


# Exemple d'applications métier avec Silverlight 3 RTM et .NET RIA Services mise à jour de juillet

Partie 11 : Le monde du client uniquement

par Brad Adams Jérôme Lambert (Traduction) ([Espace perso](#)) ([Blog](#))

Date de publication : 08 janvier 2011

Dernière mise à jour :

Cet article est une traduction d'un des articles de la série " **Business Apps Example for Silverlight 3 RTM and .NET RIA Services July Update**" de Brad Adams.

I - Le monde du client uniquement..... 3

## I - Le monde du client uniquement

J'ai eu beaucoup de plaisir à ce jour avec cette série... J'espère que vous en avez également retiré quelque chose. Certains lecteurs m'ont demandé si les services RIA sont requis pour la validation qualité du client. Non, les .NET RIA Services ne sont pas obligatoires... Vous pouvez effectuer ces chouettes opérations clientes (validation, tri, filtrage, etc) avec toutes les données clientes ou des données que vous avez obtenu via n'importe quel autre mécanisme.

Le titre est donc un peu trompeur - nous n'allons pas utiliser un seul .NET RIA Services dans cette partie...

Au sein de l'équipe design pour ce travail, nous disions que nous voulions un ensemble de glaçons et non des icebergs... Des éléments de technologies qui fonctionnent vraiment bien ensembles et que l'on peut coordonner à volonté. Plutôt qu'un monolithique iceberg dont vous devez prendre tout ou rien.

La partie la plus difficile dans cette méthodologie est de récupérer vos données au sein de l'application Silverlight... Vous ne voudriez pas l'utiliser si vous aviez un autre moyen plus facile pour récupérer des données... Cette démo couvre cette problématique et l'aborde avec les données en mémoire.

Cette démo nécessite (tout est 100% gratuit et pour toujours) :

- 1 **VS2008 SP1**
- 2 **Silverlight 3 RTM**
- 3 **.NET RIA Services July '09 Preview** <--- N'est en réalité pas nécessaire pour cette démo ! Mais c'est néanmoins bien de le savoir ;-)

Téléchargez ensuite l'ensemble des fichiers des démos ([lien sur le site original](#) - [lien sur developpez.com](#)).

Pour cette démo, nous nous mettrons uniquement du côté client.. Nous n'allons donc récupérer aucune données du serveur. La seule chose qui provient du serveur est une page HTML contenant le XAP. Ceci peut être hébergé sur n'importe quel serveur web.

Au niveau du client, j'ai créé un répertoire DataAccess et défini ma classe SuperEmployee dedans.

```
public class SuperEmployee : INotifyPropertyChanged, IEditableObject
{
    private SuperEmployee cache;
    private int _employeeID;
    private string _gender;
    private Nullable<int> _issues;
    private Nullable<DateTime> _lastEdit;
    private string _name;
    private string _origin;
    private string _publishers;
    private string _sites;
    private static int empIDCt = 0;

    public SuperEmployee()
    {
        this._employeeID = empIDCt++;
    }
}
```

Notez que cette classe implémente un couple d'interfaces utiles pour la liaison... Nous verrons très vite comment elles sont implémentées. Dans le constructeur, j'incrémente automatiquement la clé employeeID.. Vous pouvez faire comme vous le souhaitez bien sûr.

Ensuite pour chaque propriété, nous implémentons le modèle suivant :

```
[DataMember()]
[Key()]
```

```
[ReadOnly(true)]
public int EmployeeID
{
    get
    {
        return this._employeeID;
    }
    set
    {
        if ((this._employeeID != value))
        {
            ValidationContext context = new ValidationContext(this, null, null);
            context.MemberName = "EmployeeID";
            Validator.ValidateProperty(value, context);
            this._employeeID = value;
            this.OnPropertyChanged("EmployeeID");
        }
    }
}
```

Notez tout l'amusement qui se passe dans le set de la propriété. A cet endroit, nous créons un ValidationContext et appelons ValidateProperty.. C'est cette chose qui démarre et va voir les attributs défini sur ce membre afin d'effectuer le travail de validation. Une exception est générée lorsque il y a un problème de validation au sein du client Silverlight. Notez ici que nous avons aussi écrit le code pour déclencher une notification lorsque il y a un changement de valeur de la propriété, encore pour faciliter le binding. Chacune des propriétés ressemble à cette propriété..

Ensuite, pour le support du DataForm, nous avons besoin d'implémenter IEditableObject qui permet le mécanisme d'annulation. Je choisis un modèle très simple pour ceci. Notez que j'ai simplement mis en cache la valeur lors du commencement de l'édition et je copie le contenu du cache si la méthode CancelEdit est appelée.

```
public void BeginEdit()
{
    this.cache = new SuperEmployee();
    this.cache.EmployeeID = this.EmployeeID;
    this.cache.Gender = this.Gender;
    this.cache.Issues = this.Issues;
    this.cache.LastEdit = this.LastEdit;
    this.cache.Name = this.Name;
    this.cache.Origin = this.Origin;
    this.cache.Publishers = this.Publishers;
    this.cache.Sites = this.Sites;
}

public void CancelEdit()
{
    this.EmployeeID = this.cache.EmployeeID;
    this.Gender = this.cache.Gender;
    this.Issues = this.cache.Issues;
    this.LastEdit = this.cache.LastEdit;
    this.Name = this.cache.Name;
    this.Origin = this.cache.Origin;
    this.Publishers = this.cache.Publishers;
    this.Publishers = this.cache.Publishers;
    this.Sites = this.cache.Sites;
    this.cache = null;
}

public void EndEdit()
{
    this.cache = null;
}
```

Ensuite, je déclenche simplement le mécanisme de changement de propriété.

```
public event PropertyChangedEventHandler PropertyChanged;
protected virtual void OnPropertyChanged(string propName)
{
}
```

```
if (this.PropertyChanged != null)
{
    this.PropertyChanged(this, new PropertyChangedEventArgs(propName));
}
}
```

Après ça, nous avons besoin d'un groupe entier de SuperEmployees... Pour cette démo, je les crée simplement à partir d'objets en mémoire. Vous aurez besoin de penser à comment vous allez récupérer vos données.. Services WCF ? Astoria ? Générique REST ? RIA Services évidemment ;- ) Peu importe, voici à quoi ça ressemble dans cette démo :

```
public class SuperEmployeeList
{
    List<SuperEmployee> list = new List<SuperEmployee>()
    {
        new SuperEmployee() {
            Gender="Male",
            Issues=982,
            Name = "Alfred",
            Origin="Human",
            Publishers="DC",
            Sites="first appears in Batman #16"},

        new SuperEmployee() {
            Gender="Male",
            Issues=518,
            Name = "Alfred E. Neuman",
            Origin="Human",
            Publishers="Ec",
            Sites="first appears in MAD #21"},
    }
}
```

J'ai ensuite simplement ajouté quelques méthodes pour m'aider à accéder aux données de la classe à partir l'interface graphique :

```
public string OrginFilter { get; set; }
public IEnumerable<SuperEmployee> GetEmployees()
{
    if (OrginFilter != null && OrginFilter != String.Empty)
    {
        return list.Where(emp=>emp.Origin.Contains(OrginFilter)).ToArray();
    }
    else return list.ToArray();
}
```

L'interface ressemble beaucoup à ce que nous avons vu auparavant mais j'ai mis la source de données dans le code pour la rendre un peu plus claire.

```
SuperEmployeeList Context = new SuperEmployeeList();

public Home()
{
    InitializeComponent();
    LoadData();
}

void LoadData()
{
    PagedCollectionView pcv = new PagedCollectionView(Context.GetEmployees());

    dataGrid1.ItemsSource = pcv;
    pager1.Source = pcv;

    originFilterBox.ItemsSource = Context.GetOrigins();
}
```

}

Ce sont vraiment les grandes lignes.. Voici ce que vous obtenez au final.. Exactement ce que nous avons vu auparavant ! Validation fonctionne, tri, filtrage, etc. Tout côté client.

