

CONSTRUIRE UNE STRUCTURE XML « COMPLEXE » GRACE A LA SERIALISATION

Venant du monde de l'EDI à l'origine, j'ai souvent été surpris par les structures XML très linéaires. Entête, détails, adresses se suivent souvent sans respecter une arborescence logique. En fait, dans une commande, il ne peut y avoir de détail s'il n'y a pas d'entête.

J'ai donc entrepris de construire un XML WebService qui nécessite une structure complexe, ou dirons nous plutôt avancée :

- Un premier niveau, l'interchange, qui va donner les informations d'authentification ;
- Un second, qui est fils du premier, qui correspond à l'entête de ma commande ;
- Un troisième qui est fils de l'entête avec les adresses de livraison et de facturation
- Un quatrième, toujours fils de l'entête avec le détail de mes lignes de commande
- Un dernier, fils de l'interchange qui correspond au pied du document.

Nous obtenons donc une structure de ce type

Interchange	Obligatoire	1,1
Entete	Obligatoire	1,1
Adresse Liv	Obligatoire	1,1
Adresse Fact	Obligatoire	1,1
Detail	Obligatoire	1,n
Pied	Obligatoire	1,1

Nous allons donc voir dans ce tutoriel :

- La création des structures
- La sérialisation des celles-ci
- L'invocation de WebServices, parce que c'est toujours bon de le rappeler et cela ne fait pas de mal !
- La sortie du flux XML vers le navigateur.

N'hésitez pas à télécharger et à utiliser les sources, elles sont faites pour cela.

1. Les structures

1.1. Les informations communes

Si vous souhaitez procéder à une sérialisation « avancée » de vos structures, il est impératif d'ajouter l'assembly System.Xml.Serialization.

Votre classe doit également avoir comme attribut personnalisé [Serializable].

Enfin, si vous souhaitez que les membres de votre structure apparaissent comme attribut d'un élément XML ajoutez l'attribut [XmlAttribute()].

1.2. La structure de base, wsInterchange

```
using System;
using System.Xml.Serialization;

namespace SerialXml
{
    [Serializable]
    public class WsInterchange
    {
        public WsInterchange()
        {
        }
        private struct_entete m_entete;
        private struct_pied m_pied;

        private string m_login = "";
        private string m_pwd = "";
        private int m_test = 0;

        [XmlAttribute()]
        public string Login
        {
            get
            {
                return m_login;
            }
            set
            {
                m_login = value;
            }
        }
    }
}
```

C'est la structure principale

Elle est bien entendu [Serializable].

Notez qu'elle inclut l'entête et le pied. Le détail et les adresses sont inclus dans l'entête.

Dans l'exemple, Login, Pwd et Test sont des données qui apparaîtront dans l'attribut de mon élément WsInterchange.

1.3. La structure de l'entête

```

[Serializable]
public class struct_entete
{
    public struct_entete()
    {
    }

    private struct_adresse m_livraison = new struct_adresse();
    private struct_adresse m_facturation = new struct_adresse();

    private string m_orderid;
    private DateTime m_dt;

    private struct_CollectionDetails m_details;

    public struct_adresse Livraison
    {
        get {return m_livraison;} set {m_livraison = value;}
    }

    public struct_adresse Facturation
    {
        get {return m_facturation;} set {m_facturation = value;}
    }

    [XmlAttribute()]
    public string OrderID
    {
        get {return m_orderid;} set {m_orderid = value;}
    }

    public DateTime DateCommande
    {
        get {return m_dt;} set {m_dt = value;}
    }

    [XmlArray("Details")]
    [XmlArrayItem("Detail")]
    public struct_CollectionDetails Details
    {
        get
        {
            return m_details;
        }
        set
        {
            m_details = value;
        }
    }
}

```

Mon entête comporte donc la structure pour l'adresse de livraison et de facturation ainsi que 2 éléments pour le numéro et la date de commande.

Le numéro de commande est passé en XML en attribut

L'entête comporte aussi de détail. La structure de détails est incorporée dans l'élément « Details » et l'ensemble des valeurs (car c'est une collection) sera incorporé dans des éléments « Detail »

1.4. La collection de détails

```
using System;
using System.Collections;
using System.Xml.Serialization;
namespace SerialXml
{
    // Serialisable et doit ABSOLUMENT dérivé de CollectionBase
    [Serializable]
    public class struct_CollectionDetails : CollectionBase
    {
        public struct_CollectionDetails()
        {
        }

        // Permet l'ajout, dérivé de ICollection
        public void Add(struct_detail p_value )
        {
            List.Add(p_value);
        }

        // Permet de récupérer et alimenter le tableau de valeurs
        public struct_detail this[int p_index]
        {
            get
            {
                return (struct_detail)List[p_index];
            }
            set
            {
                List[p_index] = value;
            }
        }
    }
}
```

N'oubliez pas d'insérer l'assembly System.Collections car elle doit absolument dériver de CollectionBase

Ajoutez également la méthode Add, qui dérive de ICollection.

Construisez et récupérez la structure de cette collection

struct_detail comprend la référence, les quantités, le conditionnement et la description.

2. Le XML WebService

La rien de plus simple, un XML WebService commande.asmx est créé et attend la structure « avancée » wsInterchange. Dans un premier temps, il ne fait pas de traitement mais renvoie le numéro de commande. A la portée de tous.

```
[WebService(Namespace="http://www.dotnet-tech/webservices/")]
public class Commande : System.Web.Services.WebService
{
    public Commande()
    {
        //CODEGEN : Cet appel est requis par le Concepteur des services Web ASP.NET
        InitializeComponent();
    }

    Code généré par le Concepteur de composants

    [WebMethod(Description="Générateur de commande avec structure complexe")]
    public string GenererCommande(WsInterchange p_interchange)
    {
        // Récupération des informations pour traitement...
        //...
        //...
        return p_interchange.Entete.OrderID.ToString();
    }
}
```

3. L'invocation du XML WebService

La page AppelWS.aspx invoque le WebService lors du clic sur le bouton. Il renvoie son retour dans un label :

```
private void bt_ws_Click(object sender, System.EventArgs e)
{
    // Invoque le WebService
    wsCommande.Commande ws = new wsCommande.Commande();
    string v_retour = ws.GenererCommande(ConstructionCommande.ConstruitCommande());

    // Gere le retour simple
    lb.Text = "Commande " + v_retour + " enregistrée";
}
```

Le contenu de la commande est générée par une méthode externe, ConstruitCommande() :

```
public static wsCommande.WsInterchange ConstruitCommande()
{
    wsCommande.WsInterchange v_com = new SerialXml.wsCommande.WsInterchange();
    // Interchange
    v_com.Login = "MonLogin";
    v_com.Pwd = "PasBien";
    v_com.Test = 1;
    // Entete
    v_com.Entete = new wsCommande.struct_entete();
    v_com.Entete.DateCommande = DateTime.Now;
    v_com.Entete.OrderID = "00011453421";
    // Adresse de Livraison
    v_com.Entete.Livraison = new wsCommande.struct_adresse();
    v_com.Entete.Livraison.ADDRID = "010123654055";
    v_com.Entete.Livraison.Nom = "Dotnet-tech.com";
    v_com.Entete.Livraison.Adresse1 = "rue du code";
    v_com.Entete.Livraison.Cp = "75000";
    v_com.Entete.Livraison.Ville = "Paris";
    // Adresse de facturation
    v_com.Entete.Facturation = new wsCommande.struct_adresse();
    v_com.Entete.Facturation.ADDRID = "010123686455";
    v_com.Entete.Facturation.Nom = "Dotnet-tech.com";
    v_com.Entete.Facturation.Adresse2 = "Service Facturation";
    v_com.Entete.Facturation.Adresse2 = "rue du code";
    v_com.Entete.Facturation.Cp = "75000";
    v_com.Entete.Facturation.Ville = "Paris";
    // Le détail
    wsCommande.struct_detail[] v_detail = new wsCommande.struct_detail[2];
    v_detail[0] = new wsCommande.struct_detail();
    v_detail[0].Reference = "003949434";
    v_detail[0].Qte = 34;
    v_detail[0].Cond = 10;
    v_detail[0].Designation = "Comment commentez son code source";
    v_detail[1] = new wsCommande.struct_detail();
    v_detail[1].Reference = "003949414";
    v_detail[1].Qte = 3;
    v_detail[1].Cond = 100;
    v_detail[1].Designation = "ASP.net pour les neuneux";

    v_com.Entete.Details = v_detail;

    // Pied du document
    v_com.Pied = new wsCommande.struct_pied();
    v_com.Pied.Infos = "Merci de bien vouloir emballer le tout dans un gros paquet cadeau !";

    return v_com;
}
```

4. La sortie XML vers le flux http

La page AppelXml renvoie la sérialisation non plus vers un Webservice mais vers un flux http.

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Récupère la commande depuis son "constructeur"
    wsCommande.WsInterchange v_xml = ConstructionCommande.ConstruitCommande();

    // Objet de sérialisation
    XmlSerializer s = new XmlSerializer( typeof(wsCommande.WsInterchange));

    // Définit la sortie XML Vers la page
    XmlTextWriter writer = new XmlTextWriter(Page.Response.Output);

    // Sérialise la structure complexe en XML
    s.Serialize(writer, v_xml);
}
```

Toujours aussi déconcertant, 4 lignes de codes et ma structure se sérialise et est envoyée sur mon navigateur Internet.

5. En Savoir plus

Introduction à la sérialisation

<http://www.dotnet-tech.com/tutoriels/serialisation>

Serialisation

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

L'auteur : Laurent GEFROY

<http://www.laurentgeffroy.com>

lgeffroy-at-club-internet.fr