Silverlight 4 + RIA Services - Prêt pour les affaires

Mettre à jour les données dans le client

par Brad Abrams (Blog) Deepin Prayag (Traduction) (Home)

Date de publication : 14/02/2012

Dernière mise à jour : 20/02/2012

Cet article fait partie d'une série de traductions d'articles de Brad Abrams sur le développement d'applications métier avec Silverlight 4 et .NET RIA Services.

Cette série se concentre uniquement sur la base des applications métier : l'interrogation, la mise à jour, la validation et la sécurisation de vos données métier importantes.

Elle sera également utilisée pour mettre à jour certains billets de la **série Silverlight 3**.

Traduction

Cet article est la traduction la plus fidèle possible de l'article original de **Brad Abrams**, **Silverlight 4 + RIA Services - Ready for Business: Updating Data in the Client**

Prérequis

La procédure pas à pas requiert :

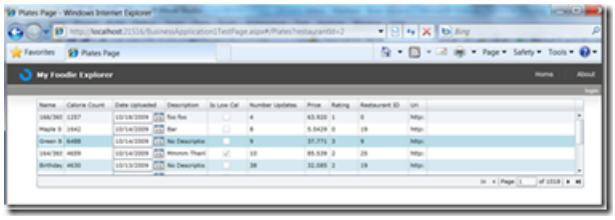
- **Visual Studio 2010** (ou la **version express gratuite**)
- **Silverlight 4 Tools** (inclut **RIA Services**)

Vous pouvez **télécharger l'application complète**.

J'ai implémenté cela avec Silverlight 4 RC, mais je m'attends à ce que cela fonctionne avec Silverlight 4 RTM.

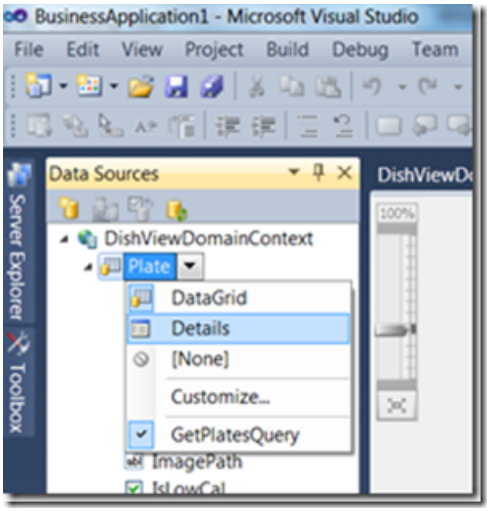
Mettre à jour les données dans le Client

Pour **continuer notre série**, jetons un coup d'oeil à la mise à jour de données. J'ai créé une page Plates.xaml avec une structure très similaire à **celle du dessus**. Pour les détails sur la création de cette page, jetez un oeil à **ma procédure pas à pas de démonstration lors du Professional Developers Conference de 2009 (PDC '09)**.

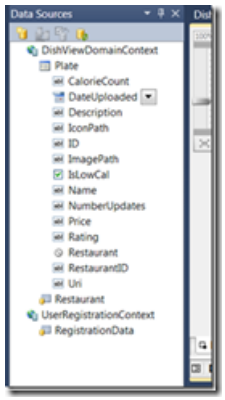


Maintenant, regardons la mise à jour des données *Plate.*

D'abord nous allons créer une interface utilisateur « formulaire » par défaut en glissant une entité de la fenêtre DataSources, en grande partie de la même façon que nous l'avions fait précédemment.



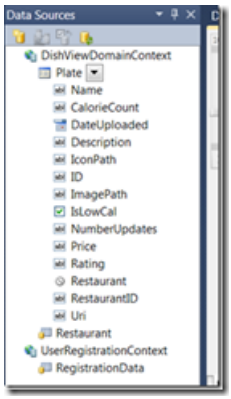
Mais avant de créer l'interface utilisateur, remarquez l'ordre des champs - il correspond à l'ordre dans lequel ils vont être générés dans l'interface utilisateur. Ils sont dans l'ordre alphabétique, mais ce n'est pas toujours ce que vous voulez. Par exemple, je pense que *Name* devrait arriver en premier.



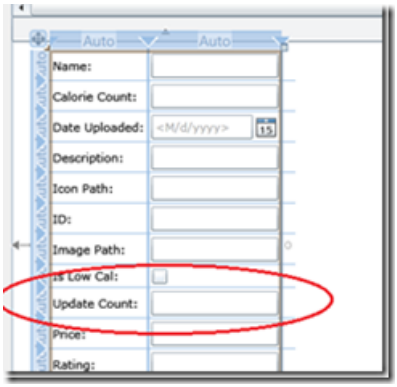
Pour y remédier, dans le projet serveur, ouvrez le fichier DishViewService.metadata.cs et ajoutez l'attribut *Display* au champ *Name* de la classe *Plate*. Pendant que nous y sommes, « Number Updates » n'est pas une si bonne appellation à utiliser dans l'interface utilisateur, changeons donc cela pour quelque chose de plus lisible.

```
[Display(Order = 0)]
public string Name { get; set; }
[Display(Name="Update Count")]
public Nullable<int> NumberUpdates { get; set; }
```

Maintenant, en revenant au projet client, nous voyons *Name* au-dessus



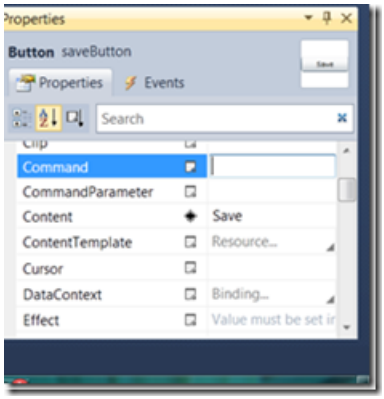
et quand nous le déposons sur le formulaire, nous voyons *Update Count*.



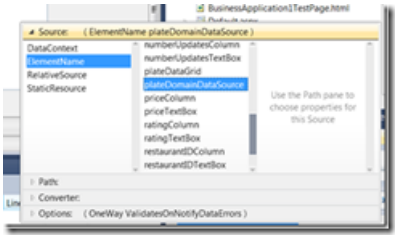
Maintenant nous avons un formulaire, et il est déjà relié à la même source de données que la grille. Donc, ajoutons un bouton pour enregistrer les modifications.

Donc, ajoutons un bouton « Save » au formulaire

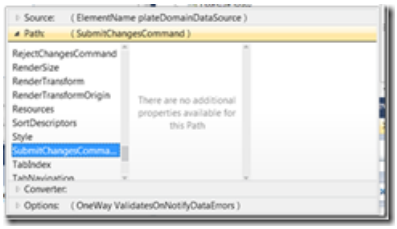
Et relions sa propriété *Command* (celle-ci est appelée quand le bouton est pressé).



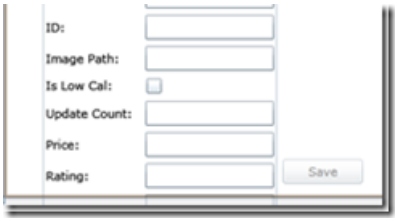
Maintenant nous devons relier cela à la commande *Submit* sur le DomainDataSource. Donc nous utilisons la liaison élément-à-élément. Sélectionnez le *plateDomainDataSource*



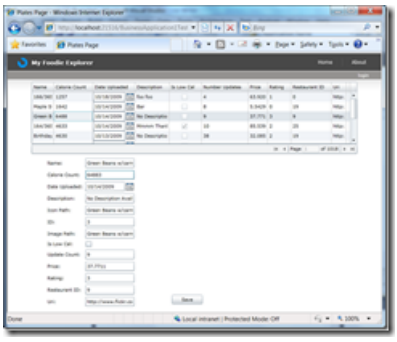
Ensuite sélectionnez le chemin au *SubmitChangesCommand*



Vous savez que vous avez le droit d'enregistrer les modifications , si le bouton est désactivé (même sur la surface de conception... après tout, il n'y a aucun changement à soumettre, n'est-ce pas ?)

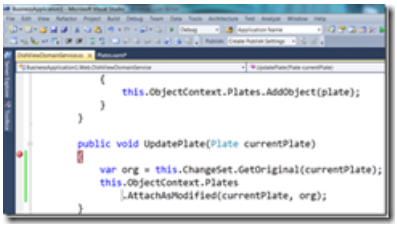


Maintenant, exécutez-le et voyez les résultats



Remarquez que quand vous changez de sélection dans la grille, l'interface utilisateur contenant les détails est automatiquement mise à jour, pas de code nécessaire. Aussi, remarquez que le bouton « Save » est désactivé jusqu'à ce qu'il y ait des modifications à soumettre au serveur. Essayez d'ajouter un bouton « Cancel » afin d'enlever les changements locaux. Il est très facile de relier au RejectChangesCommand de la même manière.

Maintenant voyons comment réellement mettre à jour les données dans la base. Définissez un point d'arret dans la méthode UpdatePlate

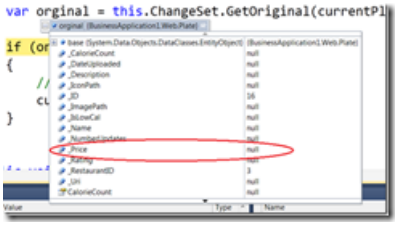


Remarquez que cette méthode de mise à jour est appelée deux fois ; une fois pour chaque entité modifiée sur le client.

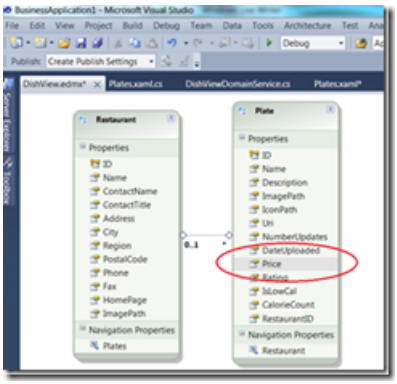
Maintenant, ajoutons un peu plus de logique à la mise à jour afin de gérer un peu de logique métier.

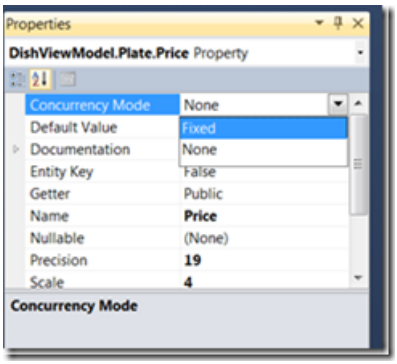
```
 1:          public void UpdatePlate(Plate currentPlate)
 2:          {
 3:              currentPlate.NumberUpdates++;
 4:
 5:              var orginal = this.ChangeSet.GetOriginal(currentPlate);
 6:
 7:              if (orginal.Price != currentPlate.Price)
 8:              {
 9:                  // add 1 dollar fee for changing price
10:                  currentPlate.Price += 1;
11:              }
12:          }
```

Premièrement, à la ligne 3, nous mettons à jour le compteur *NumberUpdates*, ensuite si le prix de cet élément de menu change, nous ajoutons une redevance d'un dollar. Remarquez que nous pouvons facilement vérifier si cette valeur est modifiée du point de vue de ce client en utilisant sa valeur d'origine... Le client envoie au serveur la valeur d'origine, qu'il avait obtenu à partir du serveur quand il a été chargé la première fois, pour n'importe quel champ qui a changé, est une clef, ou est marqué comme un jeton d'accès concurrentiel. (Ceci est une petite modification par rapport à la version précédente où *toutes* les valeurs d'origine étaient renvoyées au serveur. Par exemple, sur une mise à jour qui ne change que le décompte de *Calorie*, en inspectant la valeur d'origine, nous voyons :

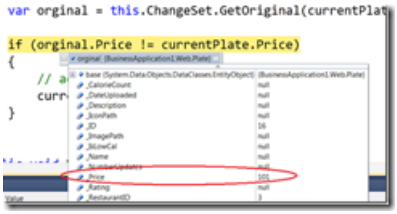


Si vous vouliez inclure *Price* sur n'importe quelle mise à jour, vous devriez le marquer comme un jeton d'accès concurrentiel. Pour marquer un champ comme un jeton de concurrence dans Entity Framework, sélectionnez le champ dans le concepteur et changez le *Concurrency Mode* à « fixed »





Avec ce changement, la valeur d'origine pour le prix sera toujours envoyée au serveur. Ca ne coûte qu'un tout petit peu plus cher pour la plupart des champs, donc je recommande de le faire chaque fois que la logique métier l'exige.



Conclusion

Ceci conclut la quatrième partie de cette série. Dans la cinquième partie nous verrons comment valider les données.

Liens

- **Visual Studio 2010**
- **Visual Studio 2010 Express**

- **Silverlight 4 Tools**
- **RIA Services**
- **Télécharger l'application complète**

Remerciements

Je tiens ici à remercier **Brad Abrams** pour son aimable autorisation de traduire l'article.
Je remercie **tomlev** pour sa relecture technique et ses propositions.
Je remercie également **xyz** pour sa relecture orthographique et ses propositions.