de ces différentes fonc

tions, rendez vous sur cette page du site MSDN : [http://msdn2.microsoft.com/en-us/library/aa373342\(VS_85\).aspx](http://msdn2.microsoft.com/en-us/library/aa373342(VS_85).aspx)

La prochaine étape consiste à mettre en place dans l'application un système qui enregistrera périodiquement (ici toutes les 5 secondes) le texte présent dans notre TextBox. Pour ce faire, nous allons utiliser un simple Timer :

```
void _elapsedTime_Tick(object sender, EventArgs e)
{
    _time Text = string.Format("{0} secondes ", ++_elapsedTimeTicks);
    if ((_elapsedTimeTicks % 5) == 0)
    {
        using (FileStream fs = new FileStream("~/lastChanges.txt",
        FileMode.Create))
        {
            System.Text.UTF8Encoding encoding = new System.Text.UTF8
            Encoding();
            byte[] rawText = encoding.GetBytes(_myDocument.Text);
            fs.Write(rawText, 0, rawText.Length);
            fs.Close();
        }
    }
}
```

Ajoutons maintenant la méthode permettant de recharger le document `~/lastChanges.txt` dans notre TextBox :

```
private void Recover()
{
    using (StreamReader sr = new StreamReader("~/lastChanges.txt"))
    {
        _myDocument.Text = sr.ReadToEnd();
    }
}
```

Nous allons également utiliser une boucle infinie pour simuler un gel de l'application. Reste maintenant à modifier la fonction `Main` afin d'enregistrer notre application auprès du service ARR via les fonctions `RegisterApplicationRestart` et `RegisterApplicationRecoveryCallback`.

```
[STAThread]
static void Main(string[] args)
{
```

```
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
ApplicationRecovery.RegisterApplicationRestart("Crashed", 0);
ApplicationRecovery.RegisterApplicationRecoveryCallback(Recovery, null,
30000, 0);
Form1 myForm = new Form1(args.Length == 1 && args[0] == "Crashed");
Application.Run(myForm);
\
```

ApplicationRecovery.RegisterApplicationRestart("Crashed", 0) enregistre notre application auprès du service ARR de l'OS. Nous indiquons ici que la chaîne de caractère "Crashed" devra être passée en paramètre à la prochaine instance de l'application si elle est redémarrée, afin de lui indiquer dans quel contexte elle démarre. Le paramètre 0 indique à l'OS de redémarrer l'application quelle que soit la raison du plantage.

ApplicationRecovery.RegisterApplicationRecoveryCallback(Recovery, null, 30000, 0) permet de préciser le nom de la méthode à appeler en cas de plantage de l'application, c'est précisément dans cette méthode que la sauvegarde de l'état de l'application sera effectué. Dans notre cas, la sauvegarde de l'état de l'application étant déjà prise en charge avec notre Timer, la méthode Recovery n'aura aucun code supplémentaire pour sauvegarder l'application. Le deuxième argument, positionné à null, représente un " object " qui sera passé à la méthode Recovery si besoin. 30 000 représente en millisecondes le temps laissé à l'application pour indiquer à l'OS qu'elle est bien en train de sauvegarder son état. Passé ce délai, Windows considérera que l'application n'est plus en mesure de sauvegarder son état, et annulera cette procédure. Le dernier paramètre est réservé, et doit être égal à 0.

Remarquez enfin la modification de la signature du constructeur de la classe Form1 où nous passons maintenant en paramètre un booléen indiquant si l'application a été redémarrée suite à un plantage. Cette valeur sera utilisée par l'instance de Form1 pour savoir s'il est nécessaire d'appeler la méthode Recover avant affichage de la form.

```
public Form1(bool isRestarting)
{
InitializeComponent();
if (isRestarting)
{
Recover();
}
//Code d'initialisation ...
}
```

Bien que la méthode Recovery n'effectue dans notre application aucune action de sauvegarde, il est important de prêter attention au mécanisme de cette méthode en environnement réel.

```
static int Recovery(object o)
{
Timer t = new Timer();
t.Interval = 1000;
t.Tick+=delegate(object sender, EventArgs e)
{
bool cancelled;
```

```
ApplicationRecovery.ApplicationRecoveryInProgress(out cancelled);
if (cancelled)
{
Console.WriteLine("Sauvegarde de l'état annulé par l'utilisateur.");
}
};
t.Start();
/*****
Code de sauvegarde de l'état l'application (éventuellement long)
*****/
t.Stop();
ApplicationRecovery.ApplicationRecoveryFinished(true);
return 0;
```

Le processus de sauvegarde peut être assez long en fonction de l'application, et ce processus peut lui-même planter. C'est la raison pour laquelle la fonction RegisterApplicationRecoveryCallback prévoit un délai accordé à la méthode de sauvegarde pour effectuer ses instructions. Si le processus de sauvegarde dure plus longtemps que prévu, il est nécessaire d'indiquer à l'OS que nous avons besoin de plus de temps pour terminer la sauvegarde, et ce aussi souvent que nécessaire. C'est précisément le rôle de la fonction ApplicationRecoveryInProgress utilisée. Nous utilisons un Timer pour indiquer à l'OS, toutes les secondes, que le processus de sauvegarde est en cours et que nous avons besoin de plus de temps. Nous récupérerons également en sortie un booléen indiquant si l'utilisateur a décidé d'interrompre cette procédure afin de stopper définitivement l'application.

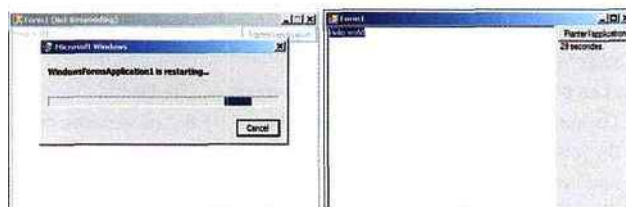
Une fois la sauvegarde terminée, nous devons l'indiquer au service ARR via la fonction ApplicationRecoveryFinished.

En exécutant notre application nous obtenons alors ce comportement Lancement de l'application :



1 - Plantage de l'application

2 - Nous demandons à Windows 2008 de redémarrer l'application



3 - Notre application est restaurée dans l'état dans lequel elle était.

4 - Notez que l'application doit obligatoirement s'exécuter pendant au moins 60 secondes pour que son redémarrage soit proposé.

■ Yann ALET - Développeur, [Neos-SDI](#)