

Introduction à Subsonic

par Philippe Vialatte ([ma page DVP](#)) ([Blog](#))

Date de publication : 06 Avril 2009

Dernière mise à jour :

Dans cet article, on va présenter le Framework de génération de code d'accès aux données Subsonic. On va voir comment on l'installe, les principes de bases, et comment l'utiliser.

I - Introduction.....	3
II - Installation et Configuration.....	3
II-A - Installation.....	3
II-B - Configuration.....	5
II-B.1 - Configuration à la main.....	5
II-B.2 - Configuration automatique avec SubStage.....	6
II-B.3 - Génération du code.....	8
III - Utilisation de la couche d'accès aux données.....	10
III-A - Structure générale des classes.....	10
III-B - Opérations CRUD.....	11
III-B-1 - Récupération d'objets depuis la base de données.....	11
Récupération d'un seul objet.....	11
Récupération d'un ensemble d'objets.....	12
III-B-2 - Ajouter ou mettre à jour un objet dans la base de données.....	13
III-B-3 - Suppression d'un objet de la base de données.....	14
III-C - Requêtes.....	15
III-C-1 - Requêtes de sélection.....	15
III-C-2 - Requêtes d'insertion, suppression, mises à jour.....	16
III-C-3 - Requêtes d'agrégation.....	17
III-C-4 - Transactions.....	17
IV - Conclusion.....	18
V - Remerciements.....	18

I - Introduction


Un problème récurrent dans les développements d'applications de gestion est le moyen que l'on va utiliser pour s'interfacer avec la base de données, qui est, dans ces applications, assez souvent le coeur du système.

Dans cet article, on va voir comment utiliser Subsonic pour générer notre couche d'accès aux données, simplement, et de façon efficace.

Subsonic est un générateur de DAL (Data Access Layer, couche d'accès aux données).

C'est un projet open source, développé depuis août 2006 par une équipe dirigée par Rob Conery, qui a récemment été embauché par Microsoft, pour continuer son travail sur Subsonic...

Subsonic est basé sur le patron de conception "Active Record".

 *En génie logiciel, le patron de conception (design pattern) active record est une approche pour lire les données d'une base de données. Les attributs d'une table ou d'une vue sont encapsulés dans une classe. Ainsi l'objet, instance de la classe, est lié à un tuple de la base. Après l'instanciation d'un objet, un nouveau tuple est ajouté à la base au moment de l'enregistrement. Chaque objet récupère ses données depuis la base; quand un objet est mis à jour, le tuple auquel il est lié l'est aussi. La classe implémente des accesseurs pour chaque attribut.*

(Source : Wikipédia)

En termes simplifiés, une classe **Client** utilisant Active Record permettra de faire:

```
Client objClient = new Client();  
objClient.Nom="Vialatte";  
objClient.Prenom="Philippe";  
objClient.Save();
```

Subsonic permet en peu de temps d'obtenir une couche d'accès aux données

- Performante
- Indépendante de la base de données
- Facile à utiliser
- Facile à synchroniser

II - Installation et Configuration

II-A - Installation

L'installation de Subsonic se fait très simplement.

En effet, il suffit de se rendre sur le site <http://www.subsonicproject.com/>, et de récupérer la dernière version stable (en ce moment, la version 2.1).



Au moment où j'écris l'article, on peut trouver la version 2.1 finale au lien suivant : http://subsonicproject.googlecode.com/files/SubSonic_2.1_Final_Source.zip

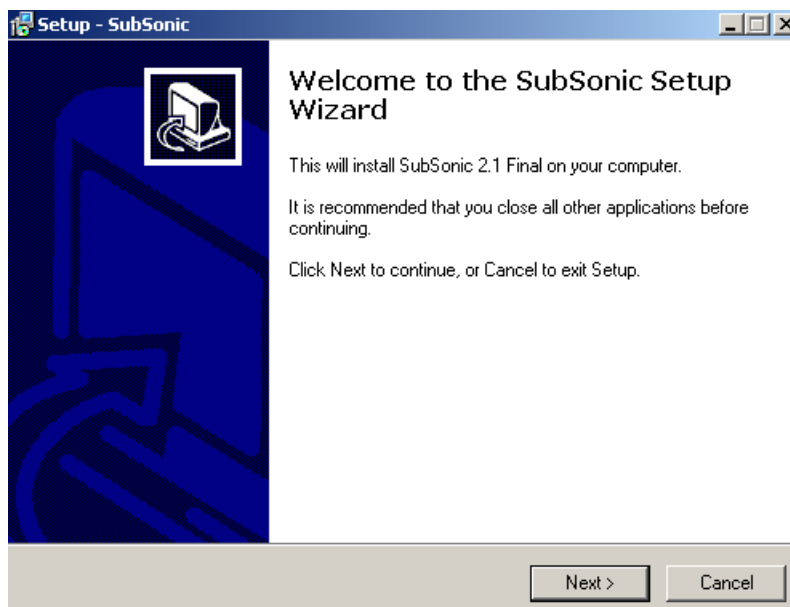
Par contre, pour être sûr d'avoir la dernière version, il vaut mieux passer par le site web.

! *Jusqu'à très récemment, les versions majeures de Subsonic étaient hébergées sur Codeplex.*

Depuis le 28 Mars, cependant, le site Codeplex de Subsonic a été fermé au public, et le code est désormais disponible sur Google code, à cet url :

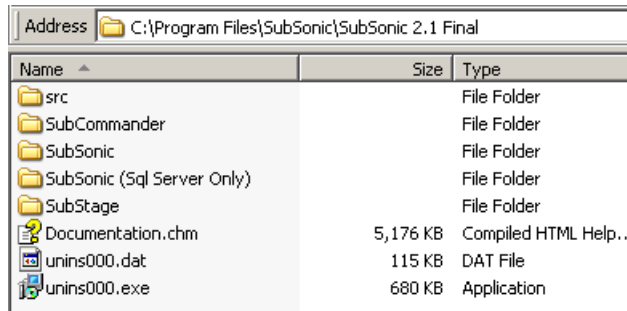
<http://subsonicproject.googlecode.com/svn/trunk/>

L'installation en elle-même est simplissime, il suffit de lancer l'exécutable.



Puis de cliquer "Next", "Next", "Next" et "Finish"

Une fois l'installation terminée, on obtient le répertoire suivant:



II-B - Configuration

On peut configurer Subsonic de deux façons différentes:

- à la main
- en utilisant SubStage (fourni dans le package d'installation)

Comme on est courageux, on va commencer par faire la configuration à la main.

La configuration sera la même, que ce soit pour un site web ou une application Console/Winform.

II-B.1 - Configuration à la main

Avant tout, on va ajouter, dans le fichier de configuration (App.Config ou Web.Config), la section de configuration pour Subsonic.

```

<configuration>
...
  <configSections>
    ...
    <section name="SubSonicService" type="SubSonic.SubSonicSection, SubSonic"
      allowDefinition="MachineToApplication" restartOnExternalChanges="true" requirePermission="false"/>
    ...
  </configSections>

```

Ceci est la configuration standard pour la section de configuration de Subsonic. Pour plus d'informations sur **requirePermission**, **restartOnExternalChanges** et **allowDefinition**, je vous renvoie à la MSDN:

Propriétés SectionInformation

On va ensuite ajouter une chaîne de connexion dans notre fichier de configuration. On va utiliser une chaîne de connexion tout ce qu'il y'a de plus standard en .net. . Les bases de données que l'on peut utiliser par défaut dans Subsonic sont les suivantes:

- SQL Server
- MySql
- Oracle
- Access (avec un provider spécifique)
- SQLite
- SQL CE

Dans l'exemple actuel, on va prendre une chaîne de connexion sur une base de données SQL Server.

```

<connectionStrings>
  <add name="AdventureWorks"
    connectionString="Data Source=(local)\sqlexpress;Initial Catalog=AdventureWorks;Integrated Security=True"/>

```

```
</connectionStrings>
```

On va ensuite créer la section `SubsonicService`, dans laquelle on va déterminer le fournisseur de données par défaut que l'on va utiliser.

```
<SubsonicService defaultProvider="AdventureWorks">
  <providers>
    <clear/>
    <add name="AdventureWorks" type="SubSonic.SqlDataProvider, SubSonic"
      connectionStringName="AdventureWorks" />
  </providers>
</SubsonicService>
```

Cette configuration signifie que le fournisseur de données par défaut de Subsonic sera le fournisseur Adventureworks, qui utilise (comme c'est original) la chaîne de connexion AdventureWorks, et que c'est un fournisseur de données SQL Server.

Par défaut, l'espace de nom dans lequel seront générées les classes de Subsonic sera `Subsonic`. On peut, au niveau de cette partie du fichier de configuration, modifier l'espace de nom, grâce à la propriété **generatedNamespace**. Ici, on va changer l'espace de nom en `AdventureWorks.DB`:

```
<SubsonicService defaultProvider="AdventureWorks">
  <providers>
    <clear/>


    <add name="AdventureWorks" type="SubSonic.SqlDataProvider, SubSonic" generatedNamespace="AdventureWorks.DB"
      connectionStringName="AdventureWorks" />
  </providers>
</SubsonicService>
```

Et...c'est tout pour un projet winform/console.

Pour un projet web, on peut, en plus, ajouter une étape supplémentaire, à savoir l'ajout d'un **buildprovider**, qui va, au moment de la compilation du site web, générer automatiquement la couche de mapping depuis la base de données. Pour cela, il suffit d'ajouter, d'une part, une section `buildProviders` dans la section **compilation** de la section **system.web** du `Web.Config`.

```
<compilation debug="true" >
  <buildProviders>
    <add extension=".abp" type="SubSonic.BuildProvider, SubSonic"/>
  </buildProviders>
</compilation>
```

Il suffit ensuite d'ajouter, dans le répertoire `App_Code` du site web, un fichier avec une extension `.abp` (le nom du fichier ou son contenu n'importent pas), et à la première configuration, tout sera généré.

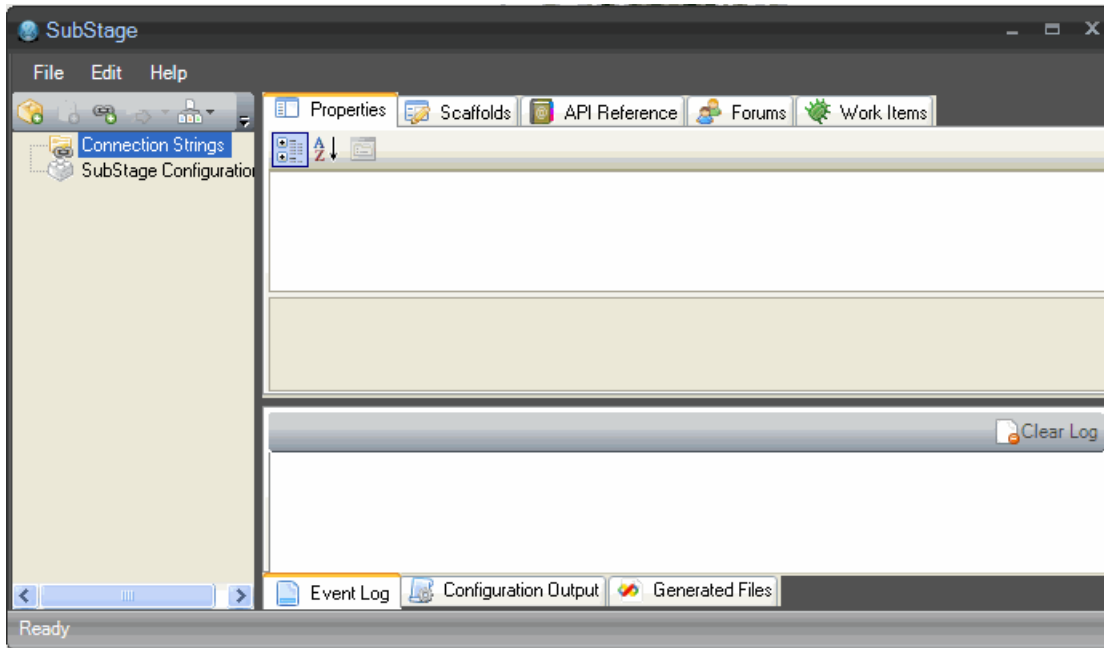
 **Attention, le `buildprovider` ne fonctionne que si le site web est déployé dans un environnement qui fonctionne en **Full Trust**, ce qui rend, dans la pratique, cette méthode inutilisable avec la plupart des hébergeurs.**

Une fois ces étapes passées, on n'a plus qu'à commencer à travailler.

II-B.2 - Configuration automatique avec SubStage

Savoir tout configurer à la main peut être nécessaire, mais vu que des outils existent pour faire le travail à notre place, autant les utiliser.

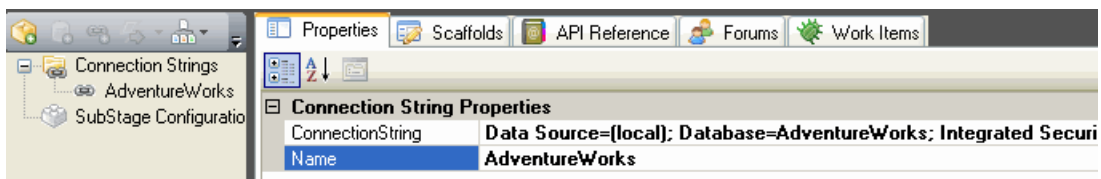
SubStage est une interface graphique permettant de configurer Subsonic de façon visuelle, ainsi que d'éditer les données, d'accéder à la documentation de l'API, au forum dédié à Subsonic, et aux Work Items en cours sur le projet.



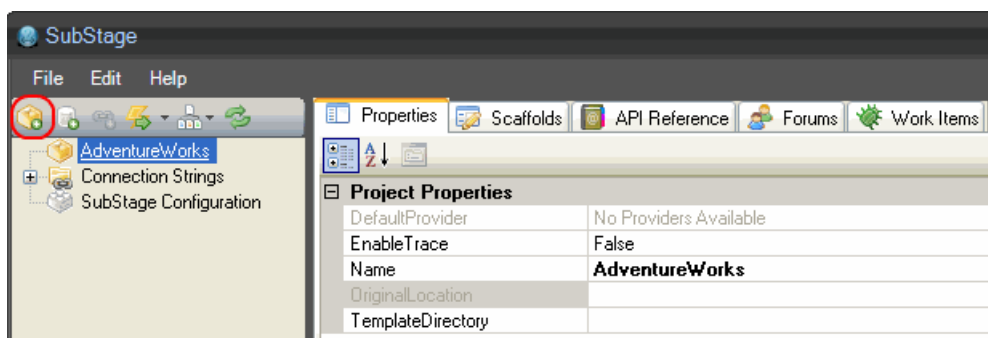
On va refaire plus ou moins les manipulations que l'on vient de faire en manuel. Pour cela, on va commencer par créer une chaîne de connexion, en faisant un clic sur "New Connection String"



On va ensuite configurer notre chaîne de connexion, tout comme on l'a fait dans notre fichier de configuration.

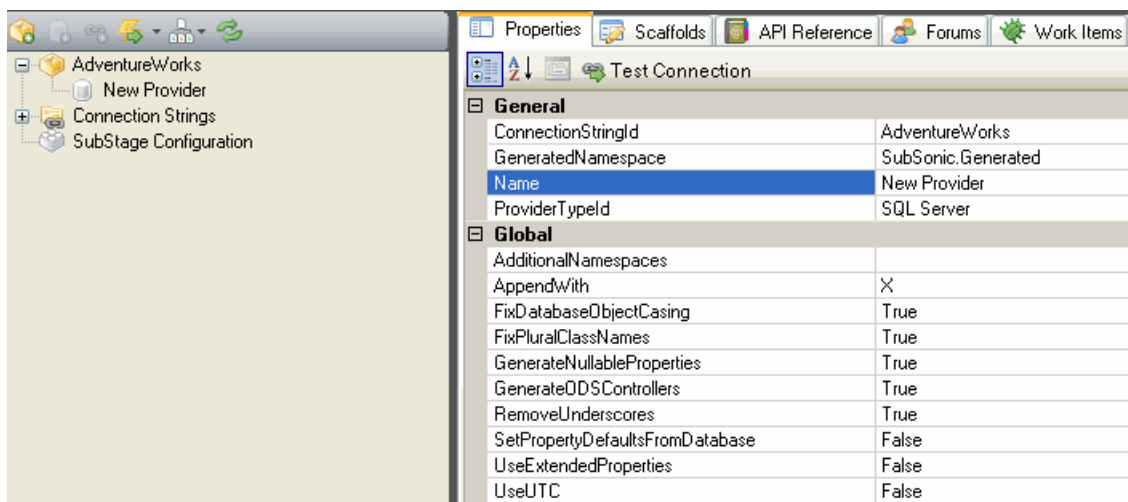


On va ensuite créer un nouveau projet, en cliquant sur "new project", et nommer notre projet (dans mon cas, toujours AdventureWorks)

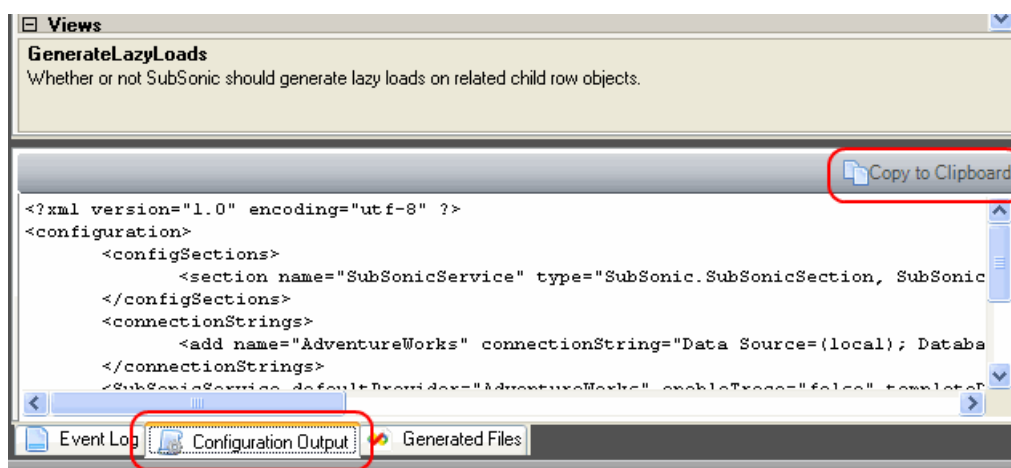


Enfin, on va configurer le fournisseur de données.

Pour cela, on va sélectionner le projet, puis cliquer sur "new provider". Par défaut, le fournisseur sélectionné est SQL Server, ce qui me convient tout à fait. Cet écran va me permettre de gérer de nombreuses choses, comme le namespace dans lequel se trouvera le code généré, le lazy load, la possibilité de sélectionner quelles tables et procédures stockées on veut inclure, et cætera.



Une fois que notre configuration nous convient, il ne reste plus qu'à copier, depuis l'onglet "Configuration Output", la configuration qu'on n'aura plus qu'à coller dans le fichier de configuration.



II-B.3 - Génération du code

Maintenant que l'on a créé (deux fois) notre fichier de configuration, on en est...toujours au même point (sauf si on travaille sur un site web, auquel cas, tout marche déjà).

On va maintenant générer notre couche d'accès aux données. Pour cela, il faut lancer un exécutable, nommé sonic.exe, et situé dans le répertoire d'installation de Subsonic, dans le sous-répertoire SubCommander.

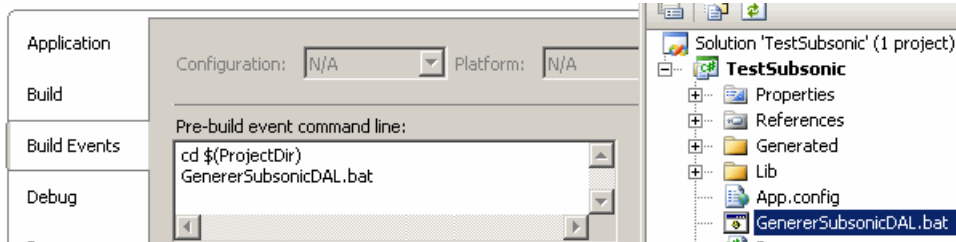
Donc, dans mon cas, **C:\Program Files\SubSonic\SubSonic 2.1 Final\SubCommander\sonic.exe**

Comme je suis fainéant, et que le but est de générer le code régulièrement, je vais ajouter, dans les événements de compilation de mon projet, une condition de précompilation qui va générer le code à chaque fois que je compile mon projet.

Pour cela, je vais ajouter un fichier **GenererSubsonicDAL.bat** dans le répertoire de mon projet, dans lequel je vais ajouter le texte suivant :

```
C:\Program Files\SubSonic\SubSonic 2.1 Final\SubCommander\sonic.exe generate /out Generated
```

Et dans les propriétés de mon projet, dans la rubrique "Build Events", et je vais ajouter comme ligne de commande de pré compilation:

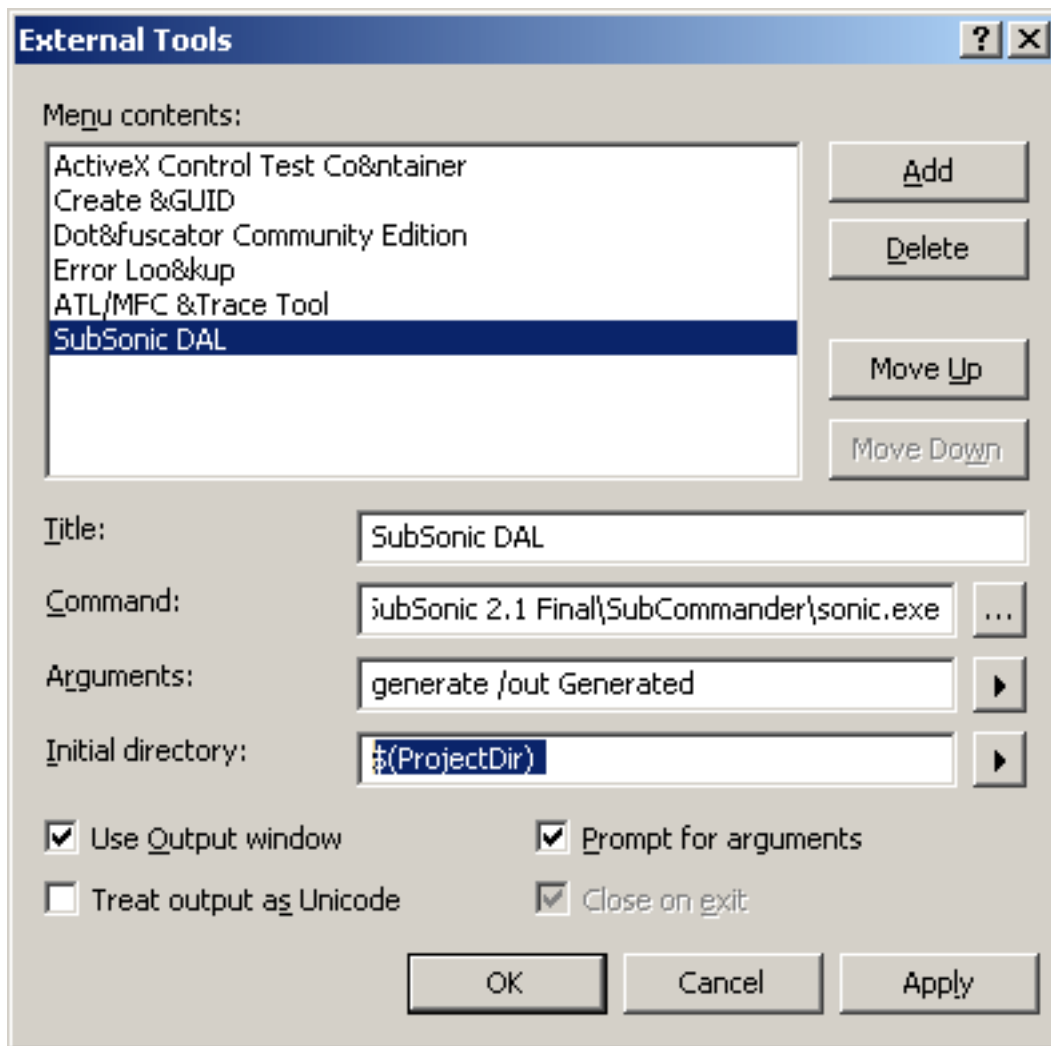


Ainsi, à chaque compilation, mon code est re-généré, et je bénéficie d'une vérification à la compilation de mon code.

! *Historiquement, Subsonic était un projet prévu pour les environnements web, où il était apprécié pour la facilité avec laquelle on générait le code pour une application web. Cependant, même pour une application Web, je déconseille d'utiliser la méthode du BuildProvider, qui, au final, incite à conserver tout le code d'accès aux données dans le même projet que la présentation.*

Une autre option est possible, à savoir ajouter une référence à Subsonic comme outil externe à Visual Studio. Pour cela, il suffit d'aller dans "Outils"/"Outils Externes", puis de cliquer sur ajouter, de renseigner le nom de la commande, l'exécutable à lancer (dans ce cas, sonic.exe), les arguments adéquats (generate /out Generated), et le répertoire initial (ici, on va entrer \$(ProjectDir) pour utiliser le répertoire du projet courant).

Une fois configuré, on doit obtenir l'écran suivant (mon Visual Studio est en anglais, désolé...) :



Et on peut accéder à la nouvelle commande dans Outils/SubsonicDAL

III - Utilisation de la couche d'accès aux données

Maintenant que l'on a une (superbe) couche d'accès aux données, on va voir comment l'utiliser. Plutôt que de faire un long discours, je vais directement ouvrir mon projet web, et essayer d'utiliser ce que SubSonic m'a généré.

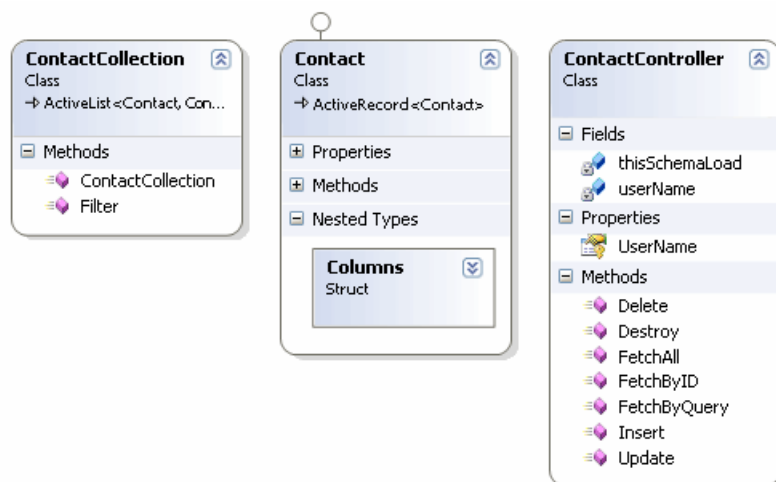
⚠ Avant de commencer, pour utiliser subsonic dans un projet, le projet contenant la couche d'accès aux données nécessite les références suivantes:

- Subsonic
- System.Configuration
- System.Web

Dans le cas où la couche de données est séparée du projet principal, celui-ci nécessite aussi une référence sur la dll de subsonic (pour permettre l'ajout des sections spécifiques dans le fichier de configuration).

III-A - Structure générale des classes

Pour voir ce qui a été généré, je vais me rendre dans la vue par classe de Visual Studio. Dans un diagramme de classe, on voit que, pour la table Contact, les quatre classes suivantes sont générées (Columns est une structure représentant les différentes colonnes de Contact).



Dans notre cas, Contact est la représentation d'un contact. Cette classe va mapper chaque colonne de la table Contact sur une propriété. En plus de ces informations, cette classe va contenir un ensemble de propriétés assez intéressantes. On trouvera entre autres :

- GetPrimaryKeyValue : renvoie la valeur de la clef primaire de l'objet en cours
- IsDirty : l'objet a-t-il été modifié ?
- IsLoaded : l'objet vient-il de la base de données ?
- TableName : nom de la table dans la base de données

Cette classe contiendra, de plus, une structure mappant les noms de colonne sur un ensemble de propriétés, permettant de manipuler les colonnes sans avoir à connaître le nom exact de ces colonnes.

ContactCollection va représenter une collection de contacts. En plus d'être une collection fortement typée de contacts, elle possède un certain nombre de propriétés et de fonctions supplémentaires, que l'on étudiera plus loin.

Enfin, ContactController est une classe qui va encapsuler les méthodes des deux autres classes, de façon à pouvoir sélectionner/modifier/créer les objets sans avoir à manipuler directement les classes des objets. On verra aussi plus loin des exemples des différentes méthodes proposées par subsonic pour manipuler les données.

III-B - Opérations CRUD

Comme toute couche de données qui se respecte, Subsonic permet de faire des opérations CRUD.

III-B-1 - Récupération d'objets depuis la base de données

Récupération d'un seul objet

La façon la plus courante (on va dire, 80% des cas), avec une base de données standard, de récupérer un objet va être de faire une requête sur l'identifiant unique de notre objet, la clef primaire. Pour cela, on a deux façons de faire.

Subsonic génère une méthode FetchById, disponible à la fois sur l'objet de mapping et sur le contrôleur. Par exemple, pour récupérer un **Product** dont la clef primaire est 1, on va pouvoir faire, soit :

```
Product objProduct = Product.FetchById(1);
```


soit :

```
ProductController controller = new ProductController();  
Product objProduct = controller.FetchByID(1);
```

A noter, au cas où il n'existe pas de produit avec l'identifiant 1 dans la base de données, les deux méthodes renverront un produit dont tous les champs ont des valeurs par défaut.

On peut aussi utiliser le constructeur de Product pour récupérer un produit existant dans la base de données. Pour cela, on va soit passer au constructeur l'id demandé, soit lui passer un nom de colonne, et la valeur de l'entité pour cette colonne. Par exemple, les deux lignes de code suivantes récupéreront, dans la base AdventureWorks, le même produit:

```
Product objProduct = new Product(712);  
Product objProduct = new Product(Product.Columns.Name, "AWC Logo Cap");
```

 **Dans le cas où la colonne que l'on utilise dans la seconde forme contient des données qui ne sont pas uniques, Subsonic retournera un ensemble de produits vérifiant la condition, et objProduct contiendra le premier élément de cette liste.**

Récupération d'un ensemble d'objets

Imaginons que l'on veuille récupérer une liste de tous les produits de notre base de données, l'équivalent d'un Select * en SQL. En fonction de ce que l'on veut réaliser, Subsonic nous donne plusieurs possibilités "out of the box."

Si on veut récupérer tous les produits sous la forme d'une collection, il suffit de faire:

```
ProductCollection products = new ProductCollection().Load();  
ProductCollection products = new ProductController().FetchAll();
```

Si on veut récupérer tous les produits dont la couleur est rouge, on va pouvoir simplement faire :

```
ProductCollection products = new ProductCollection()  
    .Where(Product.Columns.Color, "red")  
    .Load();
```

Et me voilà avec une belle collection de produits rouges.

Si je veux trier cette liste par nom, il ne me restera qu'à faire:

```
ProductCollection products = new ProductCollection()  
    .Where(Product.Columns.Color, "red")  
    .OrderByAsc(Product.Columns.Name)  
    .Load();
```

A noter, l'ordre des instructions Load, Where, et OrderByAsc n'aura pas d'impact sur le résultat, mais peut en avoir sur les performances...en effet, si on avait fait:

```
ProductCollection products = new ProductCollection()  
    .Load()  
    .Where(Product.Columns.Color, "red")  
    .OrderByAsc(Product.Columns.Name);
```

On aurait demandé à Subsonic de rapatrier l'ensemble des données, puis de filtrer sur les couleurs, puis de trier les informations, avec un impact non négligeable sur le temps d'exécution nécessaire...

Ces manipulations restent relativement simples, mais couvrent environ 50% des besoins. On verra plus loin que le mécanisme de requêtes de Subsonic est beaucoup plus riche que cela.

III-B-2 - Ajouter ou mettre à jour un objet dans la base de données

Ajouter un nouvel objet dans la base de données se fait de façon très intuitive. Comme mentionné précédemment, on a deux moyens différents pour insérer des objets. Le premier est d'utiliser la classe représentant la table en question. Pour cela, on va faire:

```
objProduct.Save();
```

La seconde option est d'utiliser le contrôleur de la classe. En effet, le contrôleur nous fournit le moyen d'insérer directement l'objet dans la base, en lui passant l'intégralité des données à insérer.

```
new ProductController().Insert(namespace, ProductNumber...
```

Dans les deux cas, le code appelé sera sensiblement le même, le code de la méthode **Save** de **ProductController** appelant le **Save** de **Product**.

```
public void Insert(string Name, string ProductNumber, string Color,
decimal StandardCost, decimal ListPrice, string Size, decimal? Weight,
int? ProductCategoryID, int? ProductModelID, DateTime SellStartDate,
DateTime? SellEndDate, DateTime? DiscontinuedDate, byte[] ThumbnailPhoto,
string ThumbnailPhotoFileName, Guid Rowguid, DateTime ModifiedDate) {
    Product item = new Product();
    item.Name = Name;
    item.ProductNumber = ProductNumber;
    item.Color = Color;
    item.StandardCost = StandardCost;
    item.ListPrice = ListPrice;
    item.Size = Size;
    item.Weight = Weight;
    item.ProductCategoryID = ProductCategoryID;
    item.ProductModelID = ProductModelID;
    item.SellStartDate = SellStartDate;
    item.SellEndDate = SellEndDate;
    item.DiscontinuedDate = DiscontinuedDate;
    item.ThumbnailPhoto = ThumbnailPhoto;
    item.ThumbnailPhotoFileName = ThumbnailPhotoFileName;
    item.Rowguid = Rowguid;
    item.ModifiedDate = ModifiedDate;
    item.Save(Username);
}
```

Mettre à jour un objet se fait de la même façon pour la classe **Product**. La classe **ProductController** va, elle, exposer une méthode **Update**. Cette méthode **Update** est identique à 80% à la méthode **Insert**, elle contient en fait seulement deux lignes supplémentaires, à savoir:

```
item.MarkOld();
item.IsLoaded = true;
```

La raison en est que, lorsque l'on appelle **Save** (**Save** étant une méthode de la classe **ActiveRecord<T>**, qui fait partie du Framework Subsonic), les propriétés **IsDirty** et **IsNew** sont testées, permettant de savoir, au moment de l'appel à **Save**, si on veut faire un insert ou un update.

Les méthodes de sauvegardes proposent des surcharges assez sympathiques. En effet, on peut décider de passer l'utilisateur responsable de la sauvegarde à la fonction **Save**, en faisant, par exemple, ceci:

```
objProduct.Save("pvialatte");
```

Subsonic propose comme convention que, si une colonne **CREATED BY** ou **MODIFIED BY** existe, la chaîne de caractères, l'entier ou le GUID passé à la fonction **Save** renseigne cette colonne. Evidemment, le comportement est différent pour une mise à jour, qui ne mettra à jour que **MODIFIED_BY**.

De la même manière, si une colonne **CREATED_ON** ou **MODIFIED_ON** existe, on va modifier cette colonne automatiquement, à la sauvegarde ou à la mise à jour d'un Produit...


III-B-3 - Suppression d'un objet de la base de données

Tout comme la sauvegarde, la suppression d'un objet en base est simplissime. Pour cela, on va appeler la méthode **Delete** du contrôleur de produits.

```
new ProductController().Delete(712);
```

On pourra aussi, appeler la méthode **Delete** statique de l'objet **Product**. On peut soit supprimer un objet par son **Id**, soit par une autre valeur unique.

```
Product.Delete(712);  
Product.Delete(Product.Columns.Name, "AWC Logo Cap");
```

 *Si le colonne choisie n'est pas unique, on risque de supprimer plusieurs Produits dans la base. En effet, en interne, le Framework va faire un where...*

Une fois de plus, une convention spécifique de Subsonic existe au niveau de la suppression. En effet, si une colonne **DELETED** ou **IS_DELETED** (de type booléen ou bit) existe, on ne va pas supprimer l'enregistrement de la base, mais simplement modifier la valeur de la colonne de suppression de façon à ce qu'elle soit représentée comme supprimée. Attention, c'est une convention, les requêtes de sélection ne se limitent pas, elles aux éléments dont cette valeur est fausse...

Bien entendu, si ces colonnes n'existent pas, on efface les enregistrements...

Pour être sûr de supprimer définitivement les enregistrements, on peut utiliser la méthode **Destroy**, qui s'appelle exclusivement à partir des objets:

```
Product.Destroy(712);  
Product.Destroy(Product.Columns.Name, "AWC Logo Cap");
```

Les collections permettent aussi de supprimer un ensemble d'objets. Pour cela, il faut préalablement charger une liste d'objets, et marquer les objets que l'on veut supprimer dans cette liste. Ceci se fait en utilisant, au choix, **BatchDelete** ou **SaveAll**, après avoir supprimé les éléments que l'on voulait supprimer

```
ProductCollection products = new ProductCollection()  
    .Load()  
    .Where(Product.Columns.Color, "red")  
    .OrderByAsc(Product.Columns.Name);  
while(products.Count > 0) {  
    products.RemoveAt(0); // RemoveItem() gets called  
}  
  
products.SaveAll(); // supprime tous les objets, et sauvegarde les modifications  
  
products.BatchDelete(); // supprime tous les objets dans une transaction, ne sauve pas les autres objets
```

III-C - Requêtes

La où on va **vraiment** gagner à utiliser Subsonic, c'est sur les requêtes. Subsonic possède en effet un système de requête simple, puissant, et surtout, dans la majorité des cas, qui permet de créer des requêtes qui seront vérifiées à la compilation par rapport aux données générées. Pour ceux qui utilisent Linq To Sql ou l'Entity Framework, cela peut ne pas représenter grand chose, mais pour tous les pauvres mortels qui sont bloqués sur le Framework 2.0, cela change la vie.

Jusqu'à la version 2.1, l'élément de base pour effectuer une requête avec Subsonic était l'objet SubSonic.Query. Depuis la version 2.1, une nouvelle fabrique d'objet a été introduite, pour gérer les requêtes, l'objet DB. Cet objet va exposer les fonction suivantes:

- Select
- Insert
- Update
- Delete
- Query

Ces objets gèrent les sélections, insertions, mises à jour, suppressions et les requêtes Ad Hoc. De plus, ils gèrent les jointures entre tables, les IN, le paging, les transactions, les sous requêtes, les agrégats, et même un mécanisme de batch.

III-C-1 - Requêtes de sélection

Le but des requêtes de sélection de Subsonic est d'être le plus proche possible syntaxiquement d'une requête SQL. L'API de Subsonic est en effet conçue sur le mode "fluent interface", ce qui permet de chaîner les contraintes.

La syntaxe de base sera donc de la forme :

```
DB.Select("Champ1", "Champ2", "Champ3").
    From("Table").
    Where("Champ1").
    IsEqualTo(1).
    And("Champ2").
    IsEqualTo(2);
```

La fonction DB.Select renverra un objet de type SqlQuery. Cet objet de type query expose les méthodes suivantes:

- GetRecordCount: renvoie le nombre d'enregistrement correspondant dans la base de données
- Execute : Equivaut à ExecuteNonQuery, renverra le nombre d'enregistrements modifiés
- ExecuteScalar : Equivaut à ExecuteScalar, renvoie le premier élément retourné, sous forme d'objet
- ExecuteScalar<T> : Equivaut à ExecuteScalar, renvoie le premier élément retourné, sous forme de T
- ExecuteReader : Equivaut à ExecuteReader, renvoie un IDatareader
- ExecuteDataSet : Renvoie le résultat de la requête sous forme de DataSet
- ExecuteSingle<T>: Renvoie la première ligne du résultat, en le convertissant en objet de type T
- ExecuteAsCollection<T>: Renvoie le résultat de la requête, sous forme d'une collection de T
- ToString : Renvoie la requête SQL qui va être exécutée

Pour que ce soit plus parlant, voici une liste de requêtes, récupérées dans les tests unitaires de la solution de Subsonic.

On peut donc exécuter les requêtes avec des chaines :

```
int records = new DB.Select("productID").
```

```
From("Products").GetRecordCount();
```

La même, avec des colonnes typées :

```
int records = new DB.Select(Product.ProductIDColumn).  
From<Product>().GetRecordCount();
```

ou encore :

```
int records = new DB.Select(Product.Columns.ProductID).  
From<Product>().GetRecordCount();
```

Pour récupérer un objet, on va faire :


```
Product p = new DB.Select().From<Product>().  
Where("ProductID").IsEqualTo(1).ExecuteSingle<Product>();
```

ou même :

```
Product p = new Select().From<Product>().  
Where(Product.Columns.ProductID).  
IsEqualTo(1).ExecuteSingle<Product>();
```

Si on veut récupérer une collection, on va faire:

```
ProductCollection products = DB.Select().From<Product>().  
.Where(Product.Columns.Category).IsEqualTo(5)  
.And(Product.Columns.ProductID).IsGreaterThan(50)  
.ExecuteAsCollection<ProductCollection>();
```

 *Si l'objet vers lequel on fait la conversion avec **ExecuteSingle** ou **ExecuteAsCollection** n'est pas un des objets de Subsonic, le Framework va mapper les résultats de la requête sur les champs de l'objet demandé, en utilisant, dans ce cas, la réflexion pour détecter quel champ de la base mapper sur quelle propriété. Le coût est assez infime, mais il vaut mieux le savoir à l'avance.*

Et, pour une jointure, on va faire:

```
CustomerCollection customersByCategory = new Select()  
.From<Customer>()  
.InnerJoin<Order>()  
.InnerJoin(OrderDetail.OrderIDColumn, Order.OrderIDColumn)  
.InnerJoin(Product.ProductIDColumn, OrderDetail.ProductIDColumn)  
.Where("CategoryID").IsEqualTo(5)  
.ExecuteAsCollection<CustomerCollection>();
```

III-C-2 - Requetes d'insertion, suppression, mises à jour

De la même façon que pour les requêtes de sélection, Subsonic expose des interfaces permettant des requêtes d'insertion, de suppression, et de mises à jour assez puissantes.

Elles prendront la forme suivante :

```
new Insert().Into<Region>().Values("test1").Execute();
new
Update<Region>().Set("UnitPrice").EqualTo("test2").Where("regiondescription").IsEqualTo("test1").Execute();
new Delete().From<Region>().Where("regiondescription").Like("test%").Execute();
```

Une fois de plus, on retrouve une pseudo-syntaxe SQL, à laquelle on s'habitue très vite.

III-C-3 - Requêtes d'agrégation

```
double result = new
Select(Aggregate.Sum("UnitPrice*Quantity", "ProductSales"))
.From<OrderDetail>()
.ExecuteScalar<double>();
```

```
double result = new
Select(Aggregate.Avg(Product.UnitPriceColumn))
.From<Product>()
.ExecuteScalar<double>();
```

```
double result = new
Select(Aggregate.Max(Product.UnitPriceColumn))
.From<Product>()
.ExecuteScalar<double>();
```

```
double result = new
Select(Aggregate.Min(Product.UnitPriceColumn))
.From<Product>()
.ExecuteScalar<double>();
```

III-C-4 - Transactions

Pour finir avec les requêtes, on va voir les possibilités de transactions de Subsonic

Subsonic propose deux méthodes différentes pour profiter du mécanisme de transactions. La première est de générer un ensemble de requêtes, et de les passer à la fonction **ExecuteTransaction** de `SubSonic.SqlQuery`

```
List<Insert> queries = new List<Insert>();
queries.Add(new Insert().Into(Region.Schema).Values("test1"));
queries.Add(new Insert().Into(Region.Schema).Values("test2"));

SubSonic.SqlQuery.ExecuteTransaction(queries);
```

La seconde est de créer un nouveau scope de transaction, et d'exécuter toutes nos requêtes dans ce scope.

```
using(TransactionScope scope = new TransactionScope()){
    new Insert().Into(Region.Schema).Values("test1");
    new Insert().Into(Region.Schema).Values("test2");
    new Insert().Into(Region.Schema).Values("test3");
}
```

Dans les deux cas, Subsonic fera automatiquement le commit final, et un rollback en cas d'erreur

IV - Conclusion

Cet article ne présente que dans les grandes lignes Subsonic, qui possède encore quelques fonctionnalités sympathiques, quoique plus avancées. Une question qu'on pourrait se poser à la lecture de l'article est l'intérêt d'investir du temps dans l'apprentissage d'un Framework d'accès aux données, lorsque des ORM plus puissants existent déjà sur le marché (NHibernate, LightSpeed), et que des Frameworks d'accès aux données estampillés Microsoft commencent à faire parler d'eux (Entity Framework).

A mon avis, chacun de ces produits répond à une problématique spécifique, NHibernate est très puissant, mais demande un temps d'apprentissage assez long, Entity Framework est encore en cours de maturation, et a encore quelques limitations (dont celle, qui ne changera pas, de ne pouvoir fonctionner qu'à partir du Framework 3.5).

L'ambition de Subsonic est de répondre à 80% des cas de figures, fourchette dans laquelle on se retrouve quasi systématiquement dans le cadre du développement d'applications de gestion, et permet de commencer à travailler vite une fois que la base de données est fonctionnelle.

V - Remerciements

Mille mercis à toute l'équipe de rédaction .Net, en particulier à **tomlev**, ainsi qu'à **evarisnea** pour la relecture.