EntitySpaces 2009 WCF Demo

Table of Contents

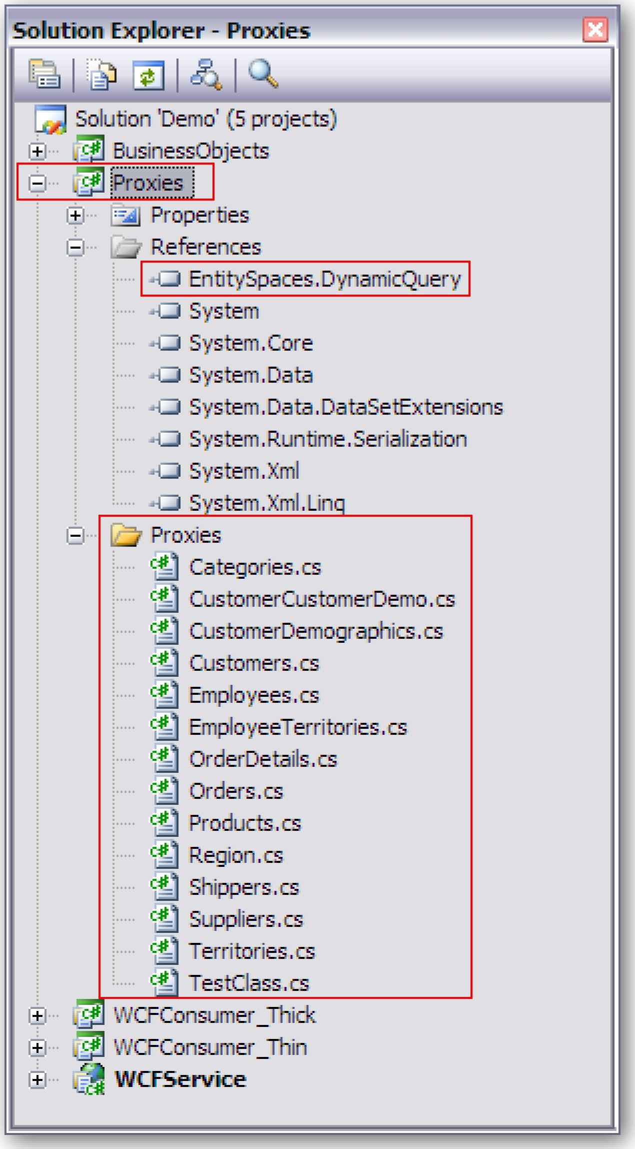
About the WCF Demo

The WCF Demo contains a single WCF Service that is accessed by what we call a thick client (that uses our full business objects on both sides of the conversation) and a thin client that uses our lightweight proxies on the client side and our full business objects on the server side. In both of these examples (the thick and thin client) we avoid using Visual Studio generated proxy classes when we add our Service Reference. Of course, you can use the Visual Studio generated service reference classes but our proxies are much more powerful to use.

You could for instance, provide your clients with something similar to our thin client example and then consuming your WCF Reference would be very easy for them. IF you have any questions just register on our forums and ask.

- The EntitySpaces Team

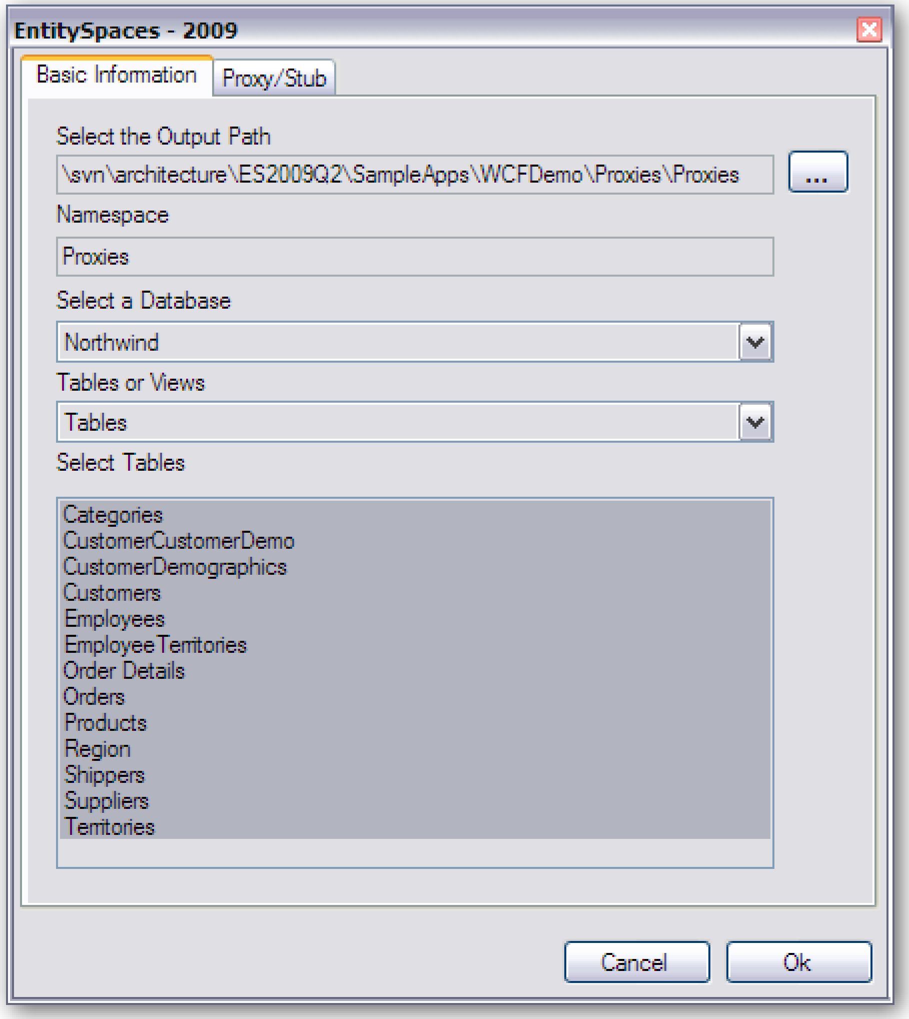
How We Generated the Lightweight Client Side Proxies


Solution Explorer - Proxies

```
Solution 'Demo' (5 projects)
+ BusinessObjects
- Proxies
  + Properties
  - References
      EntitySpaces.DynamicQuery
      System
      System.Core
      System.Data
      System.Data.DataSetExtensions
      System.Runtime.Serialization
      System.Xml
      System.Xml.Linq
  - Proxies
      Categories.cs
      CustomerCustomerDemo.cs
      CustomerDemographics.cs
      Customers.cs
      Employees.cs
      EmployeeTerritories.cs
      OrderDetails.cs
      Orders.cs
      Products.cs
      Region.cs
      Shippers.cs
      Suppliers.cs
      Territories.cs
      TestClass.cs
+ WCFConsumer_Thick
+ WCFConsumer_Thin
+ WCFService
```

This is how we generated the "*Proxies"* Assembly. This must be created as a class library when you choose "New Project". The reason we put the lightweight proxies in their own class library is so that we can choose the "Proxies" class library when we add the service reference, this will be shown later in this document.

Notice that we include the EntitySpaces.DynamicQuery assembly in our refernecees. This assembly allows you to do full EntitySpaces DynamicQueries in your client side application. You actually serialize the query and send it to the server to execute. Obviously you do not access your database directly from your client side application. This is the only EntitySpaces assembly necessary for the thin client version and it's only around 60k in size. You can even choose not to use the DynamicQuery feature and avoid needed to use any EntitySpaces class libraries. However, the DynamicQuery feature is very nice.



EntitySpaces - 2009

Basic Information | Proxy/Stub

Select the Output Path: \svn\architecture\ES2009Q2\SampleApps\WCFDemo\Proxies\Proxies

Namespace: Proxies

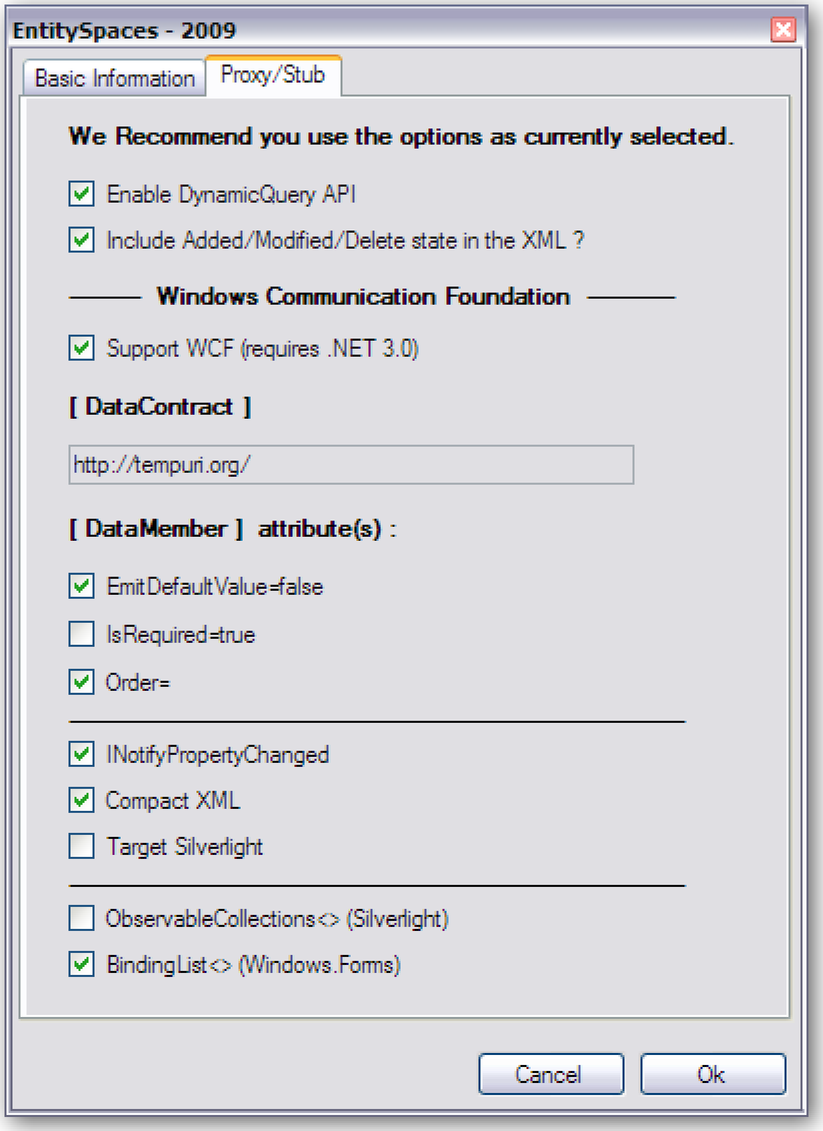
Select a Database: Northwind

Tables or Views: Tables

Select Tables:

Categories
CustomerCustomerDemo
CustomerDemographics
Customers
Employees
EmployeeTerritories
Order Details
Orders
Products
Region
Shippers
Suppliers
Territories

There wasn't much to do here on the "Basic Information" Tab. We merely selected all of the tables in the Northwind database and we used "Proxies" as our namespace. It is important that the proxies be in their own class library. See the next page for the interesting UI choices …


EntitySpaces - 2009

Basic Information | Proxy/Stub

We Recommend you use the options as currently selected.

- [✓] Enable DynamicQuery API
- [✓] Include Added/Modified/Delete state in the XML ?

Windows Communication Foundation

- [✓] Support WCF (requires .NET 3.0)

[DataContract]

http://tempuri.org/

[DataMember] attribute(s) :

- [✓] EmitDefaultValue=false
- [ ] IsRequired=true
- [✓] Order=

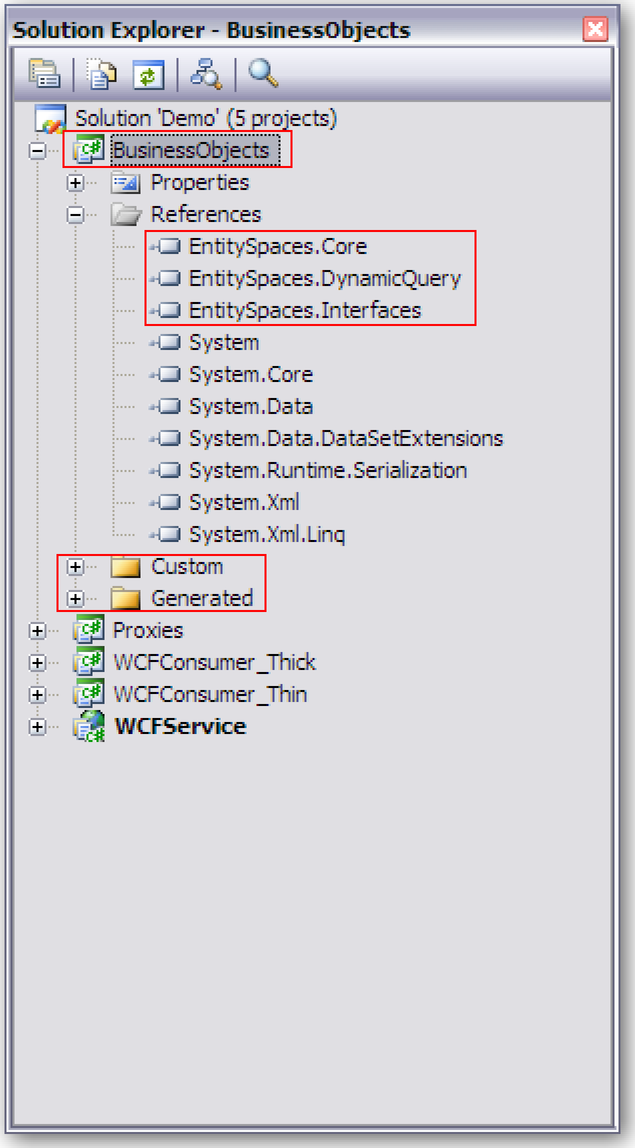
- [✓] INotifyPropertyChanged
- [✓] Compact XML
- [ ] Target Silverlight

- [ ] ObservableCollections<> (Silverlight)
- [✓] BindingList<> (Windows.Forms)

These are the choices we recommend for WCF development on the lightweight Proxy/Stub tab. That's it, merely generate the classes, include them in your "Proxies" class library, add a reference to the EntitySpaces.DynamicQuery assembly which you will find in the C:\Program Files\EntitySpaces 2009\Runtimes\.NET 3.5 folder and you're off and running.

How We Generated the Main Business Objects

These objects are housed in our "BusinessObjects" class library. These are your normal full business objects that you would use in any normal EntitySpaces Application.



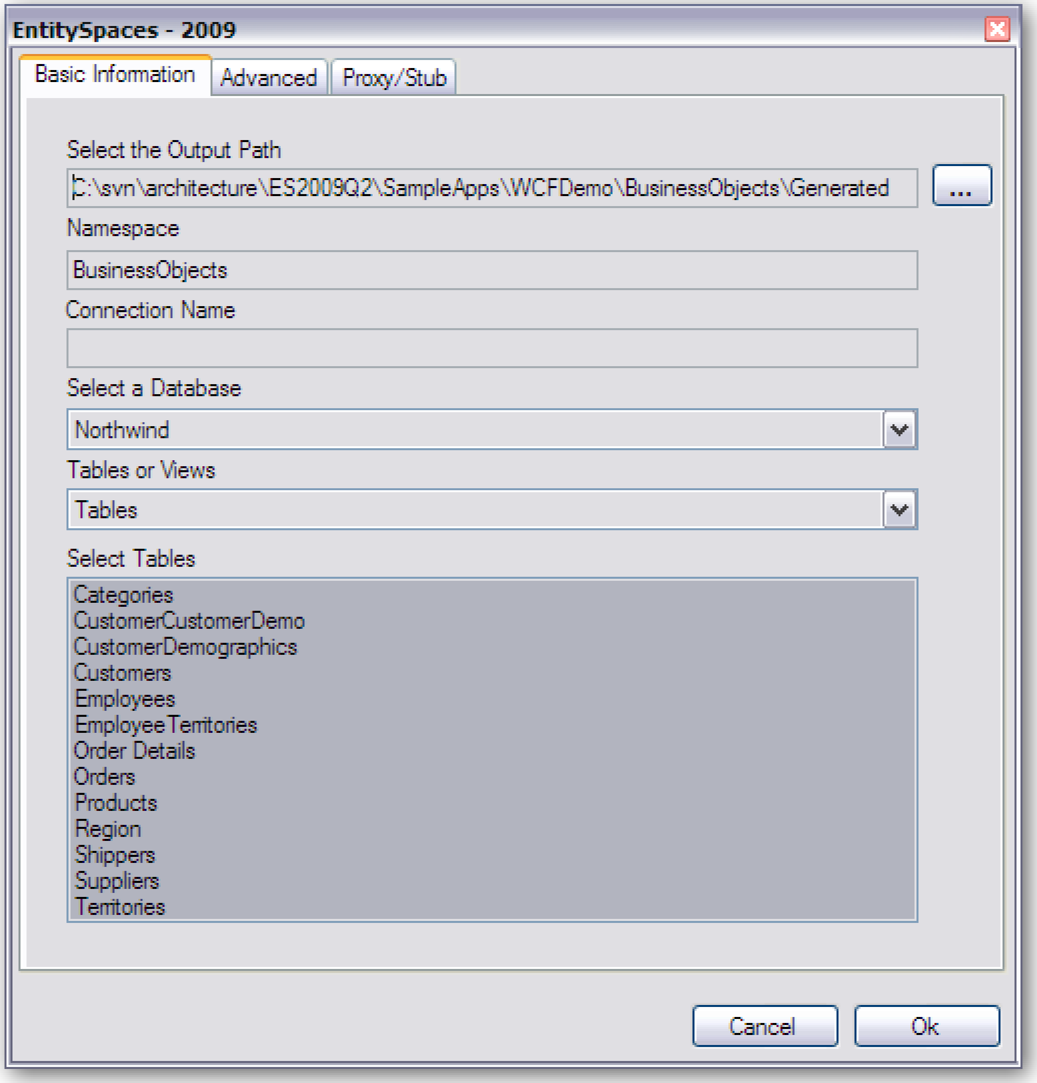
Solution Explorer - BusinessObjects

Solution 'Demo' (5 projects)
- BusinessObjects
  - Properties
  - References
    - EntitySpaces.Core
    - EntitySpaces.DynamicQuery
    - EntitySpaces.Interfaces
    - System
    - System.Core
    - System.Data
    - System.Data.DataSetExtensions
    - System.Runtime.Serialization
    - System.Xml
    - System.Xml.Linq
  - Custom
  - Generated
- Proxies
- WCFConsumer_Thick
- WCFConsumer_Thin
- **WCFService**

Since this class library will house our full business objects we add the three main references shown on the left. Also, we create our "Custom" and "Generated" classes and include them in this class library.

The creation of the generated classes is shown on the following pages.

First we merely select all of the tables in our Northwind Database.



Select the Output Path
C:\svn\architecture\ES2009Q2\SampleApps\WCFDemo\BusinessObjects\Generated

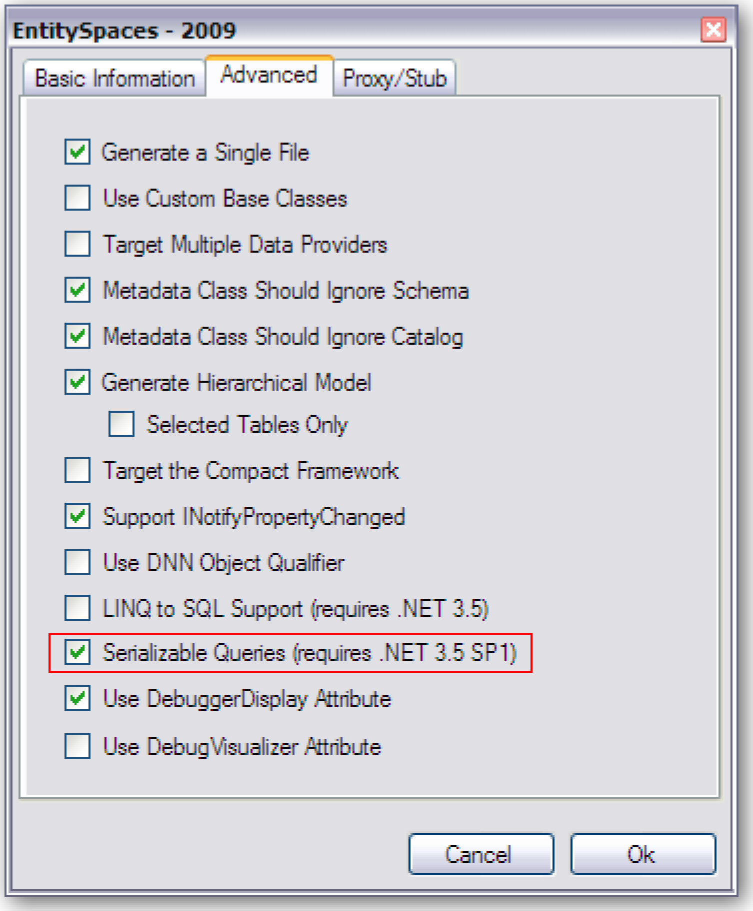
Namespace
BusinessObjects

Connection Name

Select a Database
Northwind

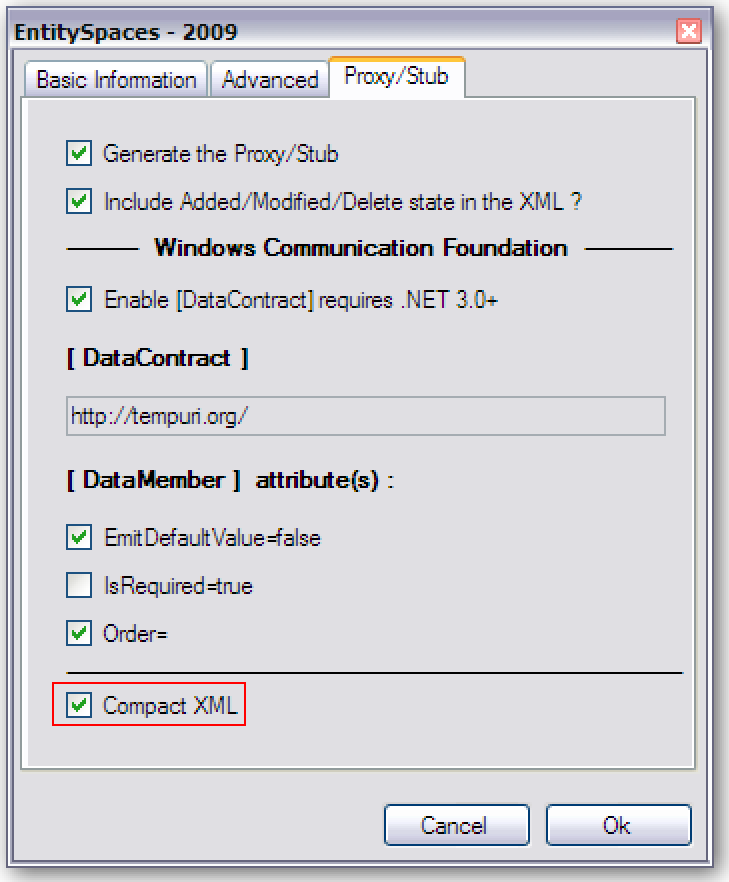
Tables or Views
Tables

Select Tables
Categories
CustomerCustomerDemo
CustomerDemographics
Customers
Employees
EmployeeTerritories
Order Details
Orders
Products
Region
Shippers
Suppliers
Territories


EntitySpaces - 2009

- [✓] Generate a Single File
- [ ] Use Custom Base Classes
- [ ] Target Multiple Data Providers
- [✓] Metadata Class Should Ignore Schema
- [✓] Metadata Class Should Ignore Catalog
- [✓] Generate Hierarchical Model
  - [ ] Selected Tables Only
- [ ] Target the Compact Framework
- [✓] Support INotifyPropertyChanged
- [ ] Use DNN Object Qualifier
- [ ] LINQ to SQL Support (requires .NET 3.5)
- [✓] Serializable Queries (requires .NET 3.5 SP1)
- [✓] Use DebuggerDisplay Attribute
- [ ] Use DebugVisualizer Attribute

The key things to notice here is that we checked **Serializable Queries** which is necessary if we want to be able to deserialize the queries sent to us from our client Application. This is not required, you don't have to allow queries from the client side, but for this sample we wanted to show how this feature works.



[✓] Generate the Proxy/Stub

[✓] Include Added/Modified/Delete state in the XML ?

Windows Communication Foundation

[✓] Enable [DataContract] requires .NET 3.0+

[DataContract]

http://tempuri.org/

[DataMember] attribute(s) :

[✓] EmitDefaultValue=false

[ ] IsRequired=true

[✓] Order=

[✓] Compact XML

Notice here we make the same choices as we did when created the lightweight proxies. The important thing here is that we need to be sure to check the "Compact XML" checkbox too since we also made this choice for our lightweight proxies.

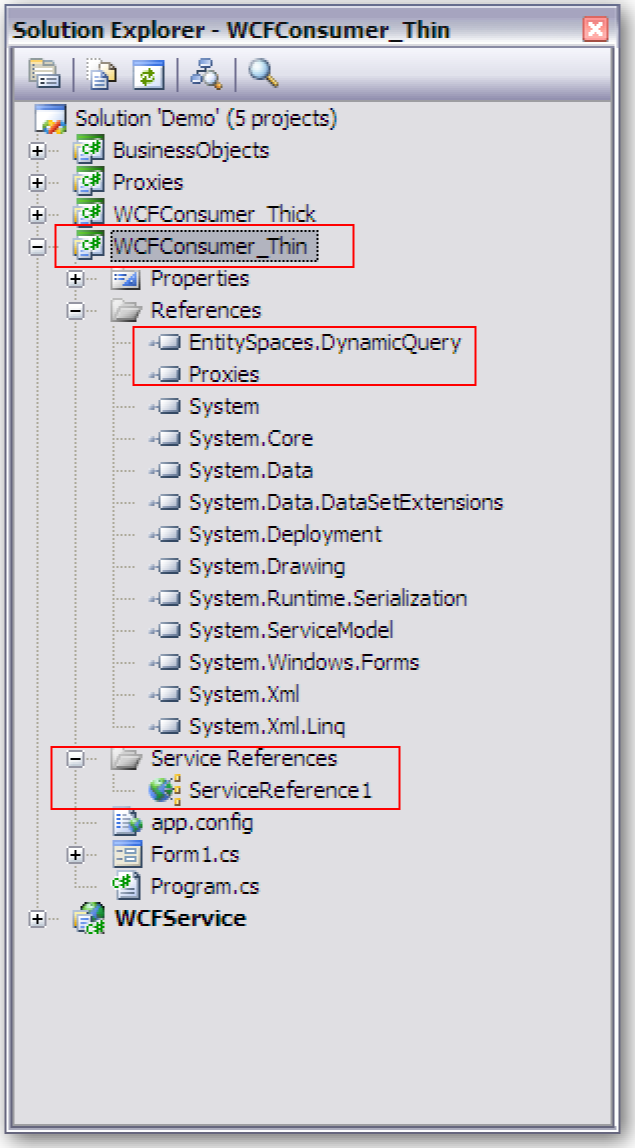
The WCFService

The WCF Service is a simple service that shows the basic mechanic of using EntitySpaces in WCF Scenarios. The WCF Service has the API shown below.

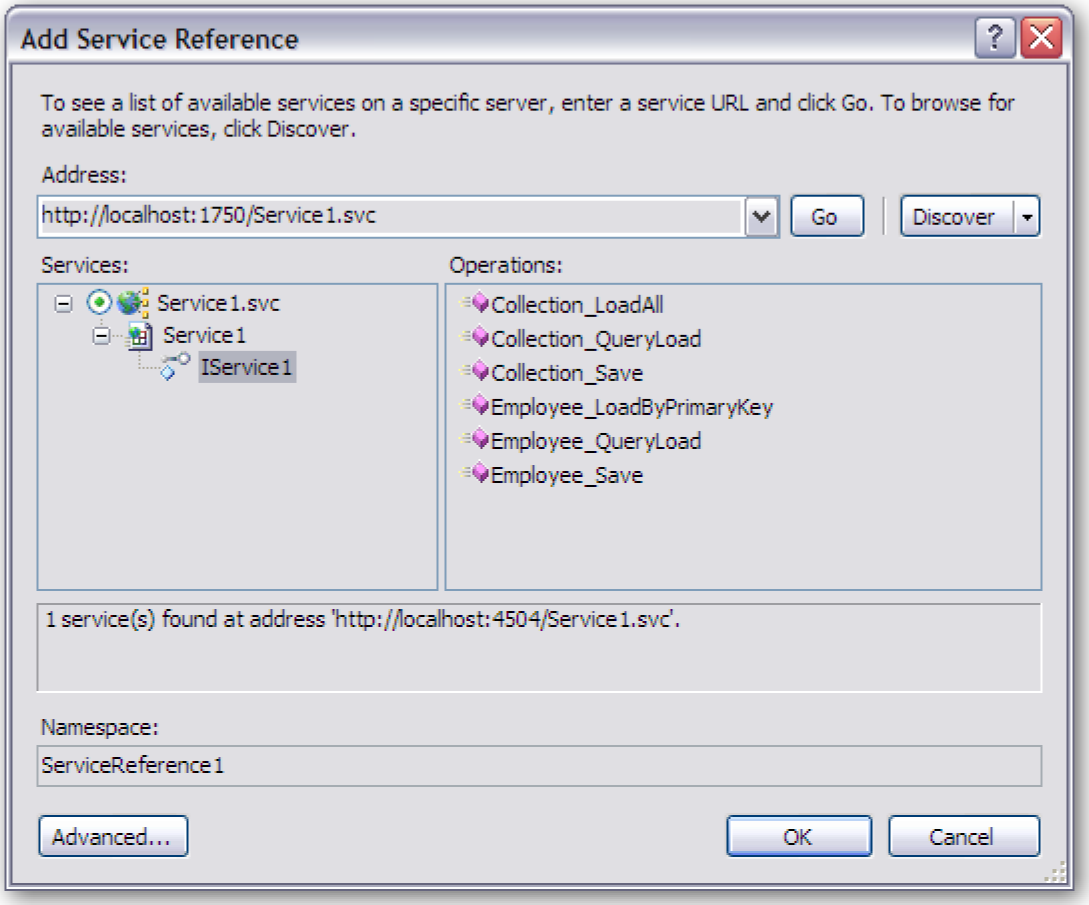
```
[ServiceContract]
public interface IService1
{
    //---------------------------------------
    // Single Entity Operations
    //---------------------------------------
    [OperationContract]
    EmployeesProxyStub Employee_LoadByPrimaryKey(int id);

    [OperationContract]
    EmployeesProxyStub Employee_QueryLoad(string query);

    [OperationContract]
    EmployeesProxyStub Employee_Save(EmployeesProxyStub emp);

    //---------------------------------------
    // Collection Operations
    //---------------------------------------
    [OperationContract]
    EmployeesCollectionProxyStub Collection_LoadAll();

    [OperationContract]
    EmployeesCollectionProxyStub Collection_QueryLoad(string query);

    [OperationContract]
    EmployeesCollectionProxyStub Collection_Save(EmployeesCollectionProxyStub emp);
}
```

How We Setup the Added the WCFConsumer_Thin Executable

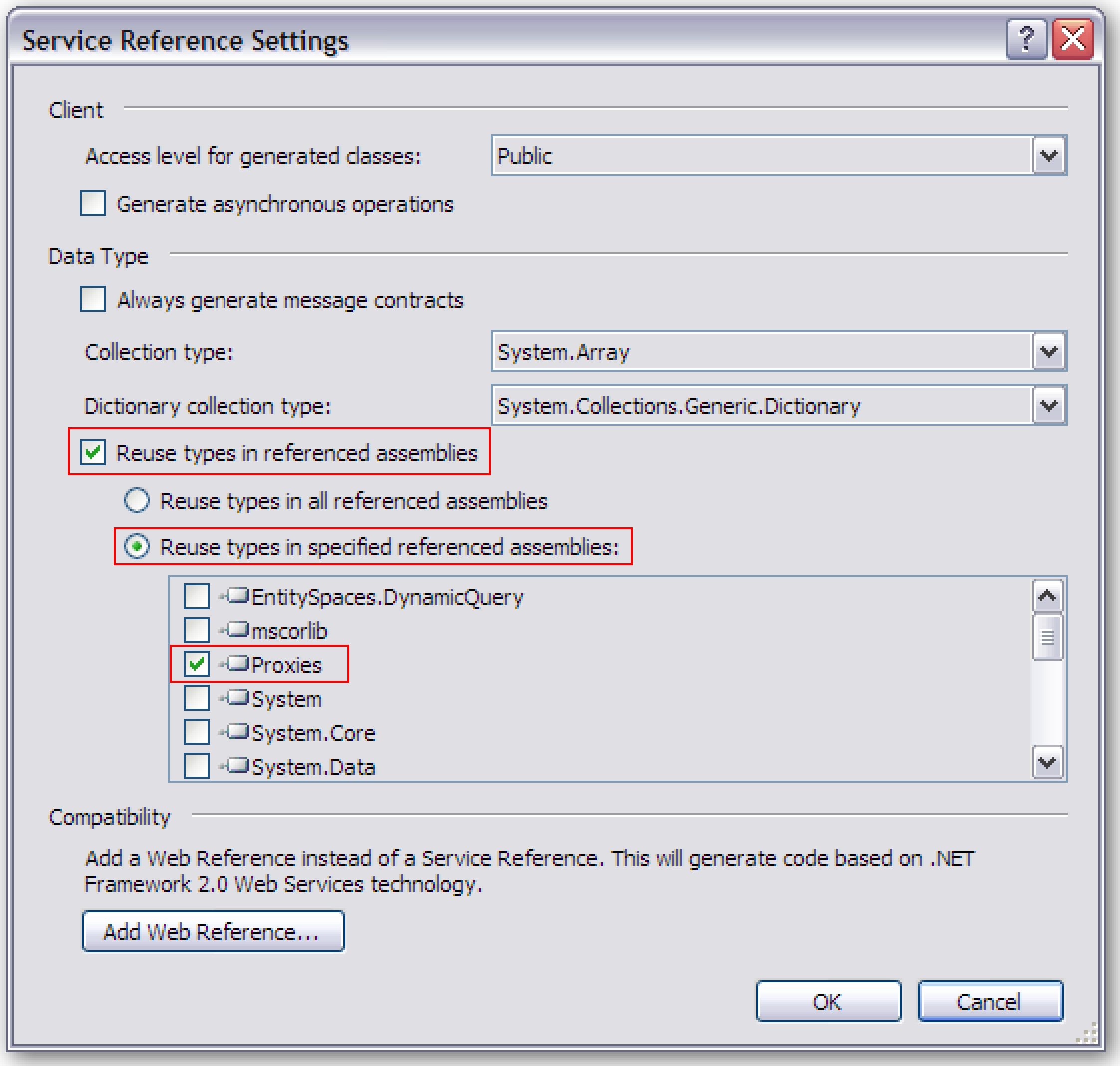
This is a very simple executable that uses the lightweight client side proxies on the client side. We include two references here, our "Proxies" class library and the EntitySpaces.DynamicQuery class library since we plan to use the serializable query feature.



We also needed to add a reference to our WCF Service. Let's take a look on the following two pages and see how we added our Service Reference.


Add Service Reference
To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.
Address:
http://localhost:1750/Service1.svc
Go
Discover
Services:
Service1.svc
Service1
IService1
Operations:
Collection_LoadAll
Collection_QueryLoad
Collection_Save
Employee_LoadByPrimaryKey
Employee_QueryLoad
Employee_Save
1 service(s) found at address 'http://localhost:4504/Service1.svc'.
Namespace:
ServiceReference1
Advanced...
OK
Cancel

The above is pretty standard, since our Service is in the same solution we can merely use the "Discover" button and it will automatically find our Northwind service. The next thing we need to do is let Visual Studio know we want it to use our Proxies and not the Visual studio generated proxies. To do this we must press the "Advanced" Button.



Service Reference Settings

Client

Access level for generated classes: Public

[ ] Generate asynchronous operations

Data Type

[ ] Always generate message contracts

Collection type: System.Array

Dictionary collection type: System.Collections.Generic.Dictionary

[✓] Reuse types in referenced assemblies

( ) Reuse types in all referenced assemblies

(●) Reuse types in specified referenced assemblies:

- [ ] EntitySpaces.DynamicQuery
- [ ] mscorlib
- [✓] Proxies
- [ ] System
- [ ] System.Core
- [ ] System.Data

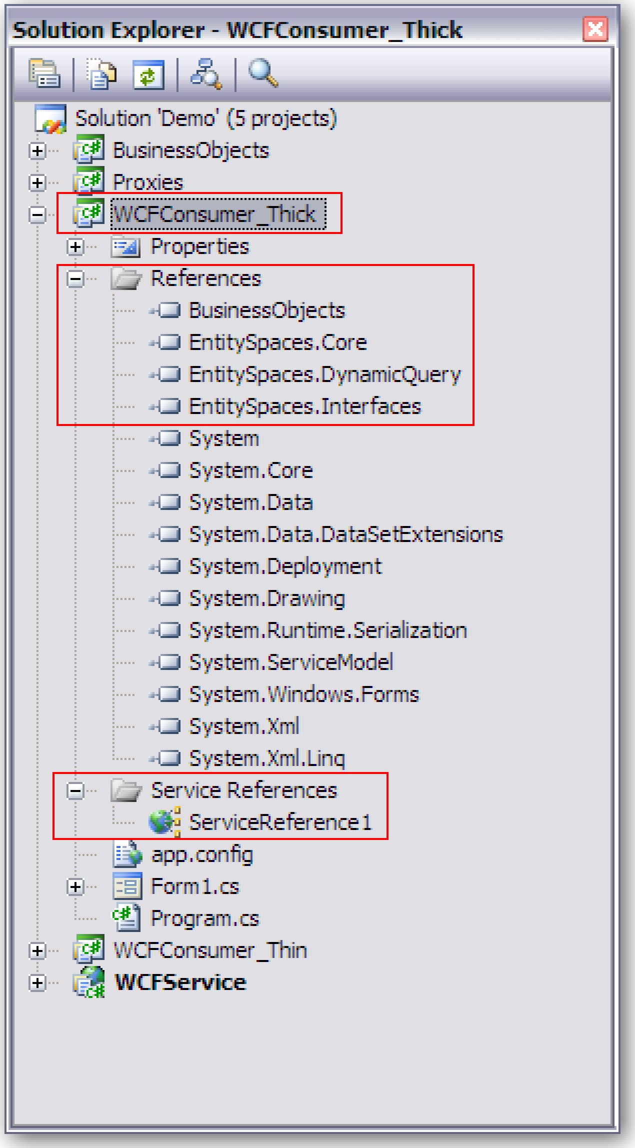
Compatibility

Add a Web Reference instead of a Service Reference. This will generate code based on .NET Framework 2.0 Web Services technology.

The importing thing to note here are the choices we made to indicate that we want to use the “Proxies” class library. That’s it, click “OK”

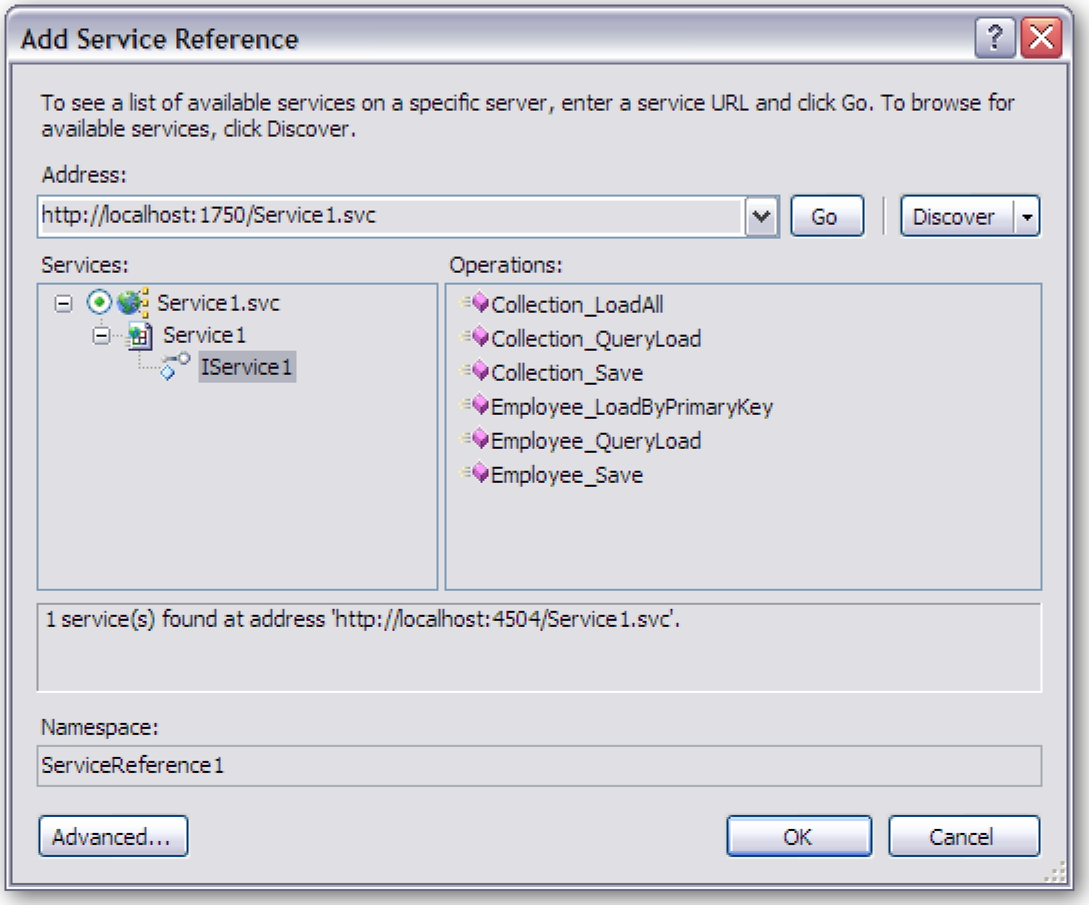
How We Setup the Added the WCFConsumer_Thick Executable

This is a very simple executable that use the lightweight client side proxies on the client side. We need to include the BusinessObjects class library since this is our thick client example. We are going to use our full business entities on both sides of the conversation. We also include the main three EntitySpaces class libraries as shown on the left.



```
Solution 'Demo' (5 projects)
+ BusinessObjects
+ Proxies
- WCFConsumer_Thick
  + Properties
  - References
      BusinessObjects
      EntitySpaces.Core
      EntitySpaces.DynamicQuery
      EntitySpaces.Interfaces
      System
      System.Core
      System.Data
      System.Data.DataSetExtensions
      System.Deployment
      System.Drawing
      System.Runtime.Serialization
      System.ServiceModel
      System.Windows.Forms
      System.Xml
      System.Xml.Linq
  - Service References
      ServiceReference1
    app.config
  + Form1.cs
    Program.cs
+ WCFConsumer_Thin
+ WCFService
```

We also needed to add a reference to our WCF Service. Let's take a look on the following two pages and see how we added our Service Reference.



To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

Address: http://localhost:1750/Service1.svc

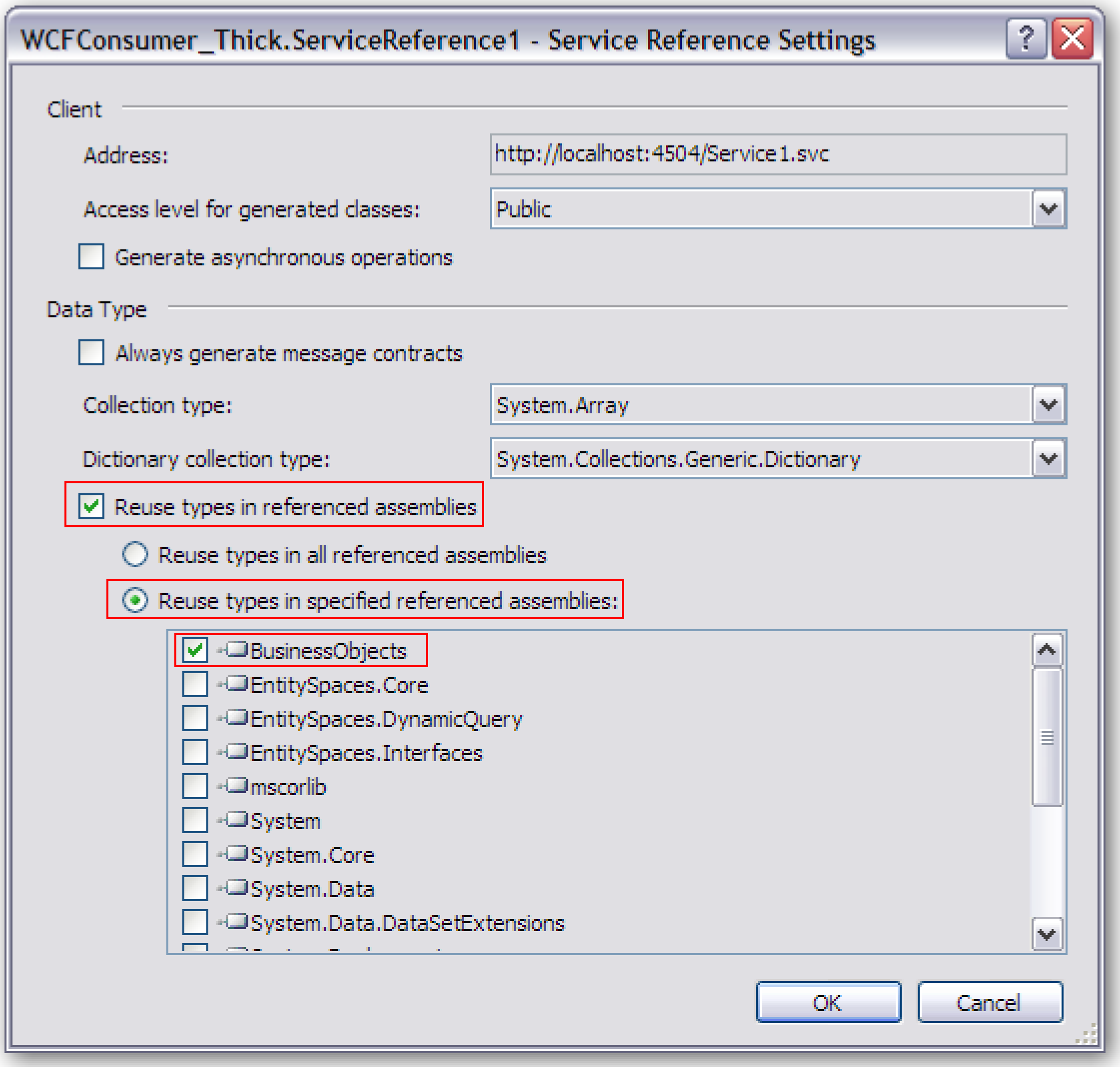
Services:
- Service1.svc
  - Service1
    - IService1

Operations:
- Collection_LoadAll
- Collection_QueryLoad
- Collection_Save
- Employee_LoadByPrimaryKey
- Employee_QueryLoad
- Employee_Save

1 service(s) found at address 'http://localhost:4504/Service1.svc'.

Namespace: ServiceReference1

The above is pretty standard, since our Service is in the same solution we can merely use the “Discover” button and it will automatically find our Northwind service. Then we changed the Namespace to “NorthwindClient”. The next thing we need to do is let Visual Studio know we want it to use our Proxies and not the Visual studio generated proxies. To do this we must press the “Advanced” Button.



WCFConsumer_Thick.ServiceReference1 - Service Reference Settings

Client

Address: http://localhost:4504/Service1.svc

Access level for generated classes: Public

☐ Generate asynchronous operations

Data Type

☐ Always generate message contracts

Collection type: System.Array

Dictionary collection type: System.Collections.Generic.Dictionary

☑ Reuse types in referenced assemblies

○ Reuse types in all referenced assemblies

● Reuse types in specified referenced assemblies:

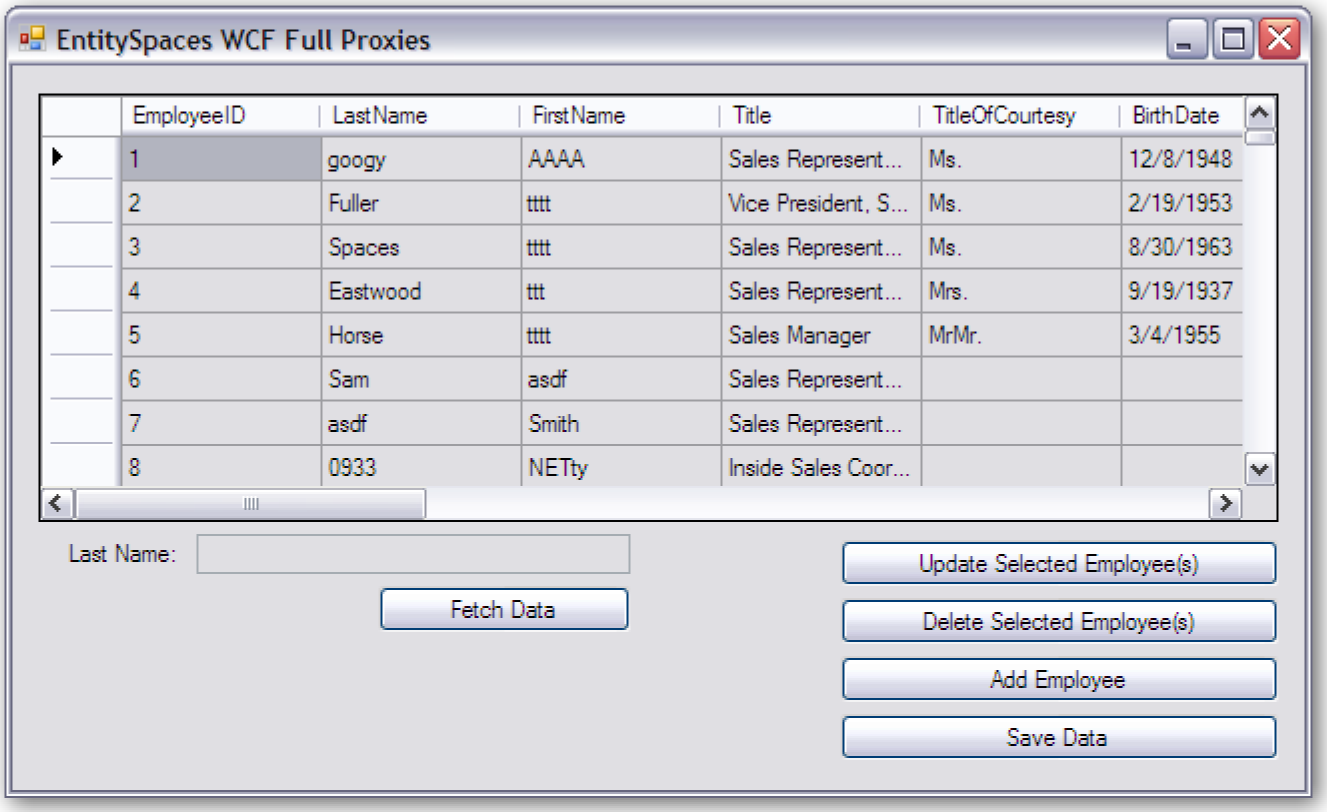
- ☑ BusinessObjects
- ☐ EntitySpaces.Core
- ☐ EntitySpaces.DynamicQuery
- ☐ EntitySpaces.Interfaces
- ☐ mscorlib
- ☐ System
- ☐ System.Core
- ☐ System.Data
- ☐ System.Data.DataSetExtensions

Notice that we set the collection type to “ObservableCollection” the same type that we chose when generating our lightweight proxy stubs. We also indicate that we want it to use the proxies in our EntitySpaces.Proxies.Silverlight assembly (that’s just what we happened to name our assembly that contains the lightweight proxies).

That’s it, click “OK”

Running the Demos

The first thing you need to do is make sure the connection string in the web.config file is correct for your Northwind database. The web..config file in in the WCFService project. Find the **connectionStrings** section in your web.config file and set it up appropriately.



EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	BirthDate
1	googy	AAAA	Sales Represent...	Ms.	12/8/1948
2	Fuller	tttt	Vice President, S...	Ms.	2/19/1953
3	Spaces	tttt	Sales Represent...	Ms.	8/30/1963
4	Eastwood	ttt	Sales Represent...	Mrs.	9/19/1937
5	Horse	tttt	Sales Manager	MrMr.	3/4/1955
6	Sam	asdf	Sales Represent...		
7	asdf	Smith	Sales Represent...		
8	0933	NETty	Inside Sales Coor...		

Last Name:

You can run either the Thick or Thin clients and add data, delete, or update data in the grid and then press the Update button. You can test the search feature and test the serializable DynamicQuery feature. The search feature searches on both FirstName and LastName fields.