

# Connexion distante en http ou https « server side » avec authentification (Framework .net 2.0 Béta 2)

10/06/2005

Par [Laurent Geffroy](#)

## **Droit de diffusion:**

L'ensemble ou partie de ce document ainsi que le code mis à disposition, ne peut être diffusé sur d'autres sites Web sans l'autorisation au préalable de son créateur.

## **Avant Propos :**

Ce sujet m'a pris pas mal de temps à résoudre, jusqu'au moment où j'ai découvert que le problème venait du serveur distant !

Néanmoins, je pense qu'il est intéressant de résumer la manière de générer une requête « post » coté serveur. Nous étudierons donc :

- L'utilisation de WebRequest et du WebResponse ;
- Le contexte de connexion avec le ServicePointManager et le NetworkCredential.

L'objectif est d'envoyer des données XML à un serveur Web distant en https, le tout coté serveur en méthode POST. Le serveur distant me renverra un flux XML afin de valider ou non ma transaction.

J'ai donc tout d'abord créé une application Windows pour tester ma transmission.

Ce tutoriel peut vous servir à monter des relations B2B, mais également pour la mise en place de solutions de paiement électronique sans que votre utilisateur n'ait à changer de site internet.

La démo est réalisée sur la version Beta 2 du Framework .net 2.0.

## **Sommaire :**

### **La méthode de connexion**

### **L'appel de la méthode**

### **Pour en savoir plus**

## La méthode de connexion

Ma classe HttpPost permet de gérer le transfert et l'ensemble de son contexte.

L'assembly System.Security.Cryptography s'est étoffée avec le framework 2.0.

N'oubliez pas d'ajouter l'assembly Security dans les références du projet. Par défaut, il prendra l'assembly 1.1 installée sur votre machine et vous ne comprendrez pas pourquoi X509Chain n'est pas reconnu !!

J'autorise ici l'acceptation de tous les certificats. Cette partie de code est nécessaire dans le cadre d'une connexion https à un serveur distant. On peut également déterminer le nombre de connexions à maintenir. Le MaxServicePointIdleTime permet de déterminer le temps durant lequel la connexion est maintenue. Au-delà, elle est supprimée par le garbage collector.

```
public PostHttp()
{
    // Détermine le nombre de connexion concurrentes
    ServicePointManager.DefaultConnectionLimit = 20;

    // En millisecondes, durée d'utilisation de l'instance
    ServicePointManager.MaxServicePointIdleTime = 100000;

    ServicePointManager.ServerCertificateValidationCallback = delegate(Object obj,
X509Certificate certificate, X509Chain chain, SslPolicyErrors errors) { return true; };
}
```

J'ai donc créé une méthode **LanceRequete** qui reçoit 7 arguments nécessaires à ma connexion distante. Elle me retourne une string que je peux ensuite étudier. Le mieux serait bien entendu de retourner un Stream afin de faciliter le parsing XML.

J'initialise ma connexion. Si un certificat X509 est requis pour l'authentification, on le transmet dans le WebRequest. p\_strCer est le chemin du fichier .cer.

```
public string LanceRequete(string p_strCer,
    string p_strLogin,
    string p_strPwd,
    string p_strData,
    string p_strURI,
    string p_strMethod,
    string p_host)
{
    // Valeur de retour
    string v_result = "ERROR:";

    // Récupère la connexion au serveur distant
    ServicePoint sp = ServicePointManager.FindServicePoint(new Uri(p_strURI));

    // Credential dans le cadre d'une connexion authentifiée
    NetworkCredential credentials;

    try
    {
        // Création de la requete
        HttpWebRequest req = (HttpWebRequest)WebRequest.Create(new Uri(p_strURI));

        // Gère le certificat X509 nécessaire à l'authentification de l'émetteur
        if (p_strCer != null)
        {
            // Ajout du certificat à la Requete HTTP(s) en donnant le chemin du fichier
            .cer

            req.ClientCertificates.Add(X509Certificate.CreateFromCertFile(p_strCer));
        }
    }
}
```

Afin de s'authentifier sur le site http(s), on envoie dans le contexte de connexion les informations. Je l'ajoute également dans le Header de la requête. deux précautions valent mieux qu'une !

```
// Dans le cadre d'une authentification
if ((p_strLogin != null) && (p_strPwd != null))
{
    // Création du NetworkCredential
    credentials = new NetworkCredential(p_strLogin, p_strPwd);

    // Nécessite une pré authentification
    req.PreAuthenticate = true;

    // Met en cache le contexte d'authentification
    CredentialCache cache = new CredentialCache();
    cache.Add(new Uri(p_strURI), "Basic", credentials);
    req.Credentials = cache;

    // En fonction des cas, il vaut mieux également envoyer les données
    // d'authentification
    // Dans le Header de la requete HTTP(s)
    Encoding asciiEncoding = Encoding.ASCII;
    byte[] v_ba = new byte[asciiEncoding.GetByteCount(p_strLogin + ":" +
    p_strPwd)];
    v_ba = asciiEncoding.GetBytes(p_strLogin + ":" + p_strPwd);

    // Ajoute au Header
    req.Headers.Add(HttpRequestHeader.Authorization, "Basic " +
    Convert.ToBase64String(v_ba));
}
```

On finit de paramétrer la Requête http(s) en indiquant la méthode, le referer, le timeout et le type de contenu.

```
// Si le serveur de destination redirige vers une autre adresse...
req.AllowAutoRedirect = true;

// Le host
req.Referer = p_host;
// La methode
req.Method = p_strMethod;
// Gèere le type de contenu envoyé
req.ContentType = "text/xml";
// Gère le timeout de la requete
req.Timeout = 100000;
req.KeepAlive = false;
```

Je mets maintenant la chaine de caractères dans la requête. Il s'agit de mon contenu XML.

```
// Dépose les données dans la requete HTTP(s)
if (p_strData != null)
{
    byte[] v_Bytes = null;
    v_Bytes = Encoding.UTF8.GetBytes(p_strData);
    // Spécifie la taille de la requete HTTP(s)
    req.ContentLength = v_Bytes.Length;

    // Dépose les données dans la requete
    Stream v_st = req.GetRequestStream();
    v_st.Write(v_Bytes, 0, v_Bytes.Length);
    v_st.Close();
}
```

Connexion distante en http ou https « server side » avec authentification (Framework .net 2.0 Béta 2)

```
else
{
    req.ContentLength = 0;
}
```

Je récupère la réponse et je le mets dans le v\_result.

```
// Gestion de la réponse
try
{
    WebResponse v_wr = req.GetResponse();
    Stream v_s = v_wr.GetResponseStream();

    // Gestion de l'encodage de la réponse
    Encoding v_encode = System.Text.Encoding.GetEncoding("utf-8");
    StreamReader v_sr = new StreamReader(v_s, v_encode);

    // Dépose les données dans la valeur de retour
    v_result = v_sr.ReadToEnd();

    v_sr.Close();
    v_wr.Close();
    v_s.Close();
}
```

Je gère les erreurs, récupère le message, l'URL de réponse et les Headers reçus.

```
catch (WebException wex)
{
    // Gestion des erreurs et envoi dans la valeur de retour
    v_result += "\r\nMessage : " + wex.Message + "\r\n";
    v_result += "ResponseURI : " + wex.Response.ResponseUri + "\r\n";
    if (wex.Response.Headers.Count > 0)
    {
        for (int i = 0; i < wex.Response.Headers.Count; i++)
        {
            v_result += "Header " + (i + 1) + " : " +
                wex.Response.Headers[i].ToString() + "\r\n";
        }
    }
}
```

On gère ici les erreurs du WebRequest.

```
}
catch (Exception v_ex)
{
    v_result += v_ex.Message;
}

return v_result;
}
```

## L'appel de la méthode

Je récupère le fichier, je le lis et le dépose dans la variable v\_strx.

Je finis par lancer ma méthode et miracle, mon serveur me retourne sa réponse que je peux exploiter.

```
if (File.Exists(txt_file.Text))
{
    // Lit le fichier
    StreamReader v_srx = new StreamReader(txt_file.Text);
    string v_strx = v_srx.ReadToEnd();

    PostHttp v_post = new PostHttp();

    // Envoi de la demande
    txt_response.Text = v_post.LanceRequete(null,
                                            "login",
                                            "password",
                                            v_strx,
                                            "https://monserveurdistant/mapage",
                                            "POST",
                                            "monserveurlocal");
}
else
{
    txt_response.Text = "Le fichier n'existe pas";
}
```

## Pour en savoir plus

A approfondir, l'utilisation du certificat X509.

La méthode `ServicePointManager.CertificatePolicy` est maintenant obsolète sur le Framework 2.0. Il faut maintenant utiliser `ServicePointManager.ServerCertificateValidationCallback`.

<http://blogs.msdn.com/adarshk/archive/category/7225.aspx>

Utilisation du `NetworkCredential`

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfssystemnetnetworkcredentialclasstopic.asp>

Gestion des headers

<http://forum.hardware.fr/hardwarefr/Programmation/Header-http-sujet-67998-1.htm>

l'auteur : <http://www.laurentgeffroy.com>

PS : Merci Fred et Elise pour le coup de pied au c..

Les produits mentionnés ne sont pas encore commercialisés. Ils sont en phase de test. Si vous souhaitez obtenir Visual Studio 2005 en version beta ou en version finale dès sa disponibilité, vous pouvez souscrire un abonnement MSDN

<http://www.microsoft.com/france/msdn/abonnements/presentation.asp>