Silverlight 4 + RIA Services - Prêt pour les affaires

Exposer les données d'Entity Framework

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....3
Prérequis....3
Exposer les données d'Entity Framework....3
Conclusion....5
Liens....5
Remerciements....5

Traduction

Cet article est la traduction la plus fidèle possible de l'article original de **Brad Abrams**, **Silverlight 4 + RIA Services - Ready for Business: Exposing Data from Entity Framework**

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.

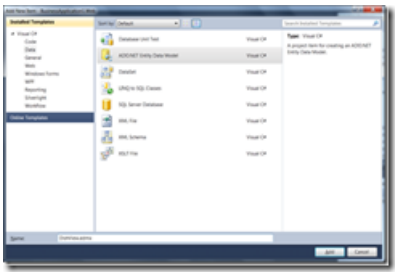
Exposer les données d'Entity Framework

Pour **continuer notre série**, j'ai voulu ensuite regarder la façon d'exposer vos données du côté serveur de votre application.

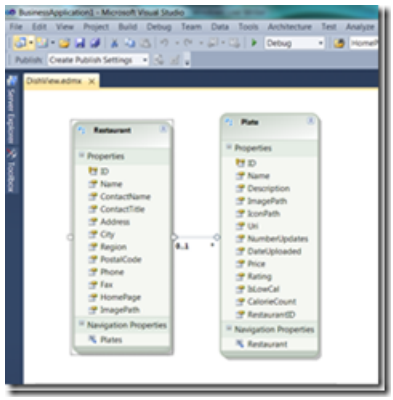
Les données intéressantes dans vos applications métier proviennent d'une grande variété de sources de données. D'une base de données SQL, d'Oracle DB, de SQL Azure, de SharePoint, d'un mainframe et vous avez probablement déjà choisi un modèle de données tel que NHibernate, Linq2Sql, Entity Framework, une procédure stockée, un service. Le but de RIA Services dans ce release est de faciliter le travail avec les données d'une ou de plusieurs sources, d'une manière transparente, à partir d'une application Silverlight. Cette procédure pas à pas utilisera Entity Framework pour accéder à la base de données SQL Express, mais le concept de base s'applique à n'importe quelle source de données.

Ajoutez DishView.mdf à BusinessApplication1.Web\App_Data - Bien entendu, dans un exemple réel vous devriez juste avoir une chaîne de connexion à une base de données existante.

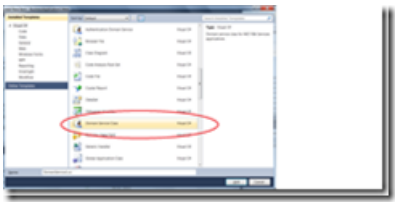
Ensuite, créez un modèle Entity Framework au-dessus de celui-ci.



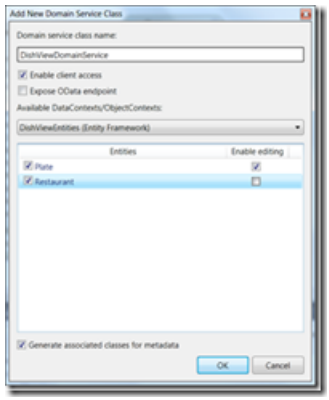
Comme vous pouvez le voir, nous avons un jeu d'entités très simple. Chaque Restaurant peut avoir n'importe quel nombre de Plats.



Ensuite, nous avons besoin d'écrire une certaine logique métier qui contrôle et façonne les données telles qu'elles sont perçues par le client Silverlight. Pour ce faire, nous ajoutons un nouveau DomainService. Un DomainService est simplement un type particulier de service WCF qui rend BEAUCOUP PLUS facile d'interroger, de mettre à jour, de sécuriser et de valider vos données. Mais si vous êtes un expert WCF et connaissez tous les détails pour la configuration d'un service WCF, vous pouvez certainement personnaliser ce service, d'une façon à ce qu'il corresponde exactement à vos besoins. Bien sûr, dans 90% des cas, nous espérons que vous n'aurez pas besoin d'y avoir recours.



Ensuite, il nous est donné la possibilité de pré-remplir le DomainService.



Dans ce cas, nous allons exposer Plat et Restaurant, mais seul Plat pourra être mis à jour. Nous allons également générer une classe de métadonnées pour accrocher les attributs de validation afin que vous puissiez régénérer le modèle EF sans perdre aucune personnalisation. Nous obtenons une classe de démarrage DishViewDomainService. Je l'ai mise à jour aux lignes 8-9 avec un peu de logique métier. Examinons chaque ligne pas à pas...

```
 1:     [EnableClientAccess]
 2:     public class DishViewDomainService : LinqToEntitiesDomainService<DishViewEntities>
 3:     {
 4:
 5:         public IQueryable<Restaurant> GetRestaurants()
 6:         {
 7:             return this.ObjectContext.Restaurants
 8:                 .Where (r=>r.Region != "NC")
 9:                 .OrderBy(r=>r.ID);
10:         }
```

```
11:
```

Ligne 1 : Cet attribut marque le DomainService comme étant accessible via http. Sans lui le service ne peut être appelé que du processus. Ceci est utile dans le scenario ASP.NET Webforms / Dynamic Data.

Ligne 2 : Nous dérivons cette classe de LinqToEntitiesDomainService. C'est une classe utilitaire qui fournit quelques helpers pour travailler avec EF. Le vrai travail est effectué par la classe de base DomainService, celle-ci étant la classe à partir de laquelle vous dérivez pour POCO et d'autres scenarios personnalisés.

Ligne 5 : Remarquez que nous renvoyons un IQueryable de notre type DAL Restaurant. IQueryable est l'interface sur laquelle est construite LINQ. Cela permet des fonctions d'interrogation depuis le client de sorte que vous puissiez faire des choses comme le tri, la pagination, le filtrage depuis le client et que ça se compose avec votre logique métier personnalisée et que ça s'exécute sur la couche de données. Cela signifie qu'aucune donnée supplémentaire n'est aspirée dans la couche intermédiaire et le client, mais vous n'avez pas à « crasser » votre logique métier pour faire face à cela.

Lignes 8-9 : Nous avons une logique métier qui élimine tout Restaurant en Caroline du Nord et met les résultats dans un ordre prédéfini. Vous pouvez bien sur imaginer une logique métier plus intéressante.

Conclusion

Ceci conclut la deuxième partie de cette série. Dans la prochaine partie nous verrons comment consommer les données dans le client Silverlight.

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.