Telerik
OpenAccess ORM

Quick Start Guide for Telerik OpenAccess ORM

Contents

Overview

Getting started with Telerik OpenAccess ORM is quick and easy. Following the **steps** below will ensure it is a smooth process.

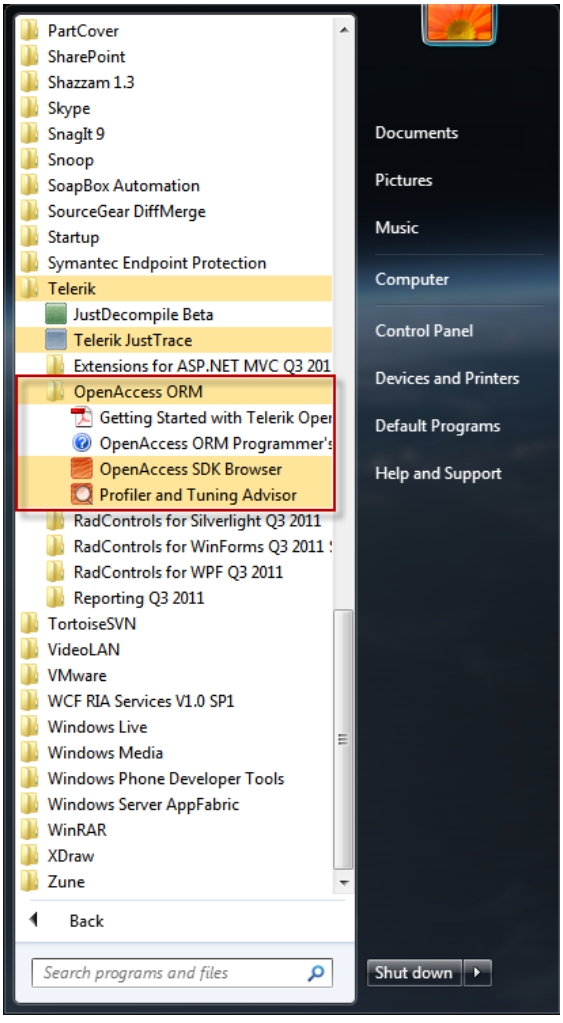
1. Install Telerik OpenAccess ORM
2. Build a Data Model
3. Connect the Model to UI

Product Installation

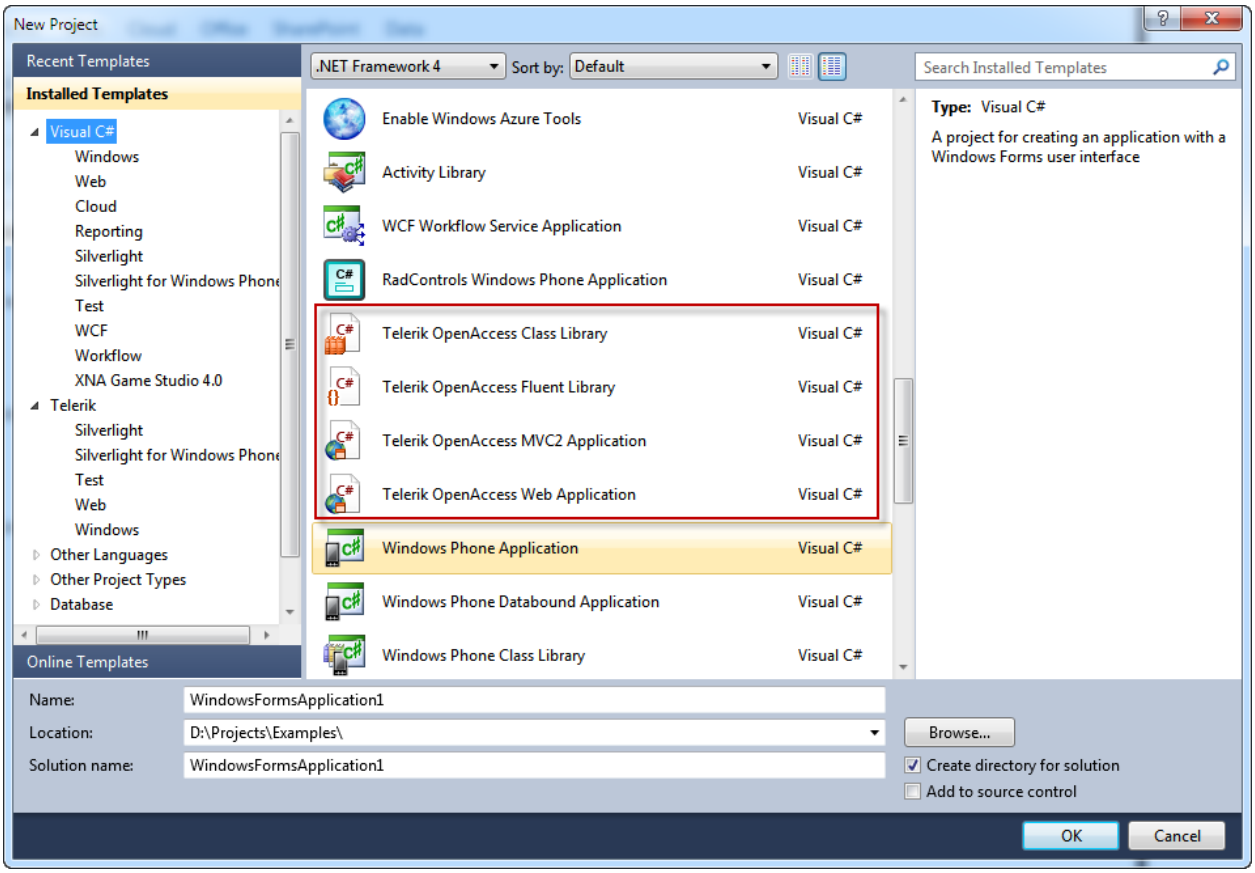
The Telerik installer can be downloaded through your account on Telerik.com or from the OpenAccess ORM page (trial version only).

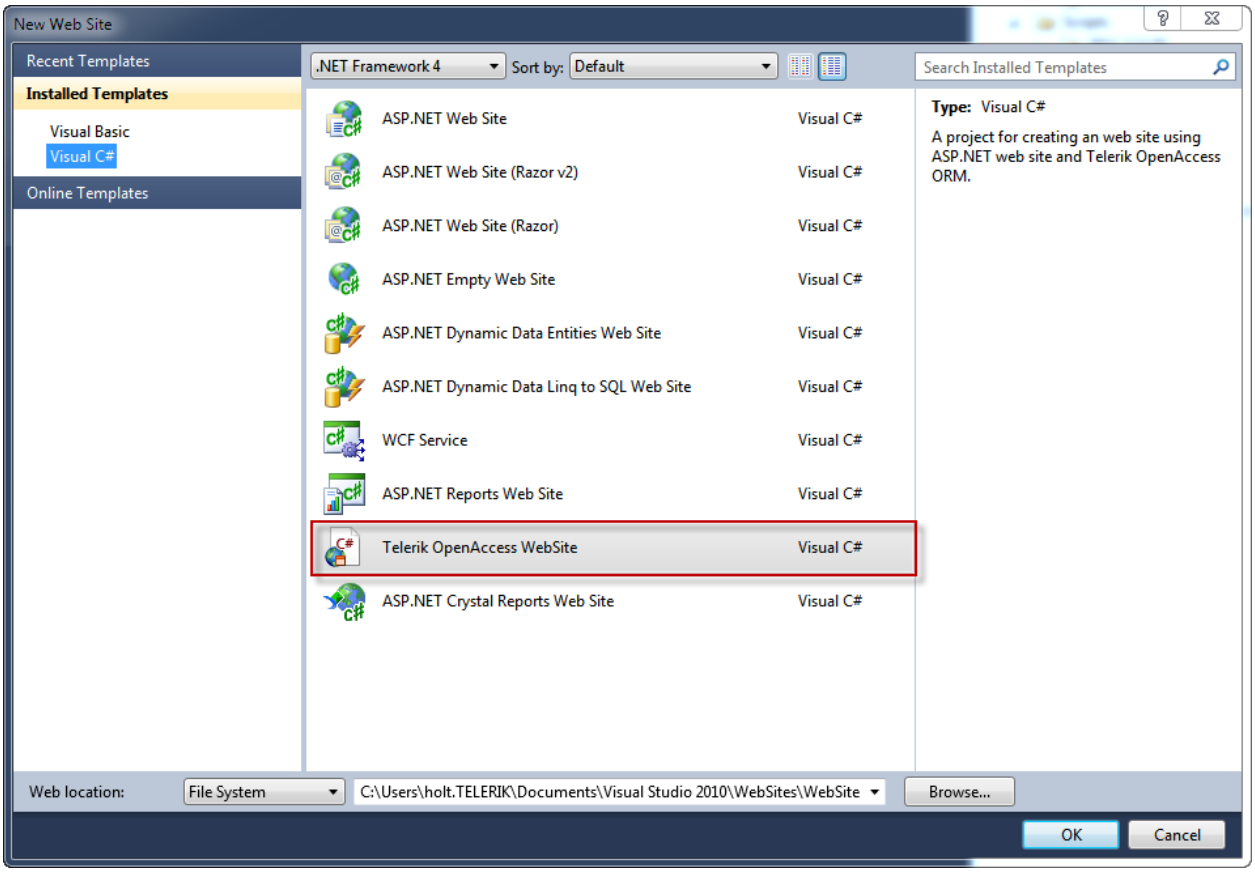
Once the download completes, unzip the package, and run the installer. The installer will walk you through the installation steps.

The installer will create an OpenAccess Folder under the Telerik folder in the start menu. In this folder there it will add a shortcut to launch the Profiler and Tuning Advisor, as well as a shortcut to the complete documentation and to this document.



The installer will also add new project templates to Visual Studio:





What Do These Templates Do?

Telerik OpenAccess Class – Creates a new class library project with an OpenAccess domain model built using the visual designer and forward or reverse mapping.

Telerik OpenAccess Fluent Library - Creates a new project using the OpenAccess ORM Fluent Mapping API.

Telerik OpenAccess MVC2 Application – Creates a new solution with an MVC 2 web project, and a Telerik OpenAccess Fluent Library project.

Telerik OpenAccess Web Application – Creates a new ASP.Net AJAX web project, with a Telerik OpenAccess Domain model built using the visual designer and forward or reverse mapping.

Next Steps…

Now that OpenAccess is installed, the next step is to build a Domain Model

Building a Domain Model

Telerik OpenAccess ORM makes it very simple to start working with data. The first choice you need to make is how you would like to configure the mapping:

- If you have an existing database, and would like to create entities from the tables, views, or stored procedures, Database-First (Reverse) Mapping will be the best path.
- If you do not have an existing database, and would like to build your classes first, and have the database generated automatically, then Model-First (Forward) Mapping will be the best path for you.
- If you would like complete control over the code in your entities, databases updates, and data mapping, then the Fluent Mapping API will be the best path for you.

Database-First (Reverse) Mapping

OpenAccess ORM reverse mapping allows developers to map an existing database schema to .Net objects, which can be utilized by any .Net application.

> **Note**
> For this section the guide will use a MS SQL database named **OpenAccessQuickStartDB**. To create this example database run the following script:

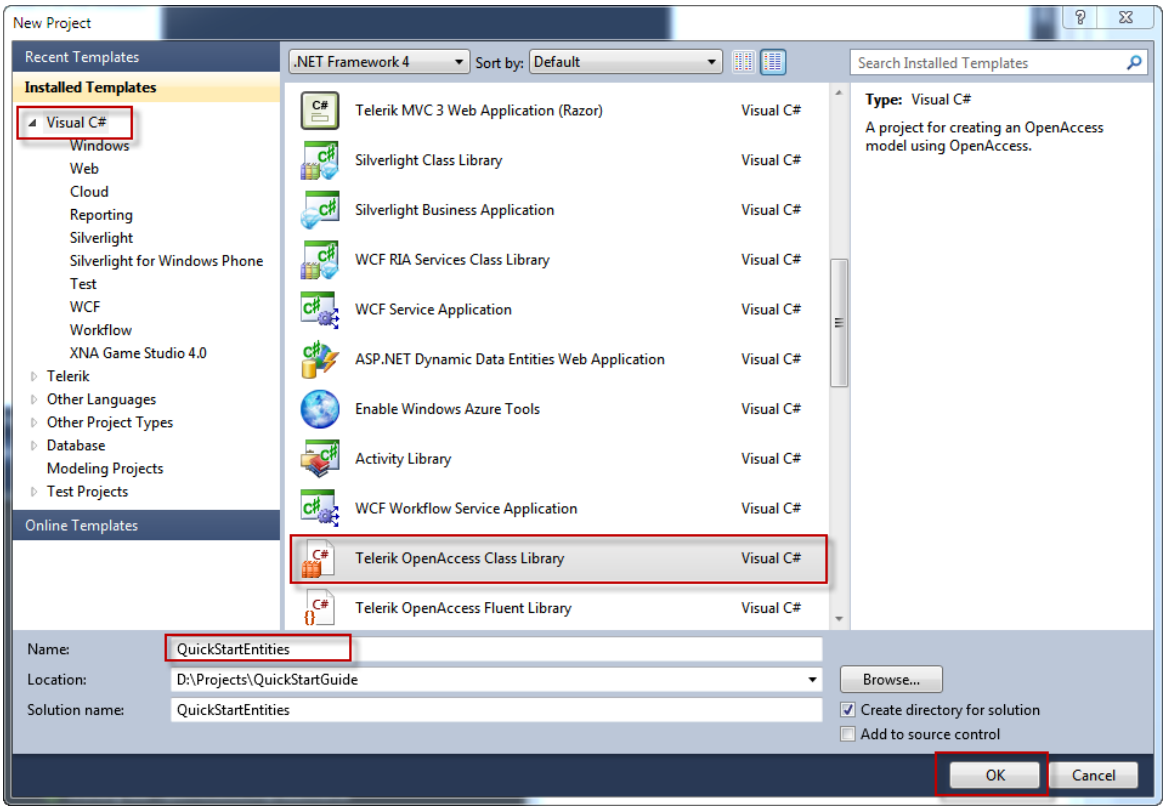
```
USE [master]
GO
CREATE DATABASE [OpenAccessQuickStartDB] ON  PRIMARY
( NAME = N'OpenAccessQuickStartDB', FILENAME = N'C:\Program
Files\Microsoft SQL
Server\MSSQL10.SQLEXPRESS\MSSQL\DATA\OpenAccessQuickStartDB.mdf' , SIZE
= 2048KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
 LOG ON
```

```
( NAME = N'OpenAccessQuickStartDB_log', FILENAME = N'C:\Program
Files\Microsoft SQL
Server\MSSQL10.SQLEXPRESS\MSSQL\DATA\OpenAccessQuickStartDB_log.ldf' ,
SIZE = 1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO
USE [OpenAccessQuickStartDB]
GO
CREATE TABLE [dbo].[Customer](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [Name] [varchar](255) NULL,
      [DateCreated] [datetime] NULL,
      [EmailAddress] [varchar](255) NULL,
 CONSTRAINT [PK_Customer] PRIMARY KEY ([Id]))
GO
```

Creating the Project

To create a new Telerik OpenAccess domain model using reverse mapping we will need to create a project first:

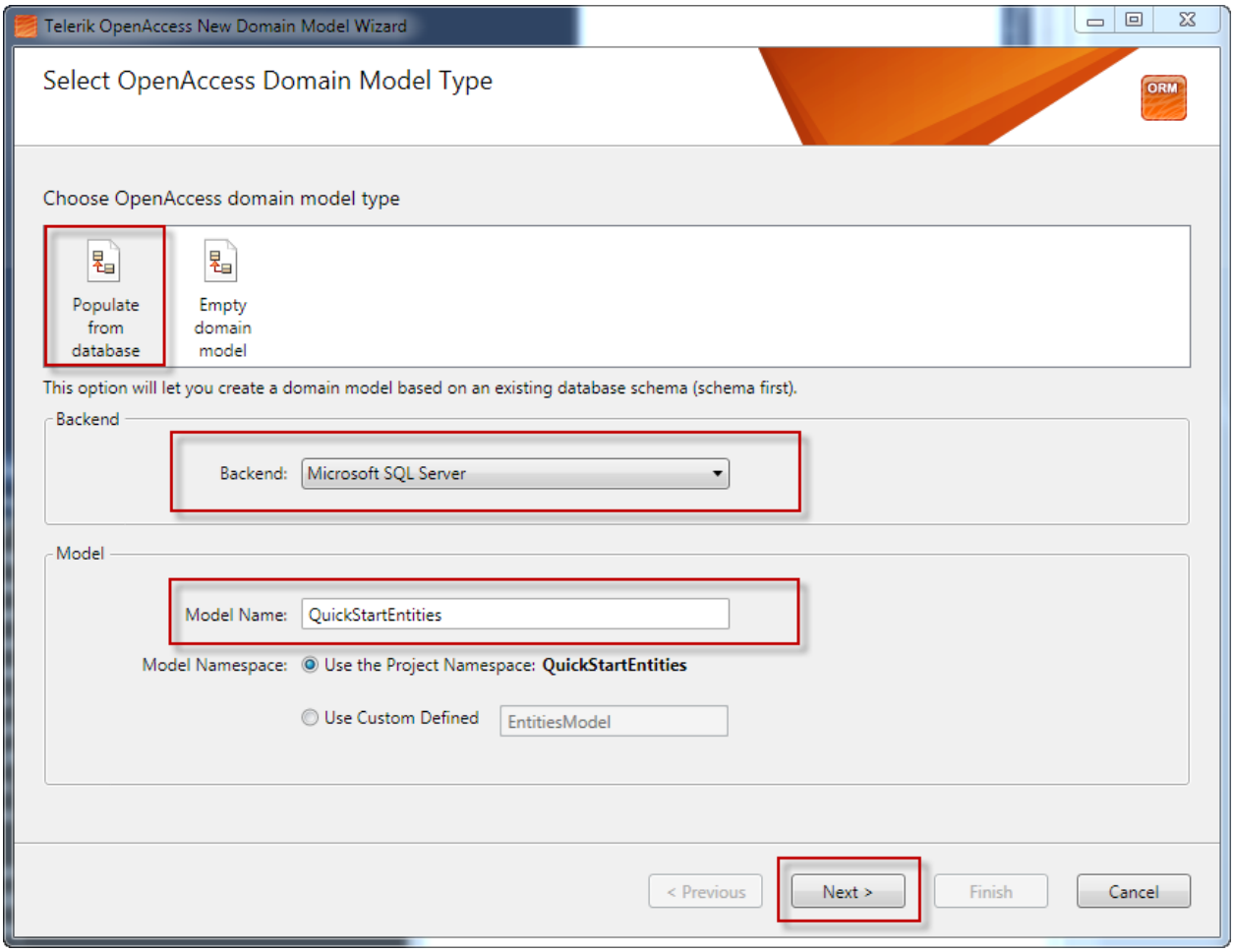
1. Select **File** > **New Project** in Visual Studio
2. Select **Visual C#** / **Visual Basic** in the **Installed Template** tree.
3. Then select **Telerik OpenAccess Class Library**. Name the project **QuickStartEntities**, and then click **OK.**



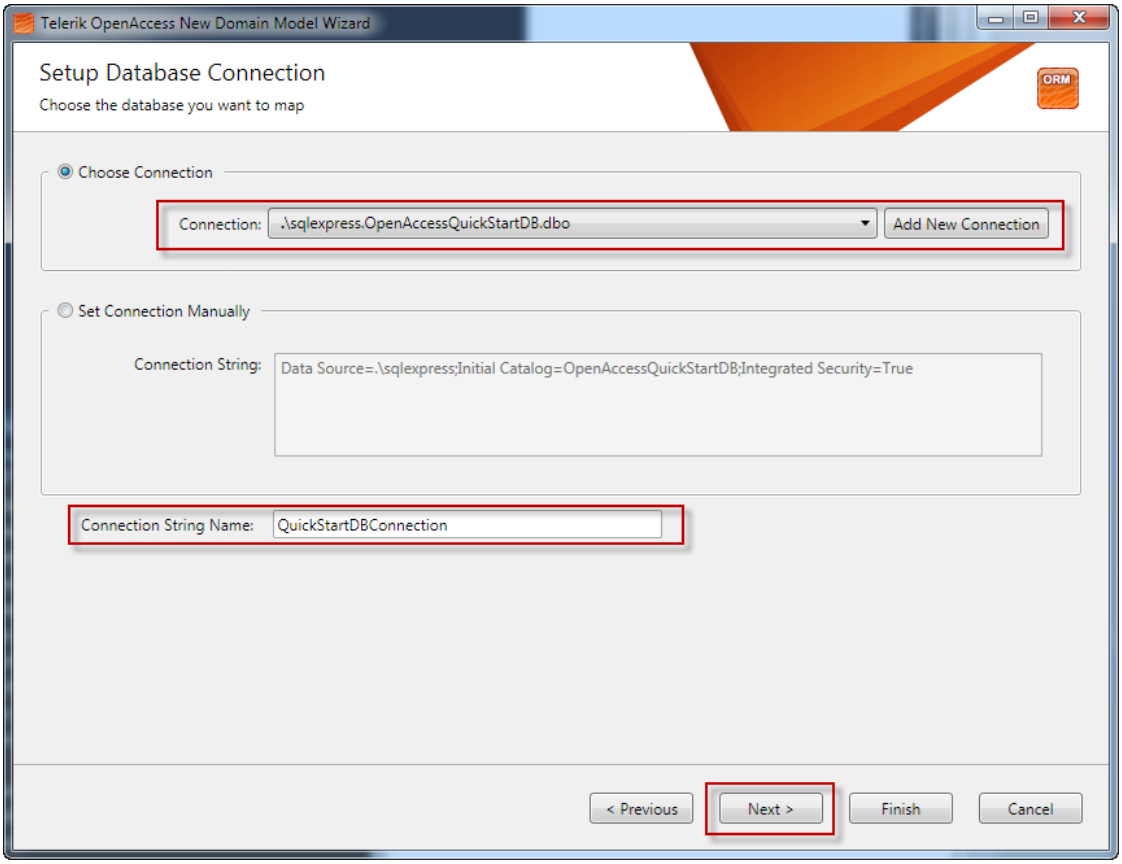
This will envoke the New Domain Model Wizard for

Creating Entities From the Database Schema

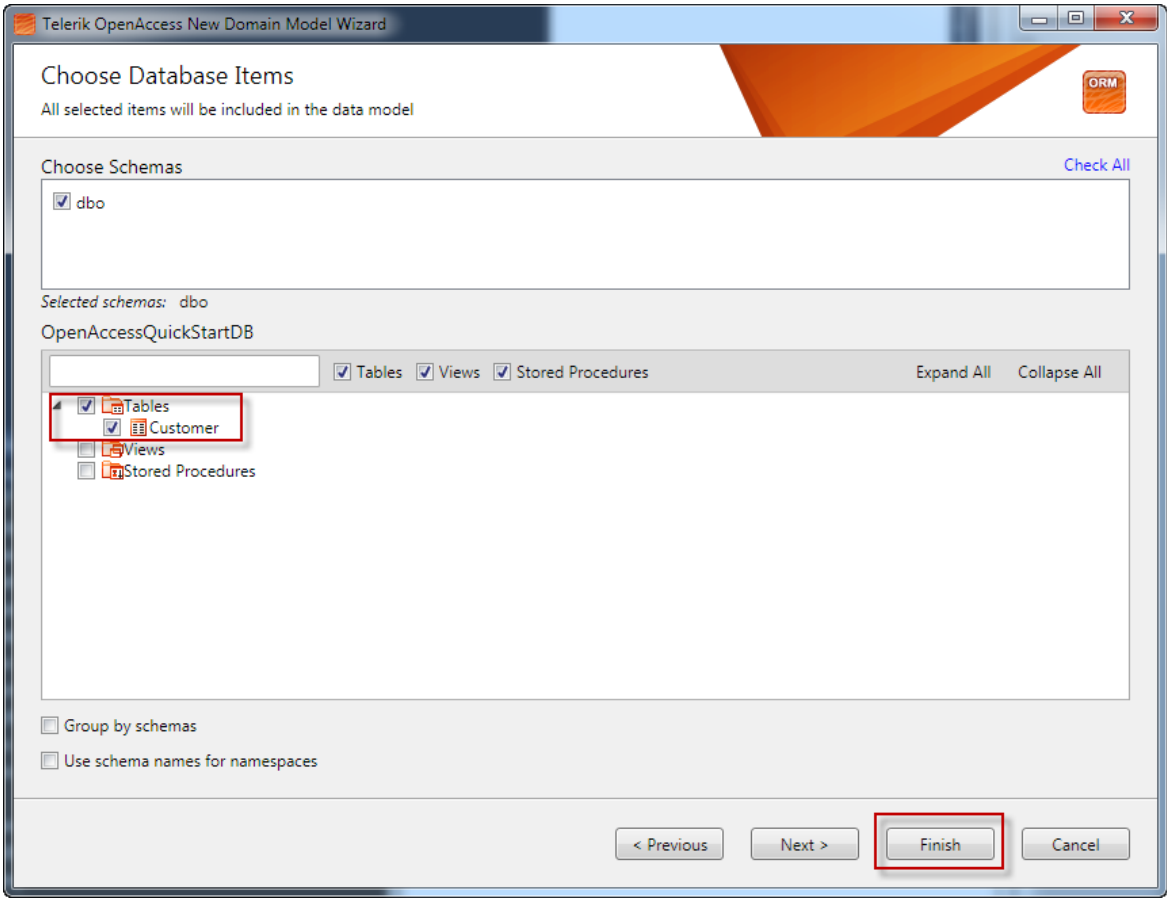
1. In the **Telerik OpenAccess New Domain Model Wizard** select **Populate from Database.** Then select the target database, name the model **QuickStartEntities**, and click **Next**.



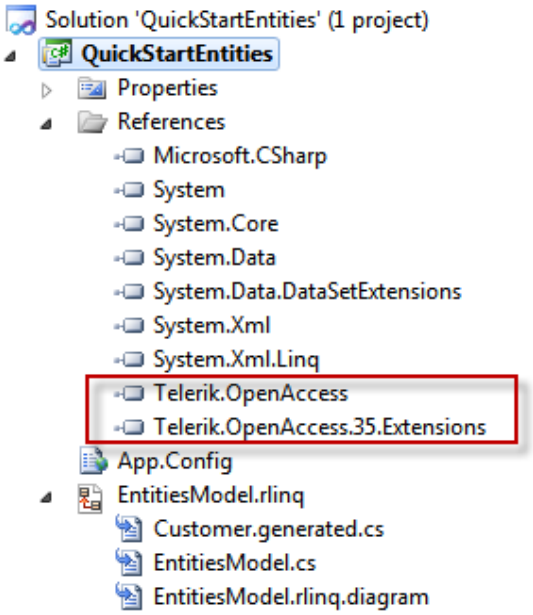
2. **Add** or **select** an existing connection, name the connection string **QuickStartDBConnection**, and then click **Next**.



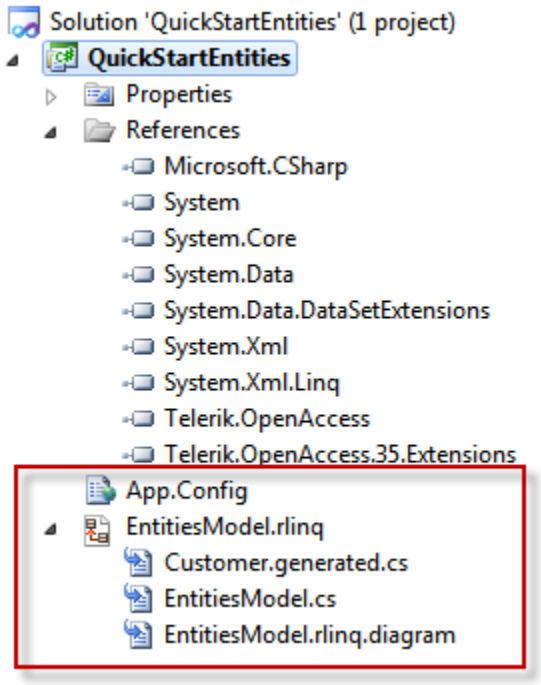
3. In the next screen, select the **tables, views,** and **stored procedures** to import into the domain model, for this guide the **Customer** table will be imported. Next, Click **Finish.**



When you click **Finish** , Telerik OpenAccess ORM will add two references to your project:



It will also add an app.config, which stores the connection string, and an .rlinq file, which stores information about the mapping.



Next Steps…

At this point the model is ready to be used in an application, or run the Data Service Wizard to create a service layer over the model.

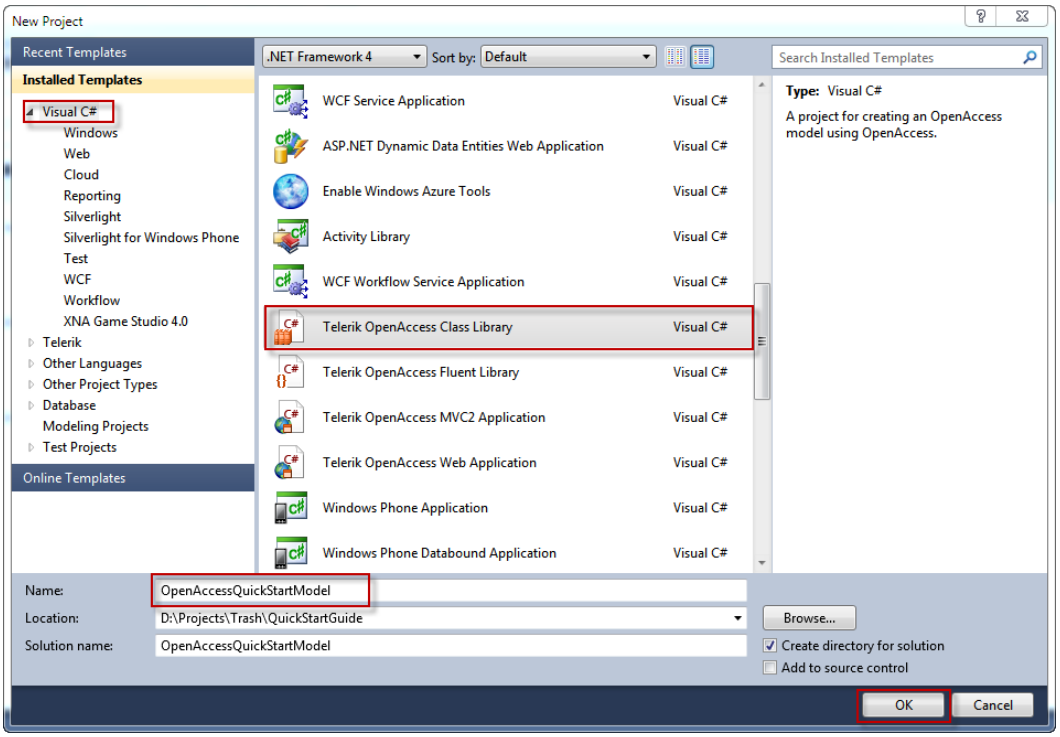
Model-First (Forward) Mapping

OpenAccess ORM forward mapping allows developers to concentrate on building classes, and will automatically generate a database based on those classes.

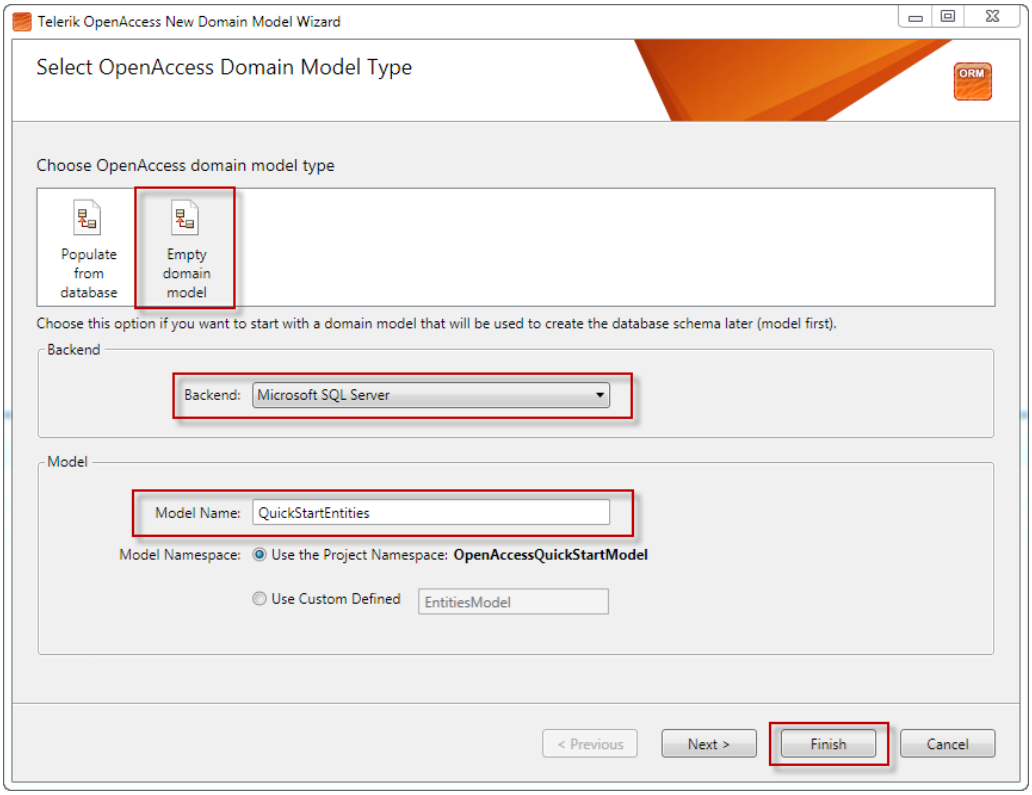
Creating the Project

To create a new Telerik OpenAccess domain model using forward mapping:

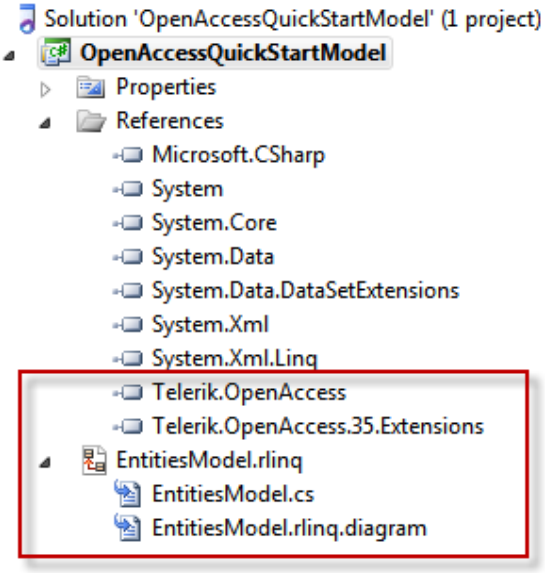
1. Select **File > New Project** in Visual Studio
2. Select Visual C# or Visual Basic in the **Installed Template** tree.
3. Then select **Telerik OpenAccess Class Library**. Name the project **OpenAccessQuickStartModel**, and then click **OK.**



4. In the **Telerik OpenAccess New Domain Model Wizard** select **Empty Domain Model.** Then select the target database, name the model **QuickStartEntities**, and click **Finish**.



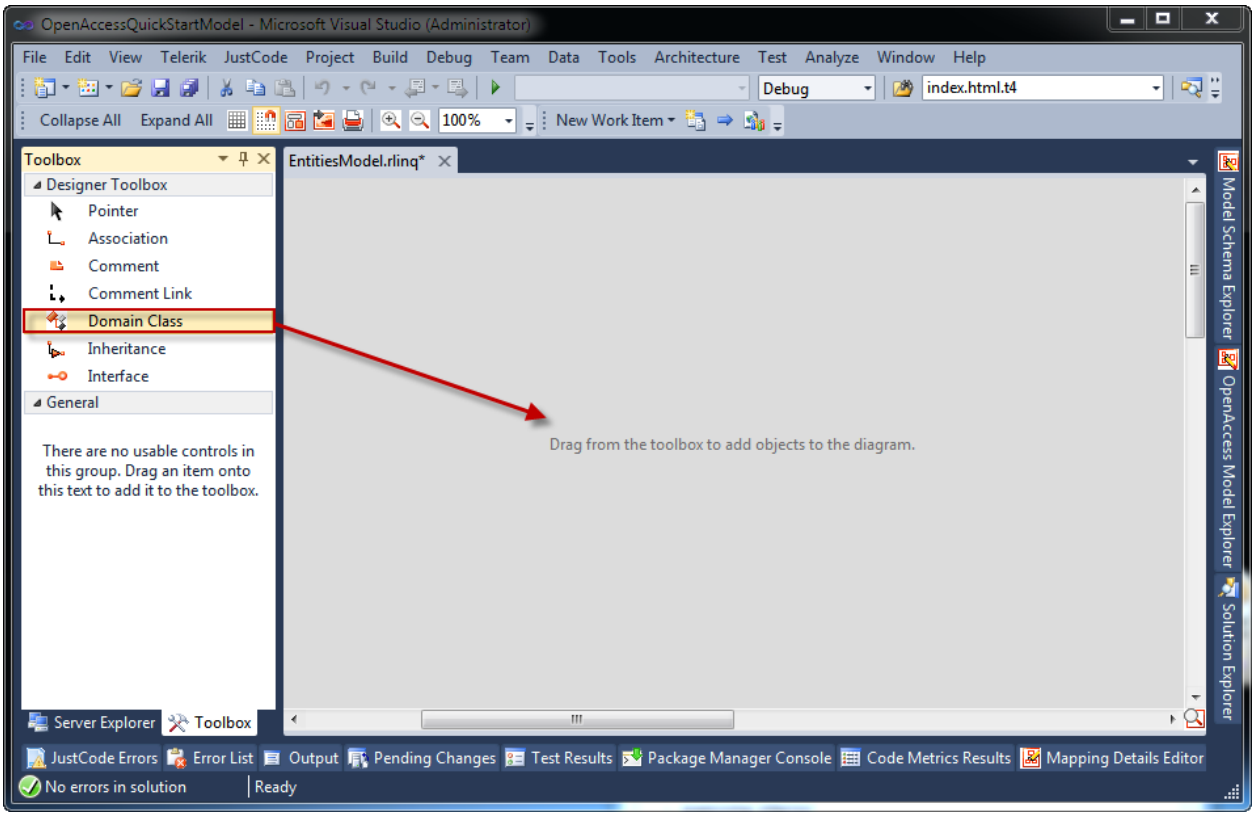
When you click finish, the wizard will enhance the project to work with OpenAccess. In addition, it will add the required references, and an .rlinq file to the project.



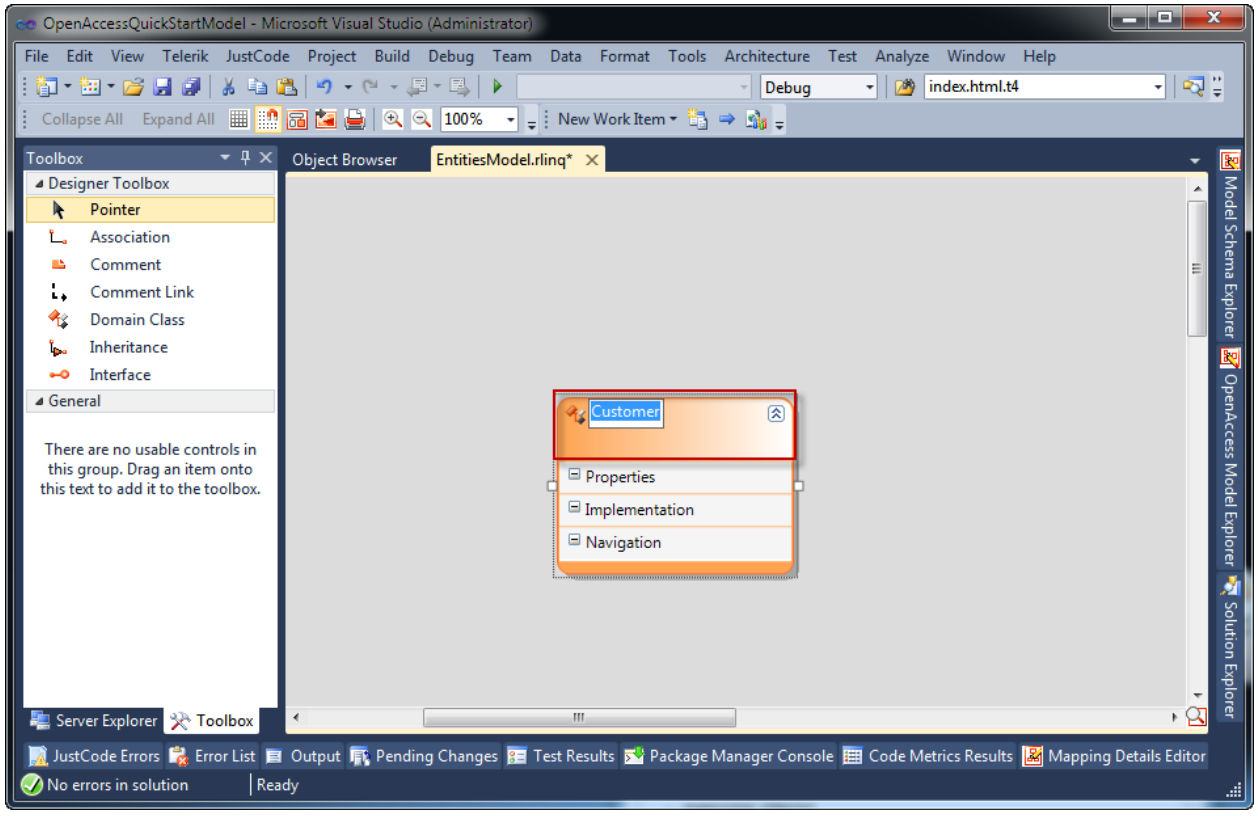
Creating an Entity

OpenAccess provides new toolbox items to make model creation a simple drag and drop operation.

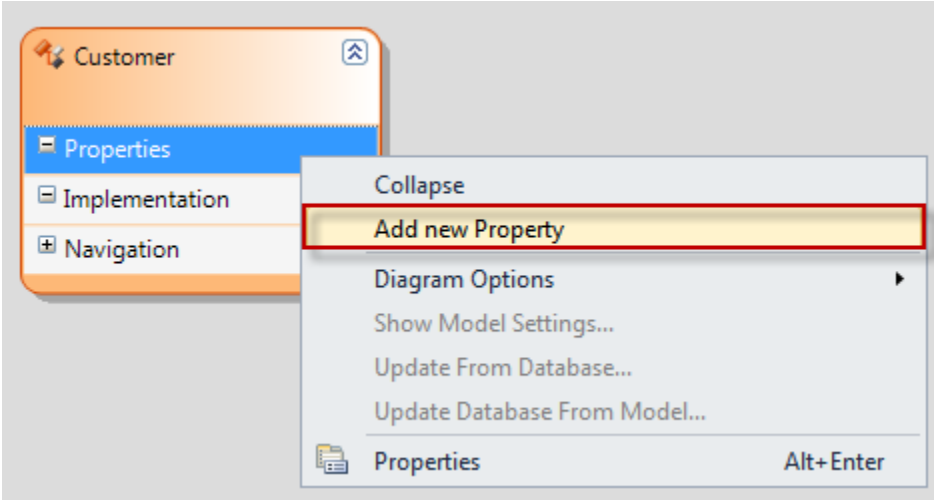
1. Drag a new **Domain Class** out of the toolbox, and **drop** it on the design surface.



2. **Double click** on the entity title, which will enable you to edit the entity's name. Name the entity **Customer**.



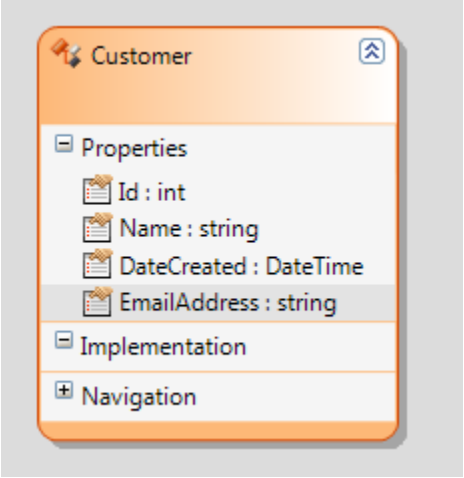
3. Add properties to an entity by **right clicking** on the **properties** section, and selecting **Add new Property**.



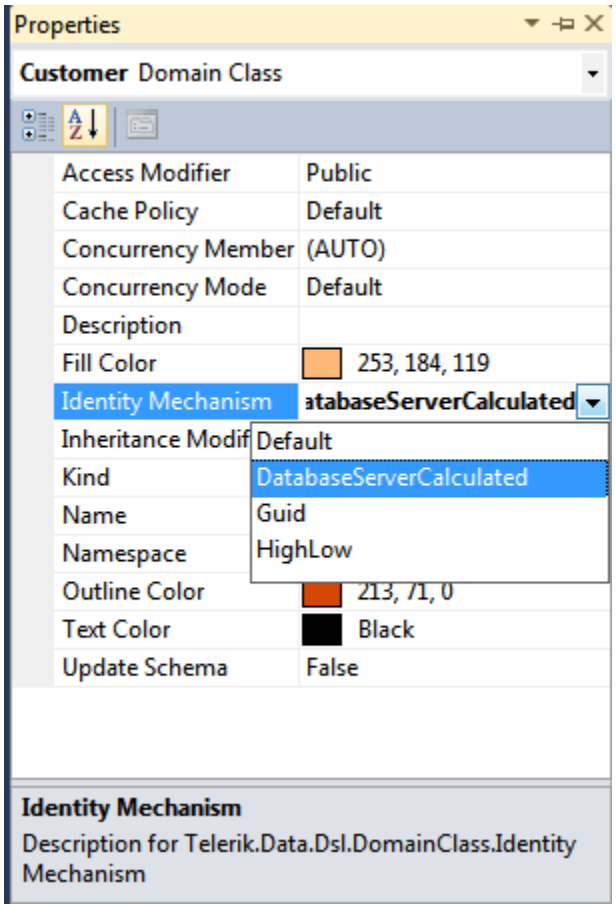
Add the following Properties to the **Customer** entity**:**

- Id : int
- Name : string
- DateCreated : DateTime
- EmailAddress : string

The entity should now look like this:

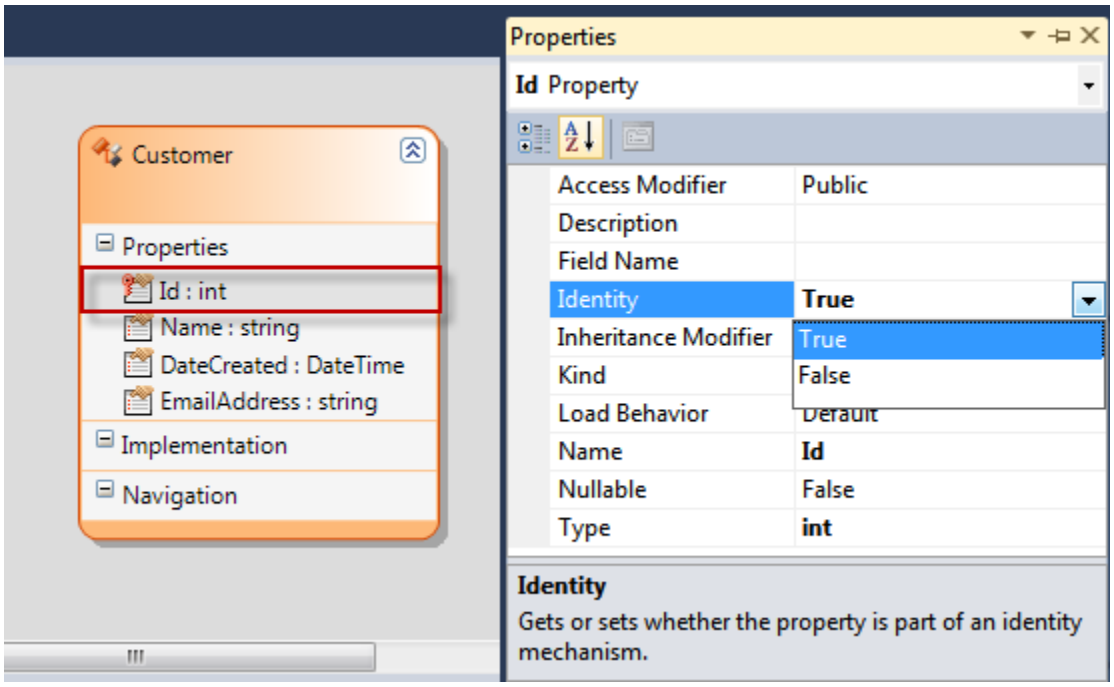


4. Next we need to configure the primary key, and set which property will be used as the identity mechanism.
5. With the **Customer** entity selected, expand the Properties window.
6. In the Properties window, **Select** the **Identity Mechanism** dropdown, and choose **DatabaseServerCalculated**. This tells OpenAccess that the value of the Entities identity field will set by the database server.

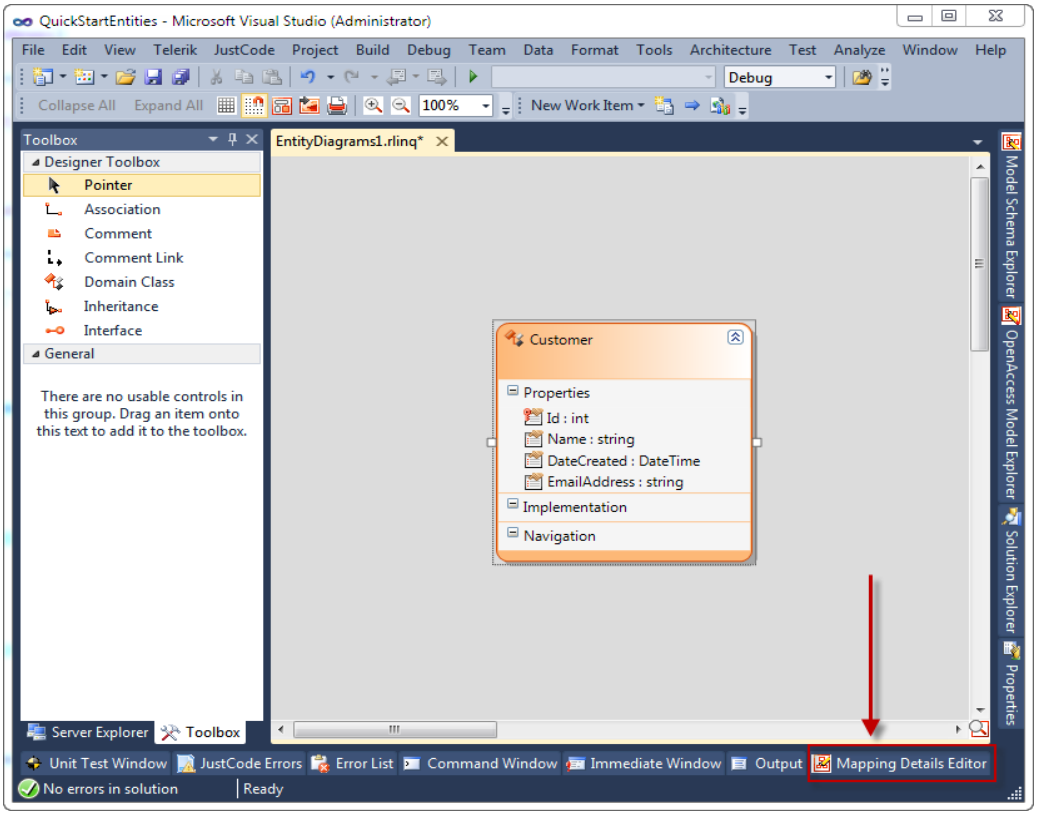


For more information about the other Identity Mechanisms supported by OpenAccess please read this documentation article.

7. Now we need to specify which property will be used as the entiti's Identity. To do this, **select** the **Id** property of the **Customer** entity. Expand the **Properties** window, and set the **Identity** property to **True**.

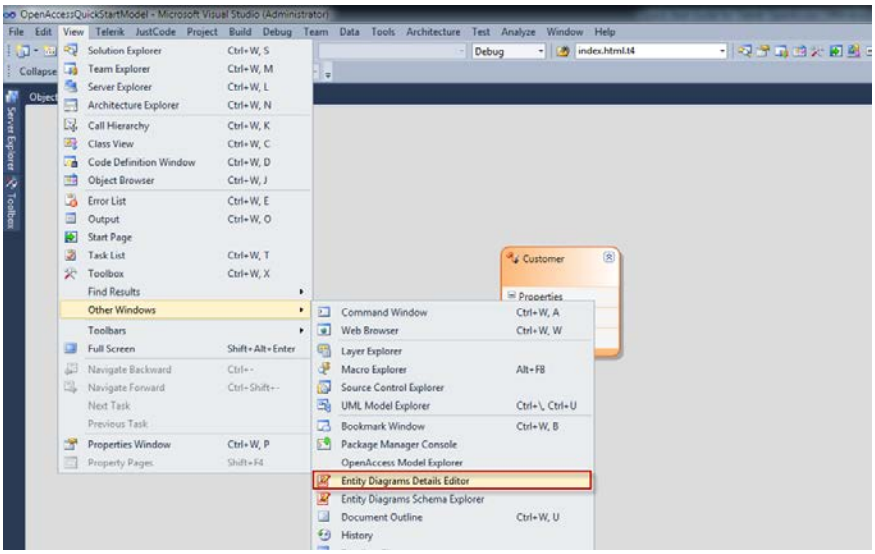


8. The last step is to map the **Customer** entity to a table in the **database**. OpenAccess provides a mechanism to automatically create this mapping. To use this feature, expand the **Mapping Details Editor**.

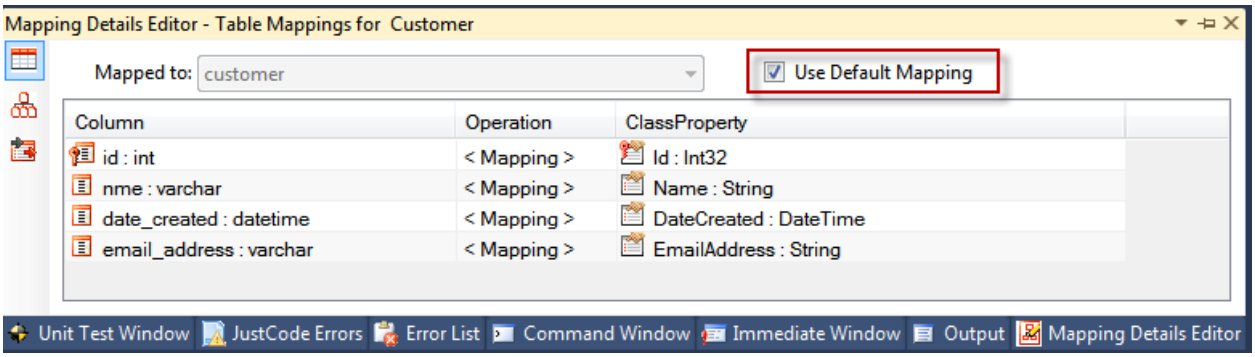


Note

If the mapping details tab is not visible as indicated in the screenshot, access through the Visual Studio Menu, by navigating to: **View>Other Windows>Model Details Explorer**.



9. With the Mapping Details Editor expanded, check **Use Default Mapping**.



What Just Happened?

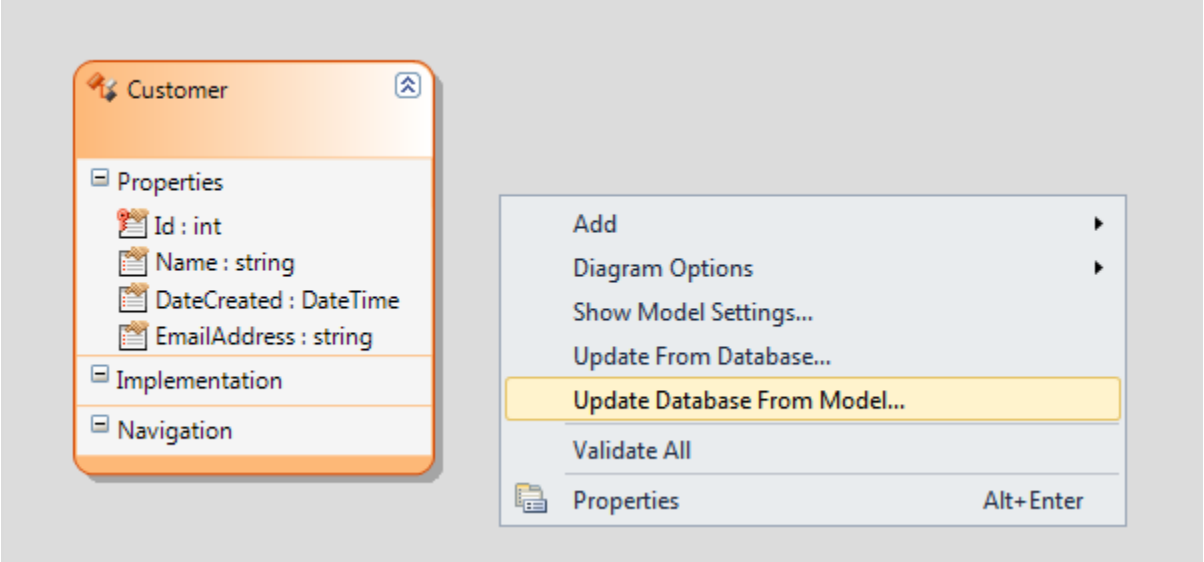
When using the **Use Default Mapping** option, OpenAccess will automatically build a database table schema for an entity based on its properties, and their data types. To customize this mapping please read this documentation article.

With the domain model built, the next step is to create the database the domain model will use to save information.

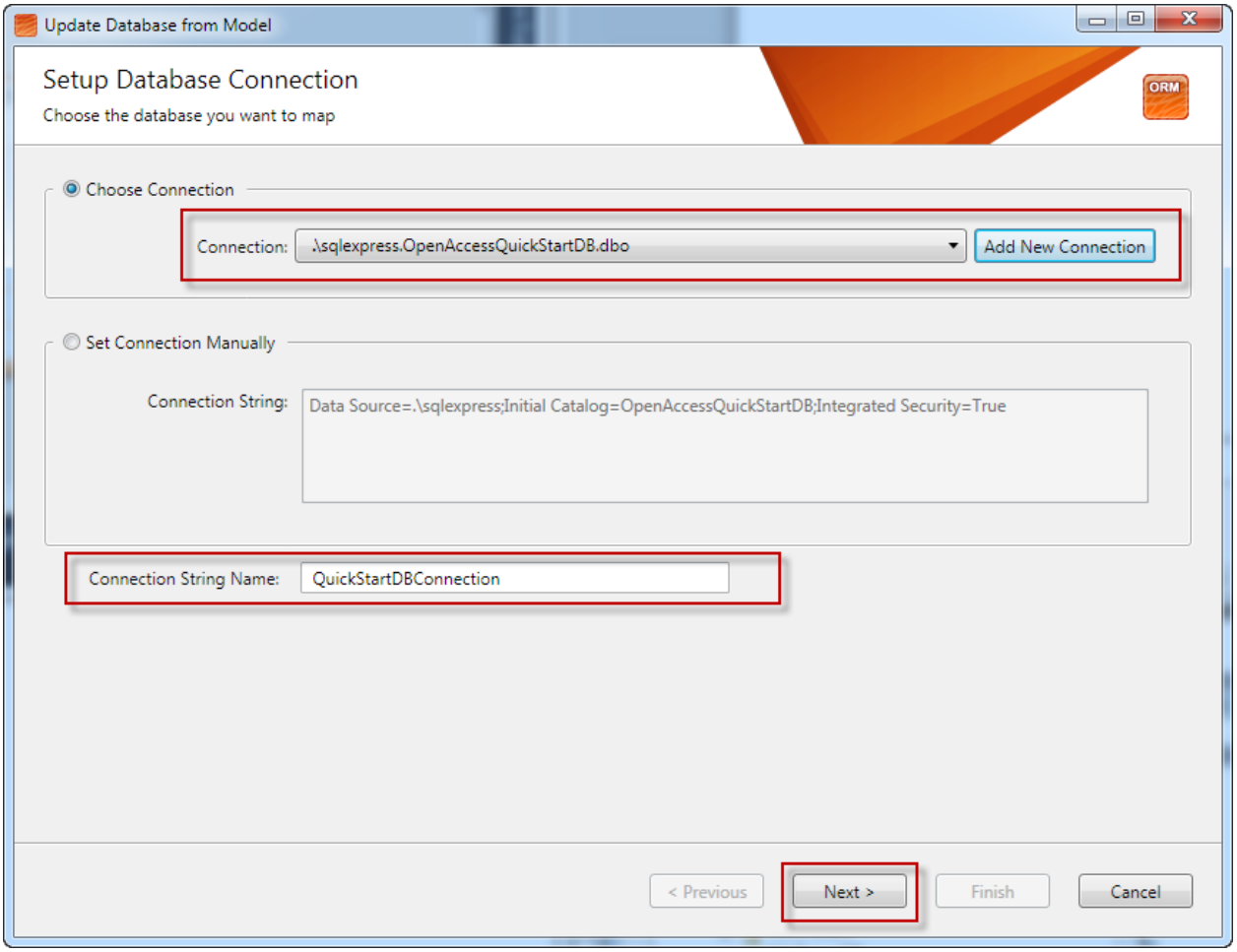
Creating the Database

OpenAccess will generate database creation, or migration, scripts based on the changes made in the designer. In addition, it can execute these scripts, making database creation, and migration very simple.

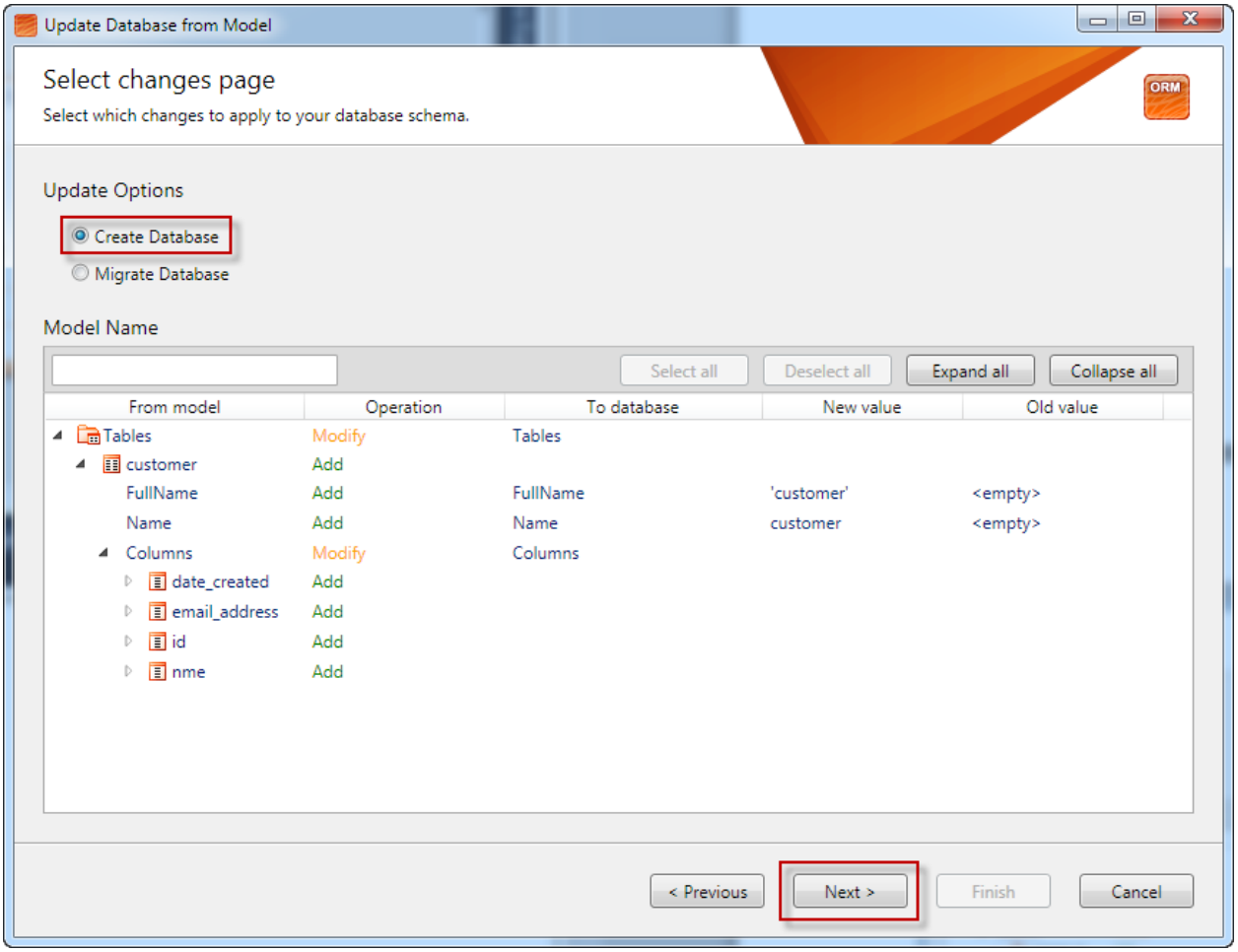
1. **Save** the OpenAccess Domain model by pressing **Ctrl+s**.
2. **Right click** on the design surface, and select **Update Database from Model**.



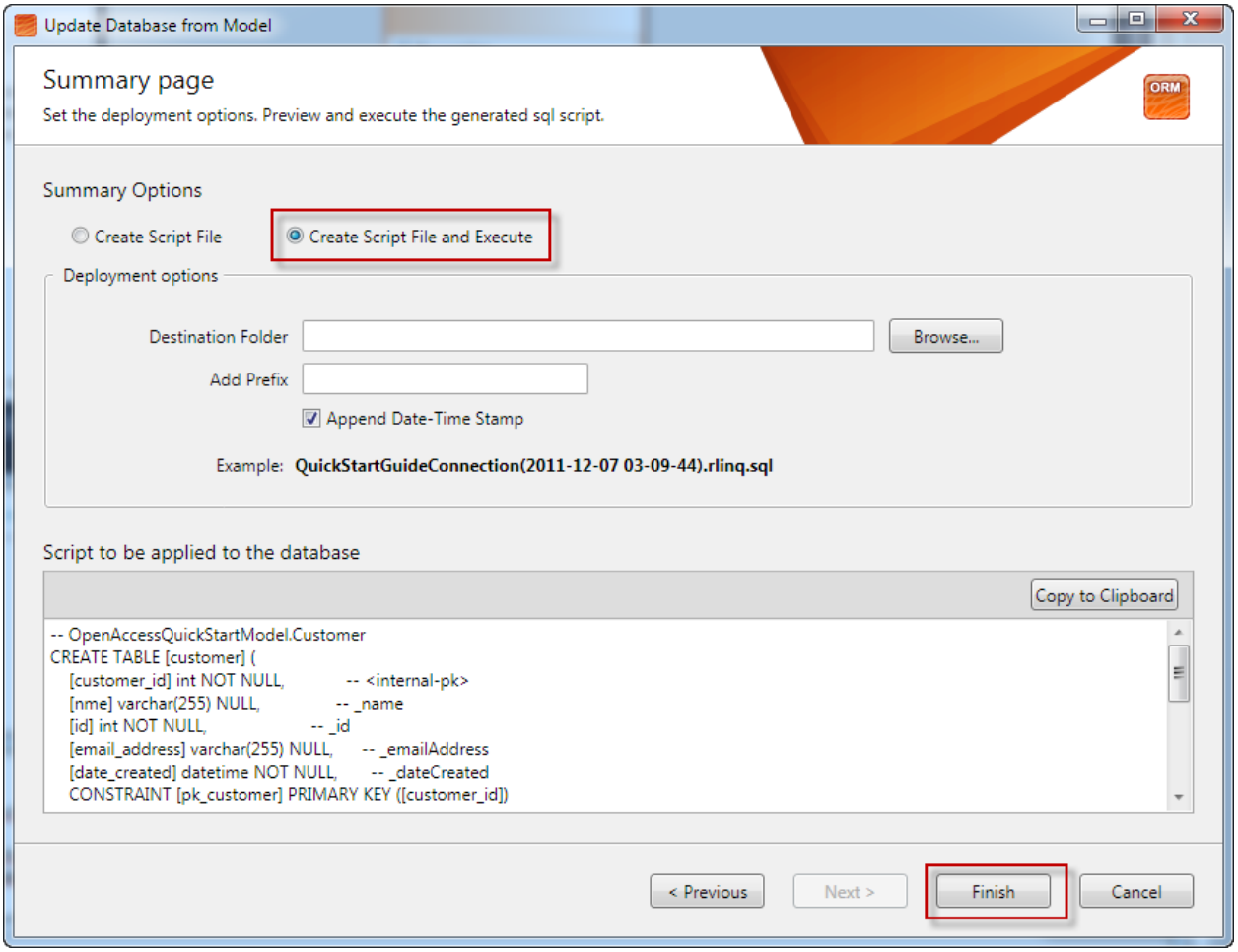
3. In the wizard, **Add** or **select** an existing database connection. Name the connectionstring **QuickStartDBConnection**, and then click **Next**.



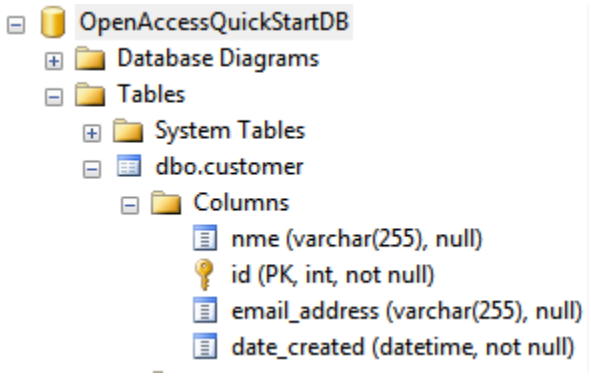
4. Ensure **Create Database** is the selected **Update Option**, and click **Next**.



5. Select **Create Script File and Execute**, and click **Finish**.



6. After clicking finish, OpenAccess will save a copy of the script, and execute it against the database.
7. The database should look like this:



Next Steps…

At this point the database, and model are ready to be used in an application., or run the Data Service Wizard to create a service layer over the model.

Updaing the Model and/or the Database (aka Round-trip Mapping)

Round trip mapping is built into the OpenAccess Visual Designer. It allows developers to make changes in the database schema, and pull those into their domain model, or update their

domain model, and push those changes to the database schema. To leverage round trip mapping, start with either **Forward Mapping**, or **Reverse Mapping**. Once you have a model in place, you are free to push and pull changes as needed.

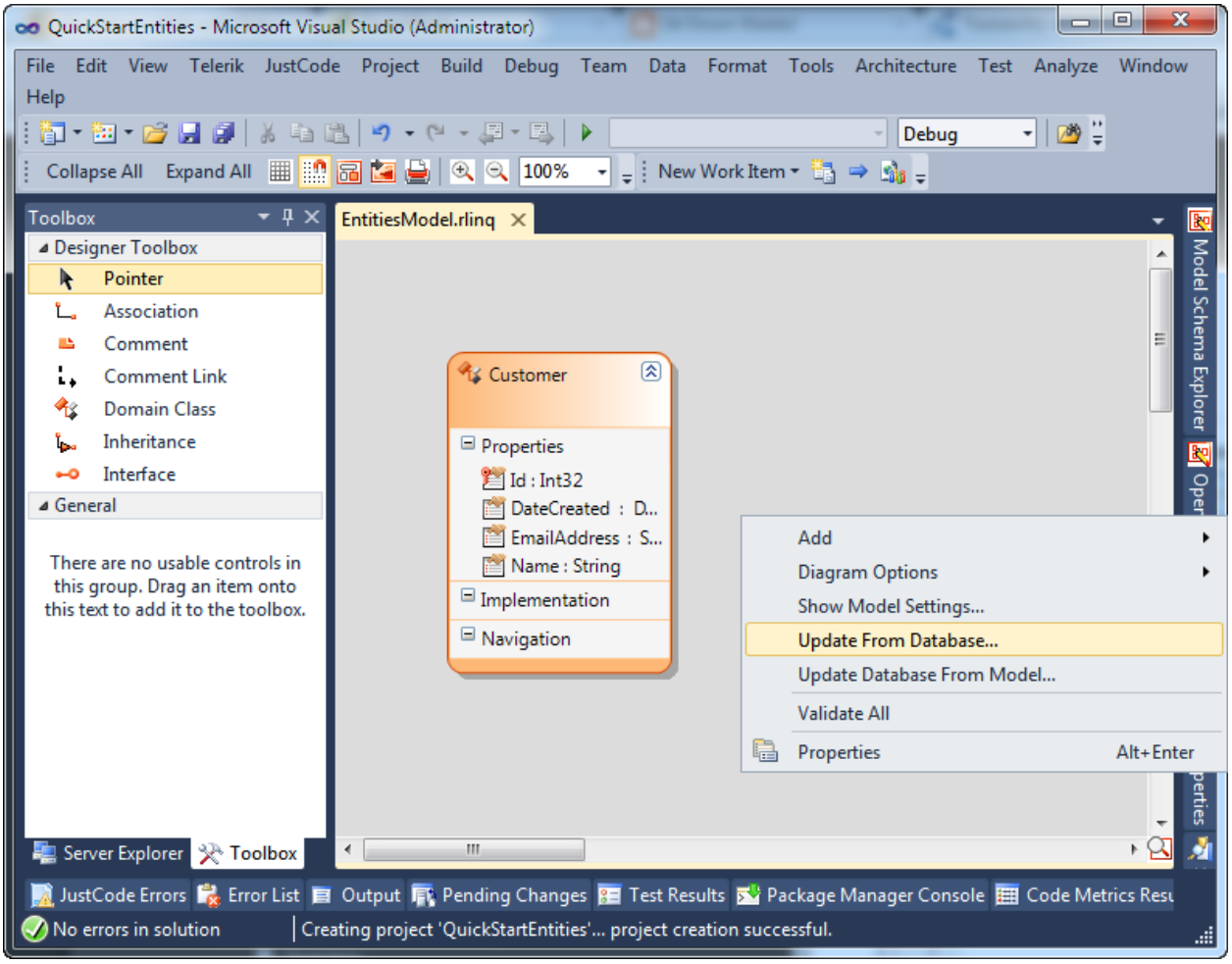
Pulling in Database Schema Changes

Note

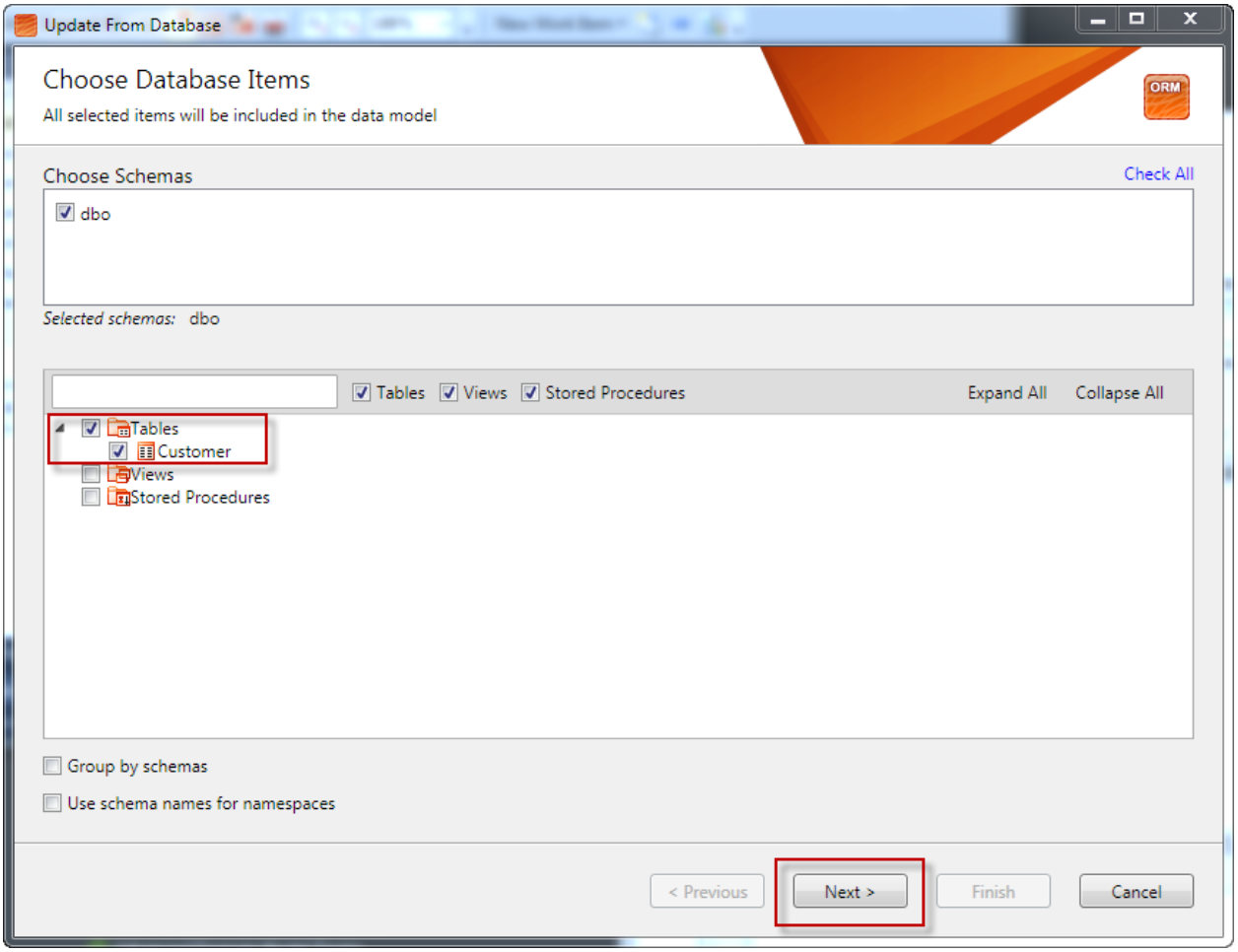
For this example, a column named DOB was added to the Customer table. Use the following script to create the new column in the database:

```
ALTER TABLE dbo.Customer ADD
      DOB datetime NULL
GO
```

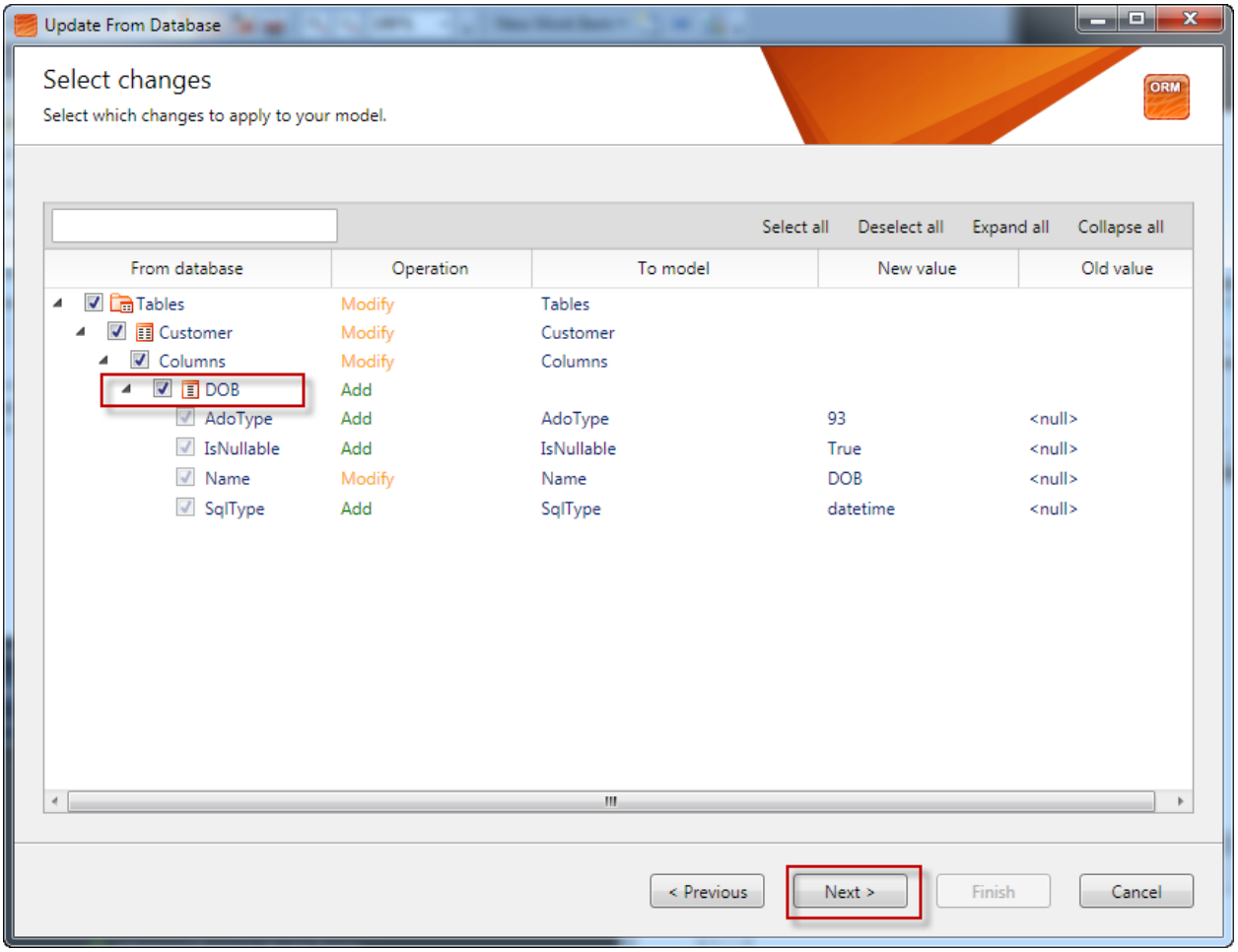
1. **Right click** on the visual designer surface, and click **Update from Database.**



2. In the first screen, **check** each table, view, or stored procedure that should be imported or updated in the domain model, and click **Next**. In this example the **Customer** entity will be updated.

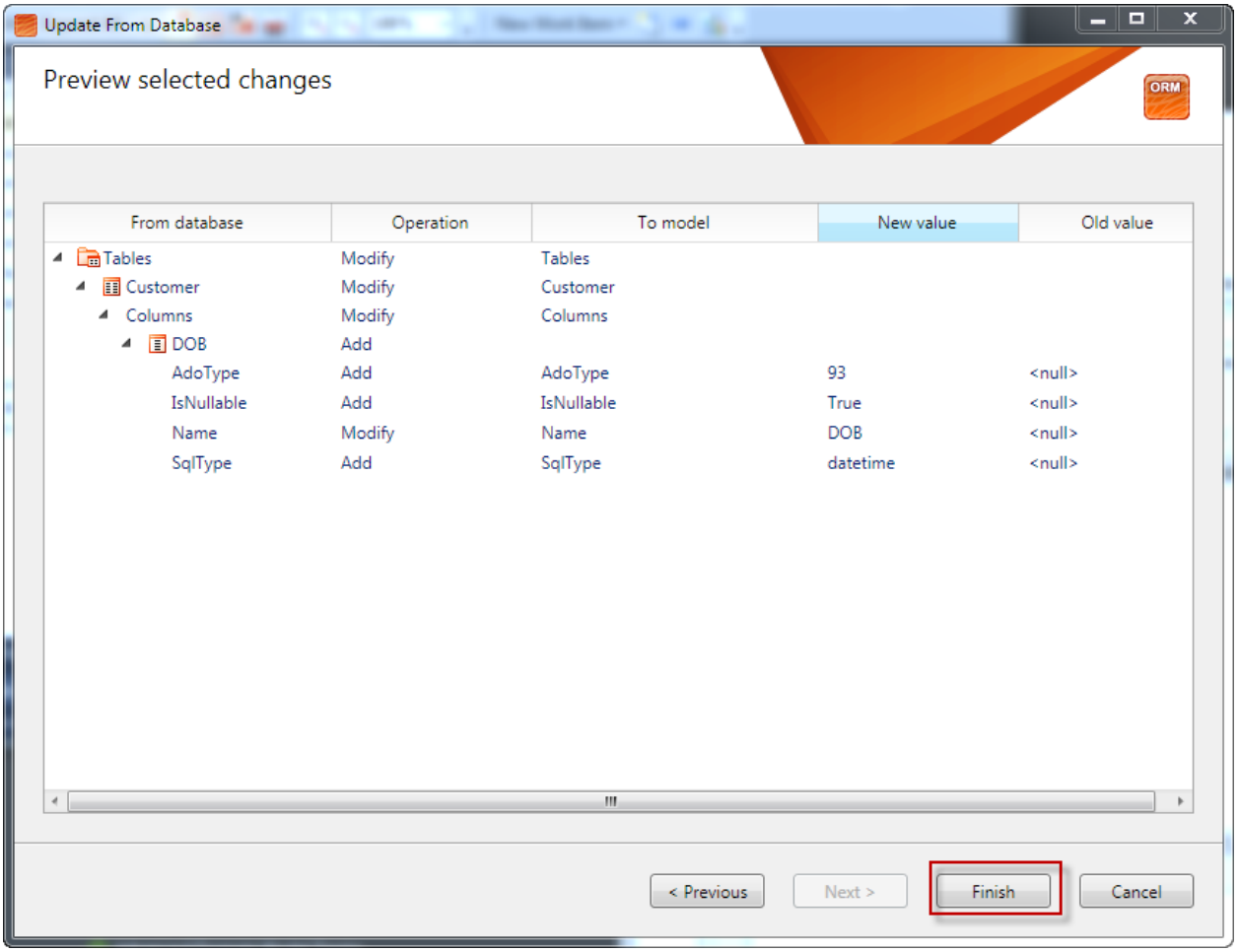


3. On the next screen, **select** which changes you would like to **apply** to the domain model, and click **Next**.

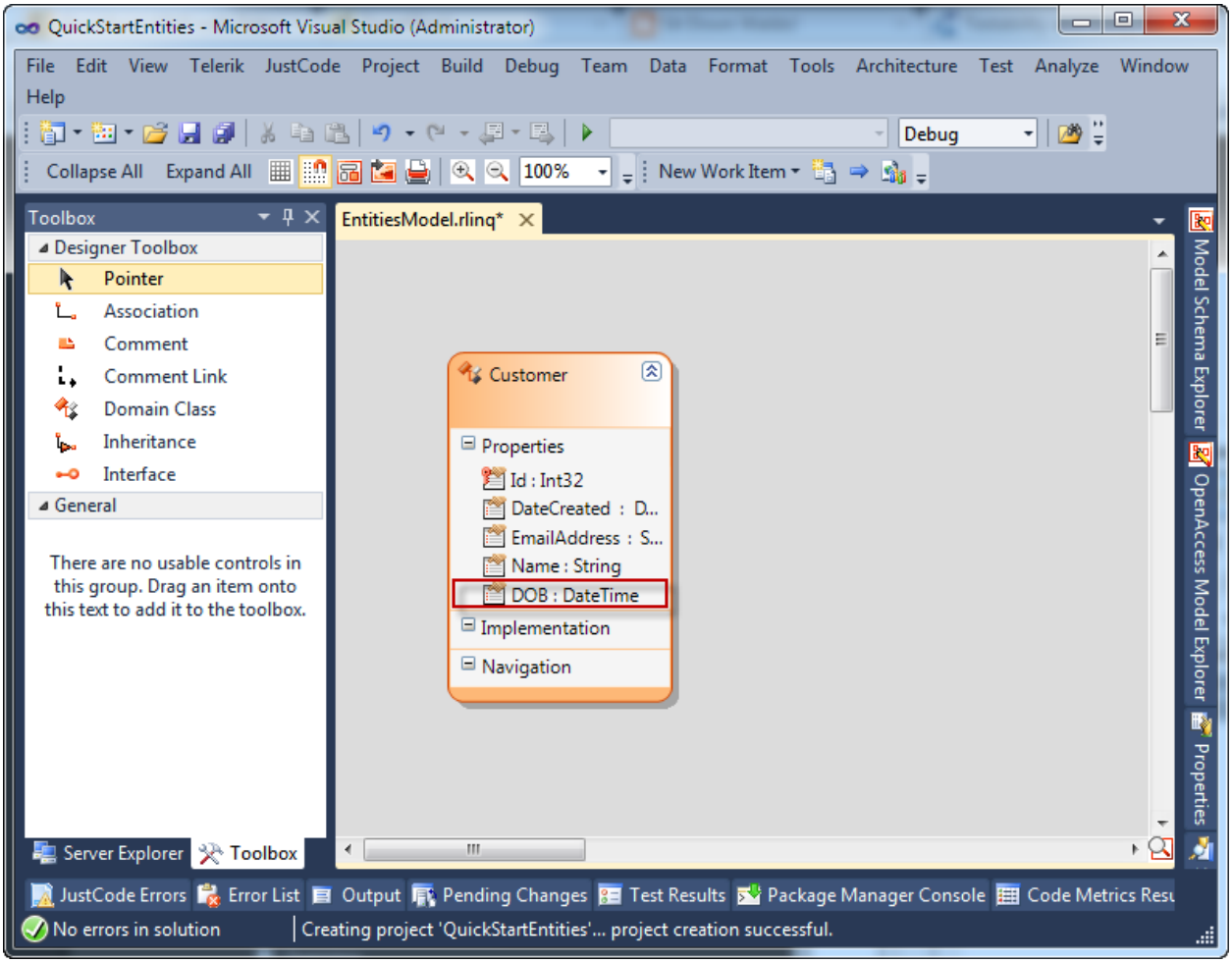


Important Note: Not all changes in the database schema have to be pushed over to the domain model. Only push changes OpenAccess should know about.

4. On the final screen, review the changes that will be applied to the domain model, and click **Finish**.

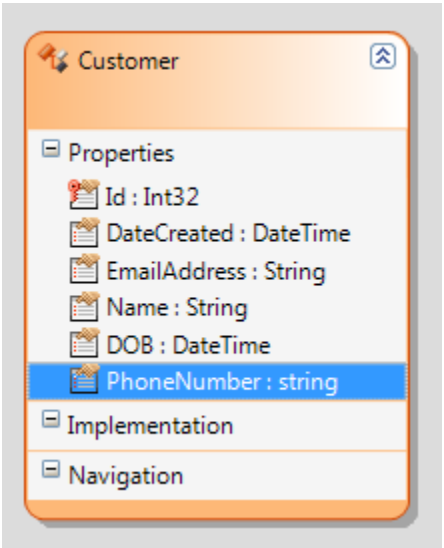


5. When you click finish the domain model objects will be updated based on the selections made in the **Update from Database** wizard. In this case, a DateTime property named **DOB** was added to the **Customer** entity**.**

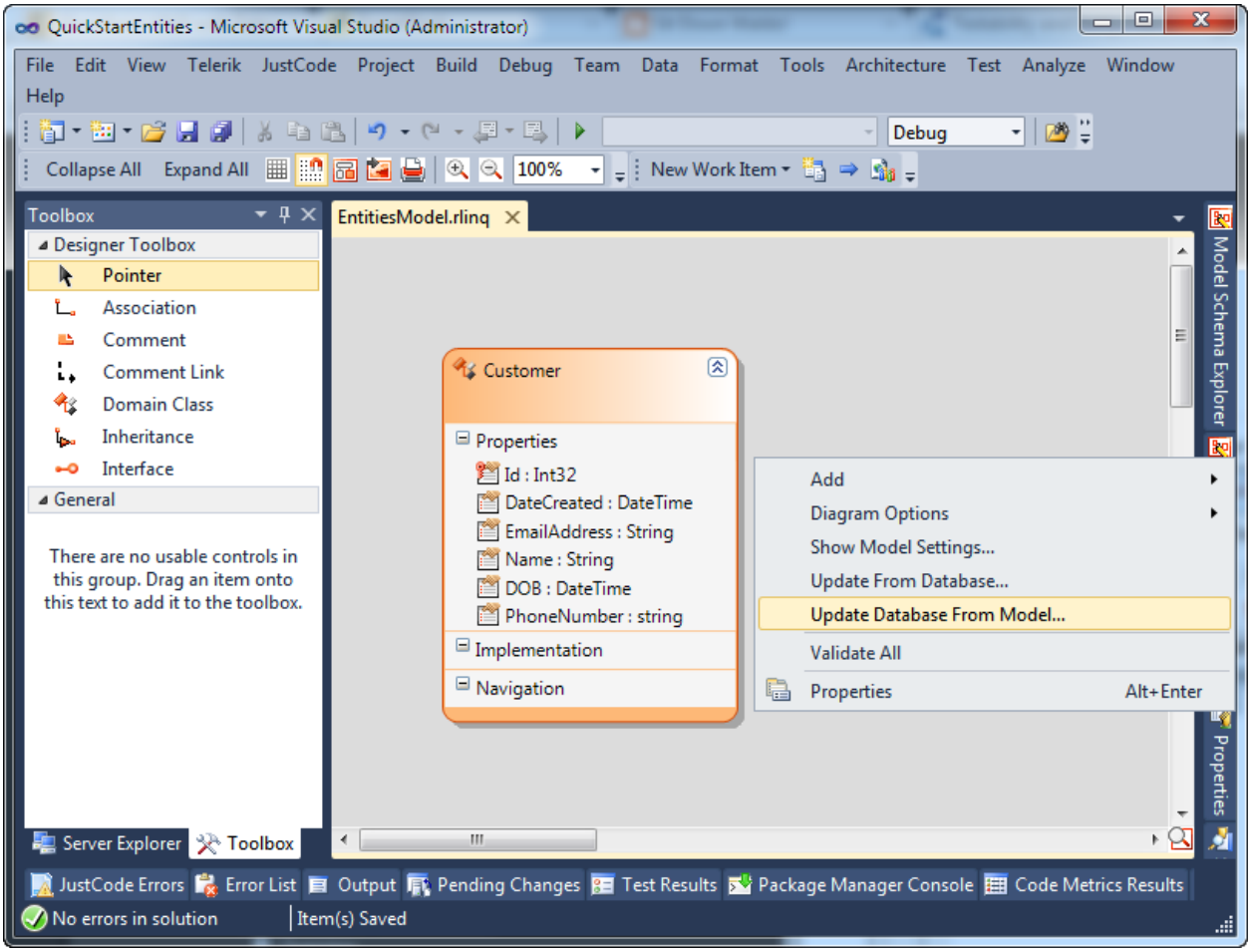


Pushing Domain Model Changes to the Database

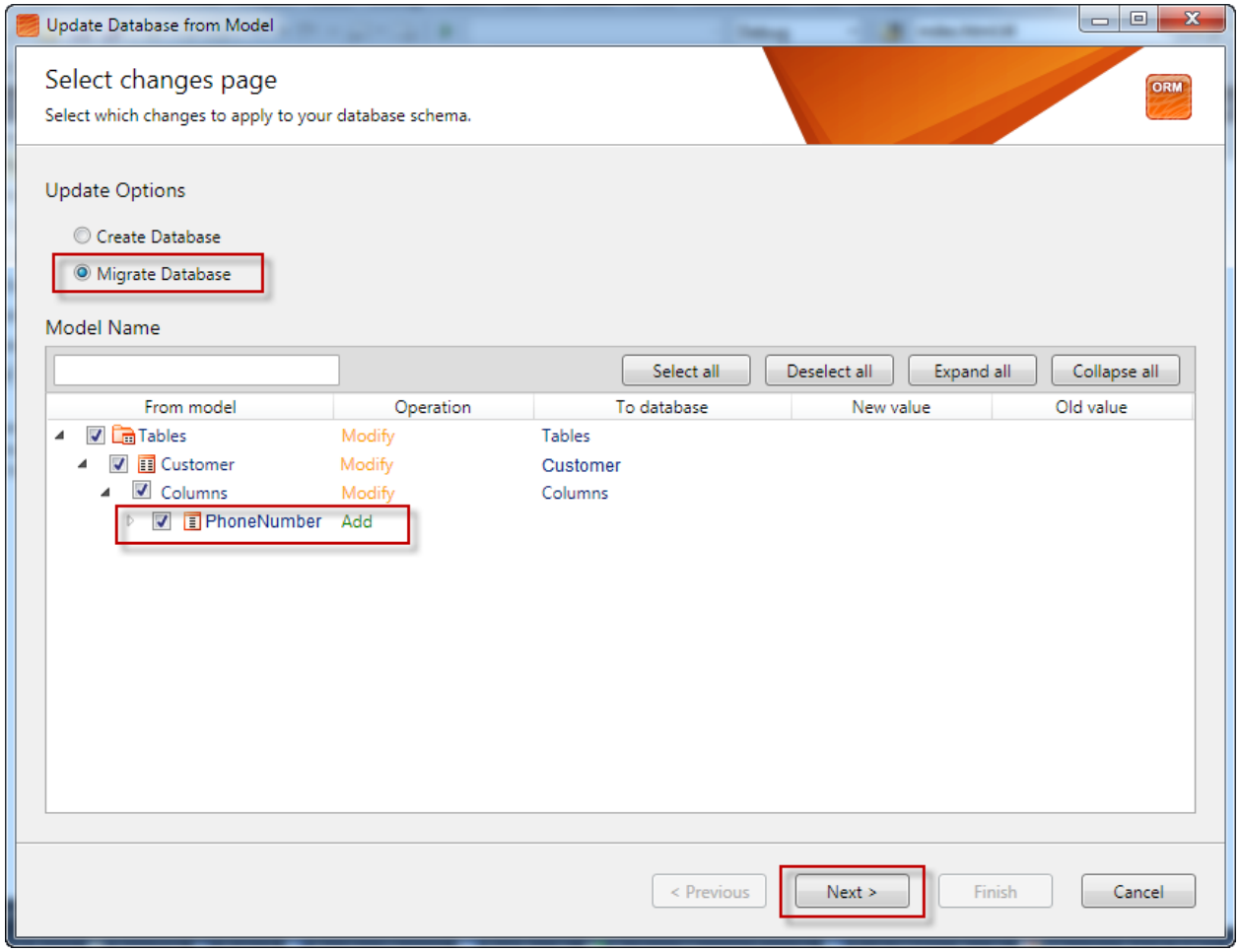
1. **Add** a new property string named **PhoneNumber** to the **Customer** entity.



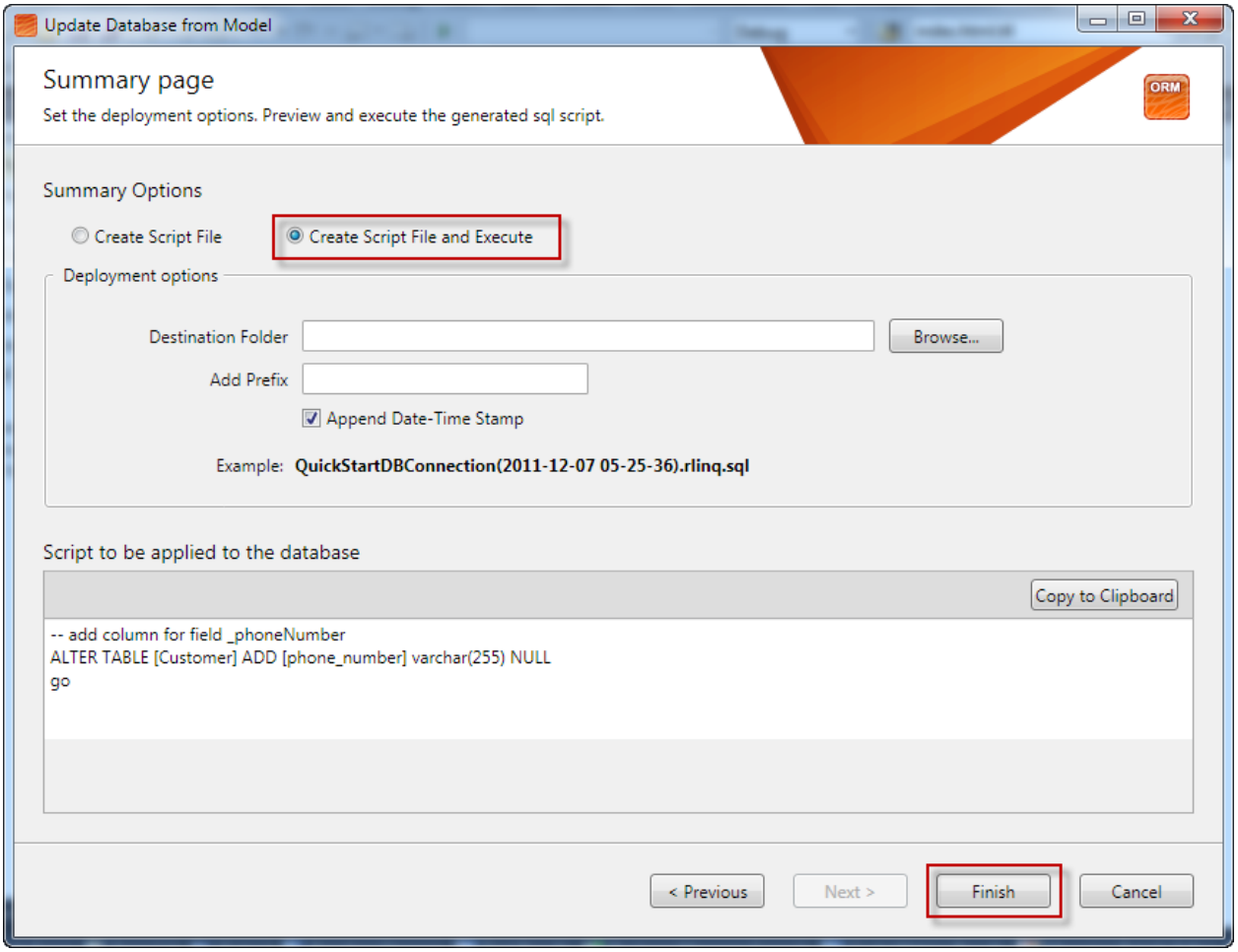
2. **Save** all pending changes to the domain model by pressing **Ctrl + S**.
3. **Right Click** the design surface, and select **Update Database from Model**.



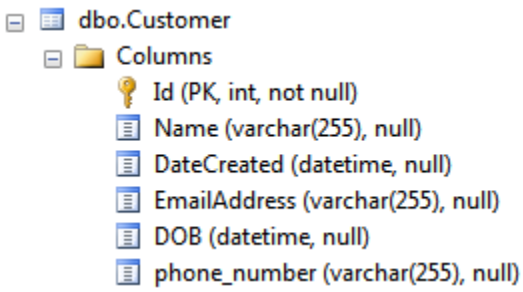
4. Select **Migrate Database**, check each change that should be **applied** to the database, and click **Next**. In this case,



5. Select **Create script file and execute**, and then click **Finish**.



6. At this point OpenAccess will create, and save, a copy of the script in the specified destination folder, and it will execute the migration script against the database. Once complete the database should look like the following:



Note

In this example the column name was not specified for the PhoneNumber property. In this case OpenAccess will automatically create a column name based on the property name. To learn how to configure the column name, please read this documentation article.

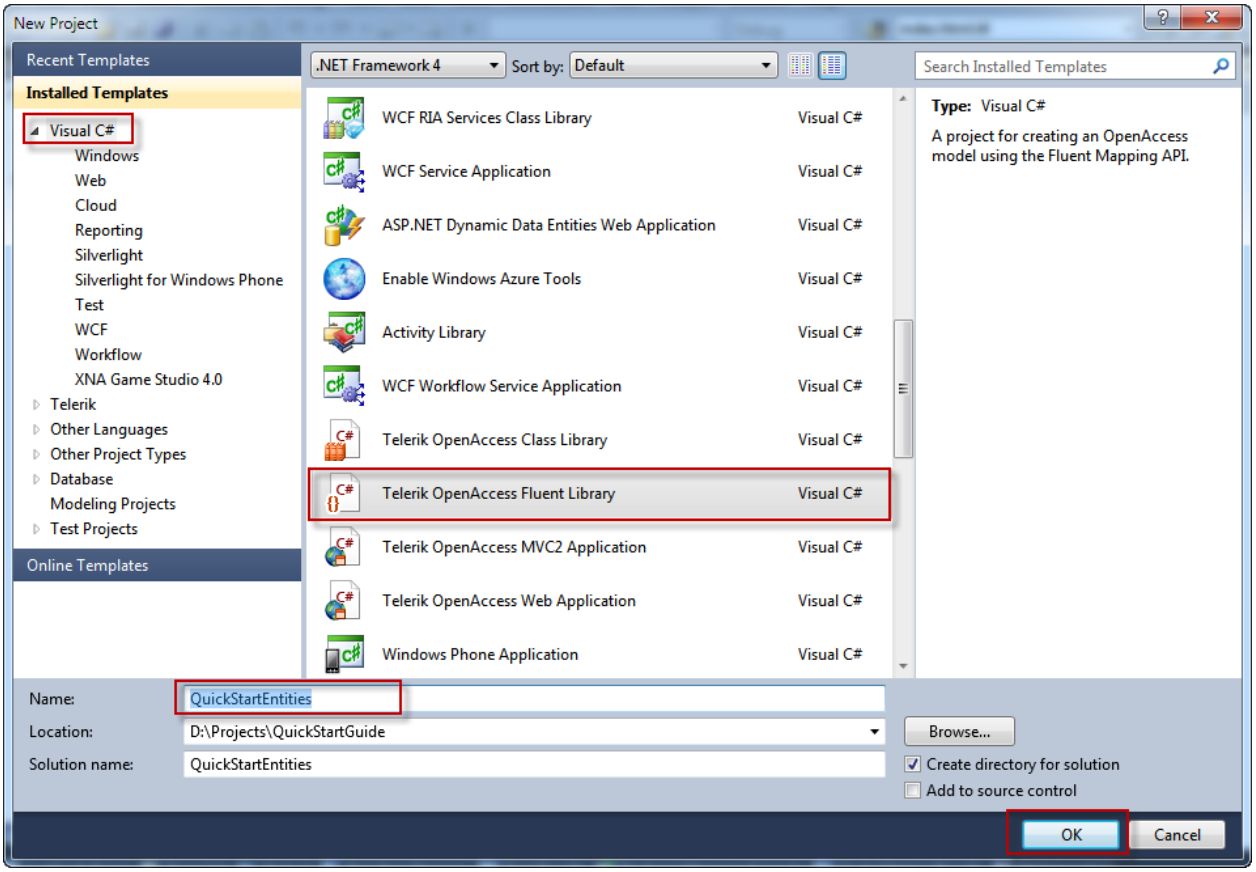
Using Fluent Mapping API

OpenAccess provides a fluent mapping api, also known as code-first, for defining data model models using only code. The fluent mapping api gives developers complete control over the domain model mapping configuration, and schema management.

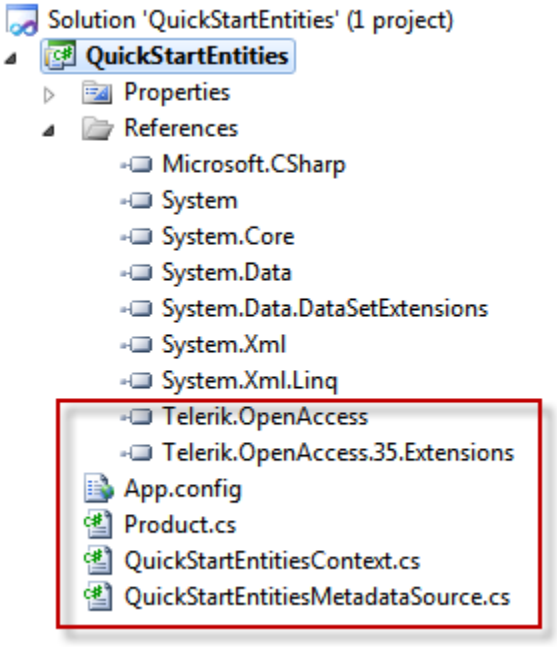
Creating the project

To create a new Telerik OpenAccess domain model using the Fluent Mapping API:

1. Select **File > New Project** in Visual Studio
2. Select **Visual C# or Visual Basic** in the **Installed Template** tree.
3. Then select **Telerik OpenAccess Fluent Library**. Name the project **QuickStartEntities**, and then click **OK.**



4. At this point Telerik OpenAccess will create a new project with the required references, and add a basic template for a fluent mapping project.



What are these files?

The app.config contains the connection string, QuickStartEntityContext.cs defines an OpenAccessContext class, which is the primary way developers access the domain model in an application. QuickStartEntitiesMetadataSource.cs defines a FluentMetadataSource class, which is used to specify object to schema mappings. Product.cs is a simple example POCO object.

Building the Model

1. **Expand** the solution explorer.
2. **Rename Product.cs** to **Customer.cs**, open the file, and **replace** the product class definition with the code below:

```
public class Customer
{
    public int ID { get; set; }
    public string Name { get; set; }
    public DateTime DateCreated { get; set; }
    public string EmailAddress { get; set; }
}
```

What Does this Code Do?

This code creates a simple customer object that can be mapped to a database table by OpenAccess.

3. Next, **Open QuickStartEntitiesMetadataSource.cs**, and **remove** the existing code in the **PrepareMapping** method.
4. **Paste** the following code into the **PrepareMapping** method to configure the mapping for the customer object:

```
var configurations = new List<MappingConfiguration>();

var customerMapping = new MappingConfiguration<Customer>();

//map the customer class to a table named customer
customerMapping.MapType(customer => new
{
    ID = customer.ID,
    Name = customer.Name,
    EmailAddress = customer.EmailAddress,
    DateCreated = customer.DateCreated
}).ToTable("Customer");

//Specify the id property for the entity
customerMapping.HasProperty(x => x.ID)
                .IsIdentity(KeyGenerator.Autoinc);

configurations.Add(customerMapping);

return configurations;
```

What Does this Code Do?
This code passes an anonymous type to the MapType method. The properties of the anonymous type will be what OpenAccess uses to create columns in the database. The ToTable method tells OpenAccess what the customer table should be named in the database. Next the entity's Identity field is specified using the IsIdentity() method. KeyGenerator.Autoinc tells OpenAccess that the identity will be calculated by the database.

5. Open **QuickStartEntitiesContext.cs**, **remove** the property exposing the **Products** collection.
6. **Paste** in the following code to expose the customer collection.

```
public IQueryable<Customer> Customers
{
    get
    {
        return this.GetAll<Customer>();
    }
}
?
```

This code exposes the customer objects as an IQueryable<Customer> this allows OpenAccess to offload queries to the database to do the heavy lifting. It also makes it easy to query entity sets using standard LINQ.

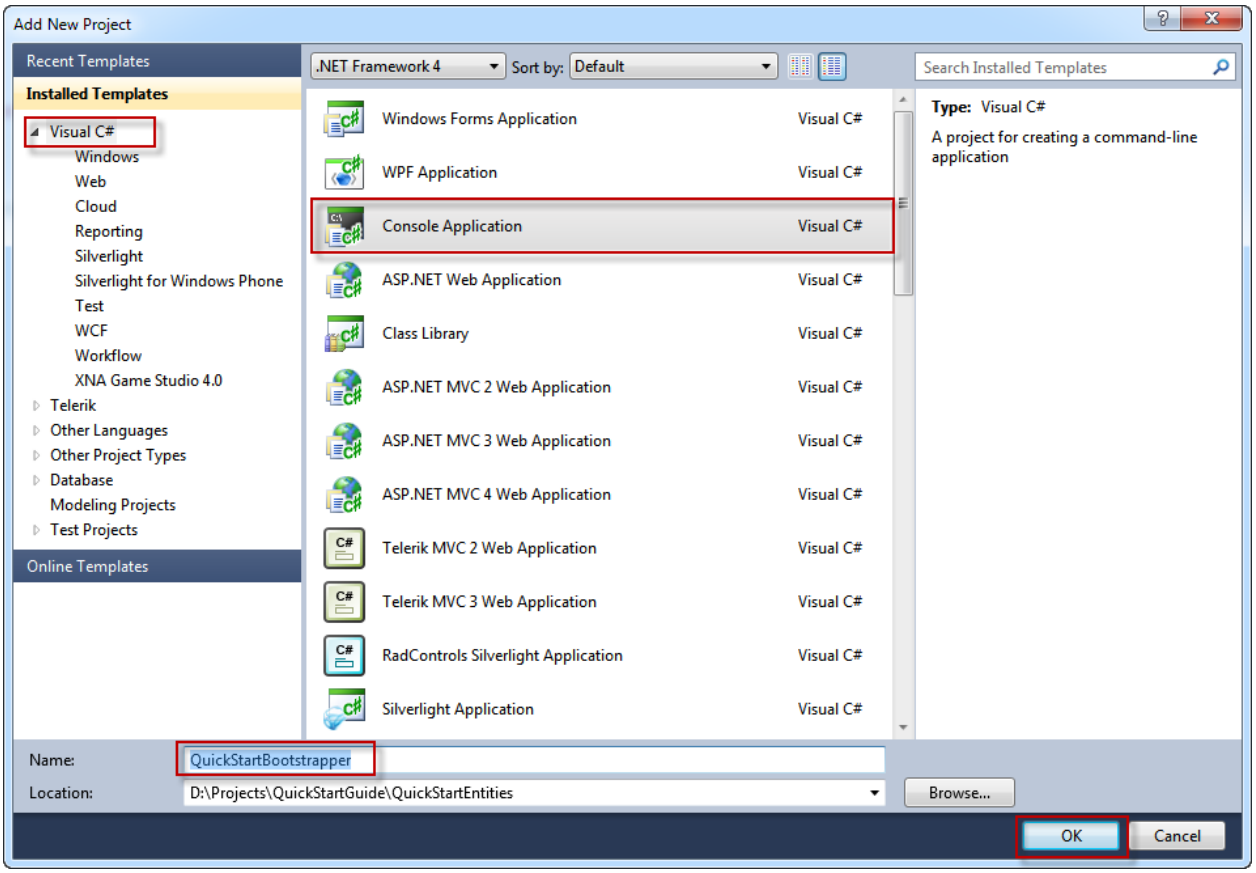
7. At this point the model is fully configured, next the database needs to be created.

Deploying the Database

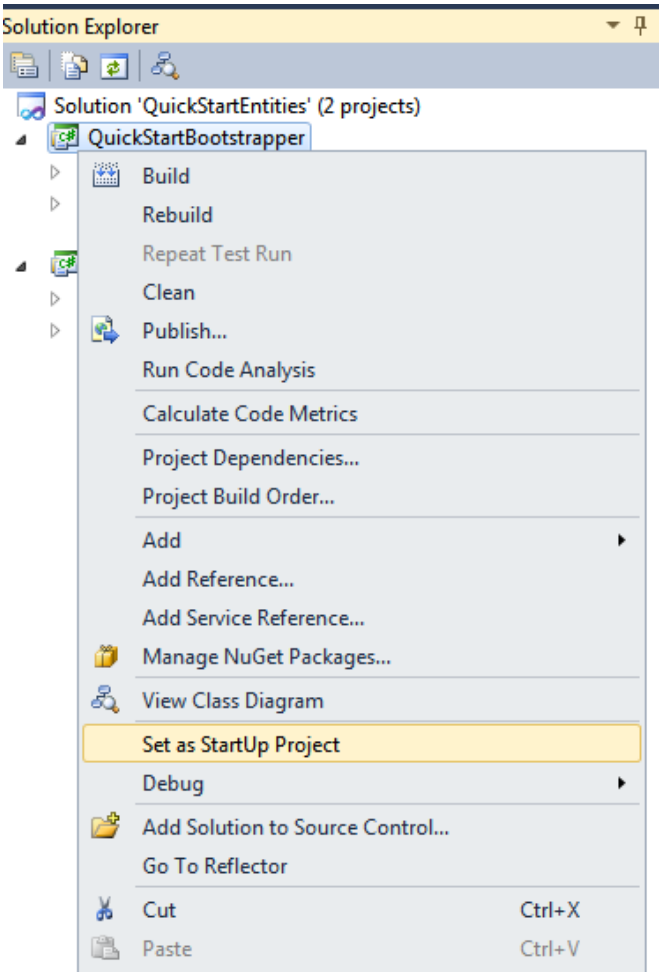
OpenAccess can generate database creation, or migration, scripts based on changes made in the QuickStartEntitiesMetadataSource.cs schema mapping file. In addition, it can execute these scripts, making database creation, and migration very simple.

To create or update the database based on the fluent mapping configuration:

1. **Create** a new **Console Application**, name it **QuickStartBootstrapper**, and click **OK.**



2. Right click on the **Console Application**, and select **Set as StartUp Project**



3. Expand the **Solution Explorer**, right click on **References**, and select **Add Reference**.
4. **Add References** to:
   - Telerik.OpenAccess.dll
   - Telerik.OpenAccess.35.Extenstions.dll
   - The model project, in the case of this guide: **QuickStartEntites**
5. Copy the **app.config** from the **QuickStartEnities** project to the **QuickStartBootstrapper** project.
6. Open **Program.cs** in the **Console Application.**
7. **Replace** all of the existing code with the following:

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

namespace QuickStartBootstrapper
{
    class Program
    {
        static void Main(string[] args)
        {
            UpdateDatabase();
```

```
            Console.Write("Database update complete! Press any key to
        close.");
            Console.ReadKey();

        }

        private static void UpdateDatabase()
        {
            using (var context = new QuickStartEntitiesContext())
            {
                var schemaHandler = context.GetSchemaHandler();

                EnsureDB(schemaHandler);
            }
        }

        private static void EnsureDB(ISchemaHandler schemaHandler)
        {
            string script = null;

            if (schemaHandler.DatabaseExists())
            {
                script = schemaHandler.CreateUpdateDDLScript(null);
            }
            else
            {
                schemaHandler.CreateDatabase();
                script = schemaHandler.CreateDDLScript();
            }

            if (!string.IsNullOrEmpty(script))
            {
                schemaHandler.ExecuteDDLScript(script);
            }
        }
    }
}
```

What Does this Code Do?
The first thing this code does is create a new instance of the **QuickStartEntitiesContext**, it is wrapped in a using statement to make sure it is properly disposed. Next it checks to see if the database exists. If it does not exist, the database is created, and then the schema is applied. If the database already exists, OpenAccess will create and run a migration script against the database.

At this point when you run the console application OpenAccess will update the database based on any changes made in the **QuickStartEntitiesMetadatasource** class.

Next Steps…

At this point the model is ready to be used in an application, or run the Data Service Wizard to create a service layer over the model.

Working with the OpenAccess Data Model

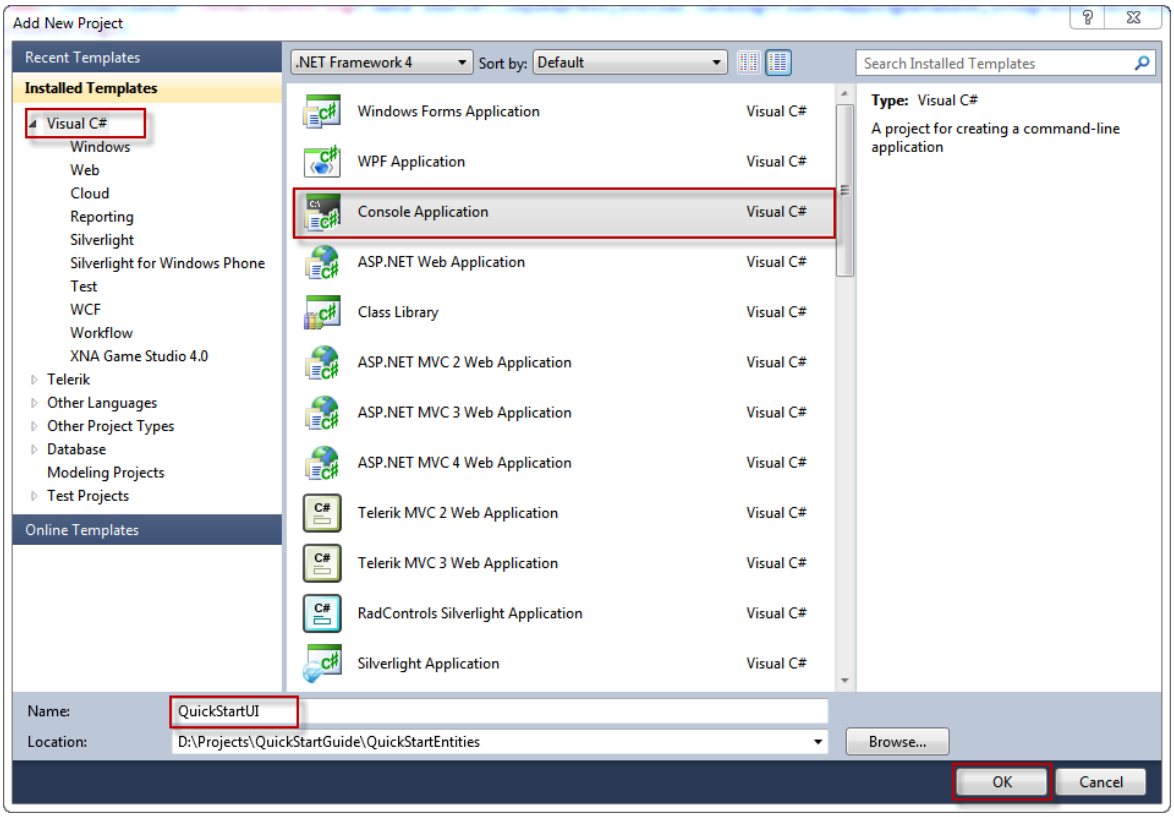
OpenAccess data models can be used with just about any .NET platform. The model can be used directly in web, and windows application, and for rich client.

Creating the Client Application

Note

For this guide we will create a basic console application to interact with the OpenAccessContext; however, the same concept apply regardless of the application platform(WinForms, WPF, MVC, ASP.NET, etc.) being used . For more in-depth guides about using OpenAccess on other platforms please refer to the OpenAccess ORM SDK, or this documentation article.

1. **Create** a new **Console Application** in the same solution as the previously built OpenAcccess Data Model, give the application a name, and click **OK.**



2. Right click on the **Console Application**, and select **Set as StartUp Project**
3. Expand the **Solution Explorer**, right click on **References**, and select **Add Reference**.
4. **Add References** to:
   - Telerik.OpenAccess.dll
   - Telerik.OpenAccess.35.Extenstions.dll

- The **QuickStartEntities** project containing OpenAccess data model created using this guide

5. Copy, and paste the **app.config** from the **QuickStartEntities** project to the **Console Application**.
6. Open **Program.cs** in the **Console Application.**
7. **Replace** the existing code with the following:

```
using System;
using System.Linq;
using QuickStartEntities;

namespace QuickStartUI
{
    class Program
    {
        static void Main(string[] args)
        {

            using (var dbContext = new QuickStartEntitiesContext())
            {
                // Add a new Customer.
                Customer newCustomer = new Customer();
                newCustomer.Name = "New Customer";
                newCustomer.DateCreated = DateTime.Now;
                dbContext.Add(newCustomer);

                // Add another new Customer.
                Customer newCustomer2 = new Customer();
                newCustomer2.Name = "New Customer 2";
                newCustomer2.DateCreated = DateTime.Now;
                dbContext.Add(newCustomer2);

                // Commit new customers to the database.
                dbContext.SaveChanges();

                // Get the first Customer using LINQ and modify it.
                Customer firstCustomer =
dbContext.Customers.FirstOrDefault();
                firstCustomer.Name = firstCustomer.Name + "_Updated";

                // Commit changes to the database.
                dbContext.SaveChanges();

                // Use LINQ to retrieve Customer with name 'New
Customer'.
                Customer customerToDelete = (from c in
dbContext.Customers
                                             where c.Name == "New
Customer 2"
                                             select
c).FirstOrDefault();
```

```
                    // Delete the 'New Customer' from the database.
                    dbContext.Delete(customerToDelete);

                    // Commit changes to the database.
                    dbContext.SaveChanges();
                }

                Console.Write("All changes executed properly, press any key
        to close.");
                Console.ReadKey();

            }
        }
    }
```

What Does this Code Do?
The first thing it does is create an instance of the **OpenAccessContext** created early in this guide. The context is created in a using statement to make sure it is properly disposed of after it is used. The next two section of code add two new customers to the OpenAccess context, and then saves them to the database. Next, LINQ is used to retrieve the first customer record, and modify its name. Those changes are then sent to the database using the SaveChanges method on the OpenAccessContext. The last section retrieves a customer by its name using LINQ, and then uses the Delete method on the OpenAccessContext to remove that record from the database.

8. At this point the application is ready to run. When it runs, it will add two new customers to the database, modify one, and then delete one. After running the application there should be one customer entry in the database named "New Customer_Updated."

At this point the data model has been built, an application has been created to interact with the data model, and changes to objects can be persisted to the database. The next step is to expand the data model, and UI application to meet your needs.

Consuming a Model – CRUD and WCF Services

Please, check the following help topics on how to use the created OpenAccess ORM model in the application(s) that you are working on:

- Consuming a Model - CRUD
- Using OpenAccess with WCF Services

Next Steps…

Once the model and application building is complete, hook up the **OpenAccess Profiler and Tuning Advisor** and check performance, or look for potential issues in the data model using the profiler's built in alert system.

Profiling an Application

The **OpenAccess Profiler and Tuning Advisor** makes it easy for developers to see how OpenAccess is working behind the scenes. All of the SQL queries sent to the database server, and the LINQ statement that generated them are easily browsable in the profiler. In addition, the profiler has a built in alert system which will notify developers about potential issues in the domain model, and suggest resolutions.

Configure The Data Model for Profiling

Configuring a Fluent Mapping Model for Profiling

1. Create a fluent mapped data model using the previous section in this guide.
2. Add a reference to **Telerik.OpenAccess.ServiceHost.dll** in the project that will consume the domain model.
3. Open the OpenAccessContext class, if created from this guide, it is the file named **QuickStartEntitiesContext.cs**.
4. **Replace** the existing backendConfiguration declaration with the following depending on the desired method of profiling:

Real-time Profiling

In Real-time profiling, OpenAccess reports its events, and metrics directly to the profiler via a service.

```
private static BackendConfiguration GetConfiguration()
{
BackendConfiguration backendConfiguration = new BackendConfiguration()

     {
          Backend = "mssql"
      };
      // defines the size of the metric store in memory
      backendConfiguration.Logging.MetricStoreCapacity = 3600;

      // defines the size of the event store in memory
      backendConfiguration. Logging.EventStoreCapacity = 10000;

      // defines the log level
      backendConfiguration.Logging.LogEvents = LoggingLevel.Normal;

      return backendConfiguration;
  }
```

Offline Profiling

In offline profiling, all OpenAccess metrics and events are written to a log file, which can be reviewed in the Profiling and Tuning Wizard at a later time.

```
private static BackendConfiguration GetConfiguration()
        {
```

```
            var backendConfiguration = new BackendConfiguration()
            {
                Backend = "mssql"
            };

            // defines the log level
            backendConfiguration.Logging.LogEvents =
LoggingLevel.Normal;
            backend.Logging.Downloader.Filename =
"c:\\QuickStartProfilingSession";
            backend.Logging.Downloader.EventBinary = true;
            backend.Logging.Downloader.MetricBinary = true;

            return backendConfiguration;
        }
```

Real-time & Offline Profiling

This will allow OpenAccess to report events and metrics to the profiler in real time, and it will also output all data to a log file, which can be reviewed at a later time.

```
private static BackendConfiguration GetConfiguration()
        {
            var backendConfiguration = new BackendConfiguration()
            {
                Backend = "mssql"
            };

            // defines the log level
            backendConfiguration.Logging.LogEvents =
LoggingLevel.Normal;
// defines the size of the metric store in memory
            backendConfiguration.Logging.MetricStoreCapacity = 3600;

            // defines the size of the event store in memory
            backendConfiguration. Logging.EventStoreCapacity = 10000;

            backend.Logging.Downloader.Filename =
"c:\\QuickStartProfilingSession";
            backend.Logging.Downloader.EventBinary = true;
            backend.Logging.Downloader.MetricBinary = true;

            return backendConfiguration;
        }
```

Note

Real-time profiling, and offline profiling can be used together, or separately.

5. **Update** the **constructor** to call the new GetConfiguration method created in step 4, rather than passing in the previously defined backendConfiguration object.
   As show here:

```
public QuickStartEntitiesContext()
    : base(DbConnection, GetConfiguration(), metadataSource)
{}
```

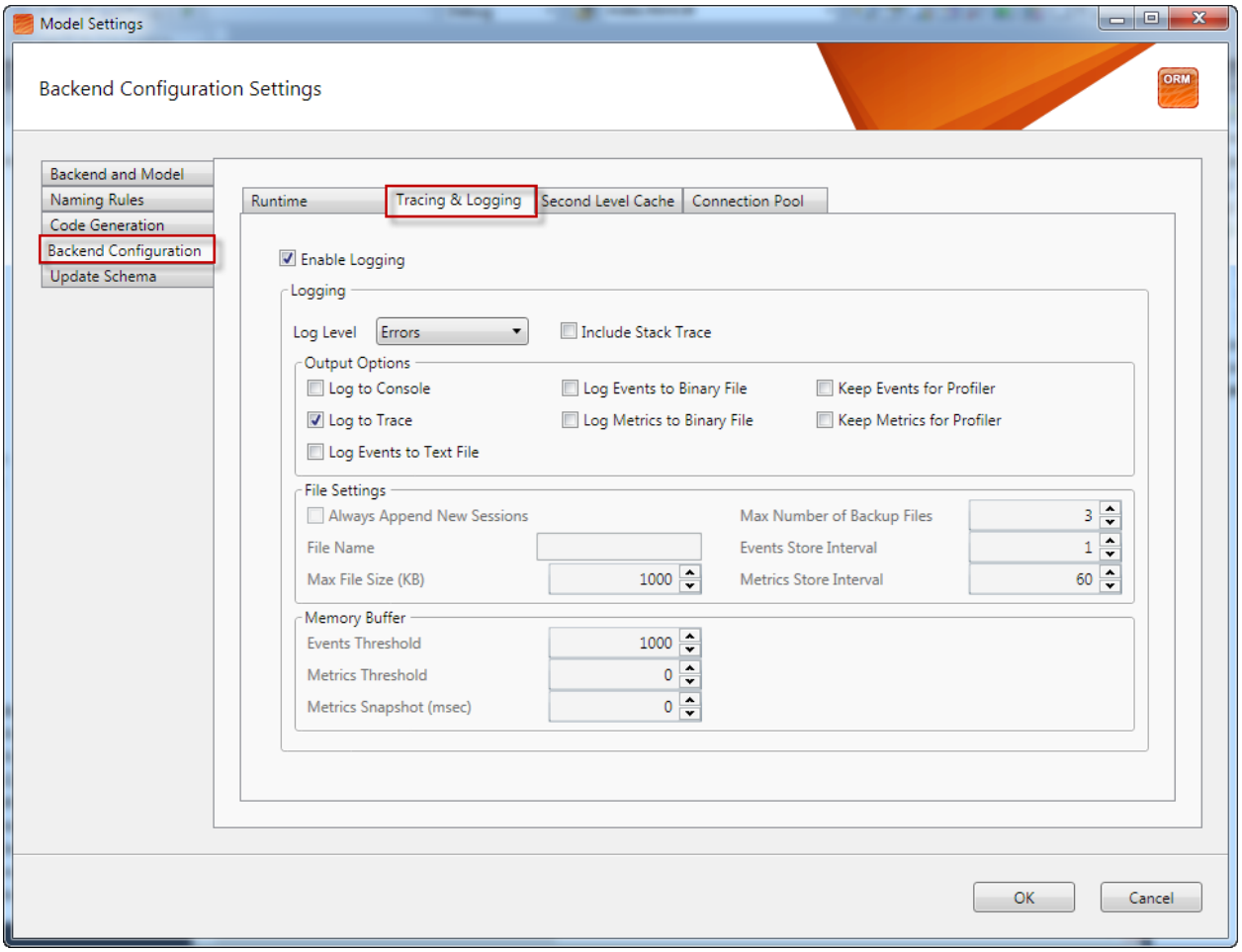
6. The fluent data model is now configured for profiling.

Next Steps…
With the fluent model configured, the next step is to connect the profiler for real time profiling, or view the logs generated during offline profiling.

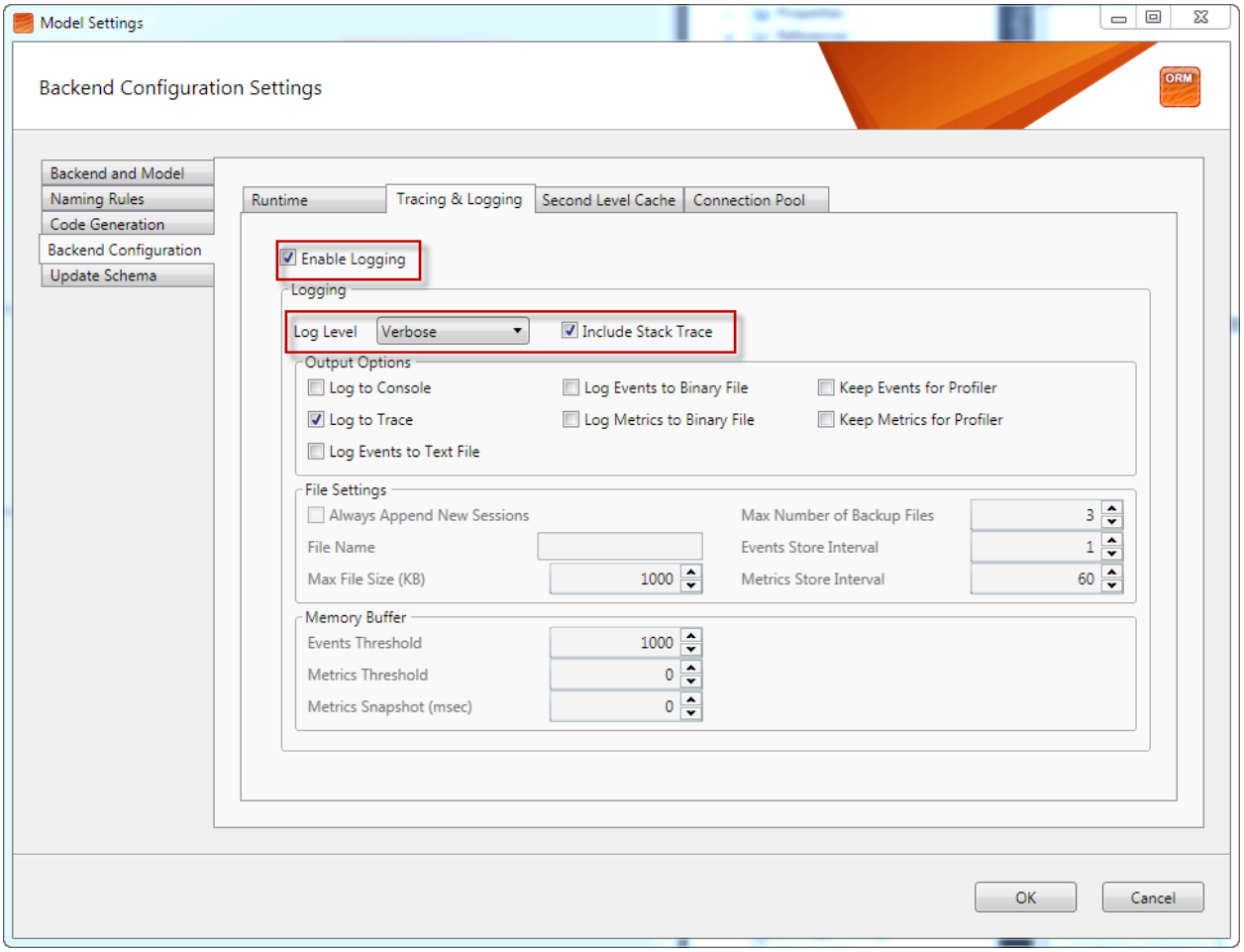
Configuring a Visual Designer Model (Forward or Reverse Mapping) for Profiling

To configure the profiler for online capture:

1. Create a data model using the Visual Designer section of this guide.
2. Add a reference to **Telerik.OpenAccess.ServiceHost.dll** in the project that will consume the domain model.
3. Open the OpenAccess Visual Designer.
4. Right click on the design surface, and select **Show Model Settings.**
5. In the model settings window navigate to **Backend Configuration > Tracing and Logging**.



6. Ensure **Enable Logging** is checked, and adjust the log level. By default only errors are reported by OpenAccess, to send more detail to the profiler, set the Log Level to Normal, Verbose, or All. Also, check **Include Stack Trace** so it is reported to the profiler.

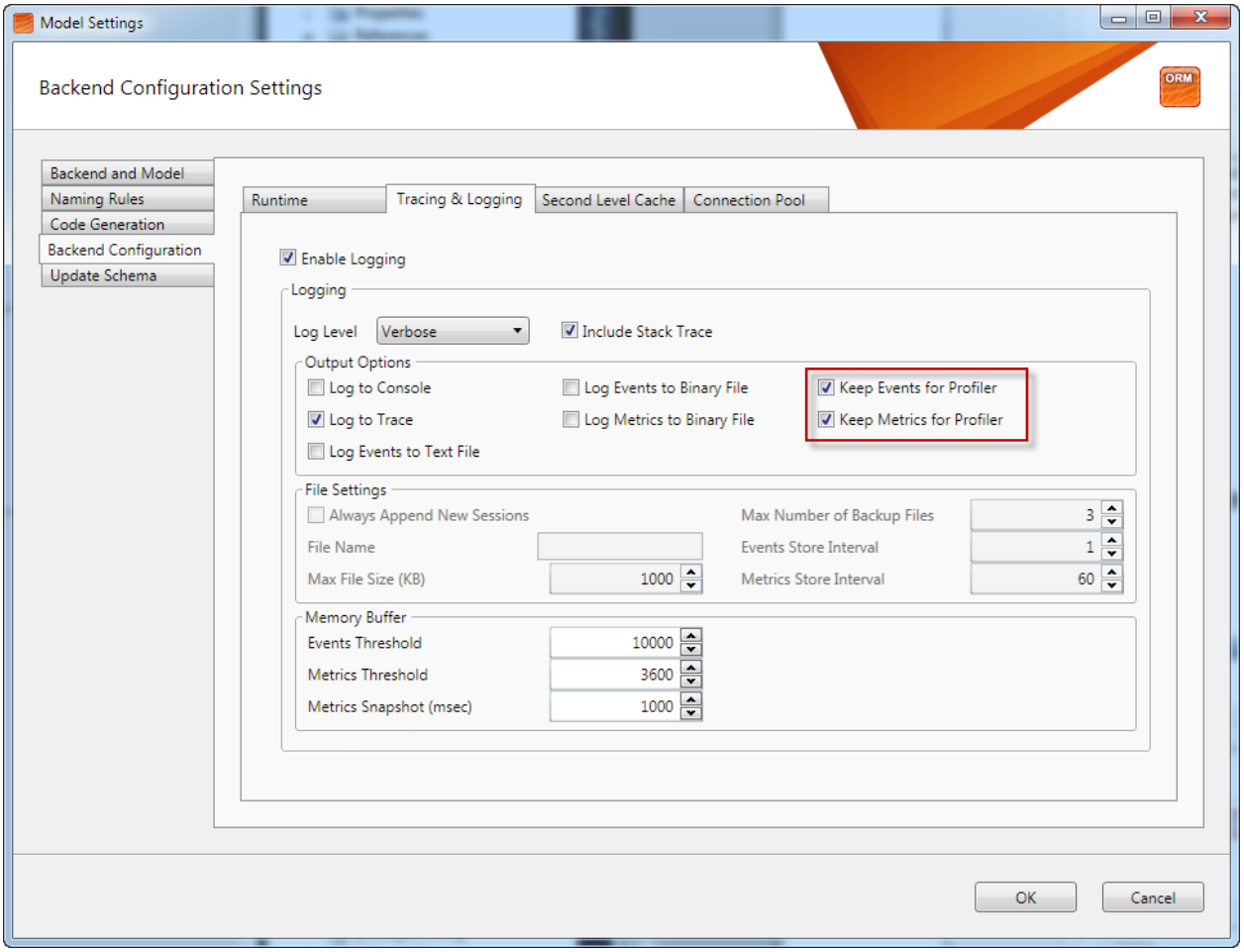


7. Now the profiling method must be configured. OpenAccess provides the follow profiling methods:

Real-time Profiling

In real-time profiling, OpenAccess reports its events, and metrics, directly to the profiler via a service.

To enable real-time profiling, check **Keep Events for Profiler**, and **Keep Metrics for Profiler**.



Offline Profiling

In offline profiling, all OpenAccess metrics and events are written to a log file, which can be loaded and reviewed in the Profiler and Tuning Advisor at a later time.

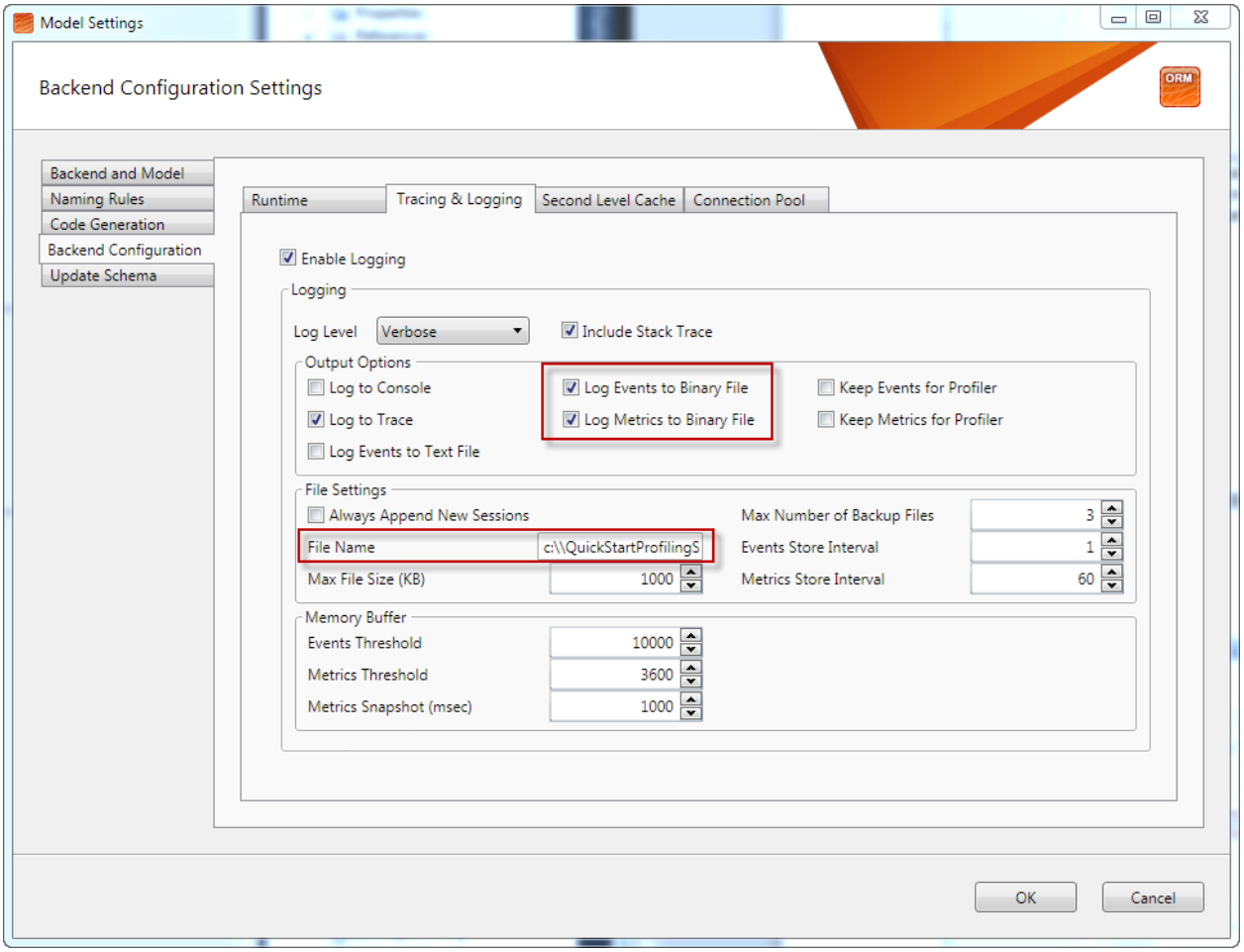
To enable offline mode , check **Log Events to Binary File** and **Log Metrics to Binary File.** This will enable the **File Settings** section. In the File name section, put the following path:

C:\QuickStartProfilingSession

OpenAccess will now log all events, and metrics a file at this location.

Note

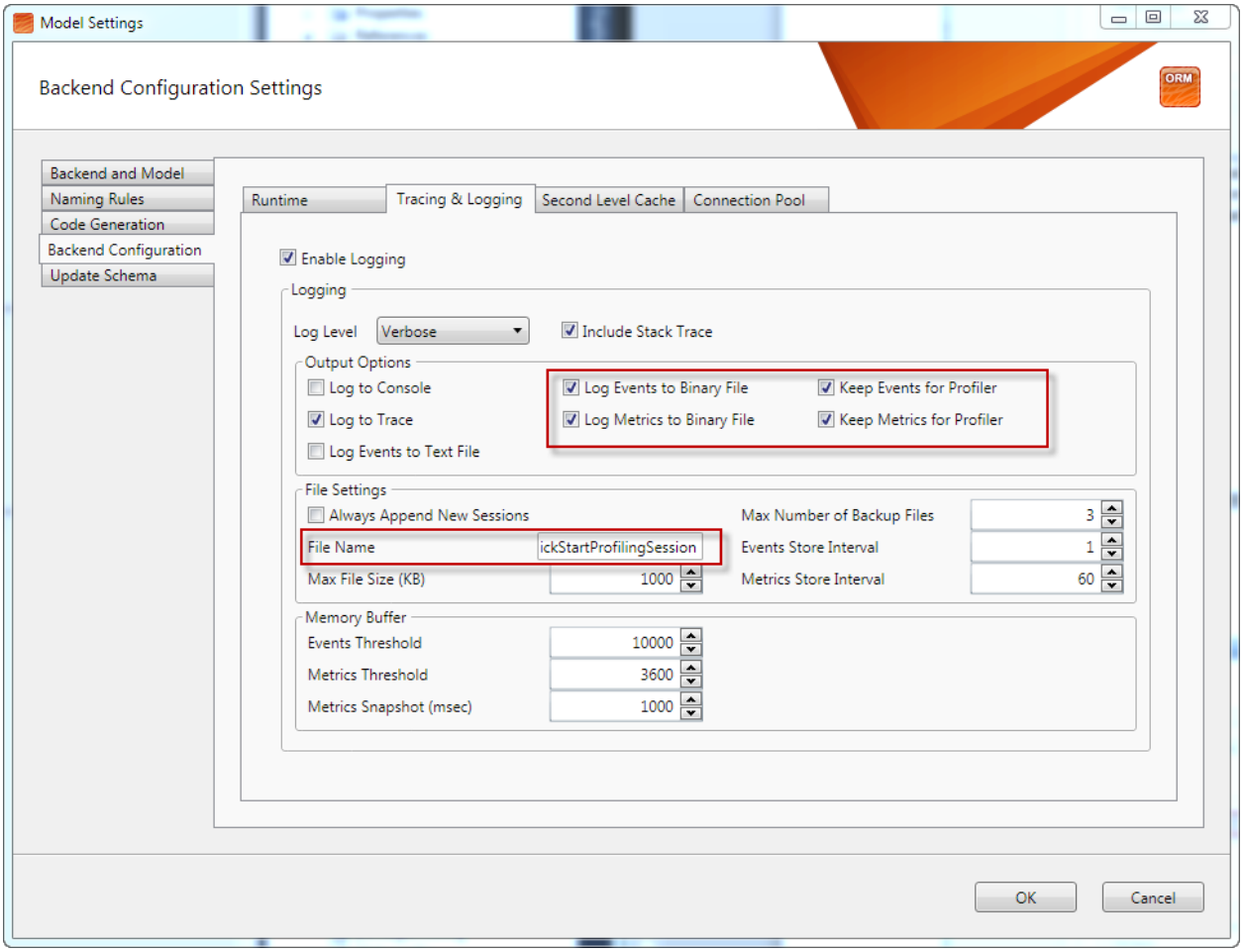
The OpenAccess logger provides the ability to configure dynamic log file names, to learn more please read this documentation article.



Real-time & Offline Profiling

This will allow OpenAccess to report events and metrics to the profiler in real time, and it will also output all data to a log file, which can be reviewed at a later time.

To enable both modes, simply make the selections described in both of the above configurations.



Note

Real-time profiling, and offline profiling can be used together, or separately.

8. Now click **OK**. At this point the model is configured to report SQL Events, and Metrics.

Next Steps…

With the fluent model configured, the next step is to connect the profiler for real time profiling, or view the logs generated during offline profiling.

Real-time Profiling

Applications using OpenAccess ORM can report SQL events, and metrics to the Profiler and Tuning Advisor in real time. To do this, first a service must be added to the application consuming the OpenAccess Data Model. Then the profiler must be configured to connect to the service.

Configuring an Application

1. When the application is starting, the profiler service needs to started.

   In a **web application**, this should be pasted in **Application_Start**

   In a **windows application**, paste this **before** showing the first form.

```
Telerik.OpenAccess.ServiceHost.ServiceHostManager.StartProfilerService(15555);
```

This will start the profiler service on port 15555. A different port can be used by passing a different port number to the `StartProfilerService` method.

2. When the application is ending, the service needs to be stopped by calling:

```
Telerik.OpenAccess.ServiceHost.ServiceHostManager.StopProfilerService()
```

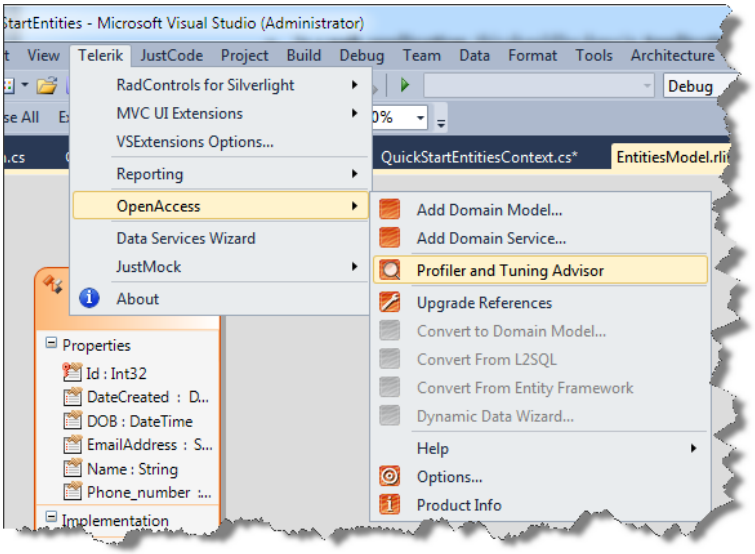
In a **web application**, this should be added in **Application_End**

In a **windows application**, this should be added to the closing event for the main form.

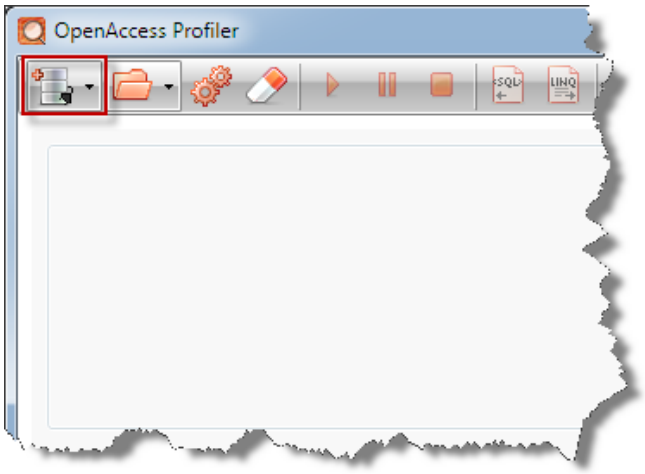
3. **Run** the application.

Connecting the Profiler

1. Make sure the application is running.
2. **Open** the profiler**,** by navigating to **Telerik > OpenAccess > Profiler and Tuning Advisor** in **Visual Studio**

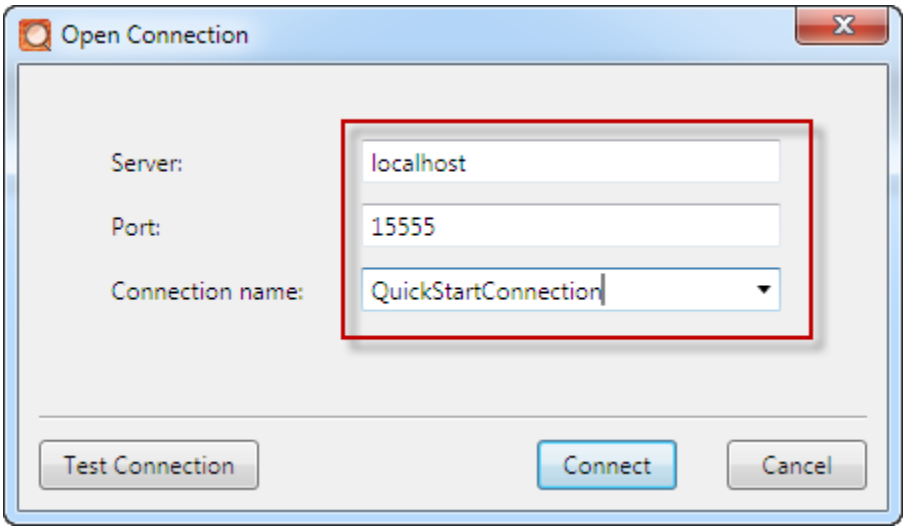


3. Once the profiler opens, **click** the **Connect To** menu item



4. **Configure** the connection

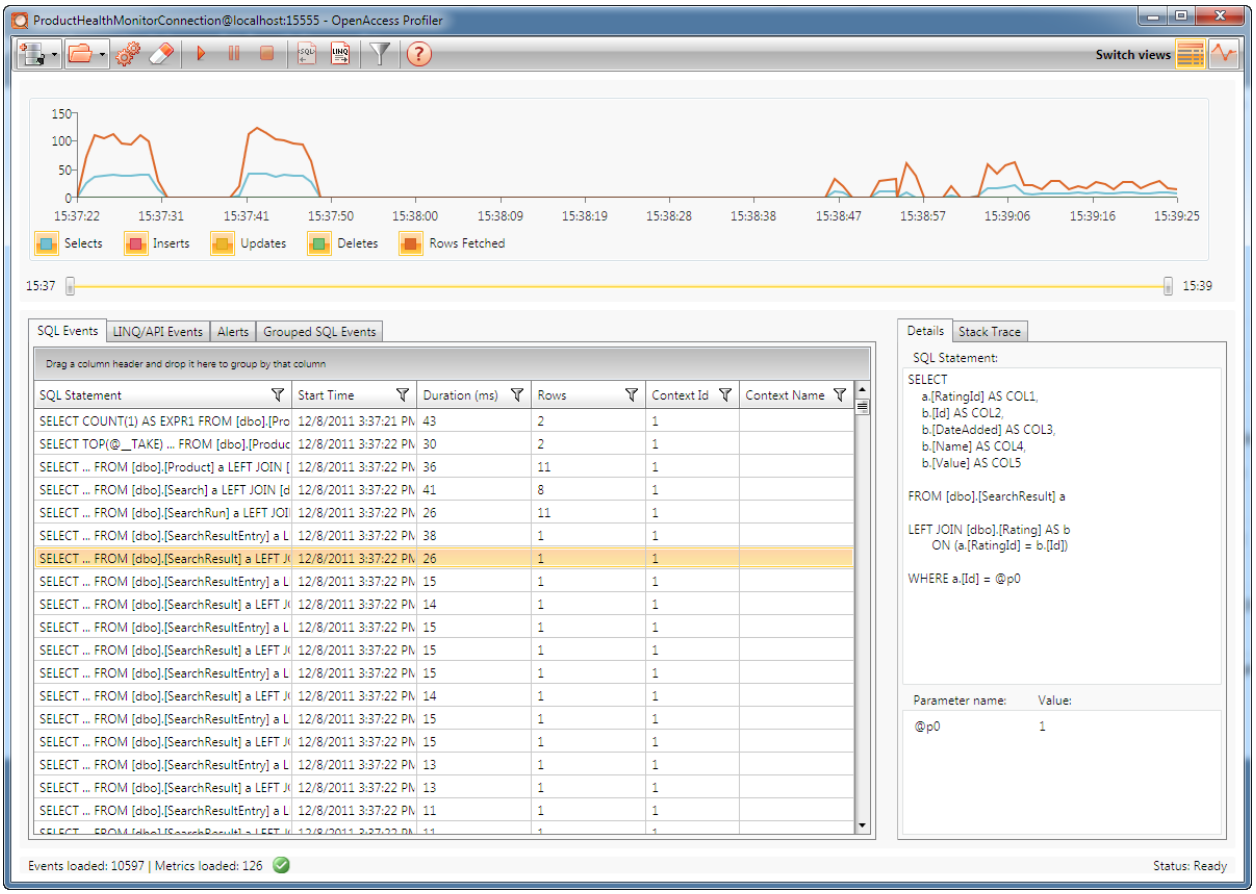
a. Server – The server where your application is running. In most cases this will be localhost.
b. Port – The service host added to the application earlier uses a specific port to communicate with the profiler. This port is what is passed into `StartProfilerService`(*port*);
c. Connection Name– The profiler needs to know which connection to monitor. Specifying the connection name gives the profiler the needed information to properly report activity.



5. Once configured, click **Test Connection** at which point the profiler will indicate whether or not it was able to successfully connect to the service host.

   If **Test Connection** reports a **failure**, ensure that the application is running, and that you specified correct info on the **Open Connection** dialog.

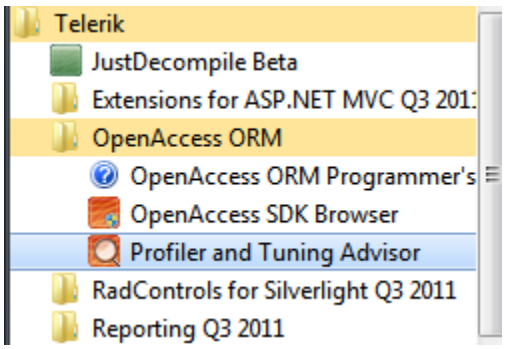
6. Once **Test Connection** comes back successful, click **Connect**.
7. At this point the application being profiled can be used, and all activity will be reported to the profiler.



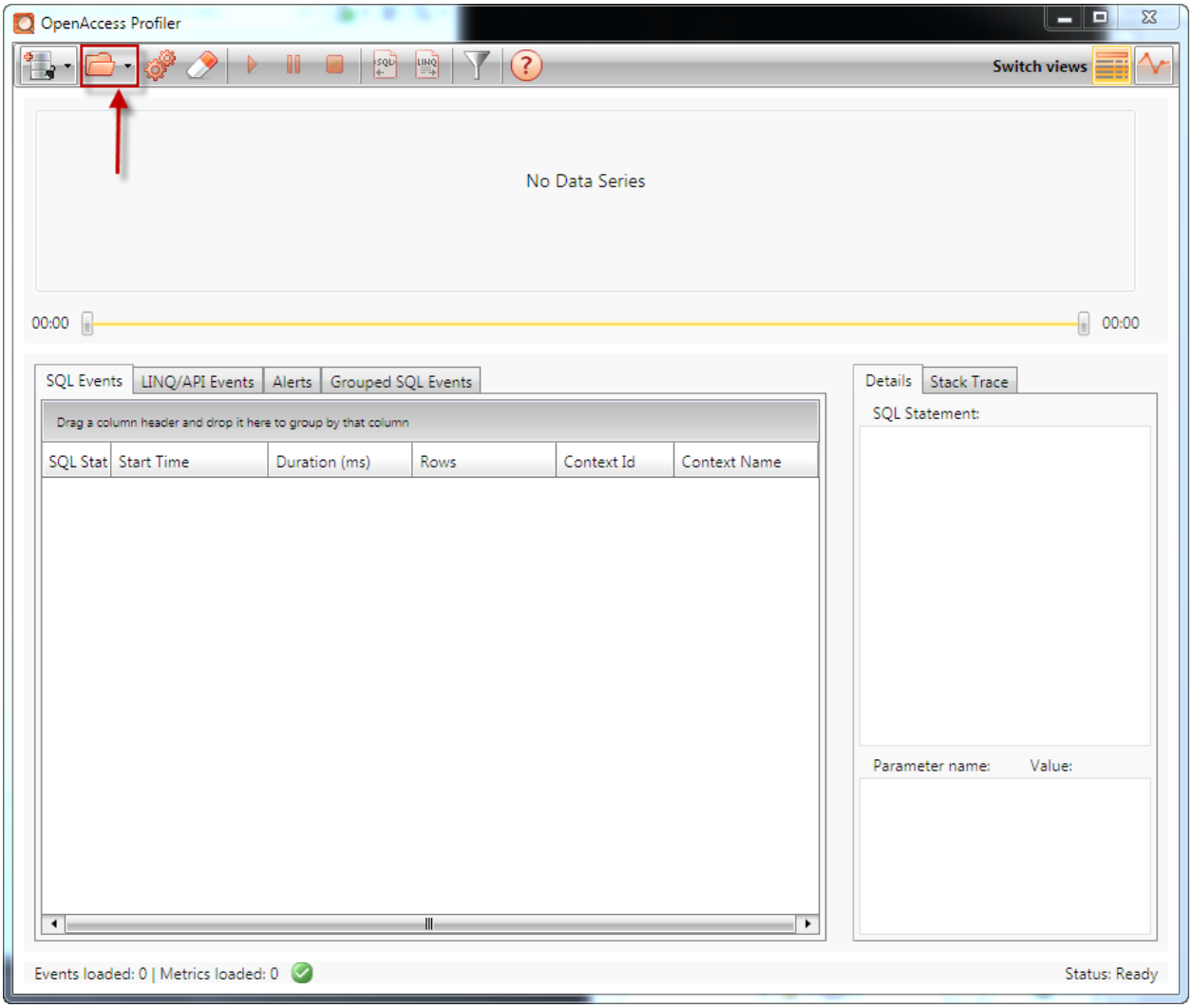
For more information on all the available features of the profiler make sure to check out this documentation article.

Viewing Offline Profiling Session Logs

1. Open **OpenAccess Profiler and Tuning Advisor** from the start menu by navigating to Start>All Programs >Telerik > OpenAccess ORM > **Profiler and Tuning Advisor**.



2. Click the **Folder** icon in the toolbar of the profiler.



3. **Navigate** to the directory where the log was saved, for example, in this guide it was saved to the root C drive.
4. **Select** the **.oalog** file.

   The OpenAccess logger produces two files:

   a. .oalog - contains all of the operation information.
   b. .oametrics – contains all of the counters for the OpenAccess context. Things like context lifetime, inserts per second, selects per second, cache hits, cache misses, etc. are stored in this file.

5. The log should now be ready for to explore in the **OpenAccess Profiler and Tuning Advisor**

For more information on all the available features of the profiler make sure to check out this documentation article.