Introduction to Model Driven Development with Sculpture – Part 1

Ahmed Negm, 3 Sep 2008

★★★★★ 5.00 (23 votes) Rate: ☆☆☆☆☆

This article introduces how to create and manage .NET enterprise applications using your favorite technology (Data Access Application Block, LINQ, NHibernate, ASMX, and WCF) with the Model Driven Development approach by Sculpture.

Download sample - 625 KB

Introduction

What is Sculpture?

- Sculpture is a .NET open source Model-Driven Development code generation framework, ideal for creating and managing .NET Enterprise Applications.
- With Sculpture, you can model your application components, and then transform this model to deployable components for your favorite technology.
- Sculpture comes with a host of ready-made Molds (the word “Molds” come from molding) like DAAB, NHibernate, LINQ, WCF, ASMX, SQL Server, MYSQL ....
- Sculpture contains a Guidance Package for building your own Mold, or customizes existing ones. If you have a custom architecture, using this Guidance Package, you can build a custom code generator with your favorite technology to fit your needs.
- Sculpture can generate any kind of text output using templates (source code, database scripts, web pages, XML, configuration files, etc.).
- Sculpture raises the level of abstraction; for example, the data access layer part in your model may be transformed to an NHibernate implementation, and with minor changes, it can be transformed to a LINQ implementation, and in the future can be transformed to an “X” framework, which we don’t know about now.

Sculpture is divided into:

- **Sculpture Core Engine**: it is a platform that hosts the Molds, and takes care of making all the Molds work together. It includes the Model Designer, Mold discovery and loader, a generic validation engine, a generic code generation engine, a command holder, and an editor controls holder.
- **Mold**: The primary plug-in of the Sculpture framework. With Molds, you can extend all the power of Sculpture to manage the model and the produced code acoording your needs. Molds provide the ability to process model elements.

Ready-made Molds

- For Data Source Layer:

    1. SQL Server

- For Data Access Layer:

    1. DAAB (Data Access Application Block).
    2. NHibernate.
    3. LINQ to SQL.

- For Service Layer:

    1. Service Library.
    2. ASMX (ASP.NET Web Service).
    3. WCF (Windows Communication Foundation).

Let's begin

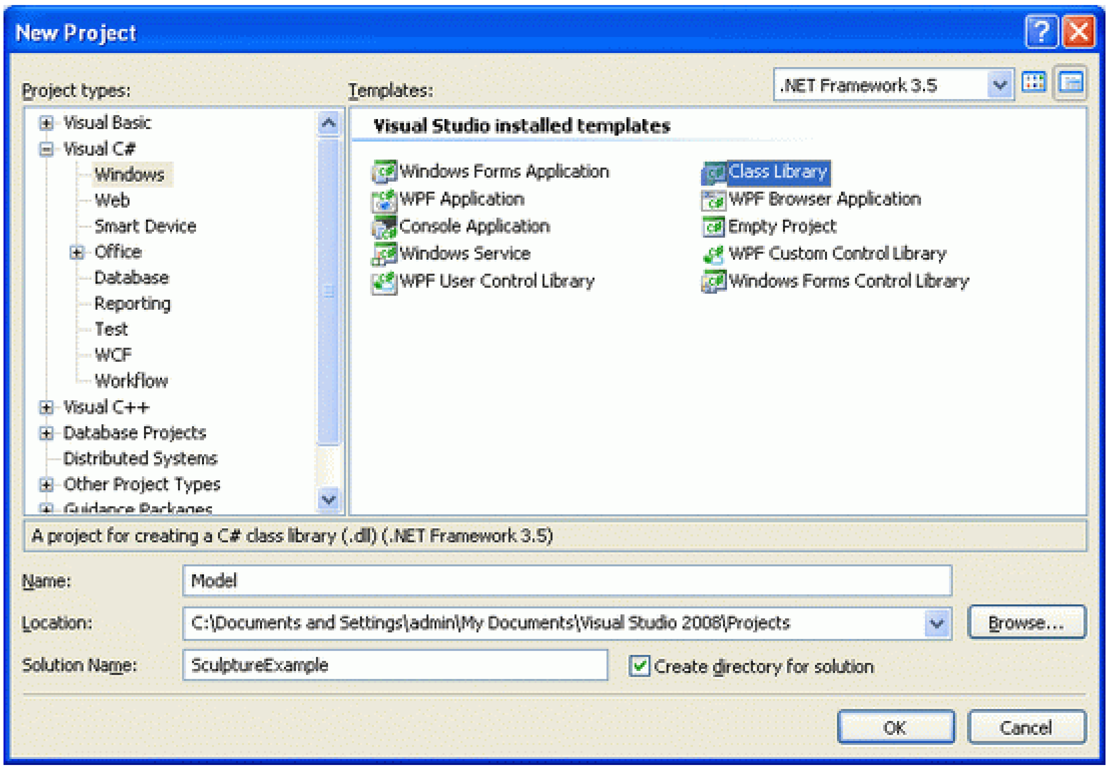
Through this article, I am going to show you how to use Sculpture to create and manage your applications. We will begin by building Entities and a Data Access Layer by using LINQ, and a Service Layer by using WCF, Then, we will generate the same model with an NHibernate + ASMX web service.

Table of contents

1. Setting up your projects

The first thing you need to do is set up your projects.

1. Create a new Visual C# Class Library project, In the 'Name' field, type 'Model'. In the 'Solution Name' field, type 'SculptureExample', and click OK. This project will host the model.



A project for creating a C# class library (.dll) (.NET Framework 3.5)

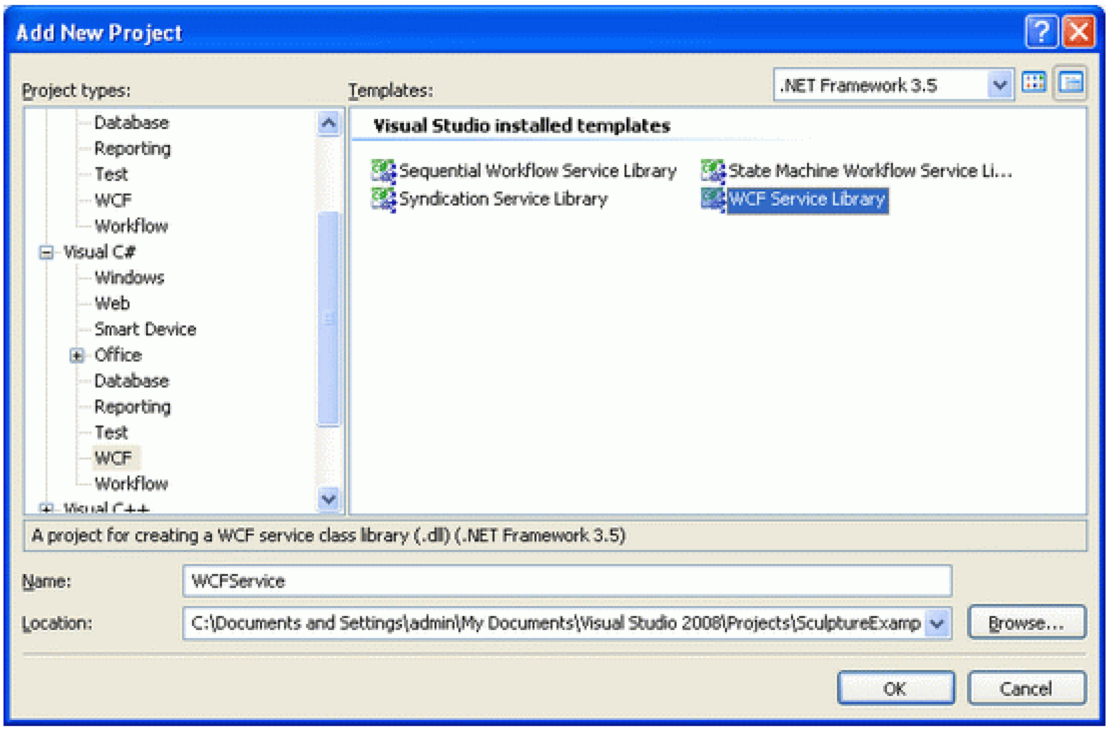
Name: Model

Location: C:\Documents and Settings\admin\My Documents\Visual Studio 2008\Projects

Solution Name: SculptureExample

[✓] Create directory for solution

2. Add two class library projects for this solution, one called ‘Entities’ that will host the business entities, and another called ‘DataAccess’ that will host the repository classes.
3. Add a WCF Service Library called ‘WCFService’ that will host the service layer.

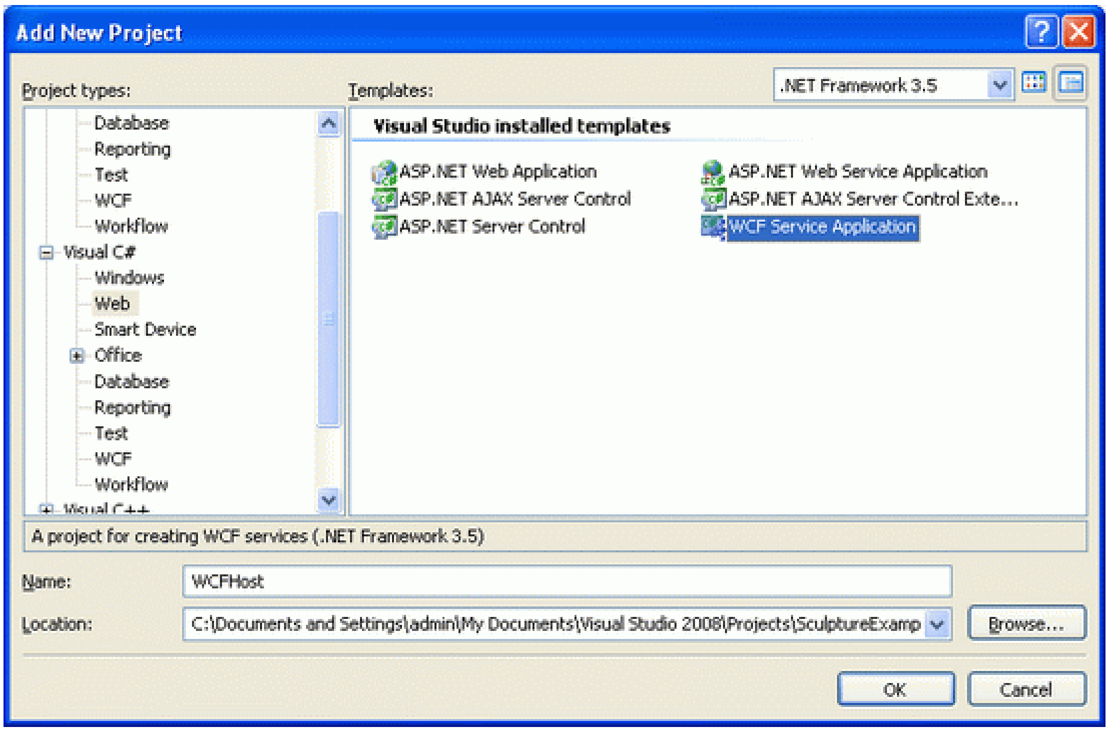


A project for creating a WCF service class library (.dll) (.NET Framework 3.5)

Name: WCFService

Location: C:\Documents and Settings\admin\My Documents\Visual Studio 2008\Projects\SculptureExamp

4. Add a WCF Service application called ‘WCFHost’ that will be the host application for the WCF Service.

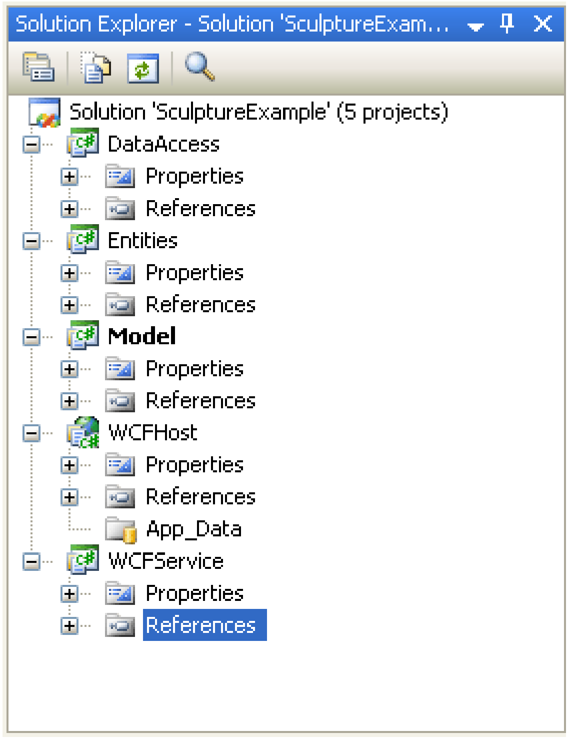


A project for creating WCF services (.NET Framework 3.5)

Name: WCFHost

Location: C:\Documents and Settings\admin\My Documents\Visual Studio 2008\Projects\SculptureExamp

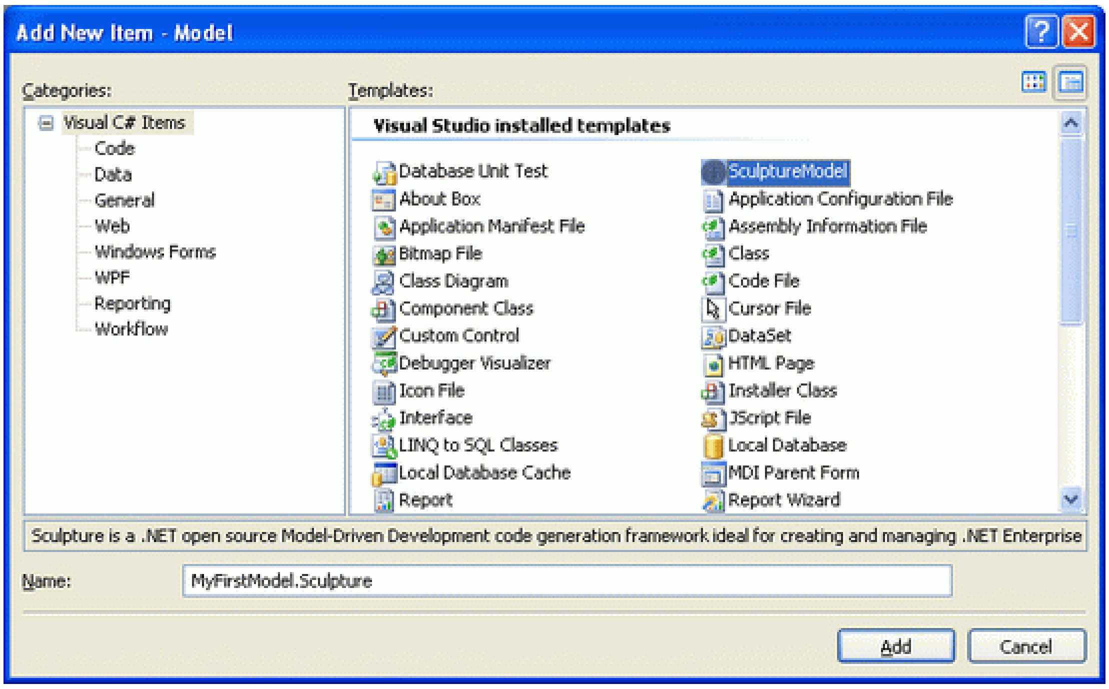
5. Delete the unused files from all of these projects; the Ssolution Explorer must be like in the following figure.



```
Solution 'SculptureExample' (5 projects)
  DataAccess
    Properties
    References
  Entities
    Properties
    References
  Model
    Properties
    References
  WCFHost
    Properties
    References
    App_Data
  WCFService
    Properties
    References
```

2. Add the Sculpture model to your application

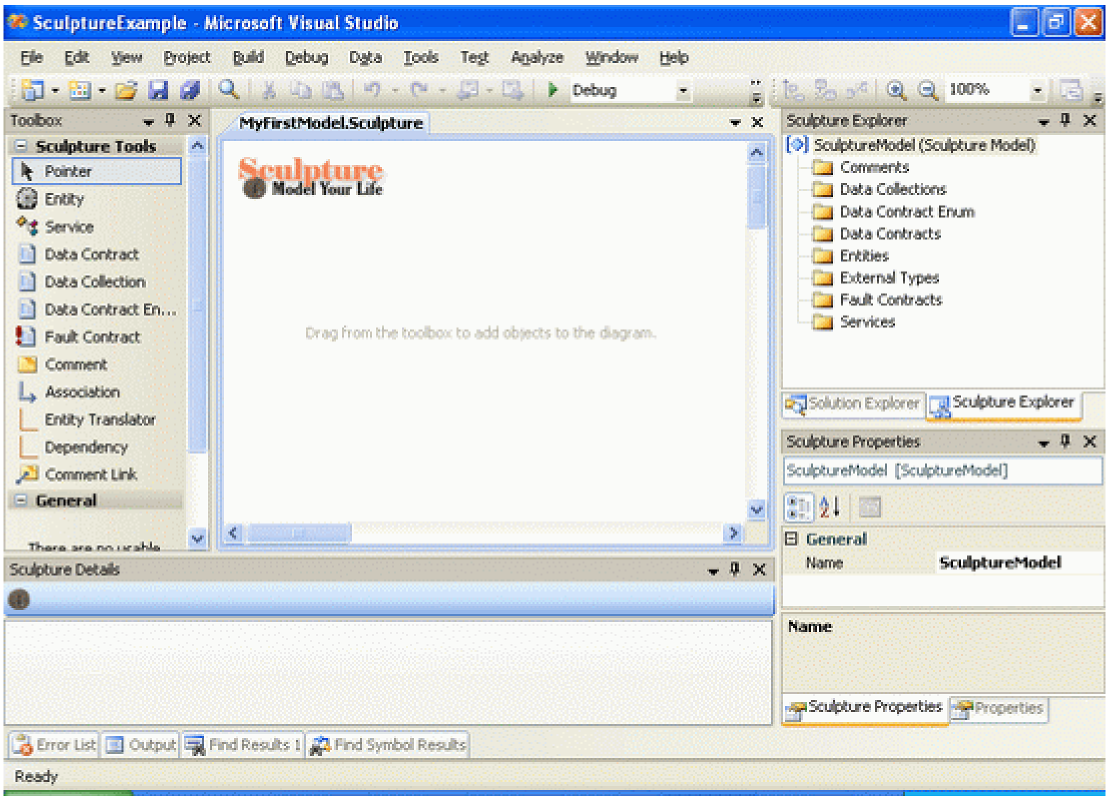
1. Point to the model project, then right click Add -> New Item; from the template list, choose *Sculpture Model*; in the 'Name' field, type 'MyFirstModel.Sculpture'.



Sculpture is a .NET open source Model-Driven Development code generation framework ideal for creating and managing .NET Enterprise

Name: MyFirstModel.Sculpture

2. Now, you have Sculpture model in your development environment. The Sculpture environment contains five parts:



1. **Diagram Surface** hosts the Model elements.
2. **Toolbox** contains the available tools to design your application components.
3. **Sculpture Explorer** holds all the elements of the model from entities to data contracts to services; you can use it to navigate through all the model elements easily.
4. **Sculpture Properties** window holds the properties of the model elements.
5. **Sculpture Details** window holds editor controls for creating and configuring your model easily.

3. Add Molds to your model

As discussed before, the Sculpture Core Engine does nothing alone, you must plug it with your favorite Molds; we will begin by adding these Molds:

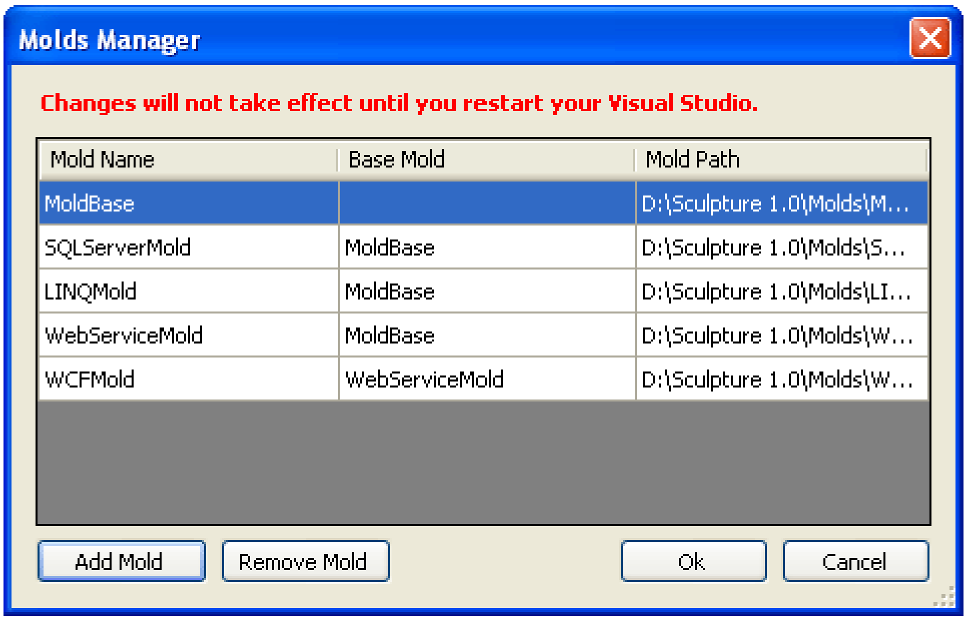
- **Mold Base**: is the base of all the other Molds; it contains all the common properties and activities that are shared among the Molds. All Molds must be inherited from it directly or from another Mold.

- **SQL Server Mold**: concerns in all activities related to Microsoft SQL Server. The reverse engineering engine parses the database and translates it to a model, so you can start your project from a database. Additionally, any updates in the database schema can reflect on the model easily without losing any metadata. The reverse engineering engine supports building entities from views to generate a script for the CRUD stored procedures, and database schema, so you can start your application from the model, design your entities, then generate the script of the database, and the script of the CRUD stored procedures (if needed).
- **LINQ Mold**: this Mold generates LINQ to SQL entities, and Data Context, then generates repository classes for each entity with the default data access methods (Get All, Get by Id, Find, Save, Update, and Delete …).
- **Web Service Mold**: this Mold does not generate any code, it just gathers the common properties for child web service Molds (ASMX, WCF); and if you want to generate any kind of web service (ASMX or WCF), you must include this Mold in your model.
- **WCF Mold**: this mold generates WCF Services, it adds WCF attributes to data contracts, data members, data collections, data contract enums, service contracts, and service implementation classes. It generates an SVC file and a *web.config* file to the host project. The SVC file has a reference to the service implementation, so you can publish the services. It also generates an *app.config* file in the service implementation project that gives you the opportunity to test your services with the Microsoft WCF Test Client. It includes some properties specific to WCF as Concurrency Mode, Instance Context Mode, Protection Level, Session Mode, End Point Name, End Point Binding Type, and End Point Address for Services, and Async Pattern, Is Initiating, Is Terminating, Protection Level, and Reply Action for Operations.

Each one of these Molds contains:

- An XML file with the extension '.Mold' that contains the whole structure of the Mold.
- A DLL file that contains the attached code to the Mold.
- A *Templates* folder that contains T4 templates belonging to this Mold.

To add Molds to your model, press the 'Molds Manager' button in the Sculpture Details Window, then add your Molds by choosing '*.mold*' files. (You will find the ready-made Molds in the Sculpture installed directory.) The Molds Manager looks like the following figure, save the model. Unfortunately, you must restart your Visual Studio so the Molds can take effect.

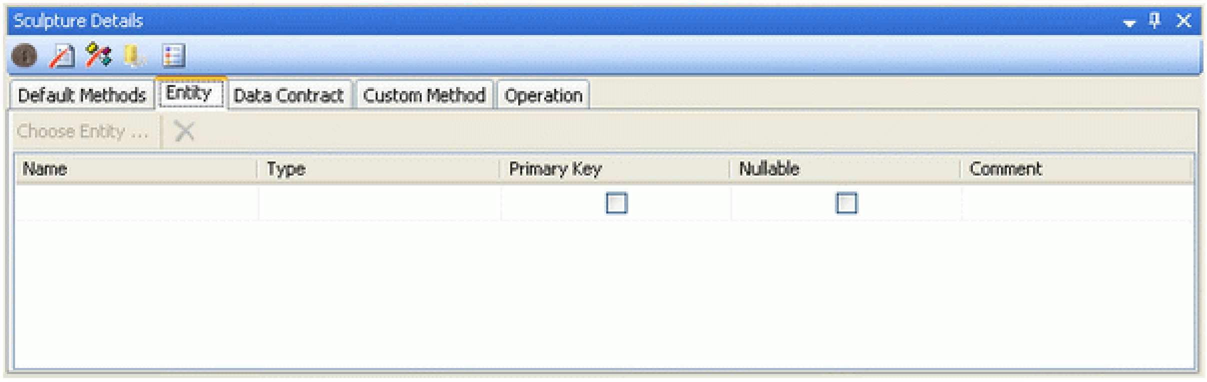


Changes will not take effect until you restart your Visual Studio.

Mold Name	Base Mold	Mold Path
MoldBase		D:\Sculpture 1.0\Molds\M…
SQLServerMold	MoldBase	D:\Sculpture 1.0\Molds\S…
LINQMold	MoldBase	D:\Sculpture 1.0\Molds\LI…
WebServiceMold	MoldBase	D:\Sculpture 1.0\Molds\W…
WCFMold	WebServiceMold	D:\Sculpture 1.0\Molds\W…

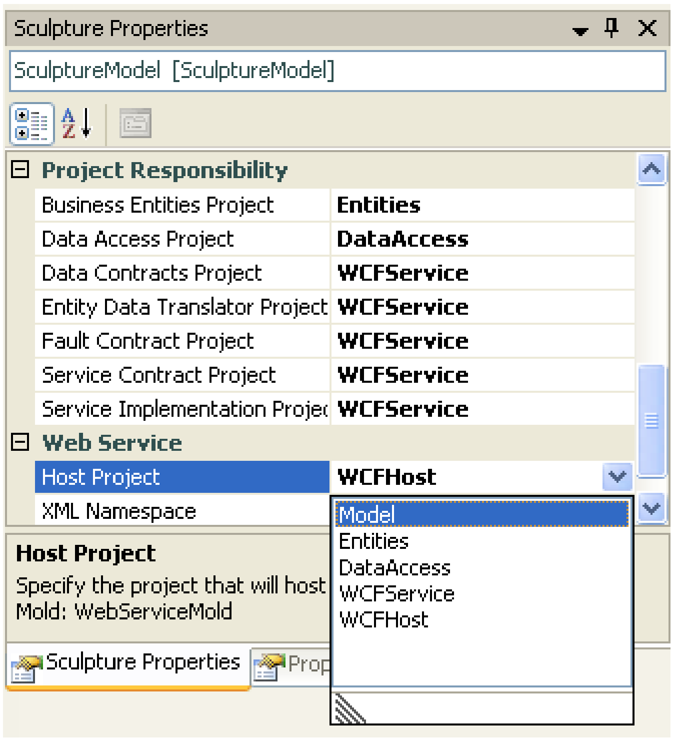
4. Design the model

After restarting Visual Studio, right click on the model surface and press 'Details'. You will notice that some buttons and editors have appeared in the Sculpture details window.



Name	Type	Primary Key	Nullable	Comment
		☐	☐	

The first thing we need to do in the model is specify the project responsibilities; this tells Sculpture where the code will be generated, and sets the values as in the next figure. (In this sample, we generate the data contracts, translators, fault contracts, service contracts, and service implementation in one project; but in a real application, you might specify a project for each one).



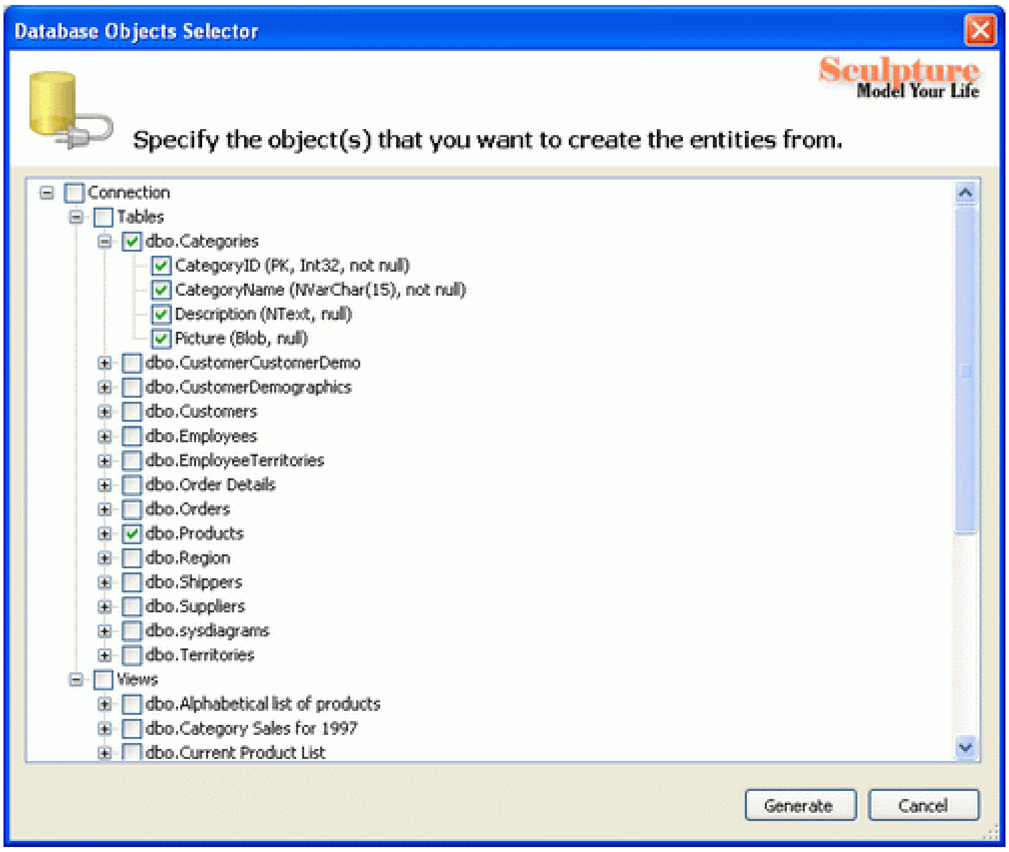
Project Responsibility	
Business Entities Project	Entities
Data Access Project	DataAccess
Data Contracts Project	WCFService
Entity Data Translator Project	WCFService
Fault Contract Project	WCFService
Service Contract Project	WCFService
Service Implementation Projec	WCFService
Web Service	
Host Project	WCFHost
XML Namespace	

Model
Entities
DataAccess
WCFService
WCFHost

Host Project
Specify the project that will host
Mold: WebServiceMold

4-1. Designing the entities

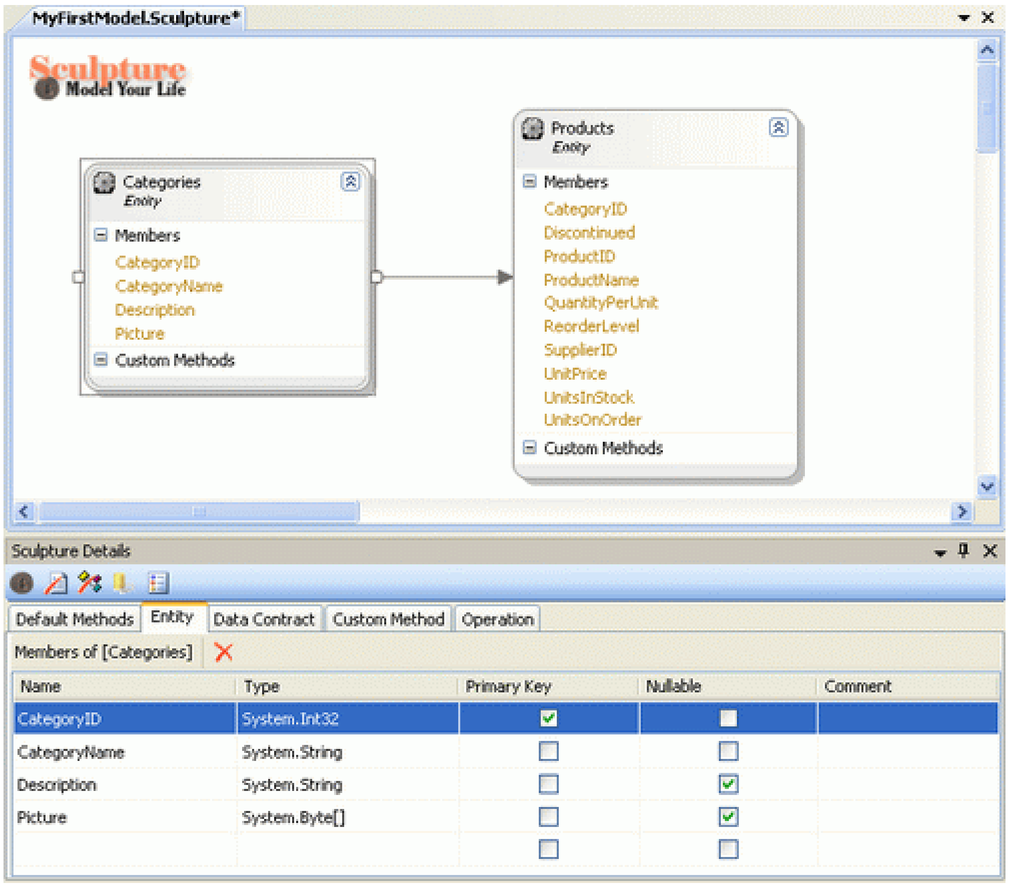
- From the toolbox, you can model your entities and their relationships by drag and drop into the model surface.
- In this article, we will use the reverse engineering tool that is supported by the SQL Server Mold to generate entities and its associations from the Northwind database:
    - Click on the model surface to get the properties of the model. In the Sculpture Properties window, you will find a connection string property, set it for the Northwind database by using the Connection String Wizard.
    - Press 'Generate entities from SQL Server database' in the Sculpture editor tool bar.



Specify the object(s) that you want to create the entities from.

```
⊟ ☐ Connection
   ⊟ ☐ Tables
      ⊟ ☑ dbo.Categories
            ☑ CategoryID (PK, Int32, not null)
            ☑ CategoryName (NVarChar(15), not null)
            ☑ Description (NText, null)
            ☑ Picture (Blob, null)
      ⊞ ☐ dbo.CustomerCustomerDemo
      ⊞ ☐ dbo.CustomerDemographics
      ⊞ ☐ dbo.Customers
      ⊞ ☐ dbo.Employees
      ⊞ ☐ dbo.EmployeeTerritories
      ⊞ ☐ dbo.Order Details
      ⊞ ☐ dbo.Orders
      ⊞ ☑ dbo.Products
      ⊞ ☐ dbo.Region
      ⊞ ☐ dbo.Shippers
      ⊞ ☐ dbo.Suppliers
      ⊞ ☐ dbo.sysdiagrams
      ⊞ ☐ dbo.Territories
   ⊟ ☐ Views
      ⊞ ☐ dbo.Alphabetical list of products
      ⊞ ☐ dbo.Category Sales for 1997
      ⊞ ☐ dbo.Current Product List
```

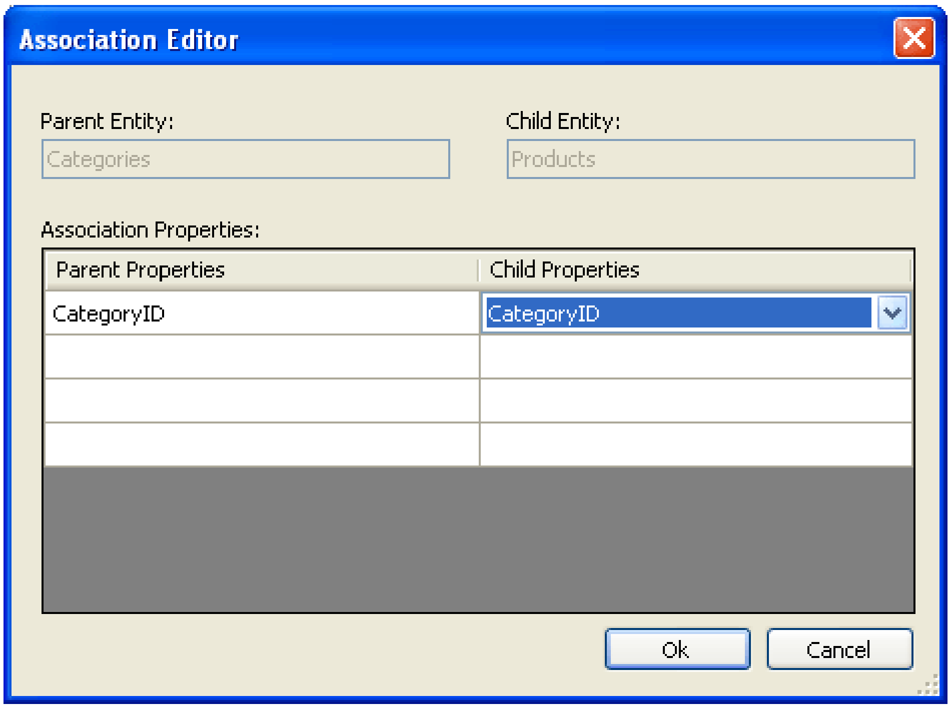
- In the Database Object Selector Form, choose two tables (*Categories* and *Products*), then press Generate.
- The two entities appear in the model.



Members of [Categories]

Name	Type	Primary Key	Nullable	Comment
CategoryID	System.Int32	☑	☐	
CategoryName	System.String	☐	☐	
Description	System.String	☐	☑	
Picture	System.Byte[]	☐	☑	
		☐	☐	

- You may edit the association relationship by selecting the association, and from the Sculpture Properties window, press the button inside the participating properties' property, and choose the relation elements as in the following figure:


Association Editor

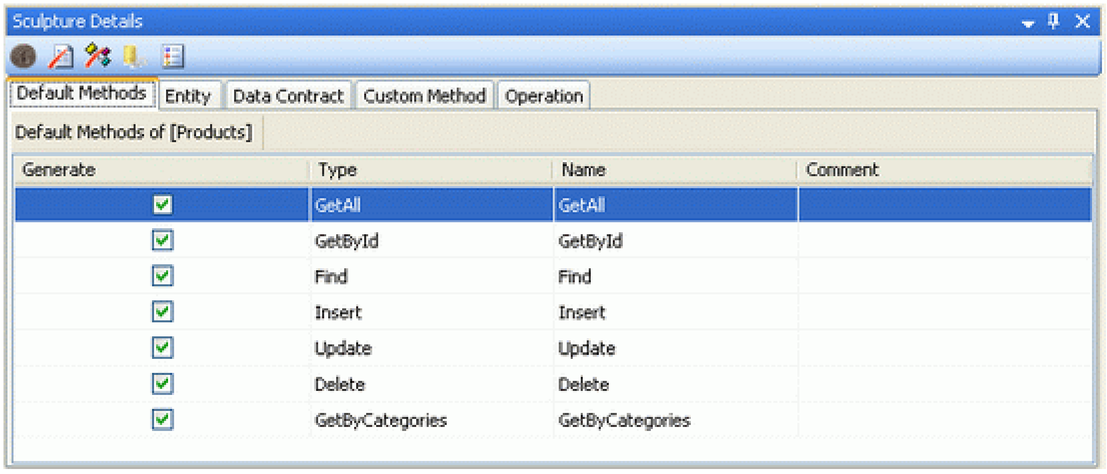
Parent Entity: Categories

Child Entity: Products

Association Properties:

Parent Properties	Child Properties
CategoryID	CategoryID

- If you choose one of them, you can edit its members from the Entity tab in the Sculpture Details window.
- Also, from the Default Methods tab, you can edit the data access default methods. Choose generating the method or not, choose another name to the method, and type comments for this method.



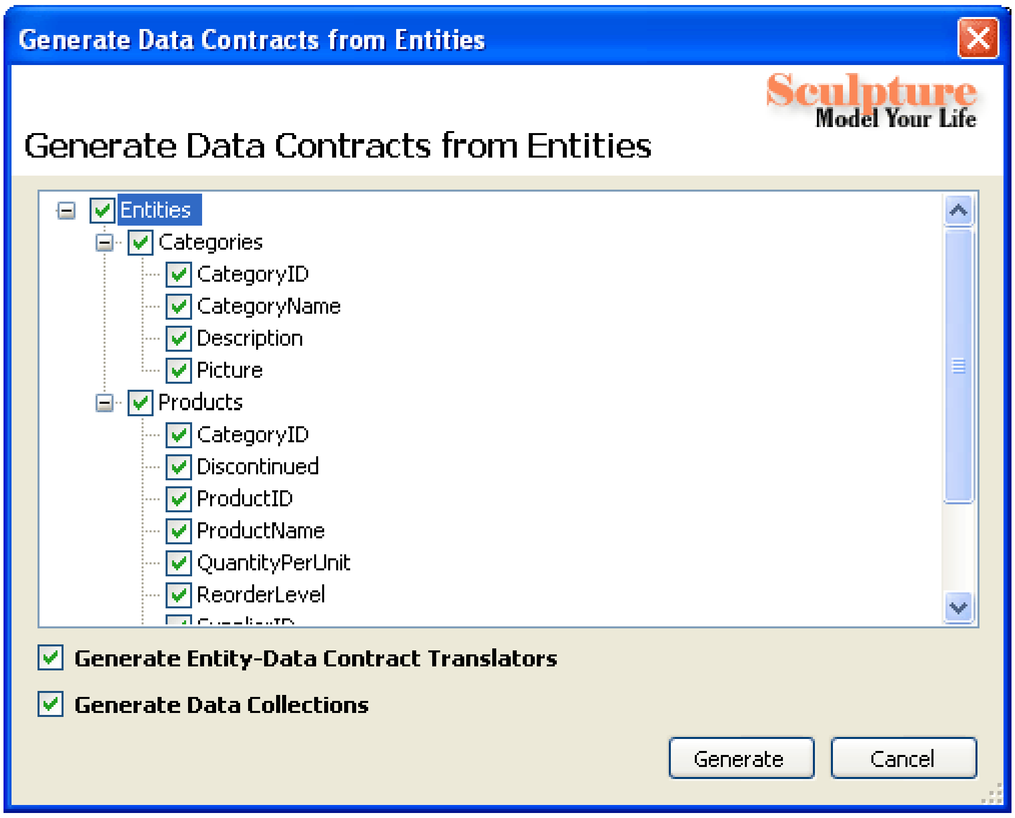
Default Methods of [Products]

Generate	Type	Name	Comment
✓	GetAll	GetAll	
✓	GetById	GetById	
✓	Find	Find	
✓	Insert	Insert	
✓	Update	Update	
✓	Delete	Delete	
✓	GetByCategories	GetByCategories	

- In addition to the default methods, you can add custom data access methods and specify their return types and parameters (the custom methods added to the interface of the data access class).

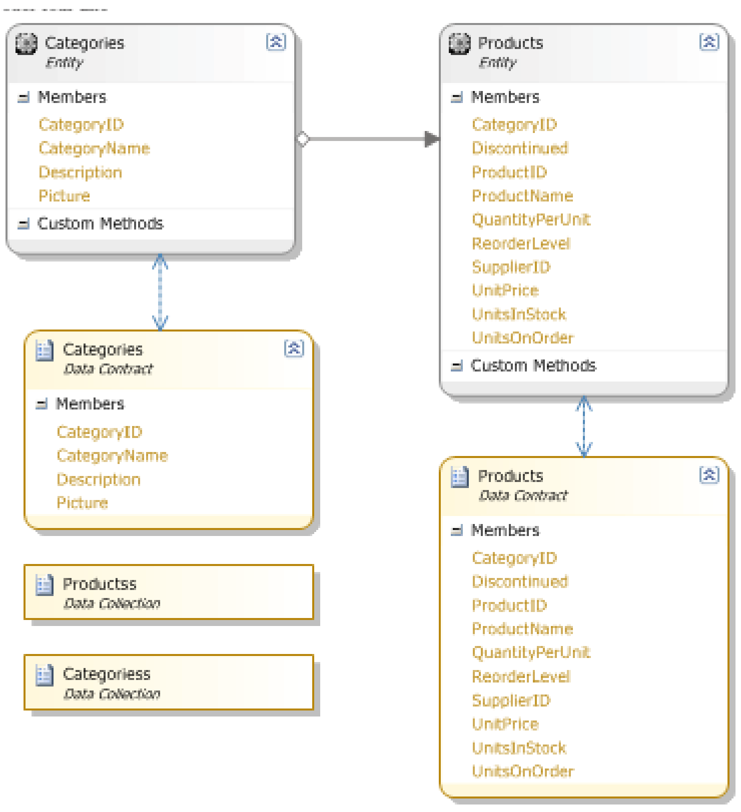
4-2. Designing the data contracts

- Data contracts (some times called ‘Data transfer objects’ or ‘Value objects’) are the objects that will propagate to the higher level (User Interface).
- You can drag and drop data contracts, data collections, fault contracts (for WCF use), and data contract enumerations from the tool box into the model surface.
- Instead of creating a data contract for each entity manually, Sculpture provides an option to generate data contracts from entities automatically.
    - Press ‘Generate data contracts from entities’ in the Sculpture editor tool bar.

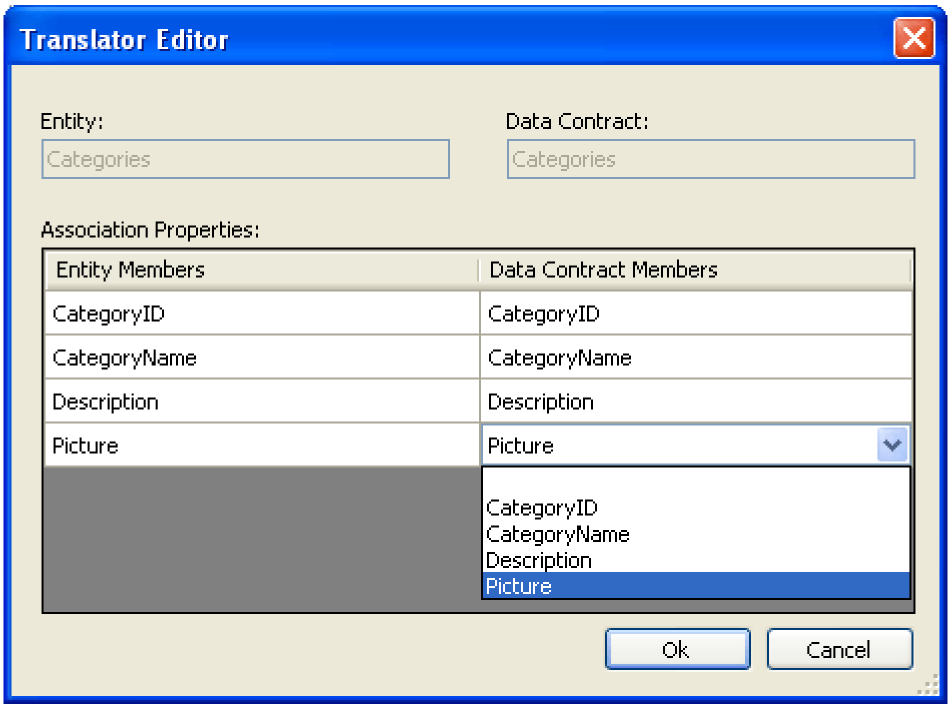


- You have the option to generate an Entity – Data Contract Translator; this will generate a static class that can be used for translation to and from entities and data contracts.
- There is another option to generate data collections for the generated data contracts.
- Choose all entities in the tree, then press Generate.

- You will find new data contracts and data collections added to the model, and there are relations between the entities and the corresponding data contracts.



- You may edit the mapping between the properties by selecting the relation, and from the Sculpture Properties window, press the button inside the mapping members' property, and choose the corresponding members as in the next figure:



Translator Editor

Entity: Categories

Data Contract: Categories

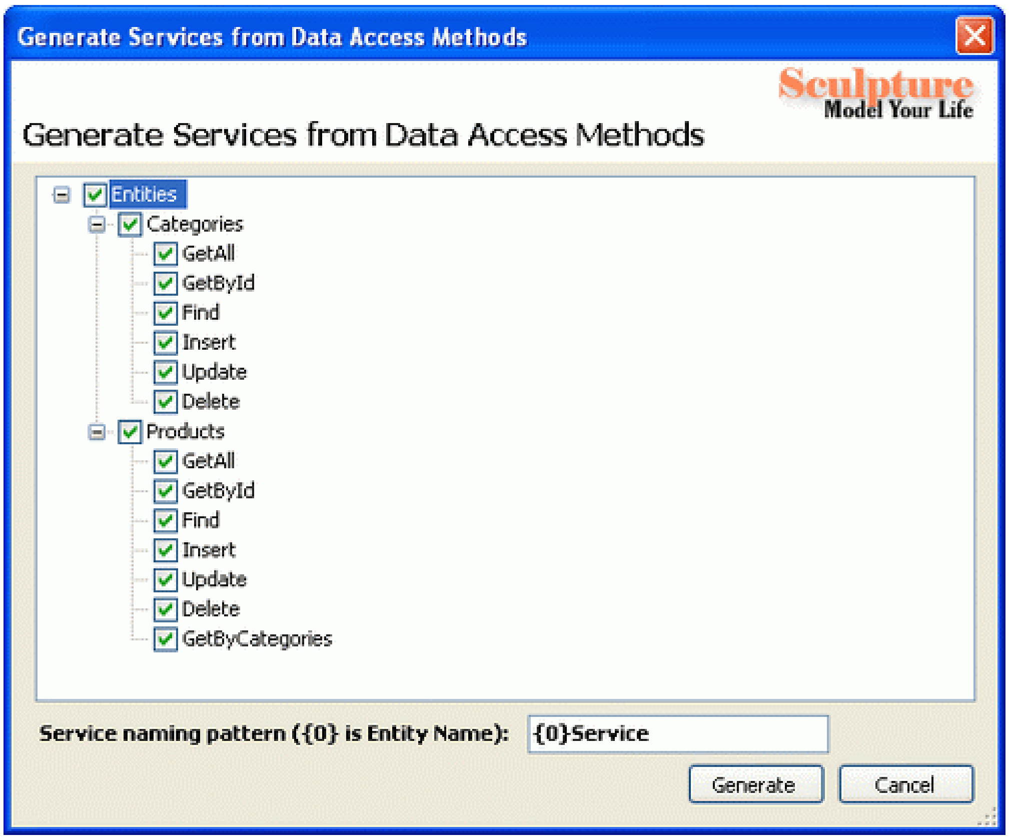
Association Properties:

Entity Members	Data Contract Members
CategoryID	CategoryID
CategoryName	CategoryName
Description	Description
Picture	Picture

- If you choose one of the data contracts, you can edit its members from the Data Contract tab in the Sculpture Details window.

4-3. Designing the services

- The final step is designing your services; you can simply drag and drop a service from the tool box, then add operations to this service.
- Instead of creating a service for each entity manually, that holds the default and custom data access methods, Sculpture provides an option to generate services from entities automatically.
    - Press 'Generate services from entities' in the Sculpture editor tool bar.



Generate Services from Data Access Methods

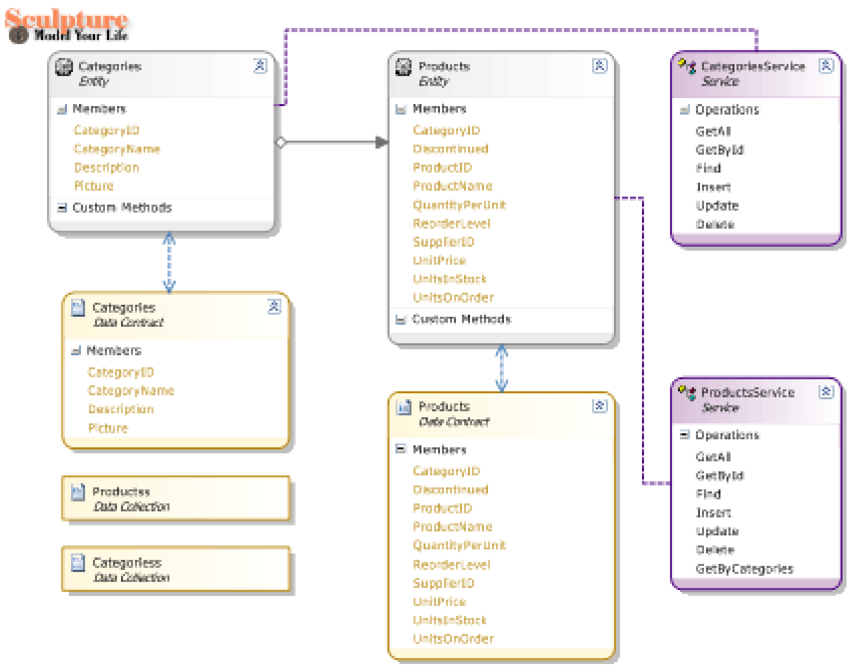
Generate Services from Data Access Methods

```
☑ Entities
  ☑ Categories
    ☑ GetAll
    ☑ GetById
    ☑ Find
    ☑ Insert
    ☑ Update
    ☑ Delete
  ☑ Products
    ☑ GetAll
    ☑ GetById
    ☑ Find
    ☑ Insert
    ☑ Update
    ☑ Delete
    ☑ GetByCategories
```

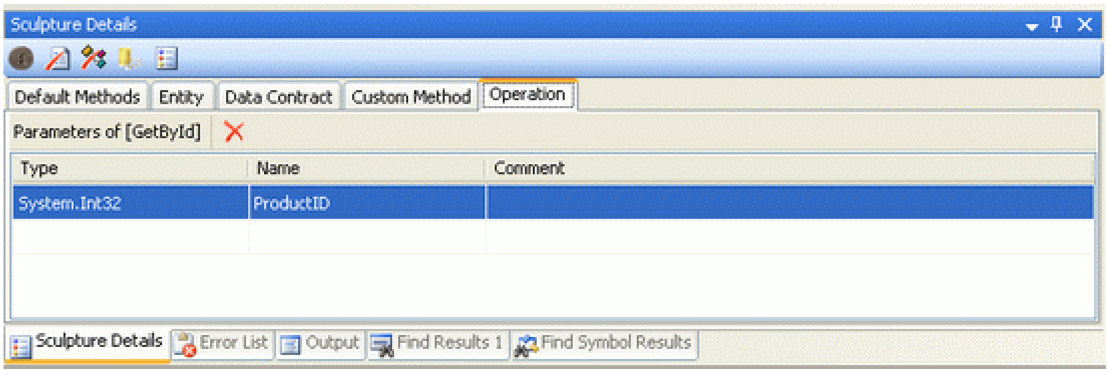
Service naming pattern ({0} is Entity Name): {0}Service

- You will notice that each entity has its default methods and custom methods.
- You have the option to specify the service name.
- Choose all the entities in the tree, then press Generate.

- You will find new services added to the model, and there are relations between the entity and the corresponding service; this relation means that the generator will define an instance of the repository class in this service.



- If you choose one of the service operations, you can edit its parameters from the Operation tab in the Sculpture Details window.

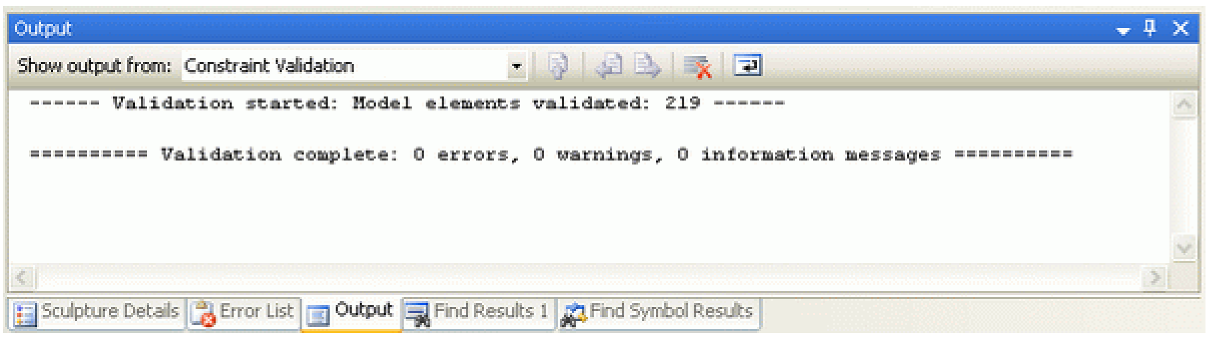


Parameters of [GetById]

Type	Name	Comment
System.Int32	ProductID	

4-4. Validating the model

- Since you have completed the designing phase, you need to validate your model. Sculpture provides validation rules for the most common errors found in the model. If you need custom validation rules, you can create your own Mold as discussed in part 2.
- You can validate your model by right clicking on the diagram surface and choosing *Validate All*; the output window must be as in the next figure.



Show output from: Constraint Validation

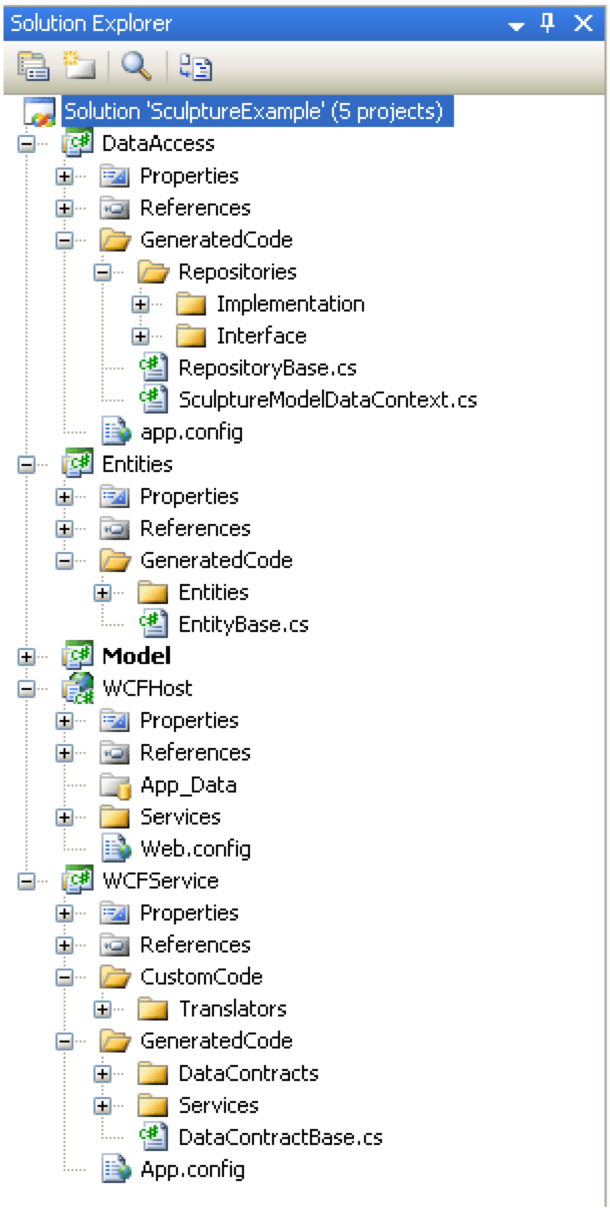
```
------ Validation started: Model elements validated: 219 ------

========== Validation complete: 0 errors, 0 warnings, 0 information messages ==========
```

5. Generate your code

It's time to produce deployable components from our model. As we mentioned earlier, we start with the LINQ Mold for the data access layer and the WCF Mold for the service layer.

The generation process is so simple, just right click on the diagram surface, and click ‘Generate’. Sculpture will generate the code and attach it to the corresponding project. After generation, the Solution Explorer must be like in the next figure:


```
Solution Explorer
Solution 'SculptureExample' (5 projects)
  DataAccess
    Properties
    References
    GeneratedCode
      Repositories
        Implementation
        Interface
      RepositoryBase.cs
      SculptureModelDataContext.cs
    app.config
  Entities
    Properties
    References
    GeneratedCode
      Entities
      EntityBase.cs
  Model
  WCFHost
    Properties
    References
    App_Data
    Services
    Web.config
  WCFService
    Properties
    References
    CustomCode
      Translators
    GeneratedCode
      DataContracts
      Services
      DataContractBase.cs
    App.config
```

Let’s explore the generated code quickly:

- Entities Project: contains the entities that are attached with the LINQ attributes, and the ‘Entity Base’ which is the base for all entities.
- Data Access Project: contains the repository classes and its interfaces, with the ‘Repository Base’ class, and the ‘Data Context’ class. It also contains a configuration file that holds the connection string, and a project setting file that refers to this connection string. Additionally, it contains a SQL folder that holds two SQL files, one for the database schema, and another for the CRUD stored procedures.
- WCF Service Project: contains a custom folder for the translators classes. These classes can be used directly for converting to and from entities and data contracts, and contains a *DataContracts* folder that holds the data contracts and data collections, and contains the service implementations and service contracts (Service Contract = the interface of this service implementation class), and also contains a configuration file that contains information about the WCF services.
- WCF Host Project: contains SVC files for our WCF services; also contains a web configuration file holding information about these services.

6. Test your application

Before testing the application, we need to add the missing references and add our custom code:

- Adding the missing references to these projects:

- In the Entities project: add a reference to `System.Data.Linq`.
- In the Data Access project: add a reference to the Entities project, and a reference to `System.Data.Linq`.
- In the WCF Service project: add a reference to the Entities project and the Data Access project.
- In the WCF Host project: add a reference to the WCF Service project.

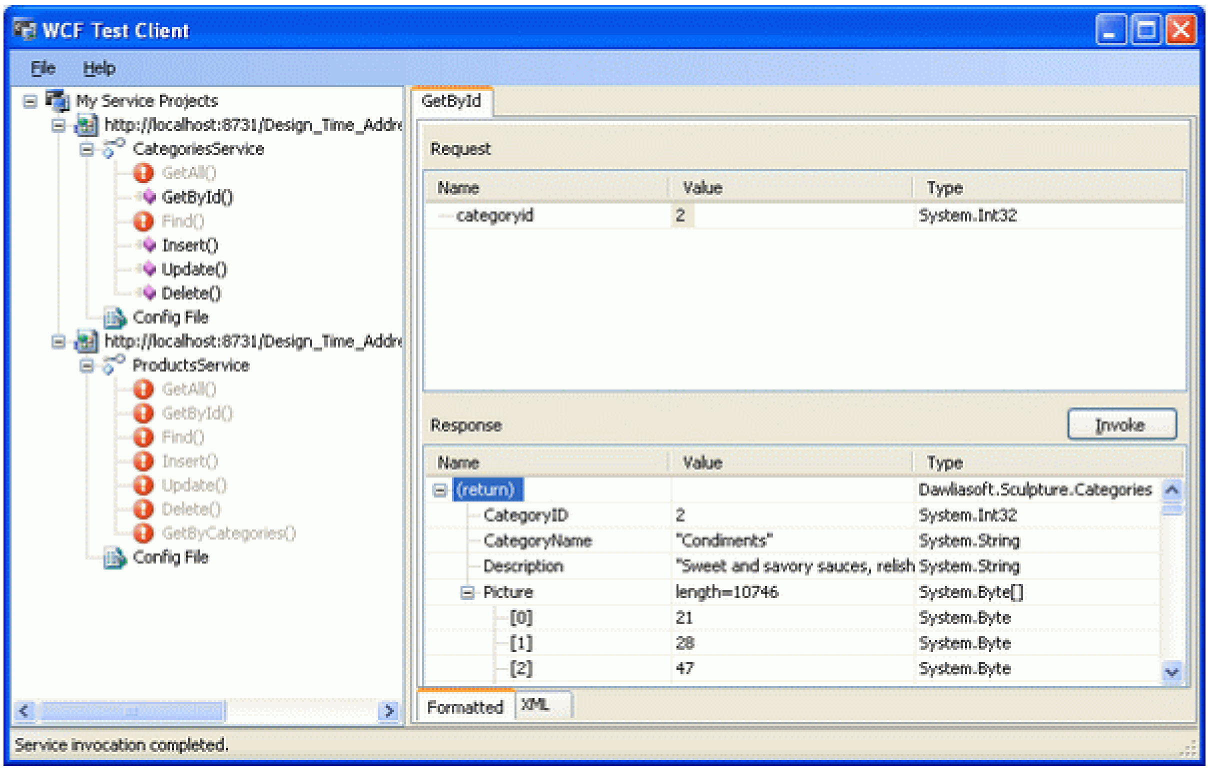
- Adding custom code. In this example, we will add the implementation to the get-by-ID operation in `CategoriesService`:
    - In the custom folder in the WCF Service project, add a folder called '*Services*'.
    - Add a new class with the name '*CategoriesService.cs*'.
    - Add the following code to this class:

Hide Copy Code

```
namespace WCFService
{
    public partial class CategoriesService
    {
        public override Categories GetById(int categoryid)
        {
            return CategoriesTranslator.ToDataContract(
                        categoriesRepository.GetById(categoryid));
        }
    }
}
```

It's time to test our project, set the WCF Service project as the startup project and run the application.

- Set the WCF Service project as the startup project.
- Build and run the application.
- In the WCF test client, invoke the operation get-by-ID in `CategoriesService`, and enjoy the result.


WCF Test Client

File Help

- My Service Projects
  - http://localhost:8731/Design_Time_Addre
    - CategoriesService
      - GetAll()
      - GetById()
      - Find()
      - Insert()
      - Update()
      - Delete()
    - Config File
  - http://localhost:8731/Design_Time_Addre
    - ProductsService
      - GetAll()
      - GetById()
      - Find()
      - Insert()
      - Update()
      - Delete()
      - GetByCategories()
    - Config File

GetById

Request

Name	Value	Type
categoryid	2	System.Int32

Response Invoke

Name	Value	Type
(return)		Dawliasoft.Sculpture.Categories
CategoryID	2	System.Int32
CategoryName	"Condiments"	System.String
Description	"Sweet and savory sauces, relish	System.String
Picture	length=10746	System.Byte[]
[0]	21	System.Byte
[1]	28	System.Byte
[2]	47	System.Byte

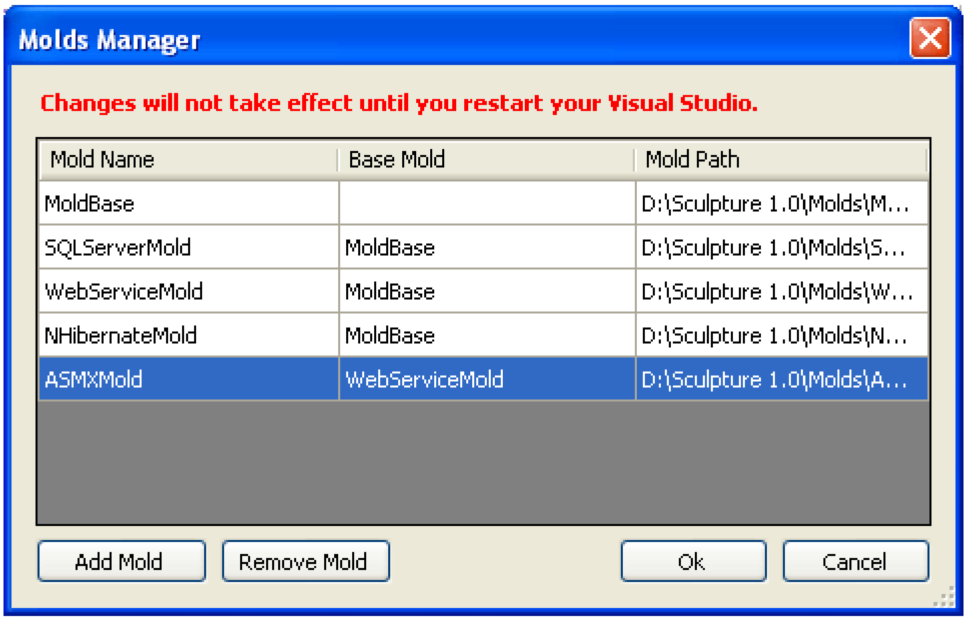
Formatted XML

Service invocation completed.

7. Switch your technologies

As we mentioned before, one of the major advantages of the Model Driven Development approach is raising the level of abstraction, so your problem is not related to its implementation technology. In this section, I will show you how you can switch your implementation technology from LINQ and WCF to NHibernate and ASMX web services.

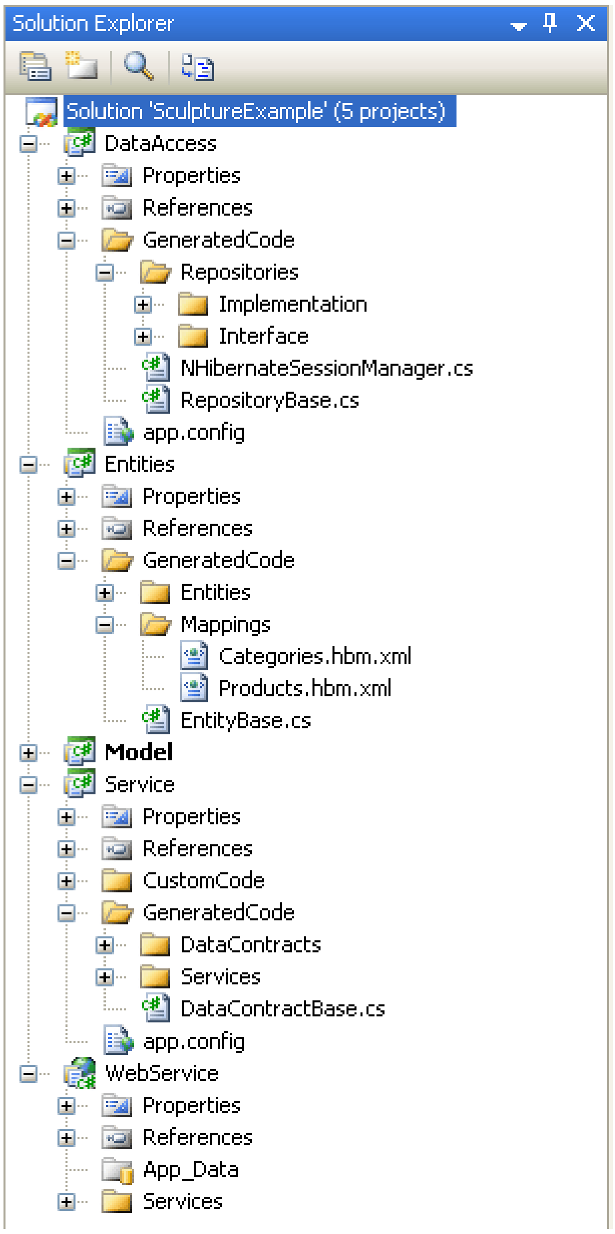
- From the Molds Manager, remove the LINQ and WCF Mold, and add an NHibernate and ASMX Mold.



Changes will not take effect until you restart your Visual Studio.

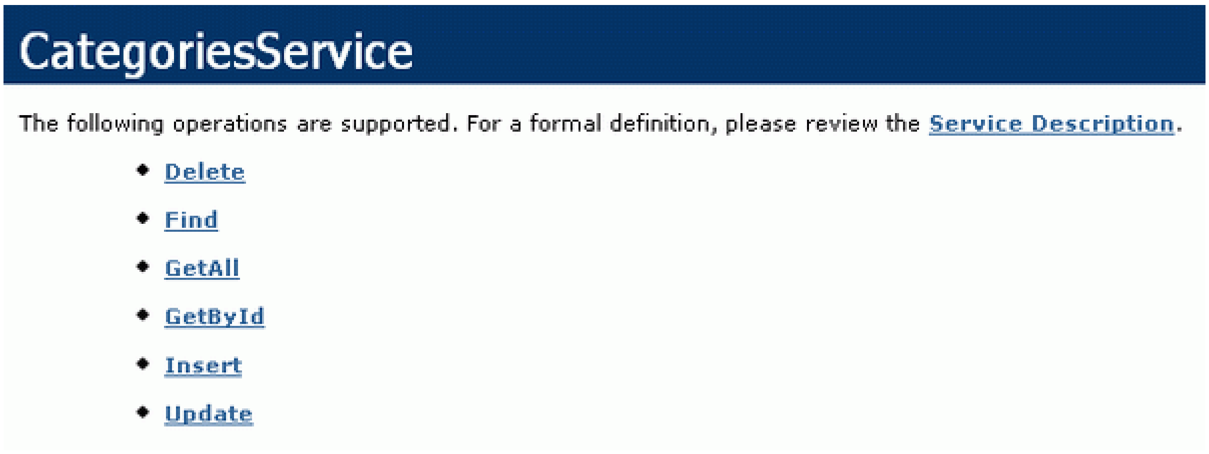
Mold Name	Base Mold	Mold Path
MoldBase		D:\Sculpture 1.0\Molds\M...
SQLServerMold	MoldBase	D:\Sculpture 1.0\Molds\S...
WebServiceMold	MoldBase	D:\Sculpture 1.0\Molds\W...
NHibernateMold	MoldBase	D:\Sculpture 1.0\Molds\N...
ASMXMold	WebServiceMold	D:\Sculpture 1.0\Molds\A...

- Restart your Visual Studio.
- Add a new Class Library project called 'Service' to the solution.
- Add a new ASP.NET Web Service Application project called 'WebService' to the solution.
- Remove all the files from the last added projects.
- Copy `CategoriesService` we created in the WCF service project to the new 'Service' project (do not forget to rename the namespace from '`WCFService`' to '`Service`').
- Remove the WCF service project and WCF host project from the solution.
- Remove the '*Generated Code*' folder from Entities project and Data Access project.
- Return to the model, and in the project responsibilities, specify the Service project instead of the WCF Service project, and the Web Service project instead of the WCF Host project.
- Save the model, you will notice some errors about the order of data contract members. You can solve it by pressing the 'Order Data Members' button in the Sculpture details window, this will reorder all the data members in the model.
- Validate the application and ensure that no errors exist, and then generate the code.
- The Solution Explorer must be as in this next figure:



```
Solution Explorer
Solution 'SculptureExample' (5 projects)
  DataAccess
    Properties
    References
    GeneratedCode
      Repositories
        Implementation
        Interface
      NHibernateSessionManager.cs
      RepositoryBase.cs
    app.config
  Entities
    Properties
    References
    GeneratedCode
      Entities
      Mappings
        Categories.hbm.xml
        Products.hbm.xml
      EntityBase.cs
  Model
  Service
    Properties
    References
    CustomCode
    GeneratedCode
      DataContracts
      Services
      DataContractBase.cs
    app.config
  WebService
    Properties
    References
    App_Data
    Services
```

- In the '*mappings*' folder, select all the HBM files and choose 'Embedded Resource' in its Build Action property.
- Adding the missing references to these projects:
  - In the Data Access project: add a reference to *nhibernate.dll* and `System.Web`.
  - In the Service project: add a reference to the Entities project, Data Access project, `System.Web.Services`, and `System.EnterpriseServices`.
  - In the Web Service project: add a reference to the Service project and the NHibernate DLLs (*log4net.dll*, *Iesi.Collections.dll*, *Castle.DynamicProxy.dll*).
- Move the '*app.config*' file from the Service project to the Web Service project and rename it to '*web.config*', where this file holds the NHibernate configurations and the connection string.
- Build the solution and right click on '*CategoriesService.asmx*' in the Web Service project, and choose View in browser.



CategoriesService

The following operations are supported. For a formal definition, please review the Service Description.

- Delete
- Find
- GetAll
- GetById
- Insert
- Update

- Invoke the operation '`GetById`' to see the same result as LINQ+WCF without writing any additional code.
- You might try LINQ+ASMX or NHibernate+WCF!!!

Points of interest

- See this article in Screencast
- Sculpture Home Page
- Sculpture online documentation

Conclusion

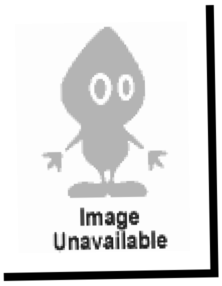
Model-Driven Development represents the next logical step in software development methods and practices. It aims at facilitating the automatic construction of a software solution from a high-level domain-specific specification. This approach seeks to promote productivity, maintainability, expressiveness, and to aid in the management of complexity by supporting higher levels of abstraction and the systematic reuse of domain-specific assets. *(From: Model-Driven Development of .NET Enterprise Applications)*. Sculpture (Model your Life) is a new implementation in this track, which you can use for modeling business entities, data contracts, and services. Based on this model, Sculpture transports it to your favorite technology. Sculpture is a pluggable engine so you can plug your custom code generator.

License

Share

EMAIL

About the Author



Ahmed Negm

Chief Technology Officer www.Dawliasoft.com

Egypt

Program Manager in Sculpture project, Interesting in .NET Model driven development.

You may also be interested in...

Introduction to Model Driven Development with AndroMDA (Part 1)

The Best Angular 2 Data Grid: FlexGrid

Introduction to Model Driven Development with AndroMDA (Part 2)

Smart Glasses to Help the Blind, With Pivothead LiveModPro and Intel Edison

Driver Development Part 3: Introduction to driver contexts

Using Internet-Scale data to guide product planning

Comments and Discussions

Add a Comment or Question

Search Comments Go

First Prev Next

Responsibility for Web Service new 3
Henk Meijerink 6-Dec-08 19:19

Thanks Negm new 5
Akrumooz 15-Sep-08 5:51

Mold new 3
Member 4533409 12-Sep-08 23:02

GREAT! new 3
pilates_elates 9-Sep-08 2:36

Mold? new 4
Peter Hayward 8-Sep-08 21:23

Well done new 4
Ralph Willgoss 4-Sep-08 18:42

Nice one. new 2
Pete O'Hanlon 3-Sep-08 14:44

Refresh 1

General News Suggestion Question Bug Answer Joke Praise Rant Admin