

OpenAccess Made Easy

by Telerik Inc.

Welcome to Telerik OpenAccess ORM Made Easy.

We hope you enjoy the book as much as we, at Falafel Software, enjoyed creating it.

Telerik OpenAccess ORM

© 2010 Telerik Inc.

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: March 2010

Publisher

Falafel Software Inc.

Authors

Noel Rice

Technical Editors

Noel Rice

Cover Designer

Matt Kurvin

Team Coordinator

Lino Tadros

Production

Falafel Software Inc.

Special thanks to:

All the team members at Telerik worldwide for creating a magnificent piece of software in Telerik OpenAccess ORM. The authors also would like to thank the Falafel team members in Colorado, Texas and California for their feedback, guidance and recommendations on the subjects of the manual.

Falafel would like to thank Vassil Terziev and Svetozar Georgiev for their trust and belief in the quality of Falafel Software's work.

Falafel would like to thank Jan Blessenohl, Todd Anglin and Kevin Babcock for their support. We would also like to thank Stephen Forte for the use of his excellent white paper "Object Relational Mapping 101" included in the "Introduction" section of this manual.

Last but not least, thank you to all our families for their support and patience while we wrote the book.

Table of Contents

Foreword	5
Part I Introduction	8
1 Who Should Read This Courseware.....	8
2 What Do You Need to Have Before Reading This Courseware.....	8
3 How This Courseware Is Organized.....	8
4 About Telerik.....	9
5 About Falafel.....	10
6 Introducing Telerik OpenAccess ORM.....	10
Part II Getting Started	22
1 ORM Enable Project.....	22
2 Forward Mapping (objects > database).....	28
3 Reverse Mapping (database > objects).....	37
Generated Class	44
4 Create, Read, Update, Delete (CRUD).....	47
5 Wrap Up.....	51
Part III Design Environment	54
1 OpenAccess Menu	54
2 Project Context Menu.....	65
3 Wrap Up.....	66
Part IV Using OpenAccess in Applications	68
1 Building the "Model" Assembly.....	68
2 WinForms Example	69
3 ASP.NET Example	86
4 Telerik Reporting Example.....	95
5 Multi-Tier Architecture	103
6 Web Services Example.....	105
7 N-Tier Example.....	135
8 N-Tier With Business Rules.....	142
9 Wrap Up.....	152
Part V References	154
1 References.....	154
2 Self Referencing.....	161
3 Wrap Up.....	165

Part VI Inheritance	168
1 Inheritance Overview.....	168
2 Flat Mapping.....	169
3 Vertical Mapping.....	171
4 Mixed Flat and Vertical Mapping.....	172
5 Horizontal Mapping.....	176
6 Configuration.....	179
7 Mapping Walkthrough.....	182
8 Wrap Up.....	188
Part VII Transactions	192
1 Basics.....	192
2 ITransaction.....	194
3 TransactionProperties.....	195
4 Threading	200
5 Concurrency.....	206
6 Wrap Up.....	208
Part VIII Database Access	210
1 Using SQL with OpenAccess.....	210
LINQ	210
Object Query Language (OQL)	215
Native SQL	219
2 Stored Procedures.....	222
3 Wrap Up.....	230
Part IX Optimization	234
1 Fetch Plans.....	234
2 Caching.....	244
3 Wrap Up.....	254
Index	255

Foreword

It was always my dream that objects should persist themselves automatically. Getting the persistence layer working has been a fulltime job for many people going back to the first days of C++. My goal during the last decade was to make this process easy.

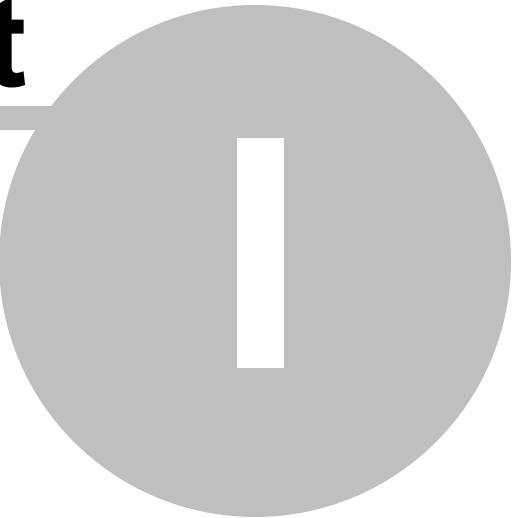
As Java and .NET introduced integrated memory management and better IDEs, new ideas about how objects could be persisted have popped up. Specifications from the Java world, like JDO and JPA, tried to define an object persistence layer that is easy to understand and to implement. But the presence of too many specifications and proprietary products has kept the process complicated. The .Net world defines comparable frameworks but these solutions are not proven or widely adopted.

With over 10 years of experience working with object persistence, the OpenAccess team is taking the best ideas in the industry to provide the perfect solution for .NET with seamless integration into Visual Studio using our many wizards and design tools.

You as the developer should not have to think about how objects persist themselves. As we get closer to making this dream real, we will work hard to make it easier for you to understand and implement data access for your projects. This courseware is an important part of learning to work with OpenAccess and discovering all it has to offer. I hope you have a lot of fun with this courseware and OpenAccess.

Jan Bleszenohl
Product Manager Telerik OpenAccess ORM

Part



I

Introduction

1 Introduction

1.1 Who Should Read This Courseware

The courseware assumes that you are familiar with either VB.NET or C# code. The courseware uses Visual Studio 2008 and assumes you know your way around this environment. You should be able to navigate the basic functional areas of the IDE (e.g. Solution Explorer, Properties, code/designer for windows forms, web pages etc.) and be able to run and debug applications and class libraries. You should have background working with databases, tables and stored procedures. Also you should be familiar with Microsoft mechanisms for working with data, i.e. connections, datasets, tables, etc.

1.2 What Do You Need to Have Before Reading This Courseware

Computer Setup

- Windows Vista Service Pack 2, Windows XP Professional .
- Microsoft .NET Framework 3.5.
- Microsoft Visual Studio 2008
- MS SQL Express 2005 or 2008 (Express 2005 should install along with Visual Studio 2008)
- Telerik OpenAccess ORM. You can purchase OpenAccess from:

<http://www.telerik.com/purchase/individual/orm.aspx>

or download the trial from:

<http://www.telerik.com/account/free-trials.aspx>

Learn more about System Requirements for Telerik OpenAccess ORM at:

<http://www.telerik.com/products/orm/resources/system-requirements.aspx>

Learn more about supported databases for Telerik OpenAccess ORM at:

<http://www.telerik.com/products/orm/resources/supported-databases.aspx>

1.3 How This Courseware Is Organized

Getting Started

This chapter demonstrates the basics for setting up your project to use OpenAccess ORM, transform .NET objects to database tables and back again. The chapter provides several walk-through examples to give you confidence using OpenAccess and a overall feel for how the product works.

Design Environment

This chapter is a quick tour of the available tools that make working with OpenAccess easy. The tour includes tools accessed from Visual Studio menus, context menus and wizards.

Using OpenAccess in Applications

This chapter shows how to integrate OpenAccess with a number of presentation platforms including ASP.NET, ASP.NET/AJAX, WinForms, Telerik Reporting and using web services. The chapter also explores some of the architectural possibilities for N-Tier applications that respect principles of multi-tiered application design.

References

This chapter explores how OpenAccess handles references between objects including one-to-one, one-to-many and many-to-many relationships. The chapter also shows how to handle self referential tables using OpenAccess.

Inheritance

This chapter demonstrates how OpenAccess handles inheritance to get the best mix of performance, data storage and conformity between the database and persistent objects. The chapter discusses the tradeoffs involved for different strategies used to implement inheritance. The chapter uses the well-worn "Animal/Dog/Breed" class example to make the point. The chapter finishes up with a walk-through example demonstrating the techniques discussed.

Transactions

This chapter discusses how OpenAccess preserves data integrity by making transactions available in your code. First, transaction basics are covered followed by a simple demonstration of a minimal transaction to show how this is achieved in code. Then the chapter explores helpful properties and methods of OpenAccess transaction objects. The chapter shows how to use OpenAccess transactions in multiple threads. Finally, the chapter discusses how concurrency options can best fit your environment.

Database Access

This chapter explores how OpenAccess can access the database using LINQ, Object Query Language (OQL) and native SQL. The chapter then explains how to hook up stored procedures using the OpenAccess wizards.

Optimization

This chapter demonstrates techniques for improving performance. "Fetch Plans" are described to show how they can fine-tune the number columns of data that are returned. The basics of the built-in caching system are explained and a step-by-step walk through details the steps to configure caching for single process access.

1.4 About Telerik

Telerik is a leading vendor of User Interface (UI) components for Microsoft .NET technologies – ASP.NET AJAX, Silverlight, WinForms and WPF, and .NET Reporting and content management solutions. Building on its expertise in interface development and Microsoft technologies, Telerik helps customers build applications with unparalleled richness, responsiveness and interactivity. Created with passion, Telerik products help thousands of developers every day to be more productive and deliver reliable applications under budget and on time.

1.5 About Falafel

Founded in 2003, Falafel Software, Inc. provides the highest quality software development, consultation, and training services available. Starting initially with consulting and training, Falafel Software expanded rapidly on the excellence of its engineers and the incredible sense of teamwork exhibited by everyone in the company. Employees include best-selling authors, industry speakers, technology decision makers, and former Microsoft and Borland engineers. All of Falafel engineers are Microsoft Certified Professionals, Certified Application Developers, or Most Valuable Professionals.

Falafel has written the following Telerik courseware:

- RadControls for ASP.NET
- RadControls for ASP.NET AJAX
- RadControls for Winforms
- Telerik Reporting

Falafel is a Microsoft Gold Certified Partner focusing on the following technologies:

- Microsoft .NET based web and windows development
- Database design and implementation (SQL server and Oracle)
- Implementation of large enterprise grade solutions based on Microsoft technology
- Windows Communication Foundation
- Windows Workflow Foundation
- ASP.NET (AJAX)
- Solutions based on DotNetNuke and SharePoint
- Compact Framework and Mobile development for handheld devices.

1.6 Introducing Telerik OpenAccess ORM

When a relational database management system (RDMS) is used by a object-oriented programming (OOP) language you run into a set of difficulties collectively termed "Object Relational Impedance Mismatch" where concepts and techniques don't translate correctly between the two systems.

As the industry has moved from a three tier model to n-tier models, the object relational impedance mismatch has become more prevalent. Object Relational Mappers (ORMs) exist to bridge this gap as best as possible. Telerik's OpenAccess ORM is an industrial strength ORM that will meet the needs of all modern applications. It offers a wizard for both Forward and Reverse mapping to all major databases including Microsoft SQL Server, tight Visual Studio integration, LINQ support, transparent persistence and lazy loading, runs in medium trust, as well as a fully scalable architecture via its Fetch Plans and Level 2 cache.

The Three Tier Model

Business applications today all access data as part of their core functionality. As relational database servers gained in popularity 20 years ago, the industry moved from a one tier (mainframe) model to a client server model where we had a client performing the presentation logic and most of the business logic and the

server with the data storage and some business logic in the form of stored queries. By the early 1990s this model broke down due to high maintenance costs and a lack of a separation of concerns and we moved to a three-tier architecture as shown in the figure below.

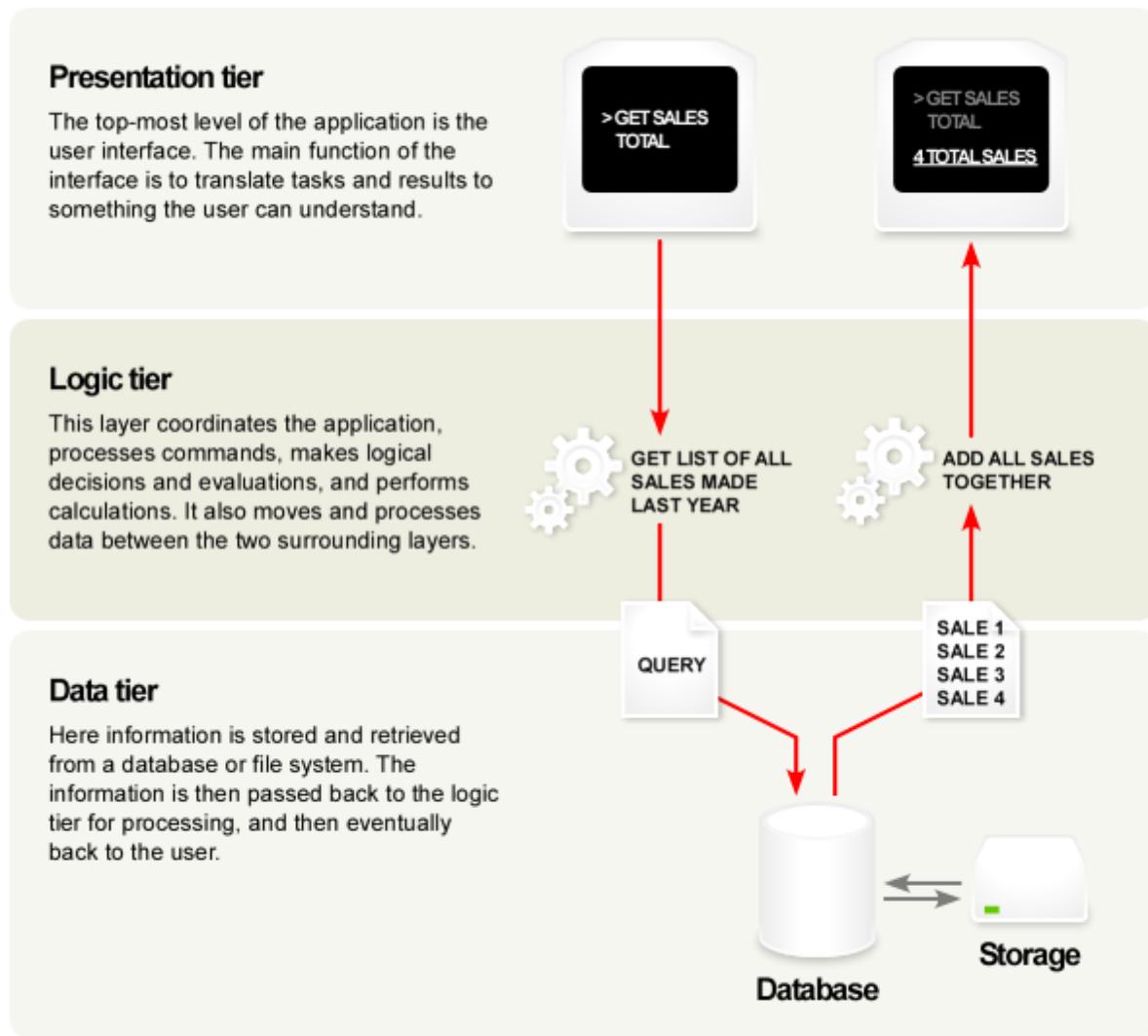


Figure 1
A 3-tier architecture. (Source: Wikipedia)

A three-tier architecture enforces a logical (and sometimes physical) separation of three major concerns:

- Presentation: the user interface components.
- Business Logic: processes commands requested by the users, makes logical decisions such as calculations and computations. Retrieves data from the data tier.
- Data storage and retrieval: storage and retrieval of data for the system and passes the data to the business layer for processing and ultimate rendering to the user by the presentation tier. In theory this tier can be file storage, XML, or a relational database, however, a relational database is by far the most common usage.

The goal of separating the logic is twofold. First there is a performance gain by having the database server focus only on database storage and retrieval. Specific hardware and topologies (such as RAID) are used for database storage and access that are different from an “application server” or middle tier of business objects

and logic. In addition with powerful client machines it made sense to push UI processing down to the client.

Second was the separation of concerns principle. By separating out the logic of a system, you can easier maintain the overall system, reuse code, and keep associated logic and code in one location.

The nTier Model

By the later 1990s, the industry extended the three-tier model to a multi-tier approach. The model is logically the same but what forced it to change was that the Internet became an important part of many applications.

Web services (and later REST data) have become more integrated into applications. As a consequence the data tier usually became split into a data storage tier (database server) and a data access layer or tier (DAL). In very sophisticated systems an additional wrapper tier is added to unify data access to both databases and web services. Web browsers were far less powerful than a traditional client tier application and the user interface logic became split across the browser with JavaScript and the server with web server UI rendering logic such as ASP or PHP.

Tiers started to get blurred ever further with the addition of stored procedures by all the major database vendors and open source databases. This spread some business logic from the business tier to the database tier, creating tiers within tiers. For example a business component inside of Microsoft Transaction Server (an Object Request Broker or ORB) is a logical business tier; however, it most likely calls a stored procedure, which is a logical business tier inside of the database tier.

As tiers got more blurred due to the Internet, technology innovations and services, the three-tier model evolved to the n-tier model as shown in an example in the figure below

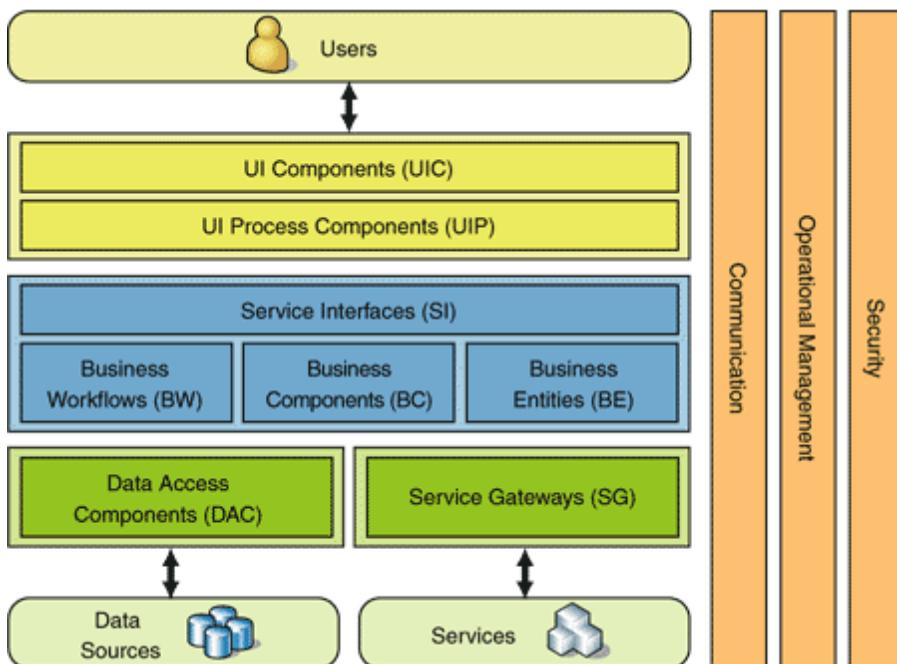


Figure 2
An n-tier architecture. (Source MS Patterns and Practices)

The Problem with the nTier Architecture

The n-tier architecture has been very successful. Most sophisticated applications today use some form of

the n-tier model. Years ago to support this model, enterprises would have as many as five different job titles around the data centric business application. The job titles were:

- Data Modeler
- Database Administrator
- SQL Programmer
- Object Modeler
- Application Developer

The data modeler would design the physical tables and relationships. The DBA would create the tables, maintain them and come up with an index and physical disk strategy and maintenance plan. The object modeler would build and object model (or API) and map behaviors to methods. The SQL programmer would work with the object modeler and application developer and write stored procedures and views under the guidance of the DBA. The application developer would use the object model components and would “glue” the application together.

That was then. While some large organizations still develop this way the advent of RAD tools in the late 1990s, the agile/XP movement in the early part of this decade, the .com boom, and ultimately offshoring and budget cuts, most firms do not organize this way anymore. Many smaller shops have one “database guy” (if at all) and one “code guy.” Some only have one person in total. Companies then push most of the modeling and procedure creation to the “db guy” and application developer.

With budget cuts, smaller teams, and the rise of cross-function teams there are not enough “database guys” to go around. Developers complain that they spend over 30% of an application’s code on database access code. This only exacerbates the object-relational impedance mismatch.

The ObjectRelational Impedance Mismatch

Database normalization theory, based on mathematics, encompasses different strategies than object oriented theory, which is based on software engineering principles. Database tables model data and prescribe storage techniques. Objects model data and behavior. The problem is that there are subtle differences between the way a database is designed compared to the way an object model is designed. The approach of doing straight mapping of database tables to objects leads to the famous “object-relational impedance mismatch.”

To demonstrate the impedance mismatch let’s take a look at an example. This sample was designed by Scott Ambler in a more detailed discussion of the impedance mismatch. Below in Figure 3 is a simple database model diagram. There are four database tables: a Customer table, an Address table, and a many-to-many table called CustomerAddress, linking them together, representing the notion that a customer can have multiple addresses and addresses are reusable, even between customers. Finally, there is also a support or “lookup” table for States, representing a one-to-many relationship between States and Addresses.

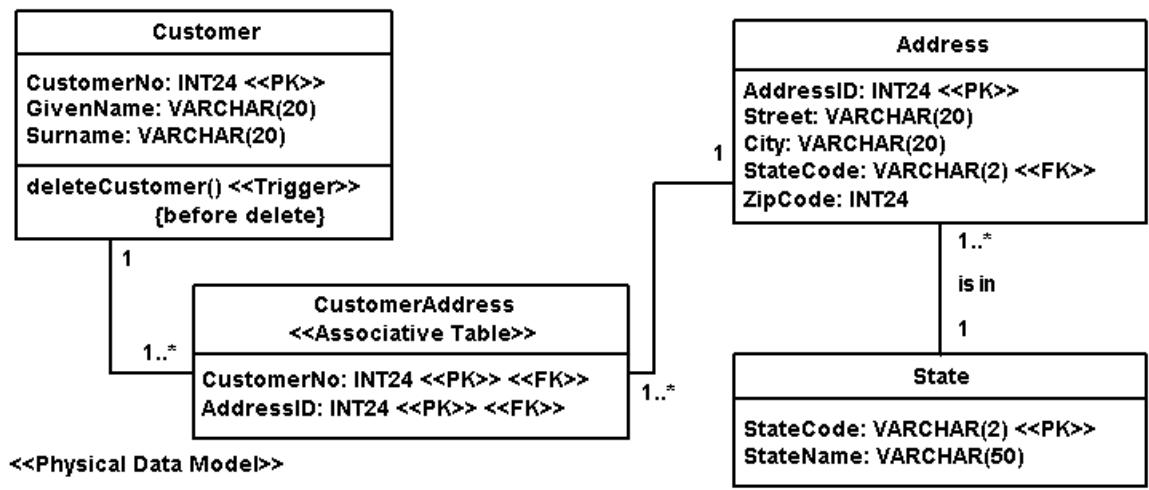


Figure 3
A database model. Source: <http://www.agiledata.org/essays/impedanceMismatch.html>

Now let's consider an object model that would interact with this data. In the figure above, we have four objects: a Customer object, an Address object and a supporting State object. Notice that there is no need for the "many to many" database table to be modeled since there is a "lives in" and 1..* arrow notation between the Customer and Address objects. This will indicate that you will have an Address collection associated with the Customer. In addition we have a separate ZipCode Object that does not exist in the database model. This object is used for validation and formatting, behaviors that are not necessary to model in the database.

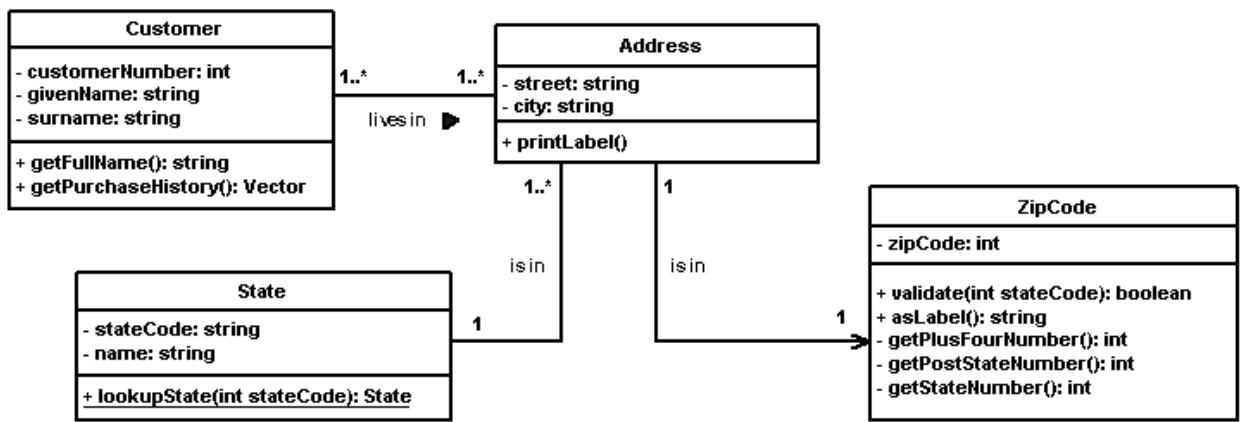


Figure 4
An object model. Source: <http://www.agiledata.org/essays/impedanceMismatch.html>

Notice the subtle difference between the figures representing the relational and object models? They each have 4 objects, some are identically named (Customer, Address, and State) and look similar, but the ZipCode object is quite different from the ZipCode represented in the database model. This is because object models structure both data and behavior while the database model only models data. Triggers are not considered a behavior, it is more about data updates and flow.

Data Access Layers (DALs)

The impedance mismatch has no easy solution, there are fundamental differences in the database normalization theory and object oriented theory. To bridge the gap as best as you can and to reduce the amount of code you have to write for database access developers have taken to write data access layers (DALs). The goal of a DAL is to decouple the data access code from the object model, allowing the object model to evolve according to its behaviors and the database to evolve based on its specific needs. A DAL should have these characteristics:

- Be completely independent of the object model. In other words the DAL should place no constraints on the object model.
- The DAL should hide all the data access code from the object model. This is sometimes referred to as persistence ignorance. Your object model should not care if you are using raw SQL, stored procedures, or an ORM. If you have the name of a stored procedure or SQL inside of your business (domain) objects then you violate this point.
- The DAL should be able to be ripped out and replaced with minimal to no impact.

The problem with DALs is that they are time consuming to write and not easily reusable from application to application. This is where ORMs come in.

Object Relational Mapping (ORM)

To overcome the impedance mismatch in a DAL you have to understand and implement the process of mapping objects to relational database tables. Class attributes will map to zero, one, or many columns in a database table. If there is proper mapping and persistence ignorance, the developer should only have to worry about working with the objects in the domain model and the DAL takes care of the rest. One way to facilitate this is to use an Object Relational Mapper or ORM.

An ORM is automated way to create your DAL based on your database and object model by mapping domain objects to database tables. Wikipedia defines an ORM as: "a programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages. This creates, in effect, a "virtual object database," which can be used from within the programming language." An ORM has to provide a facility to map database tables to domain objects, usually a design surface or wizard. This mapping is in-between your database and domain model, independent from the source code and the database. The ORM runtime then converts the commands issued by the domain model against the mapping into back end database retrieval and SQL statements. Mapping allows an application to deal seamlessly with several different database models, or even databases.

An ORM's mapping eliminates the need for a developer to manually write an entire DAL. A good ORM will eliminate 90% of the DAL code with its mapping and persistent objects. Since developers tend to focus 30% of their code on writing a DAL, an ORM will save over 25% of the total application code. Since the DAL is generated by an ORM and not by hand, it will be error free and always conform to product standards, standards that are easier to change in the future if products change.

When it comes to mapping there are two approaches that an ORM can take: forward mapping and reverse mapping. Forward mapping will take your already existing object model and then create a database schema out of it. Reverse mapping is the process of taking an already existing database and creating a set of objects from the tables. Most ORMs will support either forward or reverse mapping, and some support both methods. An automatic one-to-one mapping may not be preferred due to the differences between the object model and an appropriate data model (as show in Figures 3 and 4), most ORMs will give the user the ability to alter the mappings.

Besides mapping, an ORM has to provide a way for your domain objects to issue database agnostic commands to the DAL, have the DAL translate that command to the appropriate SQL, and then return a status (in the case of an update, etc), an object or object collection to the requesting domain object. This

has to be done in a transparent way, the objects in your domain need to be ignorant of what happens behind the scenes in the DAL (persistence ignorance). Popular ways to do this today are using an ORM LINQ provider or an ORM specific object query language (OQL). The most important thing is that you are querying your DAL in an agnostic way. For example consider this simple LINQ Query against a popular ORM:

```
var result = from o in scope.Extent<Order>()
where o.Customer.CustomerID.Matches("ALFKI")
select o;
```

This code only concerns itself with the object model and more specifically with the Order object and returns a collection of Orders filtered by the Customer ID "ALFKI". Taking the example further, following code demonstrates using an ORM to filter an ASP.NET GridView based on the results of the user input, notice that we are querying objects (the Order again) and projecting the results into a new object, never worrying about the database, connections, or any SQL code. That was handled by the ORM in the DAL.

```
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{
    //get the selected customer id
    string customerid = DropDownList1.SelectedValue.ToString();
    //linq query
    IObjectScope scope = ObjectScopeProvider1.GetNewObjectScope();
    var result = from o in scope.Extent<Order>()
    where o.Customer.CustomerID.Matches(customerid)
    select new { o.OrderID, o.ShipName, o.ShipCity, ShipCompany = o.Shipper.CompanyName };
    //databind the results
    GridView1.DataSource = result;
    GridView1.DataBind();
}
```

In addition to mapping and commands, ORMs should also integrate with development IDEs such as Microsoft Visual Studio (or Eclipse), provide caching techniques, integrate with source code version control software, and provide a testable framework. This will enable better developer productivity while saving the developer from having to write the DAL from scratch.

Telerik OpenAccess ORM

In November 2008 Telerik released OpenAccess ORM to its customers. OpenAccess is a mature ORM that was developed by Vanatec prior to its acquisition by Telerik in Autumn 2008.

Mapping

OpenAccess has deep integration with Microsoft Visual Studio and provides both forward and reverse mapping to six major databases including Microsoft SQL Server and Oracle with an easy to use Wizard shown in the figure below. As you can see, the wizard gives you full control over what to map and how to map it (as a class or collection). OpenAccess gives the user complete control over the object and member naming (or database field and table naming in the case of reverse mapping).

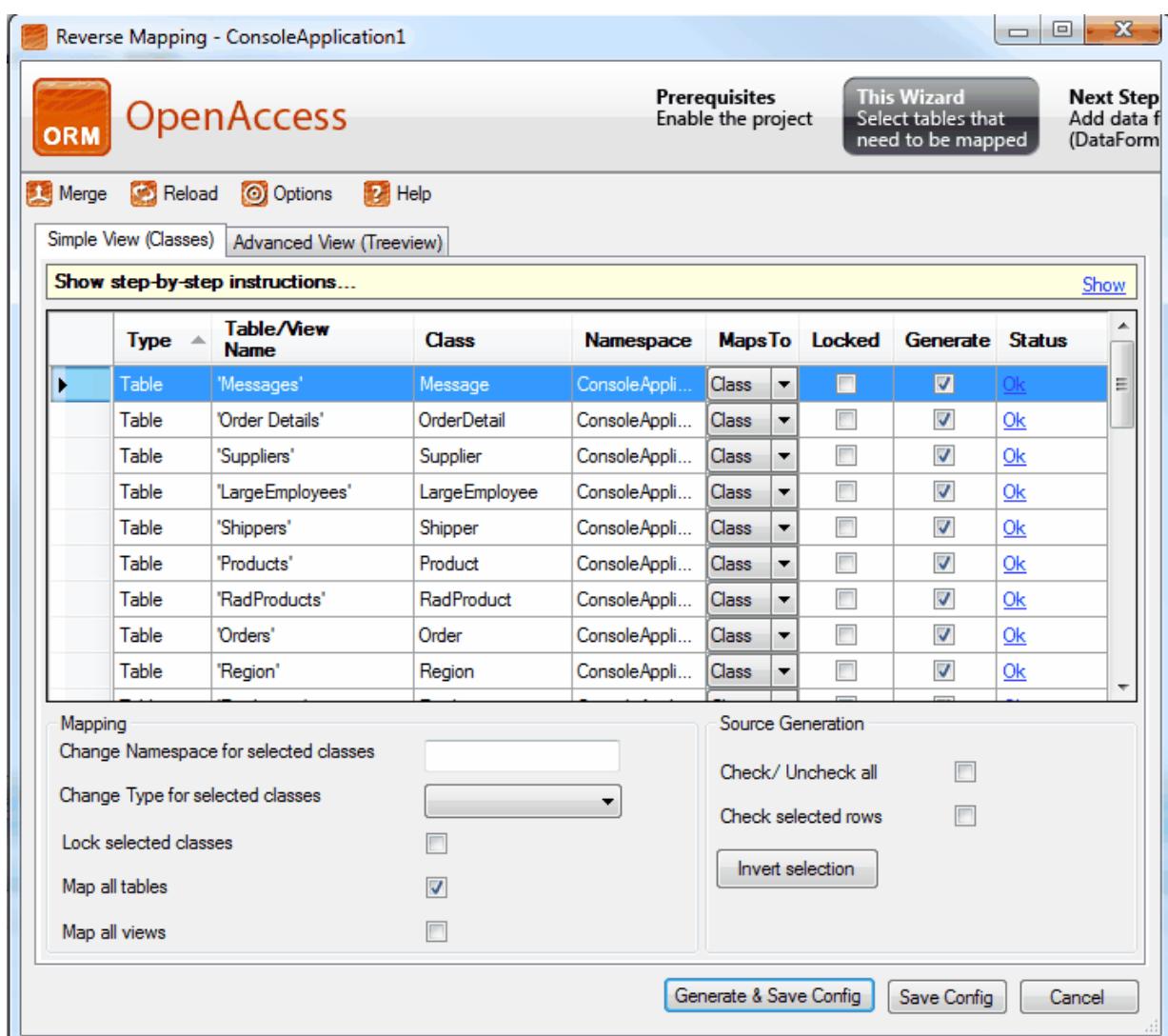


Figure 5
OpenAccess (Reverse) Engineering Wizard.

OpenAccess also generates a DAL that is transparent persistence and mostly persistence ignorant and by utilizing partial classes, generating very clean reflection free classes, allowing you to run in a medium trust environment. If you need it, you have the ability to add your own additional logic to the DAL that will not be overwritten if you need to regenerate the class. This also makes integration with source version control software much easier as well as facilitates testability and TDD. Here is a class generated by OpenAccess, it is just like a normal class that you would create:

```
using System;
using System.Collections.Generic;
namespace ConsoleApplication1
{
    //Generated by Telerik OpenAccess
    //NOTE: Field declarations and 'Object ID' class
    // implementation are added to the 'designer' file.
    // Changes made to the 'designer' file will be
    // overwritten by the wizard.
    public partial class Region
    {
```

```

//The 'no-args' constructor required by OpenAccess.
public Region()
{
}
[Telerik.OpenAccess.FieldAlias("regionID")]
public int RegionID
{
    get { return regionID; }
    set { this.regionID = value; }
}
[Telerik.OpenAccess.FieldAlias("regionDescription")]
public string RegionDescription
{
    get { return regionDescription; }
    set { this.regionDescription = value; }
}
}
}
}

```

Data Access

You can query the objects in the DAL using standard Object Query Language (OQL) and OpenAccess provides an OQL explorer tool shown in the figure below where you can test your OQL and even see what SQL OpenAccess generates against the backend datasource.

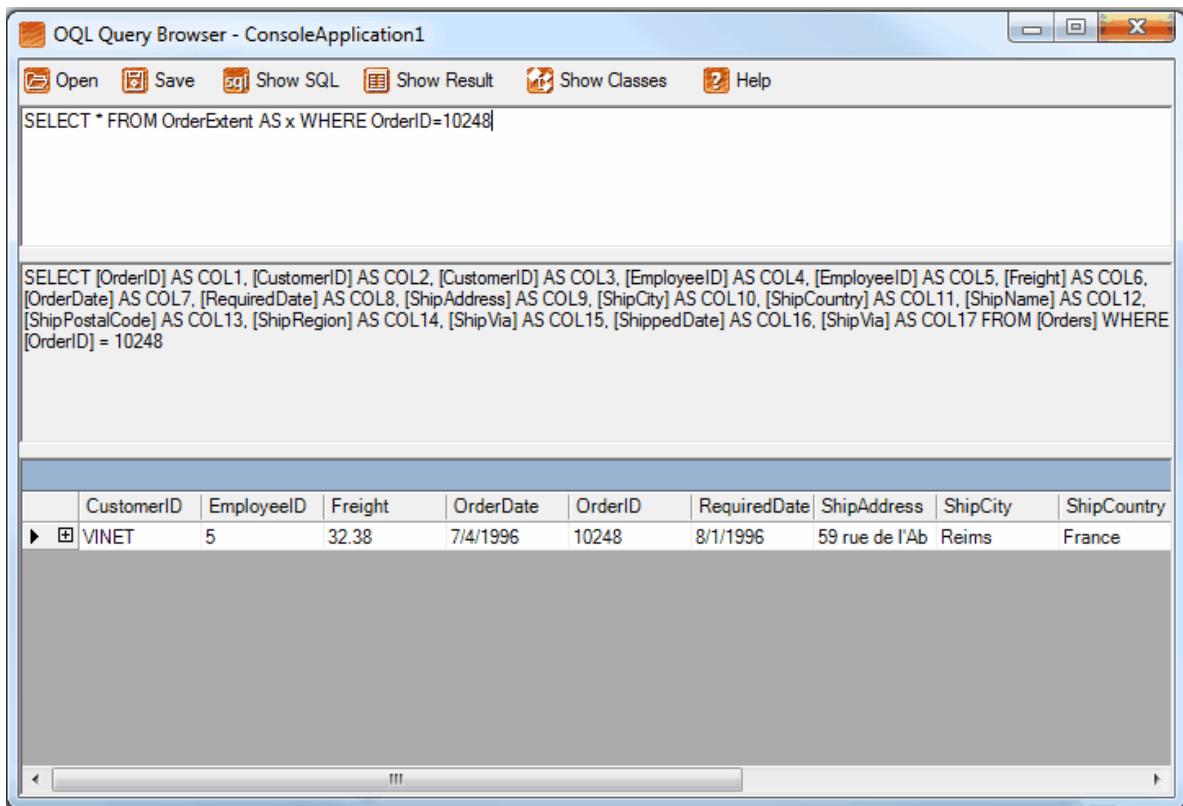


Figure 6
The OQL Query Browser

OpenAccess has first class LINQ support through its automatically generated ObjectScopeProvider. The LINQ support is specific to OpenAccess, not the backend database. An example is shown here, after you

get a reference to the ObjectScopeProvider provided by your DAL, you can use the standard LINQ query operators on that reference.

```
IObjectScope scope = ObjectScopeProvider1.GetNewObjectScope();  
var result = from o in scope.Extent<Order>()  
where o.Customer.CustomerID.Matches("ALFKI")  
select o;
```

If you are not satisfied with the SQL that OpenAccess generated via LINQ or OQL, you can optimize the query that OpenAccess will generate by defining a fetch plan. You can graphically define a fetch plan using the fetch plan browser as shown in the figure below

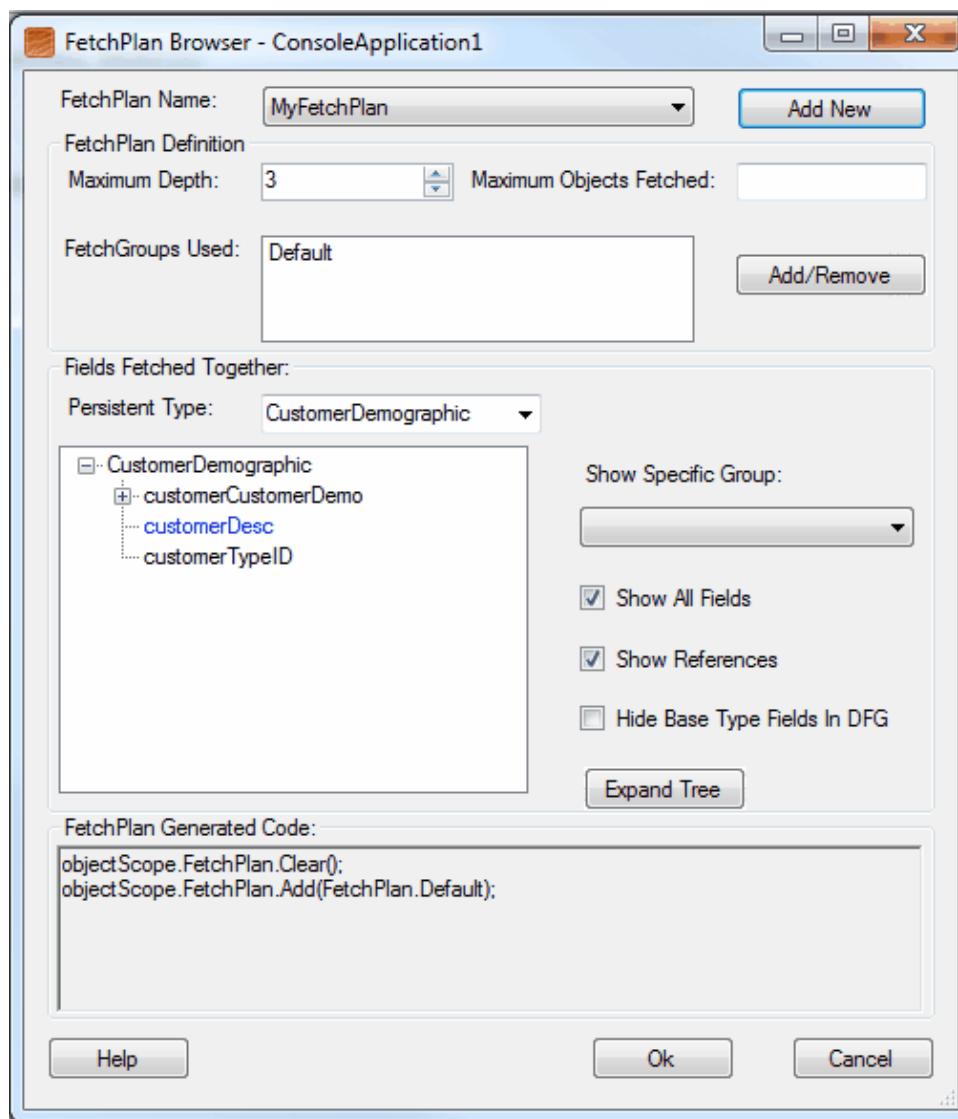


Figure 7
The Fetch Plan Browser

Lazy Loading and Level 2 Cache

OpenAccess also uses “lazy loading” or the ability to load resources as they are needed, increasing performance. OpenAccess also has a very unique caching mechanism, the 2nd level (L2) cache, which reduces the calls made to the backend database server. Database access is therefore necessary only when the retrieving data that is currently not available in the cache. This increases the efficiency and performance of your application and enables you to move effortlessly to a multi-tier application or web-farm environment.

The cache is even more scalable via the 2nd Level Cache Cluster. In an application server farm scenario where more than one application is using the L2 cache for the same database, OpenAccess synchronizes the various caches, even preserving transactional integrity. Every modifying transaction sends its eviction requests to all participants in the L2 cache cluster asynchronously, ensuring that the modified data will be expired and preventing both incorrect and dirty reads.

Distributed Applications

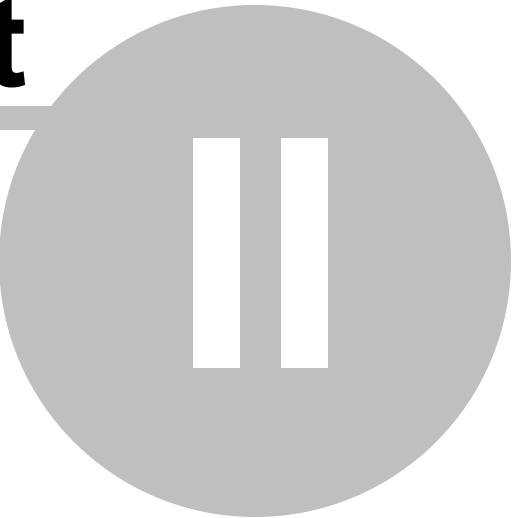
Applications today need to work across multiple tiers even if they are not always connected. Service oriented architectures via WCF, Web applications, mobile application and asynchronous applications still need to deal directly with persistent objects. OpenAccess supports distributed environments by allowing you to work with portions of your data in a disconnected mode via the OpenAccess Object Container. In addition OpenAccess integrates with client side technology such as Silverlight, allowing you to spread your processing across multiple tiers.

Conclusion

If you are looking to bridge the object-relational impedance mismatch and increase your productivity and data access performance, you should consider the Telerik OpenAccess ORM. It offers a wizard for both forward and reverse mapping to all major databases including Microsoft SQL Server, tight Visual Studio integration, LINQ support, transparent persistence and lazy loading, runs in medium trust, as well as a fully scalable architecture via its fetch plans and level 2 cache.

For more information please visit: <http://www.telerik.com/products/orm.aspx>

Part



||

Getting Started

2 Getting Started

This chapter demonstrates the basics for setting up your project to use OpenAccess ORM, transform .NET objects to database tables and back again. We won't get into too much depth yet, but instead use the walk-through examples to get confidence using OpenAccess and a overall feel for how the product works.

In this chapter you will learn:

- How to ORM-enable your project.
- The purpose and use of the ObjectScope.
- How to use the Persistent attribute.
- The concept of "persistence by reachability".
- How to map your .NET objects automatically to database tables.
- How to map your database tables to .NET objects.
- How to perform Create-Read-Update-Delete (CRUD) and the role of transactions in these operations.



Find the source projects for this chapter at Projects\ORM\<CS\VB>\Made_Easy

2.1 ORM Enable Project

Before you can move data between .NET objects and databases you must "ORM-enable" the project.

A Visual Studio menu option displays a wizard that performs the following:

- Gathers connection information
- Writes to the project configuration file
- Creates a helper class for getting at the connection
- Adds relevant assembly references



While you're getting used to OpenAccess you can use the defaults, and "dial in" project specific requirements later.

In this example we will configure OpenAccess to use a SQL Express database in a minimal number of steps.

- 1) In Visual Studio, create a new Console application and name it "1 ORM_Enable".
- 2) From the Visual Studio menu, select Telerik > OpenAccess > Enable Project to use ORM...

This step will display the OpenAccess Enable Project Wizard.

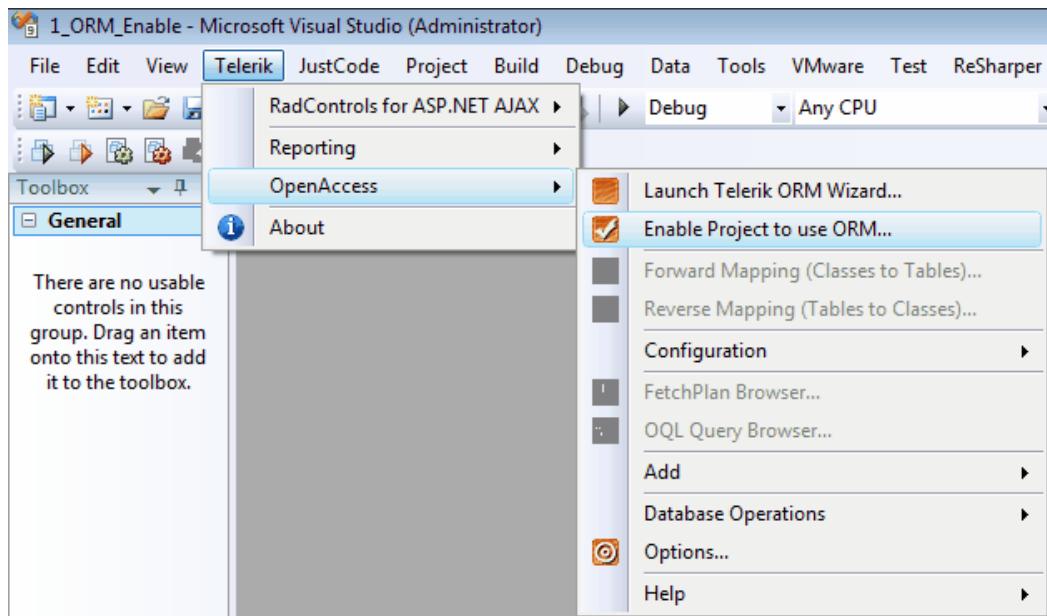


Figure 8

- 3) The Prerequisites page of the wizard lists the actions to be performed. Click the **Next** button.

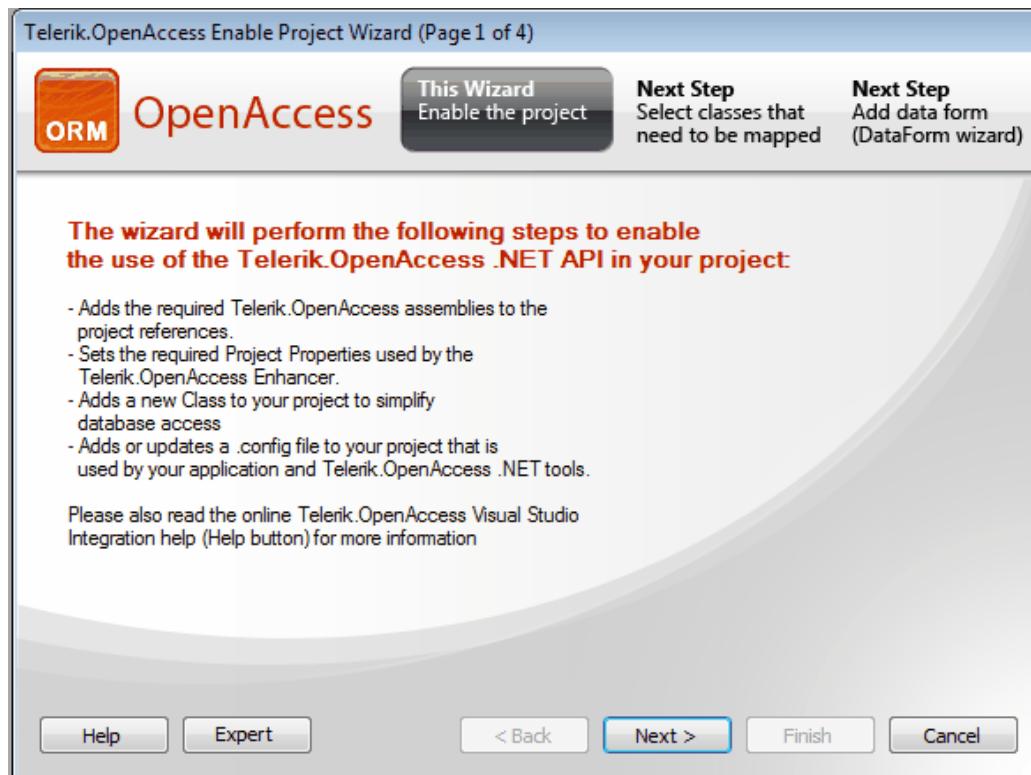


Figure 9

- 4) The second page of the wizard allows you to enable persistent classes and to automatically create a data access layer. Leave both options checked and click the **Next** button.

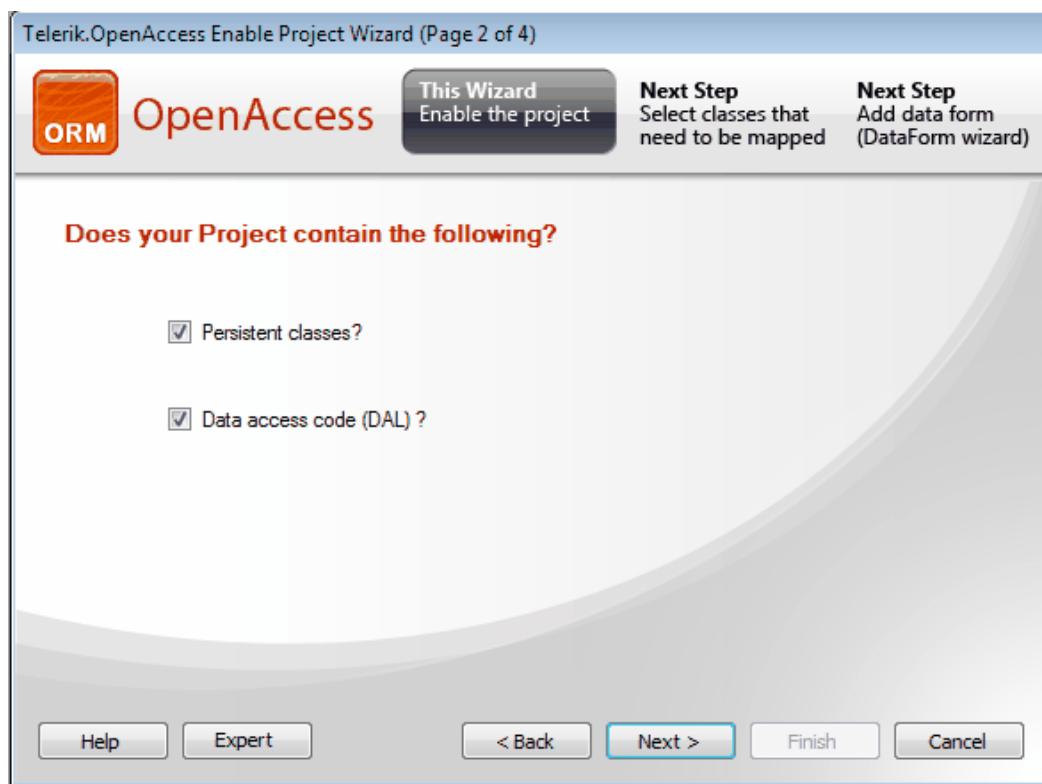


Figure 10

- 5) The third page of the wizard lets you create a data connection. Enter the following:
- Connection Id: "MyConnectionString"
 - Backend: Microsoft SQL Server
 - Server Name: "(LOCAL)\SQLEXPRESS"
 - Use integrated security: checked
 - Database Name: "MyDatabase"



Figure 11



Notice that above there are two options for the database connection. There is a choice to "Use standard connection strings (Web.config or App.config)" or a choice to "Use OpenAccess connection settings". In this example we are going to select "Use OpenAccess connection settings" but from a functionality standpoint either one will work equally well.

- f) Click the **Next** button.
- 6) The last page of the wizard summarizes the actions that will be performed. Click the **Finish** button to complete the wizard.



Figure 12

Take a look at your project in the Solution Explorer and notice that:

- The references now include Telerik.OpenAccess and Telerik.OpenAccess.Query.
- The wizard also added a "ObjectScopeProvider1.cs" if using C# or "ObjectScopeProvider1.vb" if using Visual Basic to the project. This key object manages the database connection. The ObjectScopeProvider provides access to an IObjectScope instance. The "ObjectScope" tracks changes to your persistent objects, introduces the ability to query data, update data and controls database transactions.
- The App.Config has a openaccess section that defines connections and can store mappings between objects and database tables.

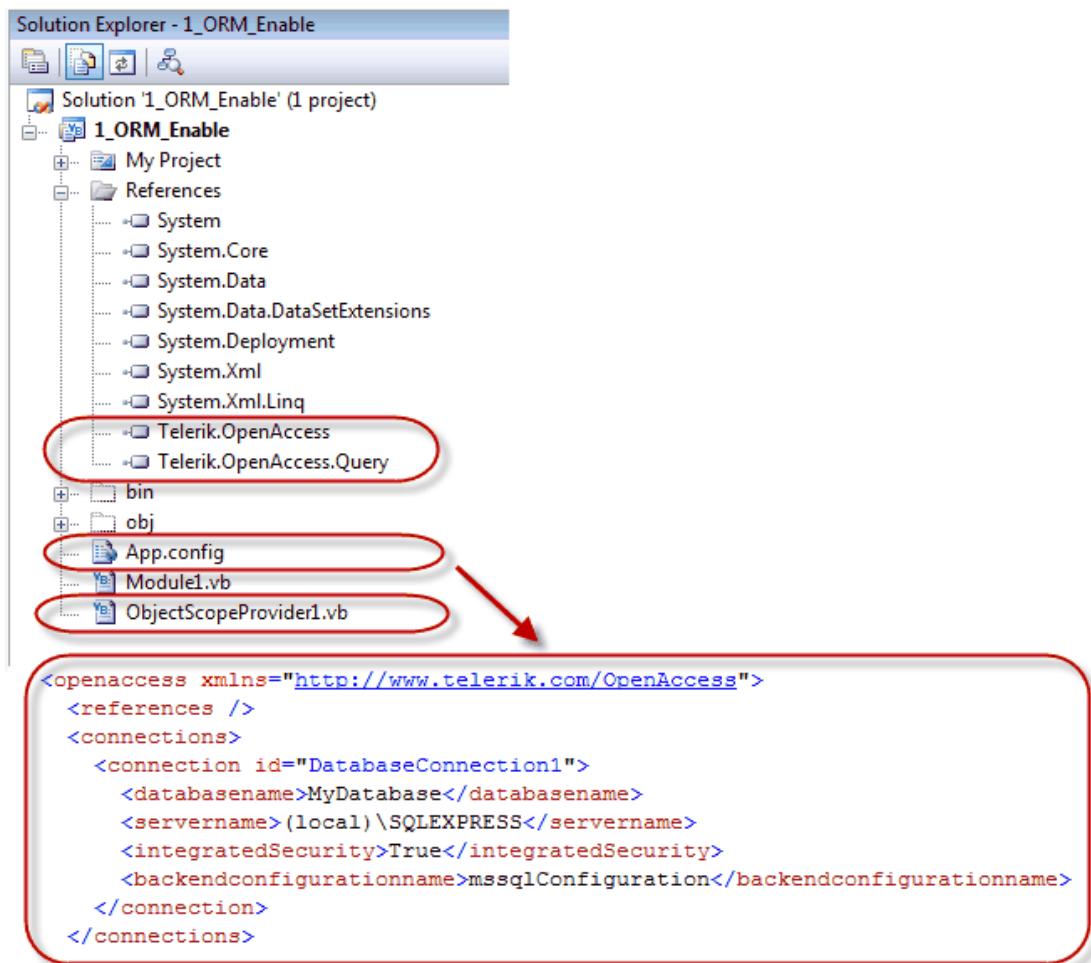


Figure 13

- 7) In the Solution Explorer, right-click the project and select **Add->Class...** from the context menu. Mark the class using the "Persistent" attribute.

OpenAccess requires that at least one class be marked as Persistent.

```

<Telerik.OpenAccess.Persistent()> _
Class Class1
End Class

```

```

[Telerik.OpenAccess.Persistent]
class Class1
{
}

```

Now your project is ORM-enabled and ready to be mapped.

2.2 Forward Mapping (objects -> database)

The beauty of OpenAccess is that you can work with the model you're most comfortable with: objects and .NET code or tables with SQL. For example, you can create a simple "Customer" object and OpenAccess creates a database and table automatically. This process of representing an object model in database form is called "Forward Mapping". All that is needed, once you have ORM-enabled the project, is to decorate your classes with the `Persistent` attribute. In this walk-through we will set the `Persistent` attribute by hand, while in later exercises we'll use the OpenAccess wizards to do that for us.

The example below will create Customer and Order classes, populate them with data, add them to the "scope" and persist the data to the database.

💡 Be aware that all changes that impact the database must be made within the context of a transaction. This is implemented in OpenAccess using the `IObjectScope.Transaction` object and three key methods `Begin()`, `Commit()` and `Rollback()`.

The general steps in the example code are:

- Define a persistent class, i.e. define a class to suit your business purpose, then mark the class with the `Telerik.OpenAccess.Persistent` attribute.
- Get an `IObjectScope` instance. We will use the "ObjectScopeProvider" created in the previous "ORM Enable Project" step to create an `IObjectScope` instance for us.
- Start a transaction using the `IObjectScope.Transaction.Begin()` method.
- Create and populate instances of your persistent class.
- Commit the transaction using the `IObjectScope.Transaction.Commit()` method.

Here's an example of what that might look like:

```
Dim scope As IObjectScope = ObjectScopeProvider1.ObjectScope()  
  
scope.Transaction.Begin()  
  
Dim _customer As New Customer()  
_customer.CustomerName = "Bob"  
_customer.CustomerNumber = 123  
  
scope.Add(_customer)  
  
scope.Transaction.Commit()
```

```
IObjectScope scope = ObjectScopeProvider1.ObjectScope();  
  
scope.Transaction.Begin();  
  
Customer _customer = new Customer();  
_customer.CustomerName = "Bob";  
_customer.CustomerNumber = 123;  
  
scope.Add(_customer);  
  
scope.Transaction.Commit();
```

Forward Mapping Walk-Through

- 1) Create a new Console application and call it 1 ORM Forward Mapping.
- 2) ORM-enable the project (see the previous "ORM Enable Project" section for details on how to do this).
- 3) In the Solution Explorer, right-click the project and select **Add->Class...** from the context menu. Mark the class using the "Persistent" attribute
- 4) Rename the new Class (should be called Class1) "Class1" file to "Customer". When prompted to rename the other references, click the Yes button.
- 5) Open the "Customer" to edit the class. Make it a public class and add an integer "CustomerNumber" property and a string "CustomerName" property.

```
 <Telerik.OpenAccess.Persistent()> _  
Public Class Customer  
  
    Private _customerNumber As Integer  
    Private _customerName As String  
  
    Public Property CustomerNumber() As Integer  
        Get  
            Return _customerNumber  
        End Get  
        Set(ByVal value As Integer)  
            _customerNumber = value  
        End Set  
    End Property  
  
    Public Property CustomerName() As String  
        Get  
            Return _customerName  
        End Get  
        Set(ByVal value As String)  
            _customerName = value  
        End Set  
    End Property  
  
End Class
```



Please take note that in the VB.NET code the Properties have fully defined Get and Set statements. Unlike C# in the current release, VB9 does not support Auto Implemented Properties. In the next release of VB (version 10), they will be supported. Until then if using VB.NET these properties will need to be fully defined when used with ORM.

```
namespace _1_ORM_Forward_Mapping
{
    [Telerik.OpenAccess.Persistent]
    public class Customer
    {
        public int CustomerNumber
        {
            get;
            set;
        }
        public string CustomerName
        {
            get;
            set;
        }
    }
}
```

- G** Be aware that as of this writing, OpenAccess does not yet fully support "Automatic Properties", i.
o e. the notation that has no specific internal fields, just the "get" and "set" without arguments.
t OpenAccess persists against fields, not properties. So in this example, the field names will be
c created automatically by OpenAccess and will be a little ugly, e.g.
h "<_my_number>k__backing_field". If you decide to reverse map, you will have an opportunity to
a rename the field back to the original property name. To avoid this issue altogether, fully define
! each property.
①

6) Open the "Program.cs" (C#) or the "Module1.vb" (VB) file for editing and add a reference to Telerik OpenAccess in the "Imports" (VB) or "using" (C#) section of code.

7) Add the following code to the Main() method of the Program class.

The code will use the ObjectScopeProvider created automatically when you ORM-enabled the project. The IObjectScope retrieved from the ObjectScope() method provides access to the underlying database connection, the transaction object and the ability to query your "Customer" object using SQL expressions.

```

Imports Telerik.OpenAccess

Module Module1

Sub Main()
    Dim scope As IObjectScope = ObjectScopeProvider1.ObjectScope()
    scope.Transaction.Begin()

    Dim _customer As New Customer()
    _customer.CustomerName = "Bob"
    _customer.CustomerNumber = 123

    scope.Add(_customer)
    scope.Transaction.Commit()
End Sub

End Module

```

```

using Telerik.OpenAccess;

namespace _1 ORM Forward Mapping
{
    class Program
    {
        static void Main(string[] args)
        {
            IObjectScope scope = ObjectScopeProvider1.ObjectScope();
            scope.Transaction.Begin();

            Customer _customer = new Customer();
            _customer.CustomerName = "Bob";
            _customer.CustomerNumber = 123;

            scope.Add(_customer);
            scope.Transaction.Commit();
        }
    }
}

```

- 8) Run the application.

Because it is a console application, the console will flash briefly on the screen. The database "MyDatabase" will be created in your local SQL express, a "Customer" table will be created and populated with a single row.

- 9) Use the Server Explorer, SQL Server Management Studio or database tool of your choose to view the new database, table and data. The figure below shows the data as it appears in Server Explorer.

OpenAccess automatically creates a primary key field "customer_id", backing fields for each property in the Customer object and a column to track the versioning of the record.

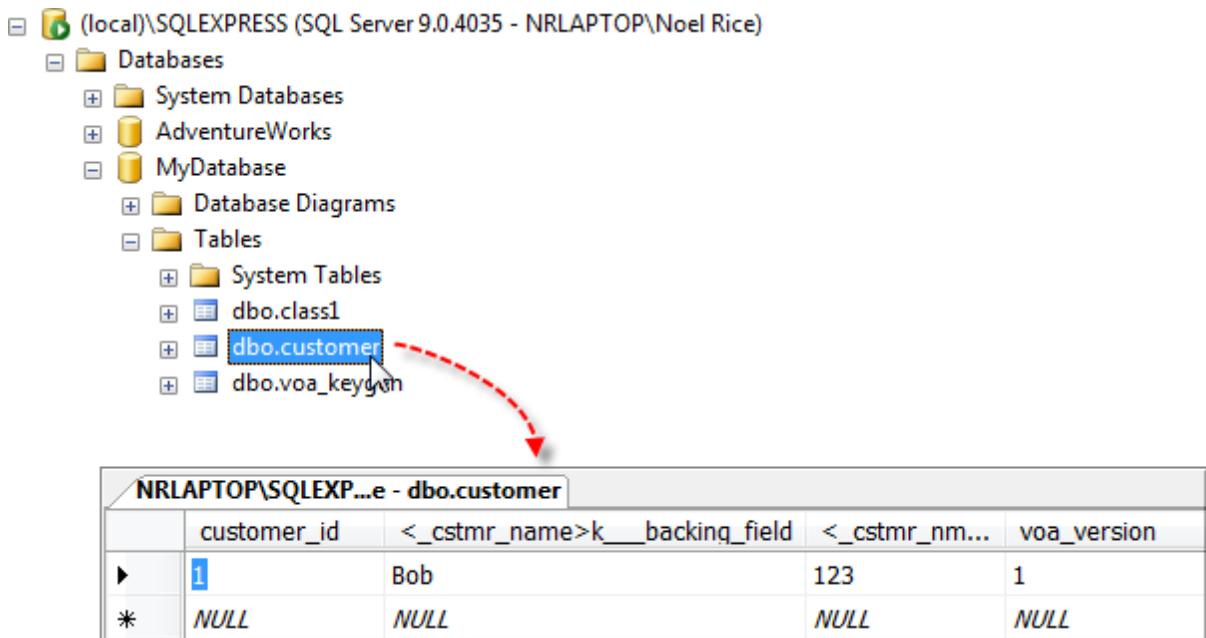


Figure 14

Now Lets see how you can update the object model and have the meta-data propagate to the database.

- 10) Add a new class to the project "Order". Add code for a single integer "OrderNumber" property so the code looks like the example below.

```

<#> <Telerik.OpenAccess.Persistent()> _
Public Class Order

    Private _orderNumber As Integer

    Public Property OrderNumber() As Integer
        Get
            Return _orderNumber
        End Get
        Set(ByVal value As Integer)
            _orderNumber = value
        End Set
    End Property

End Class

```

```
namespace _1_ORM_Forward_Mapping
{
    [Telerik.OpenAccess.Persistent]
    public class Order
    {
        public int OrderNumber
        {
            get;
            set;
        }
    }
}
```

- 11) Add a collection of Orders to the Customer object. The completed Customer class should now look like the example below. Use the IList for the Orders property type. Create the Orders instance in the constructor using a generic List of Order.

```
[<Telerik.OpenAccess.Persistent()>]
Public Class Customer

    ' Create private variables to hold the data values used
    ' by the public properties
    Private _customerNumber As Integer
    Private _customerName As String
    Private _orders As IList(Of Order)

    ' Class constructor
    Public Sub New()
        Me.Orders = New List(Of Order)
    End Sub

    Public Property CustomerNumber() As Integer
        Get
            Return _customerNumber
        End Get
        Set(ByVal value As Integer)
            _customerNumber = value
        End Set
    End Property

    Public Property CustomerName() As String
        Get
            Return _customerName
        End Get
        Set(ByVal value As String)
            _customerName = value
        End Set
    End Property

    Public Property Orders() As IList(Of Order)
        Get
            Return _orders
        End Get
        Set(ByVal value As IList(Of Order))
            _orders = value
        End Set
    End Property

End Class
```

```
using System.Collections.Generic;

namespace _1 ORM Forward Mapping
{
    [Telerik.OpenAccess.Persistent]
    public class Customer
    {
        // Class constructor
        public Customer()
        {
            this.Orders = new List<Order>();
        }

        public int CustomerNumber
        {
            get;
            set;
        }
        public string CustomerName
        {
            get;
            set;
        }

        // Create a New Generic List of Order
        public IList<Order> Orders
        {
            get;
            set;
        }
    }
}
```

- 12) Add a single order to the Customer Orders collection. The code for the Main() method should look like the example below.

```
 Imports Telerik.OpenAccess

Module Module1

Sub Main()
    Dim scope As IObjectScope = ObjectScopeProvider1.ObjectScope()
    scope.Transaction.Begin() 'Here is where the Transaction Starts

    Dim _customer As New Customer()
    _customer.CustomerName = "Bob"
    _customer.CustomerNumber = 123

    Dim _order As New Order()
    _order.OrderNumber = 1
    _customer.Orders.Add(_order)

    scope.Add(_customer)
    scope.Transaction.Commit() 'Here is where the Transaction is committed

End Sub
End Module
```

```
 static void Main(string[] args)
{
    IObjectScope scope = ObjectScopeProvider1.ObjectScope();
    scope.Transaction.Begin();

    Customer _customer = new Customer();
    _customer.CustomerName = "Bob";
    _customer.CustomerNumber = 123;

    Order order = new Order();
    order.OrderNumber = 1;
    _customer.Orders.Add(order);

    scope.Add(_customer);
    scope.Transaction.Commit();
}
```

13) Run the application. Look at the changes to the database.

OpenAccess creates a "join" table called "customer_ordr" that allows the most flexible combination of customers and orders. The new table for "Order" is automatically named "ordr" by OpenAccess to avoid naming conflicts.

	customer_id	<_cstmr_n...	<_cstmr_nm...	voa_version
▶	11	Bob	123	1
	customer_id	seq	ordr_id	
	11	0	1	
	ordr_id	<_rdr numb...	voa_version	
	1	1	1	

Figure 15

💡 Did you notice that the Orders are never explicitly added to the Scope? This feature is called "persistence by reachability". All persistent objects that are reachable from an object added to the scope are automatically stored in the database.

2.3 Reverse Mapping (database -> objects)

In a more traditional scenario, the database already exists and a data access layer needs to be created to work with the data programmatically. This next walk-through uses Customer, Orders and OrderDetails tables of the installed NorthwindOA SQL Express database.

- 1) Create a new Console application called 1 ORM_Reverse_Mapping.
- 2) ORM-enable the project (see the previous "ORM Enable Project" - this time, however click the option to "Use standard connection strings (Web.config or App.config) when the screen comes up to configure the connection.

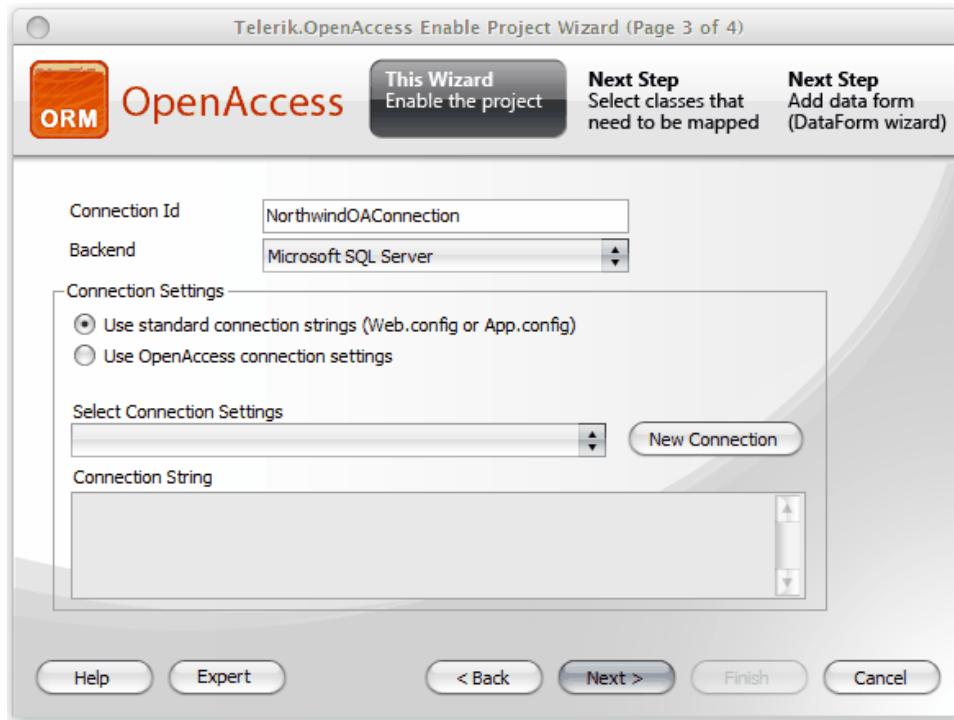


Figure 16

3) Name the Connection Id: NorthwindOAConnection

4) Click on the New Connection command button

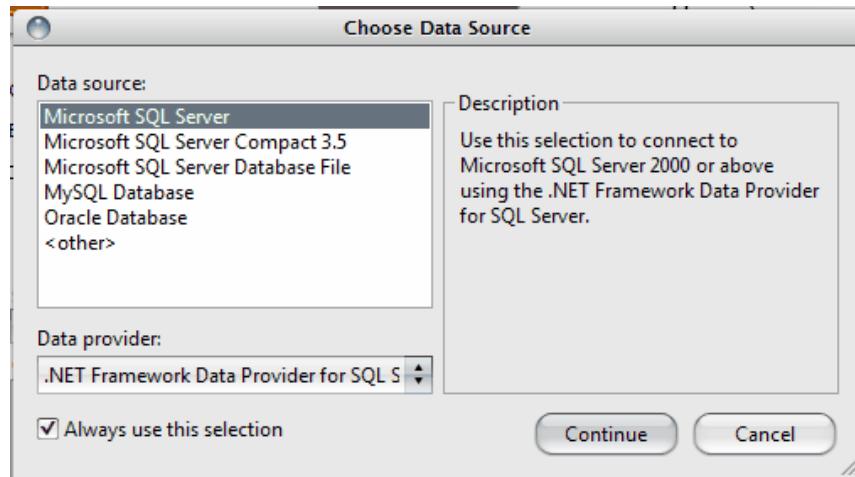


Figure 17

5) Click Continue

6) Enter the information as shown below and click OK:

a) Server name: (local)\SQLEXPRESS

b) Select or enter a database name: NorthwindOA

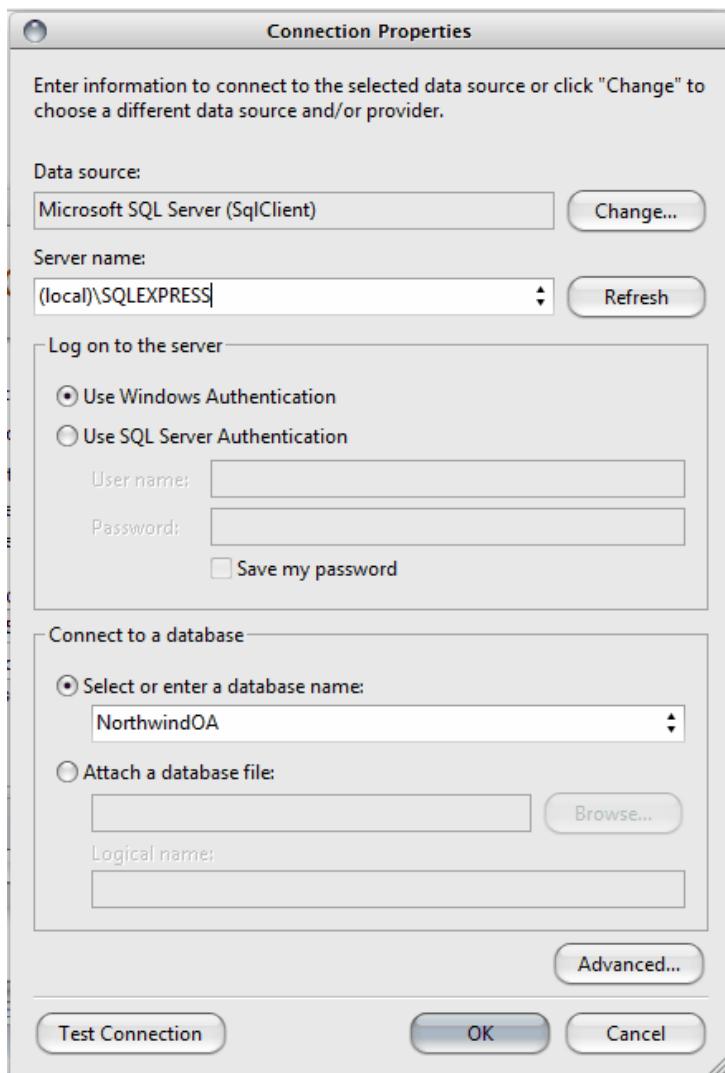


Figure 18

7) Click Next and then Click Finish

8) From the Visual Studio menu, select Telerik > OpenAccess > Reverse Mapping (Tables to Classes)...

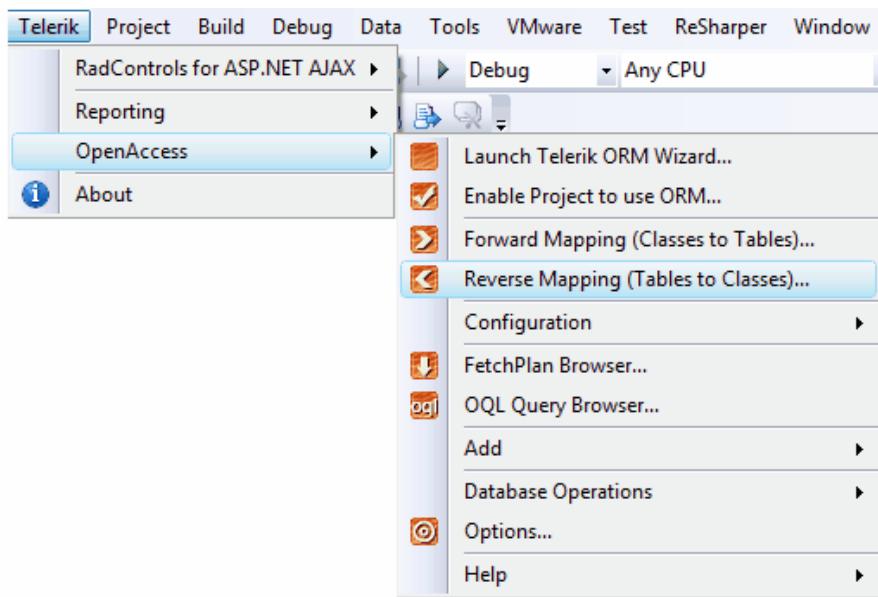


Figure 19

This step will load meta-data describing the tables from the connection defined when you ORM-enabled the project. The meta-data is displayed in the Reverse Engineering wizard dialog.

- 9) Select the Advanced Tab of the Reverse Engineering Wizard to display a tree view of all NorthwindOA tables. Configure the tables:
 - a) Click each table and select the Ignore radio button for each table except "Customer", "Orders", "Order Details" and "Products". The wizard should look something like the screenshot below.

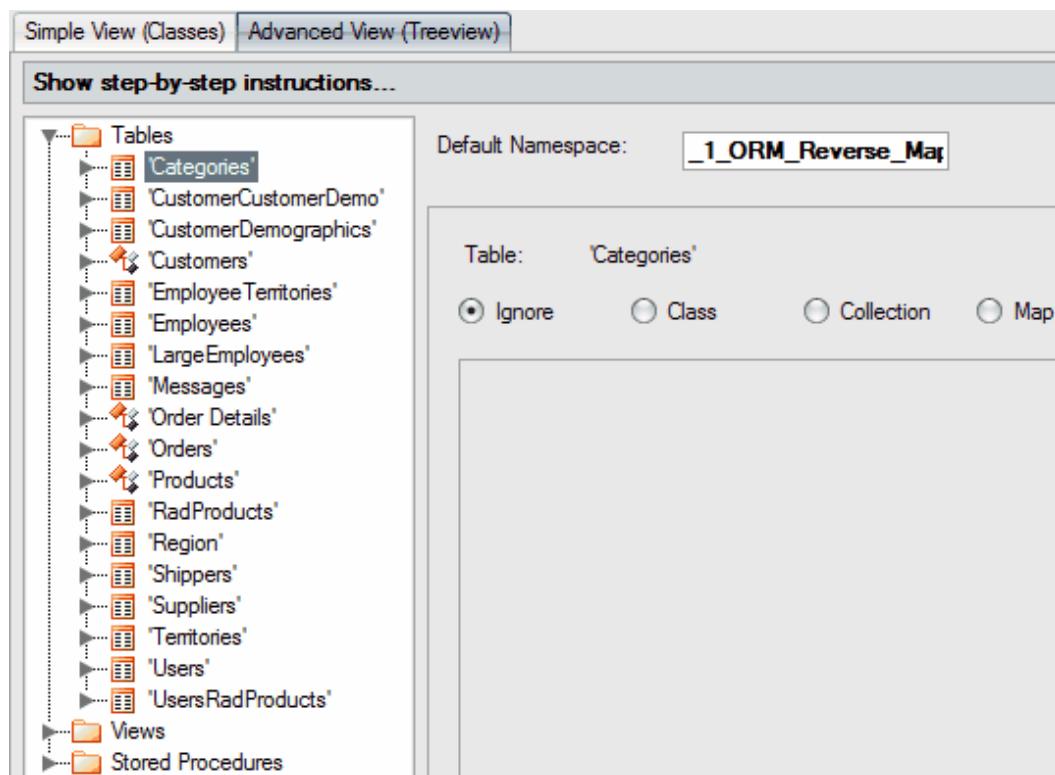


Figure 20

- b) Open up "Orders" table node in the Tables tree. Select the "shipper" reference and then click the **Remove** button.

 We don't need any of the Shippers table information, so removing the reference will prevent an error later on

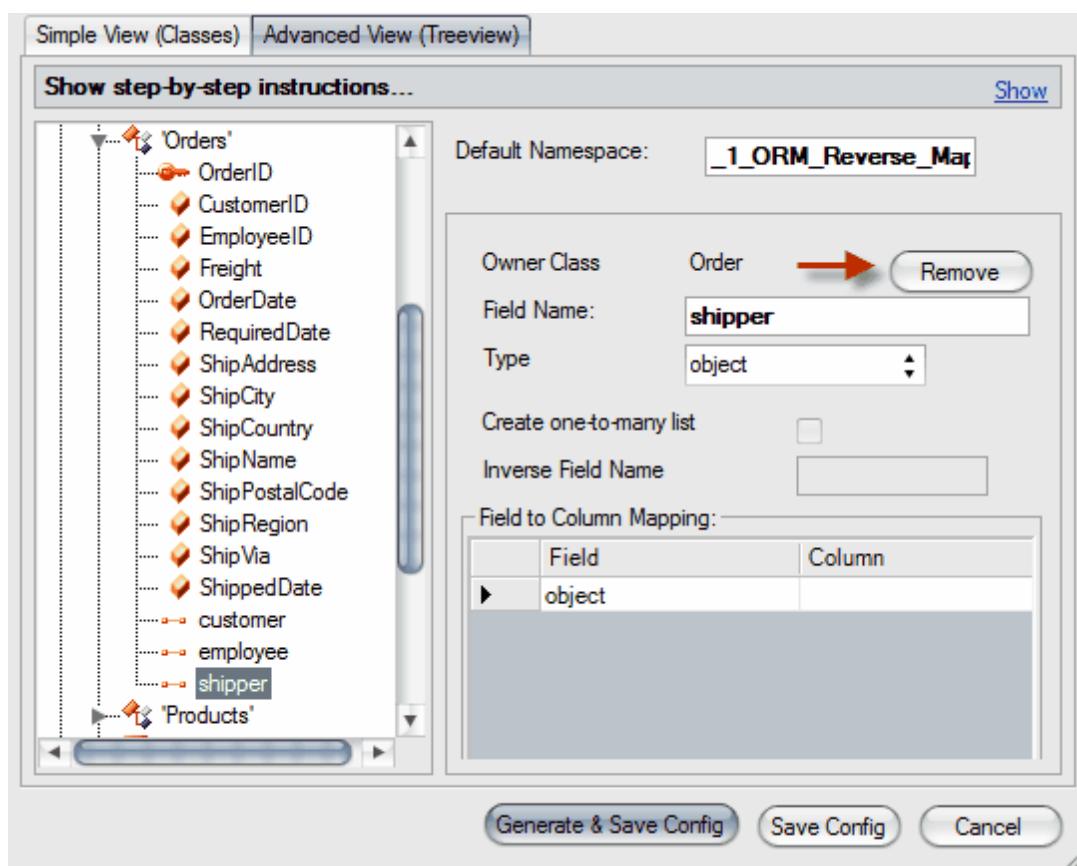


Figure 21

- c) Select the "employee" reference and remove that as well.
- d) Select the "customer" reference and select the "Create one-to-many list" checkbox.

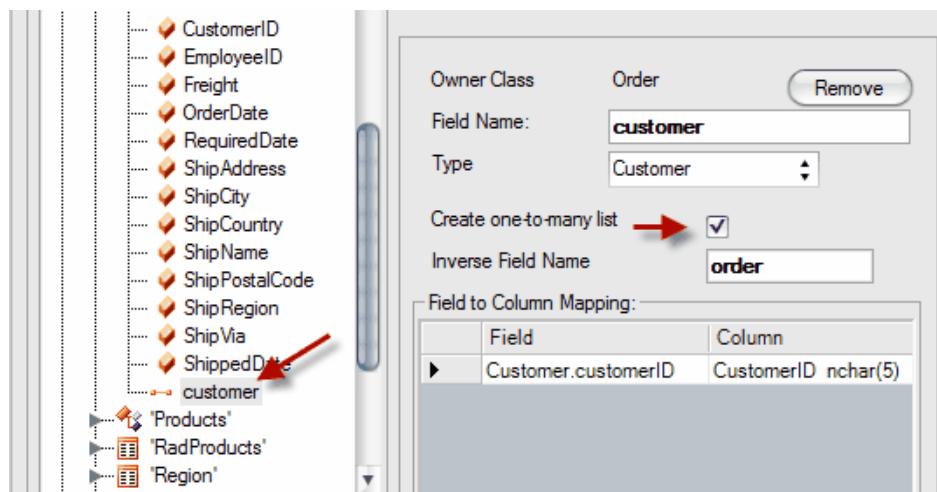


Figure 22

■ Note that in C# the field name will show as above, but in VB the field name gets an _ in front of it so that order will show as _order in the Inverse Field Name when using VB.NET.

- e) Open the "OrderDetails" node in the tree. Select the "order" reference and click the "Create one-to-many list" checkbox.
 - f) Click the Generate & Save Config button to close the dialog and create the classes. Click the **Yes** button to confirm generating source.
- 10) Add code to the Program Main() method that retrieves all customers, orders and order details.

■ *The ObjectScopeProvider passes back a IObjectScope. IObjectScope has several methods for querying data using straight SQL or the OpenAccess variety of SQL called OQL. In this case we call the GetSqlQuery() method and receive a Query object in return. The Query object has methods for returning several different flavors of collections including IEnumerable, IList and IBindingList. Once you have the collection of Customer objects you can iterate them, drill down and iterate the collection of orders and for each order, navigate through the collection of order details. Notice that you can also traverse back up from the order detail and get its associated Product and from there get the ProductName. This last is possible because you selected the Products table in the Reverse Mapping wizard tree, and so that information is automatically included.*

```
Shared Sub Main(args As String())
    Using scope As IObjectScope = ObjectScopeProvider1.ObjectScope()
        Dim query As Query(Of Customer) =
            scope.GetSqlQuery(Of Customer)("select * from Customers", "")
        Dim customers As QueryResultList(Of Customer) = query.ExecuteList()
        For Each customer As Customer In customers
            Console.WriteLine("Customer: {0}:", customer.CompanyName)
            For Each order As Order In customer.Orders
                Console.WriteLine(vbTab & "Order: {0}:", order.OrderID)
                For Each orderDetail As OrderDetail In order.OrderDetails
                    Console.WriteLine(vbTab & vbTab & "Product: {0}:", _
                        orderDetail.Product.ProductName)
                Next
            Next
        Next
    End Using

    Console.ReadLine()
End Sub
```

```
using System;
using Telerik.OpenAccess;

namespace _1_ORM_Reverse_Mapping
{
    class Program
    {
        static void Main(string[] args)
        {
            using (IObjectScope scope = ObjectScopeProvider1.ObjectScope())
            {
                Query<Customer> query =
                    scope.GetSqlQuery<Customer>("select * from Customers", "");
                QueryResultList<Customer> customers = query.ExecuteList();
                foreach (Customer customer in customers)
                {
                    Console.WriteLine("Customer: {0}:", customer.CompanyName);
                    foreach (Order order in customer.Orders)
                    {
                        Console.WriteLine("\tOrder: {0}:", order.OrderID);
                        foreach (OrderDetail orderDetail in order.OrderDetails)
                        {
                            Console.WriteLine("\t\tProduct: {0}:", orderDetail.Product.ProductName);
                        }
                    }
                }
                Console.ReadLine();
            }
        }
    }
}
```

- 11) Run the application. The complete list of customers, orders and products will be dumped to the console window.

 *Notice that we didn't have to explicitly join the orders, details or product information. We asked for the top-level data and everything else came along for the ride.*

2.3.1 Generated Class

Partial Classes

When OpenAccess generates a class by reverse mapping from the database, a partial class is created with two files that define the object. The screenshot below shows a customer object defined in the two class files.

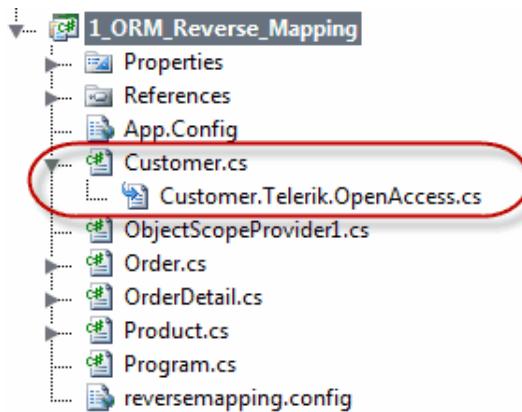


Figure 23

The partial class files that represent the object are:

- The main "Customer.cs" class represents the developer modifiable code and contains public properties for the class. By design, once this code is initially generated, OpenAccess never modifies this class file for fear of overwriting your code. The figure below shows the general appearance of the file:

```
public partial class Customer
{
    //The 'no-args' constructor required by OpenAccess.
    public Customer()
    {
    }

    [Telerik.OpenAccess.FieldAlias("customerID")]
    public string CustomerID
    {
        get { return customerID; }
        set { this.customerID = value; }
    }
}
```

Figure 24

💡 If a change is made (such as adding a new field to a table in the database where a class file has already been generated, then to utilize that changed field, code will have to be manually added to the main class file. This is by design to prevent custom code from being overwritten if a table is updated and the class regenerated. Fortunately as shown below, the changed code is in the other class file so it can be easily pasted into the main file if needed.

- The other class file has the naming convention "Customer.Telerik.OpenAccess.cs" and can be overwritten multiple times. This file contains the private members of the class.

```
[Telerik.OpenAccess.Persistent(
    IdentityField = "customerID")]
public partial class Customer
{
    private string customerID; // pk

    private string address;

    private bool? bool1;

    private string city;
```

Figure 25

The file also contains a commented out section at the bottom of the file with a #region block labeled "main class file contents". While OpenAccess doesn't automatically write this in your main class, you can copy this section back into the main class to synchronize the two files.

```
#region main class file contents
/*
using System;
using System.Collections.Generic;

namespace _12_Caching
{
    //Generated by Telerik OpenAccess
    public partial class Customer
    {
        //The 'no-args' constructor
        //required by OpenAccess.
        public Customer()
        {
        }
}
```

Figure 26

- G** As of this writing, the partial class that is regenerated when rerunning the Reverse Mapping Wizard only has the the changed code (shown above) that can be copied back into the main class file when using C#. In VB.NET this functionality is not currently available. When making changes to the fields in the database and re-running the mapping wizard, the code in the main class will need to be added by hand.
- a**
- !**
- ②**

Templates

Be aware that when OpenAccess generates a class file it uses a template to define the layout of the code. The generated class includes a comment that gives the path to the template file. You can change the template to make the generated code better fit your company standards and requirements. The following "PartialUserDefault.vm" is the template for the main code in a partial class:

```
using System;
using System.Collections.Generic;
#foreach( $import in $imports )${import}#end
#if(${hasPackage})namespace ${classPackage}
#endregion
// Generated by Telerik OpenAccess
// Used template: $usedTemplate
// NOTE: Field declarations and 'Object ID' class
// implementation are added to the 'designer' file.
// Changes made to the 'designer' file will
// be overwritten by the wizard.
public partial class $className
{
    //The 'no-args' constructor required by OpenAccess.
    public $className()
    {
    }

#foreach( $variableAccess in $allVariableAccess )$variableAccess #end
}
#endif($hasPackage))#end
```

See the topic "Using Reverse Engineering Templates" in the programming guide for a complete list of template files and available variables.

G It is possible to change everything in the templates, however please keep in mind that this could
o result in the code not being compilable any more. Also some variables should be there, for e.g., a
t field without a type and a name variable would be incomplete. You need to keep these things in
c mind before making changes to the templates.

h

a

!

①

2.4 Create, Read, Update, Delete (CRUD)

You're not "cooking with gas" until you can persist changes to the database. To update data using OpenAccess, you only need to modify the state of an object, you don't need to know any of the SQL performed on the back end. If your database happens to be Oracle or MS SQL or Firebird, the programming model does not change. The basic CRUD operations from a .NET and OpenAccess perspective are:

- **Create:** Instantiate a .NET object and add it to the object scope.
- **Read:** You have several alternatives here: use the scope Extent<>() method to return a LINQ queryable object, use a standard SQL query, or use an OQL query. OQL is similar to standard SQL but is not database specific.

- **Update:** Once you have a reference to a persistent object, you simply make changes to the object state and add the object to the scope.
- **Delete:** Pass the object to be deleted to the scope Remove() method.

This next example demonstrates each of the CRUD operations using the NorthwindOA database.

- 1) Follow the Steps in the "Reverse Mapping" project and create a new project called 1 ORM_Crud_Operations.
- 2) Replace the Main() method with the code shown below.

In the forward mapping example you created a new Customer object by instantiating and adding it to the object scope. In this example we also use LINQ to retrieve and modify the first customer. We also use LINQ to get all customers with a CompanyName that starts with "E" and delete those records from the database.

- The deletion portion of the code example traverses collections of Customer/Order/OrderDetails and removes each. We can also modify the configuration file to allow cascading deletes.

```
Imports Telerik.OpenAccess

Module Module1

    Sub Main()
        Using scope As IObjectScope = ObjectScopeProvider1.ObjectScope()
            scope.Transaction.Begin()

            ' add a new customer
            Dim newCustomer As New Customer()
            newCustomer.CustomerID = "A" + System.DateTime.Now.ToString("hmsfff")
        )

        newCustomer.CompanyName = "New Customer"
        scope.Add(newCustomer)

        ' get first customer using LINQ and modify
        Dim firstCustomer As Customer = _
            scope.Extent(Of Customer)().Take(1).[Single]()
        firstCustomer.CompanyName = _
            "Company " + DateTime.Now.ToString("hmsfff")
        scope.Add(firstCustomer)

        ' use LINQ to retrieve customer ID's starting with 'E'
        Dim results = From c In scope.Extent(Of Customer)() _
            Where c.CompanyName.StartsWith("E") _
            Select c

        ' delete 'E' customers from the database
        For Each cust As Customer In results
            For Each order As Order In cust.Orders
                For Each detail As OrderDetail In order.OrderDetails
                    scope.Remove(detail)
                Next
                scope.Remove(order)
            Next
            scope.Remove(cust)
        Next

        scope.Transaction.Commit()
    End Using
    Console.ReadLine()
End Sub

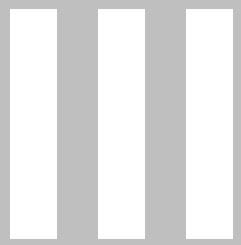
End Module
```

```
 1  using System;
 2  using System.Linq;
 3  using Telerik.OpenAccess;
 4
 5  namespace _1_ORM_CRUD_Operations
 6  {
 7      class Program
 8      {
 9          static void Main(string[] args)
10          {
11              using (IObjectScope scope = ObjectScopeProvider1.ObjectScope())
12              {
13                  scope.Transaction.Begin();
14
15                  // add a new customer
16                  Customer newCustomer = new Customer();
17                  newCustomer.CustomerID = "A" + System.DateTime.Now.ToString("hmsfff");
18                  newCustomer.CompanyName = "New Customer";
19                  scope.Add(newCustomer);
20
21                  // get first customer using LINQ and modify
22                  Customer firstCustomer =
23                      scope.Extent<Customer>().Take(1).Single();
24                  firstCustomer.CompanyName =
25                      "Company " + DateTime.Now.ToString("hmsfff");
26                  scope.Add(firstCustomer);
27
28                  // use LINQ to retrieve customer ID's starting with "E"
29                  var results =
30                      from c in scope.Extent<Customer>()
31                      where c.CompanyName.StartsWith("E")
32                      select c;
33
34                  // delete "E" customers from the database
35                  foreach (Customer cust in results)
36                  {
37                      foreach (Order order in cust.Orders)
38                      {
39                          foreach (OrderDetail detail in order.OrderDetails)
40                          {
41                              scope.Remove(detail);
42                          }
43                          scope.Remove(order);
44                      }
45                      scope.Remove(cust);
46                  }
47                  scope.Transaction.Commit();
48
49                  Console.ReadLine();
50              }
51          }
52      }
53  }
```

2.5 Wrap Up

This chapter provided basic skills for translating .NET objects into database tables and back again. You learned how to ORM-enable a project as a pre-requisite for any OpenAccess functionality. Then you transformed some basic .NET objects into database table form using forward mapping. As part of configuring a .NET object you decorated the class with the Persistent attribute. You were introduced to the ObjectScope class as the focal point for major OpenAccess functionality including handling database connections, transactions and retrieving data. You were introduced to the concept of "persistence by rechability". You used the OpenAccess Reverse mapping wizard to create persistent .NET objects using an existing database schema. You also performed basic (C)reate, (R)ead, (U)pdate and (D)elete operations within the context of a transaction.

Part



Design Environment

3 Design Environment

This chapter is a tour of the available tools that make working with OpenAccess easy. The tour includes tools accessed from Visual Studio menus, context menus and wizards.

In this chapter you will:

- Work with the Visual Studio Telerik OpenAccess menu.
- Become familiar with the Visual Studio Solution Explorer OpenAccess context menu.



Find the source projects for this chapter
at \Projects\ORM\CS\2_DesignEnvironment\2_DesignEnvironment.sln

3.1 OpenAccess Menu

The majority of OpenAccess functionality can be accessed through the Visual Studio menu. This will be your central control panel for ORM related activities.

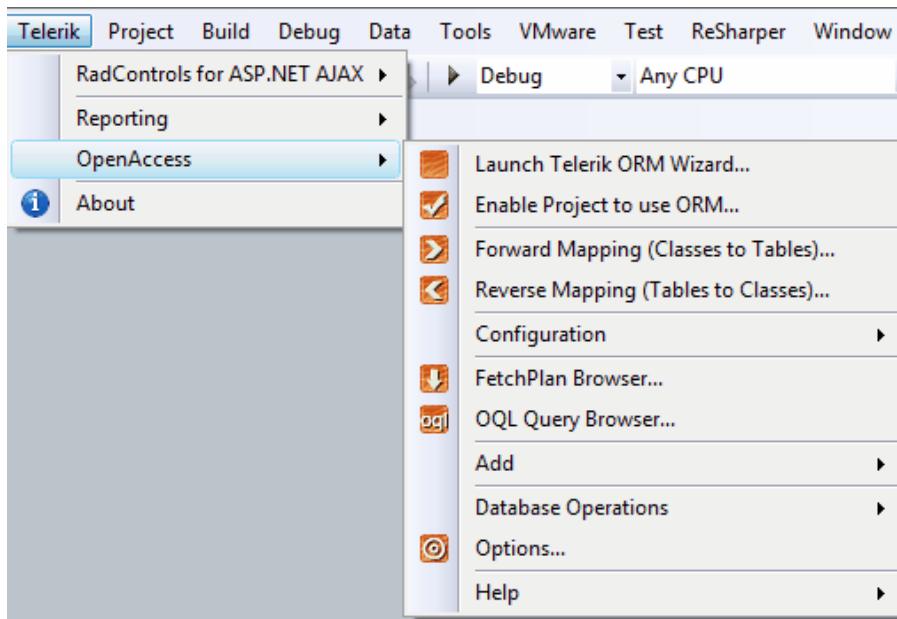


Figure 27

Use the menu options to...

- Launch the Telerik ORM Wizard. This is a central jumping off point that takes you through all phases of using ORM in your application. The ORM wizard is especially helpful if you're new to OpenAccess.
- Prepare your project for mapping.
- Forward map.
- Reverse map.
- Edit application configuration settings.
- Edit server configuration, including specifying the server backend, caching, connection details, how class names are generated and logging. Also validate and update settings.

- Browse and configure "fetch plans". Fetch plans let you tailor the most efficient retrieval from the server.
- Create and test OQL queries.
- Add data forms to your project. The OpenAccess Data Form Wizard lets you select the persistent object to use, the fields to display and the controls used to display the data.
- Use the Database Operations option to create a new database based off your connection settings or update an existing database. Classes marked with the Persistent attribute will have tables created. Additions and changes to persistent class properties show up as changes to table properties.
- Set Options for where "Persistent" and "Fetch Group" notations are found, i.e. as attributes or in the config file.

OpenAccess Menu Walk-Through

We've already used the menu to ORM-enable projects and to reverse map. In this next walk-through we will take a tour of menu options and use them to fine-tune existing projects and make changes on-the-fly.

- 1) Create a new Console application.
- 2) ORM-enable the project (See the Getting Started, "ORM Enable Project" section for detailed steps). Select both the Persistent Classes and the Data access code options. Set the Connection Id to "MyDatabaseConnection" and the Database Name to "MyDatabase".



- 3) Add a class "MyClass.cs" to the project. Add two properties to the class, "MyString" and "MyNumber" as shown in the class declaration below.

```
④ Public Class MyClass

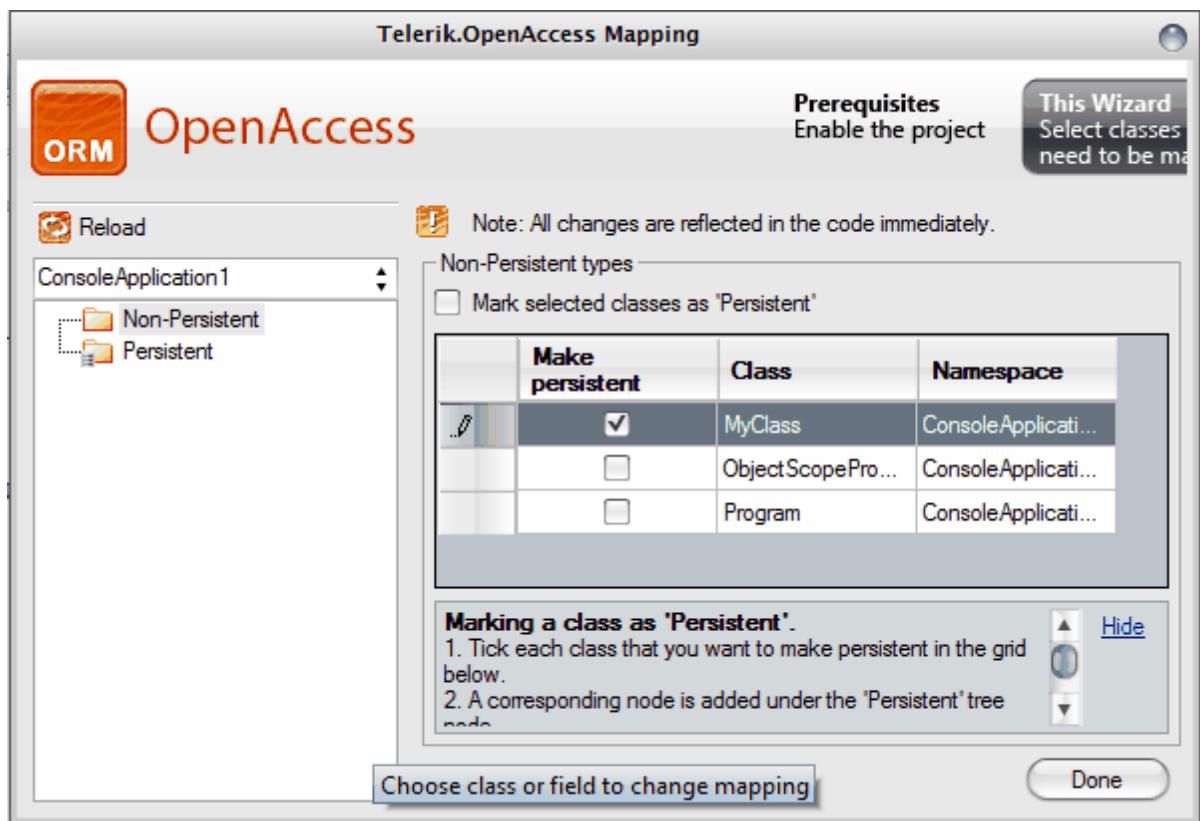
    Private _myString As String
    Private _myNumber As Integer

    Public Property MyString() As String
        Get
            Return _myString
        End Get
        Set(ByVal value As String)
            _myString = value
        End Set
    End Property
    Public Property MyNumber() As Integer
        Get
            Return _myNumber
        End Get
        Set(ByVal value As Integer)
            _myNumber = value
        End Set
    End Property
End Class
```

```
④ class MyClass
{
    public string MyString { get; set; }
    public int MyNumber { get; set; }
}
```

4) Forward map "MyClass".

- a) From the Visual Studio menu **Telerik > OpenAccess > Forward Mapping (Classes to Tables)**....
This step will display the Mapping Wizard.
- b) Make sure the "Non-Persistent" node of the tree is selected.
- c) Check the **Make Persistent** checkbox next to "MyClass".
- d) Check the **Mark selected classes as 'Persistent'** checkbox. *This last step will decorate "MyClass" with the Persistent attribute.*
- e) Click the **Done** button to close the dialog.



- 5) From the Visual Studio menu select **Telerik > OpenAccess > Database Operations > Create Database**. A dialog will display when the database is created.

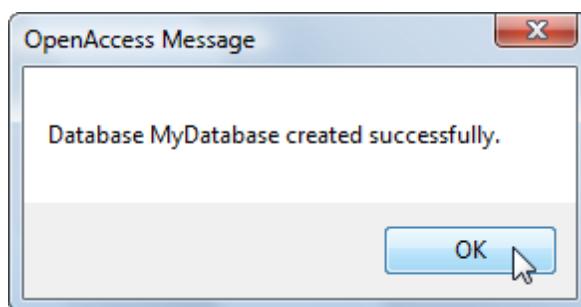


Figure 28

- 6) Also from the Visual Studio menu select **View > Server Explorer** and add a connection to your new database.
- Right-click the Data Connections node and select **Add Connection...** This step will display the Add Connection dialog.
 - In the Add Connection dialog, verify that the Data source is "Microsoft SQL Server (MS SQL)", and if it is not, click the **Change...** button and select "Microsoft SQL Server" from the list.
 - For the 'Server Name' enter ".\SQLEXPRESS".
 - For the 'Select or enter a database name' drop down the list and select "MyDatabase".
 - Click **OK** to close the dialog and return to Server Explorer.

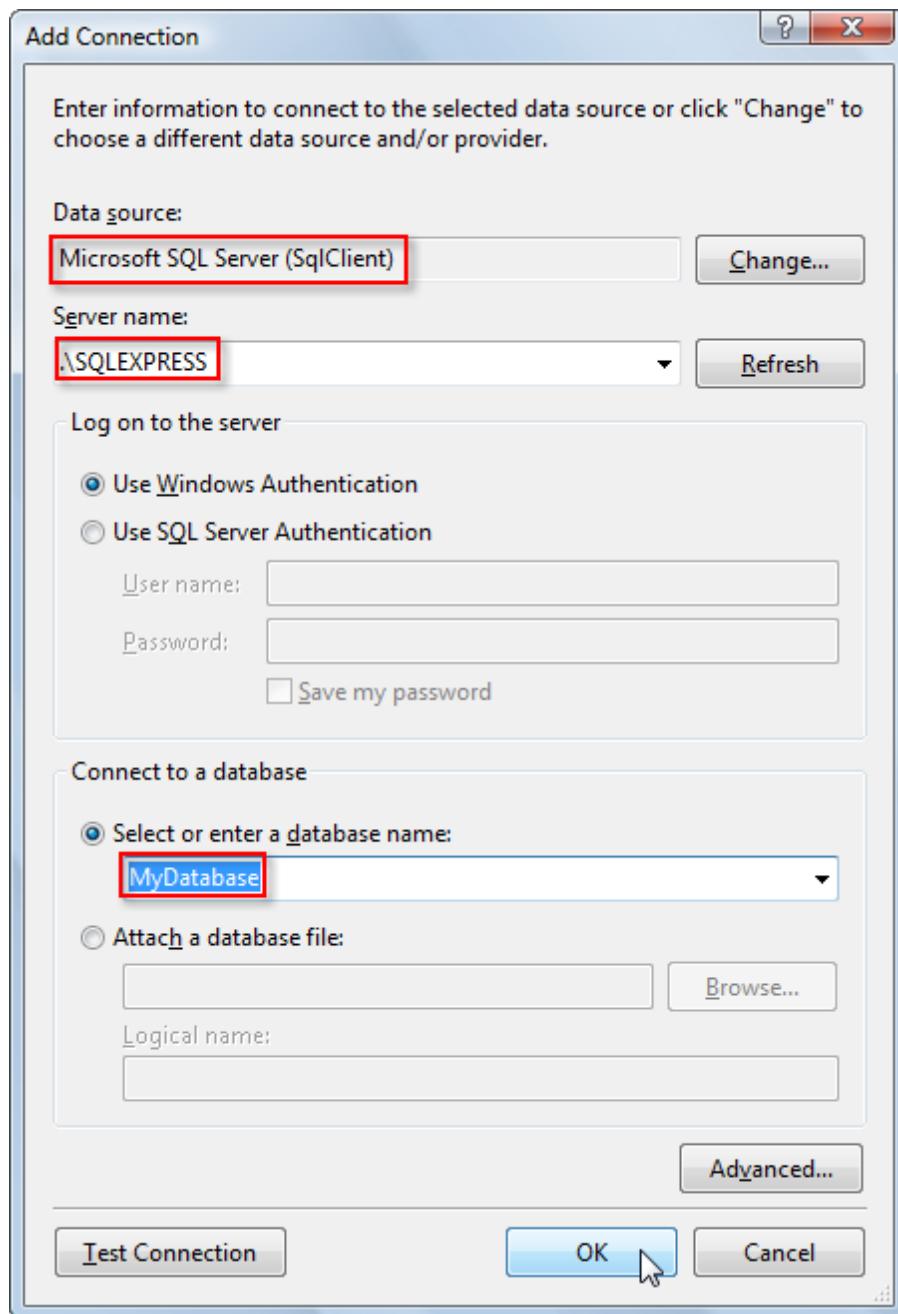


Figure 29

- 7) In Server Explorer expand and explore your new "MyDatabase" connection.
 - a) Expand the Tables node
 - b) Expand the node for the "my_class" table. The table will contain the automatically created id field, columns for "MyString" and "MyNumber", and a column for versioning.

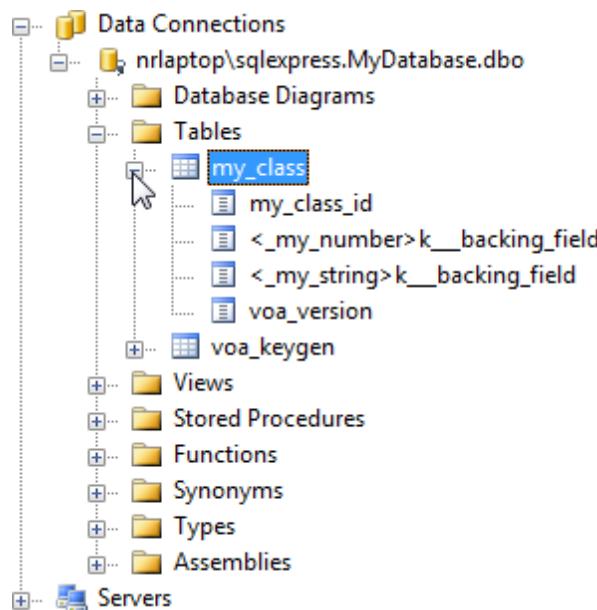


Figure 30

- c) Select the "_my_number" column node and look at the Properties Window. Notice that the Data Type property is "int".

Properties	
<_my_number>k__backing_field Column	
(Name)	<_my_number>k__backing_field
Data Type	int
Identity Increment	0
Identity Seed	0
Is Identity	False
Length	4
Nullable	False
Precision	10
Scale	0

Figure 31

- 8) Go back to the MyClass definition and change the MyNumber property from an "int" to a "double" type.
 9) Build the project. *This step will actually update the database structure, assuming that configuration settings allow it.*
 10) In the Server Explorer, right-click the "my_class" node and select the Refresh option from the context menu.
 11) Select the "_my_number" column node again and look at the Properties Window. Notice that the Data Type property is now "double".

What if you want to make sure that your database schema doesn't change accidentally? OpenAccess configuration options let you prevent database update.

- 12)Select **Telerik > OpenAccess > Configuration > Connection Settings**. The Enable Project (Expert Mode) dialog displays. The dialog allows you to change the database connection information, change project level properties and generate helper classes.

The "Enable Project (Expert Mode)" is the "expert" version of a dialog you used earlier to "ORM-Enable" the project. Buttons on the dialog allow you to toggle between "Wizard" and "Expert" modes.

- 13)Locate the "Update Database" property and set it to False.

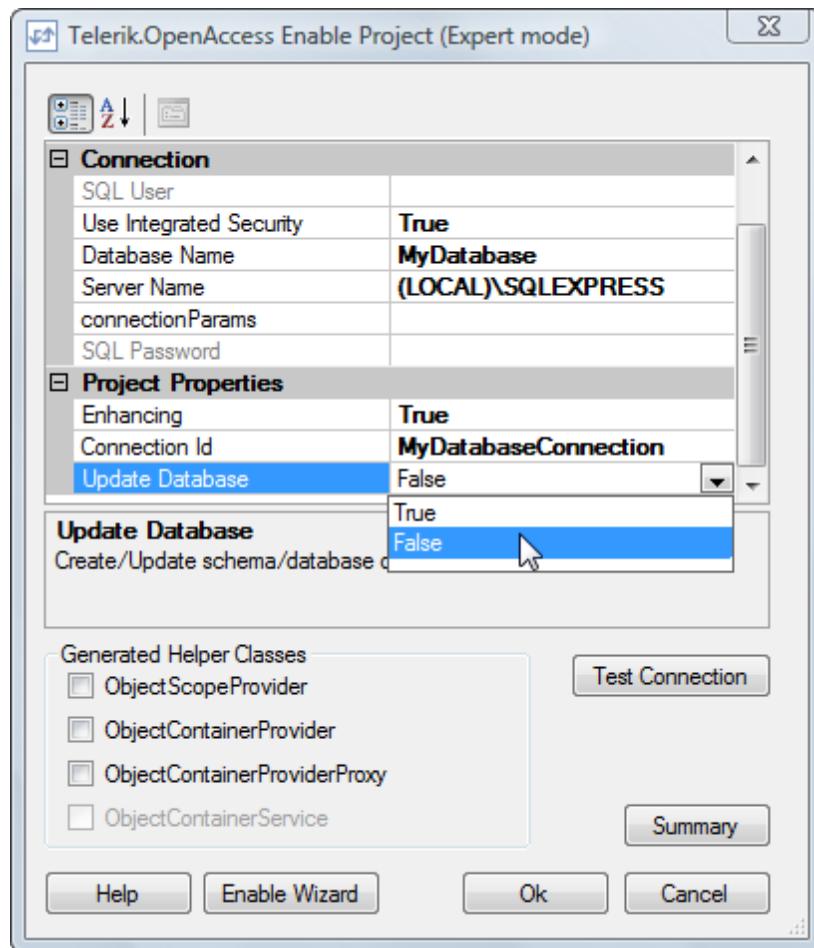


Figure 32

- 14)From the Visual Studio menu select **Telerik > OpenAccess > Database Operations > Create Database**. This will trigger an exception and prevent the database update.

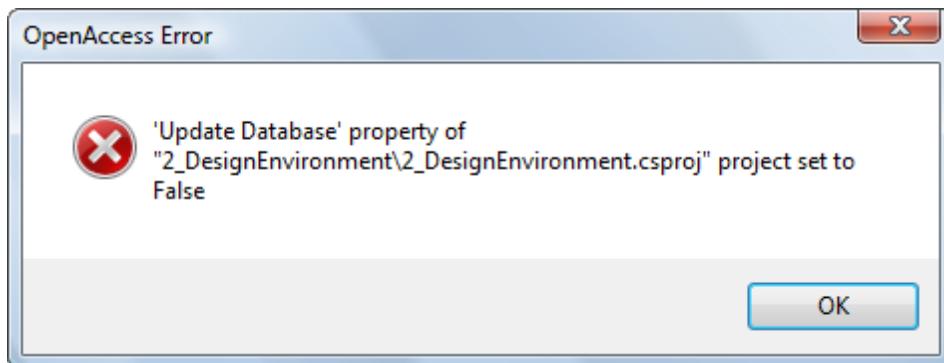


Figure 33

- 15)Now go back to the Select **Telerik > OpenAccess > Configuration > Connection Settings** option and change the Update Database option back to True.
- 16)Open the Program class for editing.
- 17)Add a reference to Telerik.OpenAccess in "Imports" (VB) or "using" (C#) section of the code.
- 18)Add code to the Main() method to create a single MyClass instance and commit it to the database.

```
Shared Sub Main(args As String())
    Dim scope As IObjectScope = ObjectScopeProvider1.ObjectScope()
    scope.Transaction.Begin()
    Dim [myClass] As New [MyClass]()
    [myClass].MyString = "Howdy World"
    [myClass].MyNumber = 123
    scope.Add([myClass])
    scope.Transaction.Commit()

    Console.ReadLine()
End Sub
```

```
static void Main(string[] args)
{
    IObjectScope scope = ObjectScopeProvider1.ObjectScope();
    scope.Transaction.Begin();
    MyClass myClass = new MyClass();
    myClass.MyString = "Howdy World";
    myClass.MyNumber = 123;
    scope.Add(myClass);
    scope.Transaction.Commit();

    Console.ReadLine();
}
```

Before we run the code, let's turn on logging and find out everything that's happening on the database backend.

- 19)Select the **Telerik > OpenAccess > Configuration > Backend Configuration Settings...** option to display the Backend Configuration dialog. Locate the Logging section and change the following properties:

- a) Event Tracing = True
- b) Log Level = Normal
- c) Write Log Output to Console = True

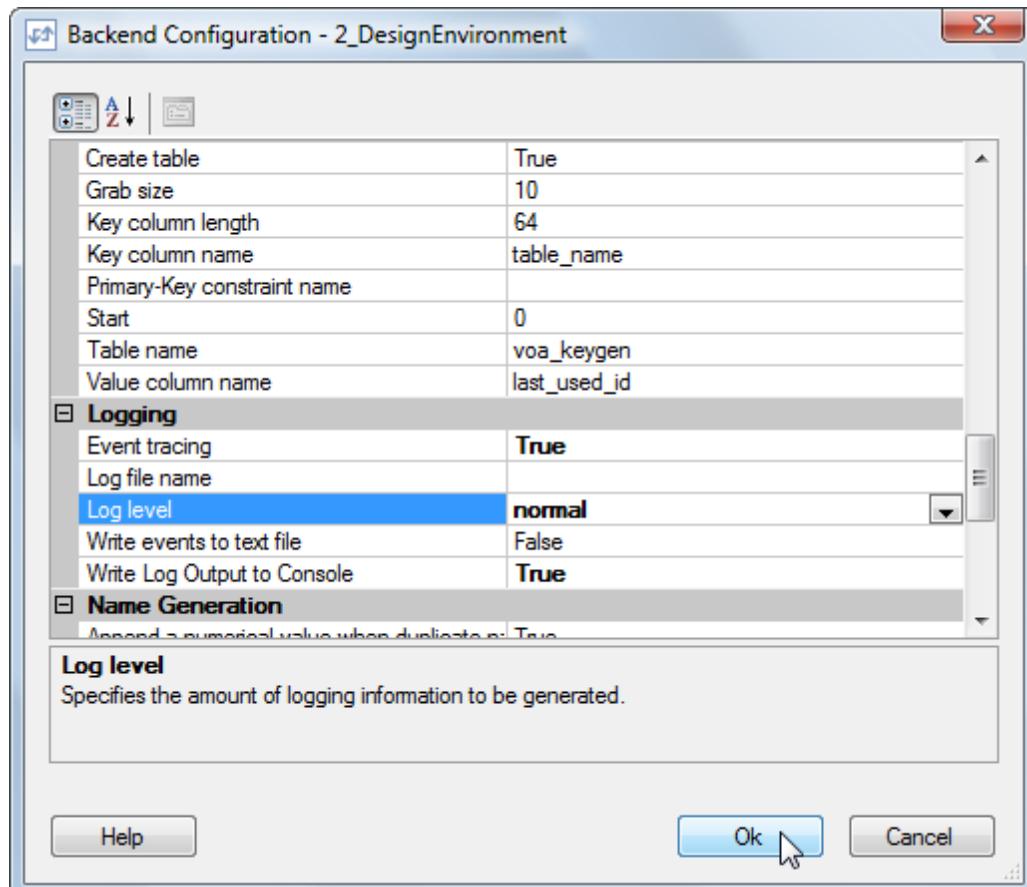
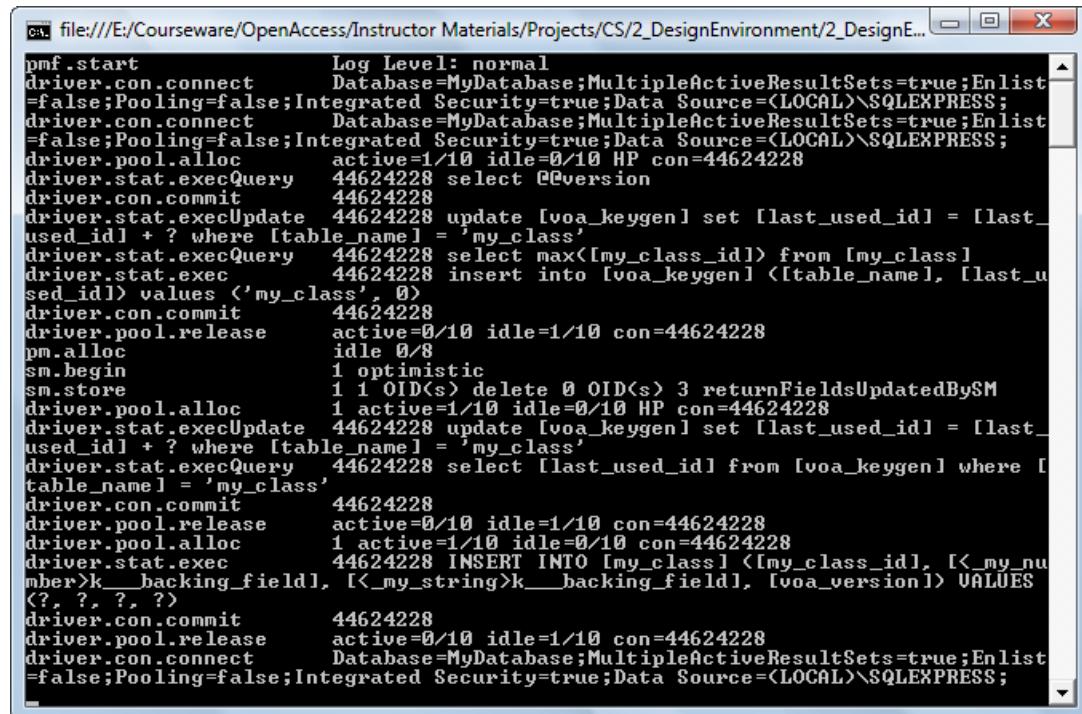


Figure 34

20) Run the application and take a look at the Console window. Using the logging to the console window or the same information redirected to a file, we can see what connection string is being used, how SQL statements are being generated and what kind of additional housekeeping OpenAccess is doing for us in the background.



The screenshot shows a Windows command-line window with a blue title bar containing the text "file:///E:/Courseware/OpenAccess/Instructor Materials/Projects/CS/2_DesignEnvironment/2_DesignE...". The main area of the window displays a log of database operations. The log includes various SQL statements such as "select @@version", "update [voa_keygen] set [last_used_id] = [last_used_id] + ? where [table_name] = 'my_class'", "insert into [voa_keygen] ([table_name], [last_u...ed_id]) values ('my_class', 0)", and "INSERT INTO [my_class] ([my_class_id], [<_my_nu...ber>k__backing_field], [<_my_string>k__backing_field], [voa_version]) VALUES (?, ?, ?, ?)". The log also shows driver-related commands like "pmf.start", "driver.con.connect", "driver.stat.execQuery", and "driver.con.commit". The log is timestamped with dates like "2010-06-15 10:45:00" and times like "10:45:00 AM". The log ends with "Driver disposed".

Figure 35

- You may have noticed when you selected the Connection Settings menu item to set the Update Database property that the Project Properties also include an "Enhancing" property. This is an important feature of OpenAccess that happens completely in the background. When mapping between .NET objects and database tables, you don't see a lot of generated code that handles communication with the database. The design reason for that is that generated code could be broken by later modification. Another reason is that you as the developer don't really need a lot of generated code, only the code that is business critical.

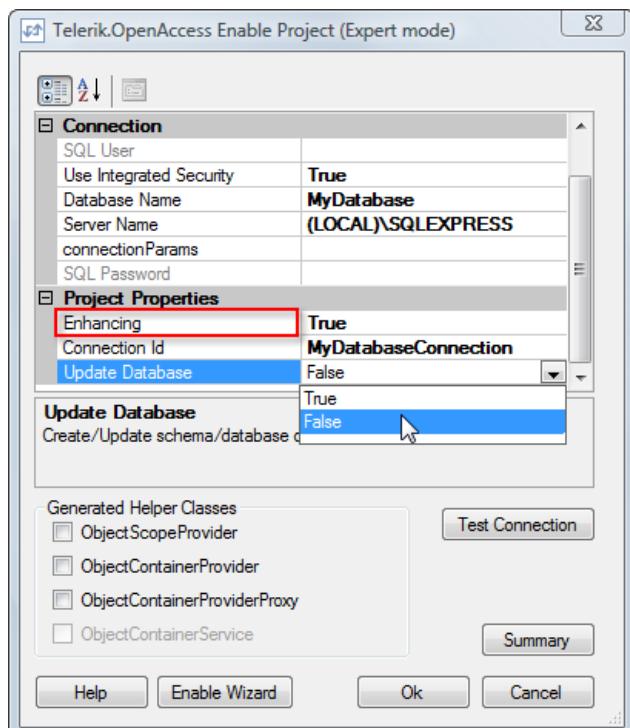


Figure 36

To get around these issues, OpenAccess "enhances" your assembly by injecting methods during the compilation process. If we open up Red Gate's ".NET Reflector" application to explore the assembly we see a series of OpenAccessEnhancedxxx methods have been added to handle the database communication plumbing.

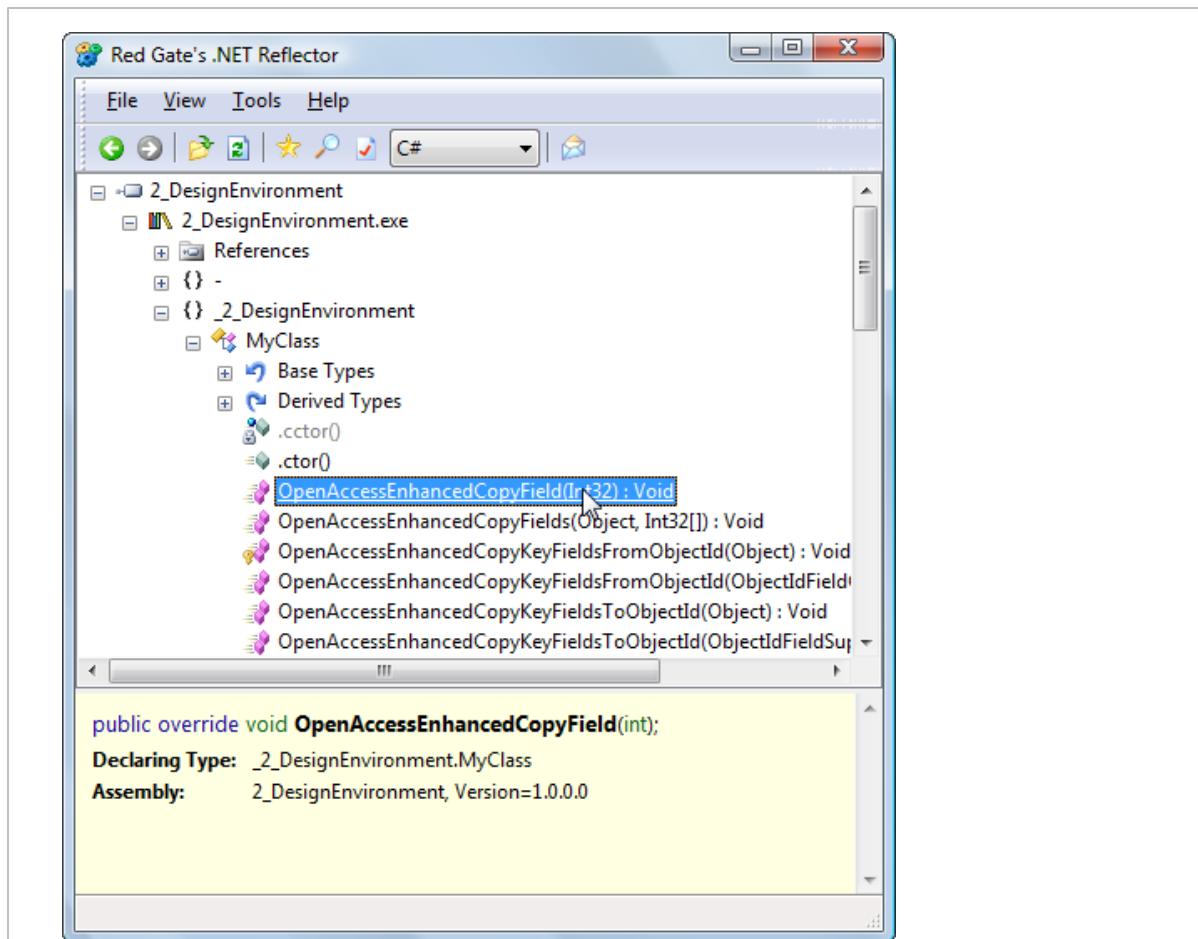


Figure 37

You should also be aware that OpenAccess includes a command-line tool, **VEnhance**, that performs the same 'enhancing' functionality in case you have a batch process where enhancing within Visual Studio would be impractical.

3.2 Project Context Menu

Some of the same options available from the main Telerik > OpenAccess menu can be found in the Solution Explorer when you right-click the project. here you can ORM-enable the project, display the OpenAccess Dataform Wizard, create a new scope provider object or update the application configuration references.

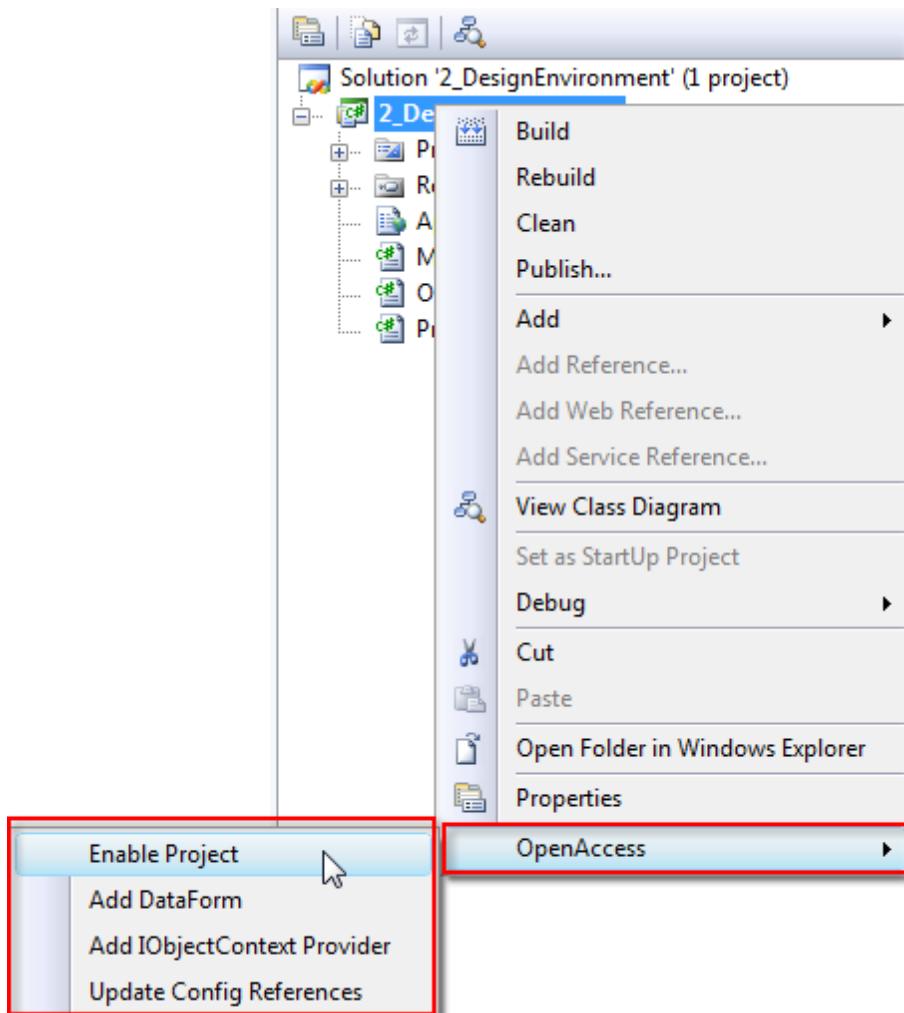


Figure 38

3.3 Wrap Up

This chapter toured some of the tools that make working with OpenAccess easy, mainly centered around the Visual Studio OpenAccess menu, the Solution Explorer context menu and some of the wizards available from these menus. You used OpenAccess menu items to ORM-enable your project, mark existing .NET classes as persistent, create a new database, prevent updating the database schema and turn on logging of server backend activities.

Part



IV

Using OpenAccess in Applications

4 Using OpenAccess in Applications

This chapter demonstrates how to build applications with several types of presentation using OpenAccess to provide the data.

In this chapter you will learn:

- How to build an assembly containing only the persistent objects and no data access code.
- How to build a WinForms application with grid and combo box that consumes data from OpenAccess..
- How to build an ASP.NET application with grid and combo box that consumes data from OpenAccess..
- How to build a Web Services application that consumes data from OpenAccess.
- How to build a Telerik Reporting module that consumes data from OpenAccess.
- How to construct N-Tier applications using OpenAccess.

4.1 Building the "Model" Assembly

In this walk-through we will create a class library that contains our persistent objects only. The applications that follow will reference this class library.



Find the source projects for this chapter at \Projects\ORM\<CS\VB>\Made_Easy\ORM_Projects.sln, project "2 ORM Model"

- 1) Create a new Class Library project called "2 ORM Model". This will be used in most of the following projects as well.
- 2) ORM-enable the project. Specify the following:
 - a) The Persistent classes option should be checked.
 - b) The Data Access Code option should not be checked.
 - c) The database connection ID should be "NorthwindOAConnection"
 - d) The database should be "NorthwindOA".



Feel free to use either of the "Connection Strings" options. Because the standard connection string option was used previously in previous chapter under "Reverse Mapping) the standard connection will likely already show up and will be configured to use the NorthwindOA database.

* Please review the previous two chapters if further clarification is needed to complete the above steps.

- 3) From the **Telerik > OpenAccess** menu, select the Reverse Mapping option.
- 4) In the Simple View tab of the Reverse Engineering Wizard, de-select the Generate checkbox for all but the "Categories" and "Products" tables.

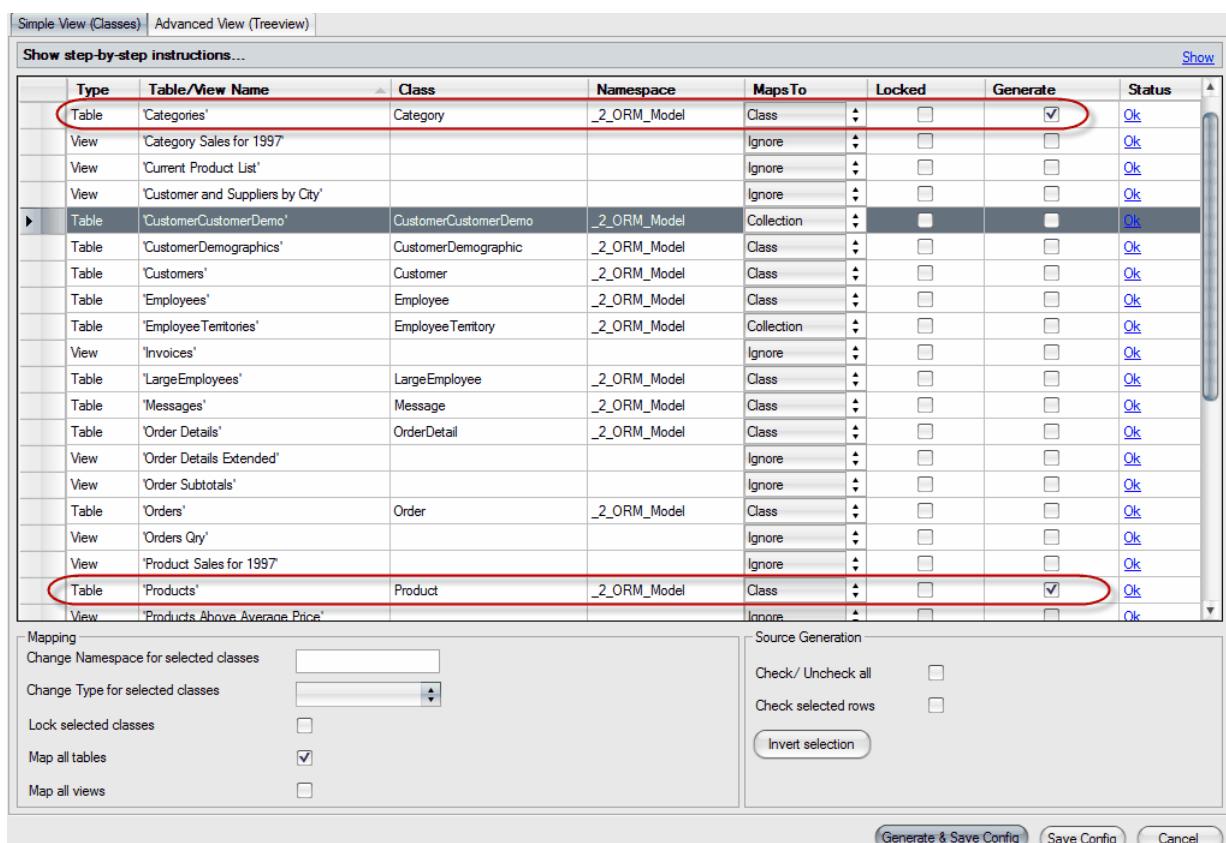


Figure 39

- 5) Select the Advanced View tab.
- 6) Expand the Tables node and Select the "Products" node.
- 7) Click the **Create Ref** button and configure the reference:
 - a) In the **Field Name** entry add "Category".
 - b) From the **Type** drop down list select "Category".
(Note: If using VB.NET the field name will automatically change to **_category**)
 - c) Click the **Create one-to-many list** checkbox.
- 8) Click the **Generate & Save Config** button.
- 9) Click **Yes** to confirm the Generate Sources dialog.
- 10)Build the project.

4.2 WinForms Example

In this project we will consume the persistent classes contained in the "2 ORM Model" class library, enable the project to access the database data, and bind a combo box and grid to the "Categories" and "Products" data.



Find the source projects for this chapter at \Projects\ORM\<CS\VB>\Made_Easy\ORM_Projects.sln, project "2 ORM MyWinFormApp"

- 1) Create a new Windows Forms Application project called "2 ORM MyWinFormApp".

- 2) ORM-enable the project. Specify the following:
 - a) The Persistent classes option should be disabled.
 - b) the Data Access Code option should be enabled.
 - c) The database connection ID should be "NorthwindOAConnection"
 - d) The database should be "NorthwindOA".
- 3) In the Solution Explorer, add a reference to the "2 ORM Model" class library.
- 4) In the Solution Explorer, rename "ObjectScopeProvider1.cs" to "NorthwindProvider.cs" (for VB change it to NortwhindProvider.vb). When prompted, click the **Yes** button to rename all references.
- 5) Build the project. You may see an OpenAccess error "No persistent class could be found". To fix this, navigate to the Solution Explorer, right-click the "MyWinFormApp" and select **Open Access > Update Config References** from the context menu.

G Why do I get an error even though the "2 ORM Model" assembly has been added as a reference?

- o** OpenAccess has its own set of consistency checks it performs during compilation. Although the project knows about the "Model" assembly, OpenAccess does not. Running the Update Config References option adds an assembly reference within the `<openaccess>` element of the config file, allowing the project to build correctly.

a

```
!   <openaccess xmlns="http://www.telerik.com/OpenAccess">
@     <references>
      <reference assemblyname="2 ORM Model" configrequired="True" />
    </references>
```

- 6) Build the application. The compilation should complete without errors.
- 7) From the Toolbox, drop a ObjectProvider component to the default windows form. This will display the ObjectProvider dialog. Configure the ObjectProvider as follows:
 - a) Context Provider = _2 ORM MyWinFormApp.NorthwindProvider, 2 ORM MyWinFormApp
 - b) Persistent Class = _2 ORM Model.Category
 - c) Result Method = Query
 - d) Query Condition = leave as default "SELECT * FROM CategoryExtent AS x". *This is an OQL query, the native SQL for OpenAccess.*
 - e) Click the **Finish** button to create the object provider "objectProvider1".

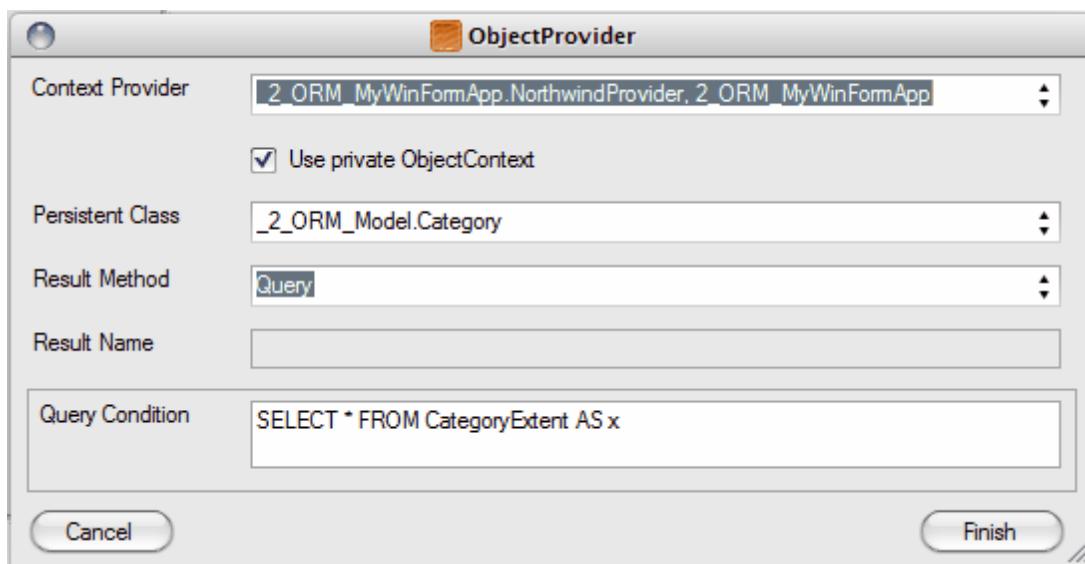


Figure 40

- 8) From the ToolBox, drag a ObjectView component to the form. This will display the ObjectView Wizard dialog. Configure the dialog as follows:
- Point the ObjectProvider at "objectProvider1" using the drop down list. *At this point in the example, only "objectProvider1" should be in the list, so it should already be selected.*
 - Leave the Root Type setting pointed at "_2 ORM Model.Category".
 - Click the **Finish** button to create "objectView1".

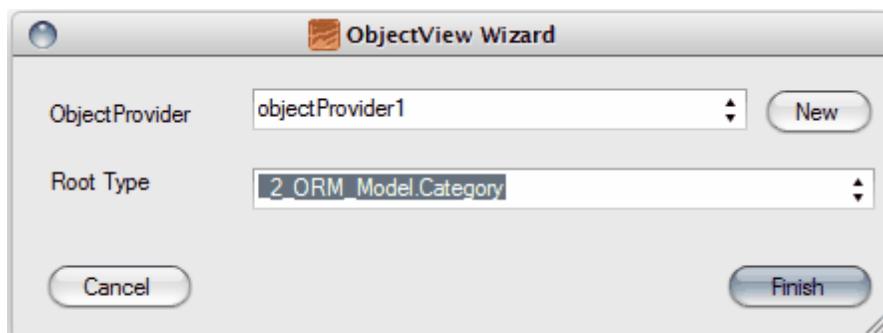


Figure 41

- 9) From the ToolBox, drag a standard Windows Forms ComboBox to the form and set properties:
- From the Properties Window, set the **Name** property to "cbCategories".
 - Set the **DataSource** property of "cbCategories" using the drop down list to select "objectView1"

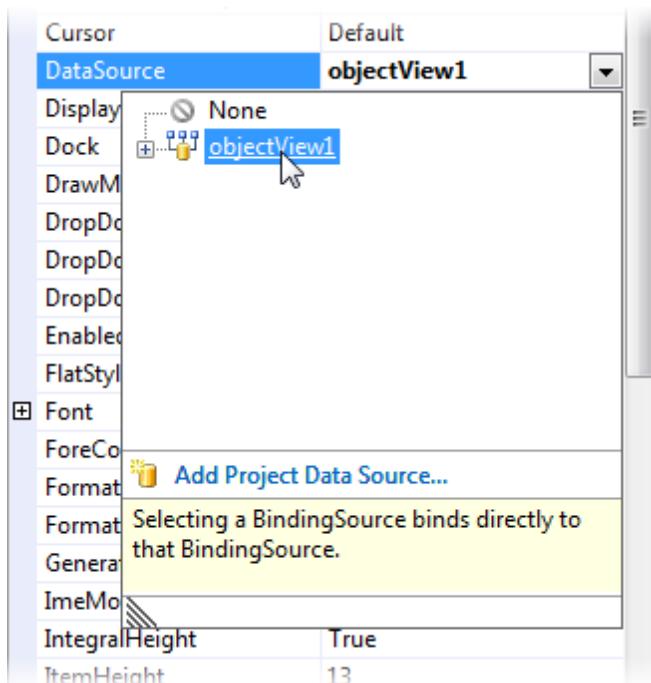


Figure 42

- c) Set the **DisplayMember** property to "CategoryName"

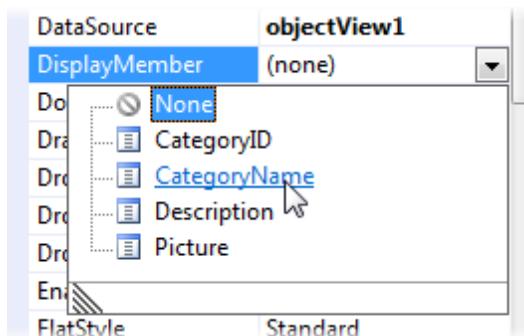


Figure 43

- d) Set the **ValueMember** property to "CategoryID".

10) Optionally, you can add a label "Category" next to the combo box.

11) Add a SelectedIndexChanged event handler for the ComboBox. *This will display the "CategoryID" in the form caption.*

```

Private Sub cbCategories_SelectedIndexChanged(ByVal sender As System.Object,
                                         ByVal e As System.EventArgs) Handles cbCategories.SelectedIndexChanged
    If TryCast(sender, ComboBox).SelectedValue <> Nothing Then
        Me.Text = TryCast(sender, ComboBox).SelectedValue.ToString()
    End If
End Sub

```

```
private void cbCategories_SelectedIndexChanged(
    object sender, EventArgs e)
{
    if ((sender as ComboBox).SelectedValue != null)
    {
        this.Text = (sender as ComboBox).SelectedValue.ToString();
    }
}
```

- 12) Set this project to be the "Startup Project" (note: in all of the following sections, ensure the proper project is the current startup project before running the application) and run the application. Drop down the list to see categories retrieved from the database. Click a category to see the CategoryID displayed in the form caption.

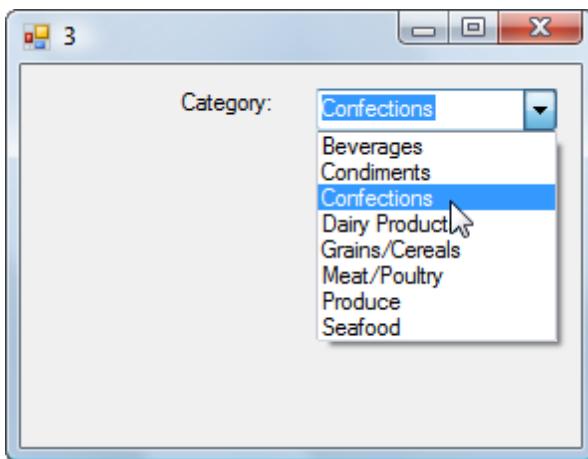


Figure 44

Next we will add a grid to display products filtered by the category combo box. Instead of using a ObjectProvider and ObjectView we will use LINQ instead to make filtering and reformatting the data an easy task.

- 13) From the ToolBox add a standard DataGridView to the form just below the category combo box. Set the **Name** property to "gvProducts".
- 14) Add Telerik.OpenAccess.Query to the "Imports" (VB) or "using" (C#) section of code to support LINQ.
- 15) Add _2 ORM Model to the "Imports" (VB) or "using" (C#) section of code to specify that the Reference to "Products" resolves to the proper namespace (there is a System.Deployment.Application.PlatformDetector.Products as well).
- 16) Change the code for the category combo box SelectedIndexChanged event handler. In the event handler, retrieve the selected "CategoryID" value. Then perform a LINQ query against the "scope" Extent<>() method, selecting only those products that have the same category id as the selected combo box value. The "select product" portion of the LINQ query indicates that we will use all the Product columns. Once you have the selection performed, call the ToList() method to return a generic list that can be bound to the grid.

```

■ Private Sub cbCategories_SelectedIndexChanged(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles cbCategories.SelectedIndexChanged

    If (TryCast(sender, ComboBox)).SelectedValue <> Nothing Then
        Dim scope As IObjectScope = NorthwindProvider.ObjectScope()

        Dim selectValue As System.Nullable(Of Integer) = _
            DirectCast(TryCast(sender, ComboBox)).SelectedValue, _
            System.Nullable(Of Integer))

        Dim filteredData = _
            From product In scope.Extent(Of Product)() _
            Where product.CategoryID = selectValue _
            Select product
        gvProducts.DataSource = filteredData.ToList()
    End If

End Sub

```

```

■ private void cbCategories_SelectedIndexChanged(
    object sender, EventArgs e)
{
    if ((sender as ComboBox).SelectedValue != null)
    {
        IObjectScope scope = NorthwindProvider.ObjectScope();

        int? selectValue = (int?) (sender as ComboBox).SelectedValue;
        var filteredData = from product in scope.Extent<Product>()
                           where product.CategoryID == selectValue
                           select product;
        gvProducts.DataSource = filteredData.ToList();
    }
}

```

G If you forget to add Telerik.OpenAccess.Query as a reference, the Extent<>() extension method
o will not be available. In fact, none of the LINQ extensions will show up in Intellisense and
t compilation will fail.
c
h
a
!
⑧

- 17) In the form's constructor, set the categories combo box SelectedIndex to the first item in the list. *This will cause the SelectedIndexChanged event to fire and the grid will be filtered against the first category in the list.*

```
④ Public Sub New()
    InitializeComponent()
    cbCategories.SelectedIndex = 0
End Sub
```

```
④ public Form1()
{
    InitializeComponent();
    cbCategories.SelectedIndex = 0;
}
```

- 18) Run the application. Select categories from the list and observe the filtered list of bound products displayed in the DataGridView.

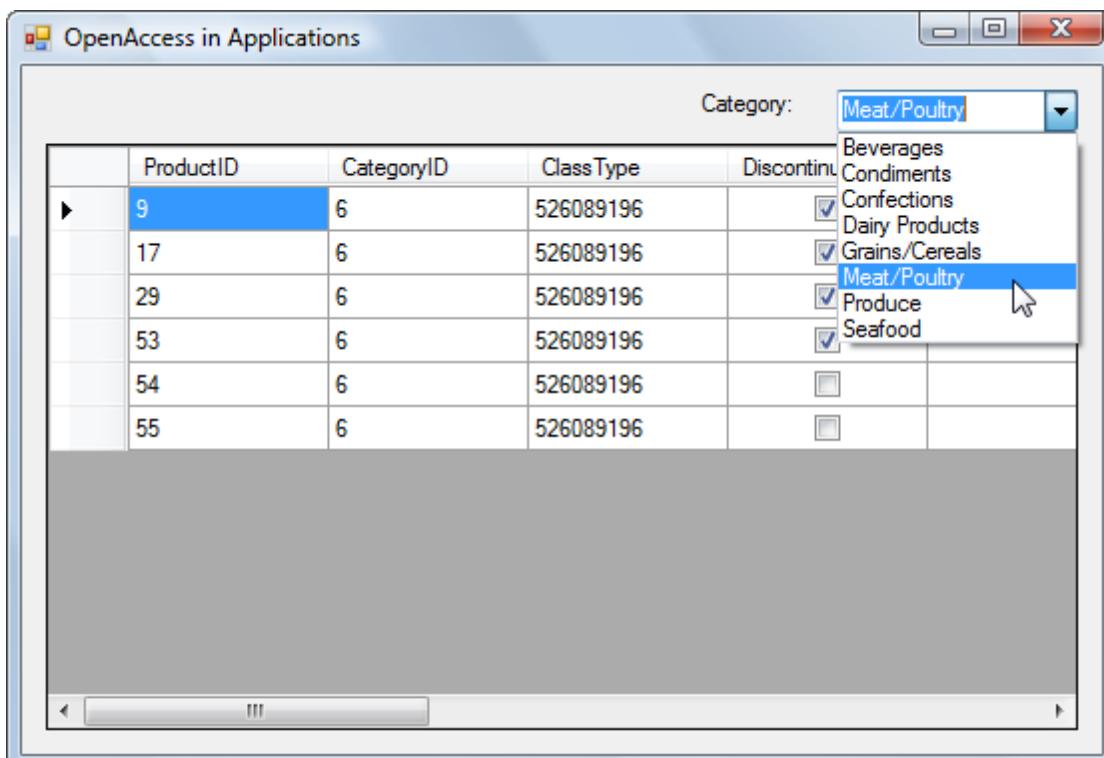


Figure 45

- 19) In this last example we get a number of columns we could do without. Using the LINQ "projection" capability we can construct a new column structure on-the-fly and fill our grid with it. We will show only the product name, category name and 'quantity per unit'. We're including category name here so you can see how a field from a subordinate object can be projected. Replace the SelectedIndexChanged event handling code with the example below.

```

Private Sub cbCategories_SelectedIndexChanged(ByVal sender As Object, _
                                         ByVal e As EventArgs)
    If (TryCast(sender, ComboBox)).SelectedValue IsNot Nothing Then
        Dim scope As IObjectScope = NorthwindProvider.ObjectScope()

        Dim selectValue As Nullable(Of Integer) = _
            CType((TryCast(sender, ComboBox)).SelectedValue, Nullable(Of Integer))

        Dim filteredData = _
            From product In scopeExtent(Of Product)() _
            Where product.CategoryID.Equals(selectValue) _
            Select _
                ProductName = product.ProductName, _
                QuantityPerUnit = product.QuantityPerUnit, _
                CategoryName = product.Category.CategoryName
        gvProducts.DataSource = filteredData.ToList()

    End If

End Sub
```

```

private void cbCategories_SelectedIndexChanged(object sender, EventArgs e)
{
    if ((sender as ComboBox).SelectedValue != null)
    {
        IObjectScope scope = NorthwindProvider.ObjectScope();

        int? selectValue = (int?) (sender as ComboBox).SelectedValue;
        var filteredData = from product in scopeExtent<Product>()
                           where product.CategoryID == selectValue
                           select new
                           {
                               Product = product.ProductName,
                               QuantityPerUnit = product.QuantityPerUnit,
                               Category = product.Category.CategoryName
                           };
        gvProducts.DataSource = filteredData.ToList();
    }
}
```

Notice that instead of "select product", there is a select clause that lists the columns we want returned and optionally lets you rename the columns. In particular, you should notice how the "Category" actually uses the Product Category property to get at the CategoryName.

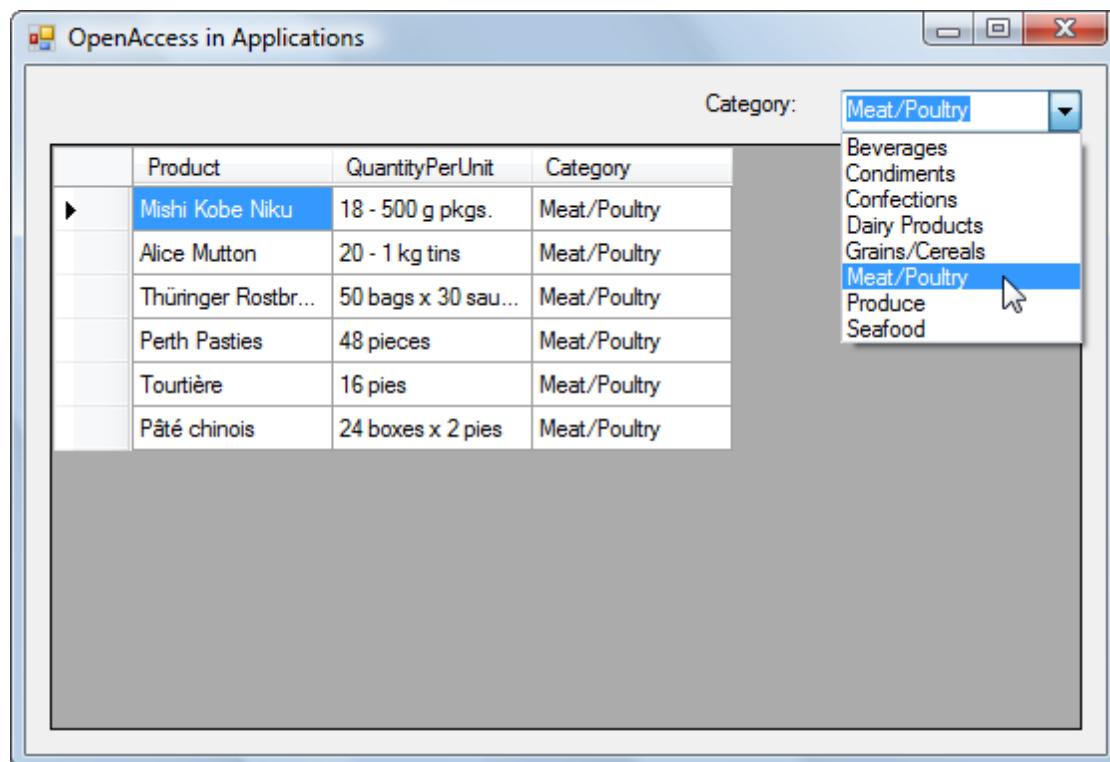


Figure 46

- G** Using projection works only if you want to view the grid in read-only mode. The reason: using **o** projection results in a collection of anonymous types. Because anonymous types are immutable, **t** the grid will open in read-only mode. To get the same columns but still allow update, bind to **c** collections of actual objects and hide unused columns.

h**a****!****⑥**

Using OpenAccess with RadControls for WinForms



Find the source projects for this chapter at \Projects\ORM\<CS\VB>\Made_Easy\ORM_Projects.sln, project "2 ORM_MyWinFormAppRad"

- 1) In the Solution Explorer, add references to **Telerik.WinForms** and **Telerik.WinForms.UI** to the project.
- 2) In the code-behind for the form, set the Inheritance class from "Form" to "RadForm". This will allow us to style the form in concert with the other controls.

```
Public Partial Class Form1
    Inherits Telerik.WinForms.UI.RadForm
```

```
public partial class Form1 : Telerik.WinForms.UI.RadForm
```

- 3) Delete the **ComboBox** and **DataGridView** controls from the page.

- 4) From the ToolBox add a RadComboBox to the form. Set the **Name** property to "cbCategories".
 - a) Set the **DataSource** property of "cbCategories" using the drop down list to select "objectView1"
 - b) Set the **DisplayMember** property to "CategoryName"
 - c) Set the **ValueMember** property to "CategoryID".
- 5) From the ToolBox drop a RadGridView control to the form. Set the **Name** property to "gvProducts".
- 6) From the ToolBox, drop a **BreezeTheme** component on the form.
- 7) Set the **ThemeName** of the form, RadComboBox and RadGridView to "Breeze".
- 8) In the constructor for the form, replace the code with the code below. *We will be displaying the grid in read-only mode, so set the AllowAddNewRow property to False. Using RadComboBox, the SelectedIndex will already be "0" during the form constructor, so work around this by first setting the SelectedIndex to "no selection", i.e. "-1", then to the first item in the list.*

```
Public Sub New()
    InitializeComponent()

    ' hide the 'Add new row' button
    gvProducts.MasterGridViewTemplate.AllowAddNewRow = False
    ' SelectedIndex is already '0', so work around by setting to
    ' 'no selection', i.e. '-1', then first item '0'
    cbCategories.SelectedIndex = -1
    cbCategories.SelectedIndex = 0

End Sub
```

```
public Form1()
{
    InitializeComponent();

    // hide the "Add new row" button
    gvProducts.MasterGridViewTemplate.AllowAddNewRow = false;
    // SelectedIndex is already "0", so work around by setting to
    // "no selection", i.e. "-1", then first item "0"
    cbCategories.SelectedIndex = -1;
    cbCategories.SelectedIndex = 0;
}
```

- 9) Create a new SelectedIndexChanged event handler for the RadComboBox and add the code below. *The code is substantially the same as the standard WinForms example shown previously but references the RadComboBox. At the end of the method we call the RadGrid MasterGridViewTemplate BestFitColumns() method to get the optimum layout for column headings and data..*

* You will need to also add the following Imports or Using Statements:

```
Telerik.OpenAccess
Telerik.WinForms.UI
_2 ORM Model
```

```
■ Private Sub cbCategories_SelectedIndexChanged(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles cbCategories.SelectedIndexChanged  
    If (TryCast(sender, RadComboBox)).SelectedValue <> Nothing Then  
        Dim scope As IObjectScope = NorthwindProvider.ObjectScope()  
  
        Dim selectValue As System.Nullable(Of Integer) = _  
            DirectCast((TryCast(sender, RadComboBox)).SelectedValue, _  
                System.Nullable(Of Integer))  
  
        Dim filteredData = _  
            From product In scopeExtent(Of Product) () _  
            Where product.CategoryID = selectValue  
  
        gvProducts.DataSource = filteredData.ToList()  
        gvProducts.MasterGridViewTemplate.BestFitColumns()  
  
    End If  
End Sub
```

```
■ private void cbCategories_SelectedIndexChanged(object sender, EventArgs e)  
{  
    if ((sender as RadComboBox).SelectedValue != null)  
    {  
        IObjectScope scope = NorthwindProvider.ObjectScope();  
  
        int? selectValue = (int?)(sender as RadComboBox).SelectedValue;  
        var filteredData = from product in scopeExtent<Product>()  
                           where product.CategoryID == selectValue  
                           select new  
                           {  
                               Product = product.ProductName,  
                               QuantityPerUnit = product.QuantityPerUnit,  
                               Category = product.Category.CategoryName  
                           };  
  
        gvProducts.DataSource = filteredData.ToList();  
        gvProducts.MasterGridViewTemplate.BestFitColumns();  
    }  
}
```

10) Run the application.

	Product	QuantityPerUnit	Category
▶	Chai	10 boxes x 25 bags	Beverages
	Chang	24 - 12 oz bottles	Beverages
	Guaraná Fantástica	12 - 355 ml cans	Beverages
	Sasquatch Ale	24 - 12 oz bottles	Beverages
	Steeleye Stout	24 - 12 oz bottles	Beverages
	Côte de Blaye	12 - 75 cl bottles	Beverages
	Chartreuse verte	950 cc per bottle	Beverages

Figure 47

CRUD Operations

To support CRUD (create, read, update, delete) operations on the Products grid and still have filter capability, we will use an ObjectView to work with both the Category and Product tables. We will create a LINQ query that selects products for a given category and assign the results to the ObjectView.DataSource. ObjectView descends from BindingSource and as such handles many background tasks, even creating an internal IBindingList implementation and populating it if no binding list exists.

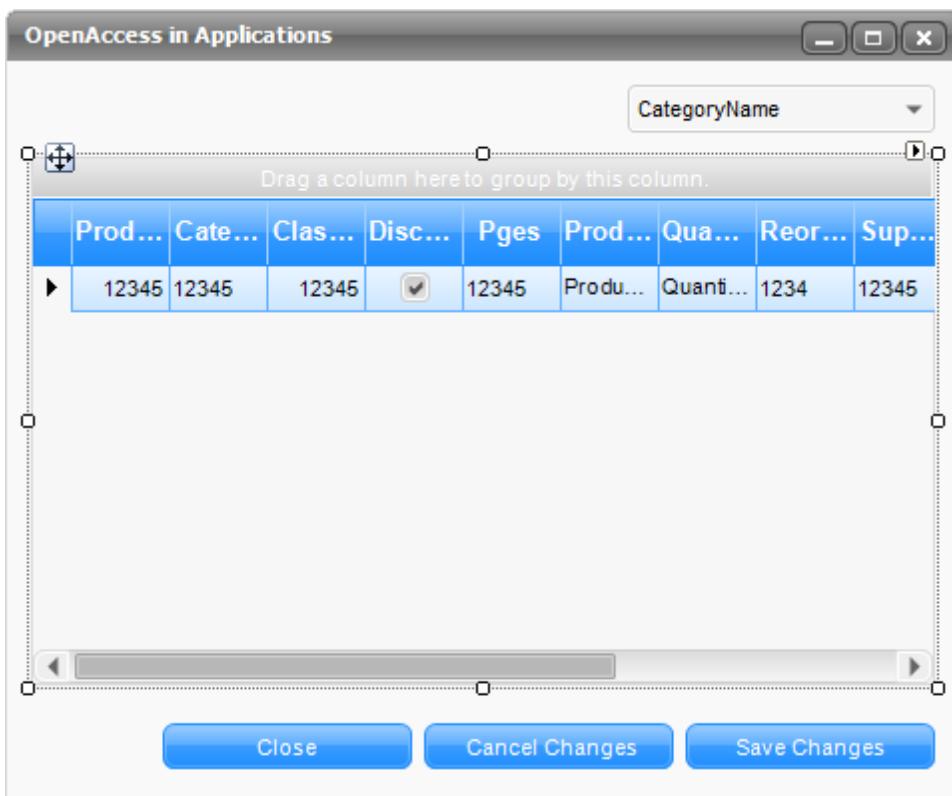


Find the source projects for this chapter at \Projects\ORM\<CS\VB>\Made_Easy\ORM_Projects.sln, project "2 ORM_MyWindFormCrud"

We will use a transaction to commit or rollback changes. Transaction properties are encapsulated with ObjectScope.TransactionProperties. The TransactionProperties.AutomaticBegin property when true automatically starts a new transaction when the previous transaction completes.

- 1) Create a new project "2 ORM_MyWindFormCrud".
- 2) ORM-enable the project with both DAL Access Code and Persistent classes checked and include the connection to "NorthwindOA".
- 3) Select the Visual Studio menu **Telerik > OpenAccess > Reverse Mapping (Tables to Classes)**. In the Forward Mapping Wizard include the "Categories" and "Products" tables and finish the Wizard. Rename the ObjectScopeProvider to "NorthwindProvider".
- 4) In the Solution Explorer add references to **Telerik.OpenAccess** and **Telerik.OpenAccess.Query**.
- 5) Using the Solution Explorer delete the default form from the project.
- 6) Using the Solution Explorer, right-click the project and select Add > New Item... > Telerik RadForm from the context menu.

- 7) Change the initial startup form to be "RadForm1". If you're working in VB.NET, select from the project properties sheet "Startup Form" drop down list. For C#, change the name in Program.cs manually.
- 8) Build the Application.
- 9) Add an ObjectProvider, Context Provider = "_2 ORM MyWindFormCrud.NorthwindProvider", Persistent Class = "_2 ORM MyWindFormCrud.Category", **Name** property = "opCategory".
- 10) Add an ObjectProvider, Context Provider = "_2 ORM MyWindFormCrud.NorthwindProvider", Persistent Class = "._2 ORM MyWindFormCrud.Product", **Name** property = "opProduct".
- 11) Add an ObjectView, ObjectProvider = "opCategory", Root Type = "_2 ORM MyWindFormCrud.Category", **Name** property = "ovCategory".
- 12) Add an ObjectView, ObjectProvider = "opProduct", Root Type = "_2 ORM MyWindFormCrud.Product", **Name** property = "ovProduct".
- 13) From the ToolBox add a standard Label and change the **Text** property to "Category:".
- 14) From the ToolBox add a RadComboBox and set properties:
 - a) **Name** = "cbCategories"
 - b) **DataSource** = "ovCategory"
 - c) **DisplayMember** = "CategoryName"
 - d) **ValueMember** = "CategoryID"
- 15) From the ToolBox add a RadGridView and set properties:
 - a) **Name** = "gvProducts"
 - b) **DataSource** = "ovProduct"
- 16) Drop three RadButtons to the bottom of the form and set properties:
 - a) **Name** = "btnClose", **Text** = "Close"
 - b) **Name** = "btnCancel", **Text** = "Cancel Changes"
 - c) **Name** = "btnSave", **Text** = "Save Changes"
- 17) From the ToolBox drop a **BreezeTheme** component on the form. Set all RadControls on the form including the form itself to use the "Breeze" theme. The form should look something like the example below:



- 18) Add System.Linq, Telerik.OpenAccess, Telerik.WinForms, and Telerik.WinControls to the Imports (VB) or using (C#) statements at top of the RadForm1 code-behind file.
- 19) In the code-behind add a private variable that holds the object scope.

```
Private _scope As IObjectScope = NorthwindProvider.ObjectScope()
```

```
private IObjectScope _scope = NorthwindProvider.ObjectScope();
```

- 20) Add code to the constructor. This is similar to previous examples except that the transaction property AutomaticBegin is set here and the AutoSizeColumnsMode is set to fill to make best use of screen real estate.

```
Public Sub New()
    InitializeComponent()

    cbCategories.SelectedIndex = -1
    cbCategories.SelectedIndex = 0

    _scope.TransactionProperties.AutomaticBegin = True
    gvProducts.MasterGridViewTemplate.AutoSizeColumnsMode =
        GridViewAutoSizeColumnsMode.Fill
End Sub
```

```
public RadForm1()
{
    InitializeComponent();

    cbCategories.SelectedIndex = -1;
    cbCategories.SelectedIndex = 0;

    _scope.TransactionProperties.AutomaticBegin = true;
    gvProducts.MasterGridViewTemplate.AutoSizeColumnsMode =
        GridViewAutoSizeColumnsMode.Fill;
}
```

21) Add a SelectedIndexChanged event handler and add the code below. *This code is also very similar to previous examples except that the result of the LINQ select is assigned not to the DataSource of the RadComboBox, but to the DataSource of the ObjectView. Also, two text box columns for the RadGridView are defined here.*

```
Private Sub cbCategories_SelectedIndexChanged(ByVal sender As System.Object,
                                         ByVal e As System.EventArgs) Handles cbCategories.SelectedIndexChanged

    If (TryCast(sender, RadComboBox)).SelectedValue <> Nothing Then

        Dim selectValue As System.Nullable(Of Integer) = _
        DirectCast((TryCast(sender, RadComboBox)).SelectedValue, _
        System.Nullable(Of Integer))
        ' select only products for this category and assign to the
        ' ObjectView DataSource
        opProduct.ObjectSource = From product In _scopeExtent(Of Product)()

        Where product.CategoryID = selectValue _
        Select product

        ' display only these three columns
        gvProducts.Columns.Clear()
        gvProducts.Columns.Add(New TextBoxColumn("CategoryID"))
        gvProducts.Columns.Add(New TextBoxColumn("ProductName"))
        gvProducts.Columns.Add(New TextBoxColumn("QuantityPerUnit"))

    End If
End Sub
```

```

    private void cbCategories_SelectedIndexChanged(object sender,
        EventArgs e)
    {
        if ((sender as RadComboBox).SelectedValue != null)
        {

            int? selectValue = (int?) (sender as RadComboBox).SelectedValue;
            // select only products for this category and assign to the
            // ObjectView DataSource
            opProduct.ObjectSource = from product in _scope.Extent<Product>()
                                      where product.CategoryID == selectValue
                                      select product;

            // display only these three columns
            gvProducts.Columns.Clear();
            gvProducts.Columns.Add(
                new GridViewTextBoxColumn("CategoryID"));
            gvProducts.Columns.Add(
                new GridViewTextBoxColumn("ProductName"));
            gvProducts.Columns.Add(
                new GridViewTextBoxColumn("QuantityPerUnit"));
        }
    }
}

```

22) Create Click event handlers for all three buttons and add the code below. *The Cancel and Save buttons call the scope's Transaction Commit() and Rollback() methods. The Close button simply calls the Close() method of the form.*

```

Private Sub btnCancel_Click(sender As Object, e As EventArgs)
    _scope.Transaction.Rollback()
End Sub

Private Sub btnSave_Click(sender As Object, e As EventArgs)
    opProduct.SaveAll()
    _scope.Transaction.Commit()
End Sub

Private Sub btnClose_Click(sender As Object, e As EventArgs)
    Me.Close()
End Sub

```

```
■ private void btnCancel_Click(object sender, EventArgs e)
{
    _scope.Transaction.Rollback();
}

private void btnSave_Click(object sender, EventArgs e)
{
    opProduct.SaveAll();
    _scope.Transaction.Commit();
}

private void btnClose_Click(object sender, EventArgs e)
{
    this.Close();
}
```

23) Add a **FormClosing** event handler and add the code shown below. *The scope's Transaction IsDirty property tells us that some data has been modified. A RadMessageBox displays a confirmation message asking if the changes should be committed or rolled back.*

```
■ Private Sub RadForm1_FormClosing(sender As Object, _
e As FormClosingEventArgs)
    RadMessageBox.SetThemeName(Me.ThemeName)

    If _scope.Transaction.IsDirty Then
        If RadMessageBox.Show("Save all changes?", _
        "Pending Changes", MessageBoxButtons.OKCancel) =
        DialogResult.OK Then
            opProduct.SaveAll()
            _scope.Transaction.Commit()
        Else
            _scope.Transaction.Rollback()
        End If
    End If
End Sub
```

```

private void RadForm1_FormClosing(object sender, FormClosingEventArgs e)
{
    RadMessageBox.SetThemeName(this.ThemeName);

    if (_scope.Transaction.IsDirty)
    {
        if (RadMessageBox.Show("Save all changes?", "Pending Changes",
            MessageBoxButtons.OKCancel) == DialogResult.OK)
        {
            opProduct.SaveAll();
            _scope.Transaction.Commit();
        }
        else
        {
            _scope.Transaction.Rollback();
        }
    }
}

```

- 24) Run the application. Make additions, deletions and modifications, save the changes and restart the application. Verify that changes persist and that the filtering by category works. Make changes to the data and close the form to make sure that the IsDirty flag works as expected.

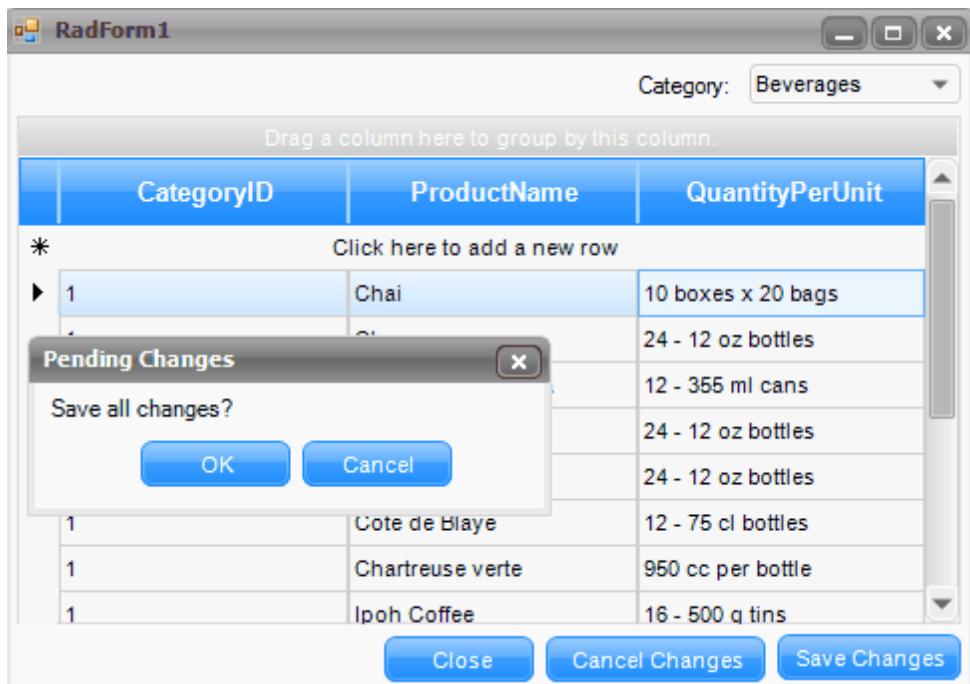


Figure 48

4.3 ASP.NET Example

In this project we will consume the persistent classes contained in the "2_ORM_Model" class library, enable the project to access the database data, and bind a combo box and grid to the "Categories" and

"Products" data.



Find the source projects for this chapter at \Projects\ORM\<CS\VB>\Made_Easy\ORM_Projects.sln, project "3 ORM MyWebApp"

- 1) Create a new ASP.NET Web Application project called "3 ORM MyWebApp".
- 2) ORM-enable the project. Specify the following:
 - a) The Persistent classes option should be disabled.
 - b) the Data Access Code option should be enabled.
 - c) In the Dropdown for Select Connection, select the "2 ORM Model \ NorthwindConnection"
- 3) In the Solution Explorer, rename "ObjectScopeProvider1.cs or ObjectScopeProvider1.vb" to "NorthwindProvider.cs or NorthwindProvider.vb". When prompted, click the **Yes** button to rename all references.
- 4) In the Solution Explorer, add a reference to the "2 ORM Model" class library.
- 5) Navigate to the Solution Explorer, right-click the "_3 ORM MyWebApp" and select **Open Access > Update Config References** from the context menu.
- 6) From the ToolBox drop a **OpenAccessDataSource**, a standard **DropDownList** and a **GridView** to the default page and set properties:
 - a) Set the DropDownList **ID** property to "ddlCategory" and set the **AutoPostBack** property to True.
 - b) Set the GridView **ID** property to "gvProducts".
- 7) Build the Project (This allows ORM to see the new Data Context for this Project)
- 8) Click the OpenAccessDataSource Smart Tag and select **Configure Data Source...** This will display the OpenAccess Data Source Wizard. On the first page of the wizard, select the "_3 ORM MyWebApp. NorthwindProvider, 3 ORM MyWebApp" from the drop down list. Click the **Next** button. On the second page, Select "_2 ORM Model.Category" from the drop down list. Click **Finish** to close the Wizard.
- 9) In the code-behind for the default form, add references to _2 ORM Model, Telerik.OpenAccess and Telerik.OpenAccess.Query to the "Imports" (VB) or "using" (C#) section of code.
- 10) Add properties and methods to handle the object scope. *The Scope property holds the IObjectScope reference throughout the life of the page and is disposed along with the page.*

```

■ Private _scope As IObjectScope = NorthwindProvider.GetNewObjectScope()

Public Property Scope() As IObjectScope
    Get
        Return _scope
    End Get
    Set(ByVal value As IObjectScope)
        _scope = value
    End Set
End Property

Public Overloads Overrides Sub Dispose()
    scope.Dispose()
    MyBase.Dispose()
End Sub

```

```

■ private IObjectScope scope = NorthwindProvider.GetNewObjectScope();

public IObjectScope Scope
{
    get { return scope; }
    set { scope = value; }
}

public override void Dispose()
{
    scope.Dispose();
    base.Dispose();
}

```

11) In the Page_Load event handler, assign properties for the DropDownList:

- Assign the OpenAccessDataSource to the DataSource property.
- Assign "CategoryName" to the DataTextField.
- Assign "CategoryID" to the DataValueField.
- Call the DropDownList DataBind() method.

```

■ Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
    If Not IsPostBack Then
        ddlCategory.DataSource = OpenAccessDataSource1
        ddlCategory.DataTextField = "CategoryName"
        ddlCategory.DataValueField = "CategoryID"
        ddlCategory.DataBind()
    End If
End Sub

```

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        ddlCategory.DataSource = OpenAccessDataSource1;
        ddlCategory.DataTextField = "CategoryName";
        ddlCategory.DataValueField = "CategoryID";
        ddlCategory.DataBind();
    }
}
```

- 12) Create a SelectedIndexChanged event handler for the DropDownList and add code to get the selected CategoryID, filter the Products data on CategoryID, assign the filtered data to the GridView and finally to bind it. *The code is essentially the same as we used in the WinForms example.*

```
Protected Sub ddlCategory_SelectedIndexChanged(ByVal sender As Object, _
                                         ByVal e As EventArgs) Handles ddlCategory.SelectedIndexChanged

    If (TryCast(sender, DropDownList)).SelectedValue <> Nothing Then

        Dim selectValue As System.Nullable(Of Integer) = _
            Int32.Parse((TryCast(sender, DropDownList)).SelectedValue)

        Dim filteredData = _
            From product In ScopeExtent(Of Product)() _
            Where product.CategoryID.Equals(selectValue) _
            Select _
                ProductName = product.ProductName, _
                QuantityPerUnit = product.QuantityPerUnit, _
                CategoryName = product.Category.CategoryName
        gvProducts.DataSource = filteredData.ToList()
        gvProducts.DataBind()

    End If
End Sub
```

```
protected void ddlCategory_SelectedIndexChanged(object sender, EventArgs e)
{
    if ((sender as DropDownList).SelectedValue != null)
    {
        int? selectValue =
            (int?)Int32.Parse((sender as DropDownList).SelectedValue);

        var filteredData = from product in scope.Extent<Product>()
                           where product.CategoryID == selectValue
                           select new
                           {
                               Product = product.ProductName,
                               QuantityPerUnit = product.QuantityPerUnit,
                               Category = product.Category.CategoryName
                           };
        gvProducts.DataSource = filteredData.ToList();
        gvProducts.DataBind();
    }
}
```

- 13) Run the application. Click the categories in the drop down list and view the data in the GridView.

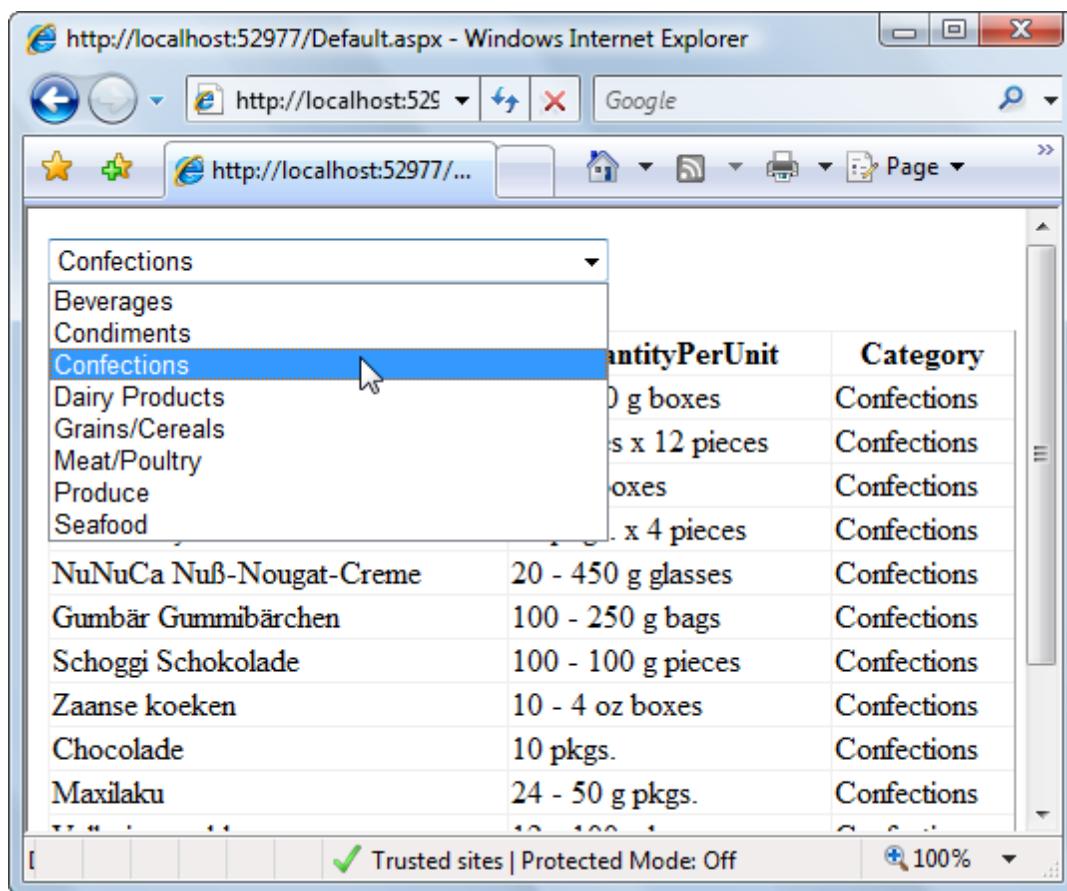


Figure 49

Using OpenAccess and RadControls for ASP.NET AJAX



Find the source projects for this chapter at \Projects\ORM\<CS\VB>\Made_Easy\ORM_Projects.sln, project "3 ORM MyWebAppAJAX"

- 1) Create a New Project called "3 ORM MyWebAppAJAX"
- 2) Follow the Steps in the Previous section (up to Step 5 - Except substitute 3 ORM MyWebAppAJAX in place of 3 ORM MyWebApp)
- 3) From the ToolBox drop a **OpenAccessDataSource**.
- 4) Build the Project (This allows ORM to see the new Data Context for this Project)
- 5) Add a RadComboBox. Open the Smart Tag and select the "Forest" skin. In the Properties Window set the **Name** property to "ddlCategory" and the **AutoPostBack** property to True.
- 6) Add a RadGrid to the form. Open the Smart Tag and set the Skin to "Forest". In the Properties Window, set the Name property to "gvProducts"
- 7) Click the OpenAccessDataSource Smart Tag and select **Configure Data Source...** This will display the OpenAccess Data Source Wizard. On the first page of the wizard, select the "_3 ORM MyWebAppAJAX.NorthwindProvider, 3 ORM MyWebAppAJAX" from the drop down list. Click the **Next** button. On the second page, Select "_2 ORM Model.Category" from the drop down list. Click **Finish** to close the Wizard.
- 8) Add Telerik.Web.UI, Telerik.OpenAccess, and _2 ORM Model to the using (C#) or Imports (VB) of the code behind file
- 9) Add the following to the code-behind file to manage the ObjectScope

```
Private _scope As IObjectScope = NorthwindProvider.GetNewObjectScope()

Public Property Scope() As IObjectScope
    Get
        Return _scope
    End Get
    Set(ByVal value As IObjectScope)
        _scope = value
    End Set
End Property

Public Overloads Overrides Sub Dispose()
    scope.Dispose()
    MyBase.Dispose()
End Sub
```

```
private IObjectScope scope = NorthwindProvider.GetNewObjectScope();  
  
public IObjectScope Scope  
{  
    get { return scope; }  
    set { scope = value; }  
}  
  
public override void Dispose()  
{  
    scope.Dispose();  
    base.Dispose();  
}
```

10) Add the following to the Page_Load Event

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)  
Handles Me.Load  
    If Not IsPostBack Then  
        ddlCategory.DataSource = OpenAccessDataSource1  
        ddlCategory.DataTextField = "CategoryName"  
        ddlCategory.DataValueField = "CategoryID"  
        ddlCategory.DataBind()  
    End If  
End Sub
```

```
protected void Page_Load(object sender, EventArgs e)  
{  
    if (!IsPostBack)  
    {  
        ddlCategory.DataSource = OpenAccessDataSource1;  
        ddlCategory.DataTextField = "CategoryName";  
        ddlCategory.DataValueField = "CategoryID";  
        ddlCategory.DataBind();  
    }  
}
```

11) In the Events tab of the Properties Window, navigate to the SelectedIndexChanged event and double-click it to create a new event handler. Add the code below to slightly change the previous event handler to match the current parameter and object naming conventions.

```

Protected Sub ddlCategory_SelectedIndexChanged(ByVal o As Object,
                                         ByVal e As Telerik.Web.UI.RadComboBoxSelectedIndexChangedEventArgs)
    Handles ddlCategory.SelectedIndexChanged

    If (TryCast(o, RadComboBox)).SelectedValue <> Nothing Then
        Dim selectValue As System.Nullable(Of Integer) = _
            Int32.Parse((TryCast(o, RadComboBox)).SelectedValue)

        Dim filteredData = From product In ScopeExtent(Of Product)() _
                           Where product.CategoryID = selectValue

        gvProducts.DataSource = filteredData.ToList()
        gvProducts.DataBind()
    End If
End Sub

```

```

protected void ddlCategory_SelectedIndexChanged1(object o,
                                             Telerik.Web.UI.RadComboBoxSelectedIndexChangedEventArgs e)
{
    if ((o as RadComboBox).SelectedValue != null)
    {
        int? selectValue =
            (int?)Int32.Parse((o as RadComboBox).SelectedValue);

        var filteredData = from product in scopeExtent<Product>()
                           where product.CategoryID == selectValue
                           select new
                           {
                               Product = product.ProductName,
                               QuantityPerUnit = product.QuantityPerUnit,
                               Category = product.Category.CategoryName
                           };
        gvProducts.DataSource = filteredData.ToList();
        gvProducts.DataBind();
    }
}

```

- 12) Add a RadAjaxManager to the webform
- 13) From the RadAjaxManager Smart Tag select the Add ScriptManager link.
- 14) Also from the Smart Tag select the Configure AJAX Manager link. Configure AJAX with the RadAjaxManager Property Builder.
- 15) In the left most pane of controls that will initiate AJAX requests, select the checkbox next to "ddlCategory".
- 16) In the middle panel of controls to update, select "gvProducts". The dialog should look like the screenshot below:

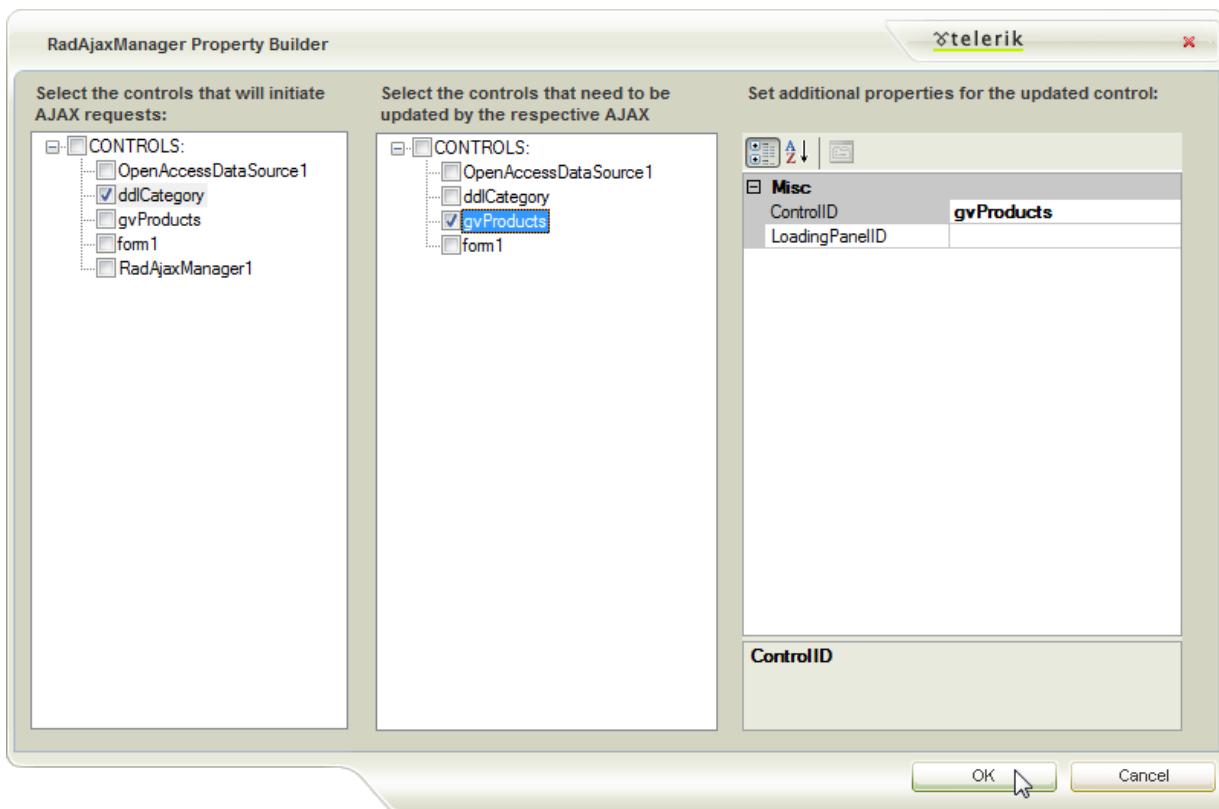


Figure 50

- 17) Click the OK button to close the dialog.
- 18) Run the application. The application should look similar to below.

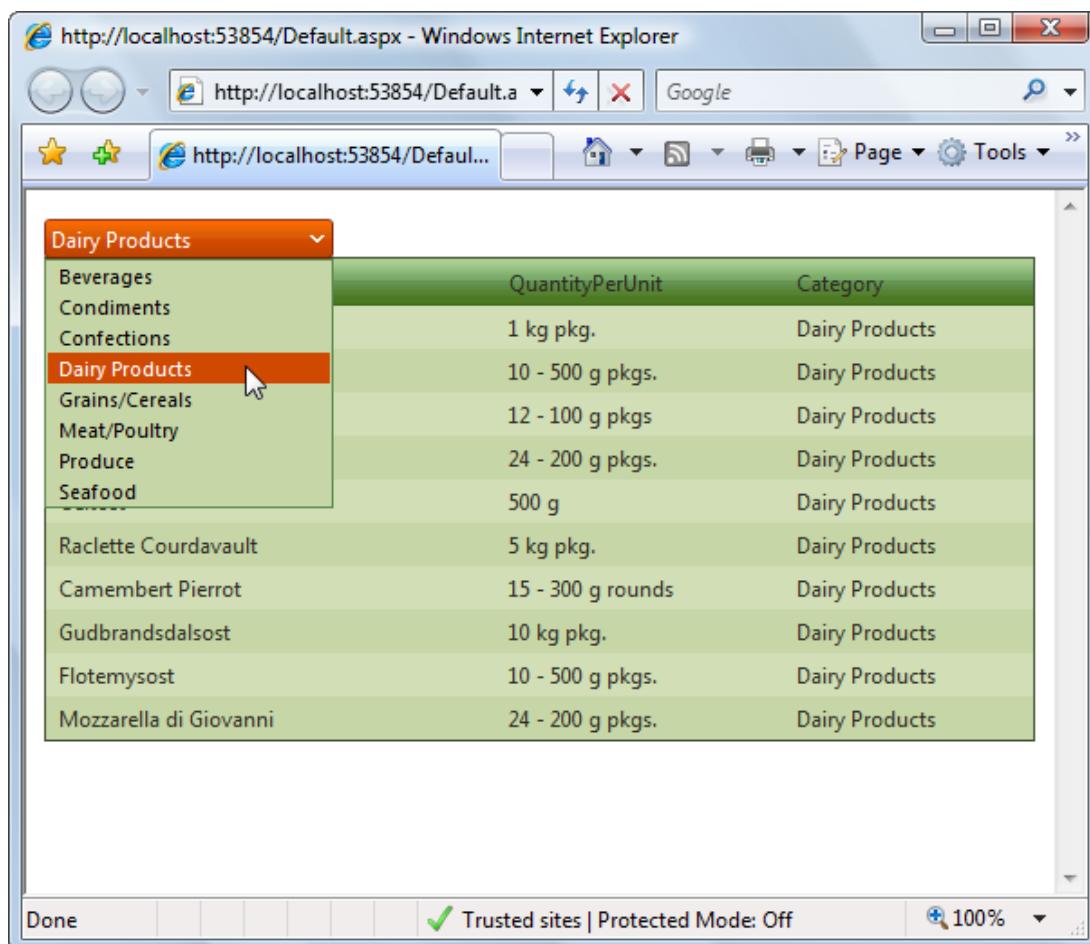


Figure 51

4.4 Telerik Reporting Example

This next example presents our Model data in Telerik Report format and is quite similar to the WinForms sample in some ways. We will use a ObjectProvider and ObjectView components to access the data, but will use a Telerik Report and SubReport for presentation. We will use the Preview window inside the Telerik Report design environment. See the Online Help or Telerik Reporting Step by Step Learning guide for more information on displaying reports in web and WinForms applications.



Find the source projects for this chapter at \Projects\ORM\<CS\VB>\Made_Easy\ORM_Projects.sln, project "5 ORM MyReports"

- 1) Create a new class library project "5 ORM MyReports".
- 2) Delete the automatically created default "class1" from the project.

- 3) ORM-enable the project. Specify the following:
 - a) The Persistent classes option should be disabled.
 - b) The Data Access Code option should be enabled.
 - c) The database connection ID should be "NorthwindOAConnection"
 - d) The database should be "NorthwindOA".
- 4) In the Solution Explorer, add a reference to the "2_ORM_Model" class library.
- 5) In the Solution Explorer, rename "ObjectScopeProvider1.cs or ObjectScopeProvider1.vb" to "NorthwindProvider.cs or NorthwindProvider.vb". When prompted, click the **Yes** button to rename all references.
- 6) Right Click on the 5_ORM_MyReports project and Update the Config References.
- 7) Build the application.
- 8) In the Solution Explorer, right-click the project and select **Add > New Item...** from the context menu. Select the **Reporting > Telerik Report** template, name it "CategoryReport" and click the **Add** button to close the dialog and create the report.

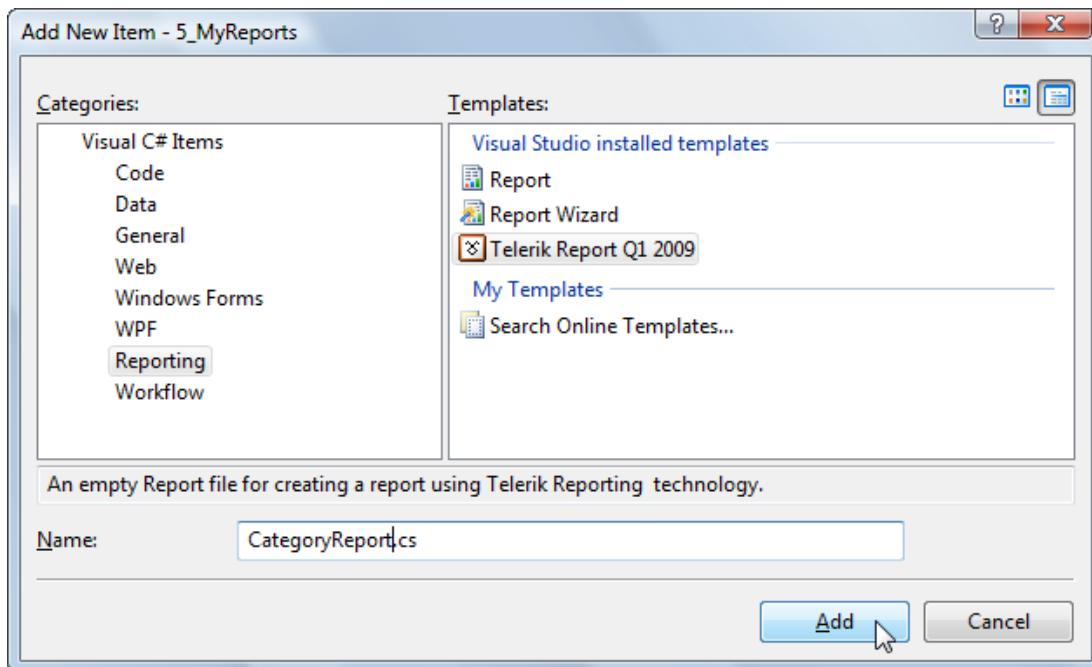


Figure 52

- 9) If the Telerik Report Wizard dialog displays, cancel it.
- 10) From the Toolbox, drop a **ObjectProvider** component to the form. This will display the ObjectProvider dialog. Configure the ObjectProvider as follows:
 - a) Context Provider = _5_ORM_MyReports.NorthwindProvider, 5_ORM_MyReports
 - b) Persistent Class = _2_ORM_Model.Category
 - c) Result Method = Query
 - d) Query Condition = leave as default "SELECT * FROM CategoryExtent AS x".
 - e) Click the **Finish** button to create the object provider "objectProvider1".

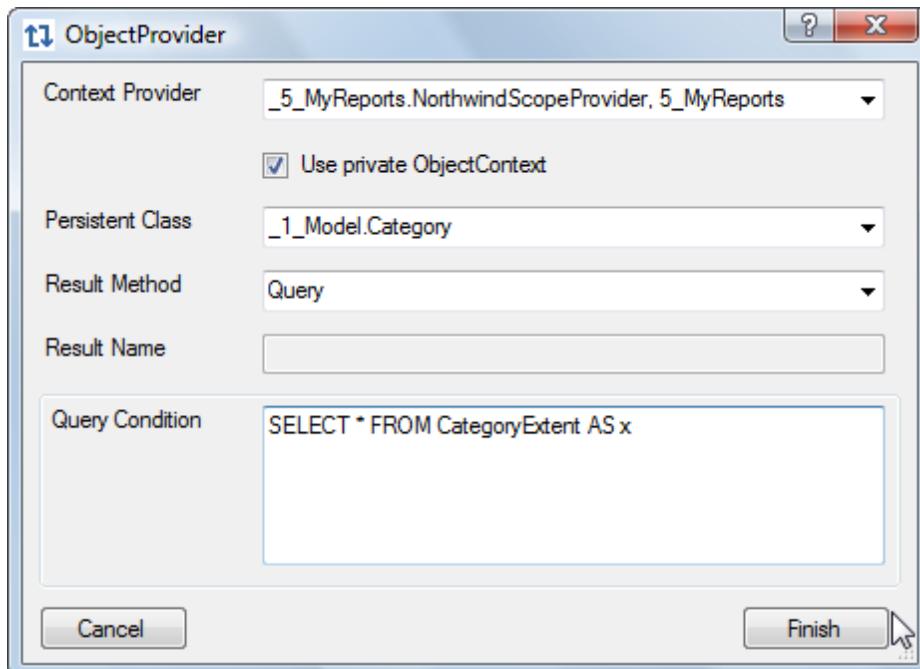


Figure 53

- 11) From the ToolBox, drag a **ObjectView** component to the form. This will display the ObjectView Wizard dialog. Configure the dialog as follows:
 - a) Point the ObjectProvider at "objectProvider1" using the drop down list.
 - b) Leave the Root Type setting pointed at "Model.Category".
 - c) Click the **Finish** button to create "objectView1".

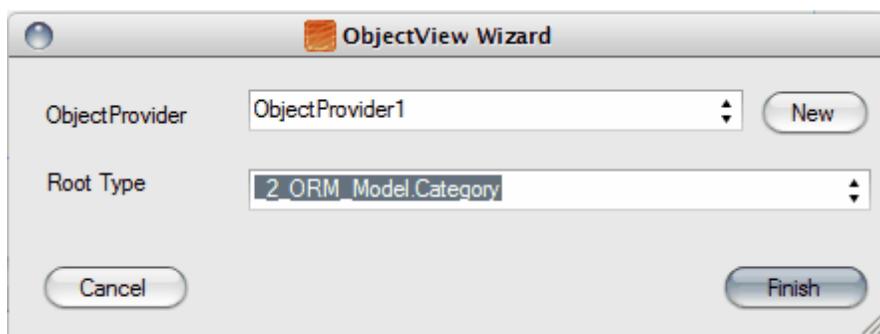


Figure 54

- 12) Rename "objectProvider1" to "opCategory" and "objectView1" to "ovCategory".
- 13) In the report design surface, select the report. You can do this by finding it in the Properties window drop down list, clicking the gray gutter area around the report bands, or clicking the button in the upper left hand side of the design surface.

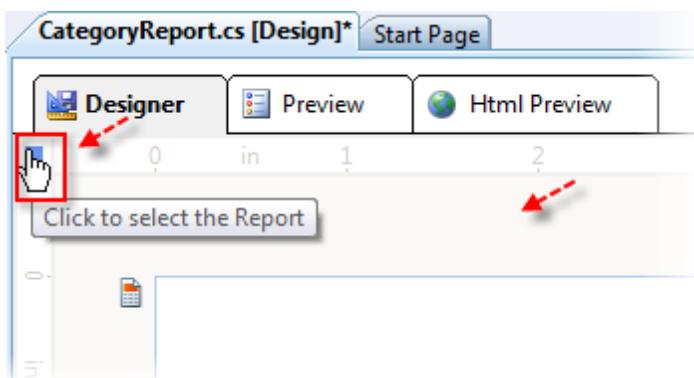


Figure 55

- 14) In the Properties Window, locate the **DataSource** property, drop down the list for the property and select "ovCategory".

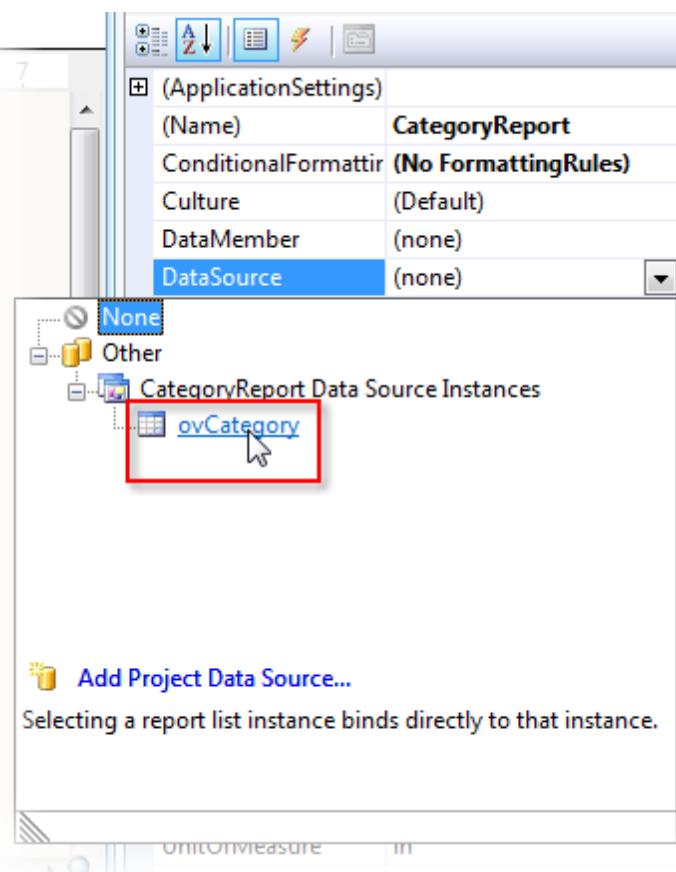


Figure 56

- 15) Save and build the project.
16) Select the "detail" section of the report (the middle band in the report designer).
17) From the Toolbox drop a TextBox reporting item in the Detail section of the report. Using the drag handles on the text box, drag the TextBox out to take up the width of the detail section. Double-click the TextBox and enter the following expression:

```
= "Products for " + Fields.CategoryName + " (" + Fields.Description + ")"
```

- 18) From the Toolbox drop a TextBox reporting item in the page header section of the report. Double-click the TextBox and enter "Products by Category". In the Properties window, set the **Style.Font.Bold** property to True.

The report designer should look something like the screenshot below:

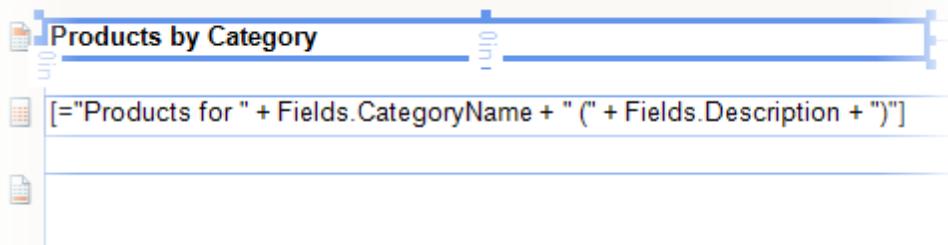


Figure 57

- 19) Create a new Telerik Report and name it "ProductReport.cs" or "ProductReport.vb" depending on which language you are using.
20) Select the PageHeaderSection, right-click and select **Delete** from the context menu to remove the section.
21) Select the PageFooterSection, right-click and select **Delete** from the context menu to remove the section. Only the detail section should remain.
22) Add ObjectProvider and ObjectView components to the form and configure them as you did earlier but point the provider at the "Products" table.

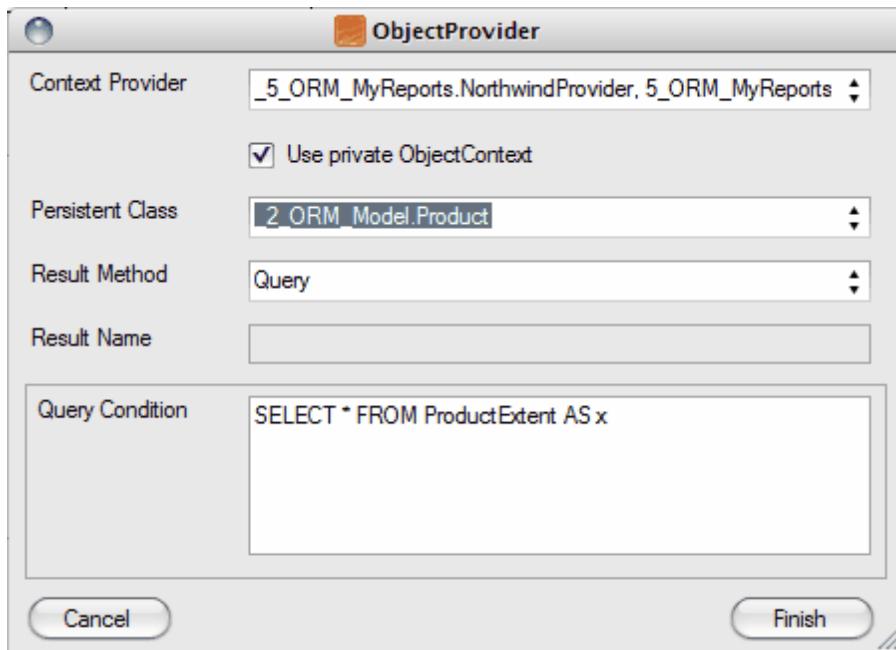


Figure 58

- 23) Set the **Name** property of the ObjectProvider "opProduct" and the ObjectView to "ovProduct".
- 24) Select the Report in the designer, navigate to the Properties window and set the following properties:
- Set the report **DataSource** property to "ovProduct".
 - Locate the **ReportParameters** property and click the ellipses. This will display the Report Parameter Collection editor. Click the **Add** button to create a new parameter. In the properties for the parameter set the **Name** property to "CategoryID", the **Type** property to "Integer" and the **Value** property to "0". Click the **OK** button to close the dialog.

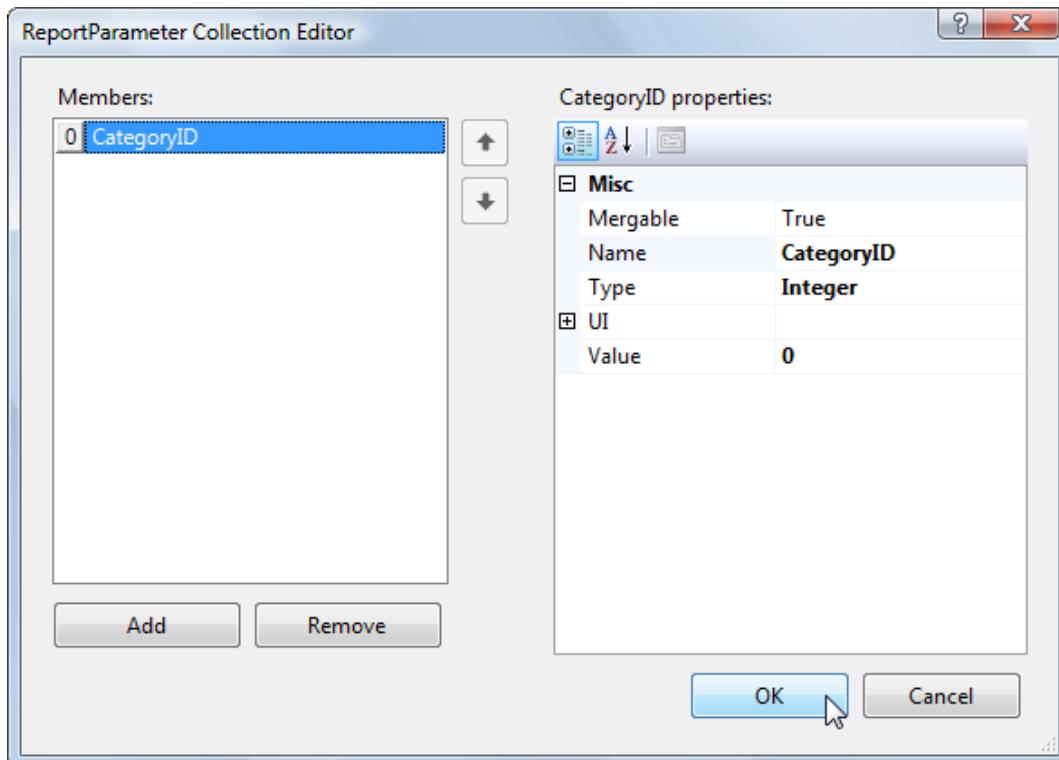


Figure 59

G If you receive an error later when attempting to preview the report that says there's a mismatch between an integer and a string type, double-check that you have set the Value property to a zero value.
c
h
a
!
o

- Locate the **Filters** property and click the ellipses. This will display the Edit Filters dialog. Click the **New** button to create a new filter. Drop down the list in the Expression column and select "=Fields. CategoryID" from the list. Leave the Operator column "=" selection. In the **Value** column, drop down the list and select "=Parameters.CategoryID". Click the **OK** button to close the dialog.

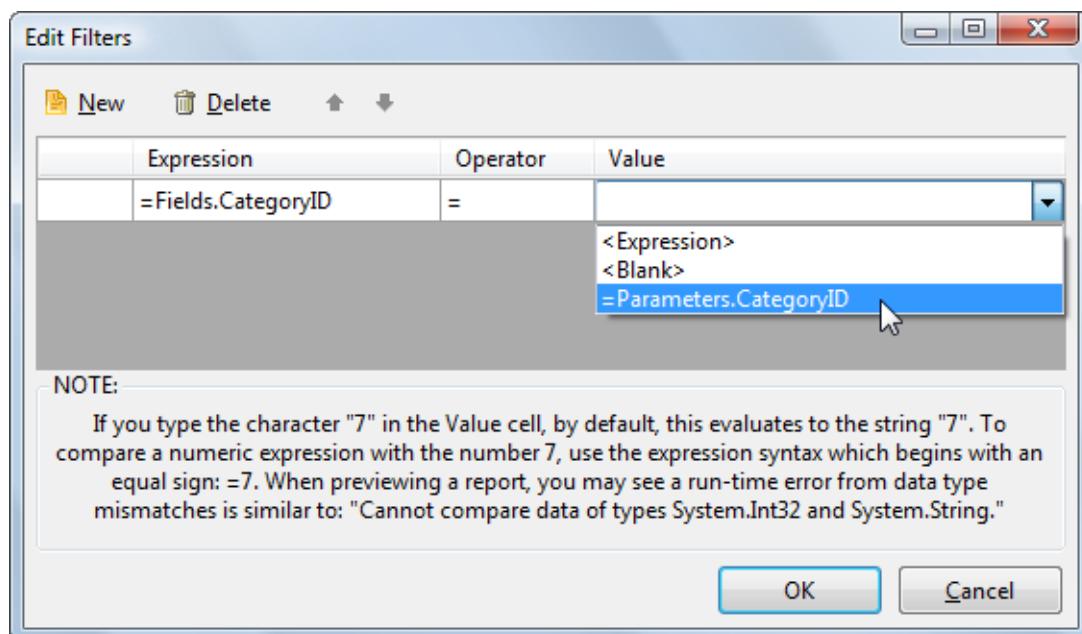


Figure 60

- 25) Select the detail section of the report.
- 26) Go to the Visual Studio menu and select **Telerik > Reporting > Data Explorer**.
- 27) From the Data Explorer drag the ProductName and QuantityPerUnit fields to the detail section of the report. Use the drag handles to resize both items to each take roughly half the width of the detail section.

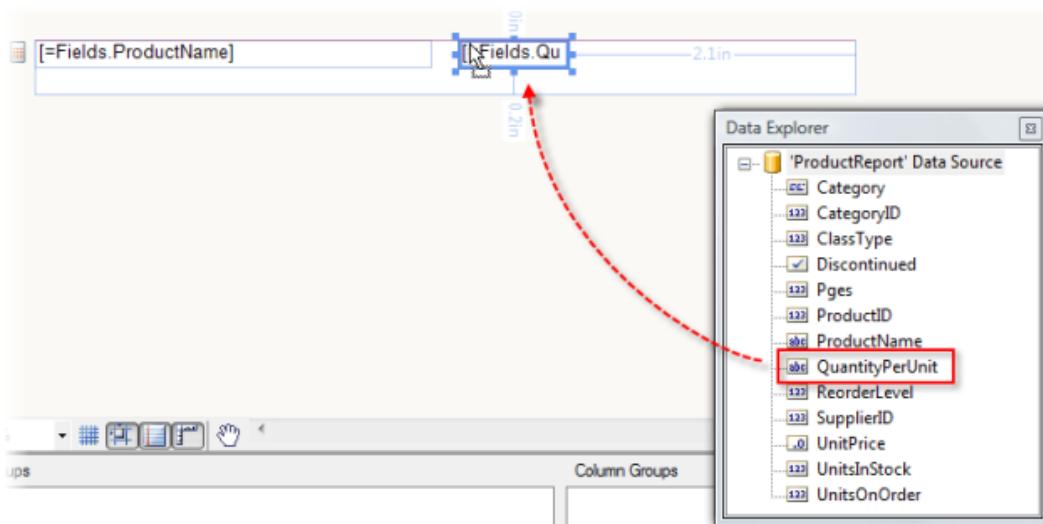


Figure 61

- 28) Navigate back to the "Category" report. From the Toolbox drop a SubReport item in the detail section just below the TextBox and indented slightly to the right. Set the SubReport properties:
 - a) Set the **ReportSource** property using the drop down list to "ProductReport".

** If you do not see ProductReport as an option, Build the Project and it should be available

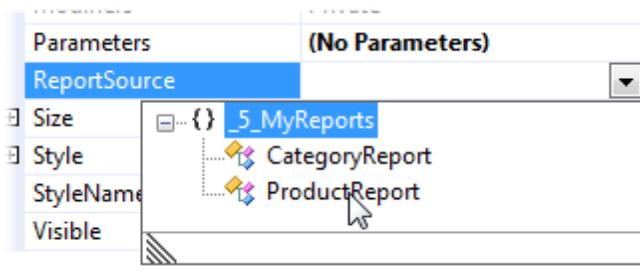


Figure 62

- b) Navigate to the **Parameters** property and click the ellipses. This will display the Edit Parameters dialog. Click the **New** button to create a new parameter. In the Parameter column enter "CategoryID" or select it from the drop down list. In the Parameter Value column, drop down the list and select "=Fields.CategoryID". Click the **OK** button to close the dialog.

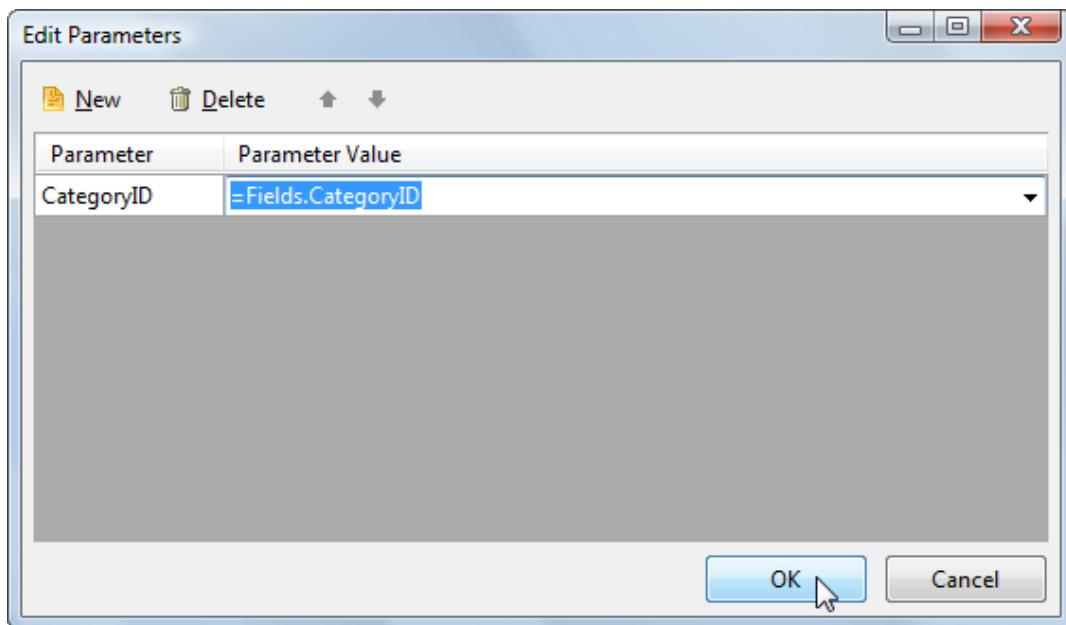


Figure 63

The "Category" report should now look something like this in the designer:

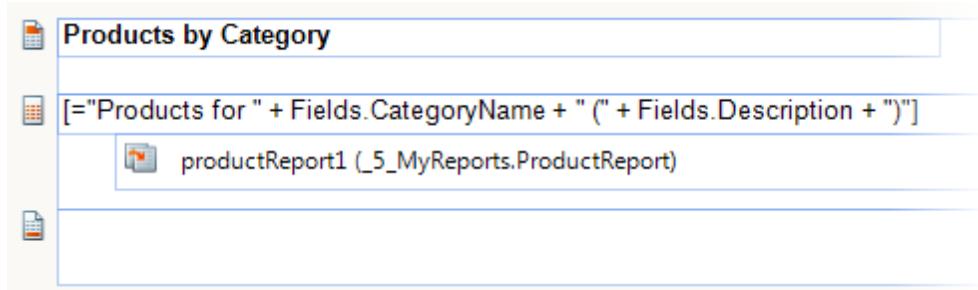


Figure 64

- 29) Click the **Preview** tab of the CategoryReport to see the finished report output.

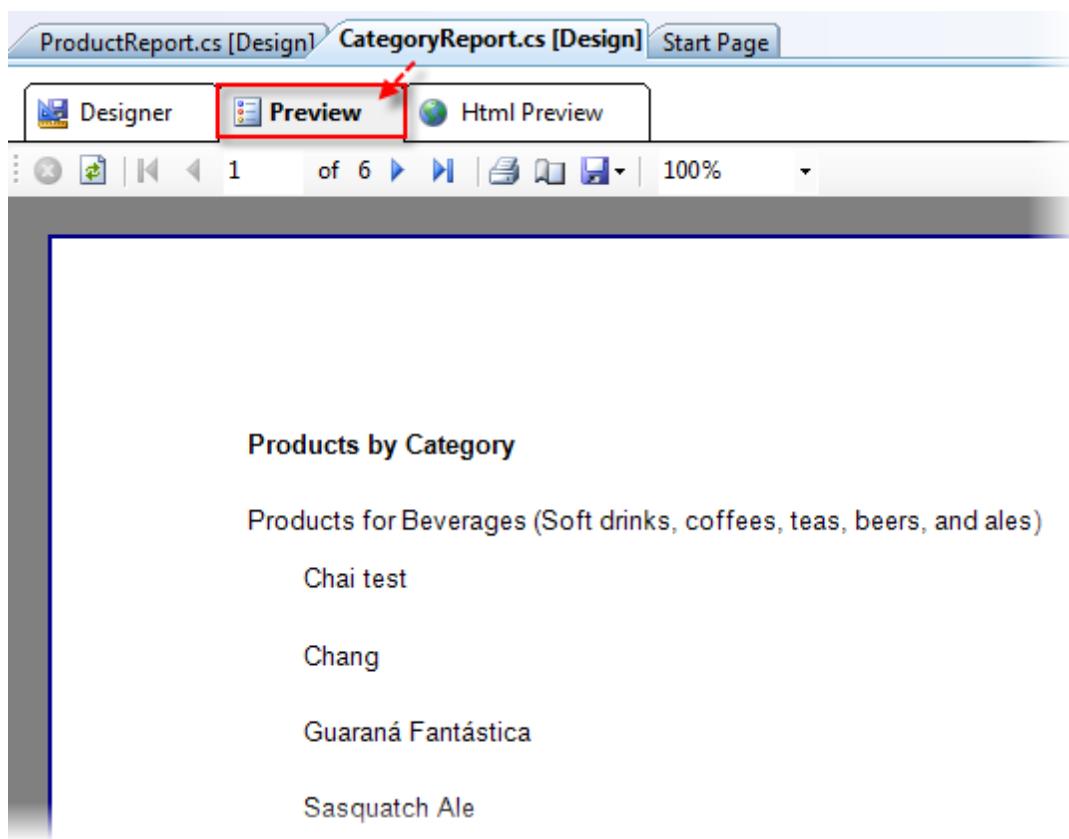


Figure 65

4.5 Multi-Tier Architecture

Our previous examples have been minimal by design and thoroughly unsuited for production. To begin building "industrial strength" applications we need a clean "separation of concerns", i.e. a more granular layering of functionality that respects principles of multi-tiered application design. Exactly how you separate these layers in your application will need to reflect the standards for your particular organization.

A "typical" production multi-tier application built with OpenAccess might have the structure below:

- **Entities (or Model):** Contains your persistent classes. The project for this layer is "OpenAccess Enabled" to define Persistent Classes (PCs) and includes database mapping info in the config file. The persistent classes in this project can be created via the Forward or Reverse Mapping OpenAccess wizards.
- **Data Layer (Optional):** This layer is optional since the actual "data plumbing" is handled "automatically" inside OpenAccess. You can skip this layer or use it for storing your LINQ queries and returning data to your business layer.
- **Business Layer:** Provides "Repository" (or "Manager") classes that define operations on the data model

(such as FindAll, FindById, Insert, etc.). This layer may acquire an ObjectScope used to manipulate objects in the database. This layer is also “OA Enabled” to consume persistent classes, which adds an ObjectScopeProvider to the project (but does not add database mapping info to project)

- **Service Layer (Optional):** To build an application that is ready for client-oriented technology (like Silverlight and ASP.NET AJAX 4.0), it is usually smart to build a service layer on top of your business logic. The service layer consumes your business layer “data manager” classes and exposes the operations as service endpoints.
- **Presentation Layer:** Consumes business layer repositories (if no service layer) or consumes services. The presentation layer can be completely ignorant of OpenAccess and it has no need to be “OA Enabled.”

The figure below demonstrates the relationship of architectural layers in diagram form:

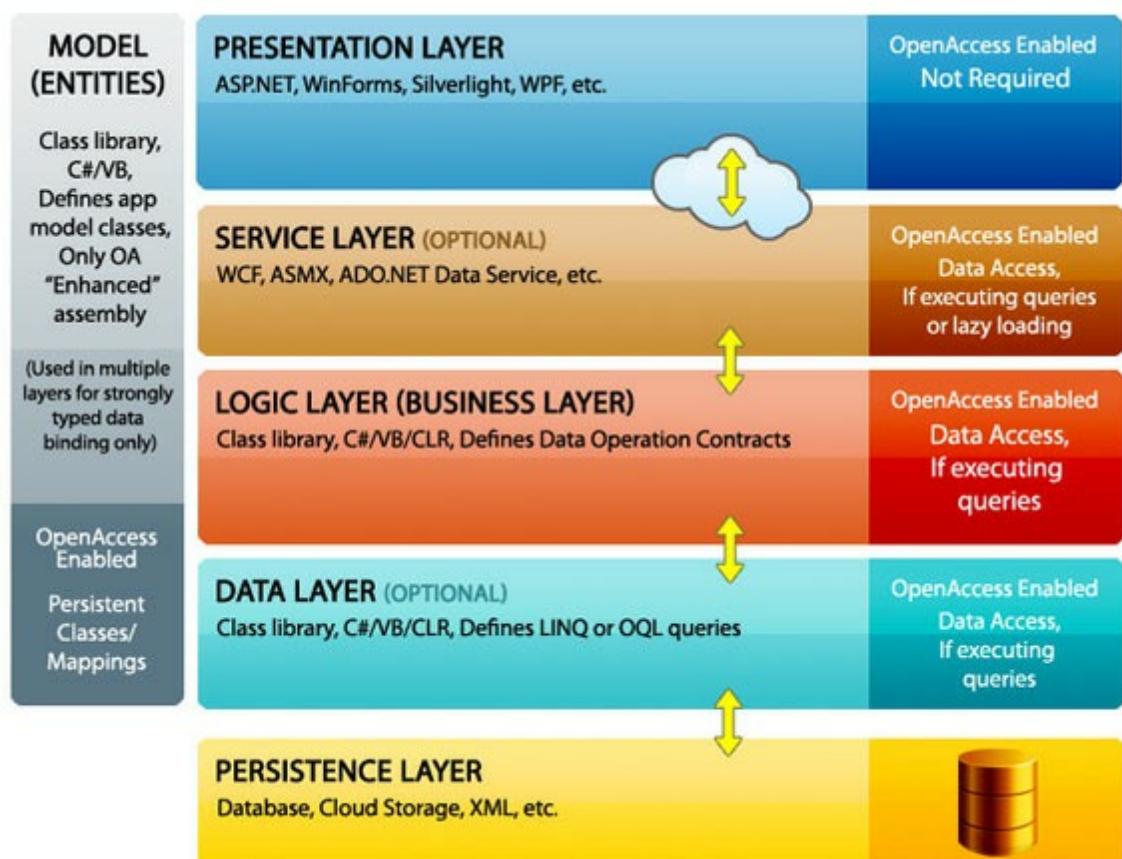


Figure 66

💡 The only time you might “break” this direct inheritance tree is if you want to re-use your model classes in your other tiers for easier data binding (i.e. I want to bind to return a List<ModelType> vs. some intermediate class type). Still, in that case, you’re only using the class model for binding- you’re not doing any data access outside of your data access tier.

What Projects Need to be OpenAccess Enabled?

One of the bigger points of confusion when it comes to using OpenAccess in n-tier applications seems to be which layers require you to run the “OpenAccess Enable” wizards. To be clear, there are two types of OpenAccess enabling provided by the wizard in Visual Studio:

1. **Project Defines Persistent Classes:** Adds database mapping information to the configuration file and adds references to OpenAccess assemblies.
2. **Project Consumes Persistent Classes/Connects to the Database:** Adds references to OpenAccess assemblies, adds an ObjectScopeProvider class (for help managing ObjectScope) and adds connection information to the project configuration file.

In general, the only layers that need OpenAccess enabling are your Model (where your persistent classes are defined) and your Data access or Business layer (or whichever layer you’re using to actually execute queries against the database). Other layers, like your Presentation layer, do not need to be enabled. And in all cases, when you run the wizard, you’re not adding references to other projects in your solution, just to the OpenAccess assemblies.

4.6 Web Services Example

This next example begins to go in the direction of N-Tier but only has three parts. A later sample project will show these three parts exploded out into layers that resemble the Multi-Tier architecture diagram from the previous section.

- The Data Model class library: This will contain two simple classes Contact and ContactGroup that will be forward mapped into a new database.
- A Windows Communication Foundation (WCF) service: This project will combine data access, business logic and service layers (although this application really has no business logic to speak of). The service will have methods to get contact groups, contacts and to save changes.
- The presentation layer in the form of a WinForms application.



Find the source at \Projects\ORM\<CS\VB>\Webservice\Webservice.sln

The Data Model

- 1) Create a new class library project "Model".
- 2) Delete the default Class1.cs or Module1.vb file
- 3) Add a new class to the project named "Contact.cs" or "Contact.vb". Add the code below:

```
■ Public Class Contact
    Private _contactName As String
    Private _phone As String
    Private _contactGroup As ContactGroup

    Public Property ContactName() As String
        Get
            Return _contactName
        End Get
        Set
            _contactName = value
        End Set
    End Property

    Public Property Phone() As String
        Get
            Return _phone
        End Get
        Set
            _phone = value
        End Set
    End Property

    Public Property ContactGroup() As ContactGroup
        Get
            Return _contactGroup
        End Get
        Set
            _contactGroup = value
        End Set
    End Property
End Class
```

```
④ public class Contact
{
    private string _contactName;
    private string _phone;
    private ContactGroup _contactGroup;

    public string ContactName
    {
        get { return _contactName; }
        set { _contactName = value; }
    }

    public string Phone
    {
        get { return _phone; }
        set { _phone = value; }
    }

    public ContactGroup ContactGroup
    {
        get { return _contactGroup; }
        set { _contactGroup = value; }
    }
}
```

- 3) Add a new class to the project named "ContactGroup.cs" or "ContactGroup.vb". Add the code below:

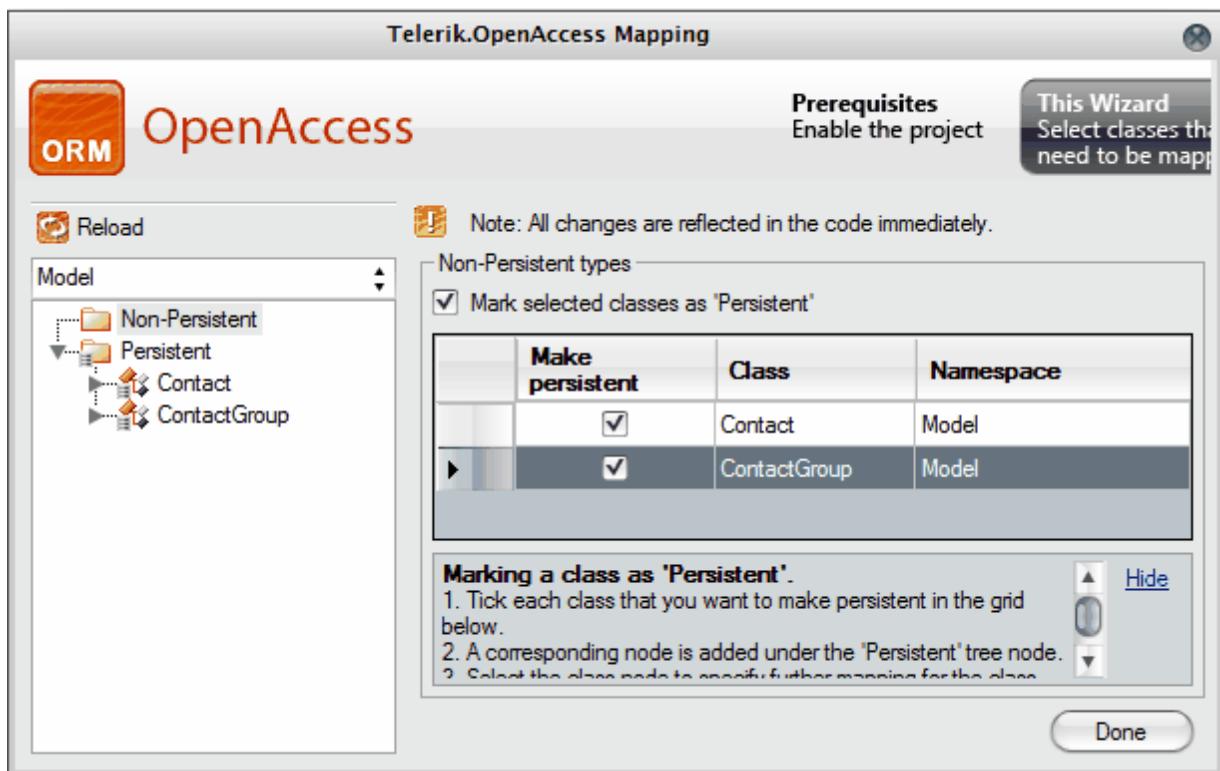
```
④ Public Class ContactGroup
    Private _contactGroupName As String

    Public Property ContactGroupName() As String
        Get
            Return _contactGroupName
        End Get
        Set
            _contactGroupName = value
        End Set
    End Property
End Class
```

```
④ public class ContactGroup
{
    private string _contactGroupName;

    public string ContactGroupName
    {
        get { return _contactGroupName; }
        set { _contactGroupName = value; }
    }
}
```

- 4) ORM-enable the project. Specify the following:
 - a) The Persistent classes option should be enabled.
 - b) The Data Access Code option should be disabled.
 - c) The database connection ID should be "MyDatabaseConnection".
 - d) The Server Name should be "(LOCAL)\SQLEXPRESS".
 - e) The database should be "MyDatabase".
- 5) Forward map the classes to the database. Using the Forward Mapping Wizard, mark both "Contact" and "ContactGroup" as Persistent.



- 6) To send these objects "over the wire" using WCF we need to mark the classes with **DataContract** and **DataMember** attributes.
 - a) Add a reference to **System.Runtime.Serialization** to your project
 - b) Open "Contact.cs" in the editor. Add the **System.Runtime.Serialization** namespace to the "Imports" (VB) or "using" (C#) section of the code. Mark the class with the **DataContract** attribute. Mark the "ContactName", "Phone" and "ContactGroup" properties with the **DataMember** attribute. The class should now look something like the code example below.

```
Imports System.Runtime.Serialization
<DataContract()> _
<Telerik.OpenAccess.Persistent()> _
Public Class Contact
    Private _contactName As String
    Private _phone As String
    Private _contactGroup As ContactGroup

    <DataMember()> _
    Public Property ContactName() As String
        Get
            Return _contactName
        End Get
        Set(ByVal value As String)
            _contactName = value
        End Set
    End Property

    <DataMember()> _
    Public Property Phone() As String
        Get
            Return _phone
        End Get
        Set(ByVal value As String)
            _phone = value
        End Set
    End Property

    <DataMember()> _
    Public Property ContactGroup() As ContactGroup
        Get
            Return _contactGroup
        End Get
        Set(ByVal value As ContactGroup)
            _contactGroup = value
        End Set
    End Property
End Class
```

```
using System.Runtime.Serialization;

namespace Model
{
    [DataContract]
    [Telerik.OpenAccess.Persistent()]
    public class Contact
    {
        private string _contactName;
        private string _phone;
        private ContactGroup _contactGroup;

        [DataMember]
        public string ContactName
        {
            get { return _contactName; }
            set { _contactName = value; }
        }

        [DataMember]
        public string Phone
        {
            get { return _phone; }
            set { _phone = value; }
        }

        [DataMember]
        public ContactGroup ContactGroup
        {
            get { return _contactGroup; }
            set { _contactGroup = value; }
        }
    }
}
```

- c) Open "ContactGroup.cs" in the editor. Add the **System.Runtime.Serializable** namespace to the "Imports" (VB) or "using" (C#) section of the code. Mark the class with the **DataContract** attribute. Mark the "ContactName", "Phone" and "ContactGroup" properties with the **DataMember** attribute. The class should now look something like the code example below.

```
Imports System.Runtime.Serialization

<DataContract()>
<Telerik.OpenAccess.Persistent()>
Public Class ContactGroup
    Private _contactGroupName As String

    <DataMember()>
    Public Property ContactGroupName() As String
        Get
            Return _contactGroupName
        End Get
        Set(ByVal value As String)
            _contactGroupName = value
        End Set
    End Property
End Class
```

```
using System.Runtime.Serialization;

namespace Model
{
    [DataContract]
    [Telerik.OpenAccess.Persistent()]
    public class ContactGroup
    {
        private string _contactGroupName;

        [DataMember]
        public string ContactGroupName
        {
            get { return _contactGroupName; }
            set { _contactGroupName = value; }
        }
    }
}
```

- 7) Click on the **Model** Project in the Solution Explorer and set the "Update Database" property to True.
- 8) From the Visual Studio menu select **Telerik > Open Access > Database Operations > Create Database**.
** You may have to delete the existing MyDatabase created in the "Getting Started Section" first.
- 9) In the Visual Studio Server Explorer, open the new database. Locate the Contact_Group table, right-click and select **Show Table Data** from the context menu. Add several records by hand to the table, e.g. "Vendor", "Partner", "Customer".

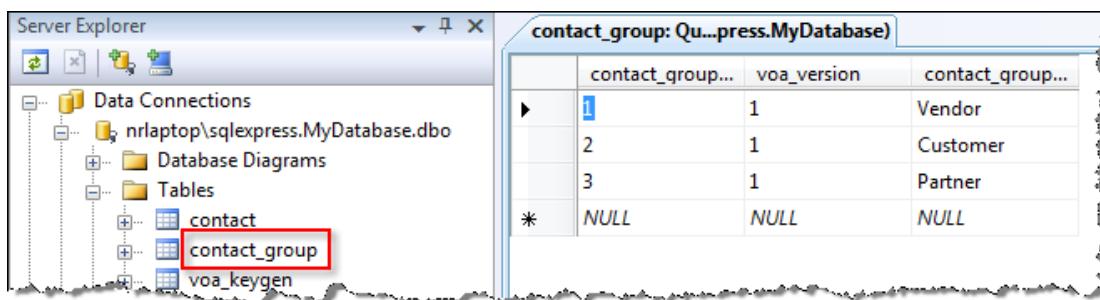


Figure 67

The WCF Service

- 1) In the Solution Explorer, right-click the solution and select **Add > New Project** from the context menu. Select the WCF Service Library template type, name it "Service" and click the **OK** button to close the dialog and create the service project.

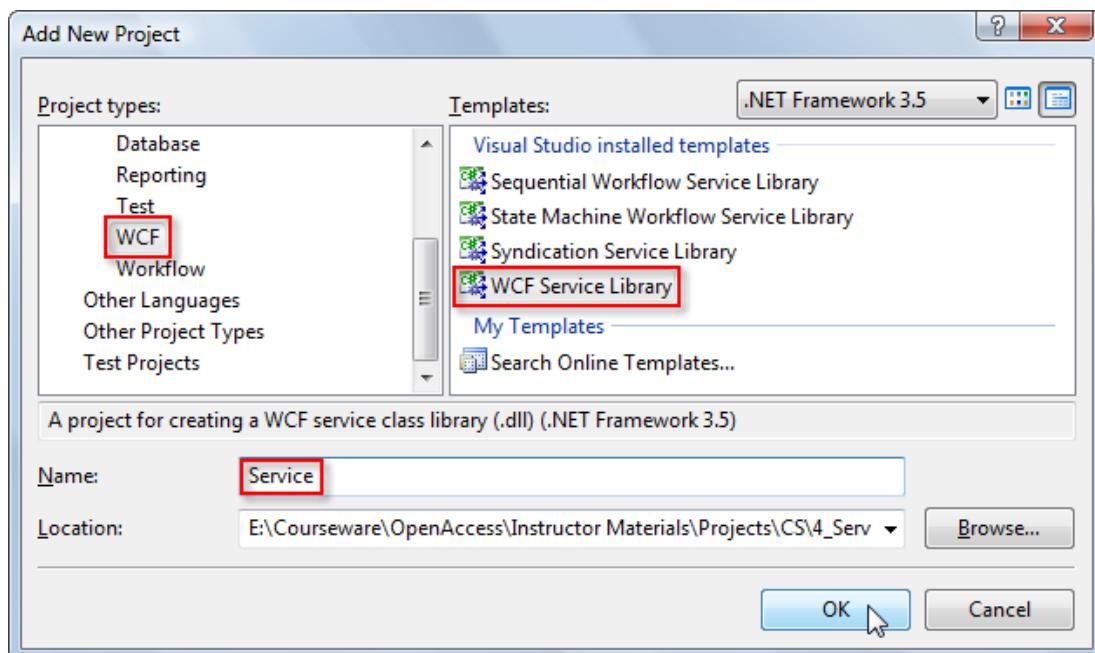
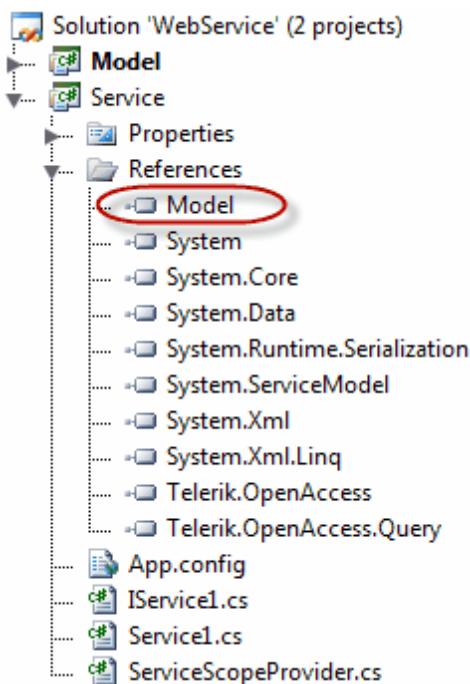


Figure 68

- 2) In the Solution Explorer, add a reference to the "Model" assembly.



- 3) ORM-enable the project. Specify the following:
 - a) The Persistent classes option should be disabled.
 - b) the Data Access Code option should be enabled.
 - c) The database connection ID should be "MyDatabaseConnection".
 - d) The Server Name should be "(LOCAL)SQLEXPRESS".
 - e) The database should be "MyDatabase".
- 4) Rename the default "ObjectScopeProvider1.cs" to "ServiceScopeProvider.cs" or "ObjectScopeProvider1.vb" to "ServiceScopeProvider.vb". Confirm the dialog to rename the other instances of the object in the project.
- 5) Edit the service interface:
 - a) Open "IService1.cs" in the editor.
 - b) Add **Telerik.OpenAccess** and **System.ServiceModel** namespaces to the "Imports" (VB) or "using" (C#) portion of the code.
 - c) Replace the body of the Service1 class with the three methods GetContactGroups(), GetContacts() and SaveContacts().

```

<OperationContract()> _
Function GetContactGroups() As ObjectContainer.ChangeSet

<OperationContract()> _
Function GetContacts() As ObjectContainer.ChangeSet

<OperationContract()> _
Function SaveContacts(ByVal changeset As ObjectContainer.ChangeSet) As
ObjectContainer.ChangeSet
  
```

```
[OperationContract]
ObjectContainer.ChangeSet GetContactGroups();

[OperationContract]
ObjectContainer.ChangeSet GetContacts();

[OperationContract]
ObjectContainer.ChangeSet SaveContacts(ObjectContainer.ChangeSet changeSet);
```

This code introduces the Telerik.OpenAccess **ObjectContainer** and **ChangeSet** classes. These important classes make disconnected database operations possible. The ObjectContainer is a client-side "stand-in" for the ObjectScope. The ObjectContainer has transaction and tracking capabilities like ObjectScope. But instead of methods to query the backend database, ObjectContainer methods copy objects into the container, create "ChangeSets" and apply ChangeSets. The ChangeSet object encapsulates a simple array of bytes and is used as a general purpose bucket to pass over the wire during WCF requests.

- 6) Open the "Service1.cs" class and implement the interface.
 - a) Add **Telerik.OpenAccess**, **Model** and **System.ServiceModel** namespaces to the "Imports" (VB) or "using" (C#) portion of the code.
 - b) Add a helper method "GetChangeSet" to the Service1 class that will copy data retrieved from the ObjectScope into the ChangeSet.

Notice in particular the CopyFrom() method that copies data from an object passed in to the container. The CopyFrom() method takes an IObjectScope, a "list name" (that can be used later to retrieve objects), an object to copy from and a IObjectCollector that defines what columns will be fetched from the database (by default, all columns).

Also notice the following call to Container.GetContent(). This returns a ChangeSet object populated with the current content of the container.

```
Private Shared Function GetChangeSet(listName As String, _
scope As IObjectScope, _
queryable As Object) As ObjectContainer.ChangeSet
    Dim container As New ObjectContainer()

    ' define the fetch groups used in the copy, by default all columns
    Dim collector As IObjectCollector =
        New FetchGroupCollector(FetchGroupCollector.DefaultFetchGroup)

    container.CopyFrom(scope, queryable.[GetType]().Name, _
        queryable, collector)

    ' get everything in the container as a changeset
    Dim changeSet As ObjectContainer.ChangeSet = container.GetContent()

    Return changeSet
End Function
```

```
private static ObjectContainer.ChangeSet GetChangeSet(string listName,
    IObjectScope scope, object queryable)
{
    ObjectContainer container = new ObjectContainer();

    // define the fetch groups used in the copy, by default all columns
    IObjectCollector collector =
        new FetchGroupCollector(FetchGroupCollector.DefaultFetchGroup);

    container.CopyFrom(scope, queryable.GetType().Name, queryable, collector);

    // get everything in the container as a changeset
    ObjectContainer.ChangeSet changeSet = container.GetContent();

    return changeSet;
}
```

- c) Add the implementations for all three methods.

Both the GetContactGroups() and GetContacts() methods follow the same pattern. The Extent method of the object scope returns the actual database records for the corresponding object type and places them into a "Var", i.e. an inferred but strongly-typed variable. This is passed to the helper GetChangeSet() method that returns the ChangeSet populated with Contact or ContactGroup data.

The SaveContacts() method takes the data in the other direction. ChangeSets sent from the client are added back into the container and committed to the database.

That's all we need to get Contact and ContactGroup information to and from the client.

```
Public Function IService1_GetContacts() As ObjectContainer.ChangeSet
Implements IService1.GetContacts
    Dim changeSet As ObjectContainer.ChangeSet = Nothing
    Using scope As IObjectScope = ServiceScopeProvider.GetNewObjectScope()
        Dim queryable = _
            From q In scope.Extent(Of Contact)() _
            Select q

        changeSet = GetChangeSet("Contacts", scope, queryable)

    End Using
    Return changeSet
End Function

Public Function IService1_GetContactGroups() As ObjectContainer.ChangeSet
Implements IService1.GetContactGroups
    Dim changeSet As ObjectContainer.ChangeSet = Nothing
    Using scope As IObjectScope = ServiceScopeProvider.GetNewObjectScope()
        Dim queryable = _
            From q In scope.Extent(Of ContactGroup)() _
            Select q

        changeSet = GetChangeSet("ContactGroups", scope, queryable)

    End Using
    Return changeSet
End Function

Public Function IService1_SaveContacts(ByVal changeset As ObjectContainer.
ChangeSet) As ObjectContainer.ChangeSet Implements IService1.SaveContacts
    Dim result As ObjectContainer.ChangeSet = Nothing

    Using scope As IObjectScope = _
        ServiceScopeProvider.GetNewObjectScope()
        ' commit the changeset from the client and return the updated set
        ' commits entire transaction?
        result = ObjectContainer.CommitChanges(changeset, _
        ObjectContainer.Verify.All, scope, True, True)
    End Using
    Return result
End Function
```

```
public ObjectContainer.ChangeSet GetContactGroups()
{
    ObjectContainer.ChangeSet changeSet = null;
    using (IObjectScope scope = ServiceScopeProvider.GetNewObjectScope())
    {
        var queryable =
            from q
            in scope.Extent<ContactGroup>()
            select q;

        changeSet = GetChangeSet("ContactGroups", scope, queryable);
    }
    return changeSet;
}

public ObjectContainer.ChangeSet GetContacts()
{
    ObjectContainer.ChangeSet changeSet = null;
    using (IObjectScope scope = ServiceScopeProvider.GetNewObjectScope())
    {
        var queryable =
            from q
            in scope.Extent<Contact>()
            select q;

        changeSet = GetChangeSet("Contacts", scope, queryable);
    }
    return changeSet;
}

public ObjectContainer.ChangeSet SaveContacts(
    ObjectContainer.ChangeSet changeSet)
{
    ObjectContainer.ChangeSet result = null;

    using (IObjectScope scope = ServiceScopeProvider.GetNewObjectScope())
    {
        // commit the changeset from the client and return the updated set
        // commits entire transaction?
        result = ObjectContainer.CommitChanges(changeSet,
            ObjectContainer.Verify.All, scope, true, true);
    }
    return result;
}
```

7) Test the WCF service.

- a) Right-click the WCF service project and set it to be the startup project using the context menu.
- b) Run the application. A WCF Test Client will automatically start, let you see your service methods and run them.
- c) Double-click the GetContactGroups() method in the left side tree view.
- d) Click the **Invoke** button.

- e) Observe the returned byte[] data.

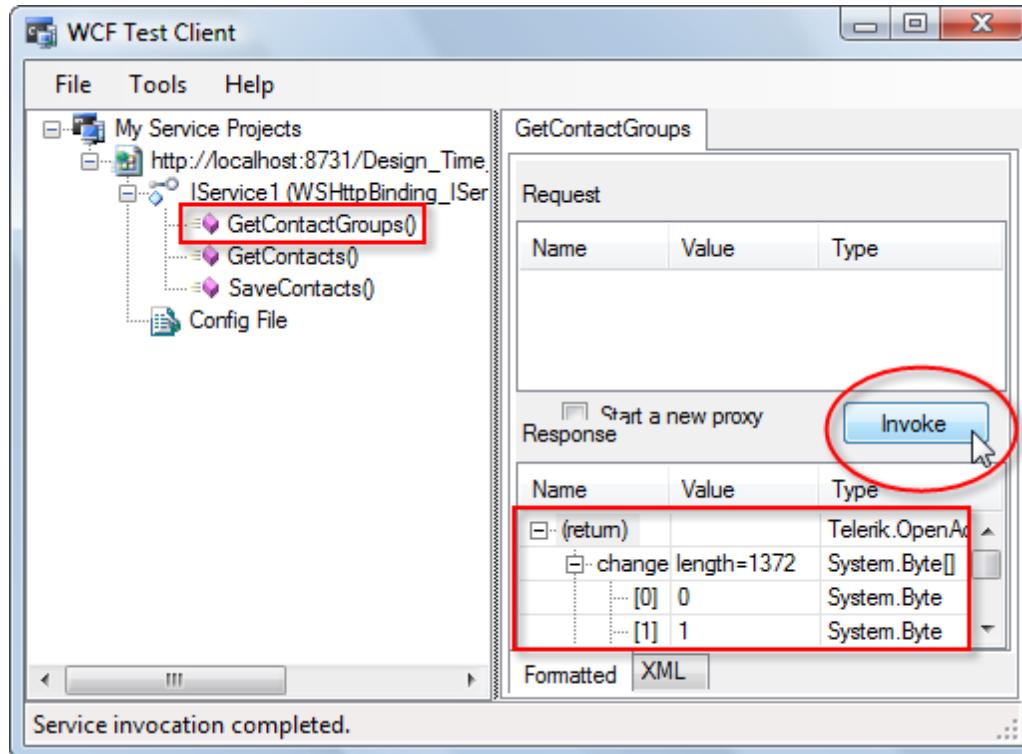


Figure 69

The WinForms Client

- 1) Add a new WinForms project to the solution and name it "WinFormClient".
- 2) From the ToolBox add a **RadComboBox**, **RadGridView** and a **RadButton** controls and set properties:
 - a) RadComboBox: **Name** = "cbContactGroups", **Text** = "".
 - b) RadGridView: **Name** = "gvContacts".
 - c) RadButton: **Name** = "btnSave", **Text** = "Save".

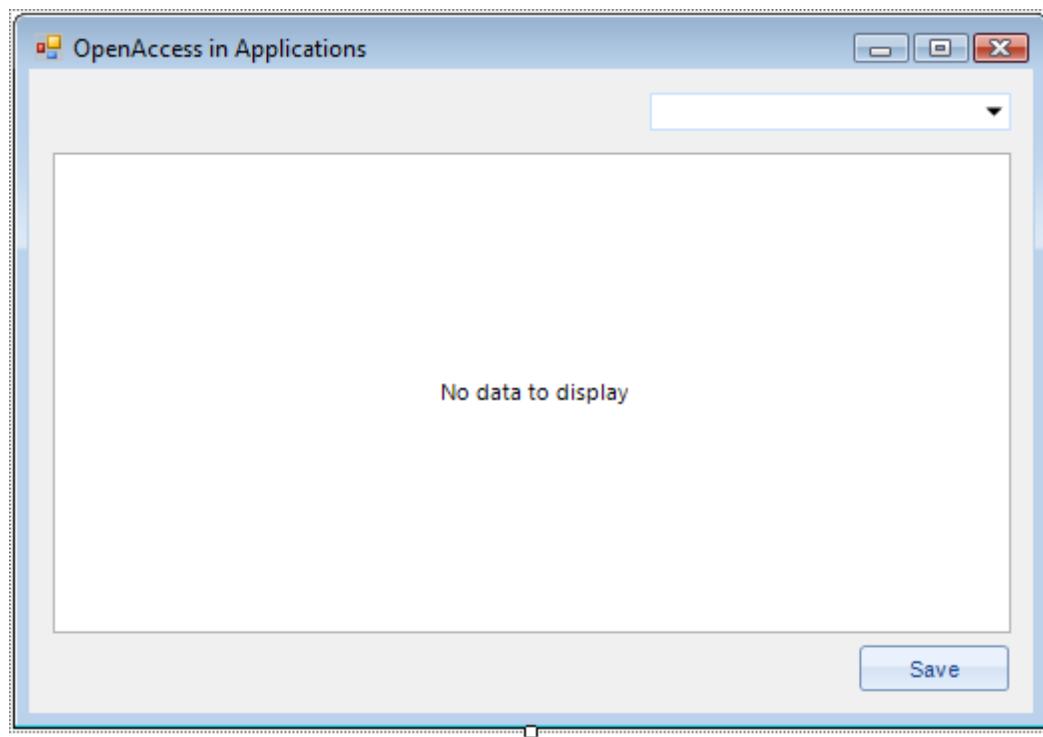
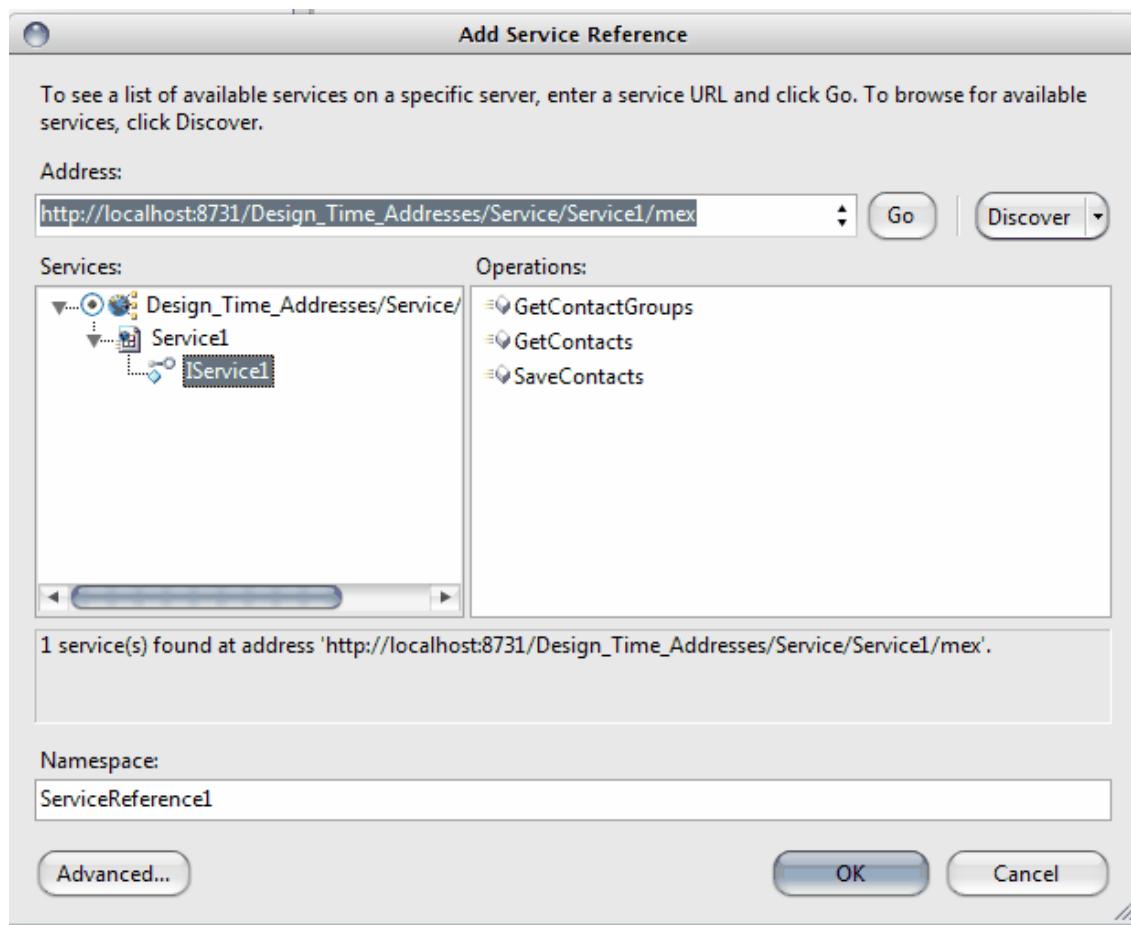
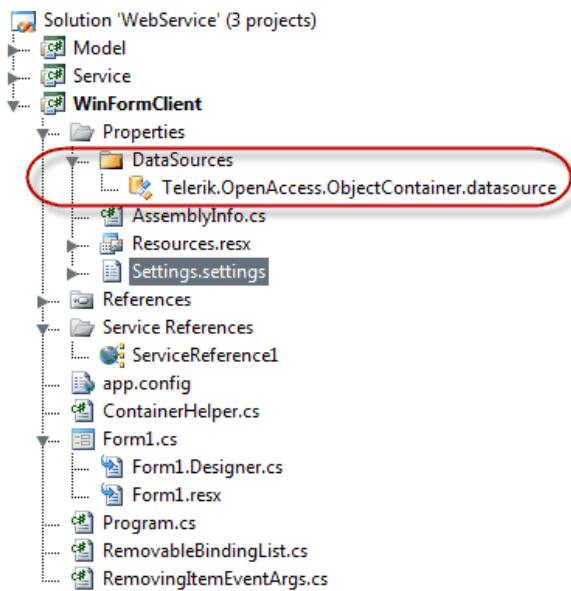


Figure 70

- 3) In the Solution Explorer, add a reference to the "**Model**" assembly.
- 4) In the Solution Explorer, add a reference to the **Telerik.OpenAccess** and **Telerik.OpenAccess.Query** assemblies.
- 5) Configure the project to use the WCF service.
 - a) In the Solution Explorer, right-click the References node a second time and select **Add Service Reference...** from the context menu.
 - b) Click the **Discover** button.
 - c) Select the IService1 node of the tree and click the **OK** button to automatically create a service client wrapper object.



- 6) If you expand the Solution explorer, you should see a new Project DataSource show up in your WinFormClient project (see image below).



- 7) Create a new class for the WinForms project and name it "ContainerHelper.cs" or "ContainerHelper".

"vb". This class will manage the activity that needs the Telerik.OpenAccess namespace, i.e. loading and saving ChangeSets on the client.

- a) Add references to **System.Collections.Generic** and **Telerik.OpenAccess** namespaces to the "Imports" (VB) or "using" (C#) section of code.
- b) Add a private ObjectContainer variable:

```
Private _container As New ObjectContainer()
```

```
private ObjectContainer _container = new ObjectContainer();
```

- c) Add a generic Load<T>() method that takes a ChangeSet and returns an IList of type T. *Any existing transaction has to be committed and inactive before applying the ChangSet into the container. The container Extent<T>() method returns a strongly typed IList.*

💡 Why is the "changeSet" parameter fed to a ObjectContainer.ChangeSet constructor in the example below?

If you've worked with web or WCF services, you may have had compatibility issues when using a type from another assembly that is also returned in the service, i.e. "Cannot convert 'Service.ChangeSet' to 'Telerik.OpenAccess.ObjectContainer.ChangeSet'". A proxy object is created automatically when you reference the service. The ChangeSet constructor takes care of the type mismatch for you by accepting an untyped object -- a ChangeSet passed in from the client.

```
Public Function Load(Of T) ( _  
    ByVal changeSet As ObjectContainer.ChangeSet) As IList(Of T)  
    ' ChangeSet constructor morphs  
    ' webservice changeset to application changeset type  
    Dim tempChangeSet As New ObjectContainer.ChangeSet(changeSet)  
  
    ' transaction must not be active before call to Apply()  
    If _container.Transaction.IsActive Then  
        _container.Transaction.Commit()  
    End If  
  
    ' apply service changeset to local container  
    ' transaction must not be active  
    _container.Apply(tempChangeSet)  
  
    ' start transaction for new add/edit/delete in the bound controls  
    _container.Transaction.Begin()  
  
    Return _container.Extent(Of T) ()  
End Function
```

```

    public IList<T> Load<T>(ObjectContainer.ChangeSet changeSet)
    {
        // ChangeSet constructor morphs
        // webservice changeset to application changeset type
        ObjectContainer.ChangeSet tempChangeSet =
            new ObjectContainer.ChangeSet(changeSet);

        // transaction must not be active before call to Apply()
        if (_container.Transaction.IsActive)
            _container.Transaction.Commit();

        // apply service changeset to local container
        // transaction must not be active
        _container.Apply(tempChangeSet);

        // start transaction for new add/edit/delete in the bound controls
        _container.Transaction.Begin();

        return _container.Extent<T>();
    }
}

```

- d) Add a delegate "SynchronizeDelegate" that will allow the caller to pass in a WCF service method to the ContainerHelper class. The delegate is used so the ContainerHelper class doesn't need to know about the service or have any direct information about the client code.

```

    Public Delegate Function SynchronizeDelegate( _
        ByVal changeSet As ObjectContainer.ChangeSet) _
        As ObjectContainer.ChangeSet

```

```

    public delegate ObjectContainer.ChangeSet
        SynchronizeDelegate(ObjectContainer.ChangeSet changeSet);

```

- e) Add a Save() method to send changes on the client back to the service for committing to the database.

The container's GetChanges() method returns a ChangeSet called "localChangeSet" with only changed data from the client. The ChangeSet is sent to the service by way of the Synchronize delegate where the client ChangeSet is applied to a Container in the service and a ChangeSet from the service is returned and applied back again to the local container. The return trip part of this voyage may not be necessary in this cut-down example where the data for the service isn't changing and isn't being re-queried from the database.

```
■ Public Sub Save(ByVal synchronize As SynchronizeDelegate)
    ' transactions cannot be active for GetChanges() or Apply()
    If _container.Transaction.IsActive Then
        _container.Transaction.Commit()
    End If

    ' get the changes from the client
    Dim localChangeSet As ObjectContainer.ChangeSet = _
    _container.GetChanges(ObjectContainer.Verify.Changed)

    ' send changes to the service, apply and return new
    ' changeset
    Dim serviceChangeSet As ObjectContainer.ChangeSet = _
    synchronize(localChangeSet)

    ' apply synchronized changeset
    If serviceChangeSet IsNot Nothing Then
        _container.Apply(serviceChangeSet)
    End If

    ' start transaction for new add/edit/delete in the bound controls
    _container.Transaction.Begin()
End Sub
```

```
■ public void Save(SynchronizeDelegate synchronize)
{
    // transactions cannot be active for GetChanges() or Apply()
    if (_container.Transaction.IsActive)
        _container.Transaction.Commit();

    // get the changes from the client
    ObjectContainer.ChangeSet localChangeSet =
        _container.GetChanges(ObjectContainer.Verify.Changed);

    // send changes to the service, apply and return new
    // changeset
    ObjectContainer.ChangeSet serviceChangeSet =
        synchronize(localChangeSet);

    // apply synchronized changeset
    if (serviceChangeSet != null)
        _container.Apply(serviceChangeSet);

    // start transaction for new add/edit/delete in the bound controls
    _container.Transaction.Begin();
}
```

- f) Include methods to add and remove objects from the container.

```

■ Public Sub Add(ByVal obj As Object)
    _container.Add(obj)
End Sub

Public Sub Remove(ByVal obj As Object)
    _container.Remove(obj)
End Sub

```

```

■ public void Add(object obj)
{
    _container.Add(obj);
}

public void Remove(object obj)
{
    _container.Remove(obj);
}

```

- 8) Now open the code-behind for the form Form1, add references for the following namespaces:
 - a) **System.Collections.Generic** (supports generic IList)
 - b) **System.ComponentModel** (supports "AddingNewEventArgs")
 - c) **System.Linq** (supports LINQ query for contacts)
 - d) **System.Reflection** (supports "Assembly" class)
 - e) **Model**
 - f) **Telerik.WinForms**
- 9) Add private variables for the service client and the ContainerHelper.

```

■ ' reference to our WCF service client
Private _client As New ServiceReference1.Service1Client()

' wrapper for OpenAccess Container and ChangeSets
Private _containerHelper As New ContainerHelper()

```

```

■ // reference to our WCF service client
private ServiceReference1.Service1Client _client =
    new ServiceReference1.Service1Client();

// wrapper for OpenAccess Container and ChangeSets
private ContainerHelper _containerHelper = new ContainerHelper();

```

- 10) Add a property to provide easy access to the currently selected ContactGroup in the combo box.

```
④ ' currently selected group name
Private ReadOnly Property CurrentGroupName() As String
    Get
        Dim index As Integer = cbContactGroups.SelectedIndex
        If index <> -1 Then
            Return cbContactGroups.Items(index).Text
        End If
        Return [String].Empty
    End Get
End Property
```

```
④ // currently selected group name
private string CurrentGroupName
{
    get
    {
        int index = cbContactGroups.SelectedIndex;
        if (index != -1)
        {
            return cbContactGroups.Items[index].Text;
        }
        return String.Empty;
    }
}
```

11) Add a helper method to load the combo box.

The service client method `GetContactGroups()` returns a `ChangeSet` from the service. The `ChangeSet` is fed into the `ContainerHelper Load<T>()` method which in turn returns an `IList` of type `T`. The `IList` is assigned to the `DataSource` of the combo box.

```
④ ' load contact groups to combo
Private Sub LoadCombo()
    ' get all contact groups as a ChangeSet from the service,
    ' load the ChangeSet to the container and return an
    ' IList of ContactGroup
    Dim contactGroups As IList(Of ContactGroup) = _
    _containerHelper.Load(Of ContactGroup)(_client.GetContactGroups())

    ' bind to the grid and display the ContactGroupName
    ' in the combo
    cbContactGroups.DataSource = contactGroups
    cbContactGroups.DisplayMember = "ContactGroupName"
End Sub
```

```
④ // load contact groups to combo
private void LoadCombo()
{
    // get all contact groups as a ChangeSet from the service,
    // load the ChangeSet to the container and return an
    // IList of ContactGroup
    IList<ContactGroup> contactGroups =
        _containerHelper.Load<ContactGroup>(_client.GetContactGroups());

    // bind to the grid and display the ContactGroupName
    // in the combo
    cbContactGroups.DataSource = contactGroups;
    cbContactGroups.DisplayMember = "ContactGroupName";
}
```

G BindingList is useful for many scenarios but doesn't have an event that tells you that an item has
o been deleted **and what the object to be removed is**. The ListChanged event, for example, lets
t you know when an item is removed, but fires *after* the object is gone. To get around this we will
c descend from BindingList, override the **RemoveItem()** method and raise our own "RemovingItem"
h event.
a
!
⑥

- 12) Add a new class "RemovingEventArgs.cs" or "RemovingEventArgs.vb" to the project. *This class encapsulates the object to be passed to our new RemovingItem event handler. In the convention of how Telerik codes events, the event will be cancelable.*

```

Imports System

Public Class RemovingEventArgs
    Inherits EventArgs
    Public Sub New()
    End Sub
    Public Sub New(ByVal cancel As Boolean, ByVal item As Object)
        Me.Cancel = cancel
        Me.Item = item
    End Sub

    Private privateCancel As Boolean
    Public Property Cancel() As Boolean
        Get
            Return privateCancel
        End Get
        Set(ByVal value As Boolean)
            privateCancel = value
        End Set
    End Property

    Private privateItem As Object
    Public Property Item() As Object
        Get
            Return privateItem
        End Get
        Set(ByVal value As Object)
            privateItem = value
        End Set
    End Property
End Class

```

```

using System;

namespace WinFormClient
{
    public class RemovingEventArgs : EventArgs
    {
        public RemovingEventArgs() { }
        public RemovingEventArgs(bool cancel, object item)
        {
            this.Cancel = cancel;
            this.Item = item;
        }

        public bool Cancel { get; set; }
        public object Item { get; set; }
    }
}

```

- 13) Add a new class "RemovableBindingList.cs" or "RemovableBindingList.vb".

This class will inherit from BindingList and override the RemoveItem method. If the RemoveItem

event is assigned, we create a `RemovingEventArgs`, set its `Cancel` property to `False` and the `Item` property to the object in the list pointed to by the "index" parameter. If the `Cancel` property has not been set to `True`, the base `RemoveItem` method is called.

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Public Class RemovableBindingList(Of T)
    Inherits BindingList(Of T)
    Public Event RemovingItem As EventHandler(Of RemovingEventArgs)

    Protected Overrides Sub RemoveItem(ByVal index As Integer)
        ' load up the item to be removed to args and fire event if assigned
        Dim args As RemovingEventArgs = Nothing
        If RemovingItemEvent IsNot Nothing Then
            args = New RemovingEventArgs(False, Me(index))
            Dim removingItem As EventHandler(Of RemovingEventArgs) = _
                RemovingItemEvent
            If RemovingItemEvent IsNot Nothing Then
                RemovingItemEvent(Me, args)
            End If
        End If

        If (Not args.Cancel) Then
            MyBase.RemoveItem(index)
        End If
    End Sub

    #Region "constructors"

    Public Sub New(ByVal list As IList(Of T))
        MyBase.New(list)
    End Sub

    Public Sub New()
        MyBase.New()
    End Sub

    #End Region

End Class
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;

namespace WinFormClient
{
    public class RemovableBindingList<T> : BindingList<T>
    {
        public event EventHandler<RemovingItemEventArgs> RemovingItem;

        protected override void RemoveItem(int index)
        {
            // load up the item to be removed to args and fire event if assigned
            RemovingItemEventArgs args = null;
            if (RemovingItem != null)
            {
                args = new RemovingItemEventArgs(false, this[index]);
                EventHandler<RemovingItemEventArgs> removingItem =
                    RemovingItem;
                if (removingItem != null)
                {
                    removingItem(this, args);
                }
            }

            if (!args.Cancel)
                base.RemoveItem(index);
        }

        #region constructors

        public RemovableBindingList(IList<T> list)
            : base(list) { }

        public RemovableBindingList() : base() { }

        #endregion
    }
}
```

14) Now back to the form Form1 - Add a helper method to load the grid.

The LoadGrid() method needs to filter the Contacts based on group. We use a LINQ query here that drills down to the contact's group and compares the group name against the current selection in the combo box. The call to ToList() causes the data to be loaded immediately and is stored in a generic List of Contact. This list is fed into the constructor for our RemovableBindingList.

Then we hook up the RemovableBindingList AddingNew and the new RemovingItem event so that we can add and remove items from the container.

```
[#] ' load all contacts for a group to grid
Private Sub LoadGrid()
    ' get all Contacts as a ChangeSet from the service,
    ' load the ChangeSet to the container and return
    ' an IList of Contacts
    Dim contacts As IList(Of Contact) = _
    _containerHelper.Load(Of Contact)(_client.GetContacts())

    ' filter the contacts for only those with the currently selected
    ' group name
    Dim filteredContacts As List( _
    Of Contact) = contacts.Where(Function(c) _
    c.ContactGroup.ContactGroupName.Equals(Me.CurrentGroupName)).ToList()

    Dim bindingList As RemovableBindingList(Of Contact) = _
    New RemovableBindingList(Of Contact)(filteredContacts)
    AddHandler bindingList.AddingNew, AddressOf bindingList_AddingNew
    AddHandler bindingList.RemovingItem, AddressOf bindingList_RemovingItem

    ' bind to the grid
    gvContacts.DataSource = bindingList

    ' hide the contact group column and fit the remaining columns
    If gvContacts.MasterGridViewTemplate.Columns.Count > 1 Then
        gvContacts.MasterGridViewTemplate.Columns(2).Visible = False
    End If
    gvContacts.MasterGridViewTemplate.BestFitColumns()
End Sub
```

```

// load all contacts for a group to grid
private void LoadGrid()
{
    // get all Contacts as a ChangeSet from the service,
    // load the ChangeSet to the container and return
    // an IList of Contacts
    IList<Contact> contacts =
        _containerHelper.Load<Contact>(_client.GetContacts());

    // filter the contacts for only those with the currently selected
    // group name
    List<Contact> filteredContacts =
        contacts.Where(c => c.ContactGroupName.Equals(
            this.CurrentGroupName)).ToList();

    RemovableBindingList<Contact> bindingList =
        new RemovableBindingList<Contact>(filteredContacts);
    bindingList.AddingNew +=
        new AddingNewEventHandler(bindingList_AuthingNew);
    bindingList.RemovingItem += new
        EventHandler<RemovingEventArgs>(bindingList_RemovingItem);

    // bind to the grid
    gvContacts.DataSource = bindingList;

    // hide the contact group column and fit the remaining columns
    if (gvContacts.MasterGridViewTemplate.Columns.Count > 1)
        gvContacts.MasterGridViewTemplate.Columns[2].IsVisible = false;
    gvContacts.MasterGridViewTemplate.BestFitColumns();
}

```

15) Handle the AddNew and RemoveItem events.

Both events keep the container up-to-date by adding or deleting the current list item. The AddingNew event handler has an additional task, to assign a new Contact object to the NewObject parameter property. Assigning NewObject simply lets us see the Contact in a new row of the grid before its added or canceled as shown in the figure below.

	ContactName	Phone
*	New Partner	
	Partner 3	(123)123-1234
	Partner 2	(123)123-1234

Figure 71

Also notice that the ContactGroupName uses a ContactGroup reference stored in the combo box item Tag property. We'll write code to populate the Tag property in the following step of this example.

```

■ Private Sub bindingList_RemovingItem(ByVal sender As Object, _
    ByVal args As RemovingItemEventArgs)
    _containerHelper.Remove(args.Item)
End Sub

' create a new contact and add to the grid and to the container.
' Note: e.NewObject shows up in the 'add new row' before being added
' to the grid data.
Private Sub bindingList_AddingNew( _
    ByVal sender As Object, _
    ByVal e As AddingNewEventArgs)
    Dim contactGroup As ContactGroup =
        TryCast((TryCast(cbContactGroups.SelectedItem, RadComboBoxItem)).Tag, _
        ContactGroup)

    e.NewObject = New Contact With {
        .ContactName = "New " & _
        contactGroup.ContactGroupName, _
        .ContactGroup = contactGroup, _
        .Phone = String.Empty}
    _containerHelper.Add(e.NewObject)
End Sub

```

```

■ void bindingList_RemovingItem(object sender, RemovingItemEventArgs args)
{
    _containerHelper.Remove(args.Item);
}

// create a new contact and add to the grid and to the container.
// Note: e.NewObject shows up in the "add new row" before being added
// to the grid data.
void bindingList_AddingNew(object sender, AddingNewEventArgs e)
{
    ContactGroup contactGroup =
        (cbContactGroups.SelectedItem as RadComboBoxItem).Tag as ContactGroup;

    e.NewObject = new Contact
    {
        ContactName = "New " + contactGroup.ContactGroupName,
        ContactGroup = contactGroup,
        Phone = String.Empty
    };
    _containerHelper.Add(e.NewObject);
}

```

16) Create event handlers for the combo box **SelectedIndexChanged** and **ItemDataBound** events.

The SelectedIndexChanged event simply triggers the LoadGrid() call.

Use the ItemDataBound to populate the Tag of each item in the combo box with the corresponding ContactGroup.

```
■ Private Sub cbContactGroups_SelectedIndexChanged_1(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles cbContactGroups.SelectedIndexChanged  
    LoadGrid()  
End Sub  
  
' save off currently selected ContactGroup  
Private Sub cbContactGroups_ItemDataBound_1(ByVal sender As System.Object, _  
    ByVal e As Telerik.WinForms.UI.ItemDataBoundEventArgs) Handles cbContactGroups.ItemDataBound  
    e.DataBoundItem.Tag = TryCast(e.DataItem, ContactGroup)  
End Sub
```

```
■ private void cbContactGroups_SelectedIndexChanged(  
    object sender, EventArgs e)  
{  
    LoadGrid();  
}  
  
// save off currently selected ContactGroup  
private void cbContactGroups_ItemDataBound(object sender,  
    ItemDataBoundEventArgs e)  
{  
    e.DataBoundItem.Tag = e.DataItem as ContactGroup;  
}
```

- 17) Handle the Form Load event. *This handler simply calls the LoadCombo() and then the LoadGrid() method.*

```
■ Private Sub Form1_Load(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles MyBase.Load  
    LoadCombo()  
    LoadGrid()  
End Sub
```

```
■ private void Form1_Load(object sender, EventArgs e)  
{  
    LoadCombo();  
    LoadGrid();  
}
```

- 18) Create an event handler for the "Save" button.

```

' send current changes to service
Private Sub btnSave_Click(ByVal sender As System.Object,
                           ByVal e As System.EventArgs) Handles btnSave.Click
    ' pass SaveContacts as a delegate
    _containerHelper.Save(AddressOf _client.SaveContacts)
End Sub

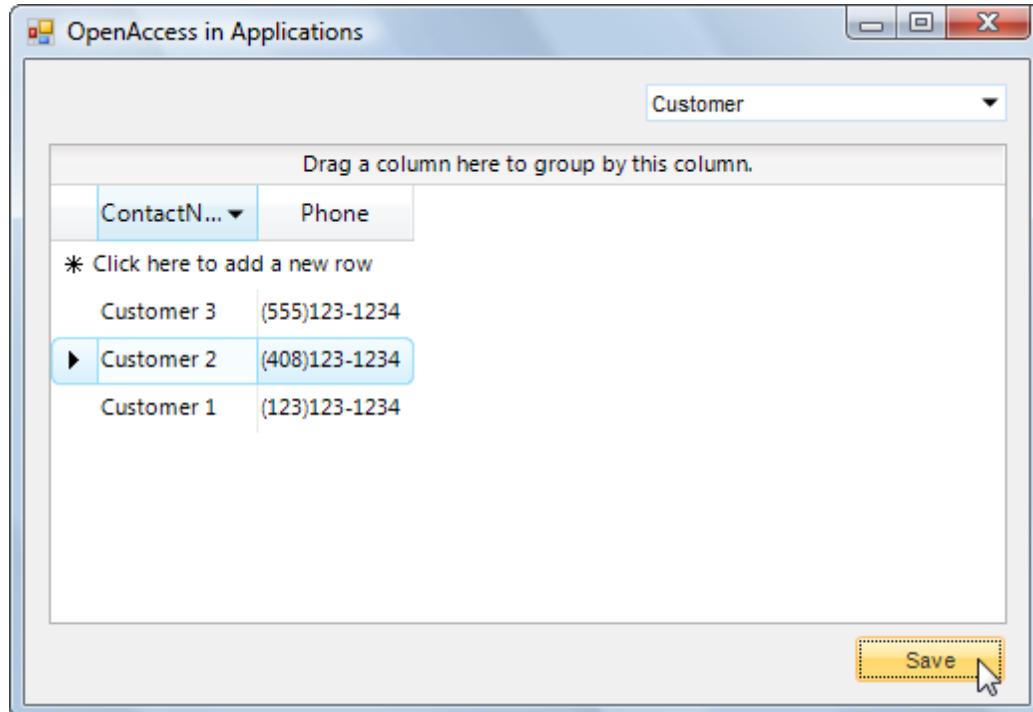
```

```

// send current changes to service
private void btnSave_Click(object sender, EventArgs e)
{
    // pass SaveContacts as a delegate
    _containerHelper.Save(_client.SaveContacts);
}

```

- 20) Set the WinForms project as the startup application and run the project.
- Try selecting different groups from the combo box.
 - Add new contacts by clicking the top row prompt in the grid.
 - Double-click grid cells to edit existing Contacts. Delete Contacts in the grid by selecting the row; click to the left of the row and hit the Delete key.
 - Click the Save button to save changes to the database.
 - Also make changes and switch between groups using the combo box to see that changes are not persisted in this case.



4.7 N-Tier Example

This next solution will break out the Service layer into Data Access, Business Logic and service layers. You won't need to change the data model or presentation layer (WinFormClient) at all. The service interface signature will remain unchanged, so we can make our modifications behind the scenes. The general game plan here is to:

- Create a ORM-enabled data access layer. This assembly will have an ObjectScopeProvider and knowledge of a database connection.
- Create a business logic layer. This assembly will reference the data access layer and will not be ORM-enabled and will not have its own ObjectScopeProvider. This business layer will not have any direct data access knowledge. Because we are passing around ChangeSet objects, it will have a reference to Telerik.OpenAccess. In a later example we'll add a single token business rule that validates contact phone numbers, but for now the business logic methods will simply consume the data access methods on a one-to-one basis.
- Modify the service layer to remove data access and instead consume the business layer methods. The service layer will reference the business layer, will not be ORM-enabled and will have no logic of its own other than to call business layer methods. Again, this assembly will have a reference to Telerik.OpenAccess only because we're using ChangeSet objects.



Find the source at \Projects\ORM\<CS\VB>\NTierExample\NTierExample.sln

Adding the Data Access Layer

- 1) Create a new Class Library project named "DataAccess".
- 2) In the Solution Explorer, add a reference to the "Model" assembly.
- 3) ORM-enable the project. Specify the following:
 - a) The Persistent classes option should be disabled.
 - b) the Data Access Code option should be enabled.
 - c) The database connection ID should be "MyConnectionString".
 - d) The Server Name should be "(LOCAL)\SQLEXPRESS".
 - e) The database should be "MyDatabase".
- 4) Rename "ObjectScopeProvider1.cs" to "DataAccessScopeProvider.cs" or if using VB change the names "ObjectScopeProvider1.vb" to "DataAccessScopeProvider.vb". Confirm the dialog to rename the other instances of the object in the project.
- 5) Delete the Class1.cs or Class1.vb file from the project.
- 6) Add a new class to the project named "ContainerManager.cs" or "ContainerManager.vb" and add the code below to the class.

The code should look familiar because it's derived from code used in the service of the previous example. The logic is more generic where there is a static method to LoadChangeSet<T> instead of separate methods to GetContacts() or GetContactGroups(). LoadChangeSet is a generic method that gets an IQueryable from the object scope for objects of type "T". Those objects are copied into the container, the GetContent() method gets a ChangeSet of everything in the container and the ChangeSet is returned to the caller. Note that the IObjectScope provides access to the database.

SaveChangeSet() performs the same purpose as SaveContacts in the service, but again is generic.

SaveChangeSet() takes a ChangeSet as a parameter, copies the contents into a container, then commits the container contents into the database. Again note that the IObjectScope provides access to the database.

This is the last point in the stack of architectural layers where we need a database connection.

```
Imports System.Linq
Imports Telerik.OpenAccess
Imports Telerik.OpenAccess.Query

Public Class ContainerManager
    ' return a ChangeSet loaded with objects populated from the database
    Public Shared Function LoadChangeSet(Of T) () _
        As ObjectContainer.ChangeSet
        Dim changeSet As ObjectContainer.ChangeSet = Nothing

        ' get a new object scope for data access in this method
        Using scope As IObjectScope =
            DataAccessScopeProvider.GetNewObjectScope()
                ' extract data for <T> and place in an IQueryable
                Dim queryable As IQueryable(Of T) =
                    From q In scope.Extent(Of T) () _
                    Select q

                ' create a container to hold the data
                Dim container As New ObjectContainer()

                ' define the FetchGroup as default for the copy (fetches all fields)
                Dim collector As IObjectCollector =
                    New FetchGroupCollector(FetchGroupCollector.DefaultFetchGroup)

                ' copy everything from the IQueryable, and name the list as the
                ' <T> type name
                container.CopyFrom(scope, GetType(T).Name, queryable, collector)

                ' save the current container content in a ChangeSet
                changeSet = container.GetContent()
            End Using

        Return changeSet
    End Function

    Public Shared Function SaveChangeSet( _
        ByVal changeSet As ObjectContainer.ChangeSet) As ObjectContainer.ChangeSet
        Dim result As ObjectContainer.ChangeSet = Nothing

        Using scope As IObjectScope =
            DataAccessScopeProvider.GetNewObjectScope()
                ' commit the changeset from the client and return the updated set
                result = ObjectContainer.CommitChanges(
                    changeSet, ObjectContainer.Verify.All, scope, True, True)
            End Using
        Return result
    End Function
```

End Class

```
using System.Linq;
using Telerik.OpenAccess;
using Telerik.OpenAccess.Query;

namespace DataAccess
{
    public class ContainerManager
    {
        // return a ChangeSet loaded with objects populated from the database
        public static ObjectContainer.ChangeSet LoadChangeSet<T>()
        {
            ObjectContainer.ChangeSet changeSet = null;

            // get a new object scope for data access in this method
            using (IObjectScope scope =
                DataAccessScopeProvider.GetNewObjectScope())
            {
                // extract data for <T> and place in an IQueryable
                IQueryable<T> queryable =
                    from q
                    in scope.Extent<T>()
                    select q;

                // create a container to hold the data
                ObjectContainer container = new ObjectContainer();

                // define the FetchGroup as default for the copy
                // (fetches all fields)
                IObjectCollector collector =
                    new FetchGroupCollector(FetchGroupCollector.DefaultFetchGroup);

                // copy everything from the IQueryable, and name the list as the
                // <T> type name
                container.CopyFrom(scope, typeof(T).Name, queryable, collector);

                // save the current container content in a ChangeSet
                changeSet = container.GetContent();
            }

            return changeSet;
        }

        public static ObjectContainer.ChangeSet SaveChangeSet(
            ObjectContainer.ChangeSet changeSet)
        {
            ObjectContainer.ChangeSet result = null;

            using (IObjectScope scope =
                DataAccessScopeProvider.GetNewObjectScope())
            {
                // commit the changeset from the client and return the
                // updated set
            }
        }
    }
}
```

```
        result = ObjectContainer.CommitChanges(changeSet,
            ObjectContainer.Verify.All, scope, true, true);
    }
    return result;
}

}
```

- 7) The project in the Solution Explorer should look something like the figure below:

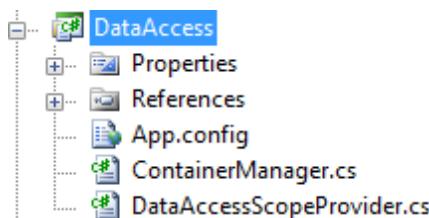


Figure 72

Adding the Business Layer

- 1) Create a new Class Library project named "Business".
- 2) In the Solution Explorer add references to the **Model**, **DataAccess** and **Telerik.OpenAccess** assemblies.
- 3) Add a new class to the project named "ContactBO.cs" or if using VB "ContactBO.vb" and add the code below to the class.

This class will reference the Model, DataAccess and Telerik.OpenAccess namespaces and only contain methods GetContactGroups(), GetContacts() and SaveContacts(). We will include additional business logic later but for now just add calls to GetContactGroups(), GetContacts() and SaveContacts() using the data layer ContainerManager LoadChangeSet() an SaveChangeSet() methods.

```

Imports DataAccess
Imports Model
Imports Telerik.OpenAccess

Public Class ContactBO
    Public Shared Function GetContactGroups() As ObjectContainer.ChangeSet
        Return ContainerManager.LoadChangeSet(Of ContactGroup)()
    End Function

    Public Shared Function GetContacts() As ObjectContainer.ChangeSet
        Return ContainerManager.LoadChangeSet(Of Contact)()
    End Function

    Public Shared Function SaveContacts( _
        ByVal changeSet As ObjectContainer.ChangeSet) _
        As ObjectContainer.ChangeSet
        Return ContainerManager.SaveChangeSet(changeSet)
    End Function
End Class

```

```

using DataAccess;
using Model;
using Telerik.OpenAccess;

namespace Business
{
    public class ContactBO
    {
        public static ObjectContainer.ChangeSet GetContactGroups()
        {
            return ContainerManager.LoadChangeSet<ContactGroup>();
        }

        public static ObjectContainer.ChangeSet GetContacts()
        {
            return ContainerManager.LoadChangeSet<Contact>();
        }

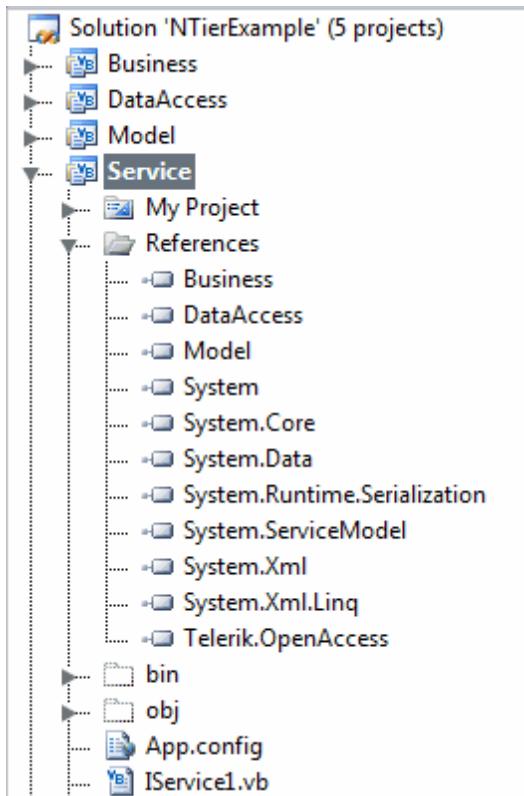
        public static ObjectContainer.ChangeSet SaveContacts(
            ObjectContainer.ChangeSet changeSet)
        {
            return ContainerManager.SaveChangeSet(changeSet);
        }
    }
}

```

Changing the Service

We have gutted the service and don't really need to keep much in it. Only a few calls to the business layer are really required.

- 1) In the Solution Explorer add a reference to the **Business** and **DataAccess** assembly. You can remove the unnecessary **Telerik.OpenAccess.Query** reference. The services references in the Solution Explorer should look like the screenshot below:



- 2) Change the Service1.cs or Service1.vb file's code to call the ContactBO class.

Have each service call the corresponding method on the business layer ContactBO object.

```
Imports Business
Imports Model
Imports Telerik.OpenAccess

Public Class Service1
    Implements IService1

    Public Function IService1_GetContacts() As ObjectContainer.ChangeSet
        Implements IService1.GetContacts
        Return ContactBO.GetContacts()
    End Function

    Public Function IService1_GetContactGroups() As ObjectContainer.ChangeSet
        Implements IService1.GetContactGroups
        Return ContactBO.GetContactGroups()
    End Function

    Public Function IService1_SaveContacts(ByVal changeset As ObjectContainer.
        ChangeSet) As ObjectContainer.ChangeSet Implements IService1.SaveContacts
        Return ContactBO.SaveContacts(changeset)
    End Function

    Private Shared Function GetChangeSet(ByVal listName As String, _
        ByVal scope As IObjectScope, _
        ByVal queryable As Object) As ObjectContainer.ChangeSet

        Dim container As New ObjectContainer()

        ' define the fetch groups used in the copy, by default all columns
        Dim collector As IObjectCollector = _
            New FetchGroupCollector(FetchGroupCollector.DefaultFetchGroup)

        container.CopyFrom(scope, queryable.[GetType]().Name, _
            queryable, collector)

        ' get everything in the container as a changeset
        Dim changeSet As ObjectContainer.ChangeSet = container.GetContent()

        Return changeSet
    End Function

End Class
```

```

    using Business;
    using Telerik.OpenAccess;

    namespace Service
    {
        public class Service1 : IService1
        {
            public ObjectContainer.ChangeSet GetContactGroups()
            {
                return ContactBO.GetContactGroups();
            }

            public ObjectContainer.ChangeSet GetContacts()
            {
                return ContactBO.GetContacts();
            }

            public ObjectContainer.ChangeSet SaveContacts(
                ObjectContainer.ChangeSet changeSet)
            {
                return ContactBO.SaveContacts(changeSet);
            }
        }
    }

```

- 3) Right click on the **ServiceReference1** Service Reference and select "Update Service Reference" to pick up the changes to the Service1 service
- 4) Run the application. Re-test the functionality by adding, editing and deleting contacts. Also try clicking the asterisked row labeled "Click here to add new row", then press ESC to cancel the add. Also try all the above with and without clicking the Save button.

4.8 N-Tier With Business Rules

Adding Business Rules

Now that the N-Tier framework is set up let's slip in a token business rule. In addition, we'll raise an exception in the Business layer and use the WCF "fault" mechanism to communicate the problem back to the presentation layer. We will need to change the following layers:

- **Data Access:** A new method LoadQueryable<T>() that will return an IQueryable given a ChangeSet. This will be used in the business layer to convert a ChangeSet to an IQueryable<Contact> that can be validated for phone numbers with a given format.
- **Business:** The SaveContacts() method will no longer just save without checking the incoming data. A new method InvalidPhoneContacts() will use a LINQ query to check the phone number format against a regular expression. The SaveContracts() method will throw an exception if the check fails on one or more phone numbers.
- **Service:** The service SaveContracts() method will trap exceptions thrown by the business layer. SaveContracts() will forward the exception by way of throwing a new FaultException. FaultException

allows us to serialize exceptions for sending over the wire during WCF requests.

- **Presentation Layer:** The WinForm client will catch the FaultException returned from the service client proxy. The form will display a meaningful error message to the user and will roll back changes to the last known good state.



Find the source at \Projects\ORM\<CS\VB>\NTierExampleBR\NTierExampleBR.sln

Changing the Data Access Layer

- 1) In the DataAccess assembly, ContainerManager.cs class, add a reference to the **System.Linq** namespace in your "Imports" (VB) or "using" (C#) section of code.
- 2) Add a new LoadQueryable<T>() method to the ContainerManager class.

The method should take a ChangeSet and return an IQueryable<T>. Do this by creating an ObjectContainer, calling the Apply() method to add the ChangeSet data to the container and finally calling the Extent<T>() method

```
④ Public Shared Function LoadQueryable(Of T) ( _
    ByVal changeSet As ObjectContainer.ChangeSet) _ 
    As IQueryable(Of T)
    Dim container As New ObjectContainer()
    container.Apply(changeSet)
    Return container.Extent(Of T) () .AsQueryable(Of T) ()
End Function

Public Shared Function LoadQueryable(Of T) ( _
    ByVal changeSet As ObjectContainer.ChangeSet) _ 
    As IQueryable(Of T)
    Dim container As New ObjectContainer()
    container.Apply(changeSet)
    Return container.Extent(Of T) () .AsQueryable()
End Function
```

```
④ public static IQueryable<T> LoadQueryable<T>(
    ObjectContainer.ChangeSet changeSet)
{
    ObjectContainer container = new ObjectContainer();
    container.Apply(changeSet);
    return container.Extent<T>().AsQueryable<T>();
}
```

Changing the Business Layer

When changes are sent to the business layer for saving, the Contacts phone numbers are validated against a regular expression. If any are invalid, an exception is thrown.

- 1) In the Business project, open the Properties node using the Solution Explorer. Click on Settings. Add a new String setting "RegExPhone" with a value of "^((?([0-9]{3}))?)?-?[0-9]{3}?-?[0-9]{4}\$" w/o the quotes.

This will validate phone numbers to the format "(999)999-9999". Your production application would likely require a more complete regular expression to handle international codes, other punctuation and layout, etc. We're using this very specific format to demonstrate validation and communication back to the client.

	Name	Type	Scope	Value
▶	RegExPhone	string	Application	<code>^(\(?[0-9]{3}\)?)(-[0-9]{3})-[0-9]{4}\$</code>
*				

Figure 73

2) In the Business project, add a new class "ContactException.cs" or "ContactException.vb"

ContactException should inherit from ApplicationException. Override the constructor to accept a message and a generic List of Contact. Include properties for a generic List of Contact, and a read-only property to retrieve the list of contact names with invalid phone numbers.

We're not sending back a List of Contact to the client from the service because Contact is marked with attribute Telerik.OpenAccess.Persistent and must be part of an ObjectScope or ObjectContainer. Instead, we will send back a simple array of string containing the contact names.

```

Imports System
Imports System.Collections.Generic
Imports Model

Public Class ContactException
    Inherits ApplicationException
    Public Sub New(ByVal message As String, ByVal contacts As List(Of Contact))
        MyBase.New(message)
        Me.Contacts = contacts
    End Sub
    Private privateContacts As List(Of Contact)
    Public Property Contacts() As List(Of Contact)
        Get
            Return privateContacts
        End Get
        Set(ByVal value As List(Of Contact))
            privateContacts = value
        End Set
    End Property

    Public ReadOnly Property NameList() As List(Of String)
        Get
            Dim names As List(Of String) = New List(Of String)()
            For Each contact As Contact In Me.Contacts
                names.Add(contact.ContactName)
            Next contact
            Return names
        End Get
    End Property
End Class

```

```
using System;
using System.Collections.Generic;
using Model;

namespace Business
{
    public class ContactException : ApplicationException
    {
        public ContactException(string message, List<Contact> contacts)
            : base(message)
        {
            this.Contacts = contacts;
        }
        public List<Contact> Contacts { get; set; }

        public List<string> NameList
        {
            get
            {
                List<string> names = new List<string>();
                foreach (Contact contact in this.Contacts)
                {
                    names.Add(contact.ContactName);
                }
                return names;
            }
        }
    }
}
```

3) Open ContactBO.cs in the editor and add the business rule logic.

- Add the **System.Linq** namespace to the "Imports" (VB) or "using" (C#) section of code.
- Add a `InvalidPhoneContacts()` method that accepts an `IQueryable` of `Contact`, constructs a LINQ query to select only those where the Phone fails the regular expression validation.

```
Private Shared Function InvalidPhoneContacts( _
    ByVal queryable As IQueryable(Of Contact)) _
    As IQueryable(Of Contact)
    Dim regexPhone As New System.Text.RegularExpressions.Regex( _
        My.Settings.Default.RegExPhone)

    Dim invalidContacts As IQueryable(Of Contact) = _
        From q In queryable _
        Where regexPhone.Matches(q.Phone).Count = 0 _
        Select q

    Return invalidContacts
End Function
```

```

④ private static IQueryables<Contact> InvalidPhoneContacts(
    IQueryables<Contact> queryable)
{
    System.Text.RegularExpressions.Regex regexPhone =
        new System.Text.RegularExpressions.Regex(
            Properties.Settings.Default.RegExPhone);

    IQueryables<Contact> invalidContacts =
        from q
        in queryable
        where regexPhone.Matches(q.Phone).Count == 0
        select q;

    return invalidContacts;
}

```

- c) Re-write the SaveContacts() method to test for valid phone numbers and throw a ContactException if any fail the validation.

This method uses the new data access ContainerManager method LoadQueryables<T> to get an IQueryables<Contact>, suitable for LINQ queries. The IQuerables is passed to the InvalidPhoneContacts() method and any contacts with invalid phone numbers are returned in "invalidContacts". If there are one or more invalid phone numbers, we throw a ContactException.

Note: you will not be able to send a list of Contact objects back to the client directly because they are marked with the Telerik.OpenAccess.Persistent attribute and would need to be part of a container or ObjectScope.

```

④ Public Shared Function SaveContacts( _
    ByVal changeSet As ObjectContainer.ChangeSet) _
    As ObjectContainer.ChangeSet
    Dim contacts As IQueryables<Of Contact> = _
        ContainerManager.LoadQueryables<Of Contact>(changeSet)
    Dim invalidContacts As IQueryables<Of Contact> = _
        InvalidPhoneContacts(contacts)
    If invalidContacts.Count() > 0 Then
        Dim message As String = "Invalid phone numbers"
        Dim businessException As New ContactException(message, _
            invalidContacts.ToList())
        Throw businessException
    End If

    Return ContainerManager.SaveChangeSet(changeSet)
End Function

```

```
public static ObjectContainer.ChangeSet SaveContacts(
    ObjectContainer.ChangeSet changeSet)
{
    IQueryable<Contact> contacts =
        ContainerManager.LoadQueryable<Contact>(changeSet);

    IQueryable<Contact> invalidContacts =
        InvalidPhoneContacts(contacts);
    if (invalidContacts.Count() > 0)
    {
        string message = "Invalid phone numbers";

        ContactException businessException =
            new ContactException(message,
                invalidContacts.ToList<Contact>());
        throw businessException;
    }

    return ContainerManager.SaveChangeSet(changeSet);
}
```

Changing the Service

The service layer catches exceptions from the business layer and forwards them to the client as WCF FaultExceptions.

- 1) In the Service project, add a new class "ContactFault.cs" or "ContactFault.vb". The class will need a reference to the **System.Runtime.Serialization** namespace to support the **DataContract** attribute so the fault can be sent like any other piece of WCF data. Also reference the **System.Collections.Generic** namespace to support generics. Mark the ContactFault class with the **DataContract** attribute. Add a generic List of string property "ContactNames" and mark the property with the **DataMember** attribute. Add a constructor that accepts a generic List of string.

*You can add other information to properties of this class to suit your purpose and mark each property with the **DataMember** attribute.*

```

Imports System.Runtime.Serialization
Imports System.Collections.Generic

<DataContract()> _
Public Class ContactFault
    Public Sub New(ByVal contactNames As List(Of String))
        Me.ContactNames = contactNames
    End Sub

    Private privateContactNames As List(Of String)
    <DataMember()> _
    Public Property ContactNames() As List(Of String)
        Get
            Return privateContactNames
        End Get
        Set(ByVal value As List(Of String))
            privateContactNames = value
        End Set
    End Property
End Class

```

```

using System.Runtime.Serialization;
using System.Collections.Generic;

namespace Service
{
    [DataContract]
    public class ContactFault
    {
        public ContactFault(List<string> contactNames)
        {
            this.ContactNames = contactNames;
        }

        [DataMember]
        public List<string> ContactNames { get; set; }
    }
}

```

- 2) Open the "IService1.cs" or "IService1.vb" interface file for editing and add a FaultContract attribute to the SaveContacts() method.

This allows ContactFault to be used to communicate exception information in the context of a WCF method.

```

<OperationContract(), FaultContract(GetType(ContactFault))> _
Function SaveContacts(ByVal changeSet As ObjectContainer.ChangeSet) _
As ObjectContainer.ChangeSet

```

```

[OperationContract]
[FaultContract(typeof(ContactFault))]
ObjectContainer.ChangeSet SaveContacts(ObjectContainer.ChangeSet changeSet);

```

- 3) Open "Service1.cs" or "Service1.vb" and change the SaveContacts() method implementation to catch ContactException and respond by throwing a FaultException of ContactFault. The ContactException NameList property can be passed directly to the ContactFault constructor. To provide other information about the fault, include a FaultReason and FaultCode in the FaultException constructor.

* Make sure the System.ServiceModel is referenced in the Service1.cs code file

Be aware that the generic List of string "NameList" will be converted automatically by the service to a more interoperable-happy array of string.

```
④ Public Function IService1_SaveContacts(ByVal changeset As ObjectContainer.
    ChangeSet) _
    As ObjectContainer.ChangeSet Implements IService1.SaveContacts
    Try
        Return ContactBO.SaveContacts(changeset)
    Catch ex As ContactException
        Throw New FaultException(Of ContactFault) _
            (New ContactFault(ex.NameList), _
            New FaultReason(ex.Message), New FaultCode("Invalid phone"))
    End Try
End Function
```

```
④ public ObjectContainer.ChangeSet SaveContacts(
    ObjectContainer.ChangeSet changeSet)
{
    try
    {
        return ContactBO.SaveContacts(changeSet);
    }
    catch (ContactException ex)
    {
        throw new FaultException<ContactFault>(new ContactFault(ex.NameList),
            new FaultReason(ex.Message), new FaultCode("Invalid phone"));
    }
}
```

Changing the Presentation Layer (WinForm Client)

All that remains to do is to catch the FaultException on the client, display a message for the user and to clean up invalid data. In this example we will simply roll back the bad data to a known good state.

- 1) First open ContainerHelper.cs or "ContainerHelper.vb" for editing and add a Reset() method to remove any new Contact objects from the container.

ObjectContainer has several handy methods for determining the state of contained objects including IsDirty(), IsRemoved() and IsReadOnly().

```
■ ' Remove 'new' objects of type T
Public Sub Reset(Of T)()
    _container.Transaction.Begin()
    Dim list As IList(Of T) = _container.Extent(Of T)()
    For Each obj As Object In list
        If _container.IsNew(obj) Then
            _container.Remove(obj)
        End If
    Next obj
End Sub
```

```
■ // Remove "new" objects of type T
public void Reset<T>()
{
    _container.Transaction.Begin();
    IList<T> list = _container.Extent<T>();
    foreach (object obj in list)
    {
        if (_container.IsNew(obj))
            _container.Remove(obj);
    }
}
```

- 2) Make sure **System.ServiceModel** and **WinformClient.ServiceReference1** are in the using or Imports of the Form1 class.
- 3) Change the "Save" button click event handler to catch the FaultException of ContactFault. Construct a helpful error message using the exception instance ContactNames property. Notice that you can get at the ContactFault using the Exception.Detail and casting to the appropriate type. Display the error in a MessageBox, then call ContainerHelper.Reset() to remove the new items. Finally, call LoadGrid() to refresh the grid.

```
④ Private Sub btnSave_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnSave.Click
    Try
        ' pass SaveContacts as a delegate
        _containerHelper.Save(AddressOf _client.SaveContacts)
        ' Catch FaultException ex As FaultException(Of ContactFault)
    Catch ex As FaultException(Of ContactFault)
        ' construct an error message that lists the invalid
        ' contact names
        Const format As String = "{0}. Rolling back changes for ""{1}"""
        Dim contactNames As String =
            String.Join(", ", (TryCast(ex.Detail, ContactFault)).ContactNames)
        Dim message As String =
            String.Format(format, ex.Reason.GetMatchingTranslation().Text, _
                contactNames)

        ' use the FaultException to get the reason, code, subcode and respond.
        MessageBox.Show(ex.Reason.GetMatchingTranslation().Text & _
            ". Rolling back changes for " & contactNames)

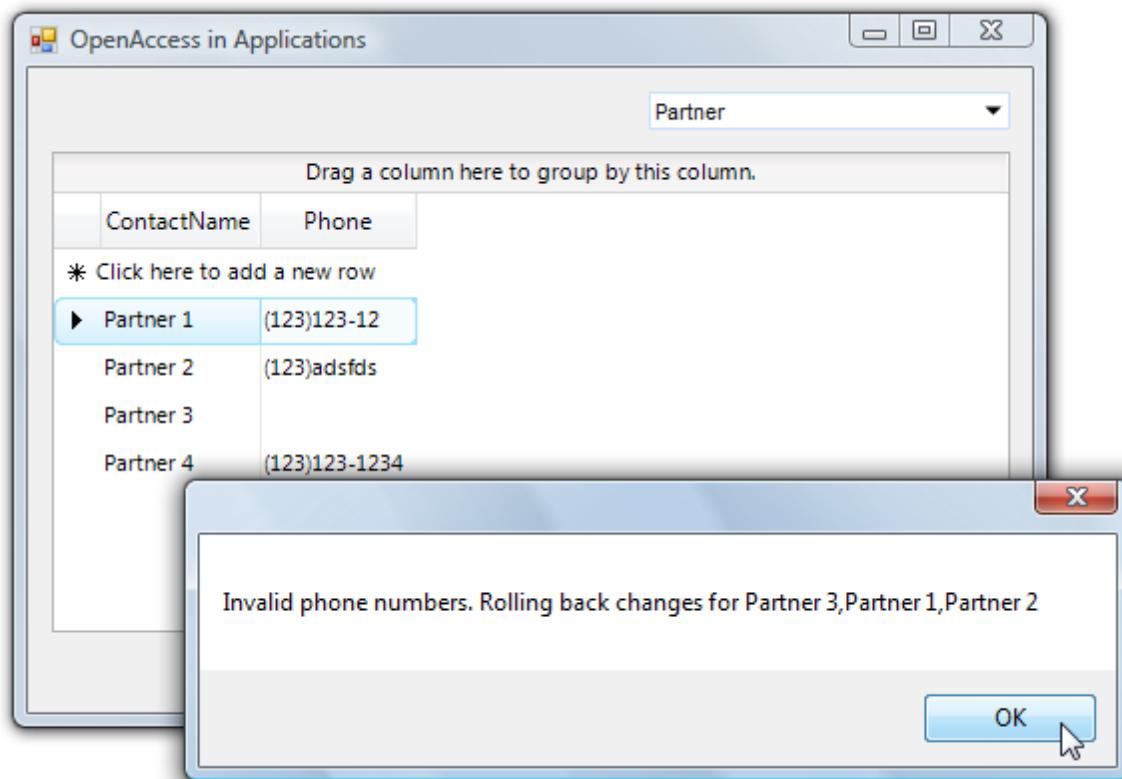
        ' remove 'new' Contact objects and reload grid
        _containerHelper.Reset(Of Contact)()
        LoadGrid()
    End Try
End Sub
```

```
④ // send current changes to service
private void btnSave_Click(object sender, EventArgs e)
{
    try
    {
        // pass SaveContacts as a delegate
        _containerHelper.Save(_client.SaveContacts);
    }
    catch (FaultException<ContactFault> ex)
    {
        // construct an error message that lists the invalid
        // contact names
        const string format = "{0}. Rolling back changes for \"{1}\\"";
        string contactNames =
            string.Join(", ", (ex.Detail as ContactFault).ContactNames);
        string message = String.Format(format,
            ex.Reason.GetMatchingTranslation().Text, contactNames);

        // use the FaultException to get the reason, code, subcode and respond.
        MessageBox.Show(ex.Reason.GetMatchingTranslation().Text +
            ". Rolling back changes for " + contactNames);

        // remove "new" Contact objects and reload grid
        _containerHelper.Reset<Contact>();
        LoadGrid();
    }
}
```

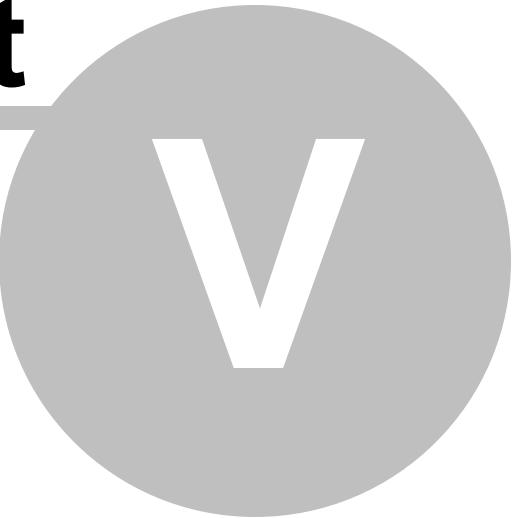
- 4) Before Running the application, in the Visual Studio IDE, Click on Debug->Exceptions->Expand the Common Language Runtime Exceptions and find System.ServiceModel in the List.Uncheck the Box under the User-unhandled column. This needs to be done or the debugger will assume that the exception thrown by the service is not handled and will break when trying to test the service. If you prefer not to uncheck the box, you can let the debugger break and then simply click continue and you should see a result similar to below,
- 5) Run the application. Try entering both valid phone numbers of the format "(999)999-9999" and invalid phone numbers.



4.9 Wrap Up

In this section you learned how to integrate OpenAccess with a number of presentation platforms including ASP.NET, ASP.NET/AJAX, WinForms, Telerik Reporting and using web services. You became familiar with some of the architectural possibilities for N-Tier applications that respect principles of multi-tiered application design.

Part



V

References

5 References

This chapter explores how OpenAccess handles references between objects including one-to-one, one-to-many and many-to-many relationships. The chapter also shows how to handle self referential tables using OpenAccess.

In this chapter you will learn:

- How to create a reference from one object to another.
- How to transform a reference into a one-to-many relationship.
- How to use OpenAccess to map "join" table, many-to-many relationships.
- How to map a self referencing table.



Find the source projects for this chapter at \Projects\ORM\CS\7_References\7_References.sln

5.1 References

OpenAccess allows you to setup references in one object to other objects. Let's use the Northwind Orders and Order Details tables to create an example. In the Reverse Mapping Wizard, the two tables look something like the example below. The two tables have been prepared for this example to only show data fields and have been stripped of any references.

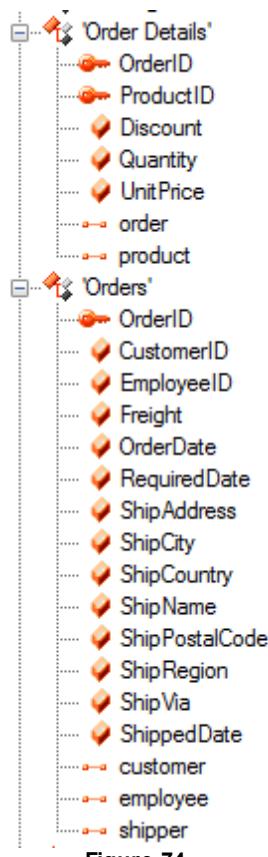


Figure 74

If you click the OrderID column of the Order Details table you will see a "Create Ref" button.

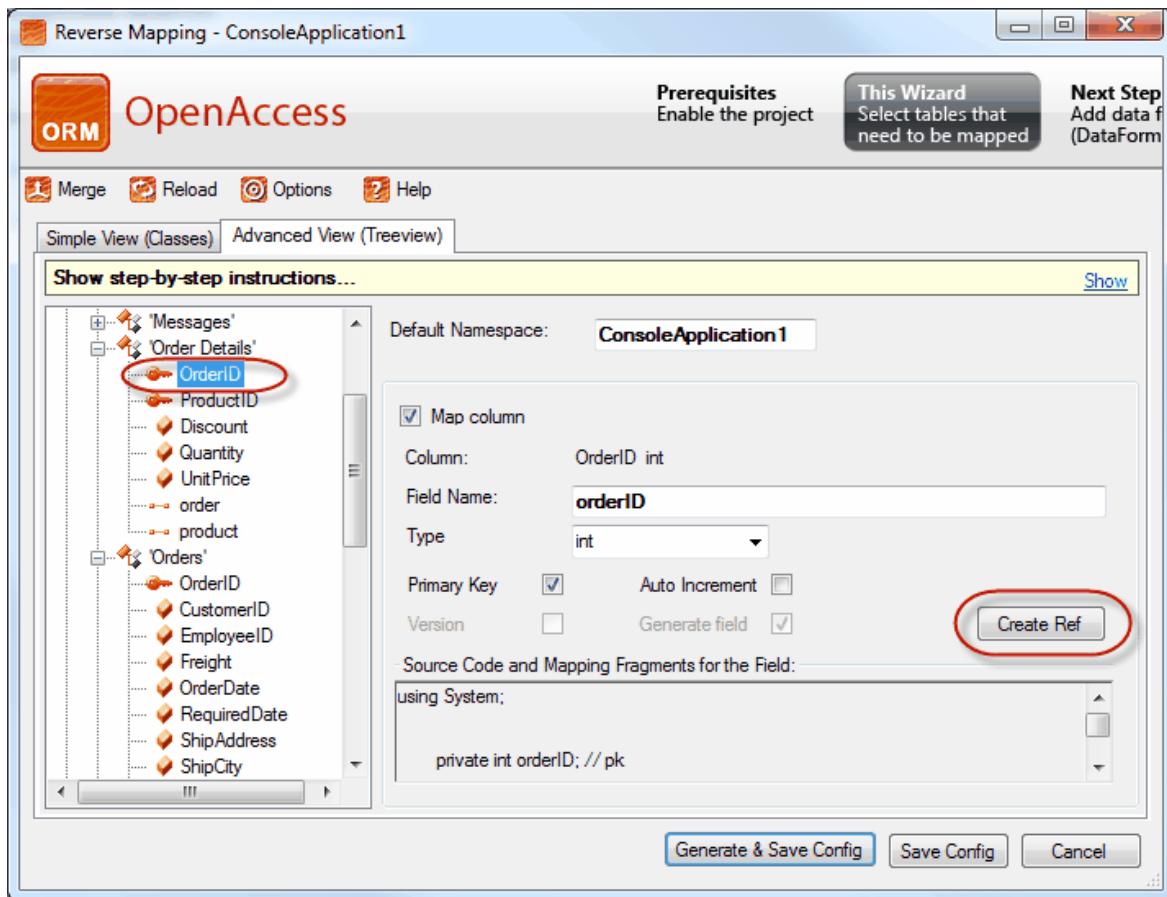


Figure 75

Clicking Create Ref adds a new node to the tree. Once the reference is created you can rename it using the "Field Name" entry. Use the "Type" drop down list to choose a class name. Instead of the Type being the default "object", in the figure below we're indicating that the reference points to an Order type. The figure below is a one-to-one reference which means that you can access the one order for that order detail using syntax similiar to "OrderDetails.Order".

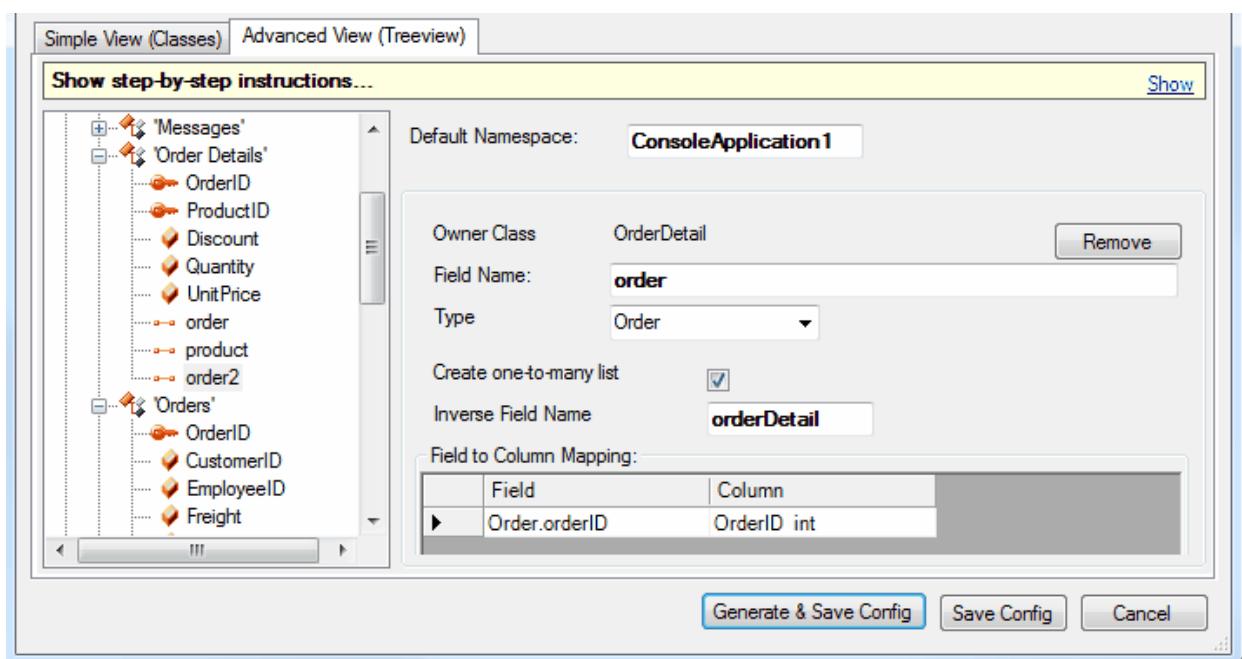


Figure 76

- The reference node icon indicates the type of relationship, i.e. one-to-one, one-to-many or many-to-many:
 - One-to-one
 - One-to-many
 - Many-to-many

Figure 77

- G** In the "Field to Column Mapping" entry the object *Field* is mapped to the database *Column*; the **o** column entry includes the column name followed by its database type. In some cases you may **t** need to enter the column name and type by hand. As of this writing, failing to include the column **c** name and type may cause some hard-to-track-down exceptions at run time.

h**a****!****©**

You can select the "Create one-to-many list" check box to place a new reference in the *Orders* table.

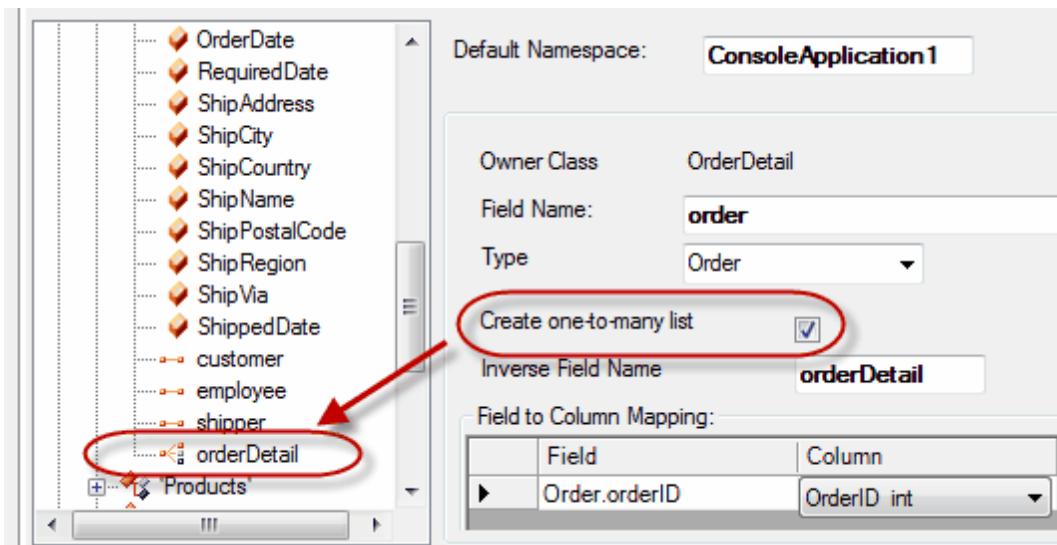


Figure 78

This creates an `IList` in the `Orders` property so that you can use syntax similar to `"Order.OrderDetails"` when you want to iterate or bind all the order details for a given order.

```

Public ReadOnly Property OrderDetails() As IList(Of OrderDetail)
    Get
        Return orderDetail
    End Get
End Property
```



```

public IList<OrderDetail> OrderDetails
{
    get { return orderDetail; }
}
```

Join Tables

For situations where you have a join table, for example the "Employees" and "Territories" tables are joined by "EmployeeTerritories". This configuration of data allows employees to have multiple territories and territories to have multiple employees. This data structure is mirrored by OpenAccess. The figure below shows the `EmployeeTerritories` with the Many-to-many check box unselected. At this point the `Employees` table has a one-to-many reference that allows you to use syntax similar to `"Employee.EmployeeTerritories"` which returns an `IList` of `Territory` objects for a given employee. The `Territory` object has no such reference back to employees.

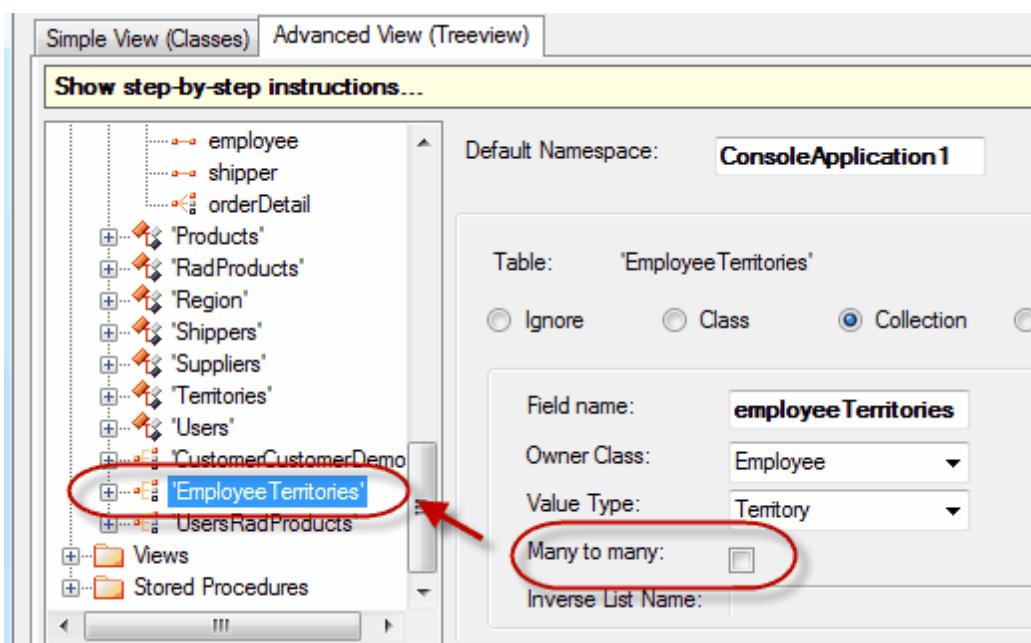


Figure 79

Once you click the "Many-to-many" check box, OpenAccess places a new reference to Employee in Territories. With this reference, you can access "Territory". Now you have data structures that with all the territories for a given employee and all the employees for a given territory.

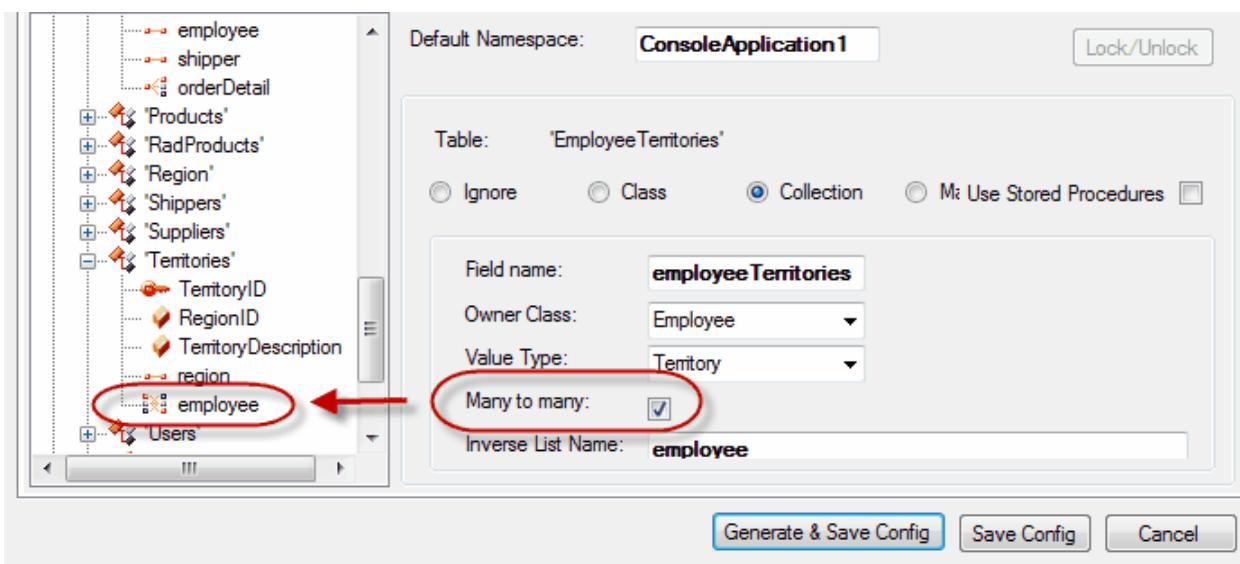


Figure 80

The code snippet below iterates all employees, lists each territory for each employee, then lists other employees in each territory.

```
④ Dim employees As List(Of Employee) = _
scope.Extent(Of Employee)().ToList()
For Each employee As Employee In employees
    Console.WriteLine(employee.FirstName & " " & _
employee.LastName)
    For Each territory As Territory In _
employee.EmployeeTerritories
        Console.WriteLine(Constants.vbTab & _
"Employees for territory: " & _
territory.TerritoryDescription)
        For Each emp As Employee In territory.Employees
            Console.WriteLine( _
Constants.vbTab + Constants.vbTab + _
emp.FirstName & _
" " & emp.LastName)
        Next emp
    Next territory
Next employee
```

```
④ List<Employee> employees = scope.Extent<Employee>().ToList();
foreach (Employee employee in employees)
{
    Console.WriteLine(employee.FirstName + " " + employee.LastName);
    foreach (Territory territory in employee.EmployeeTerritories)
    {
        Console.WriteLine("\tEmployees for territory: " +
territory.TerritoryDescription);
        foreach (Employee emp in territory.Employees)
        {
            Console.WriteLine("\t\t" + emp.FirstName + " " + emp.LastName);
        }
    }
}
```

Part of the console output is shown in the figure below. Be aware that the NorthwindOA database has only a single employee for each territory. Additional employees have been added to the join table to give the result for "Minneapolis" in this example.

```
Anne Dodsworth
Employees for territory: Hollis

    Anne Dodsworth
Employees for territory: Portsmouth

        Anne Dodsworth
Employees for territory: Southfield

            Anne Dodsworth
Employees for territory: Troy

                Anne Dodsworth
Employees for territory: Bloomfield Hills

                    Anne Dodsworth
Employees for territory: Roseville

                        Anne Dodsworth
Employees for territory: Minneapolis

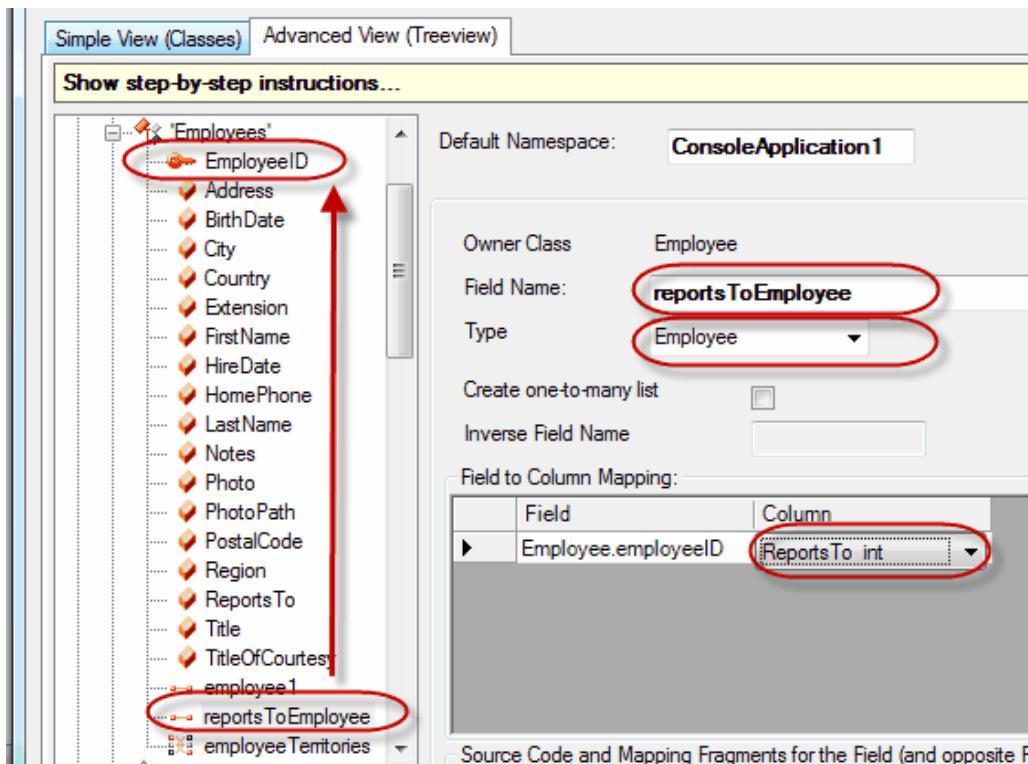
    Nancy Davolio
    Andrew Fuller
    Anne Dodsworth
```

Figure 81

5.2 Self Referencing

Self referencing is a special case where one of the columns references the primary key. For example, we have the Employee table again where EmployeeID is the primary key. The ReportsTo column for every employee (except the employee highest in the hierarchy) points to the EmployeeID in another record of the same table. Select the ReportsTo column and click the **Create Ref** button to create a new reference. We can rename this reference to something more meaningful ("ReportsToEmployee" in this case) and set its Type to be Employee.

If the Field to Column Mapping is not complete you may need to set the Column to "ReportsTo int" to indicate that the ReportsTo column is an "int" type in the database and maps to the Employee.employeeID field.



Source Code and Mapping Fragments for the Field (and opposite Fi
Figure 82

With this mapping we can find out who each employee reports to. The example below lists all the employees and who each employee directly reports to. The ReportsToEmployee property may be null in the case of a CEO who doesn't report to anyone.

```

Dim employees As List(Of Employee) = _
    scope.Extent(Of Employee)().ToList()
For Each employee As Employee In employees
    Console.WriteLine(employee.FirstName & " " & _
        employee.LastName)
    If employee.ReportsToEmployee IsNot Nothing Then
        Console.WriteLine(" reports to " & _
            employee.ReportsToEmployee.FirstName & " " & _
            employee.ReportsToEmployee.LastName)
    End If
Next employee

```

```

List<Employee> employees =
    scope.Extent<Employee>().ToList();
foreach (Employee employee in employees)
{
    Console.Write(employee.FirstName + " " +
        employee.LastName);
    if (employee.ReportsToEmployee != null)
    {
        Console.WriteLine(" reports to " +
            employee.ReportsToEmployee.FirstName + " " +
            employee.ReportsToEmployee.LastName);
    }
}

```

Click the "Create one-to-many list" check box to create a second collection. Name this collection using the "Inverse Field Name" entry. In the example below we're naming the collection "subordinates" as it contains all the employees that have a ReportsTo that points to a given employee.

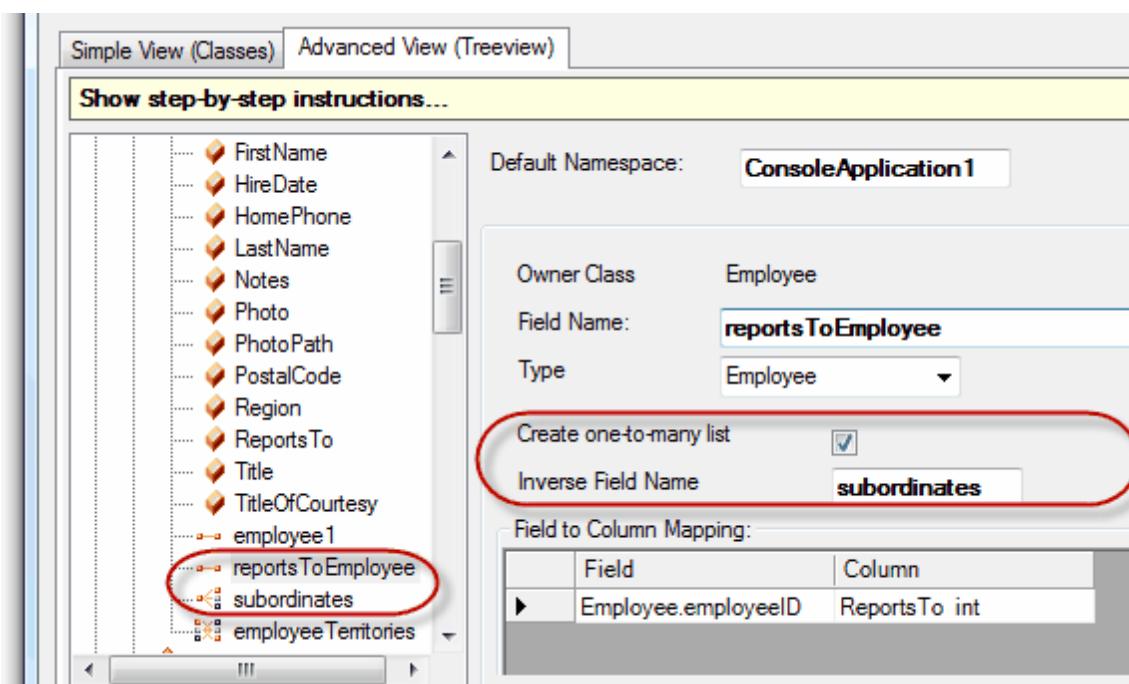


Figure 83

Now we can access the Employee.Subordinates collection as demonstrated in the example below.

```
④ For Each employee As Employee In employees
    Console.WriteLine(employee.FirstName & " " & _
employee.LastName)
    If employee.ReportsToEmployee IsNot Nothing Then
        Console.WriteLine(" reports to " & _
employee.ReportsToEmployee.FirstName & _
" " & employee.ReportsToEmployee.LastName)
    End If
    If employee.Subordinates.Count > 0 Then
        Console.WriteLine(Constants.vbTab & "Employees:")
        For Each emp As Employee In employee.Subordinates
            Console.WriteLine(Constants.vbTab + _
Constants.vbTab +
emp.FirstName & " " & emp.LastName)
        Next emp
    End If
Next employee
```

```
④ foreach (Employee employee in employees)
{
    Console.Write(employee.FirstName + " " +
employee.LastName);
    if (employee.ReportsToEmployee != null)
    {
        Console.WriteLine(" reports to " +
employee.ReportsToEmployee.FirstName + " " +
employee.ReportsToEmployee.LastName);
    }
    if (employee.Subordinates.Count > 0)
    {
        Console.WriteLine("\tEmployees:");
        foreach (Employee emp in employee.Subordinates)
        {
            Console.WriteLine("\t\t" + emp.FirstName + " " + emp.LastName);
        }
    }
}
```

The output looks something like the figure below where "Steven Buchanan" reports to "Andrew Fuller", "Michael Suyama", "Robert King" and "Anne Dodsworth" report to Steven.

```
Nancy Davolio reports to Andrew Fuller
Andrew Fuller Employees:
    Nancy Davolio
    Janet Leverling
    Margaret Peacock
    Steven Buchanan
    Laura Callahan
Janet Leverling reports to Andrew Fuller
Margaret Peacock reports to Andrew Fuller
Steven Buchanan reports to Andrew Fuller
Employees:
    Michael Suyama
    Robert King
    Anne Dodsworth
Michael Suyama reports to Steven Buchanan
Robert King reports to Steven Buchanan
Laura Callahan reports to Andrew Fuller
Anne Dodsworth reports to Steven Buchanan
```

Figure 84

5.3 Wrap Up

In this chapter you learned how OpenAccess handles references between objects. You learned how to create a simple reference from one object to another and how to transform that reference into a one-to-many relationship. You also learned how OpenAccess can be used to map "join" tables that represent many-to-many relationships. Finally, you learned how to map a self referencing table.

Part



VI

Inheritance

6 Inheritance

This chapter demonstrates how OpenAccess handles inheritance to get the best mix of performance, data storage and conformity between the database and persistent objects.

In this chapter you will learn:

- The multiple strategies employed by OpenAccess to map inherited classes to tables including Flat, Vertical, Horizontal and Mixed mapping.
- You will learn about the tradeoffs involved when implementing each kind of strategy.
- You will learn how to configure for each strategy using the Forward Mapping Wizard.



Find the source at:

\Projects\ORM\CS\8A_Inheritance\8_Inheritance.sln (Starting point project with classes but no configuration set)

and

\Projects\ORM\CS\8B_Inheritance\8_Inheritance.sln (configured project)

6.1 Inheritance Overview

The "impedance mismatch" between database tables and objects shows up clearly when trying to map inherited classes. Suppose you have an "Animal" class with "Bird" and "Dog" descendants and further suppose there are base properties and methods as well as descendant specific properties and methods, how is this represented in relational table form? OpenAccess supports four inheritance mapping strategies:

1. **Flat mapping:** All properties for all classes are mapped to a single table. A "discriminator" column identifies the class type for each row. This strategy provides good performance at the cost of consuming extra space in the database. This is the default mapping strategy. **Pros:** No extra joins are required. Insert/update/delete operations are performed against the single table. **Cons:** The extra columns are not always used and the extra discriminator column also takes up space.
2. **Vertical mapping:** Each class has its own table with only the fields for that class. In this strategy the database is more closely aligned with the class model it's imitating. A discriminator column is still required, but only for the base class. **Pros:** the database is better normalized, subclasses can be added easily and a discriminator column is not required. **Cons:** Database operations for this strategy require multiple joins and multiple insert/update/delete statements are required.
3. **Mixed (flat and vertical) mapping:** OpenAccess allows a different strategy for each class. You can mix any combination of flat and vertical mapping to achieve the maximum performance and database space usage. For example, you can put often used data in a base class and rarely used data in a descendant class.
4. **Horizontal mapping:** In this strategy the base class is marked as abstract and is not represented in the database. Each derived class is stored in its own table with a copy of the fields in the base class. **Pros:** joins are not required so you can expect performance to be as fast as flat mapping. Common attributes can all be defined in the base class. **Cons:** each derived class starts a new hierarchy and the database is de-normalized.

Let's look at a often-used example inheritance chain where a base class of "Animal" inherits to "Dog" and "Bird" classes. "Bird" will also have a "Parrot" descendant. The figure below shows a diagram of these

classes, their properties and methods.

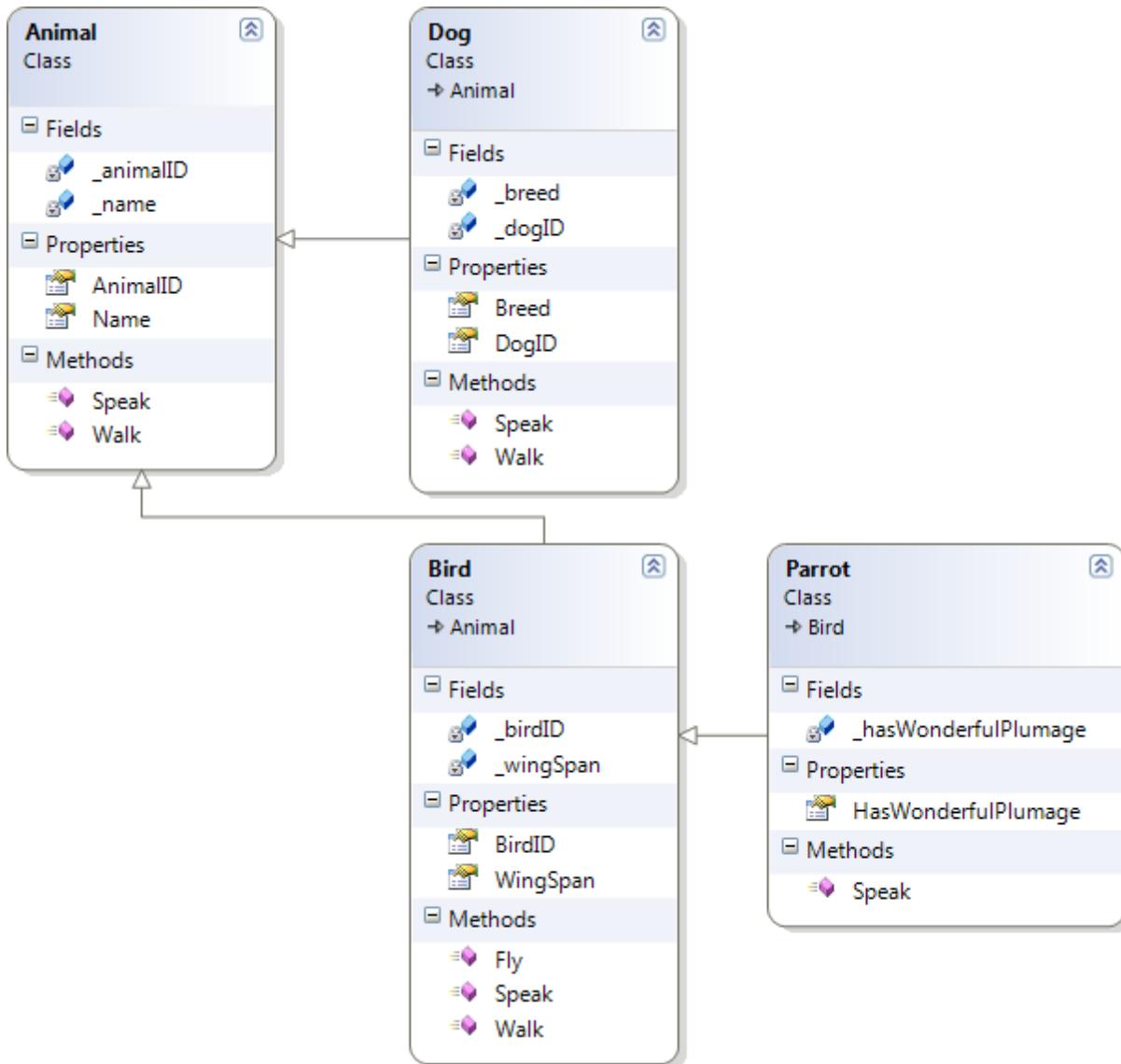


Figure 85

6.2 Flat Mapping

The default mapping strategy is "Flat", is the most efficient in terms of database processing and is more closely allied with the relational model than the object oriented way of looking at the world. If we use the "Flat" strategy against the objects and create the database, we get a single "Animal" table with all the data properties from all the classes as shown in the figure below. Notice particularly the "voa_class" column, known as a "discriminator" column that identifies which class is actually being stored.

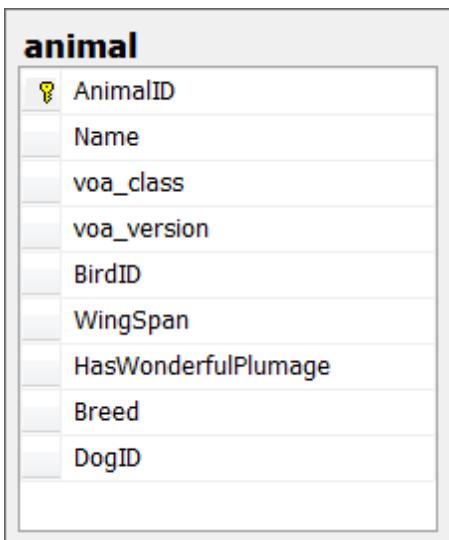


Figure 86

- Also notice that only properties are saved and that metadata about methods is lost in translation. If you were to forward map a set of classes, then reverse map the tables to reconstitute the classes, the class methods would be gone.

Let's say that Animal has a voa_class = "111" and voa_class for Dog is "222". You populate one Animal and one Dog object and persist them to the database using OpenAccess.

```
Shared Sub Main(ByVal args() As String)
    Using scope As IObjectScope =
        ObjectScopeProvider1.GetNewObjectScope()
        scope.Transaction.Begin()

        Dim animal As New Animal()
        animal.Name = "Bob"
        scope.Add(animal)

        Dim dog As New Dog()
        dog.Name = "Fluffy"
        dog.Breed = "Mutt"
        dog.Speak()
        scope.Add(dog)

        scope.Transaction.Commit()
        Console.ReadKey()
    End Using
End Sub
```

```

    static void Main(string[] args)
    {
        using (IObjectScope scope =
ObjectScopeProvider1.GetNewObjectScope())
        {
            scope.Transaction.Begin();

            Animal animal = new Animal();
            animal.Name = "Bob";
            scope.Add(animal);

            Dog dog = new Dog();
            dog.Name = "Fluffy";
            dog.Breed = "Mutt";
            dog.Speak();
            scope.Add(dog);

            scope.Transaction.Commit();
            Console.ReadKey();
        }
    }
}

```

The first record of this table will have a voa_class = "111", Breed, WingSpan and HasWonderfulPlumage will all be null. The second record will have a voa_class = "222", Breed will be populated but WingSpan and HasWonderfulPlumage will be null.

AnimalID	Name	voa_class	WingSpan	HasWonderf...	Breed
1	Bob	111	NULL	NULL	NULL
2	Fluffy	222	NULL	NULL	Mutt

Figure 87

6.3 Vertical Mapping

A vertical mapping of this same class hierarchy will look much more like its object oriented counterpart. Now the database diagram looks almost identical to the class diagram shown earlier.

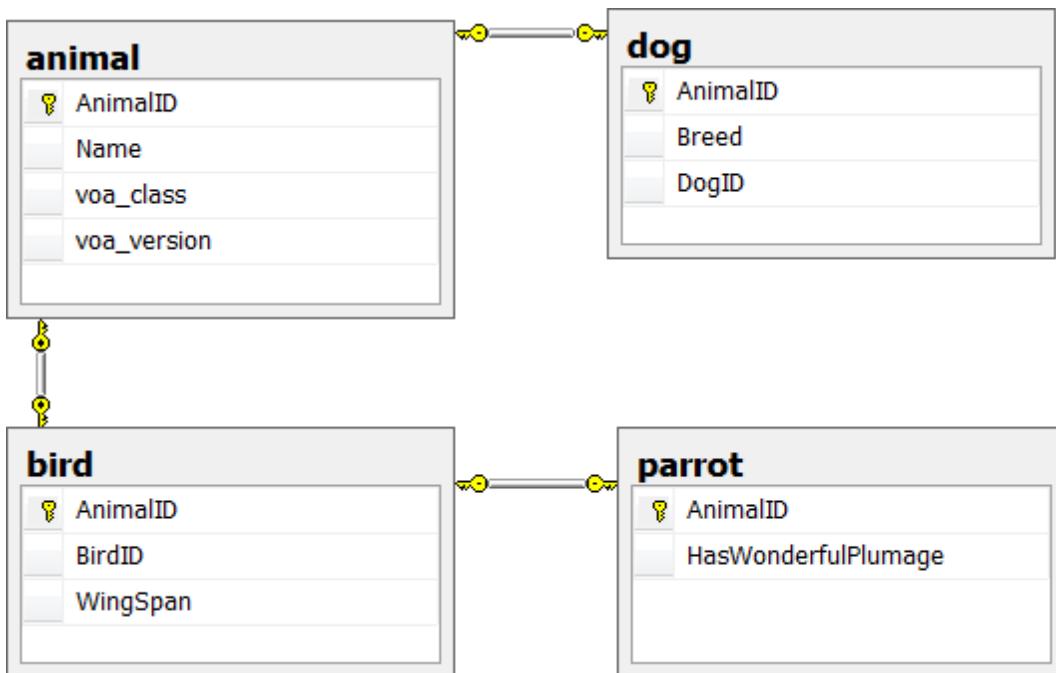


Figure 88

Using the same code snippet that persists an Animal and Dog, the resulting records look like the series of screenshots below. The database is more normalized where the Animal table only has data specific to the base class, i.e. "Name"

AnimalID	Name	voa_class	voa_version
1	Bob	111	1
2	Fluffy	222	1

Figure 89

The Dog table has a AnimalID column (a foreign key back to the Animal table), and a dog-specific Breed column.

AnimalID	Breed
2	Mutt

Figure 90

6.4 Mixed Flat and Vertical Mapping

Using a mixed mapping strategy we can use vertical mapping for parts of the hierarchy that should stay normalized and other objects can be flattened into their ancestor classes. For example, if we keep Dog and Bird classes using the vertical strategy and the seldom-used Parrot class using the flat strategy, the database diagram looks like the example below. The Parrot HasWonderfulPlumage property has been flattened into the Bird ancestor class.

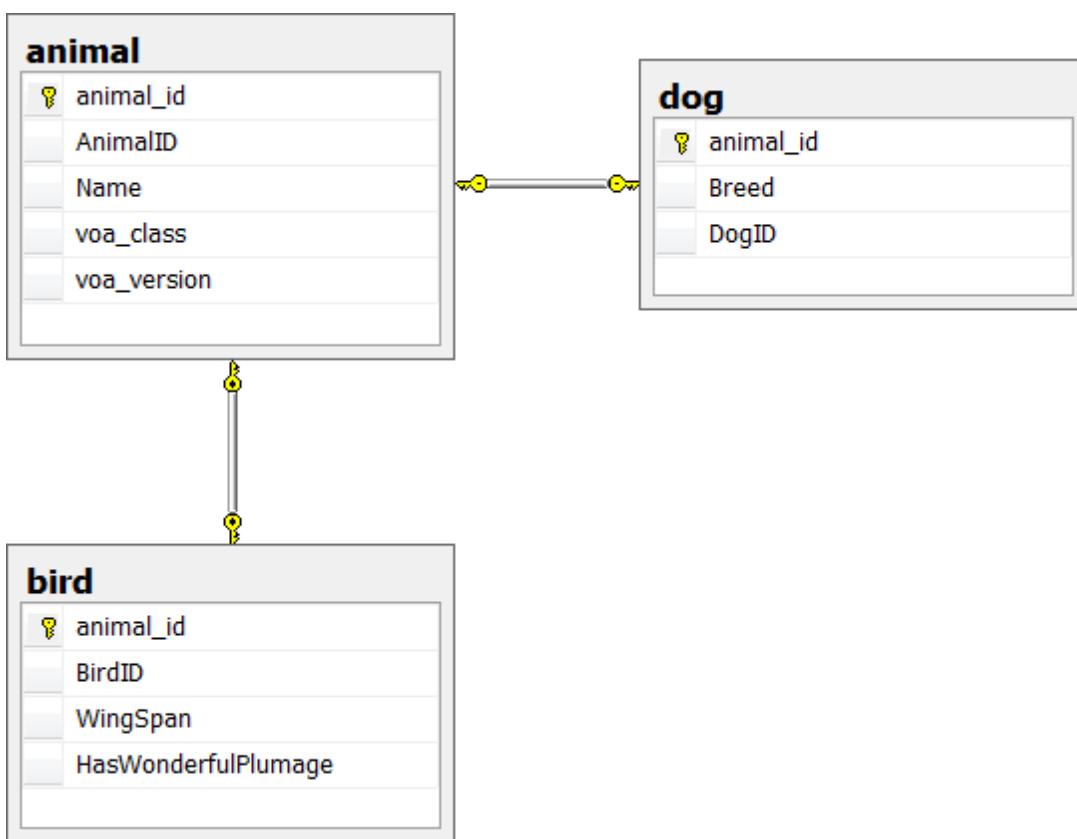


Figure 91

💡 You can balance database access and storage by tweaking mapping strategies. Vertical mapping = less storage + slower access, Flat mapping = more storage + faster access.

To see the effect of the mixed mapping described above, let's look at another code example that populates all of our objects.

```
■ Shared Sub Main(ByVal args() As String)
    Using scope As IObjectScope = ObjectScopeProvider1.GetNewObjectScope()
        scope.Transaction.Begin()

        Dim animal As New Animal()
        animal.Name = "Bob"
        scope.Add(animal)

        Dim dog As New Dog()
        dog.Name = "Fluffy"
        dog.Breed = "Mutt"
        scope.Add(dog)

        Dim bird As New Bird()
        bird.Name = "Tweety"
        bird.WingSpan = 10
        scope.Add(bird)

        Dim parrot As New Parrot()
        parrot.Name = "Polly"
        parrot.WingSpan = 12
        parrot.HasWonderfulPlumage = True
        scope.Add(parrot)

        scope.Transaction.Commit()
        Console.ReadKey()
    End Using
End Sub
```

```

static void Main(string[] args)
{
    using (IObjectScope scope = ObjectScopeProvider1.GetNewObjectScope())
    {
        scope.Transaction.Begin();

        Animal animal = new Animal();
        animal.Name = "Bob";
        scope.Add(animal);

        Dog dog = new Dog();
        dog.Name = "Fluffy";
        dog.Breed = "Mutt";
        scope.Add(dog);

        Bird bird = new Bird();
        bird.Name = "Tweety";
        bird.WingSpan = 10;
        scope.Add(bird);

        Parrot parrot = new Parrot();
        parrot.Name = "Polly";
        parrot.WingSpan = 12;
        parrot.HasWonderfulPlumage = true;
        scope.Add(parrot);

        scope.Transaction.Commit();
        Console.ReadKey();
    }
}

```

In both code examples shown so far, only the mapping strategy changes in the OpenAccess configuration. The code for working with objects and storing them is exactly the same in all cases.

The series of screenshots that follow show how the data is distributed. The Animal table again has a column for the base class Name property and a voa_class column to discriminate which class this record belongs to.

AnimalID	Name	voa_class	voa_version
1	Bob	111	1
2	Fluffy	222	1
3	Polly	444	1
4	Tweety	333	1

Figure 92

Once again the Dog table has a AnimalID column that refers back to the Animal table and a Breed column.

AnimalID	Breed
2	Mutt

Figure 93

The Parrot table mapping strategy is "Flat" so both Bird and Parrot objects are stored in the Bird table. We don't lose much space here, only the HasWonderfulPlumage boolean column is wasted when storing a Bird.

AnimalID	BirdID	WingSpan	HasWonderfulPlumage
3	0	12	1
4	0	10	NULL

Figure 94

6.5 Horizontal Mapping

Using a Horizontal mapping strategy you use an abstract base class as a template for your descendant class. The base class is marked with the **Abstract** keyword and only descendant classes are actually persisted. To adapt our Animal example to a Horizontal mapping strategy, the Animal class is marked as **MustInherit** (VB.NET) or **abstract** (C#).

```
<Telerik.OpenAccess.Persistent()> _
Public MustInherit Class Animal
'...
End Class
```

```
[Telerik.OpenAccess.Persistent()]
public abstract class Animal
{
//...
}
```

In the code example we remove the Animal class references. Otherwise the code remains the same.

```
■ Shared Sub Main(ByVal args() As String)
    Using scope As IObjectScope = ObjectScopeProvider1.GetNewObjectScope()
        scope.Transaction.Begin()

        'Animal animal = new Animal();
        'animal.Name = 'Bob';
        'scope.Add(animal);

        Dim dog As New Dog()
        dog.Name = "Fluffy"
        dog.Breed = "Mutt"
        scope.Add(dog)

        Dim bird As New Bird()
        bird.Name = "Tweety"
        bird.WingSpan = 10
        scope.Add(bird)

        Dim parrot As New Parrot()
        parrot.Name = "Polly"
        parrot.WingSpan = 12
        parrot.HasWonderfulPlumage = True
        scope.Add(parrot)

        scope.Transaction.Commit()
        Console.ReadKey()
    End Using
End Sub
```

```
■ static void Main(string[] args)
{
    using (IObjectScope scope = ObjectScopeProvider1.GetNewObjectScope())
    {
        scope.Transaction.Begin();

        //Animal animal = new Animal();
        //animal.Name = "Bob";
        //scope.Add(animal);

        Dog dog = new Dog();
        dog.Name = "Fluffy";
        dog.Breed = "Mutt";
        scope.Add(dog);

        Bird bird = new Bird();
        bird.Name = "Tweety";
        bird.WingSpan = 10;
        scope.Add(bird);

        Parrot parrot = new Parrot();
        parrot.Name = "Polly";
        parrot.WingSpan = 12;
        parrot.HasWonderfulPlumage = true;
        scope.Add(parrot);

        scope.Transaction.Commit();
        Console.ReadKey();
    }
}
```

The resulting database looks like the diagram below. The AnimalID and Name fields from the Animal class are "copied" into the Bird and Dog tables. The AnimalID from the base class is reused as the primary key for tables. The AnimalID field was marked with "protected" scope so that it would be accessible for assignment in the ancestor class.

- You could also have used another field, e.g. BirdID as the primary key and removed the AnimalID from the base class. There are a number of options for setting the primary key that will be discussed when we look at how inheritance is configured.

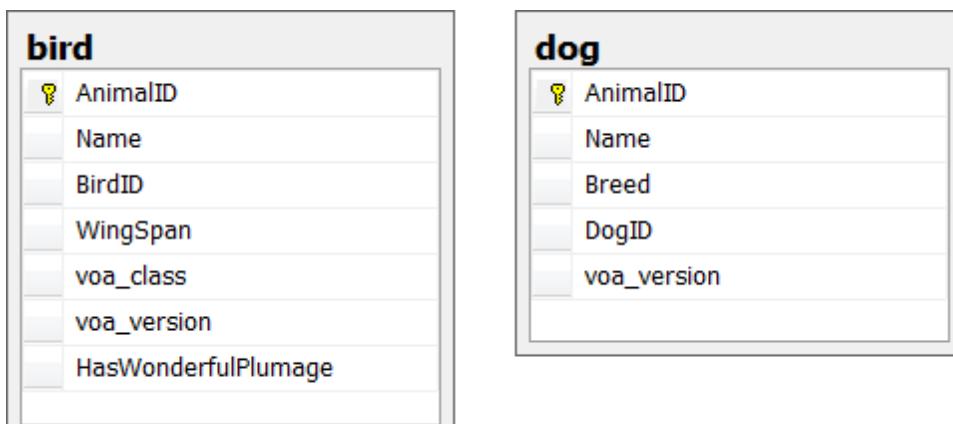


Figure 95

The Dog table has the AnimalID and Name columns from the Animal table.

AnimalID	Name	Breed	DogID	voa_version
1	Fluffy	Mutt	0	1

Figure 96

The Bird table also has AnimalID and Name columns.

AnimalID	Name	BirdID	WingSpan	voa_class	voa_version	HasWonderfulPlumage
1	Polly	0	12	444	1	1
2	Tweety	0	10	333	1	NULL

Figure 97

Note that in this particular example we left the Parrot table mapped as "Flat", so the Bird table will include columns for "voa_class" and "HasWonderfulPlumage".

6.6 Configuration

So how do we get all this magic to happen? Mapping strategies are configured using the Forward Mapping wizard. After the classes are marked as persistent in the wizard, and the Persistent node of the tree is selected, notice that Bird, Dog and Parrot classes are shaded in gray to indicate they are inherited.

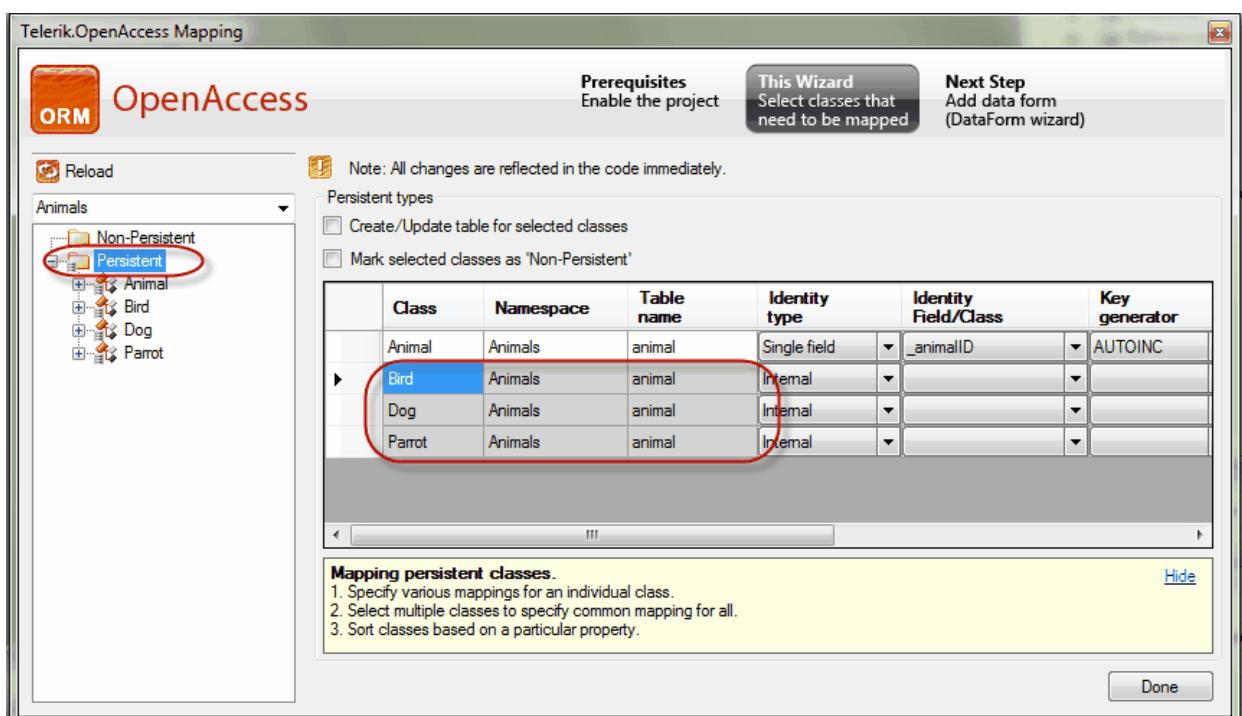


Figure 98

Mapping Strategy

Each persistent class has inheritance settings that include a Strategy drop down list. This list will include differing options depending on the position of the class within the hierarchy. The base class will include "<default>" and "horizontal". Descendant class will have "flat" and "vertical" in the list.

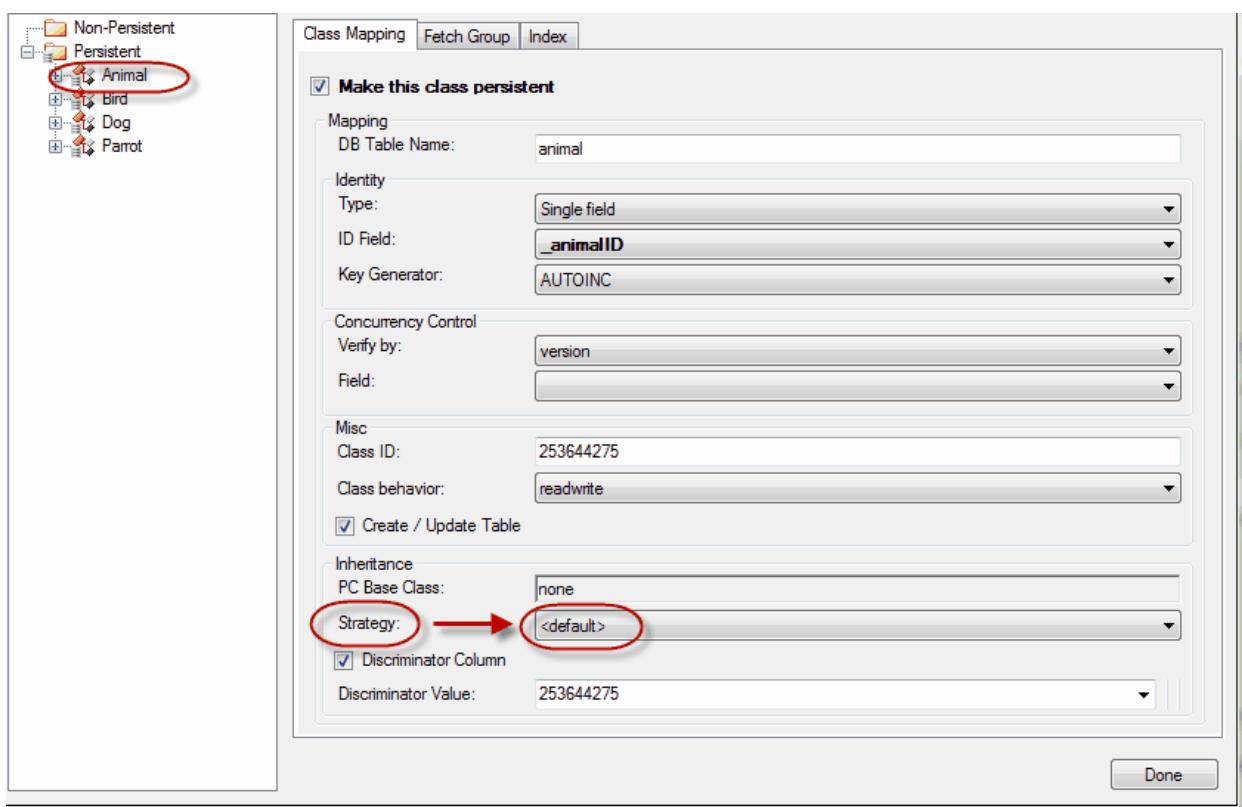


Figure 99

To configure mapping strategy for example "Animal" classes:

- **Flat:** Set the Animal base class Strategy to "<default>" and the descendant Bird, Dog and Parrot classes to "flat".
- **Vertical:** Set the Animal base class Strategy to "<default>" and the descendant Bird, Dog and Parrot classes to "vertical".
- **Mixed:** Set the Animal base class Strategy to "<default>" and the descendant Bird, Dog and Parrot classes to "flat" or "vertical". For example, set the Bird and Dog classes to "vertical" and the Parrot class to "flat".
- **Horizontal:** Set the Animal base class Strategy to "horizontal". Once the ancestor object has the "horizontal" strategy configured, the strategy for immediate ancestor objects Dog and Bird cannot be altered.

Identity

Persistent classes can be configured to use one of several identity types. The default "Internal" handles all the plumbing for you and stores the identity in an automatically created and maintained field in the database table. You can also choose "Single Field" as the identity field. This is the technique we have been using in the inheritance "Animals" example up to now. When you choose the "Single Field" you need to assign the "ID Field" from one of the fields in your object. A "Multiple Field" type lets you adapt to legacy systems when you need to create a composite key.

Depending on the type of field you're storing the identity in, the "Key Generator" provides an identity value. If you're already using auto-incrementing identity fields where the database generates new identity values,

then the "AUTOINC" choice should work well for you.

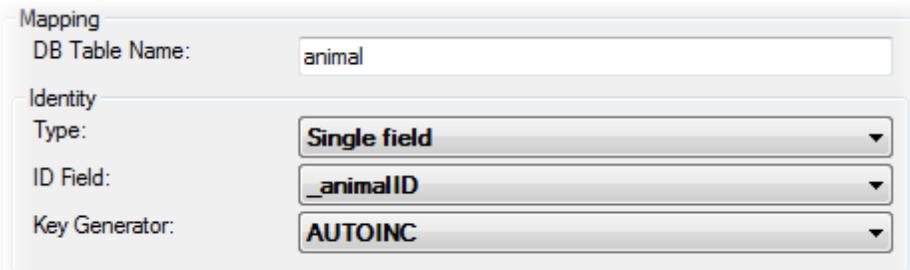


Figure 100

Other Key Generator possibilities:

- The **HIGHLOW** key Generator employs a "last-used" table to generate new identity values. This table is automatically created and maintained.
- "**None**" means that the application will have to provide a key.
- "**Verify**" checks that a key field has been specified and contains a non-default value.

In some cases the HIGHLOW key generator may lock fewer tables than AUTOINC.

6.7 Mapping Walkthrough

This walk through demonstrates mapping the Animal class hierarchy to the database using the flat mapping strategy.

Configuring the Data Model Project for Flat Mapping

- 1) Use the solution at \Projects\<CS|VB>\8A_Inheritance\8_Inheritance.sln as a starting point.
The solution contains two projects Animals and ConsoleTest. Neither project has been ORM enabled or has references added to it. The Animals project has Animal, Dog, Bird and Parrot classes already added to it.
- 2) ORM-enable the project. Specify the following:
 - a) The Persistent classes option should be enabled.
 - b) The Data Access Code option should be disabled.
 - c) The Database Connection ID should be "AnimalsConnection"
 - d) The Database should be "Animals".
- 3) From the Visual Studio menu select **Telerik > Open Access > Forward Mapping**. Note, but sure to select the Animals project in the Solution Explorer before selecting the menu option.
- 4) In the Telerik.OpenAccess Mapping dialog, select the Make Persistent checkboxes for all four classes.

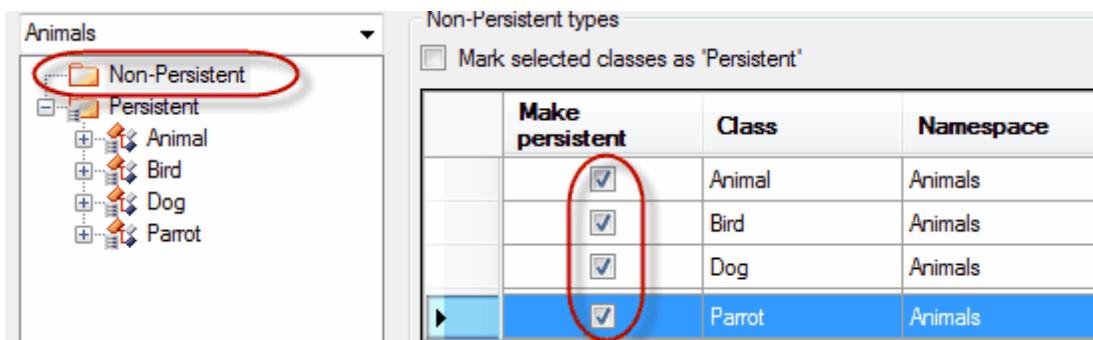


Figure 101

- 5) In the tree view on the left locate the Animal class, select it and configure the Class Mappings tab (on the right of the dialog).
- The Identity group settings should be Type = "Single Field", ID Field = "_animalID", Key Generator = "AUTOINC".

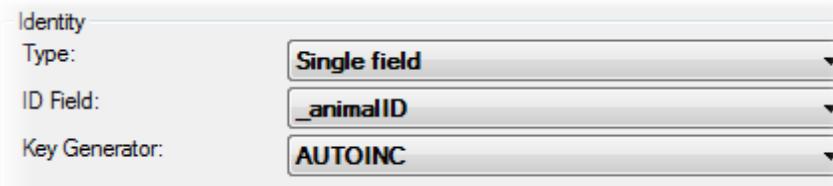
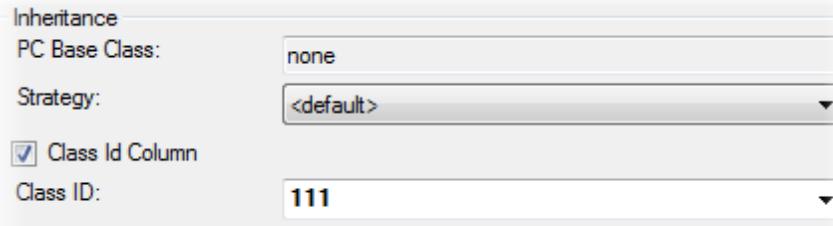


Figure 102

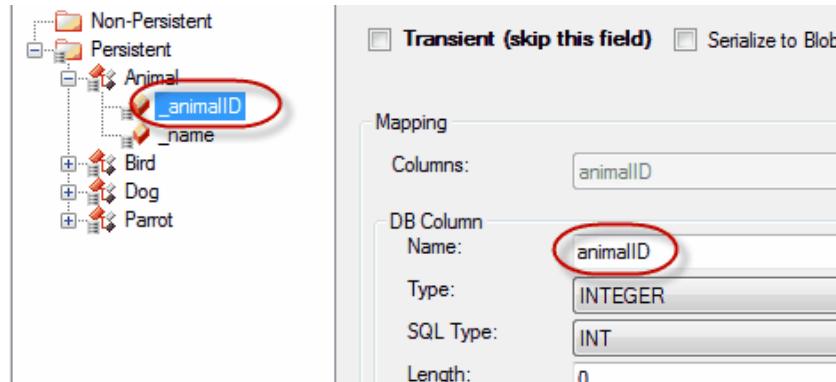
- In the Inheritance group of fields the Strategy should be "<default>" and Class ID = "111".

The Class ID is usually auto generated by OpenAccess but we're changing it here just so we can easily see where this number comes from.



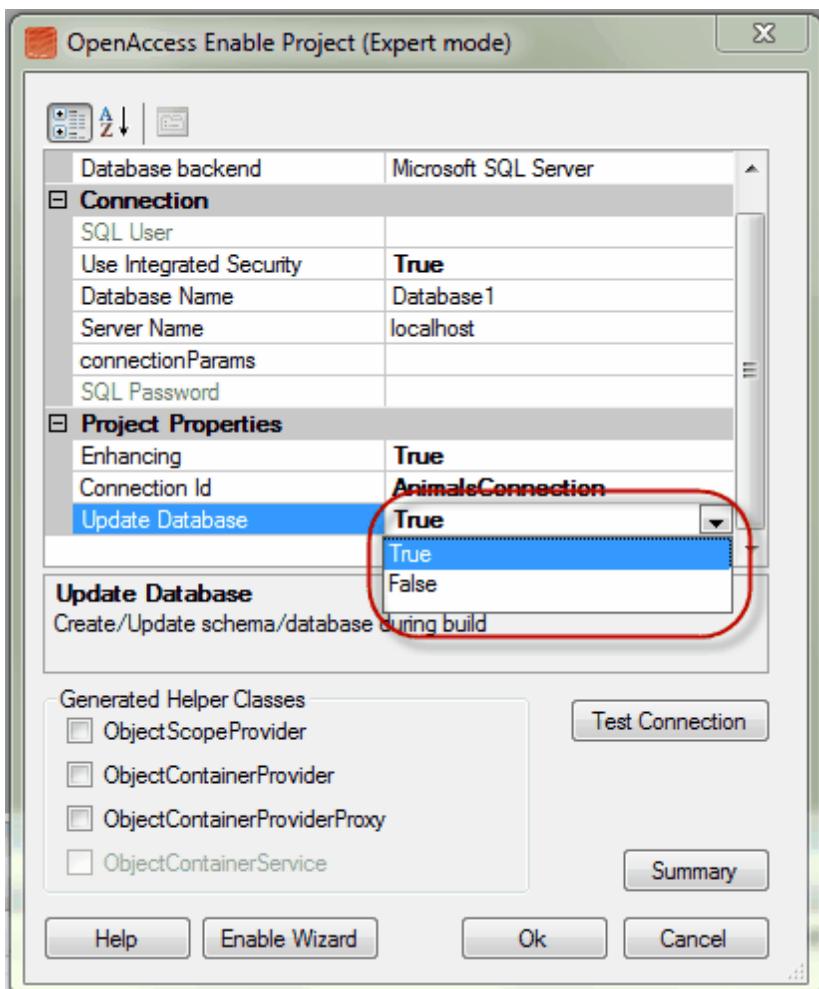
- 6) Locate the Animal node in the tree view, open the node and configure the Animal class fields.

- Select the "_animalID" field. In the DB Column section on the right, change the name to "AnimalID".

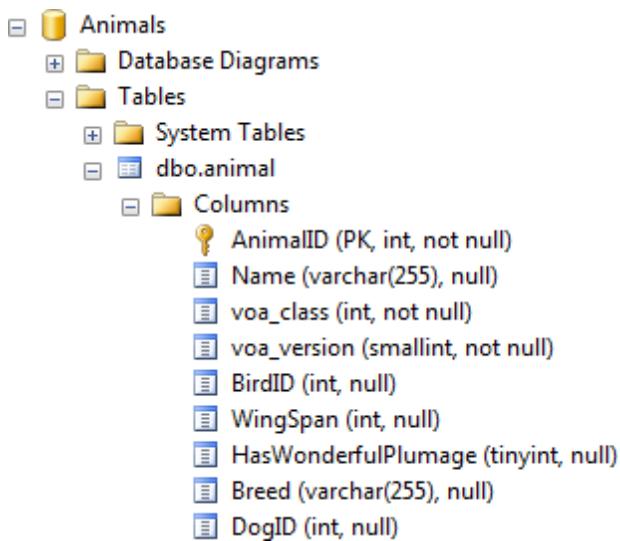


This will name the corresponding column in the "Animal" database table "AnimalID". This shows that you have complete control over the naming conventions in your database.

- b) Select the "_name" field. In the DB Column section, change the name to "Name"
- 7) Select and map the Bird class:
 - a) Set the Mapping strategy to "flat".
 - b) Set the Class ID to "222".
 - c) Set the "_birdID" column name to "BirdID"
 - d) Set the "_wingSpan" column name to "WingSpan".
- 8) Select and map the Dog class:
 - a) Set the Mapping strategy to "flat".
 - b) Set the Class ID to "333".
 - c) Set the "_dogD" column name to "DogID"
 - d) Set the "_breed" column name to "Breed".
- 9) Select and map the Parrot class:
 - a) Set the Mapping strategy to "flat".
 - b) Set the Class ID to "444".
 - c) Set the "_hasWonderfulPlumage" column name to "HasWonderfulPlumage".
- 10) Click the Done button to close the dialog.
- 11) In the Visual Studio menu, select the **Telerik > Open Access > Configuration > Connection Settings** option. Locate the Project Properties group of settings and change the Update Database setting to True.



- 12) Build the Animals project. This will also create the database.
- 13) Using Microsoft SQL Server Management Studio or the Server Explorer that comes with Visual Studio 2008, open the new "Animals" database and review the new "Animal" table there and its columns. Because you selected "_animalID" as the identity field, and provided a new column name of "AnimalID", "AnimalID" will be the primary key in the table. Once again, all the fields for all descendant classes have been flattened out to this one base class table. The "voa_class" column will store the class ID's that tell us what kind of class is being stored there. For example we should expect to see a "111" in this column if we persist an "Animal" class.



Persisting Objects

- 1) Moving on to the "ConsoleTest" project, ORM-enable the project using the following settings:
 - a) The Persistent classes option should be disabled.
 - b) The Data Access Code option should be enabled.
 - c) The Database Connection ID should be "AnimalsConnection"
 - d) The Database should be "Animals".
- 2) Add a reference to the "Animals" assembly.
- 3) Open "Program.cs" for editing.
- 4) Add references to **Telerik.OpenAccess** and **Animals** namespace in the "Imports" (VB) or "using" (C#) section of code.
- 5) Select the **Telerik > Open Access > Configuration > Update Config References** menu to update the configuration to match the OpenAccess information coming from the "Animals" assembly.
- 6) Add the code below to the Main() method.

```
■ Shared Sub Main(ByVal args() As String)
    Using scope As IObjectScope =
        ObjectScopeProvider1.GetNewObjectScope()
        scope.Transaction.Begin()

        Dim animal As New Animal()
        animal.Name = "Bob"
        scope.Add(animal)

        Dim dog As New Dog()
        dog.Name = "Fluffy"
        dog.Breed = "Mutt"
        scope.Add(dog)

        Dim bird As New Bird()
        bird.Name = "Tweety"
        bird.WingSpan = 10
        scope.Add(bird)

        Dim parrot As New Parrot()
        parrot.Name = "Polly"
        parrot.WingSpan = 12
        parrot.HasWonderfulPlumage = True
        scope.Add(parrot)

        scope.Transaction.Commit()
    End Using
End Sub
```

```

    static void Main(string[] args)
    {
        using (IObjectScope scope =
ObjectScopeProvider1.GetNewObjectScope())
        {
            scope.Transaction.Begin();

            Animal animal = new Animal();
            animal.Name = "Bob";
            scope.Add(animal);

            Dog dog = new Dog();
            dog.Name = "Fluffy";
            dog.Breed = "Mutt";
            scope.Add(dog);

            Bird bird = new Bird();
            bird.Name = "Tweety";
            bird.WingSpan = 10;
            scope.Add(bird);

            Parrot parrot = new Parrot();
            parrot.Name = "Polly";
            parrot.WingSpan = 12;
            parrot.HasWonderfulPlumage = true;
            scope.Add(parrot);

            scope.Transaction.Commit();
        }
    }
}

```

7) Set the ConsoleTest project as the Startup project and run it.

8) Open the Animals table in MS SQL Server Management Studio, Server Explorer or similar tool and check out the data.

AnimalID	Name	voa_class	voa_version	BirdID	WingSpan	HasWonderfulPlumage	Breed	DogID
1	Bob	111	1	NULL	NULL	NULL	NULL	NULL
2	Fluffy	333	1	NULL	NULL	NULL	Mutt	0
3	Polly	444	1	0	12	1	NULL	NULL
4	Tweety	222	1	0	10	NULL	NULL	NULL

To extend this example...

- Try switching the mapping strategy for Dog, Bird and Parrot to Vertical.
- Set the mapping strategy for the Parrot class to Flat, while leaving Dog and Bird as Vertical.

6.8 Wrap Up

In this chapter you learned how OpenAccess translates data from objects in an inheritance chain to database tables. You learned about strategies employed by OpenAccess to map inherited classes to tables including Flat, Vertical, Horizontal and Mixed mapping. You learned about the tradeoffs involved with

implementing each kind of strategy. Finally you learned the practical steps required to configure OpenAccess to use each kind of strategy.

Part

VII

Transactions

7 Transactions

This chapter discusses how OpenAccess preserves data integrity by making transactions available in your code.

In this chapter you will learn:

- Transaction basics including a simple demonstration of a minimal transaction to show how this is achieved in code.
- Some of the helpful properties and methods of OpenAccess transaction objects.
- How to use OpenAccess transactions in multiple threads.
- How to set concurrency options to best fit your environment.



Find the source projects for this chapter at \Projects\ORM\CS\9_Transactions\9_Transactions.sln

7.1 Basics

Operations that make database changes are encapsulated within a transaction. This ensures that all changes are committed at once to the database. It also provides the ability to rollback changes if something bad happens, so the database will not be polluted. Transactions are managed by the ObjectScope. Transaction property and its Begin(), Commit() and Rollback() methods. Note that only one transaction can be active at the same time for each object scope instance.

Data is maintained in a consistent state by OpenAccess in concert with **ACID** principles

- **A)tomicity:** Transactions are atomic, that is, they are the smallest, indivisible unit of processing and are guaranteed to complete successfully in their entirety or not at all.
- **C)onsistency:** The database is in a consistent state before the start of a transaction and after either a rollback or commit. The database is never left in an intermediate state, even when errors occur.
- **I)solation:** Changes to objects within a transaction are isolated from the same object in other transactions. Each transaction should appear to run independently from all other transactions associated with a database.
- **D)urability:** Changes to the database must be durable, i.e. data within transactions committed to the database will not be lost.

OpenAccess transactions are handled in code using the IObjectScope.ITransaction property. ITransaction comes with the three methods expected of a transaction, **Begin()**, **Commit()** and **Rollback()**. For example:

```
■ Using scope As IObjectScope = ObjectScopeProvider1.GetNewObjectScope()
  scope.Transaction.Begin()
  Try

    Dim order As New Order()
    order.CustomerID = "ANTON"
    order.OrderDate = DateTime.Today
    order.EmployeeID = 1
    order.RequiredDate = DateTime.Today.AddDays(14)
    order.ShippedDate = DateTime.Today.AddDays(5)
    order.ShipVia = 1
    order.Freight = 5.53D
    order.ShipName = "Antonio Moreno Taquería"
    order.ShipAddress = "Mataderos 2312"
    order.ShipCity = "México D.F."
    order.ShipCountry = "Mexico"
    order.ShipPostalCode = "05023"

    scope.Add(order)
    scope.Transaction.Commit()
  Catch
    ' transaction may be closed if failing during commit,
    ' so check for active
    If scope.Transaction.IsActive Then
      scope.Transaction.Rollback()
    End If
  End Try
End Using
```

```


    using (IObjectScope scope = ObjectScopeProvider1.GetNewObjectScope())
    {
        scope.Transaction.Begin();
        try
        {

            Order order = new Order();
            order.CustomerID = "ANTON";
            order.OrderDate = DateTime.Today;
            order.EmployeeID = 1;
            order.RequiredDate = DateTime.Today.AddDays(14);
            order.ShippedDate = DateTime.Today.AddDays(5);
            order.ShipVia = 1;
            order.Freight = 5.53m;
            order.ShipName = "Antonio Moreno Taqueria";
            order.ShipAddress = "Mataderos 2312";
            order.ShipCity = "México D.F.";
            order.ShipCountry = "Mexico";
            order.ShipPostalCode = "05023";

            scope.Add(order);
            scope.Transaction.Commit();
        }
        catch
        {
            // transaction may be closed if failing during commit,
            // so check for active
            if (scope.Transaction.IsActive)
                scope.Transaction.Rollback();
        }
    }
}


```

7.2 ITransaction

ITransaction is packed with handy methods and properties. Here are some example code snippets showing how you might use them.

- **Objects:** Returns an IList of all objects of a given type and state.

```


    For Each mutt As Dog In scope.Transaction.Objects(Of Dog)(ObjectState.New)
        If dog.Breed.Equals("Mutt") Then
            scope.Remove(mutt)
        End If
    Next mutt


```

```


    foreach (Dog mutt in scope.Transaction.Objects<Dog>(ObjectState.New))
    {
        if (dog.Breed.Equals("Mutt"))
        {
            scope.Remove(mutt);
        }
    }


```

- **DirtyObjects:** Is an IList of all modified objects in the transaction.

```

For Each critter As Animal In scope.Transaction.DirtyObjects
  If (Not scope.IsRemoved(critter)) Then
    Console.WriteLine(critter.Name)
  End If
Next critter

```

```

foreach (Animal critter in scope.Transaction.DirtyObjects)
{
  if (!scope.IsRemoved(critter))
    Console.WriteLine(critter.Name);
}

```

- **Flush()**: Flushes all dirty and new object instances to the database and evicts all instances from the local cache. This method allows unreferenced instances to be garbage collected making it easier to write loops that update millions of instances in a single transaction.

```

For i As Integer = 0 To 9999
  Dim newBird As New Bird()
  newBird.Name = "Bird" & i.ToString()
  scope.Add(newBird)
  If (i Mod 100) = 0 Then
    Console.WriteLine("Flush " & i.ToString())
    scope.Transaction.Flush()
  End If
Next i

```

```

for (int i = 0; i < 10000; i++)
{
  Bird newBird = new Bird();
  newBird.Name = "Bird" + i.ToString();
  scope.Add(newBird);
  if ((i % 100) == 0)
  {
    Console.WriteLine("Flush " + i.ToString());
    scope.Transaction.Flush();
  }
}

```

7.3 TransactionProperties

You can fine tune transaction behavior using the scope.Transaction.TransactionProperties object.

- **AutomaticBegin**: This handy property when true means that transactions are always started for you automatically. You can Commit the current transaction at any time and the next new transaction will be started for you. This property can be especially helpful when working with bound controls.

```

④ Using scope As IObjectScope = ObjectScopeProvider1.GetNewObjectScope()
    scope.TransactionProperties.AutomaticBegin = True
    Try
        Dim order As New Order()
        order.CustomerID = "ANTON"
        order.OrderDate = DateTime.Today
        scope.Add(order)
        scope.Transaction.Commit()

        Dim order2 As New Order()
        order2.CustomerID = "FRANK"
        order2.OrderDate = DateTime.Today
        scope.Add(order2)
        scope.Transaction.Commit()
    Catch
        scope.Transaction.Rollback()
    End Try
End Using

```

```

❷ using (IObjectScope scope = ObjectScopeProvider1.GetNewObjectScope())
{
    scope.TransactionProperties.AutomaticBegin = true;
    try
    {
        Order order = new Order();
        order.CustomerID = "ANTON";
        order.OrderDate = DateTime.Today;
        scope.Add(order);
        scope.Transaction.Commit();

        Order order2 = new Order();
        order2.CustomerID = "FRANK";
        order2.OrderDate = DateTime.Today;
        scope.Add(order2);
        scope.Transaction.Commit();
    }
    catch
    {
        scope.Transaction.Rollback();
    }
}

```

- **RefreshReadObjectsInNewTransaction:** By default, OpenAccess refreshes persistent objects at the start of each new transaction. Set the RefreshReadObjectsInNewTransaction property to false if you want to control the refreshing behavior. You will be responsible for calling `IObjectScope.Refresh()` to re-read a given object or `IObjectScope.Evict()` to remove a given object from the cache.

```
scope.TransactionProperties.RefreshReadObjectsInNewTransaction = False
scope.Transaction.Begin()

' Freight = value in database, for example '10'
Dim orders As Query(Of Order) = _
scope.GetSqlQuery(Of Order)( _
"select * from orders where orderid = 11079", String.Empty)
Dim order As Order = orders.ExecuteList()(0)
Debug.WriteLine(order.Freight)

scope.Transaction.Commit()

' breakpoint here and change freight manually in the database,
' to '99'

scope.Transaction.Begin()

' Freight still is '10' and does not match the database
orders = scope.GetSqlQuery(Of Order)( _
"select * from orders where orderid = 11079", String.Empty)
order = orders.ExecuteList()(0)
Debug.WriteLine(order.Freight)

' call Refresh() explicitly
scope.Refresh(order)

' Freight is now '99' and matches the database
orders = scope.GetSqlQuery(Of Order)( _
"select * from orders where orderid = 11079", String.Empty)
order = orders.ExecuteList()(0)
Debug.WriteLine(order.Freight)

scope.Transaction.Commit()
```

```
scope.TransactionProperties.RefreshReadObjectsInNewTransaction =
    false;
scope.Transaction.Begin();

// Freight = value in database, for example "10"
Query<Order> orders = scope.GetSqlQuery<Order>(
    "select * from orders where orderid = 11079", String.Empty);
Order order = orders.ExecuteList()[0];
Debug.WriteLine(order.Freight);

scope.Transaction.Commit();

// breakpoint here and change freight manually in the database,
// to "99"

scope.Transaction.Begin();

// Freight still is "10" and does not match the database
orders = scope.GetSqlQuery<Order>(
    "select * from orders where orderid = 11079",
    String.Empty);
order = orders.ExecuteList()[0];
Debug.WriteLine(order.Freight);

// call Refresh() explicitly
scope.Refresh(order);

// Freight is now "99" and matches the database
orders = scope.GetSqlQuery<Order>(
    "select * from orders where orderid = 11079",
    String.Empty);
order = orders.ExecuteList()[0];
Debug.WriteLine(order.Freight);

scope.Transaction.Commit();
```

- **FailFast:** This property is true by default and determines that the entire transaction fails on the first failure in the transaction. Set this property false if you need to collect information about failing objects. To reproduce the behavior, set FailFast to false, then put a breakpoint at the Commit(), change the Freight amounts in the database directly using SQL Server Management Studio or other utility. You will also have to change the order id's involved. With FailFast = False, the NumberOfConflicts shown in the catch block show the error for each order. With FailFast = True, the loop only executes one time.

```
Using scope As IObjectScope = _
ObjectScopeProvider1.GetNewObjectScope()
Try
    ' gather all conflicts
    scope.TransactionProperties.FailFast = False
    scope.Transaction.Begin()

    Dim orders As Query(Of Order) = _
    scope.GetSqlQuery(Of Order) ( _
    "select * from orders where orderid in (11079, 11080, 11081)", _
    String.Empty)

    For Each order As Order In orders.ExecuteList()
        order.Freight = 123
    Next order

    ' place break point here and manually
    ' change the same order listed above
    scope.Transaction.Commit()

    Catch ex As OptimisticVerificationException
        ' write out each conflict
        For i As Integer = 0 To ex.NumberOfConflicts - 1
            Debug.WriteLine(ex.Conflict(i).Message)
        Next i
        If scope.Transaction.IsActive Then
            scope.Transaction.Rollback()
        End If
    End Try
End Using
```

```
❶ using (IObjectScope scope =
ObjectScopeProvider1.GetNewObjectScope())
{
    try
    {
        // gather all conflicts
        scope.TransactionProperties.FailFast = false;
        scope.Transaction.Begin();

        Query<Order> orders = scope.GetSqlQuery<Order>(
            "select * from orders where orderid in (11079, 11080, 11081)",
            String.Empty);

        foreach (Order order in orders.ExecuteList())
        {
            order.Freight = 123;
        }

        // place break point here and manually
        // change the same order listed above
        scope.Transaction.Commit();
    }
    catch (OptimisticVerificationException ex)
    {
        // write out each conflict
        for (int i = 0; i < ex.NumberOfConflicts; i++)
        {
            Debug.WriteLine(ex.Conflict(i).Message);
        }
        if (scope.Transaction.IsActive)
            scope.Transaction.Rollback();
    }
}
```

7.4 Threading

Each thread must have its own scope in a multi-threaded application. Here's an example that creates a new Thread object inside of a button click event handler. After getting an IObjectScope instance, a transaction is started and all records are queried from the Orders table. The records are processed by changing some arbitrary data (Freight and ShippedDate) and the transaction is committed.

A button click event kicks off a new thread to begin processing. Because the keyword "lock" is invoked within the thread start procedure, the procedure completes before another thread starts in.

```
■ Private Shared lockObject As Object = New Object()

Public Shared Sub ThreadProc()
    SyncLock lockObject

    Const sql As String = "select * from orders"
    Dim random As New Random()

    ' each thread must have its own object scope
    Using scope As IObjectScope =
        Database.Get("NorthwindConnection").GetObjectScope()
            scope.Transaction.Begin()

            ' do some work...
            Dim orders As Query(Of Order) = _
                scope.GetSqlQuery(Of Order)(sql, String.Empty)
            For Each order As Order In orders.ExecuteList()
                ' change some arbitrary data
                order.Freight = random.Next(1, 100)
                order.ShippedDate = DateTime.Now
            Next order

            scope.Transaction.Commit()
        End Using
    End SyncLock
End Sub

Private Sub btnStartThread_Click(_
    ByVal sender As Object, ByVal e As EventArgs)
    Dim thread As New Thread(_
        New ThreadStart(AddressOf ThreadProc))
    thread.Start()
End Sub
```

```

    private static object lockObject = new object();

    public static void ThreadProc()
    {
        lock (lockObject)
        {
            const string sql =
                "select * from orders";
            Random random = new Random();

            // each thread must have its own object scope
            using (IObjectScope scope =
                Database.Get("NorthwindConnection").GetObjectScope())
            {
                scope.Transaction.Begin();

                // do some work...
                Query<Order> orders =
                    scope.GetSqlQuery<Order>(sql, String.Empty);
                foreach (Order order in orders.ExecuteList())
                {
                    // change some arbitrary data
                    order.Freight = random.Next(1, 100);
                    order.ShippedDate = DateTime.Now;
                }

                scope.Transaction.Commit();
            }
        }
    }

    private void btnStartThread_Click(
        object sender, EventArgs e)
    {
        Thread thread =
            new Thread(new ThreadStart(ThreadProc));
        thread.Start();
    }
}

```

In this second sample we want to add some requirements. The threads should process nearly simultaneously, interleaving the processing. And second, we should see some feedback to show what's happening during processing.

To get the feedback we declare a delegate `ListBoxUpdateHandler` that takes a message. We implement a method `UpdateListBox` with the same signature as the delegate and within the method add a message to a `RadListBox` on the form.

The thread procedure in this example now takes a single object as a parameter. Instead of selecting all records, this time we select only 5 orders and use a "like" statement against the argument passed to the method. This allows us to process different parts of the table. Inside the order updating loop we call the `Invoke()` method of the list box, passing the `UpdateListBox()` method to call and a status message. Note that `Invoke()` lets us call a method that executes in the control's thread context. If two threads try to update the same record, an `OptimisticVerificationException` is thrown. The exception is handled simply by displaying the error message in the list box.

The button click event handler starts three threads with different filter strings.

```

Private Delegate Sub ListBoxUpdateHandler(ByVal message As String)
Private Sub UpdateListBox(ByVal message As String)
    lbStatus.Items.Insert(0, New RadListBoxItem(message))
End Sub

Public Sub ThreadProc(ByVal argument As Object)
    Dim sql As String =
        "select top 5 * from orders where customerid like '%" & _
        argument.ToString() & "'"
    Dim random As New Random()

    ' each thread must have its own object scope
    Using scope As IObjectScope =
        Database.Get("NorthwindConnection").GetObjectContext()
        scope.Transaction.Begin()

    ' do some work...
    Dim orders As Query(Of Order) =
        scope.GetSqlQuery(Of Order)(sql, String.Empty)

    For Each order As Order In orders.ExecuteList()
        Dim message As String = "Thread: " & _
            Thread.CurrentThread.ManagedThreadId.ToString() & " Order: " & _
            order.OrderID.ToString()

        lbStatus.Invoke(New ListBoxUpdateHandler(AddressOf UpdateListBox), _
            New Object() { message })

        ' change some arbitrary data
        order.Freight = random.Next(1, 100)
        order.ShippedDate = DateTime.Now
    Next order
    Try
        scope.Transaction.Commit()
    Catch ex As OptimisticVerificationException
        lbStatus.Invoke(New ListBoxUpdateHandler(AddressOf UpdateListBox), _
            New Object() { ex.Message })

        If scope.Transaction.IsActive Then
            scope.Transaction.Rollback()
        End If
    End Try
    End Using
End Sub

Private Sub btnStartThread_Click(ByVal sender As Object, ByVal e As EventArgs)
    Dim thread As New Thread(New ParameterizedThreadStart(AddressOf ThreadProc))
    thread.Start("A")
    Dim thread2 As New Thread(New ParameterizedThreadStart(AddressOf ThreadProc))
    thread2.Start("B")
    Dim thread3 As New Thread(New ParameterizedThreadStart(AddressOf ThreadProc))
    thread3.Start("C")

```

```
End Sub
```

```
private delegate void ListBoxUpdateHandler(string message);

private void UpdateListBox(string message)
{
    lbStatus.Items.Insert(0, new RadListBoxItem(message));
}

public void ThreadProc(object argument)
{
    string sql =
        "select top 5 * from orders where customerid like '%" +
        argument.ToString() +
        "%'";
    Random random = new Random();

    // each thread must have its own object scope
    using (IObjectScope scope =
        Database.Get("NorthwindConnection").GetObjectScope())
    {
        scope.Transaction.Begin();

        // do some work...
        Query<Order> orders =
            scope.GetSqlQuery<Order>(sql, String.Empty);

        foreach (Order order in orders.ExecuteList())
        {
            string message = "Thread: " +
                Thread.CurrentThread.ManagedThreadId.ToString() +
                " Order: " +
                order.OrderID.ToString();

            lbStatus.Invoke(new ListBoxUpdateHandler(UpdateListBox),
                new object[] { message });

            // change some arbitrary data
            order.Freight = random.Next(1, 100);
            order.ShippedDate = DateTime.Now;
        }
        try
        {
            scope.Transaction.Commit();
        }
        catch (OptimisticVerificationException ex)
        {
            lbStatus.Invoke(new ListBoxUpdateHandler(UpdateListBox),
                new object[] { ex.Message });

            if (scope.Transaction.IsActive)
                scope.Transaction.Rollback();
        }
    }
}
```

```

}

private void btnStartThread_Click(object sender, EventArgs e)
{
    Thread thread = new Thread(new ParameterizedThreadStart(ThreadProc));
    thread.Start("A");
    Thread thread2 = new Thread(new ParameterizedThreadStart(ThreadProc));
    thread2.Start("B");
    Thread thread3 = new Thread(new ParameterizedThreadStart(ThreadProc));
    thread3.Start("C");

}

```

When the example is run and the button is clicked several times, the threads begin to interleave, showing that processing is concurrent.

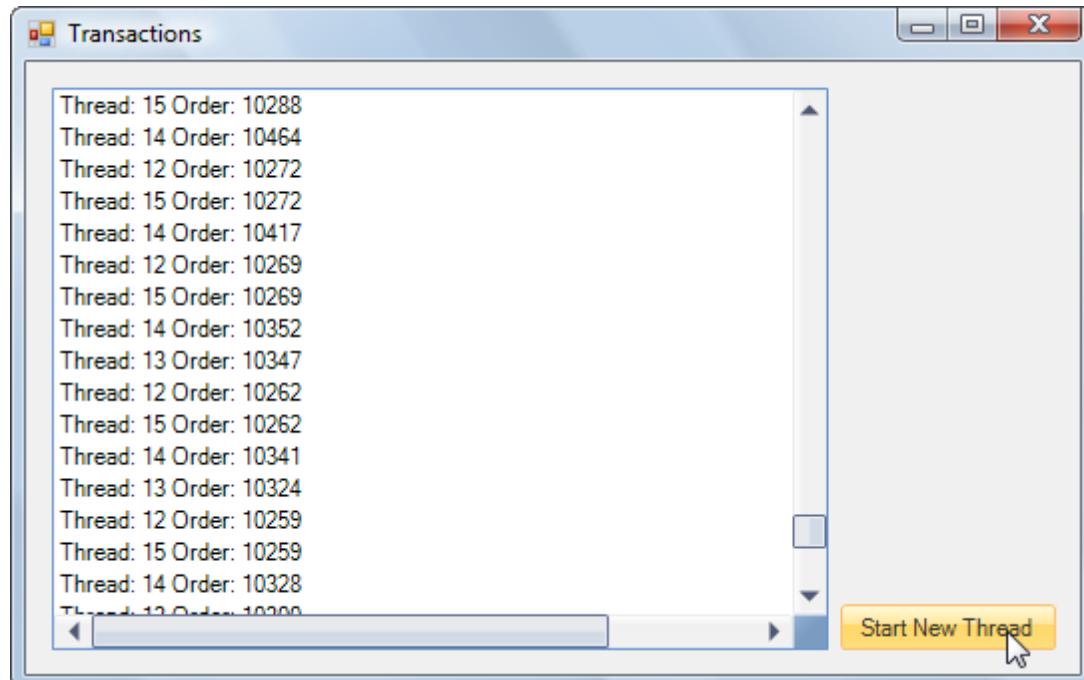


Figure 103

If you click the button enough times, eventually a clash will occur where the same record is updated from more than one thread at one time.

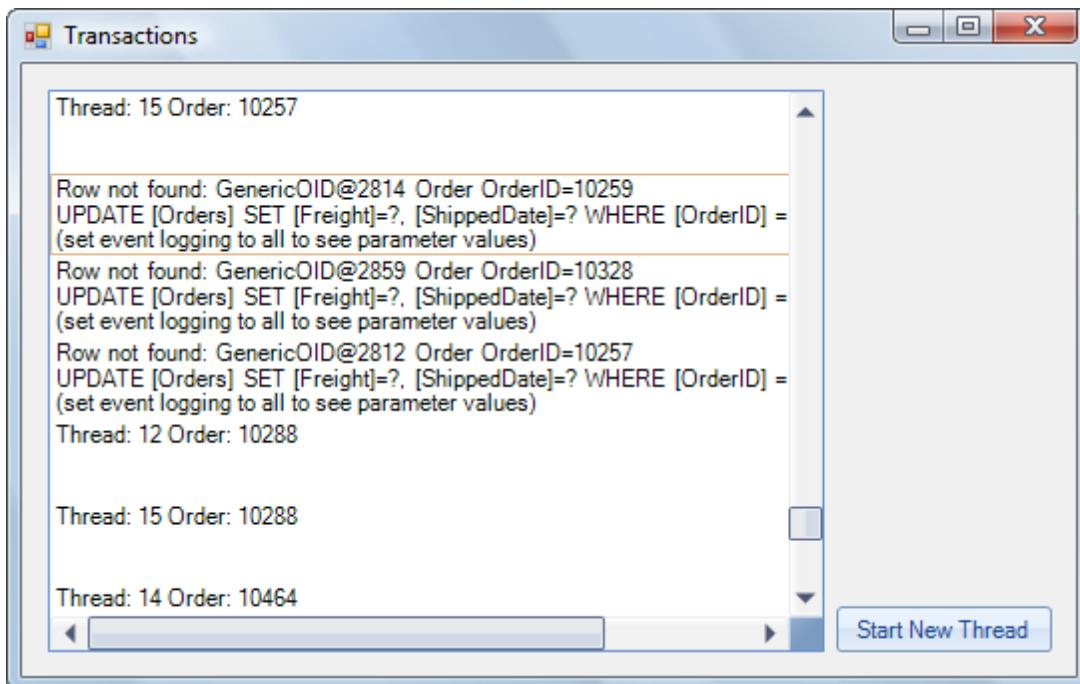


Figure 104

7.5 Concurrency

Before talking about how OpenAccess controls concurrency behavior, lets look at how things can go wrong. We saw in the threading example how concurrent access to the same object caused an exception. Much more subtle errors can occur when multiple transaction read and write to the same objects. Depending on the type of operations and the order they are executed, various kinds of concurrency anomalies can occur.

- **Dirty Read** occurs when a transaction changes a value and a second transactions reads the value before the first transaction is committed or rolled back. If the first transaction rolls back the transaction, then the second transaction contains an unintended value.
- **Lost Update** occurs when two transactions read the same record then modify the object independently. The transaction that is committed last overwrites the earlier transaction values.
- **Non-Repeatable Read** occurs when a transaction reads an object, a second transaction writes to the object and the first transaction reads a second time. Because the second read receives a different set of values the read is said to be "non-repeatable". This issue can also cause inconsistent states between objects that should have some relationship based on reading both objects at one time.
- **Phantom Read** occurs when a transaction queries a range of data, a second transaction changes this range of data (either adding or deleting), and the first transaction reads the same range of data a second time. The second read will produce data different from the first read.

Isolation

The isolation property of the ACID principle demands that every transaction should appear as though it is running all by itself on the database. Complete isolation is not practical, so Isolation Levels define the degree of relaxation of this principle:

- **Read Committed** isolation does not prevent modification of by other transactions.

- **No Lost Updates** isolation is introduced by OpenAccess as a level of isolation between Read Committed and Repeatable Read. Repeatable reads and phantom reads are allowed, but lost updates are not allowed.
- **Repeatable Read** isolation prevents modification to any retrieved data, but does not prevent modifications to *ranges* of data.

The effects of these isolation levels on the concurrency problems listed earlier are shown in the table below.

Isolation Level	Dirty read	Lost updates	Non-Repeatable Read	Phantom Read
Read Committed	not possible	possible	possible	possible
No Lost Updates	not possible	not possible	possible	possible
Repeatable Read	not possible	not possible	not possible	possible

Isolation is handled in OpenAccess by

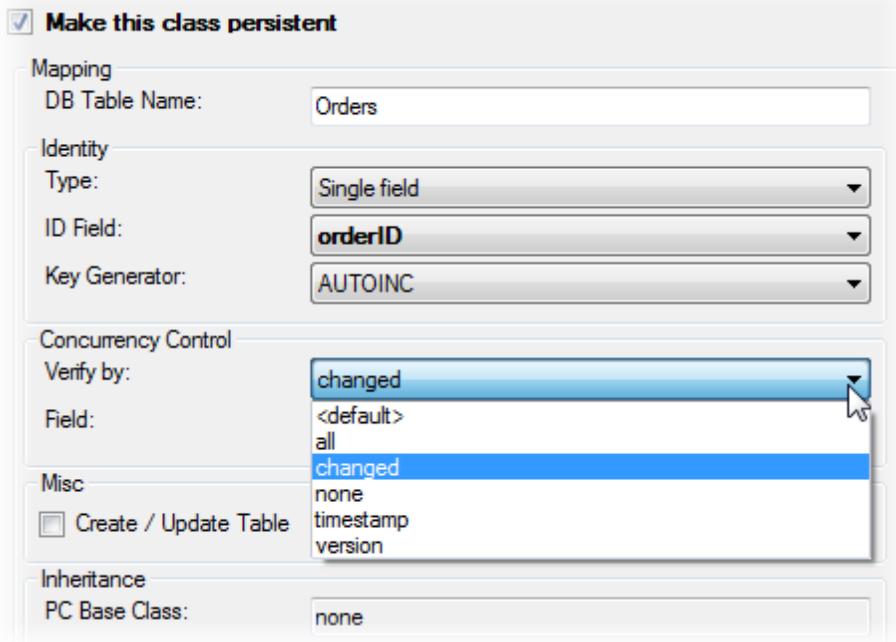
- Configuring the backend database.
- Setting the transaction properties.
- Setting mapping properties that determine how objects are versioned.

Isolation levels for the database can be defined in the "backendconfiguration" section of the application configuration file and can be one of the following: READ_COMMITTED, READ_UNCOMMITTED, REPEATABLE_READ and SERIALIZABLE. The following is an abbreviated version of the configuration file.

```
<openaccess>
  <backendconfigurations>
    <backendconfiguration>
      <isolationLevel>REPEATABLE_READ</isolationLevel>
    </backendconfiguration>
  </backendconfigurations>
</openaccess>
```

The **Concurrency** transaction property discussed previous determine if the locking scheme will be optimistic, pessimistic where write locks are obtained in code or pessimistic where write locks are obtained automatically from the database.

The last part of the equation is how the mapping properties determine how objects have changed. The Forward Mapping Wizard Concurrency Control section lets you select how changes are detected.



7.6 Wrap Up

In this chapter you learned how transactions are handled using OpenAccess scope objects. You learned transaction basics along with a simple demonstration of a minimal transaction achieved in code. You explored the helpful properties and methods of `ITransaction` and `TransactionProperties`. You learned how to use transactions in a multi-threaded scenario. Finally, you learned how concurrency is handled by OpenAccess and how to configure concurrency options to best fit your environment.

Part

VIII

Database Access

8 Database Access

This chapter explores how OpenAccess can access the database using LINQ, Object Query Language (OQL) and native SQL. The chapter then explains how to hook up stored procedures using the OpenAccess wizards.

In this chapter you will learn:

- The options available for accessing data and the tradeoffs involved for each choice.
- How to retrieve and manipulate source data using LINQ along with the scope Extent<T>() method.
- How to perform "joins" using LINQ.
- How to perform string operations with LINQ.
- How to use the Object Query Language (OQL) to perform object-oriented queries.
- How to use parameters in OQL queries.
- How to use the OQL Query Browser to audition OQL statements.
- How to use OQL to perform CRUD operations.
- How to run native SQL statements directly against the database backend.
- How to use Forward and Reverse mapping to configure OpenAccess to use stored procedures.

8.1 Using SQL with OpenAccess

There are three basic flavors of SQL queries you can run using OpenAccess: LINQ, OpenAccess Object Query Language (OQL) and native SQL. The short story is that LINQ is the most powerful option and the preferred choice, OQL comes in second because it is still an object oriented language and native SQL is the most limited option, but still necessary in some cases.



Find the source projects for this chapter
at `\Projects\ORM\CS\10_DatabaseAccess\10_DirectDatabaseAccess.sln`

8.1.1 LINQ

LINQ (Language Integrated Query) is a programming model that lets you perform operations using a SQL-like syntax against all kinds of data. The problem with writing this one-liner description of LINQ is that LINQ is such a meaty subject, the definition is inevitably too narrow. For our purposes in OpenAccess we will use LINQ to query persistent objects. For more background information on LINQ:

- Essential LINQ (Calvert and Kulkarni, ISBN 978-0-321-56416-0, Addison Wesley)
- OpenAccess Programmers Guide topics "Introduction to LINQ" and "LINQ Building Blocks".
- For a handy cheat sheet of LINQ against OpenAccess data examples see the OpenAccess Programmers Guide topic "Using LINQ with OpenAccess ORM".

- For another handy cheat sheet of LINQ examples that are not OpenAccess specific see the "LINQ 101 Samples" at <http://msdn.microsoft.com/en-us/vcsharp/aa336746.aspx>.

To use LINQ you must add a **System.Linq** namespace reference to access important classes that make LINQ possible. To get integration between LINQ and OpenAccess you need to add a **Telerik. OpenAccess.Query** namespace reference as well. This namespace surfaces the core extension method that lets your data interoperate with LINQ: **IObjectScope.Extent<T>()**.

Getting Source Data with Extent<T>()

`Extent<T>()` returns an **IObjectScopeQuery**, a gateway interface to LINQ functionality.

IObjectScopeQuery descends from **IQueryable**. **IQueryable** is a LINQ interface that has an **Expression**, a type that the expression returns and a provider that knows how to execute the query. **IObjectScopeQuery** adds properties and methods specific to OpenAccess. In particular you may want to use the **BackendQuery** property to see the actual SQL statement being run against the database server back end (you can actually copy this string and run it in a SQL utility). Also the **ToList()** method executes the query and returns a typed list. **IObjectScopeQuery** also ultimately inherits from **IEnumerable**, so we can return all the objects for a given collection in the database and iterate them:

```
Dim result = scope.Extent(Of Employee)()
For Each employee As Employee In result
    Console.WriteLine(employee.LastName)
Next employee
```

```
var result = scope.Extent<Employee>();
foreach (Employee employee in result)
{
    Console.WriteLine(employee.LastName);
}
```

Notice the **var** keyword in the snippet above. "var" is denotes a strongly typed variable where the type is inferred from the compiler. In this case if you were to place a breakpoint after `Extent()` returns, you would see that "result" is an **IObjectScopeQuery**. So we can follow up by using the **BackendQuery** property.

```
Console.WriteLine(result.BackendQuery)
```

```
Console.WriteLine(result.BackendQuery);
```

The output looks like this in the console window where the Employee LastNames are listed followed by the back end query.

```

Davolio
Fuller
Leverling
Peacock
Buchanan
Suyama
King
Callahan
Dodsworth
SELECT [EmployeeID] AS COL1, [Address] AS COL2, [BirthDate] AS COL3, [City] AS C
OL4, [Country] AS COL5, [ReportsTo] AS COL6, [Extension] AS COL7, [FirstName] AS
COL8, [HireDate] AS COL9, [HomePhone] AS COL10, [LastName] AS COL11, [Notes] AS
COL12, [PhotoPath] AS COL13, [PostalCode] AS COL14, [Region] AS COL15, [Title]
AS COL16, [TitleOfCourtesy] AS COL17 FROM [Employees]

```

Figure 105

Using LINQ Against Source Data

Once Extent() has been called and you have the IObjectScopeQuery you can really "go to town" with LINQ methods and expressions. Here is a series of statements that gets the Employee object that has the maximum employee id for employees hired in 1993. Where() and Max() are extension methods.

```

Dim employeeResult = scope.Extent(Of Employee)()
Dim employeeLastYear =
employeeResult.Where(Function(emp) emp.HireDate.Value.Year = 1993)
Dim max As Long = employeeLastYear.Max(Function(emp) emp.EmployeeID)
Console.WriteLine(max)
```

```

var employeeResult = scope.Extent<Employee>();
var employeeLastYear =
employeeResult.Where(emp => emp.HireDate.Value.Year == 1993);
long max = employeeLastYear.Max(emp => emp.EmployeeID);
Console.WriteLine(max);
```

We can also use LINQ expressions (as opposed to methods) for a more SQL-like programming experience.

```

Dim expressionResult = (
    From emp In (scope.Extent(Of Employee)())
    Where emp.HireDate.Value.Year = 1993
    Select emp).Max(Function(emp) emp.EmployeeID)
Console.WriteLine(expressionResult)
```

```

var expressionResult =
(from emp in (scope.Extent<Employee>())
 where emp.HireDate.Value.Year == 1993
 select emp).Max(emp => emp.EmployeeID);
Console.WriteLine(expressionResult);
```

As you might expect from a language styled after SQL you can order your results on any property. The example below sorts by LastName in descending order, then on FirstName within LastName in ascending order.

```

Dim sortedResult =
    From emp In (scope.Extent(Of Employee)())
    Order By emp.LastName Descending, emp.FirstName Ascending
    Select emp
For Each employee As Employee In sortedResult
    Console.WriteLine(employee.LastName)
Next employee

```

```

var sortedResult =
    from emp in (scope.Extent<Employee>())
    orderby emp.LastName descending, emp.FirstName ascending
    select emp;
foreach (Employee employee in sortedResult)
{
    Console.WriteLine(employee.LastName);
}

```

The results in the console window look like the figure below:

Suyama
Peacock
Leverling
King
Fuller
Dodsworth
Davolio
Callahan
Buchanan

Figure 106

Joins

You can also join the data from one class to another. The most straight forward way is to use the relationship between collections that already exist like this sample below. For this example the Product object has a relationship to Category so that simple dot notation "product.Category" becomes possible. The code below also happens to be an example of "projection" where instead of selecting the entire Product object we can "project" fields out of the object we're selecting from into a new anonymous object and we can also rename the two projected fields as "Product" and "Category". When the results are iterated, the projected fields can be accessed by their new names.

```

Dim query =
    From p In scopeExtent(Of Product)()
    Select New
        p.ProductName, Category = p.Category.CategoryName
        Product = p.ProductName, Category

For Each pc In query
    Console.WriteLine(pc.Product & " - " & pc.Category)
Next pc

```

```

    var query =
        from p in scope.Extent<Product>()
        select new
        {
            Product = p.ProductName,
            Category = p.Category.CategoryName
        };

    foreach (var pc in query)
    {
        Console.WriteLine(pc.Product + " - " + pc.Category);
    }

```

If there doesn't happen to be a built-in relationship between two objects, a LINQ "join" clause will create the relationship. The example below returns the same results as the first but doesn't require that Product have a Category property.

```

Dim query = _
    From c In scope.Extent(Of Category)() _
    Join p In scope.Extent(Of Product)() _
        On c.CategoryID Equals p.CategoryID _
    Select New With {Key .Product = p.ProductName, _
                    Key .Category = c.CategoryName}

```

```

var query =
    from c in scope.Extent<Category>()
    join p in scope.Extent<Product>()
    on c.CategoryID equals p.CategoryID
    select new { Product = p.ProductName, Category = c.CategoryName };

```

String Methods

Here are some examples of using string related methods in the "where" clause of a LINQ expression.

```

Console.WriteLine("String methods")
    ' tests for equality
    ' finds Great Lakes Food Market, ...
    ' finds The Big Cheese, ...
    ' finds The Cracker Box, ...
    ' finds Wilman Kala using wildcards, ...

Dim stringResult = _
    From cust In scope.Extent(Of Customer)()
    Where cust.CompanyName = "Island Trading" OrElse _
        cust.CompanyName.StartsWith("Great") OrElse _
        cust.CompanyName.EndsWith("Cheese") OrElse _
        cust.CompanyName.Contains("rac") OrElse _
        cust.CompanyName.Matches("*lma?") _
    Select cust
For Each customer In stringResult
    Console.WriteLine(customer.CompanyName)
Next customer

```

```

    Console.WriteLine("String methods");
    var stringResult = from cust in scope.Extent<Customer>()
        where
            // tests for equality
            cust.CompanyName == "Island Trading"
            // finds Great Lakes Food Market, ...
            || cust.CompanyName.StartsWith("Great")
            // finds The Big Cheese, ...
            || cust.CompanyName.EndsWith("Cheese")
            // finds The Cracker Box, ...
            || cust.CompanyName.Contains("rac")
            // finds Wilman Kala using wildcards, ...
            || cust.CompanyName.Matches("*lma?")

        select cust;
    foreach (var customer in stringResult)
    {
        Console.WriteLine(customer.CompanyName);
    }
}

```

8.1.2 Object Query Language (OQL)

OQL is an object oriented query language based off a standard published at www.ODBMS.org (Object Database Management Systems). OQL has the advantage of being database backend agnostic. OQL references classes and field names, not tables and column names.

The syntax contains one difference that frequently throws people. Instead of selecting from a table name, e.g. "select * from Employee", you append the word "Extent" to the table name, e.g. "select * from EmployeeExtent". "Extent" signifies all objects of a given persistence capable class. For example, "Employee" is a class name and a collection of all objects of that type is called "EmployeeExtent".

```

    Dim sql As String = "select * FROM EmployeeExtent"
    Dim result = scope.GetOqlQuery(sql).Execute()
    For Each emp As Employee In result
        Console.WriteLine(emp.Country)
    Next emp
}

```

```

    string sql = "select * FROM EmployeeExtent";
    var result = scope.GetOqlQuery(sql).Execute();
    foreach (Employee emp in result)
    {
        Console.WriteLine(emp.Country);
    }
}

```

Parameters

Unlike GetSqlQuery() you don't need to pass the parameter types. Notice that the parameters in the SQL statement are prepended with a "\$" not "?". Again the actual parameter values are passed in the

Execute() method.

```
④ Dim sql As String = _
    "select emp FROM EmployeeExtent AS emp " & _
    "WHERE emp.EmployeeID = $1 AND emp.Country= $2"
    Dim result = scope.GetOqlQuery(sql).Execute(1, "USA")
    For Each emp As Employee In result
        Console.WriteLine(emp.Country)
    Next emp
```

```
④ string sql = "select emp FROM EmployeeExtent AS emp " +
    "WHERE emp.EmployeeID = $1 AND emp.Country= $2";
var result = scope.GetOqlQuery(sql).Execute(1, "USA");
foreach (Employee emp in result)
{
    Console.WriteLine(emp.Country);
}
```

Wildcard Searches

If you need to perform wildcard queries, the wildcard needs to be part of the parameter, not the SQL statement. The SQL statement here compares where the employee LastName is LIKE whatever is passed in as the parameter. In this example we're passing a "D*" to the Execute() method where the asterisk "*" is the wildcard.

```
④ Console.WriteLine("Employees with last name starting with D")
Dim sqlWithWildCard As String =
"select emp FROM EmployeeExtent AS emp " & _
"WHERE emp.LastName LIKE $1"
Dim result3 = scope.GetOqlQuery(sqlWithWildCard).Execute("D*")
For Each emp As Employee In result3
    Console.WriteLine(emp.LastName)
Next emp
```

```
④ Console.WriteLine("Employees with last name starting with D");
string sqlWithWildCard = "select emp FROM EmployeeExtent AS emp " +
    "WHERE emp.LastName LIKE $1";
var result3 = scope.GetOqlQuery(sqlWithWildCard).Execute("D*");
foreach (Employee emp in result3)
{
    Console.WriteLine(emp.LastName);
}
```

Case Insensitive Comparisons

To make your OQL query comparison case insensitive prepend "<ci>" just prior to the part of the query where case should be ignored.

```
select * from EmployeeExtent as e where e.Country = <ci> "usa"
```

Joins

You can perform joins in OQL but the syntax is slightly different because we're joining objects, not

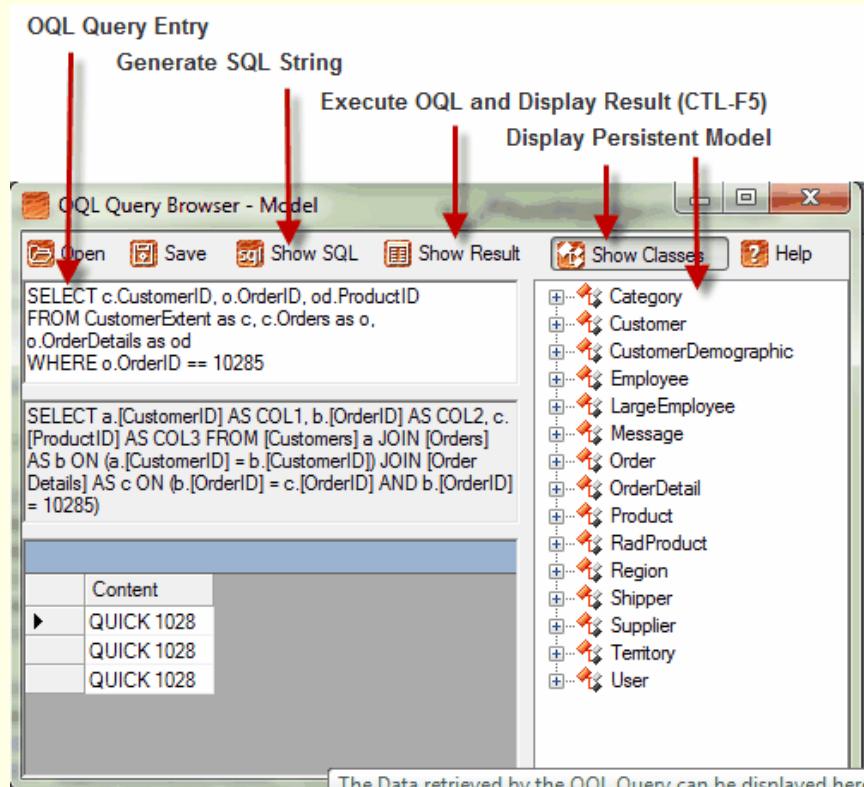
tables. Pay special attention to the 'FROM' clause of the SQL statement where the "CustomerExtent as c" gets the customer data and creates an alias "c", then "c.Orders as o" gets the orders collection and creates alias "o", then finally the "o.OrderDetails" gets the OrderDetails collection and creates alias "od". In this way the FROM clause allows us to define the sequence of joins between Customer/Order/OrderDetail.

```
④ Console.WriteLine("Employees with last name starting with D")
Dim sqlWithJoin As String =
"SELECT c.CustomerID, o.OrderID, od.ProductID " & _
"FROM CustomerExtent as c, c.Orders as o, o.OrderDetails as od " & _
"WHERE o.OrderID == 10285"
Dim result4 = scope.GetOqlQuery(sqlWithJoin).Execute()
Const format As String = "Customer: {0} Order: {1} Product: {2}"
For Each obj As Object() In result4
    Console.WriteLine(String.Format(format, obj(0), obj(1), obj(2)))
Next obj
```

```
④ Console.WriteLine("Employees with last name starting with D");
string sqlWithJoin = "SELECT c.CustomerID, o.OrderID, od.ProductID " +
    "FROM CustomerExtent as c, c.Orders as o, o.OrderDetails as od " +
    "WHERE o.OrderID == 10285";
var result4 = scope.GetOqlQuery(sqlWithJoin).Execute();
const string format = "Customer: {0} Order: {1} Product: {2}";
foreach (object[] obj in result4)
{
    Console.WriteLine(String.Format(format, obj[0], obj[1], obj[2]));
}
```

The output for the above query looks something like the console output below.

💡 To audition OQL statements, try using the OQL Query Browser via the Visual Studio Telerik menu option, **Telerik > OpenAccess > OQL Query Browser...**. Type the OQL statement into the topmost window. The equivalent SQL statement will show in the window immediately below the OQL window and the data results can be displayed in the bottommost window after hitting the "Execute OQL" button.



Why is the retrieved data a single column labeled "Content"? When you select a subset of individual fields, they no longer constitute a "Customer" object. If you enter "select * from CustomerExtent" the entire Customer object will display in the Retrieved Data window.

CRUD Operations

There is no "UPDATE" statement in OQL. Because OQL is an object-oriented language, you can only retrieve data. Once the data is retrieved you can make modifications *to the objects* and then store the objects back to the database.

```
Console.WriteLine("Update")
Const sqlForUpdate As String =
"Select * from OrderExtent Where OrderID = 10248"

scope.Transaction.Begin()

'retrieve an object using its id
Dim order As Order =
CType(scope.GetOqlQuery(sqlForUpdate).Execute()(0), Order)

'make some modifications to the object
order.Freight += 1

'store the modified object
scope.Transaction.Commit()
```

```
Console.WriteLine("Update");
const string sqlForUpdate =
"Select * from OrderExtent Where OrderID = 10248";

scope.Transaction.Begin();

//retrieve an object using its id
Order order = (Order)scope.GetOqlQuery(sqlForUpdate).Execute()[0];

//make some modifications to the object
order.Freight++;

//store the modified object
scope.Transaction.Commit();
```

8.1.3 Native SQL

This option is database backend specific. Native SQL glues the syntax and types you're using to the database. You should rarely need to use direct SQL to perform tasks that can't be handled by other options such as LINQ or OQL.

- To find the type of database being used at any one time, use the `Database.ConnectionURL` property that contains a series of semi-colon delimited name/value pairs. The "Backend" setting contains the name of the database, e.g. "Backend=mssql".

To run SQL queries against the database and return them as a series of persistent objects, call the `IObjectScope.GetSqlQuery()` method. It takes a SQL statement, the type of object you expect to return and any parameters you might have. `GetSqlQuery()` returns an `IQuery`. Call the `IQuery Execute()` method to return an `IQueryResult`. `IQueryResult` implements `IEnumerable` and `IList` so it can be iterated or indexed into.

```

Using scope As IObjectScope =
ObjectScopeProvider1.GetNewObjectScope()
Dim query As IQuery =
scope.GetSqlQuery(
"select * from categories", GetType(Category), String.Empty)
Dim result As IQueryResult = query.Execute()
For Each category As Category In result
Console.WriteLine(category.CategoryName)
Next category

Console.WriteLine(scope.Database.ConnectionURL)
Console.ReadKey()
End Using
```

```

using (IObjectScope scope =
ObjectScopeProvider1.GetNewObjectScope())
{
    IQuery query =
        scope.GetSqlQuery("select * from categories",
typeof(Category),
String.Empty);
    IQueryResult result = query.Execute();
    foreach (Category category in result)
    {
        Console.WriteLine(category.CategoryName);
    }

    Console.WriteLine(scope.Database.ConnectionURL);
    Console.ReadKey();
}
```

Parameterized Queries

Here's another example showing a parameterized query against an MS SQL database. Notice the last parameter to GetSqlQuery() is a string representation of the parameter type followed by the parameter name. Also notice that the parameter type is a valid SQL type for the database backend (MSSQL in this case). The actual parameter values are passed in an array of object fed to the IQuery Execute() method.

```

Dim query As IQuery =
scope.GetSqlQuery(
"select * from categories where CategoryID = ?", _
GetType(Category), "integer CategoryIDParameter")
Dim result As IQueryResult =
query.Execute(New Object() { 8 })
```

```

IQuery query =
scope.GetSqlQuery(
"select * from categories where CategoryID = ?",
typeof(Category),
"integer CategoryIDParameter");
IQueryResult result = query.Execute(new object[] { 8 });
```

Generic Methods

Instead of using `GetSqlQuery()` you should use the generic `GetSqlQuery<T>()` that returns a `Query<T>`. `Query<T>` has a number of methods for executing the query and returning the results as enumerable, list or binding list. For example, `ExecuteBindingList()` returns a `BindingList` that can be bound directly to UI controls. The example below is similar to the last example but uses the generic form of `GetSqlQuery()` and calls `ExecuteList()` from the `Query<T>`.

```
Dim query As Query(Of Category) = _
    scope.GetSqlQuery(Of Category)( _
        "select * from categories where CategoryID = ?", _
        "integer CategoryIDParameter")

Dim result As QueryResultList(Of Category) = _
    query.ExecuteList(New Object() { 8 })
Object() { 8 })
```

```
Query<Category> query =
    scope.GetSqlQuery<Category>(
        "select * from categories where CategoryID = ?",
        "integer CategoryIDParameter");

QueryResultList<Category> result =
    query.ExecuteList(new object[] { 8 });
```

`Query<T>` also has an `ExecuteDirect()` method that doesn't return results, but is just meant to run a SQL statement on the server.

Yet another example shows a parameterized query using a `DateTime` parameter. Notice how the result is stored in a var. Because we're only retrieving `BirthDate` and `EmployeeID` in the query, the results cannot be reconstituted as an `Employee` object. Instead the results are an array of object

```
Dim sql As String = _
    "select emp.BirthDate, emp.EmployeeID " & _
    "from Employees as emp where emp.BirthDate > ?"
Dim dateQuery As IQuery = _
    scope.GetSqlQuery(sql, Nothing, "timestamp myDate")
Dim employeeResult =
    dateQuery.Execute(DateTime.Parse("1/1/1960"))
For Each obj As Object() In employeeResult
    ' Prints the Birth Date since it was
    ' selected first in the query
    Console.WriteLine(obj(0))
    ' Prints the Employee ID
    Console.WriteLine(obj(1))
Next obj
```

```

string sql =
    "select emp.BirthDate, emp.EmployeeID " +
    "from Employees as emp where emp.BirthDate > ?";
IQuery dateQuery =
    scope.GetSqlQuery(sql, null, "timestamp myDate");
var employeeResult = dateQuery.Execute(DateTime.Parse("1/1/1960"));
foreach (object[] obj in employeeResult)
{
    // Prints the Birth Date since it was selected first in the query
    Console.WriteLine(obj[0]);
    // Prints the Employee ID
    Console.WriteLine(obj[1]);
}


```

The results look like the screenshot of the console window below:

```

8/30/1963 12:00:00 AM
3
7/2/1963 12:00:00 AM
6
5/29/1960 12:00:00 AM
7
1/27/1966 12:00:00 AM
9

```

Figure 107

8.2 Stored Procedures

Creating Stored Procedures Using Forward Mapping



Find the source projects for this chapter
at `\Projects\ORM\CS\11_StoredProcedures\11_StoredProcedures.sln`

Starting from scratch where we have a single "ContactGroup" class in a class library, here is how you can create the database table and stored procedures using forward mapping.

- 1) In Visual Studio create a new class library project called "Model".
- 2) Add a single "ContactGroup" class with a ContactGroupName property.

```

Public Class ContactGroup
    Private contactGroupName_Renamed As String
    Public Property ContactGroupName() As String
        Get
            Return contactGroupName_Renamed
        End Get
        Set(ByVal value As String)
            Me.contactGroupName_Renamed = value
        End Set
    End Property
End Class


```

```
public class ContactGroup
{
    private string contactGroupName;
    public string ContactGroupName
    {
        get { return contactGroupName; }
        set { this.contactGroupName = value; }
    }
}
```

- 3) ORM-enable the "Model" project. Specify the following: .
- 3) The Persistent classes option should be enabled.
- 4) The Data Access Code option should be disabled.
- 5) The database connection ID should be "MyStoredProcDBConnection"
- 6) The database should be "MyStoredProcDB".
- 7) In the Visual Studio Telerik menu select **OpenAccess > Configuration > Connection Settings...**
Locate the Project Properties and set the **Update Database** property to True.

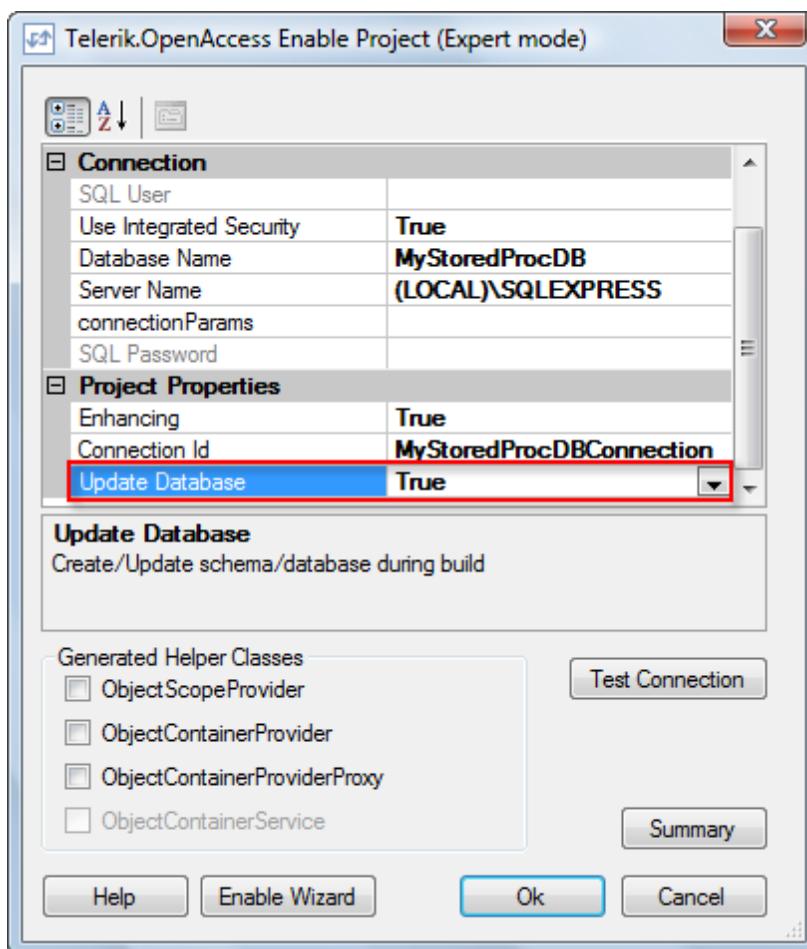


Figure 108

- 8) In the Visual Studio Telerik menu select **OpenAccess > Configuration > Backend Configuration Settings...** In the Backend Configuration dialog locate the Stored Procedures section and set the following options
- a) Use Stored Procedures for Delete Operations = True
 - b) Use Stored Procedures for Insert Operations = True
 - c) Use Stored Procedures for Update Operations = True

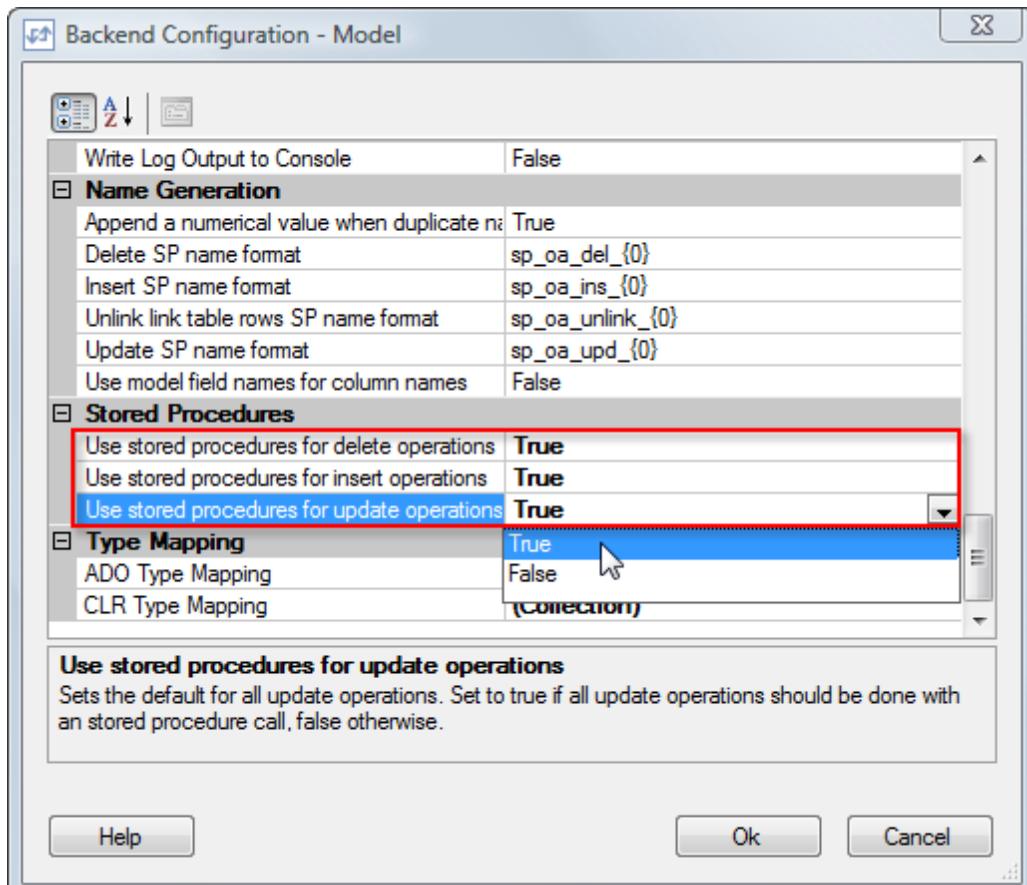


Figure 109

- 9) In the Visual Studio Telerik menu select **OpenAccess > Forward Mapping (Classes to Tables)**
 10) In the Forward Mapping Wizard, click the "Make Persistent" checkbox next to the "ContactGroup" table.

	Make persistent	Class	Namespace
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ContactGroup	Model

Figure 110

- 11) Click the **Done** button to close the Forward Mapping Wizard.
 12) Build the project. The Output window should show that the database has been updated:

```

Enhancement complete -- 0 errors, 0 warnings
Model -> E:\Courseware\OpenAccess\Instructor Materials\Projects\CS\10_StoredPr
e:\program files\telerik\openaccess orm\bin\VSchema.exe -assembly:E:\Coursewar
Create DatabaseSchema complete -- 0 errors, 0 warnings
=====
Build: 1 succeeded or up-to-date, 0 failed, 0 skipped =====
OpenAccess All: 1 succeeded, 0 failed, 0 skipped =====

```

Figure 111

13) Now if you look at your "MyStoredProcDB" database in Server Explorer or SQL Server Management Studio you will see your table created and also three stored procedures:

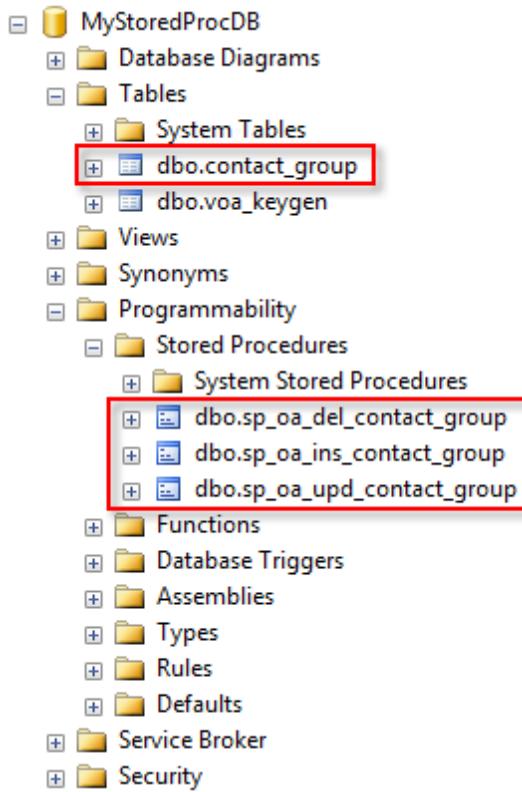


Figure 112

Where the stored procedures follow the naming convention specified by Backend Configuration options under "Name Generation":

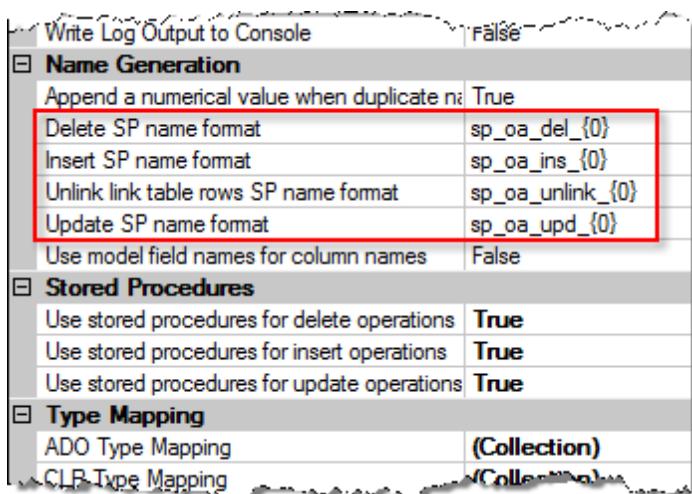


Figure 113

Here's an example of the automatically generated stored procedure for deletion:

```
-- Generating stored procedure for OpenAccess
ALTER PROCEDURE [sp_oa_del_contact_group]
( @contact_group_id INT ) AS
DELETE FROM [contact_group]
WHERE ( [contact_group_id] = @contact_group_id );
RETURN
```

Stored Procedures Using Reverse Mapping

Reverse mapping is a little different because we may already have stored procedures. If we were to run the Reverse Mapping Wizard against the tables and stored procedures created in the previous walk-through, the Advanced View tree view would look something like the figure below. Notice that the Stored Procedures node already recognizes the stored procedures we created previously. Also notice the naming convention that starts with "sp_oa", then adds the name of the operation e.g. "del", then the name of the table.

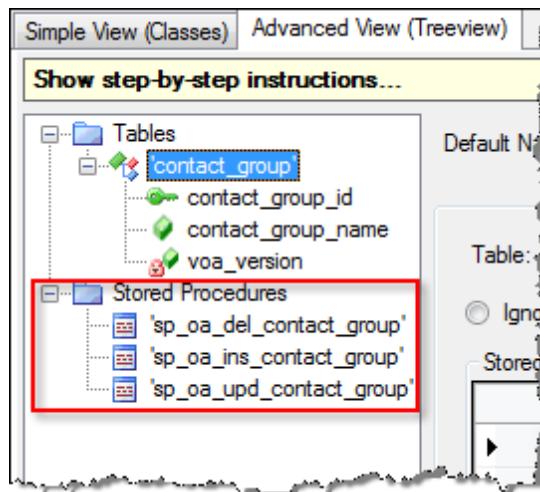


Figure 114

- G** If you receive error "Missing partial modifier on declaration of type 'Model.ContactGroup'; another
o partial declaration of this type exists", when reverse engineering remember that OpenAccess will leave
t your existing ContactGroup.cs class alone by default. You will have to open the Contact.Telerik.
c OpenAccess.cs file and copy the commented code at the bottom of the file to overwrite Contact.cs.
h
a
!
①

By clicking the table name within the Tables node of the tree you will see a number of options for configuring the reverse engineered classes. When the **Stored Procedure** checkbox is enabled, a new Stored Procedure Mapping area displays:

Required Parameter	Mapped Parameter
DELETE : sp_oa_del_contact_group3	<Use Dynamic SQL>
contact_group_id : int	
INSERT : sp_oa_ins_contact_group3	<Use Dynamic SQL>
voa_version : smallint	
contact_group_name : varchar	
contact_group_id : int	
UPDATE : sp_oa_upd_contact_group3	<Use Dynamic SQL>

Figure 115

Each possible operation (Delete, Insert, Update) has a drop down that lets you select how the operation will be performed. You can have OpenAccess automatically generate Dynamic SQL, create a new stored procedure in the database or point to one of the existing stored procedures in the database.

- If you elect the "<Create Stored Procedure>" option, be sure the project "Update Database" configuration setting is set to True.

Required Parameter	Mapped Parameter
DELETE : sp_oa_del_contact_group3	<Use Dynamic SQL>
contact_group_id : int	<Use Dynamic SQL>
INSERT : sp_oa_ins_contact_group3	<Create Stored Procedure> 'sp_oa_del_contact_group' 'sp_oa_ins_contact_group' 'sp_oa_upd_contact_group'
voa_version : smallint	
contact_group_name : varchar	
contact_group_id : int	
UPDATE : sp_oa_upd_contact_group3	<Use Dynamic SQL>

Figure 116

Also notice the set of parameters required by each operation. You can choose from a drop down list of parameters available in the stored procedure.

Required Parameter	Mapped Parameter
DELETE : sp_oa_del_contact_group	'sp_oa_del_contact_group'
contact_group_id : int	
INSERT : sp_oa_ins_contact_group	'sp_oa_ins_contact_group'
voa_version : smallint	@voa_version : smallint
contact_group_name : varchar	@voa_version
contact_group_id : int	@voa_version @contact_group_name @contact_group_id
UPDATE : sp_oa_upd_contact_group	

Figure 117

Executing Stored Procedures

Once the stored procedures are defined in the database you need a way to call them. The `IObjectScope.GetSqlQuery()` method lets you set up parameters and call a stored procedure. As shown in the previous section "Using SQL with OpenAccess", the `GetSqlQuery()` method takes the SQL to be executed, i.e. the name of the stored procedure followed by "?" for each parameter. The second parameter is the Type for the result and the third is a list of parameter types and names. The example below executes the delete stored procedure and takes a single integer parameter that denotes the contact group id.

```

    Public Shared Function spOaDelContactGroup( _
        ByVal scope As IObjectScope, _
        ByVal contactgroupid As Integer) As IQueryResult
        Dim query As IQuery = _
            scope.GetSqlQuery("sp_oa_del_contact_group ?", _
            Nothing, "INTEGER contact_group_id")

        Dim res As IQueryResult = _
            query.Execute(New Object() { contactgroupid })
        'Actually executes the query
        Dim a As Integer = res.Count

        Return res
    End Function

```

```
public static IQueryResult spOaDelContactGroup(
    IObjectScope scope, int contactgroupid)
{
    IQuery query =
        scope.GetSqlQuery("sp_oa_del_contact_group ?",
            null, "INTEGER contact_group_id");

    IQueryResult res =
        query.Execute(new object[] { contactgroupid });
    int a = res.Count; //Actually executes the query

    return res;
}
```

To make this even easier, the Reverse Mapping Wizard will generate all the code you need to execute stored procedures. In the Reverse Mapping Wizard, Advanced View tab, select one of the Stored Procedures in the tree view.

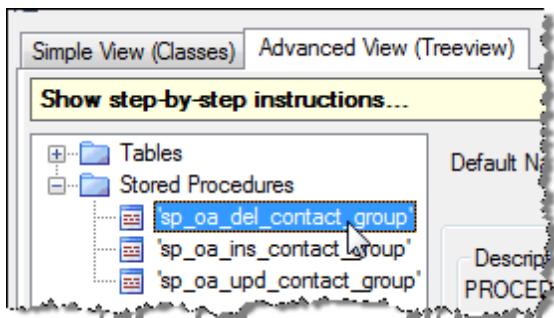


Figure 118

To the right of the tree view a panel with stored procedure information and configuration displays. The top of the panel shows the stored procedure SQL. Below that the properties let you configure a .NET method to operate on the stored procedure. The key property "Generate method" needs to be set to True so that OpenAccess will automatically generate a `StoredProcedures` class with static methods for each of your stored procedures. The bottom of the panel shows the code that will be generated.

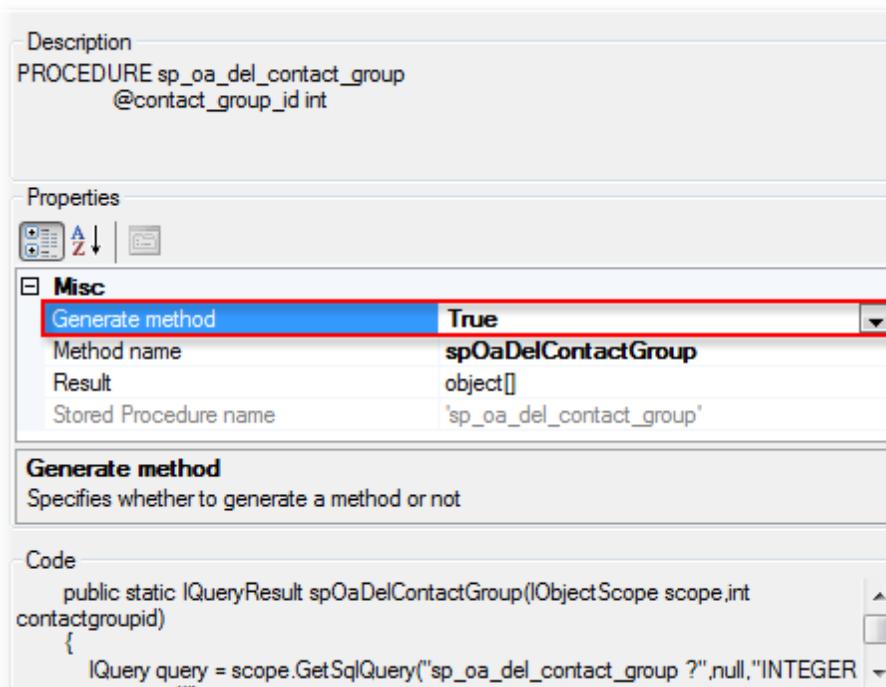


Figure 119

With the `StoredProcedures` class you can simply call the corresponding static method and pass parameters. The example below calls the `sp_oa_ins_contact_group` stored procedure and passes the version of the record, the name and ID of the group. The second stored procedure "spOaUpdContactGroup" updates the record passing in a new `GroupName` and sets the version to "2".

```

④ Using scope As IObjectScope = ObjectScopeProvider1.GetNewObjectScope()
    StoredProcedures.spOaInsContactGroup(scope, 1, "New Group", 1)
    StoredProcedures.spOaUpdContactGroup(scope, 2, "New Group Revised", 1, 1)
    StoredProcedures.spOaDelContactGroup(scope, 1)

End Using

④ using (IObjectScope scope = ObjectScopeProvider1.GetNewObjectScope())
{
    StoredProcedures.spOaInsContactGroup(scope, 1, "New Group", 1);
    StoredProcedures.spOaUpdContactGroup(scope, 2, "New Group Revised", 1, 1);
    StoredProcedures.spOaDelContactGroup(scope, 1);
}
```

8.3 Wrap Up

In this chapter you explored how OpenAccess accesses the database using LINQ, Object Query Language (OQL) and native SQL, and the tradeoffs involved with each choice. You used the `scope Extent<T>()` method to retrieve source data and then manipulated the data in several ways using LINQ expressions and

methods. You used OQL queries to perform simple object-oriented queries, parameterized queries and CRUD operations. You saw how the OQL Query Browser can be used to audition OQL statements. Finally you learned how to run native SQL statements against the database backend as well as use native stored procedures to perform CRUD operations.

Part



IX

Optimization

9 Optimization

This chapter demonstrates how to use Fetch Plans and caching to improve performance.

In this chapter you will learn:

- How to configure Fetch Plans to reduce the number of columns of data retrieved.
- How to configure the built-in caching system to quickly improve performance for single process access.

9.1 Fetch Plans

Tour of Fetch Plans



Find the source projects for this section at \Projects\ORM\CS\12_FetchPlans\12_FetchPlans.sln

By default OpenAccess implements "Lazy Loading" where data is loaded on demand when referenced. For example, if you are loading products and each product has a product category, a category is not loaded until its referenced. **Fetch Plans** allow you to achieve more optimized database queries by specifying fields that are retrieved from the database immediately. Using fetch plans can result in huge performance increases where the number of records that would need to be retrieved individually are brought back all at one time.

To create a fetch plan you first mark individual fields with the **FetchField** attribute. The FetchField attribute takes the name of a "fetch group". Fetch groups are collections of fields that should be retrieved together. The example snippet below shows "_contactName" and "contactGroup" marked with the FetchField attribute. Both fields will be part of the "ContactFetch" fetch group. "_phone" will be "lazy loaded" and will not be retrieved until requested.

```
✉ <Telerik.OpenAccess.FetchField("ContactFetch")> _  
    Private _contactName As String  
  
    Private _phone As String  
  
<Telerik.OpenAccess.FetchField("ContactFetch")> _  
    Private _contactGroup As ContactGroup
```

```
✉ [Telerik.OpenAccess.FetchField("ContactFetch")]  
    private string _contactName;  
  
    private string _phone;  
  
    [Telerik.OpenAccess.FetchField("ContactFetch")]  
    private ContactGroup _contactGroup;
```

Then, in code, you add the name of your fetch group to the IObjectScope FetchPlan before retrieving any data.

```
✉ scope.FetchPlan.Clear()  
    scope.FetchPlan.Add("ContactFetch")
```

```

scope.FetchPlan.Clear();
scope.FetchPlan.Add("ContactFetch");

```

You can use the "OpenAccessTracer" object to trace database calls made by OpenAccess. This allows you to see how changes in your fetch plan help or hinder the number of calls made to the database.

- 💡 You can find OpenAccessTracer.cs in the "OpenAccess VB/C# Examples" project. There are two versions, one for web applications and one for WinForms. You can copy this object into your project. .

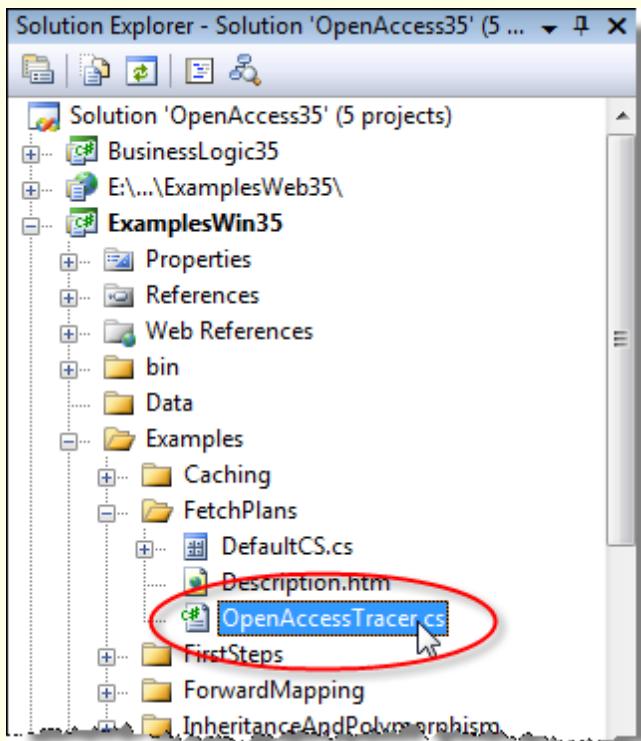


Figure 120

Create an instance of the tracer and pass a text box and label in the constructor.

```

Private tracer As OpenAccessTracer = Nothing

Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
    tracer = New OpenAccessTracer(textBoxStatus, labelStatus)
    '...
End Sub

```

```

private OpenAccessTracer tracer = null;

private void Form1_Load(object sender, EventArgs e)
{
    tracer = new OpenAccessTracer(textBoxStatus, labelStatus);
    //...
}

```

The number of database calls will display in the label and each query will be listed in the text box.

4 queries executed.

```

1: select @@version
2: SELECT a.[contact_id] AS COL1, a.[contact_name] AS COL2, a.[contact_group_id] AS COL3, a.
   [voa_version] AS COL4, b.[contact_group_id] AS COL5, b.[contact_group_name] AS COL6, b.[voa_version]
   AS COL7 FROM [contact] a LEFT JOIN [contact_group] AS b ON (a.[contact_group_id] = b.
   [contact_group_id])
3: SELECT [contact_group_name], [voa_version]  FROM [contact_group] WHERE [contact_group_id] = ?
4: SELECT [contact_group_name], [voa_version]  FROM [contact_group] WHERE [contact_group_id] = ?

```

Figure 121

Let's look at an example using the ContactGroup and Contact objects and see how each change in the fetch plan changes the resulting traffic to the database. To start with, the ContactGroup and Contact objects are not decorated with any FetchField attributes. The object scope Extent<Contact>() method retrieves all the contacts in the table. A grid displays ContactName and ContactGroup.ContactGroupName columns.

Drag a column here to group by this column.		
	Contact	Group
▶	Partner 2	Partner
	Customer 1	Customer
	Customer 3	Customer
	Partner 1	Partner

Figure 122

The trace shows the queries sent to the database:

```

1: select @@version
2: SELECT [contact_id] AS COL1
   FROM [contact]
3: SELECT [contact_group_id], [contact_name], [phone], [voa_version]
   FROM [contact] WHERE [contact_id] = ?
4: SELECT [contact_group_name], [voa_version]
   FROM [contact_group] WHERE [contact_group_id] = ?
5: SELECT [contact_group_id], [contact_name], [phone], [voa_version]
   FROM [contact] WHERE [contact_id] = ?
6: SELECT [contact_group_name], [voa_version]
   FROM [contact_group] WHERE [contact_group_id] = ?
7: SELECT [contact_group_id], [contact_name], [phone], [voa_version]
   FROM [contact] WHERE [contact_id] = ?
8: SELECT [contact_group_id], [contact_name], [phone], [voa_version]
   FROM [contact] WHERE [contact_id] = ?

```

Even though we're only after four contacts and two contact groups ("Partner" and "Customer"), eight queries

are generated! OpenAccess first retrieves the version of the database for its own internal use in line #1. Then OpenAccess selects all the contact_id's from the Contact table in line #2. In lines #3, #5, #7 and #8 the ContactName is retrieved for each contact. In lines #4 and #6 "contact_group_name" is pulled from the database. That's a ton of traffic for so small a result set.

Now if we add a FetchField attribute to the _contactName field we can see some improvement right away. The top of the Contact class now looks like the example below:

```
④ <Telerik.OpenAccess.Persistent()> _
Public Class Contact
    <Telerik.OpenAccess.FetchField("ContactFetch")> _
    Private _contactName As String

    Private _phone As String
    Private _contactGroup As ContactGroup
    ...
End Class
```

```
④ [Telerik.OpenAccess.Persistent()]
public class Contact
{
    [Telerik.OpenAccess.FetchField("ContactFetch")]
    private string _contactName;

    private string _phone;
    private ContactGroup _contactGroup;
//...
}
```

The form Load event handler sets up the fetch plan, configures and binds the grid:

```
④ Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
    tracer = New OpenAccessTracer(tbStatus, lblStatus)

    Dim scope As IObjectScope = ObjectScopeProvider1.GetNewObjectScope()

    scope.FetchPlan.Clear()
    scope.FetchPlan.Add("ContactFetch")

    Dim list As List(Of Contact) = scope.Extent(Of Contact)().ToList()

    radGridView1.MasterGridViewTemplate.AutoGenerateColumns = False
    radGridView1.ReadOnly = True
    radGridView1.Columns.Add(New GridViewTextBoxColumn("ContactName"))
    radGridView1.Columns.Add( _
        New GridViewTextBoxColumn("ContactGroup.ContactGroupName"))
    radGridView1.Columns(0).HeaderText = "Contact"
    radGridView1.Columns(1).HeaderText = "Group"
    radGridView1.DataSource = list
    radGridView1.MasterGridViewTemplate.BestFitColumns()
End Sub
```

```

1  private void Form1_Load(object sender, EventArgs e)
2  {
3      tracer = new OpenAccessTracer(tbStatus, lblStatus);
4
5      IObjectScope scope = ObjectScopeProvider1.GetNewObjectScope();
6
7      scope.FetchPlan.Clear();
8      scope.FetchPlan.Add("ContactFetch");
9
10     List<Contact> list = scope.Extent<Contact>().ToList();
11
12     radGridView1.MasterGridViewTemplate.AutoGenerateColumns = false;
13     radGridView1.ReadOnly = true;
14     radGridView1.Columns.Add(new GridViewTextBoxColumn("ContactName"));
15     radGridView1.Columns.Add(
16         new GridViewTextBoxColumn("ContactGroup.ContactGroupName"));
17     radGridView1.Columns[0].HeaderText = "Contact";
18     radGridView1.Columns[1].HeaderText = "Group";
19     radGridView1.DataSource = list;
20     radGridView1.MasterGridViewTemplate.BestFitColumns();
21 }

```

After running this example with "_contactName" included in the fetch plan we're down to six statements. Now we're getting all the contact name data in a single call on line #2.

```

1: select @@version
2: SELECT [contact_id] AS COL1, [contact_name] AS COL2, [voa_version] AS COL3
   FROM [contact]
3: SELECT a.[contact_group_id], a.[voa_version], b.[contact_group_id],
   b.[contact_group_name], b.[voa_version]
   FROM [contact] a LEFT JOIN [contact_group] AS b
   ON (a.[contact_group_id] = b.[contact_group_id])
   WHERE a.[contact_id] = ?
4: SELECT a.[contact_group_id], a.[voa_version], b.[contact_group_id],
   b.[contact_group_name], b.[voa_version]
   FROM [contact] a LEFT JOIN [contact_group] AS b
   ON (a.[contact_group_id] = b.[contact_group_id])
   WHERE a.[contact_id] = ?
5: SELECT a.[contact_group_id], a.[voa_version], b.[contact_group_id],
   b.[contact_group_name], b.[voa_version]
   FROM [contact] a LEFT JOIN [contact_group] AS b
   ON (a.[contact_group_id] = b.[contact_group_id])
   WHERE a.[contact_id] = ?
6: SELECT a.[contact_group_id], a.[voa_version], b.[contact_group_id],
   b.[contact_group_name], b.[voa_version]
   FROM [contact] a LEFT JOIN [contact_group] AS b
   ON (a.[contact_group_id] = b.[contact_group_id])
   WHERE a.[contact_id] = ?

```

This is certainly an improvement but we're still retrieving the contact_group_name columns four times in lines #3 - #6, even though there's only two contact groups to worry about. By specifying "_contactGroup" as a FetchField we are able to cut this down to only two additional calls.

```

1: select @@version
2: SELECT a.[contact_id] AS COL1, a.[contact_name] AS COL2,
       a.[contact_group_id] AS COL3, a.[voa_version] AS COL4,
       b.[contact_group_id] AS COL5
  FROM [contact] a
 LEFT JOIN [contact_group] AS b
    ON (a.[contact_group_id] = b.[contact_group_id])
3: SELECT [contact_group_name], [voa_version]
  FROM [contact_group]
 WHERE [contact_group_id] = ?
4: SELECT [contact_group_name], [voa_version]
  FROM [contact_group]
 WHERE [contact_group_id] = ?

```

But the statement in #2 already joins to the contact_group table so we really don't want even those two calls. The FetchField attribute has a "Path" parameter that can be used to get specific fields in a referenced object. We can use this to tell OpenAccess that we will be needing the "contact_group_name" column. The attribute will look like this in code when we decorate the _contactGroup field.

```

④ <Telerik.OpenAccess.FetchField( _
      "ContactFetch", Path := "_contactGroupName")> _
Private _contactGroup As ContactGroup

④ [ Telerik.OpenAccess.FetchField("ContactFetch",
      Path = "_contactGroupName")]
private ContactGroup _contactGroup;

```

- G** This last step may not always prevent the last two queries. Why? The garbage collector
- o** sometimes removes referenced data before you access it. This is because weak references to the
- t** data are kept only if referenced directly. To avoid this behavior you can set weak references to
- c** strong by adding this node to the backend configuration section in your app.config file:
- h**
- a** <pmCacheRefType>STRONG</pmCacheRefType>
- !** Using strong references, all objects are kept in memory until the ObjectScope is disposed.

From the Programmers Guide topic "Caching Optimization":

"Each objectscope maintains a dictionary of object identity to in-memory objects, these references are of type System.WeakReference. Additionally the objectscope keeps strong references to all inserted, updated and deleted objects for as long as the transaction lasts."

By default, the reference type is set to "Weak". In case of weak references, unreferenced instances are garbage collected very quickly. Weak references are useful for applications with generally short lived objectscopes and large heaps (e.g. web applications). Less objects on the heap will reduce garbage collection time. In case of "Strong" references, unreferenced instances are only garbage collected when they are manually evicted by the application or when the objectscope is closed. Strong references are useful for applications that always have short lived objectscopes (e.g. applications using the "one objectscope per request" model)."

Fetch Plans Using The Forward Mapping Wizard

You can use the Forward Mapping Wizard to specify fetch fields and groups. Using the wizard simply adds the `FetchField` attributes to the appropriate fields. The steps are:

- 1) Select an object from the Persistent node of the tree view found on the left side of the wizard.
- 2) Select the "Fetch Group" tab found on the right side of the wizard.
- 3) Click the Add/Remove button to display the "Add/Remove FetchGroup" dialog.
- 4) Click the Add button and enter a name for the fetch group. Click the Ok button to close the "Add/Remove FetchGroup" dialog. This will add a new column in the "Class Fetch Groups" list found in the Fetch Group tab.
- 5) Click one or more fields to include in the fetch group.

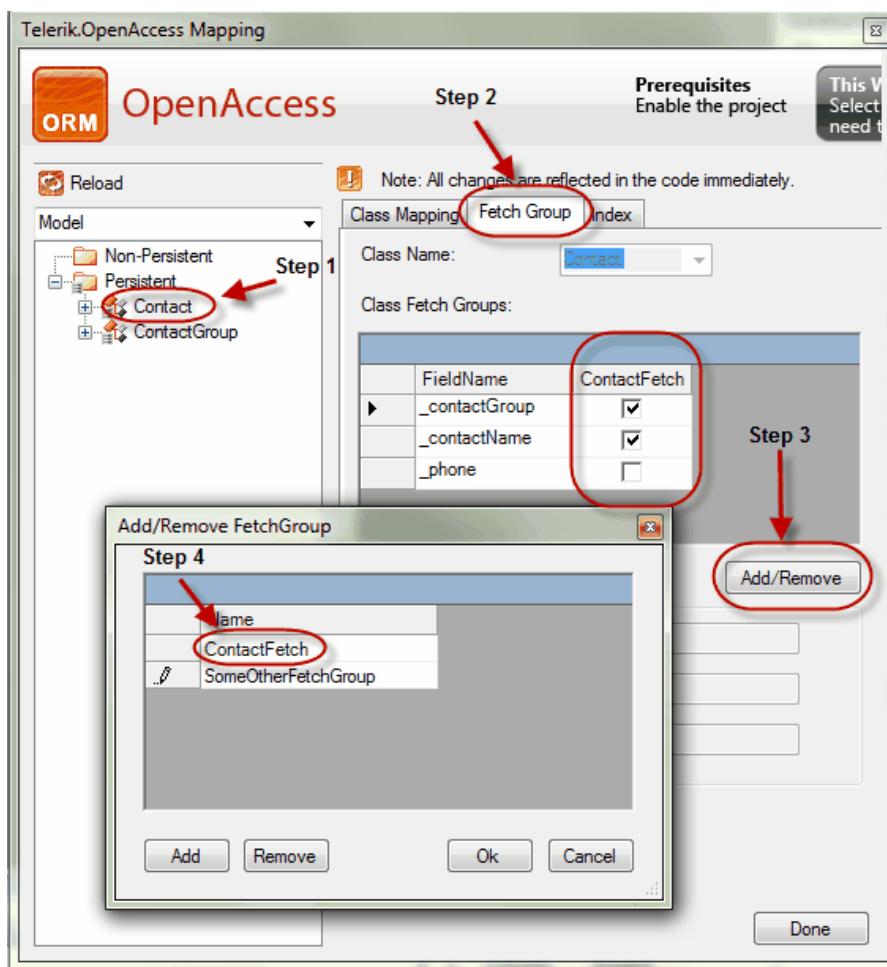


Figure 123

Notice the wizard "Reference Field Settings" area where you can tailor `FetchField` attribute parameters.

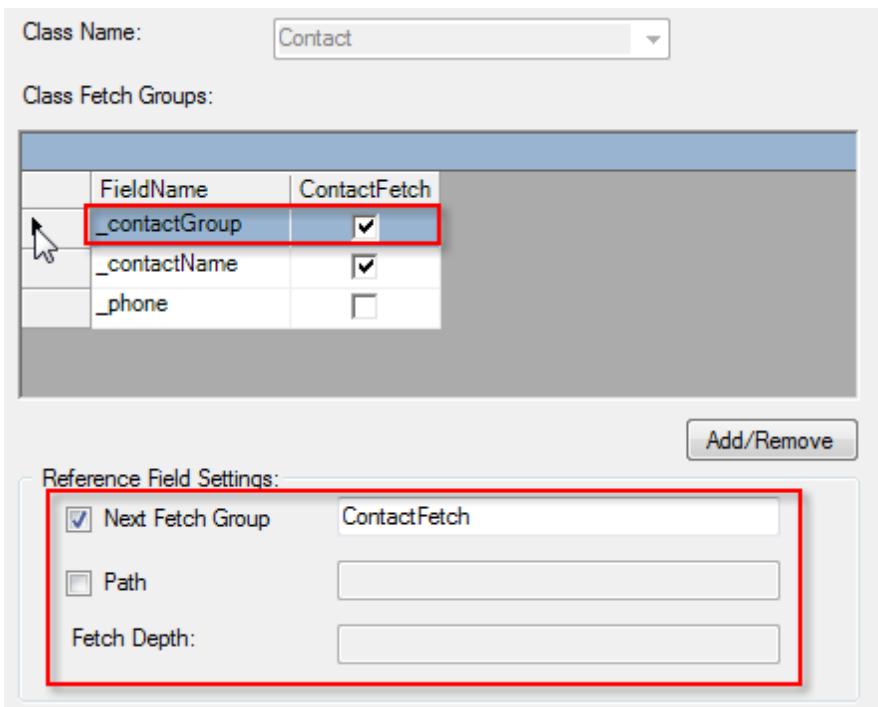


Figure 124

- **Next Fetch Group** takes the name of a fetch group that should be used for the currently selected referenced object.
- **Path** is the name of a field in a referenced object that should be loaded. In the snippet below, "_contactGroupName" from the ContactGroup object is loaded with the other fields of the "ContactFetch" group. You can add the FetchField attribute multiple times with the Path parameter to specify individual fields in a referenced object.

```
<Telerik.OpenAccess.FetchField("ContactFetch", Path := "_contactGroupName")> _  
Private _contactGroup As ContactGroup
```

```
[Telerik.OpenAccess.FetchField("ContactFetch", Path = "_contactGroupName")]  
private ContactGroup _contactGroup;
```

- **Fetch Depth:** This parameter is used while defining the depth of references, in cases where an object of a particular type refers to objects of the same type or results in a circular reference. For example, if you have an employees table and want to retrieve a clerk and his manager you could indicate a recursion depth of 1. If you want the clerk, manager and all managers up to the CEO you could leave the Depth parameter out.

Using the Fetch Plan Browser

From the Telerik, OpenAccess menu you can select the **Fetch Plan Browser...** option to manage fetch plans at a higher level and to create an object to store all the fetch plans in your project. The dialog looks like the screenshot below and can generally be worked with in a top-to-bottom manner.

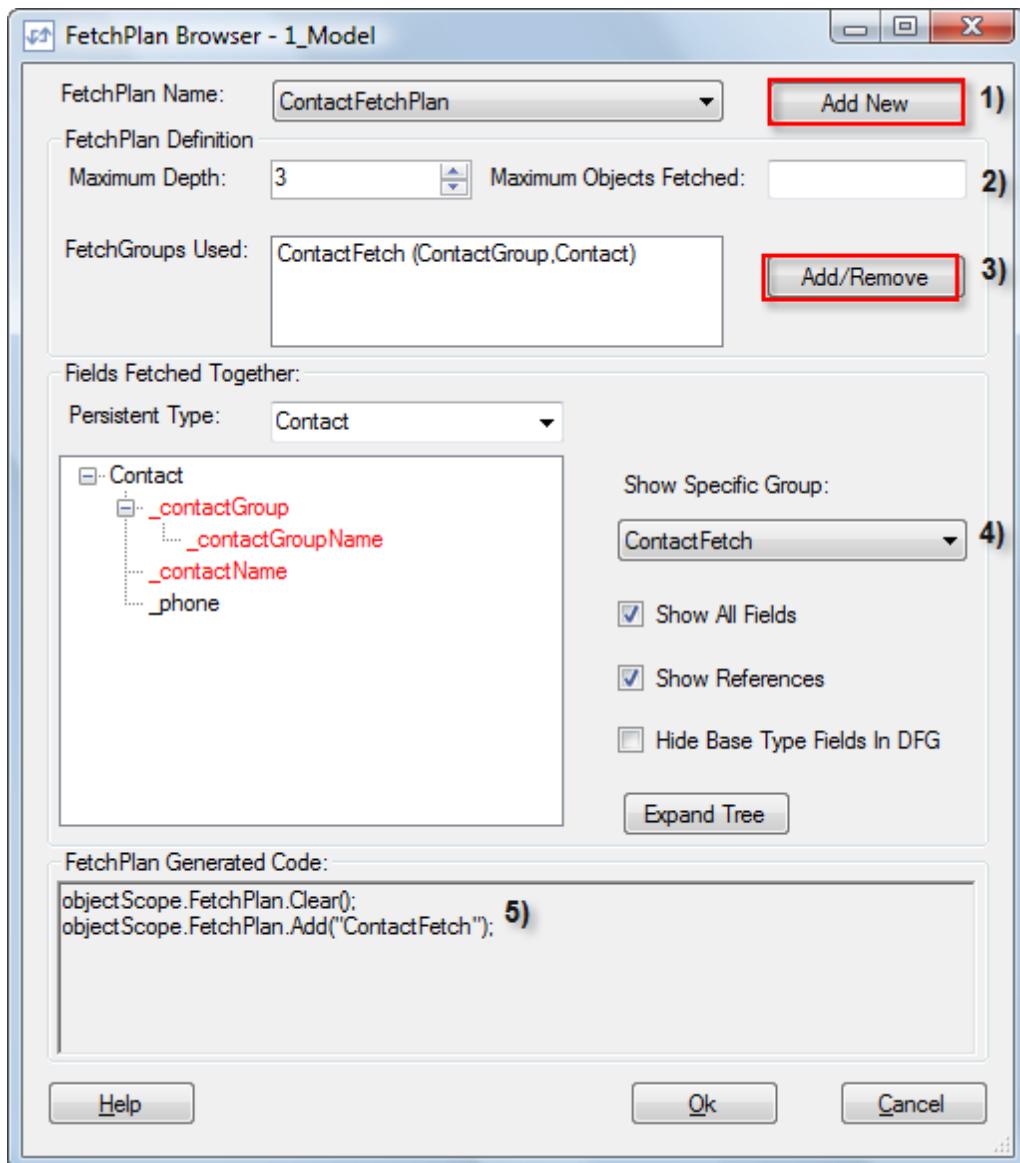


Figure 125

- 1) Click the **Add New** button to create a new FetchPlan instance. Use the drop down list to the left of this button to select existing FetchPlans.
- 2) Enter the **Maximum Depth** of levels to be fetched and the **Maximum Objects Fetched** to control the total number of objects that can be retrieved.
- 3) Click the **Add/Remove** button to add or remove existing fetch groups from the fetch plan. Clicking the button displays the Edit FetchPlan dialog showing all possible fetch groups on the left and allowing you to move one or more groups to the fetch plan you're currently editing.

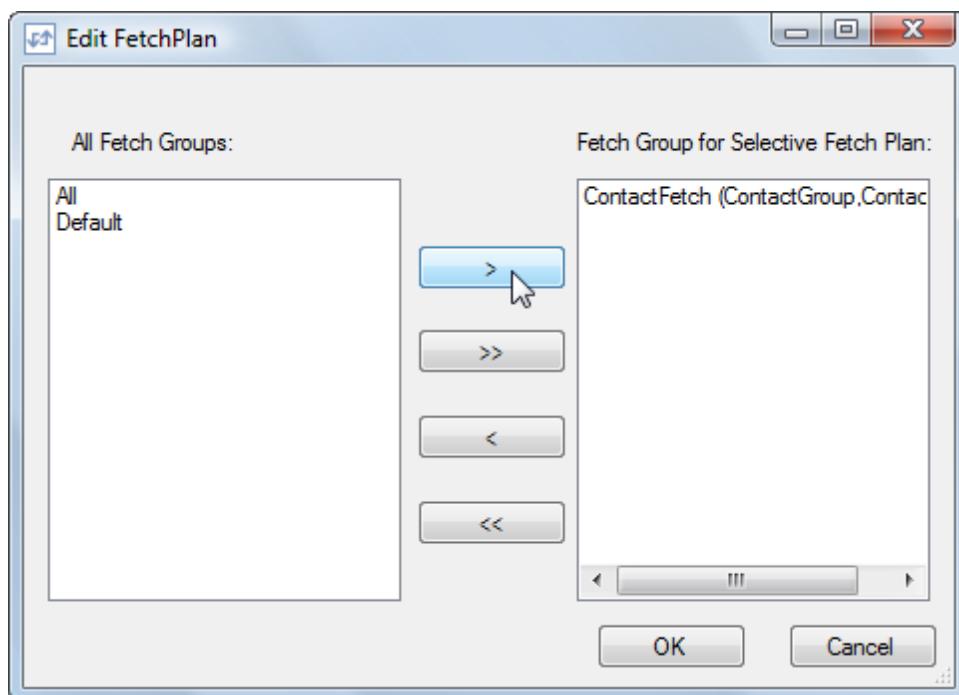


Figure 126

- 4) The section collectively titled **Fields Fetched Together** can select fetch groups from the drop down list and see how they affect selected persistent objects in the treeview on the left. Notice that the fields that will be fetched are highlighted in red.
- 5) The FetchPlan Generated code section shows what the relevant portion of code will look like when you click the Ok button.

Clicking the Ok button will generate a new FetchPlans.cs class with your plans predefined as static members.

```
Public Class FetchPlans
    Public Shared ContactFetchPlan As New FetchPlan(New String() _
    { "ContactFetch" }, FetchPlan.DefaultMaxDepth, FetchPlan.NoLimit)
End Class
```

```
public class FetchPlans
{
    public static FetchPlan ContactFetchPlan =
        new FetchPlan(
            new string[] { "ContactFetch" },
            FetchPlan.DefaultMaxDepth,
            FetchPlan.NoLimit);
}
```

Now you can use the new FetchPlans object in place of your existing code:

```
'scope.FetchPlan.Clear();
'scope.FetchPlan.Add('ContactFetch');
scope.FetchPlan = FetchPlans.ContactFetchPlan
```

```
 //scope.FetchPlan.Clear();  
//scope.FetchPlan.Add("ContactFetch");  
scope.FetchPlan = FetchPlans.ContactFetchPlan;
```

9.2 Caching

Caching Overview

Not only is rewriting a data layer a prime example of "re-inventing the wheel", re-writing your own caching mechanism is another big time consumer with little payoff. OpenAccess supplies a "2nd level" cache that automatically bypasses trips to the database. The cache is a container for object and query data. The cache is filled only by read access, which means the content will be whatever the application demands from the database. When the application updates or deletes persistent objects, the entries that depend on them are evicted from the cache. This means, that if you change the name of a persistent Person instance, the corresponding entry is removed from the data cache, but also all query cache entries depending on the class Person are removed.

All this happens in the process memory space, as the 2nd level cache is an in-process cache.

When there is a need for multiple processes or applications to use the 2nd level cache, we need a way to evict changed and deleted entries. Currently, OpenAccess provides an implementation using MSMQ as the transport vehicle for eviction messages. Why MSMQ? Because MSMQ supplies a reliable multicasting mechanism to send eviction messages to many participants. Again, we are only evicting entries from the various participants. New and updated data is never pushed directly to the 2nd level caches and local reading is still the only way to populate the local cache instance.

Single Process Caching Walk Through

In this next walk-through we will load different sets of data with tracing viewable in the form to see what query statements are sent to the database. Then, we turn on caching and observe how the data comes back immediately without sending a query request to the database.



Find the source at \Projects\ORM\CS\13_Caching\13_Caching.sln

Build the Data Model Project

- 1) Create a new Class Library project called "Model".
- 2) ORM-enable the project. Specify the following:
 - a) The Persistent classes option should be enabled.
 - b) The Data Access Code option should be disabled.
 - c) The database connection ID should be "NorthwindConnection"
 - d) The database should be "NorthwindOA".
- 3) From the Telerik > OpenAccess menu, select the Reverse Mapping option.
- 4) In the Simple View tab of the Reverse Engineering Wizard, de-select the Generate checkbox for all but

the "Customers", "Orders", and "Products" tables.

- 5) Select the Advanced View tab.
- 6) Open the "Orders" node, select the "employee" reference and click the Remove button. Also click the "shipper" reference and click the Remove button.
- 7) Click the Generate & Save Config button.
- 8) Click Yes to confirm the Generate Sources dialog.
- 9) Build the project.

Build the Windows Form Application

- 1) Create a new Windows Forms application called "WinApp".
- 2) ORM-enable the project. Specify the following:
 - a) The Persistent classes option should be disabled.
 - b) the Data Access Code option should be enabled.
 - c) The database connection ID should be "NorthwindConnection"
 - d) The database should be "NorthwindOA".
- 3) In the Solution Explorer, add a reference to the "Model" class library.
- 4) Add references to the **System.Reflection**, **Telerik.OpenAccess** and **Telerik.WinForms.UI** namespaces.
- 5) Add a new class **OpenAccessTracer.cs**. This class is available in the example projects that ship with OpenAccess but for now you can copy in the class code:

```
Imports Microsoft.VisualBasic
Imports System
Imports System.Diagnostics
Imports System.Threading
Imports Telerik.WinForms.UI

Public Class OpenAccessTracer
    Inherits TraceListener
    Private id As Integer
    Private count As Integer
    Private textBox As RadTextBox
    Private label As RadLabel
    Private myThread As Thread

    Public Sub New(ByVal textBox As RadTextBox)
        MyBase.New("OpenAccess")

        myThread = Thread.CurrentThread

        Telerik.OpenAccess.Diagnostics.TraceAdapter.Instance.Level = "4"
        id =
        Telerik.OpenAccess.Diagnostics.TraceAdapter.Instance.Listeners.Add(Me)
        Me.textBox = textBox

        count = 0
    End Sub
```

```

Public Sub New(ByVal textBox As RadTextBox, ByVal label As RadLabel)
    MyBase.New("OpenAccess")

    myThread = Thread.CurrentThread

    Telerik.OpenAccess.Diagnostics.TraceAdapter.Instance.Level = "4"
    id =
    Telerik.OpenAccess.Diagnostics.TraceAdapter.Instance.Listeners.Add(Me)
    Me.textBox = textBox
    Me.label = label
    count = 0
End Sub

Protected Overrides Overloads Sub Dispose(ByVal disposing As Boolean)
    Me.textBox.Text += count.ToString() & " queries executed."
    Telerik.OpenAccess.Diagnostics.TraceAdapter.Instance.Listeners.RemoveAt(id)
    MyBase.Dispose(disposing)
End Sub

Public Sub Reset()
    textBox.Text = String.Empty
    label.Text = String.Empty
    count = 0
End Sub

Public Overrides Overloads Sub Write(ByVal message As String)
End Sub

Public Overrides Overloads Sub WriteLine( _
ByVal message As String)
    If (Not myThread.Equals(Thread.CurrentThread)) Then
        Return
    End If

    If label IsNot Nothing Then
        label.Text = count.ToString() & " queries executed."
    End If

    If message.StartsWith("driver.stat.execQuery") Then
        count += 1
        textBox.Text += String.Format("{0}: {1}" & _
        Constants.vbCrLf, count.ToString(), _
        message.ToString().Substring(32).Trim())
    End If
End Sub

End Class

```

```

<# using System;
using System.Diagnostics;
using System.Threading;
using Telerik.WinForms.UI;

public class OpenAccessTracer : TraceListener

```

```
{  
    int id;  
    int count;  
    RadTextBox textBox;  
    RadLabel label;  
    Thread myThread;  
  
    public OpenAccessTracer(RadTextBox textBox)  
        : base("OpenAccess")  
    {  
  
        myThread = Thread.CurrentThread;  
  
        Telerik.OpenAccess.Diagnostics.TraceAdapter.Instance.Level = "4";  
        id =  
        Telerik.OpenAccess.Diagnostics.TraceAdapter.Instance.Listeners.Add(this);  
        this.textBox = textBox;  
  
        count = 0;  
    }  
  
    public OpenAccessTracer(RadTextBox textBox, RadLabel label)  
        : base("OpenAccess")  
    {  
  
        myThread = Thread.CurrentThread;  
  
        Telerik.OpenAccess.Diagnostics.TraceAdapter.Instance.Level = "4";  
        id =  
        Telerik.OpenAccess.Diagnostics.TraceAdapter.Instance.Listeners.Add(this);  
        this.textBox = textBox;  
        this.label = label;  
        count = 0;  
    }  
  
    protected override void Dispose(bool disposing)  
    {  
        this.textBox.Text += count.ToString() + " queries executed.";  
        Telerik.OpenAccess.Diagnostics.TraceAdapter.Instance.Listeners.RemoveAt(  
id);  
        base.Dispose(disposing);  
    }  
  
    public void Reset()  
    {  
        textBox.Text = String.Empty;  
        label.Text = String.Empty;  
        count = 0;  
    }  
  
    public override void Write(string message)  
    {  
  
    }
```

```
public override void WriteLine(string message)
{
    if (!myThread.Equals(Thread.CurrentThread))
        return;

    if (label != null)
    {
        label.Text = count.ToString() + " queries executed.";
    }

    if (message.StartsWith("driver.stat.execQuery"))
    {
        count++;
        textBox.Text += string.Format("{0}: {1}\r\n", count.ToString(),
            message.ToString().Substring(32).Trim());
    }
}
```

6) Add controls to the form and set properties:

- a) RadComboBox: **Name** = "cbSelect", **Anchor** = "Top, Right". From the Smart Tag open the **Edit Items** link. Use the **Add** button to create three items and change the item **Text** properties to "Select all products", "Select all customers" and "Select all Orders"
- b) RadGridView: **Name** = "gvMain", **Anchor** = "Top, Bottom, Left, Right".
- c) RadLabel: **Name** = "lblStatus", **Anchor** = "Top, Left".
- d) RadButton: **Name** = "btnReset", **Anchor** = "Bottom, Right".
- e) RadTextBox: **Name** = "tbStatus", **Anchor** = "Bottom, Left, Right".

The form should look something like this:

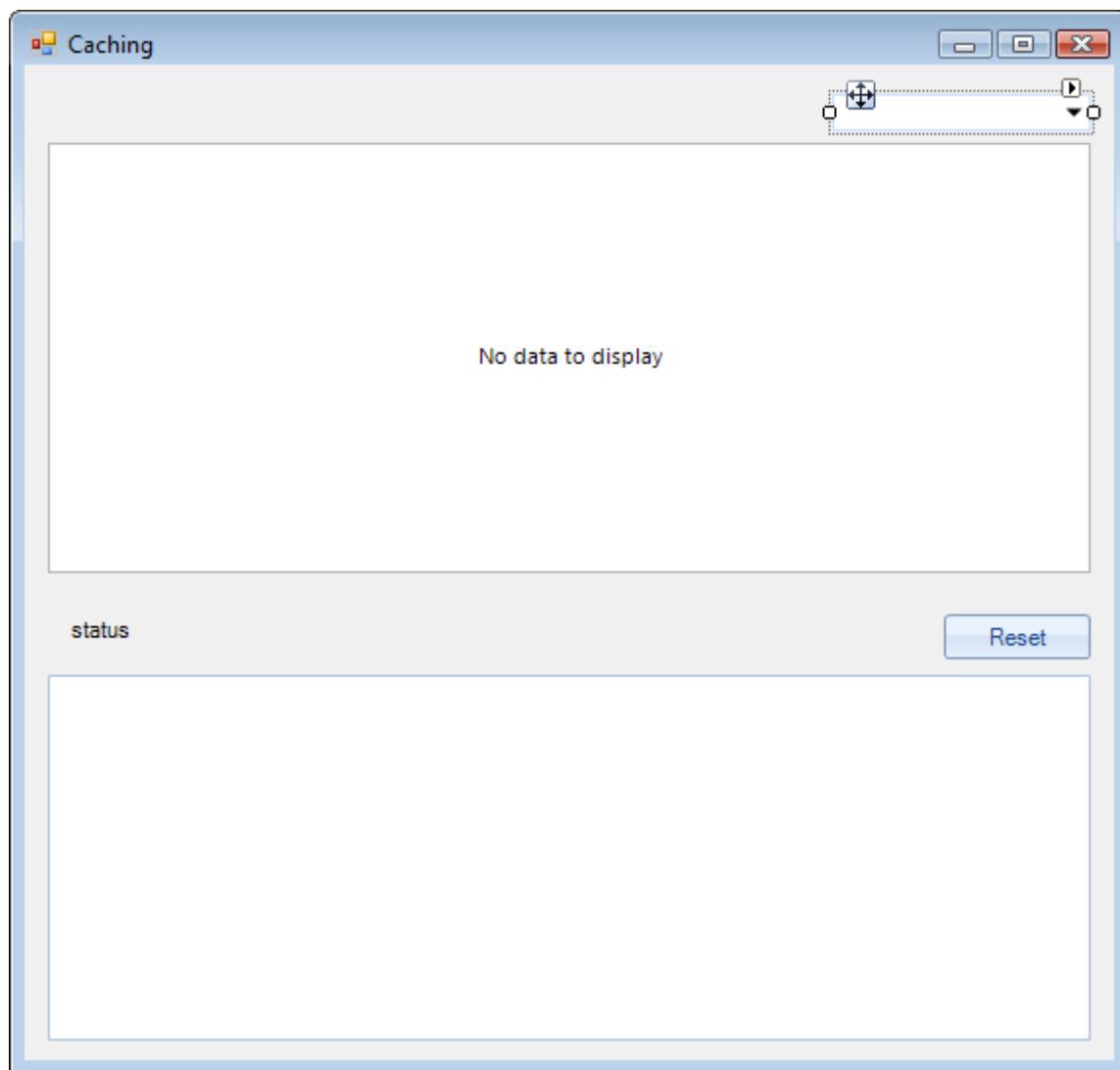


Figure 127

7) In the code behind for the form add a private member for the OpenAccessTracer.

```
Dim tracer As OpenAccessTracer = Nothing  
  
OpenAccessTracer tracer = null;
```

8) Create a Load event handler for the form and add code.

This will create the tracer object and make sure that the compiler knows we're using Model classes.

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)  
    tracer = New OpenAccessTracer(tbStatus, lblStatus)  
    System.Reflection.Assembly.GetAssembly(GetType(Customer))  
End Sub
```

```
■ private void Form1_Load(object sender, EventArgs e)
{
    tracer = new OpenAccessTracer(tbStatus, lblStatus);
    Assembly.GetAssembly(typeof(Customer));
}
```

- 9) Handle the RadComboBox SelectedIndexChanged event and add code to get the current selection and create a corresponding query.

```
■ Private Sub cbSelect_SelectedIndexChanged( _
    ByVal sender As Object, ByVal e As EventArgs)
    Dim selectedQuery As String =
        CType(Me.cbSelect.SelectedItem, RadComboBoxItem).Text
    Dim query As String = String.Empty

    Select Case selectedQuery
        Case "Select all products"
            query = "select * from ProductExtent as p"
            Exit Select
        Case "Select all customers"
            query = "select * from CustomerExtent as c"
            Exit Select
        Case "Select all orders"
            query = "select * from OrderExtent as o"
            Exit Select
    End Select

    Dim scope As IObjectScope =
        ObjectScopeProvider1.GetNewObjectScope()
    Dim result As IQueryResult =
        scope.GetOqlQuery(query).Execute()
    Me.gvMain.DataSource = result
End Sub
```

```
■ private void cbSelect_SelectedIndexChanged(object sender, EventArgs e)
{
    string selectedQuery =
        ((RadComboBoxItem)this.cbSelect.SelectedItem).Text;
    string query = string.Empty;

    switch (selectedQuery)
    {
        case "Select all products":
        {
            query = "select * from ProductExtent as p";
            break;
        }
        case "Select all customers":
        {
            query = "select * from CustomerExtent as c";
            break;
        }
        case "Select all orders":
        {
            query = "select * from OrderExtent as o";
            break;
        }
    }

    IObjectScope scope = ObjectScopeProvider1.GetNewObjectScope();
    IQueryResult result = scope.GetOqlQuery(query).Execute();
    this.gvMain.DataSource = result;
}
```

10)Add a Click event handler for the "Reset" button and add code to call the OpenAccessTracer Reset() method:

```
■ Private Sub btnReset_Click(
    ByVal sender As Object, ByVal e As EventArgs)
    tracer.Reset()
End Sub
```

```
■ private void btnReset_Click(object sender, EventArgs e)
{
    tracer.Reset();
}
```

11)Run the application. Use the drop down list to select all products, customers and orders. Then go back and reselect the same items. Notice the queries logged into the trace textbox at the bottom of the form.

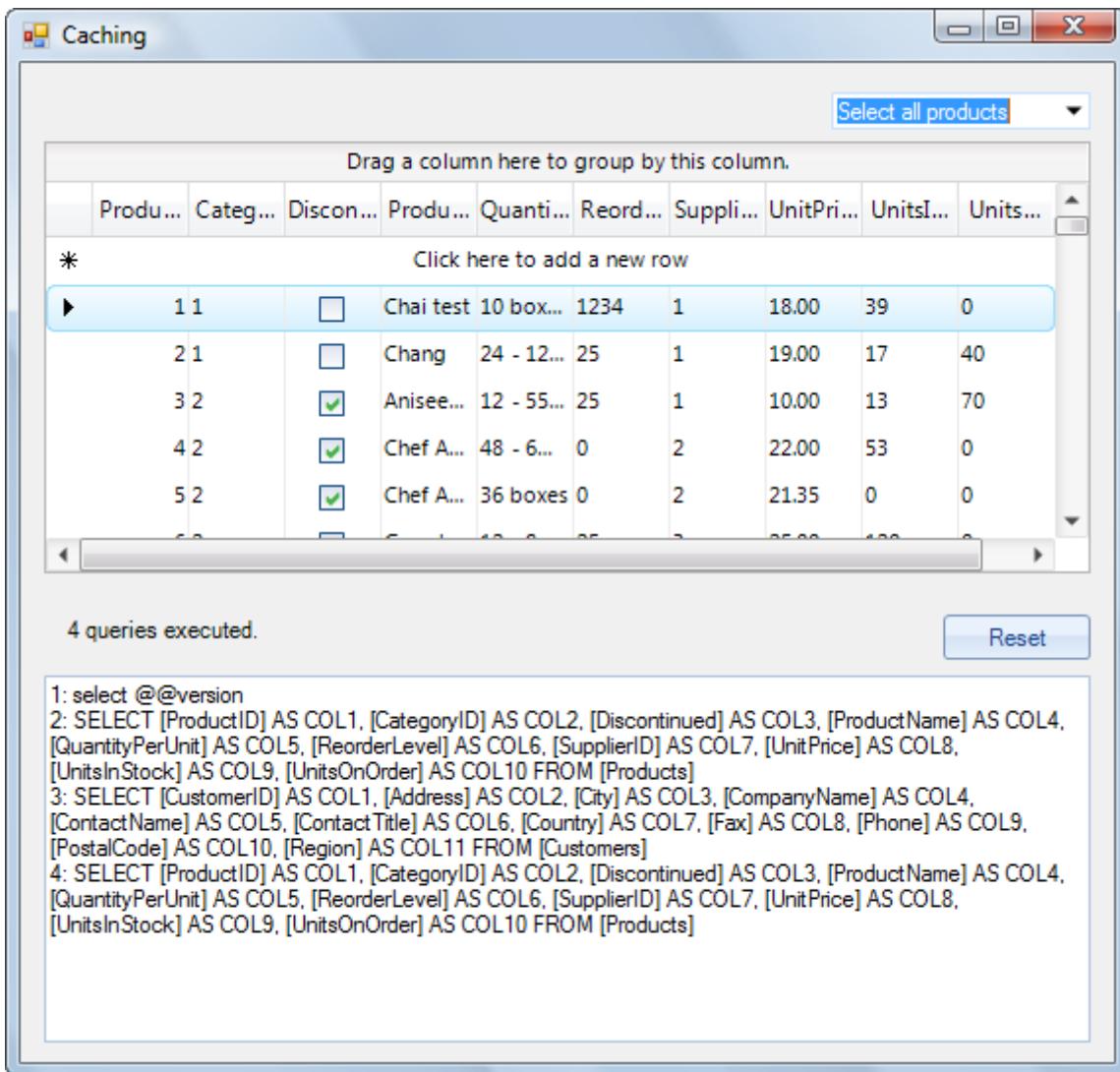


Figure 128

The figure above shows that new queries are sent to the database every time a new selection is made, even for a previous selection.

12) Close the application.

13) In the Telerik menu choose the **OpenAccess > Configuration > Backend Configuration Settings...** option. In the Caching section of the Backend Configuration dialog, turn on the **2nd Level Cache** and **Cache Query Results** options. Leave the Maximum Objects in Cache and Maximum Queries in Cache settings at their defaults. Click **Ok** to write the settings to the configuration file and close the dialog.

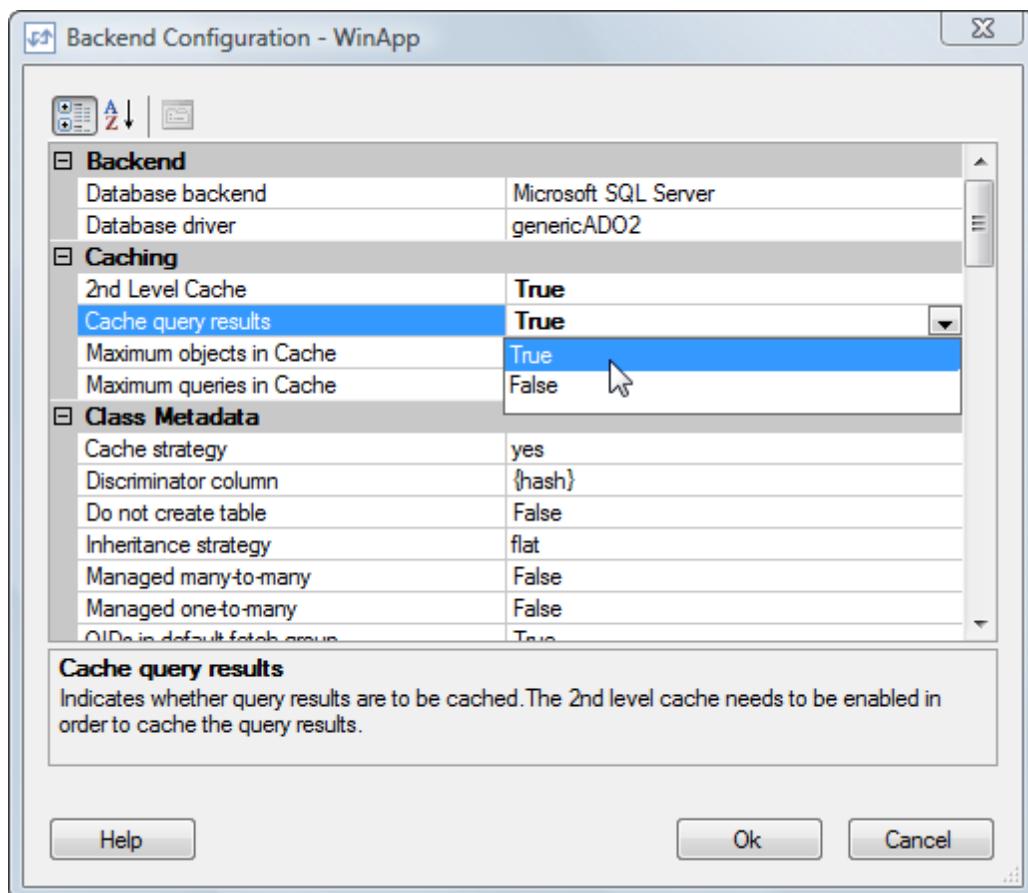


Figure 129

14) Run the application. Use the drop down list to select all products, customers and orders.

Once you select an option once and the query is run on the database, repeated selections use the cache and no further hit to the database is reported in the trace text box. The maximum number of entries in the trace text box will be four: one for OpenAccess to get the database version and one for each select statement.

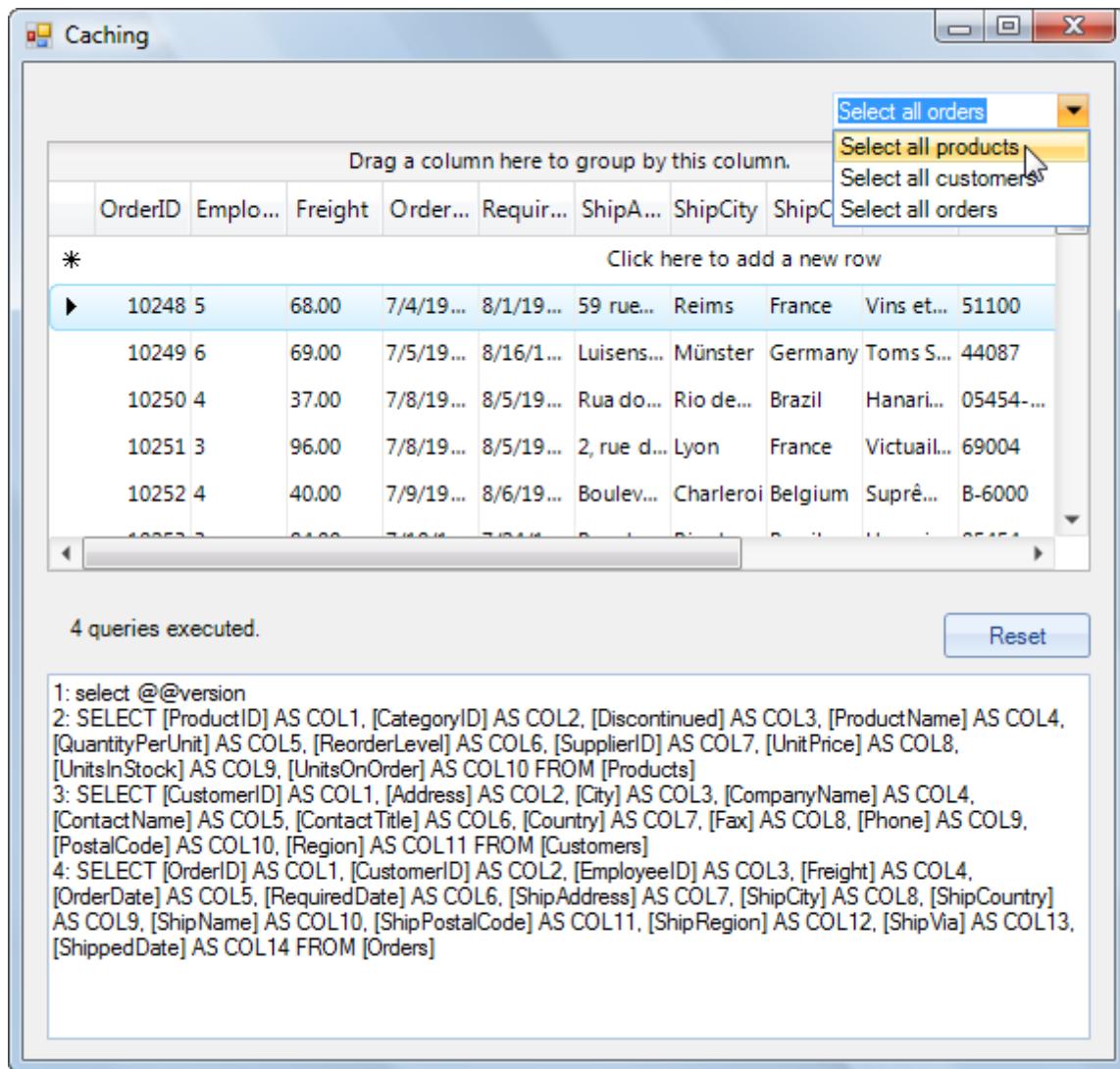


Figure 130

9.3 Wrap Up

In this chapter you learned how Fetch Plans and caching can be used to improve performance. You learned how fetch plans contain fetch groups made of fields marked with the `FetchField` attribute. You saw how small changes to the fetch plan could result in a reduction in the number of columns of data retrieved. You learned how to configure the built-in caching system to quickly improve performance for single process access.

Index

- - -

.Refresh() 195

- 2 -

2nd level cache 244

- A -

A)tomicity 192
abstract 176
ACID 192
Add New 234
Add Service Reference... 105
Add/Remove 234
Advanced View 222
AllowAddNewRow 69
ApplicationException 142
AUTOINC 179, 182
auto-incrementing 179
Automatic Properties 28
AutomaticBegin 69, 195
AutoPostBack 86
AutoSizeColumnsMode 69

- B -

BackColor 69
Backend 219
Backend Configuration 222
Backend Configuration Settings... 54, 222, 244
BackendQuery 210
Begin() 192
BestFitColumns() 69
BindingList 105, 219
BreezeTheme 69
Business Layer 103

- C -

C)onsistency 192

Cache Query Results 244
Caching Optimization 234
Calvert 210
ChangeSet 105
Class Mappings 182
ClientFillPrimitive 69
Close() 69
ComboBox 69
Commit() 69, 192, 195
concurrency 206
Concurrency Control 206
Configure Data Source... 86
Connection Settings 54, 182
Connection Settings... 222
ConnectionURL 219
CopyFrom() 105
Create Database 54, 105
Create Ref 161
CRUD 47, 215

- D -

D)urability 192
Data Access Code 68, 69, 222, 244
Data Layer 103
Database Operations 54, 105
DataGridView 69
delegate 105
Dirty Read 206
DirtyObjects 194
DropDownList 86

- E -

Enable Project 22
Entities 103
Essential LINQ 210
Event Tracing 54
Evict() 195
Execute() 219
ExecuteBindingList() 219
ExecuteList() 219
Extent<>() 47, 69
Extent<T>() 210

- F -

FailFast 195
 FaultCode 142
 FaultException 142
 FaultReason 142
 Fetch Depth 234
 Fetch Group 54
 Fetch Plan Browser... 234
 Fetch Plans 234
 FetchField 234
 Field to Column Mapping 161
 Fields Fetched Together 234
 Firebird 47
 Flat mapping 168
 Flush() 194
 FormClosing 69
 FormElement 69
 Forward Mapping 54, 182, 222
 Forward Mapping (Classes to Tables) 222
 Forward Mapping Wizard 105, 206, 222, 234

- G -

garbage collector 234
 Generate 244
 GetChanges() 105
 GetContent() 135
 GetSqlQuery() 37, 215, 219, 222
 GradientStye 69
 GridView 86

- H -

HIGHLOW 179
 Horizontal mapping 168, 176

- I -

I)solation 192
 IBindingList 37
 IEnumerable 37, 219
 IList 37, 105, 154, 194
 impedance mismatch 168
 inheritance 168
 Internal 179

Introduction to LINQ 210
 IObjectCollector 105
 IObjectScope 28, 37, 86, 135, 192, 200, 210, 222, 234
 IObjectScopeQuery 210
 IQueryable 135, 142, 210
 IQueryResult 219
 IsDirty 69
 IsDirty() 142
 Isolation 206
 IsReadOnly() 142
 IsRemoved() 142
 ItemDataBound 105
 ITransaction 192, 194

- K -

Key Generator 179
 Kulkarni 210

- L -

Language Integrated Query 210
 LINQ 47, 69, 210, 219
 LINQ 101 Samples 210
 LINQ Building Blocks 210
 ListChanged 105
 Log Level 54
 Lost Update 206

- M -

Make Persistent 182
 MasterGridViewTemplate 69
 Maximum Depth 234
 Maximum Objects Fetched 234
 Maximum Objects in Cache 244
 Maximum Queries in Cache 244
 Microsoft SQL Server Management Studio 182
 Missing partial modifier 222
 Mixed (flat and vertical) mapping 168
 MS SQL 47
 MSMQ 244
 Multiple Field 179
 multi-tier 103
 MustInherit 176

- N -

Next Fetch Group 234
 No Lost Updates 206
 Non-Repeatable Read 206

- O -

Object Database Management Systems 215
 Object Query Language 215
 ObjectContainer 105, 142
 ObjectProvider 69, 95
 Objects 194
 ObjectScope 69, 142, 234
 ObjectScope() 28
 ObjectScopeProvider 37, 135
 ObjectView 69, 95
 OpenAccess Data Form Wizard 54
 OpenAccess Dataform Wizard 65
 OpenAccess Enable Project Wizard 22
 OpenAccess Programmers Guide 210
 OpenAccessDataSource 86
 OpenAccessTracer 234, 244
 OptimisticVerificationException 200
 OQL 47, 215, 219
 OQL Query Browser... 215
 Oracle 47
 ORM Enable Project 28, 37, 54
 ORM-enable 68, 135

- P -

partial class 44
 PartialUserDefault.vm 44
 Path 234
 Persistent 22, 28, 54, 68, 69, 179, 222
 Persistent classes 222, 244
 Phantom Read 206
 pmCacheRefType 234
 Presentation Layer 103
 Project Properties 54
 projection 69

- Q -

Query<T> 219

- R -

RadAjaxManager 86
 RadComboBox 69, 244
 RadForm 69
 RadListBox 200
 RadMessageBox 69
 Read Committed 206
 READ_COMMITTED 206
 READ_UNCOMMITTED 206
 Reflector 54
 RefreshReadObjectsInNewTransaction 195
 Remove 244
 Remove() 47
 RemoveItem() 105
 RemovingItemEventArgs 105
 Repeatable Read 206
 REPEATABLE_READ 206
 Report 95
 ReportParameters 95
 Reset() 244
 Reverse Engineering Wizard 37, 68, 244
 Reverse Mapping 37, 68, 69, 244
 Reverse Mapping Wizard 154, 222
 Rollback() 69, 192

- S -

SelectedIndexChanged 69, 105, 244
 Self referencing 161
 SERIALIZABLE 206
 Server Explorer 28, 182
 Service Layer 103
 Single Field 179
 SQL Server Management Studio 28, 195, 222
 Stored Procedure 222
 Stored Procedure Mapping 222
 Stored Procedures 222
 STRONG 234
 SubReport 95
 System.Collections.Generic 105, 142
 System.ComponentModel 105
 System.Linq 105, 142, 210
 System.Reflection 105, 244
 System.Runtime.Serializable 105
 System.Runtime.Serialization 142

System.ServiceModel 105
System.Linq 142

- T -

Telerik ORM Wizard 54
Telerik.OpenAccess 22, 54, 69, 86, 105, 182, 244
Telerik.OpenAccess.Query 22, 69, 86, 210
Telerik.WinFormsControls 69
Telerik.WinFormsControls.UI 69, 105, 244
Templates 44
ThemeName 69
Thread 200
ToList() 210
Transaction 28, 69, 192, 195
TransactionProperties 69, 195

- U -

UPDATE 215
Update Config References 69, 86, 182
Update Database 54, 222
Using LINQ with OpenAccess ORM 210

- V -

VEnhance 54
Verify 179
Vertical mapping 168

- W -

WCF 105
Weak 234
Windows Communication Foundation 105
Write Log Output to Console 54