Silverlight 4 + RIA Services - Prêt pour les affaires

Valider les données

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: Validating Data**

Prérequis

La procédure pas à pas requiert :

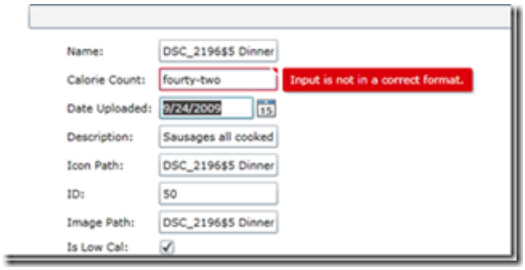
- **Visual Studio 2010** (ou la **version express gratuite**)
- **Silverlight 4 Tools** (inclut **RIA Services**)

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.

Valider les données

Pour **continuer notre série**, regardons la validation de données de nos applications métier. C'est bien de mettre à jour les données, mais quand vous activez la mise à jour de données vous devez souvent vérifier les données pour vous assurer qu'elles soient valides. RIA Services propose une façon propre et clairement définie de gérer cela. D'abord, voyons ce que vous obtenez gratuitement. La valeur pour n'importe quel champ entré doit être valide pour le domaine de ce type de données. Par exemple, vous n'avez jamais besoin d'écrire du code pour vous assurer que quelqu'un n'a pas tapé « quarante-deux » dans une zone de texte liée à un champ de type *int*. Vous bénéficiez également d'une belle présentation de la validation, qui se comporte bien, dans l'interface utilisateur.



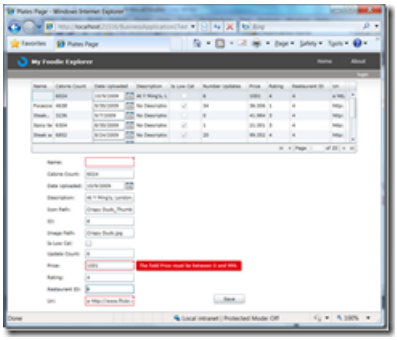
Note : Si vous ne voyez pas cela, assurez-vous que « *ValidatesOnExceptions=True* » est dans l'expression de liaison pour chaque champ.

Bien sûr, ce type de validation a une limite, dans une application réelle, vous avez besoin de validation plus étendue. Et pour cette validation vous devez absolument vérifier la validation avant que votre logique métier ne soit lancée car vous ne savez pas quel client pourrait être en train de vous transmettre la mise à jour, et en plus, vous voulez vérifier la validation sur le client afin d'offrir à l'utilisateur une expérience utilisateur très agréable et réduire le nombre de cas d'erreur de visite sur votre serveur, ce qui réduit la charge du serveur. Dans les modèles traditionnels, vous devez effectuer le contrôle des erreurs deux fois pour couvrir ces deux cas. Mais cela est de toute évidence sujet à l'erreur et peut facilement se désynchroniser. Donc, RIA Services offre un modèle commun pour la validation.

Les cas les plus communs sont couverts par un ensemble d'attributs personnalisés que vous appliquez à votre modèle sur le serveur. Ces attributs sont communs à l'ensemble du framework .NET pris en charge par ASP.NET Dynamic Data, ASP.NET MVC et RIA Services. Vous pouvez trouver l'ensemble complet dans System.ComponentModel.DataAnnotations. Mais pour vous donner un avant-gout :

```
[Display(Order = 0)]
[Required(ErrorMessage = "Please provide a name")]
public string Name { get; set; }
```

```
[Range(0, 999)]
public Nullable<decimal> Price { get; set; }
[RegularExpression(@"^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$",
                   ErrorMessage = "Please use standard Url format")]
public string Uri { get; set; }
```



Comme vous pouvez le voir ci-dessus, les validations sur le client sont gérées automatiquement, mais elles sont également exécutées à nouveau sur le serveur AVANT que votre méthode Update ne soit appelée. Cela vous permet de garder le cambouis de la validation hors de votre logique métier pour la mise à jour. L'autre atout est que ces validations s'appliquent exactement de la même façon peu importe où cette entité est utilisée dans l'interface utilisateur car elles sont intégrées dans le modèle. Vous pouvez bien sûr localiser les messages d'erreurs en passant un identifiant de ressource plutôt qu'une chaîne codée en dur. Vous pouvez également lire ces métadonnées de validation à partir d'un fichier de configuration externe ou d'une base de données plutôt que d'utiliser des attributs dans le code.

Mais il est clair que cela ne couvre pas tous les cas. Souvent vous devez écrire un peu de code de procédure réelle. Prenons un exemple de validation de la description pour s'assurer qu'elle est vraiment complète. Il n'y a pas d'attribut personnalisé pour ça ;-). Donc, écrivons un peu de code C#. Nous devons d'abord indiquer que le champ *Description* a une certaine validation personnalisée.

```
[CustomValidation(typeof(PlateValidationRules),
                  "IsDescriptionValid")]
public string Description { get; set; }
```

Ensuite vous pouvez facilement créer cette classe et implémenter la méthode *IsDescriptionValid*.

```
1:          public static ValidationResult IsDescriptionValid(string description)
2:          {
3:              if (description != null && description.Split().Length < 5)
4:              {
5:                  var vr = new ValidationResult("Valid descriptions must have 5 or more words.",
6:                      new string[] { "Description" });
7:                  return vr;
8:              }
9:
10:              return ValidationResult.Success;
11:          }
12:
```

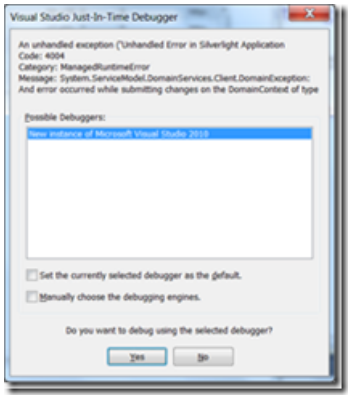
Remarquez qu'à la ligne 1, la signature de la méthode doit renvoyer un *ValidationResult* - c'est une classe de *DataAnnotations* qui contient des informations sur les erreurs de validation. La méthode doit aussi prendre un paramètre qui est du même type que le champ auquel cette méthode est en train d'être appliquée. Vous pourriez le faire au niveau entité pour effectuer une validation de champ croisée.

Ensuite, à la ligne 3, j'implémente un algorithme bidon pour déterminer si la description est valide ou non.

Aux lignes 5 et 6, je renvoie une erreur et j'indique le champ auquel cela s'applique.

Maintenant, exécutez l'application. Vous verrez que nous pouvons éditer la description et appuyer sur la touche de tabulation sans erreur, mais si nous soumettons, nous obtenons à ce moment-là une erreur exactement de la même manière que nous avons vue auparavant. Remarquez que j'aurais pu envoyer plusieurs entités et chacune d'entre elle pourrait comporter des erreurs. RIA Services s'adapte par rapport à chacune d'entre elles (nous vous donnons même une liste) et pendant que l'utilisateur édite, nous affichons une interface utilisateur comme celle-ci.

Note : Si à la place vous voyez ce genre de dialogue :

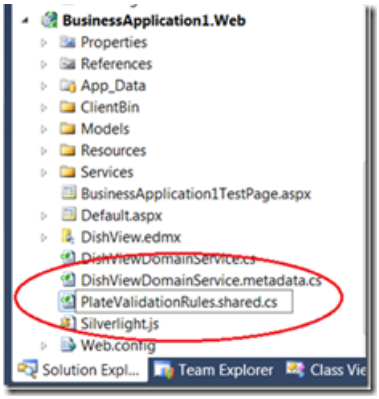


Cela signifie probablement que vous devez écrire un gestionnaire pour *SubmitChanges* sur votre DomainDataSource.

```
<riaControls:DomainDataSource  SubmittedChanges="plateDomainDataSource_SubmittedChanges"

        private void plateDomainDataSource_SubmittedChanges(object sender, SubmittedChangesEventArgs e)
        {
            if (e.HasError &&
                e.EntitiesInError.All(t => t.HasValidationErrors))
            {
                e.MarkErrorAsHandled();
            }
            else
            {
                System.Windows.MessageBox.Show(e.Error.ToString(), "Load Error",
 System.Windows.MessageBoxButton.OK);
                e.MarkErrorAsHandled();
            }
        }
```

Ça c'est très cool parce que nous avons toute la puissance de .NET pour écrire nos règles de validation. Mais l'inconvénient est que je ne reçois la validation qu'une fois le changement est soumis. Ce que je voudrais vraiment faire dans certains cas, c'est d'écrire une certaine logique de validation personnalisée et de l'avoir s'exécuter sur le serveur ET le client. Heureusement, nous avons la puissance de. NET sur le client et le serveur afin que nous puissions utiliser la validation du code partagé. Pour activer cela il suffit de changer le nom du fichier contenant la règle de validation pour inclure le suffixe « .shared.cs ». Ce changement entraîne la logique RIA Services MSBuild à compiler le fichier pour le client et le serveur.



Maintenant, exactement le même code sera exécuté sur le client et le serveur. Donc, s'il y a une mauvaise description, nous n'avons plus besoin de faire un aller-retour au serveur pour y remédier.

Bien sûr, dans un cas réel vous êtes susceptible d'avoir des règles de validation qui ne s'exécutent que sur le serveur, et d'autres qui sont partagées ; RIA Services prend aussi en charge ce scenario. Définissez simplement les règles de validation partagée dans les fichiers .shared.cs et toute règle de validation serveur-uniquement dans un autre fichier.

Conclusion

Ceci conclut la cinquième partie de cette série. Dans la sixième partie nous aborderons l'authentification et la personnalisation.

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.