

Four Simple Steps

Creating a Silverlight Application Using *IdeaBlade DevForce Silverlight*

DevForce has been designed from the ground up by experienced enterprise application developers with several very important goals in mind:

- Make it (following Albert Einstein’s famous advice) “as simple as possible, but not simpler.”
- Support (and encourage) “separation of concerns” in the architecture: your business model (and all business logic) is entirely separate from your user interface, so you can reuse it across more than one user interface, and more than one application.
- “Don’t fence me in.” You build the application *you* want to build, with the functionality and user interface you need. We make that much easier for you, and we make sure your end product is truly industrial strength -- but we don’t take over, and we don’t get in your way. Not in the beginning, and – even more importantly – not later on, *after* you’ve invested so much into your development effort that changing course is no longer an option.

So what’s it like to build an application with DevForce Silverlight? Let’s take a walk through the four simple steps:

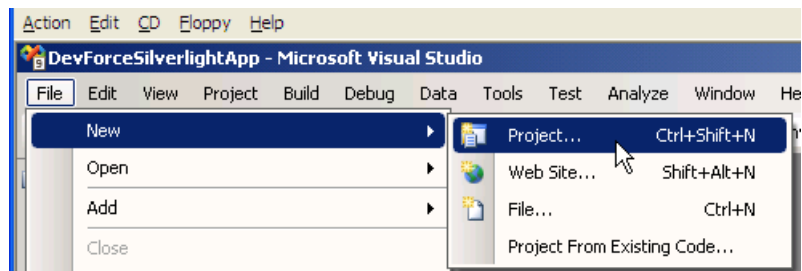
- Generate your Silverlight and web projects using the DevForce Silverlight project template
- Create your Entity Data Model
- Create your DevForce Domain Model
- Create your UI.

You can navigate to the Visual Studio solutions for this article using the following links:

- [CodeCS](#)
- [CodeVB](#)

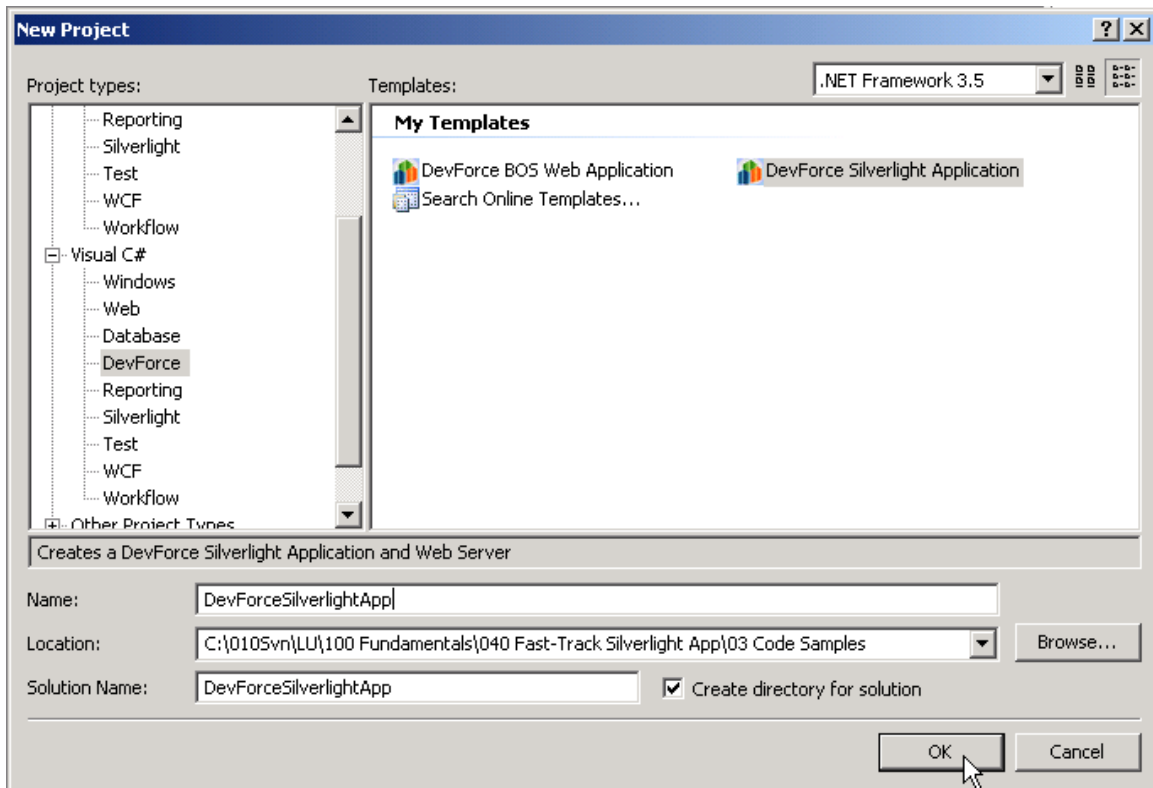
Step1. Generate Your Silverlight and Web Projects

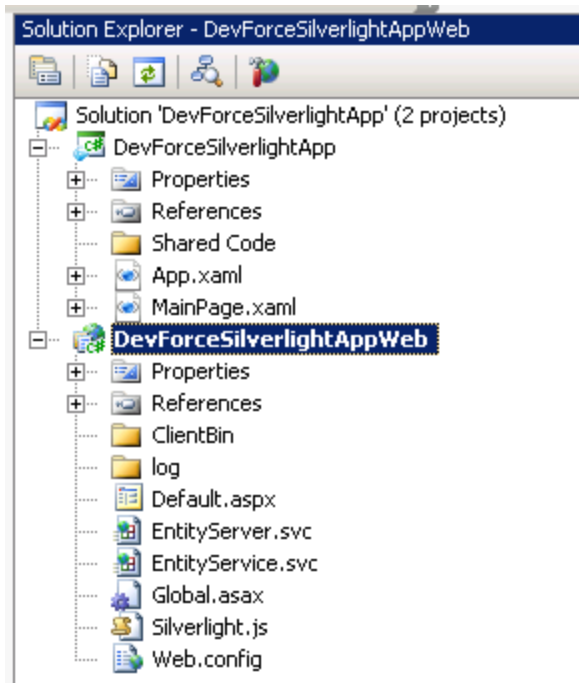
DevForce provides a Visual Studio project template to get off the ground quickly with your application. Choose File / New / Project from the main menu, find the DevForce section under the language of your choice, and select the DevForce Silverlight



Application project template.

Specify the name and storage location for your new solution, and click <OK>:





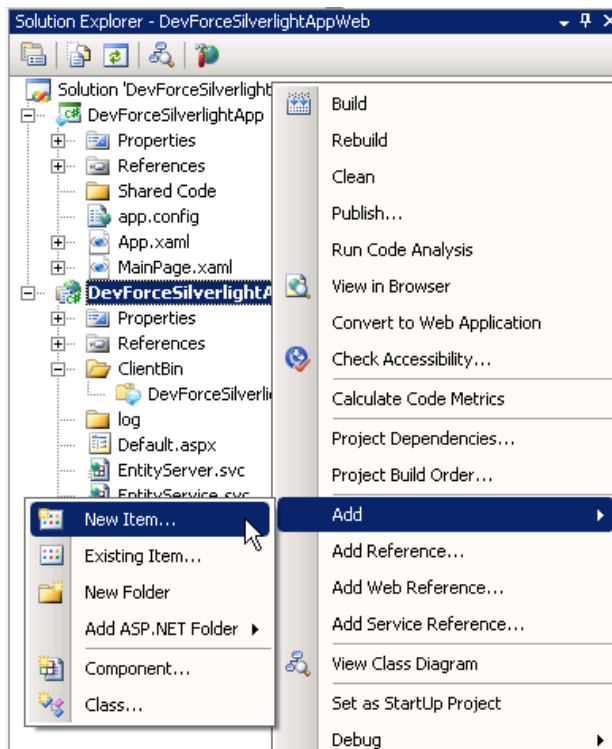
The template creates two projects. The web project (here named DevForceSilverlightAppWeb) will ultimately be deployed on your IIS server; the Silverlight project (DevForceSilverlightApp) will contain the application parts that will be downloaded automatically to the client's browser via a .XAP file.

Note that the web project is set as the Startup Project for the solution. That's important! If the Silverlight app is set as startup, the MainPage will still display, but all operations that require connection to the server – like login and data retrieval – will be dead in the water!

That's it for step one. We have our application structure, and needed references to DevForce and .NET assemblies are already set. Let's get on with creating our application's business model!

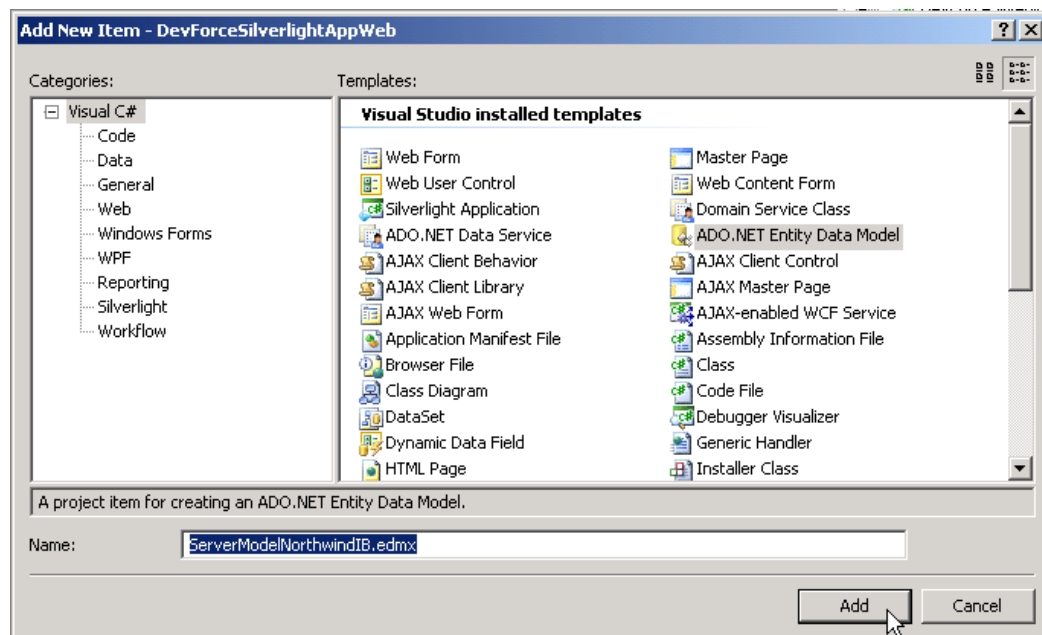
Step2. Create your Entity Data Model

The ADO.NET Entity Data Model is used only server-side in our DevForce Silverlight application, so let's add it to the web project:



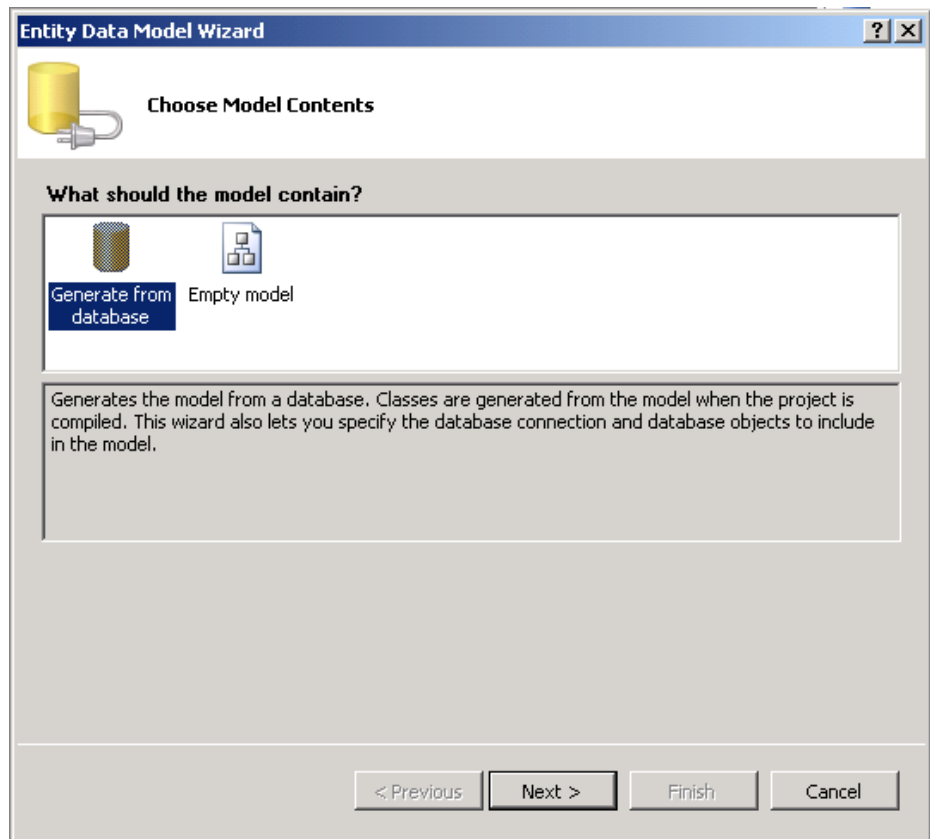
Please note that adding the Entity Data Model to the web project isn't your only option: you may prefer to give it a project of its own. The only requirement is that it should ultimately reside in an assembly that is deployed server-side.

We'll name to indicate its function in the app, and the database to which it will map:

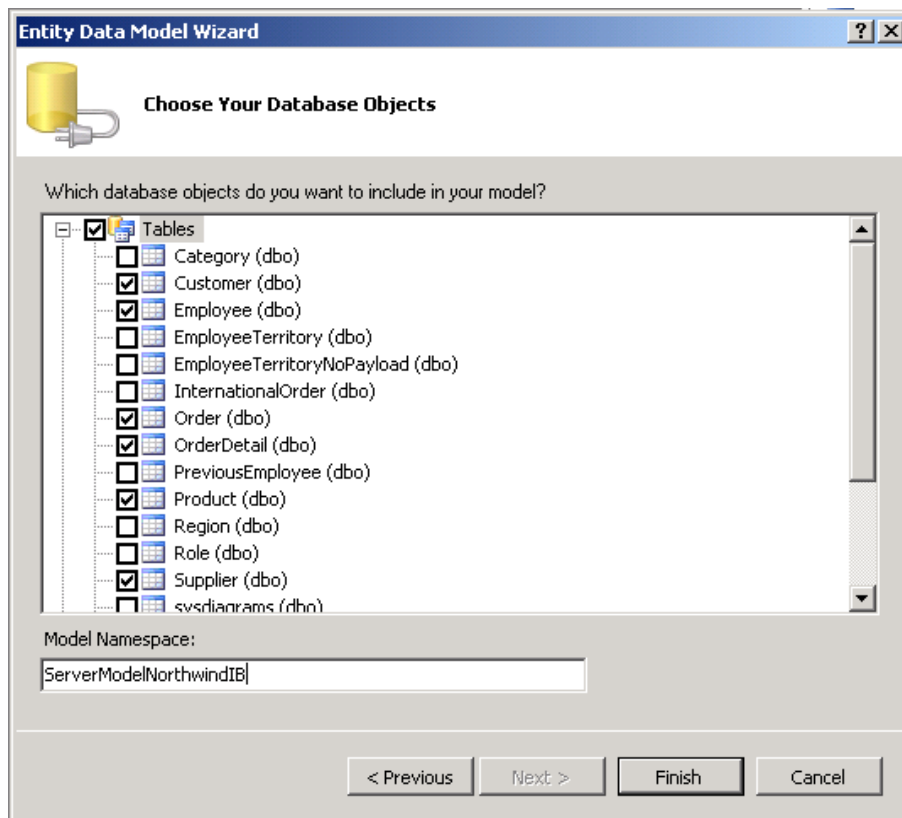
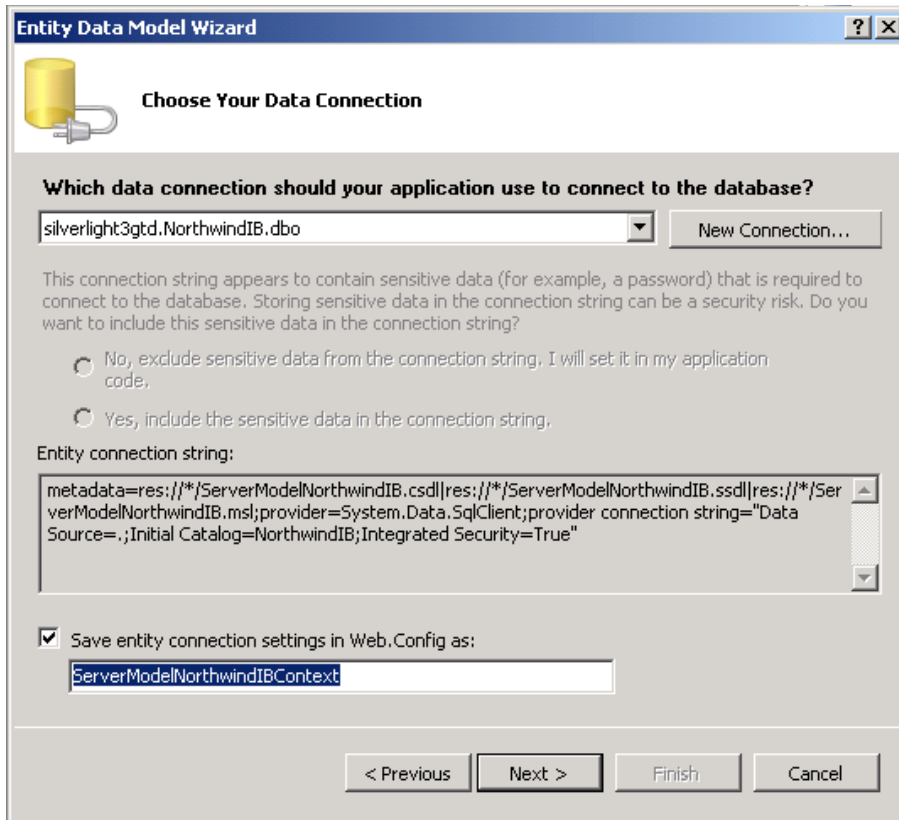


Please note: We strongly suggest, for your first pass through this tutorial, that you use the *NorthwindIB* database and follow the tutorial quite literally, as written. If you wish to use a different database or make other departures from the tutorial, do so on your *second* pass through. We've found this approach not only produces the greatest success and satisfaction, but also saves you time.

We'll generate the app from a database...

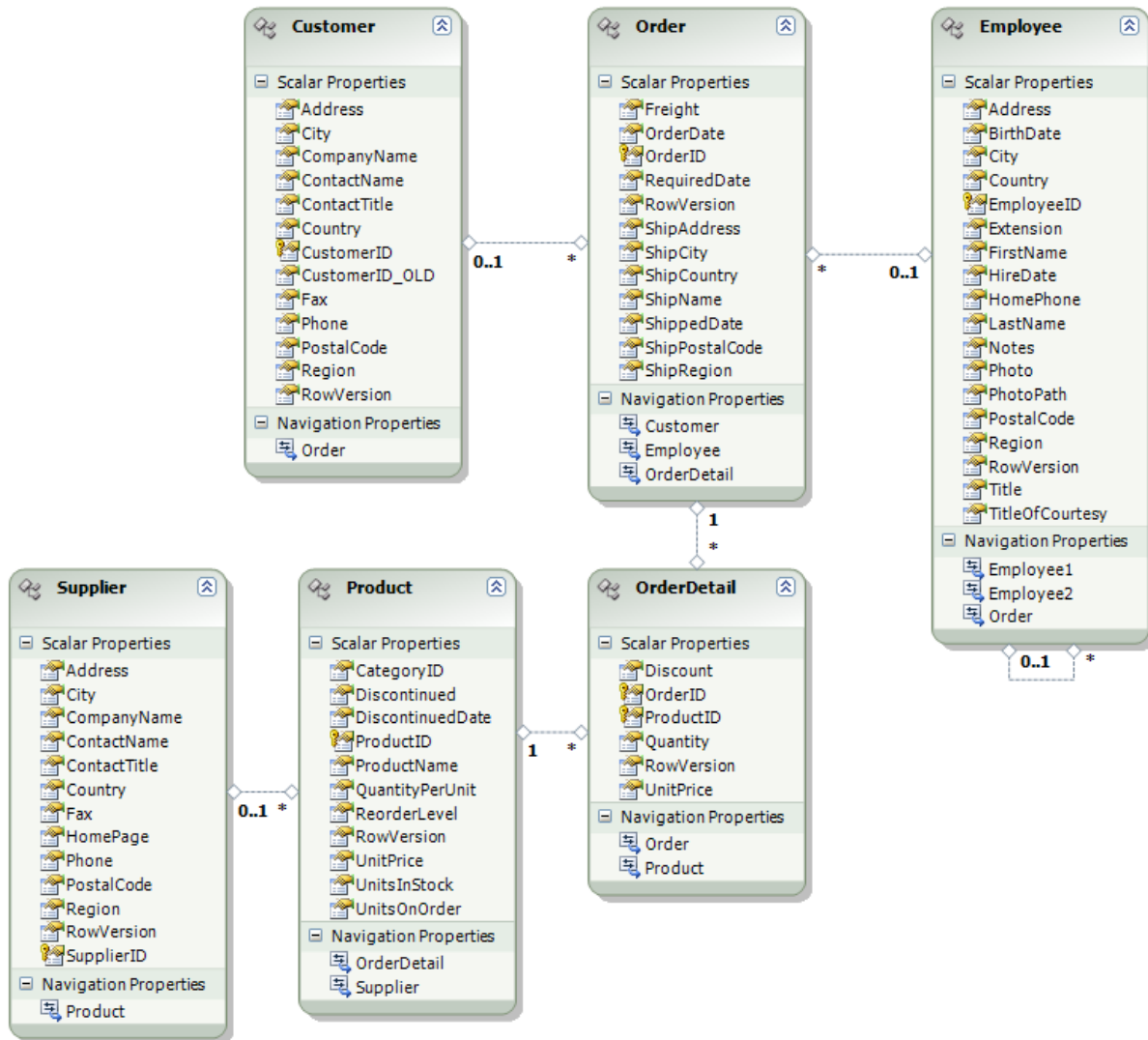


... target the NorthwindIB database, name our connection settings...



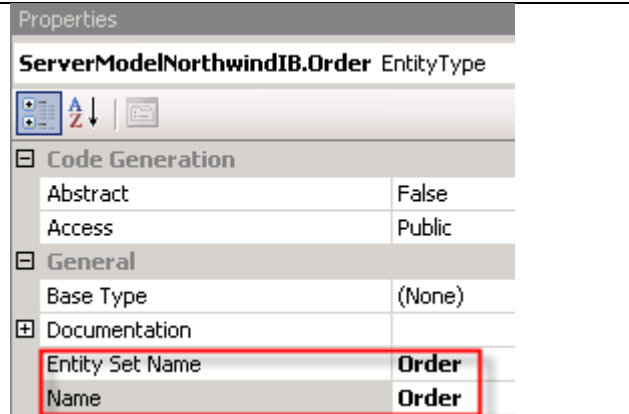
...select the tables we want mapped...

...and click <Finish>. The wizard cranks out the Entity Data Model, which looks like the following:



There are a few naming problems with the model as generated by the EDM wizard. For one, it's impossible to distinguish a navigation property that returns a collection from one that returns a single related object. The Order property that appears on both the Customer and Employee entities returns, in each case, a *collection* of Orders. So does OrderDetail in the Order type. But Customer and Employee in the Order type return single objects, not collections. It would be helpful if the pluralization of the names reflected the actual content of the properties.

Also, if we inspect the Properties for any of the entities defined in our model, we'd see that the name of the Entity Set that will hold instances of that entity is the same as the name of the type itself: again, not so helpful, as we might well prefer that the set names be plural and only the type name singular.



We could clean those things up using the EDM designer, but as it so happens, it's quite a bit easier to do in the DevForce Object Mapper, so we'll defer that cleanup and proceed straight to step three. For now, we just save our Entity Data Model as is.

Step3. Create your DevForce Domain Model

Now that we have an Entity Data Model, we're ready to create our DevForce-generated Domain Model. Before we do that, though, let's take a few moments to answer a couple of basic questions:

- Why two models? and
- What do I get in a DevForce Silverlight app that I wouldn't get in *any* Silverlight app?

The ADO.NET Entity Data Model provided by Microsoft provides a great structure for mapping objects to a relational database, and as used by the Microsoft Entity Framework, permits you, the developer, to completely offload the work of writing data access code. Not only do you no longer need to master the various dialects of SQL supported by different DBMS vendors: you no longer need to write SQL in *any* form. In your application code, you talk to an object model that can be designed to fit your application like a glove. You are buffered from the design details of the back-end database, which may be at once a poor fit to your application and also difficult or impossible to change.

Unfortunately the Entity Data Model can't be used in a Silverlight application, because there's nothing in Silverlight that knows how to do anything useful with it. (The Entity Framework isn't part of Silverlight.)

Darn!

But wait...DevForce to the rescue!

DevForce leverages the power and benefits of the Entity Data Model and the Entity Framework *on the application server*, but gives you a model and a surrounding apparatus that can be used, in very powerful ways, *within your Silverlight client*.

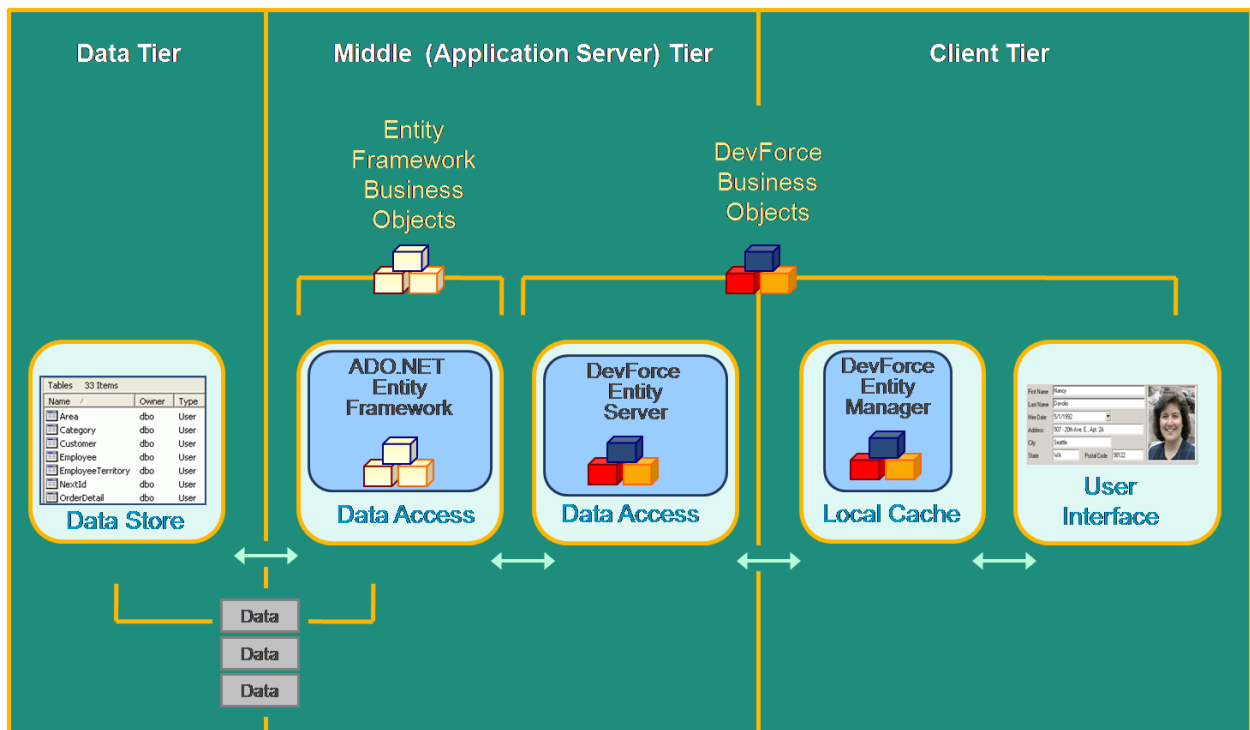
DevForce includes an EntityManager, similar in function to the Entity Framework'sObjectContext, but much more powerful. This EntityManager resides client-side and maintains a business object cache that you can query using LINQ!

But the DevForce-generated DomainModel will also be used server-side. DevForce uses the model server-side and client-side in ways that are appropriate to the two different environments. *You* have one business model to maintain – not two.

The figure below shows how DevForce enables you to leverage the power of the Entity Framework in your Silverlight application. The Devforce Entity Manager maintains a *queryable client-side cache* of business objects retrieved from the back-end data store(s).

DevForce-enabled LINQ queries can be directed *against the client-side cache or against back-end data stores.*

When directed against the data store, they are automatically translated by the DevForce Entity Server into LINQ-to-Entities queries that the Entity Framework can process. The Entity Framework then generates the necessary SQL to retrieve the data from the database, which it delivers to the DevForce Entity Server. The Entity Server very efficiently ships the data to the client where it is converted into DevForce business objects of the types defined in your DevForce Domain Model.



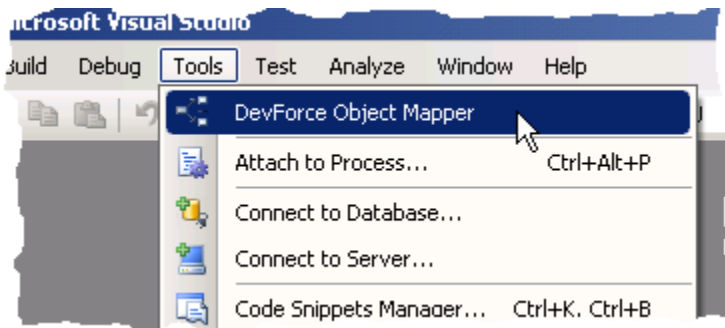
Note, by the way, that unlike the Entity Data Model, the DevForce Domain Model doesn't limit you to using a single database as a data source. The DevForce Domain Model can encompass any number of Entity Data Models, each mapping to a different relational database. Even further, it can use DevForce-

generated Entity Data Models that are based on *web services* – something entirely beyond the standard Entity Data Model’s capabilities!

Our Entity Data Model(s) will get a few automated modifications from DevForce, but will otherwise remain as generated. The purpose of the modifications will be to link the DomainModel against which you write all of your application code to the powerful data retrieval and storage capabilities of the Microsoft Entity Framework.

Okay, enough with the background. Let’s see just how easy it is to create our DevForce DomainModel.

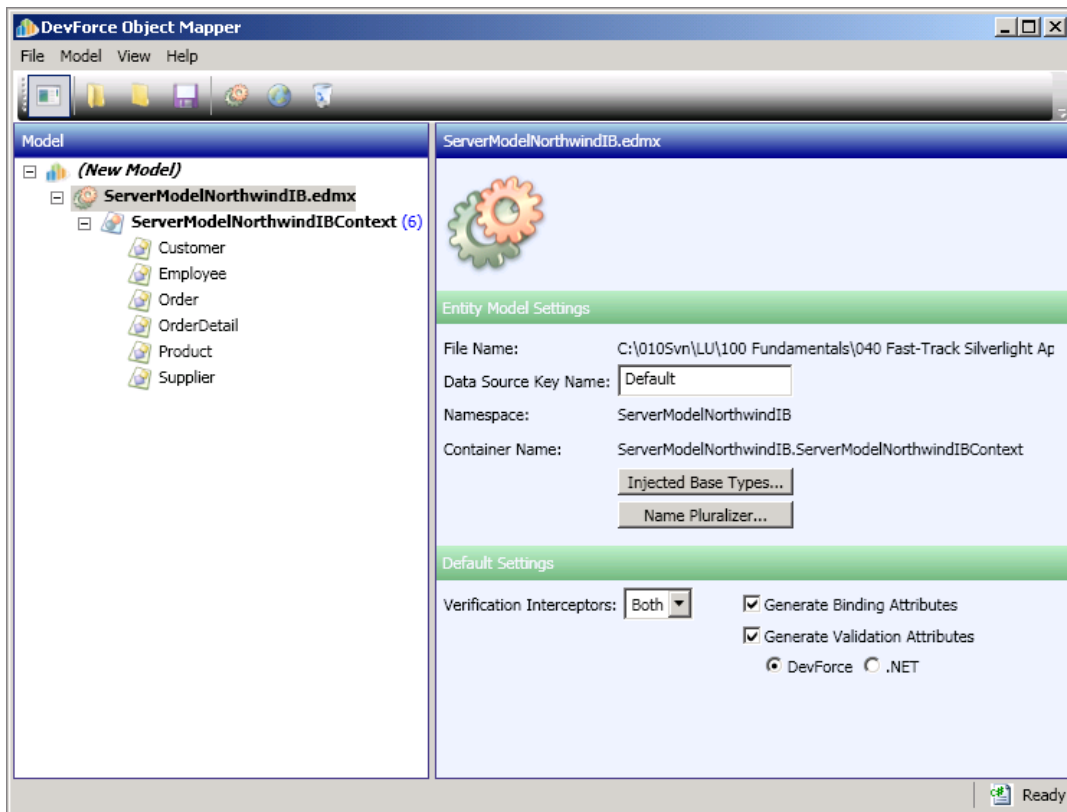
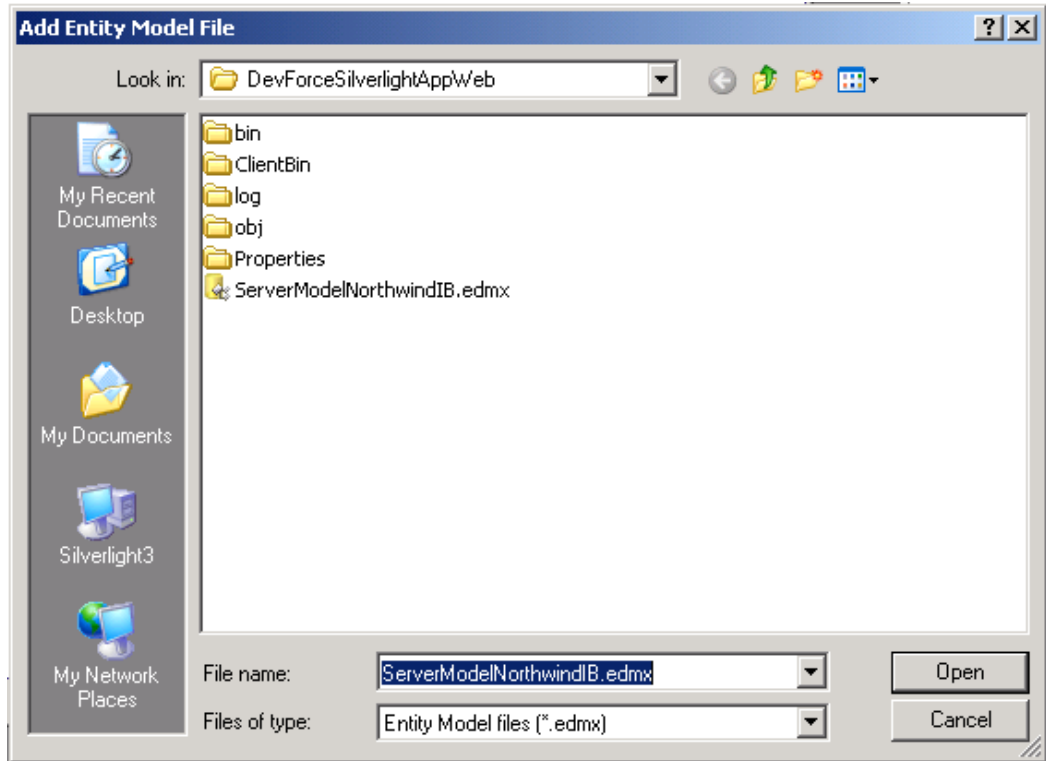
Start by launching the DevForce Object Mapper from the Tools menu in Visual Studio:



The Object Mapper launches. By pulling down the Model menu or right-clicking the (New Model) node in the left panel, we can begin the process of linking our Domain Model to an existing Entity Data Model:

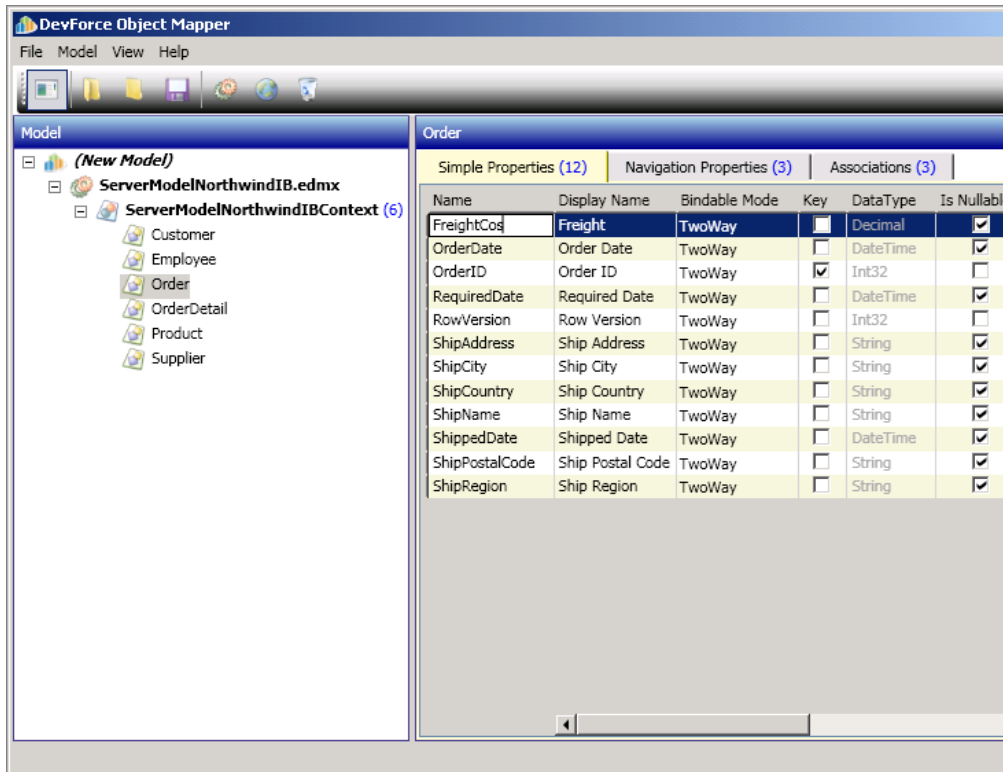
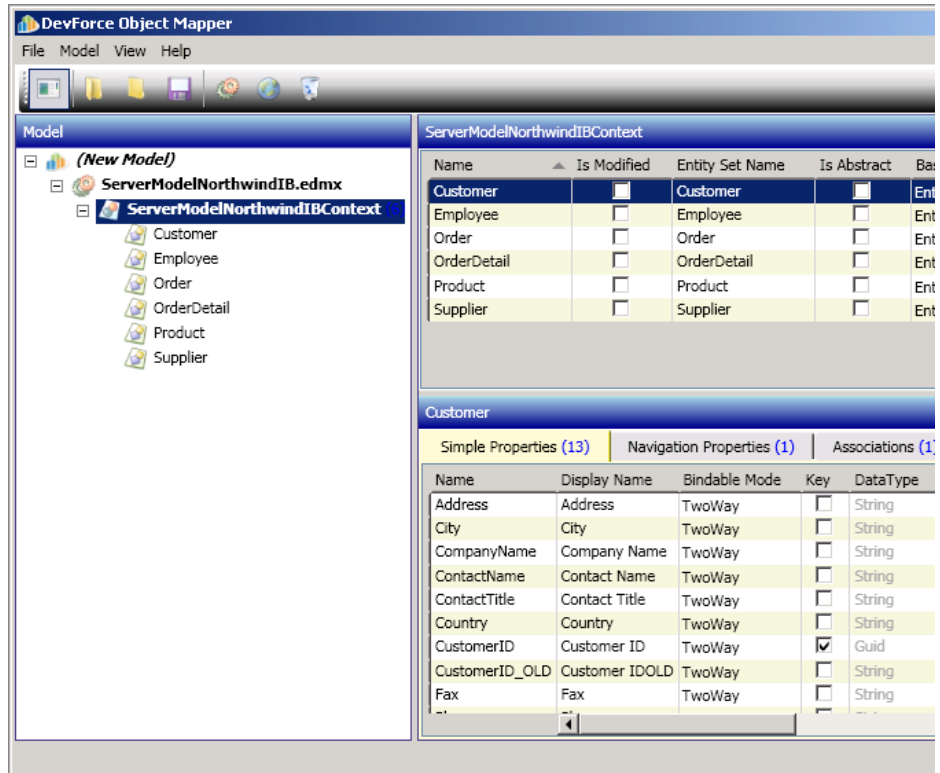


Since we have only a single Entity Data Model in our solution, the Object Mapper automatically finds it. You have only to confirm that this is an Entity Data Model you'd like to include in your Domain Model by clicking the <Open> button.



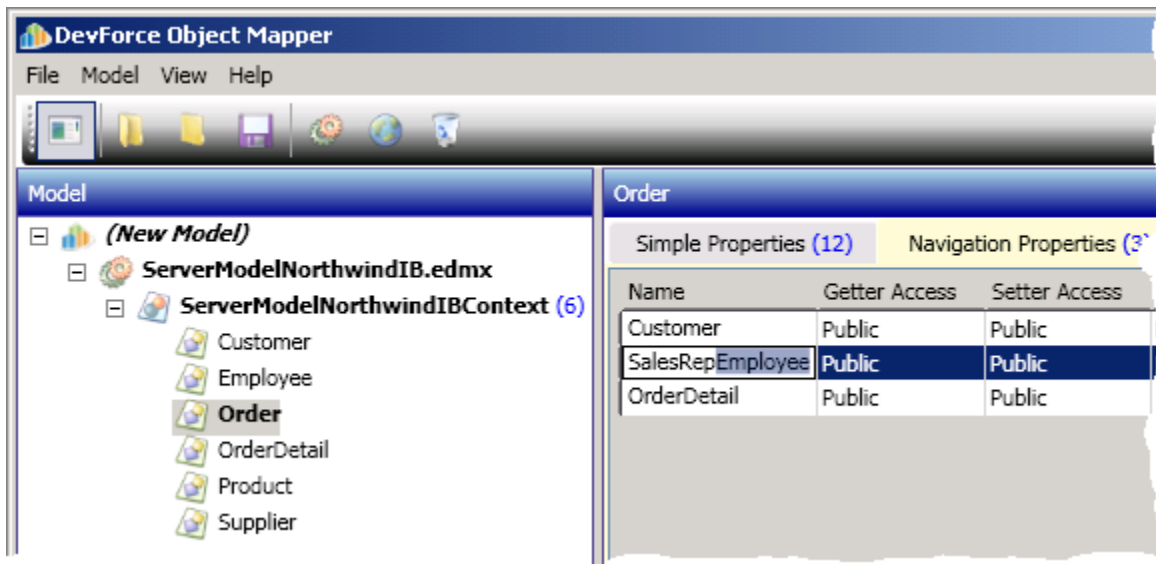
The Object Mapper will quickly mine the EDM for information and then display its structure.

By selecting the ServerModelNorthwindIB Context node in the tree, you can see (and edit) much more of the detail about the EDM.

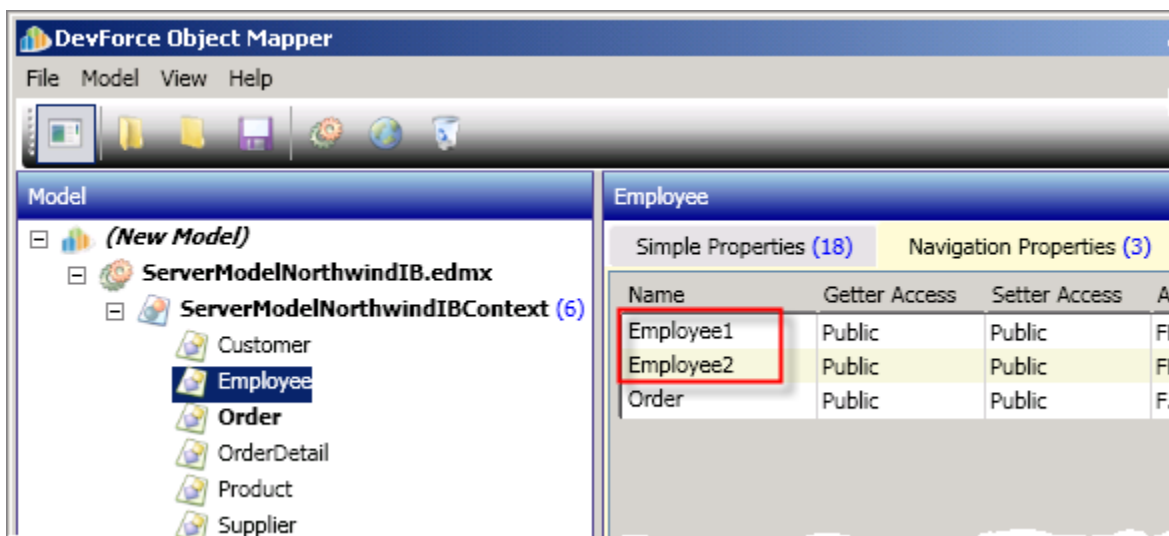


We'll drill into the Order type and change the name of the *Freight* property (based on a *Freight* column in the Order table in the NorthwindIB database) to *FreightCost*.

Similarly, we'll change the Navigation Property named Employee to "SalesRep" to reflect the Employee's actual role with respect to an Order:



The navigation properties Employee1 and Employee2 (on the Employee type) were generated because the EDM wizard found a self-referencing relationship on the Employee table. That relationship reflects the recursive hierarchy among Employees: any Employee reports to a single manager, but can be the manager for many other Employees. The EDM wizard did the best it could naming the needed navigation properties...

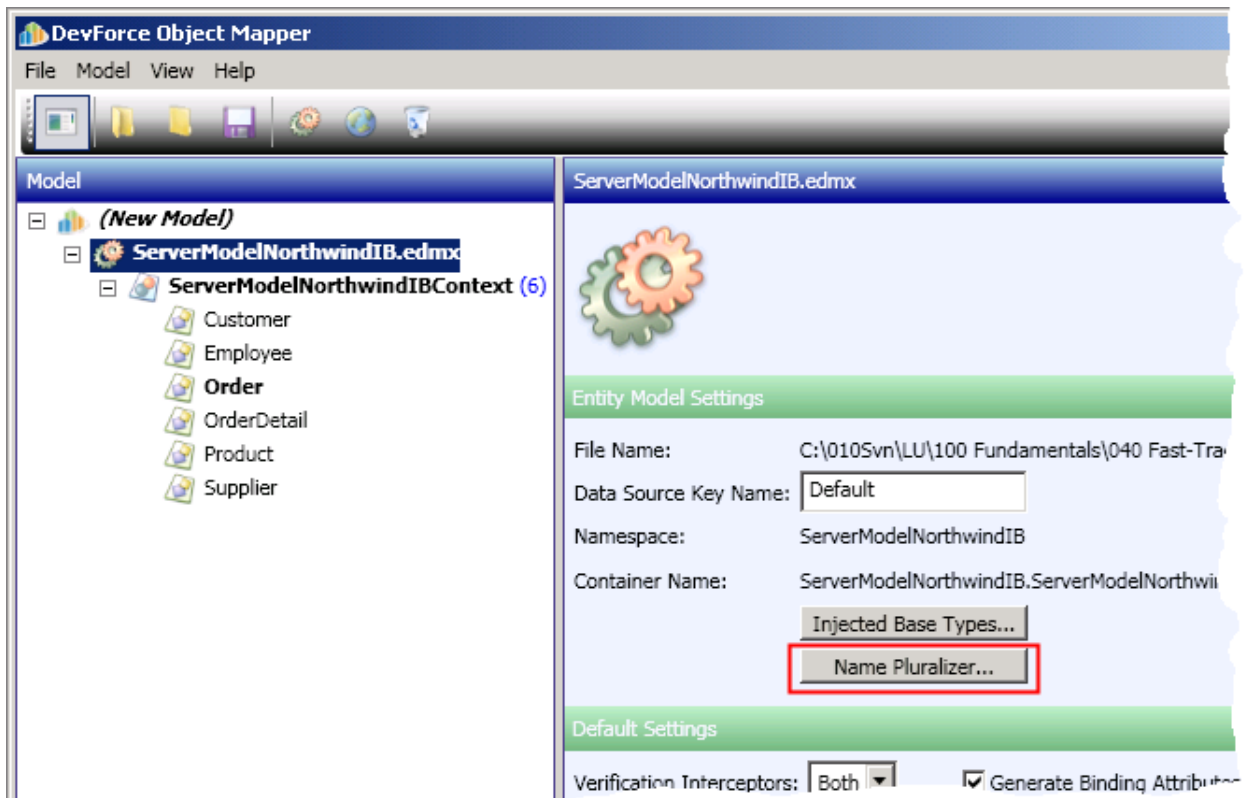


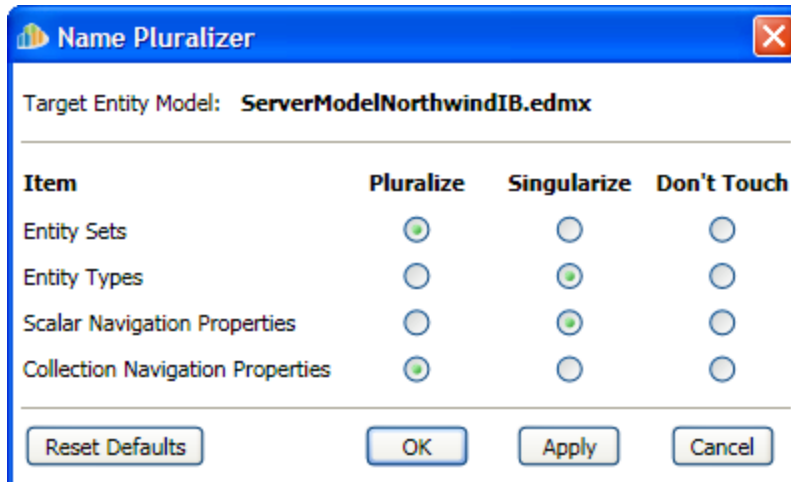
... but we'd really like to do better. Problem is, we can't easily tell which one of the navigation properties will return the current Employee's single manager, and which will return the collection of her direct reports.

We've got other annoying little naming problems in our model. If you re-examine one of the earlier screenshots you can see that entity sets are named the same as entity types:

Name	Is Modified	Entity Set Name
Customer	<input type="checkbox"/>	Customer
Employee	<input type="checkbox"/>	Employee
Order	<input checked="" type="checkbox"/>	Order
OrderDetail	<input type="checkbox"/>	OrderDetail
Product	<input type="checkbox"/>	Product
Supplier	<input type="checkbox"/>	Supplier

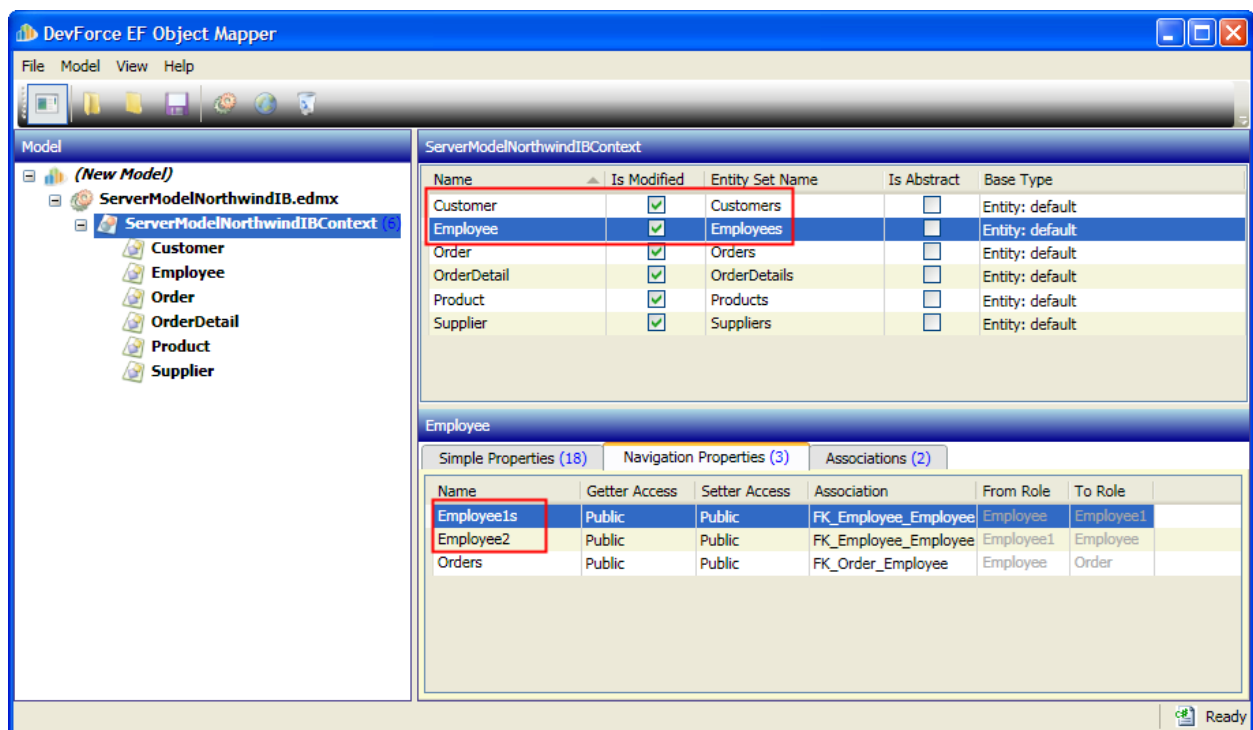
We'd really like to distinguish the set names from the name of the types they contain. We could do it one at a time, but the Object Mapper has a handy little tool to fix all of our pluralization-related problems at a single stroke: the Name Pluralizer on the detail screen for the .edmx node:





We'll just accept the default settings and click <OK>.

The Entity Sets now have plural names...



...and so do the navigation properties that return collections.

Now it's easy to see how to change our navigation properties for the management hierarchy:

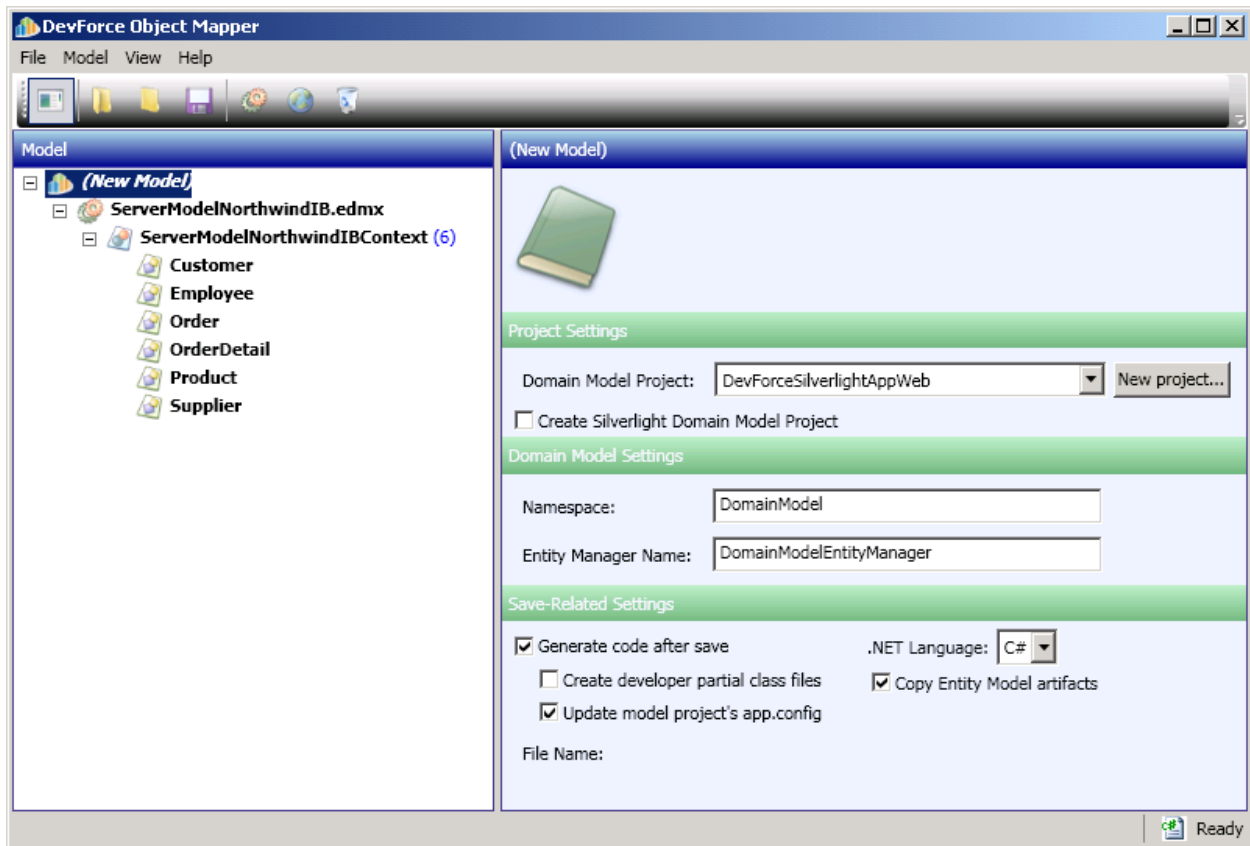
Employee			
Simple Properties (18)		Navigation Properties (3)	
Name	Getter Access	Setter Access	As
Employee1s	Public	Public	FK
Employee2	Public	Public	FK
Orders	Public	Public	FK

=>

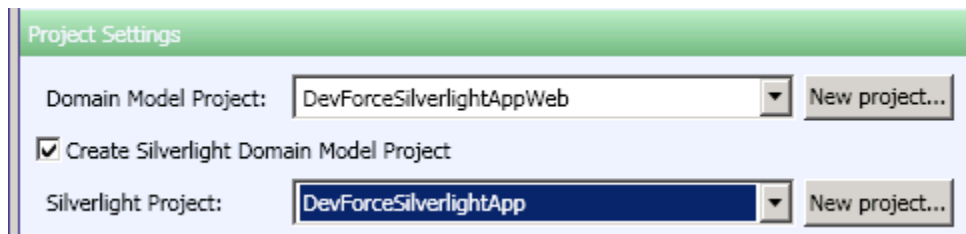
Employee			
Simple Properties (18)		Navigation Properties (3)	
Name	Getter Access	Setter Access	As
DirectReports	Public	Public	FK
Manager	Public	Public	FK
Orders	Public	Public	FK

There are many other settings we can change in the Object Mapper, but let's say we decide we've done enough to generate our model code and get started with other aspects of our development process. (We can always come back to the Object Mapper for further work.)

We select the Domain Model node in the model tree:

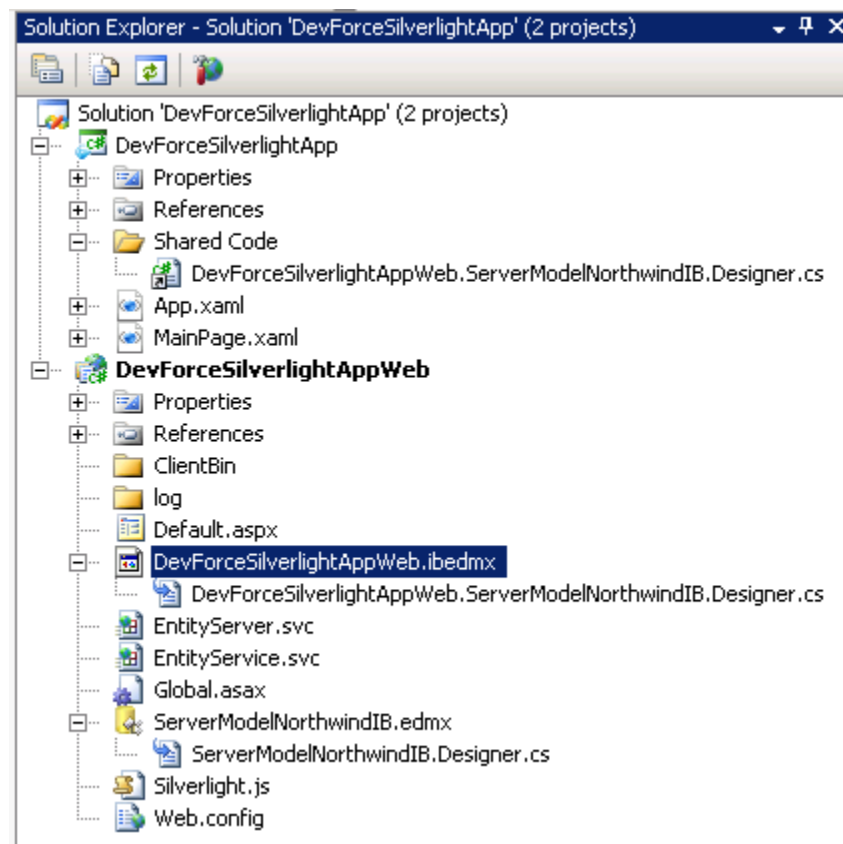


So that the Object Mapper will generate, in addition to the main model in the web project, a shadow model in the Silverlight project, check the checkbox labelled "Create Silverlight Domain Model Project" and set the target project to "DevForceSilverlightApp":



(Don't worry: if you forget to do this, you can just regenerate the model, this time with the box checked.)

If we accept the default location suggested by the Object Mapper for the Domain Model Project, leave the "Create developer partial class files" CheckBox unchecked, and click the "Save Domain Model" tool bar button, we'll end up with a solution that looks like the following:



The Object Mapper generated a code file:

`DevForceSilverlightAppWeb.ServerModelNorthwindIB.Designer.cs`¹

into the web project, and placed a linked copy of the same code file into the Silverlight project, so that the same code will be compiled into the Silverlight assembly created from that project.

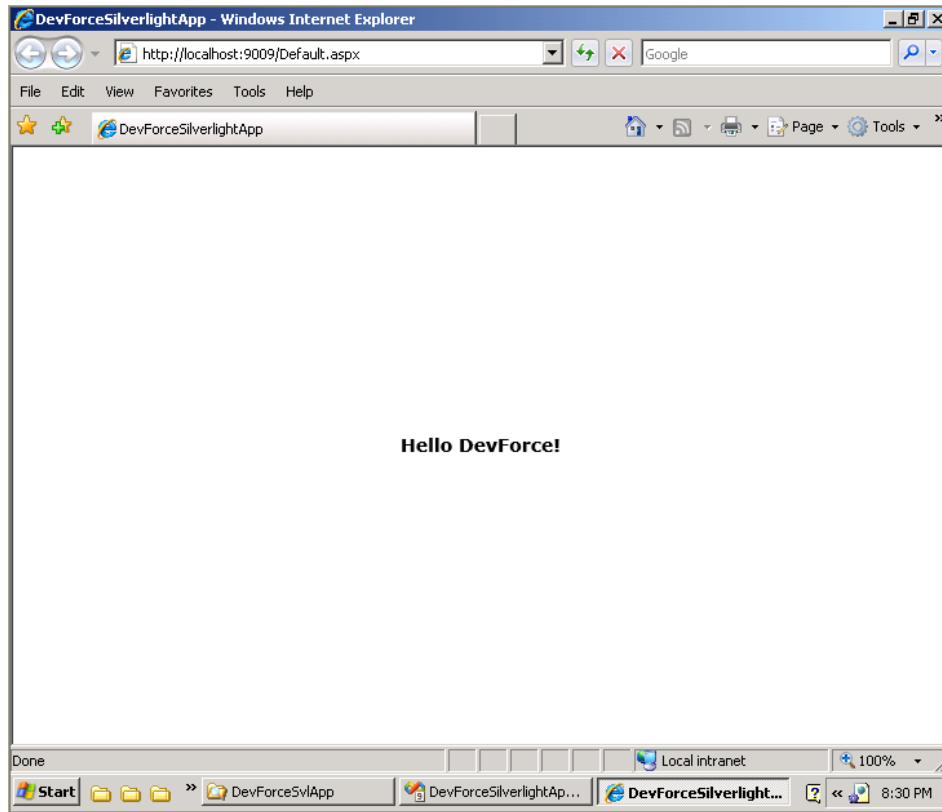
There is a great deal more you can do in the Object Mapper: defining and assigning base types to create a business object inheritance hierarchy; allowing or disallowing nulls; enforcing column widths on strings; setting up facilities for diagnosing concurrency conflicts; and so forth. But your model doesn't have to get any more complicated than you need it to be, and you can introduce detail incrementally. The Object Mapper round trips, so you can return to it throughout your application development cycle to add, remove, or modify your objects.

You'll almost certainly want to add custom business logic to your developer partial classes (housed in those `Customer.cs`, `Employee.cs`, and `Order.cs` files) including custom properties and methods, property interceptors, validation logic (using DevForce's very robust Verification features), security logic, event handlers, and the like. But our model is ready now for sophisticated data retrieval and storage, so let's slap on a UI and let 'er rip.

Step4. Create Your User Interface in the Configured Silverlight Project

Actually, we already have a UI of sorts. Our Silverlight project got generated with a XAML `MainPage` (and associated "code behind" file) that is set as the application's start page. We can, in fact, run the application right now. If we do, we'll see this:

¹ For VB users, code files will have a `.vb` extension: e.g., `DevForceSilverlightAppWeb.ServerModelNorthwindIB.Designer.vb`. We'll only refer to the `.cs` files in this article, but you can do the translation.



It's comforting to see that our application already runs, but let's put something together that uses those powerful data persistence facilities. For the purpose of this tutorial, the UI whose construction we step through will be pretty spartan, but nevertheless handle a fair amount of data.

Important! The user interface added with the following XAML uses the Silverlight *DataForm* control. If you're following along and reproducing this solution in code, you will need the Silverlight 3 Toolkit installed on your machine for the DataForm to be available. You can download the Silverlight 3 Toolkit here:

<http://silverlight.codeplex.com/Release/ProjectReleases.aspx?ReleaseId=24246>

Note also that the *Silverlight 3 Toolkit* is a different piece of software than the *Silverlight 3 Tools* (which contains the basic Silverlight SDK). See the DevForce Release Notes, Prerequisites section, for more information on both.

On a separate note: some readers, depending upon your version of Adobe Acrobat and other factors, find that code snippets copied from this document and pasted into Visual Studio lose their end-of-line characters, requiring those to be re-added in the Visual Studio code window. If you have this problem, remember that you also have a completed code solution for the *Four Simple Steps* app from which you can copy and paste code, while still following along in the tutorial.

Replace the existing XAML markup in the MainPage with the following:

XML

```
<UserControl x:Class="DevForceSilverlightApp.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:navigation="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Navigatio
n"
  xmlns:data="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
  xmlns:df="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data.Data
Form.Toolkit"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  Width="Auto" Height="Auto" mc:Ignorable="d" Background="{x:Null}">
<Grid x:Name="LayoutRoot" Margin="20,20,20,20" >
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width=".1*"/>
    <ColumnDefinition Width=".9*"/>
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height=".1*"/>
    <RowDefinition Height=".35*"/>
    <RowDefinition Height=".40*"/>
    <RowDefinition Height=".15*"/>
  </Grid.RowDefinitions>
  <TextBlock
    x:Name="_layoutRootTitleTextBlock"
    Grid.Row="0"
    Grid.Column="0"
    Grid.ColumnSpan="2"
    FontWeight="Bold"
    FontSize="20"
    VerticalAlignment="Center" Foreground="#FF000000"
  ><Run Text="NorthwindIB Employees with Orders"/>
</TextBlock>

  <df:DataForm x:Name="_employeeDataForm"
    Grid.Row="1"
    Grid.Column="1"
    MinHeight="80"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    AutoGenerateFields="True"
    Header=""
    Margin="0,0,0,10"
  >
</df:DataForm>
</Grid>
</UserControl>
```

```

</df:DataForm>
<data:DataGrid
  x:Name="_ordersDataGrid"
  Grid.Row="2"
    Grid.Column="1"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    AutoGenerateColumns="True"
    MinWidth="250"
    MinHeight="80"
    Background="#FFB5BAB5"
    Margin="0,0,20,0"
  >
</data:DataGrid>
<ScrollViewer
  x:Name="_statusMsg_ScrollViewer"
    Height="60"
    Grid.Row="3"
    Grid.Column="1"
  Margin="0,0,20,0"
  >
  <TextBlock x:Name="_statusTextBlock"/>
</ScrollViewer>

</Grid>
</UserControl>

```

We've added essentially four user interface controls:

1. A TextBlock to give the form a title
2. A Silverlight DataForm
3. A Silverlight DataGrid, and
4. A TextBlock wrapped in a ScrollViewer to display status information.

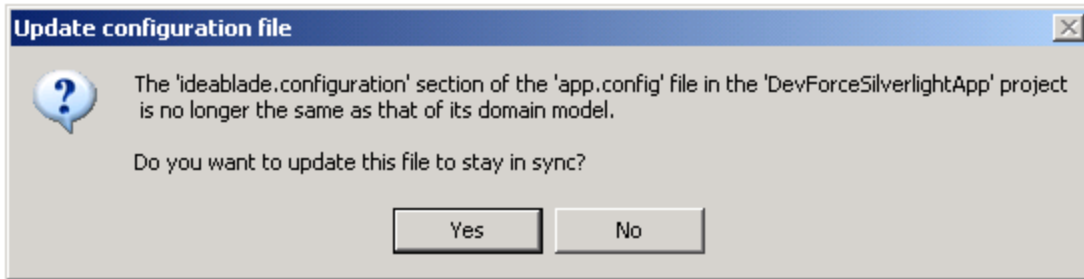
Both the DataForm (which will display one Employee at a time in form view) and the DataGrid (which will display the Orders taken by that Employee) are set to determine automatically what properties to display from the entities fed to them. They don't even know the types of those entities (though we do).

The page uses some assemblies we need to will need to reference in the Silverlight project (DevForceSilverlightApp):

- System.Windows.Controls
- System.Windows.Controls.Data
- System.Windows.Controls.Data.DataForm.Toolkit
- System.Windows.Controls.Navigation

Add those references.

If build the project, you may see the following message:



This message is displayed during builds whenever DevForce notices that the DevForce configuration information in the Silverlight project doesn't match that in the project where the Domain Model resides. Say yes, and DevForce will sync up the DevForce information in the *app.config* file in the Silverlight project with the information in the *web.config* file in the web project.

MainPage will also need some code behind to work. Here's that:

C#

```
#region Using Statements
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

using IdeaBlade.EntityModel;
using DomainModel;
using System.Collections.ObjectModel;
using System.Windows.Navigation;
using System.Collections;
using System.Text;
#endregion Using Statements

namespace DevForceSilverlightApp {
    public partial class MainPage : UserControl {

        #region ctor and startup
        public MainPage() {
            InitializeComponent();
            Loaded += Page_Loaded;
        }

        void Page_Loaded(object sender, RoutedEventArgs e) {
            CreateEntityManager();
            ConfigureDataForm();
            ConfigureDataGrid();
            Login();
        }
        #endregion ctor and startup

        #region Login
        private void Login() {
```

```

        var cred = new LoginCredential("demo", "demo", "demo");
        WriteMessage("Logging in ...");
        _mgr.LoginAsync(cred, LoggedIn, null);
    }

private void LoggedIn(LoginEventArgs args) {
    if (args.Error != null) {
        WriteMessage(args.Error.Message);
    }
    else {
        WriteMessage("Logged in");
    }
    LoadData();
}

#endregion Login

#region Load Data
private void CreateEntityManager() {
    WriteMessage("Creating EntityManager ...");
    _mgr = new DomainModelEntityManager(false);
}

private void LoadData() {
    _mgr.ExecuteQueryAsync(_mgr.Employees, GotEmployees, null);
}

private void GotEmployees(EntityFetchedEventArgs args) {
    if (args.Error != null) {
        WriteMessage(args.Error.Message);
    }
    else {
        foreach (Employee aEmployee in args.Result) {
            _employees.Add(aEmployee);
        }
        // Set CurrentItem to first Employee so form will flesh out
        if (_employees.Count > 0) {
            _employeeDataForm.CurrentItem = _employees[0];
        }
        ReportFetchCount(args.Result, "Employee");
    }
}

void Orders_PendingEntityListResolved(object sender,
PendingEntityListResolvedEventArgs<Order> e) {
    PopulateOrderList(e.ResolvedEntities);
    ReportFetchCount(e.ResolvedEntities, "Order");
    ((RelatedEntityList<Order>)sender).PendingEntityListResolved -=
Orders_PendingEntityListResolved;
}
#endregion Load Data

#region Configure Environment
private void ConfigureDataForm() {
    _employeeDataForm.CommandButtonsVisibility =
DataFormCommandButtonsVisibility.Navigation;
    _employeeDataForm.ItemsSource = _employees;
    _employeeDataForm.CurrentItemChanged += new
EventHandler<EventArgs>(_employeeDataForm_CurrentItemChanged);
}

void _employeeDataForm_CurrentItemChanged(object sender, EventArgs e) {
    Employee currentEmployee = (Employee)_employeeDataForm.CurrentItem;

```



```

        WriteMessage("Now viewing Employee " + (currentEmployee).LastName + ",
" + (currentEmployee).FirstName);
        _orders.Clear();
        if (currentEmployee.Orders.IsPendingEntityList) {
            currentEmployee.Orders.PendingEntityListResolved +=
                new
EventHandler<PendingEntityListResolvedEventArgs<Order>>(Orders_PendingEntity
ListResolved);
        }
        else {
            PopulateOrderList(currentEmployee.Orders);
        }
    }

    private void PopulateOrderList(IEnumerable<Order> orders) {
        foreach (Order anOrder in orders) {
            _orders.Add(anOrder);
        }
    }

    private void ConfigureDataGrid() {
        this._ordersDataGrid.ItemsSource = _orders;
    }
#endregion Configure Environment

#region Methods to Display Status
private void ReportFetchCount(IEnumerable result, string entityType) {
    int entitiesRetrieved = ((ICollection)result).Count;
    WriteMessage(string.Format("Retrieved {0} entities of type " +
entityType, entitiesRetrieved));
}

private void WriteMessage(string msg) {
    _msgNumber += 1;
    _statusStringBuilder.Append(String.Format("{0}",
_msgNumber.ToString("D4")) +
        " " + msg + Environment.NewLine);
    _statusTextBlock.Text = _statusStringBuilder.ToString();

    _statusMsg_ScrollView.ScrollToVerticalOffset(_statusMsg_ScrollView.Scr
lableHeight);
}
#endregion Methods to Display Status

#region Private Fields
ObservableCollection<Employee> _employees = new
ObservableCollection<Employee>();
ObservableCollection<Order> _orders = new ObservableCollection<Order>();
DomainModelEntityManager _mgr;
int _msgNumber = 0;
StringBuilder _statusStringBuilder = new StringBuilder();
#endregion Private Fields
}
}

```

VB

```

#Region "Imports Statements"
Imports System
Imports System.Collections.Generic
Imports System.Linq

```

```

Imports System.Net
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Documents
Imports System.Windows.Input
Imports System.Windows.Media
Imports System.Windows.Media.Animation
Imports System.Windows.Shapes

Imports IdeaBlade.EntityModel
Imports DevForceSilverlightApp.DomainModel
Imports System.Collections.ObjectModel
Imports System.Windows.Navigation
Imports System.Collections
Imports System.Text
#End Region 'Imports Statements

Partial Public Class MainPage
    Inherits UserControl

    #Region "ctor and startup"
        Public Sub New()
            InitializeComponent()
            AddHandler Loaded, AddressOf Page_Loaded
        End Sub

        Private Sub Page_Loaded(ByVal sender As Object, ByVal e As
RoutedEventArgs)
            CreateEntityManager()
            ConfigureDataForm()
            ConfigureDataGrid()
            Login()
        End Sub
    #End Region ' ctor and startup

    #Region "Login"
        Private Sub Login()
            Dim cred = New LoginCredential("demo", "demo", "demo")
            WriteMessage("Logging in ...")
            _mgr.LoginAsync(cred, AddressOf LoggedIn, Nothing)
        End Sub

        Private Sub LoggedIn(ByVal args As LoginEventArgs)
            If args.Error IsNot Nothing Then
                WriteMessage(args.Error.Message)
            Else
                WriteMessage("Logged in")
            End If
            LoadData()
        End Sub
    #End Region ' Login

    #Region "Load Data"
        Private Sub CreateEntityManager()
            WriteMessage("Creating EntityManager ...")
            _mgr = New DomainModelEntityManager(False)
        End Sub

        Private Sub LoadData()
            _mgr.ExecuteQueryAsync(_mgr.Employees, AddressOf GotEmployees,
Nothing)
        End Sub
    #End Region

```

```

Private Sub GotEmployees(ByVal args As EntityFetchedEventArgs)
    If args.Error IsNot Nothing Then
        WriteMessage(args.Error.Message)
    Else
        For Each aEmployee As Employee In args.Result
            _employees.Add(aEmployee)
        Next aEmployee
        ' Set CurrentItem to first Employee so form will flesh out
        If _employees.Count > 0 Then
            _employeeDataForm.CurrentItem = _employees(0)
        End If
        ReportFetchCount(args.Result, "Employee")
    End If
End Sub

Private Sub Orders_PendingEntityListResolved(ByVal sender As Object,
ByVal e As PendingEntityListResolvedEventArgs(Of Order))
    PopulateOrderList(e.ResolvedEntities)
    ReportFetchCount(e.ResolvedEntities, "Order")
    RemoveHandler (CType(sender, RelatedEntityList(Of
Order))).PendingEntityListResolved, AddressOf
Orders_PendingEntityListResolved
End Sub
#End Region ' Load Data

#Region "Configure Environment"
Private Sub ConfigureDataForm()
    _employeeDataForm.CommandButtonsVisibility =
DataForm.CommandButtonsVisibility.Navigation
    _employeeDataForm.ItemsSource = _employees
    AddHandler _employeeDataForm.CurrentItemChanged, AddressOf
_employeeDataForm_CurrentItemChanged
End Sub

Private Sub _employeeDataForm_CurrentItemChanged(ByVal sender As Object,
ByVal e As EventArgs)
    Dim currentEmployee As Employee =
CType(_employeeDataForm.CurrentItem, Employee)
    WriteMessage("Now viewing Employee " & (currentEmployee).LastName &
", " & (currentEmployee).FirstName)
    _orders.Clear()
    If currentEmployee.Orders.IsPendingEntityList Then
        AddHandler currentEmployee.Orders.PendingEntityListResolved,
AddressOf Orders_PendingEntityListResolved
    Else
        PopulateOrderList(currentEmployee.Orders)
    End If
End Sub

Private Sub PopulateOrderList(ByVal orders As IEnumerable(Of Order))
    For Each anOrder As Order In orders
        _orders.Add(anOrder)
    Next anOrder
End Sub

Private Sub ConfigureDataGrid()
    Me._ordersDataGrid.ItemsSource = _orders
End Sub
#End Region ' Configure Environment

#Region "Methods to Display Status"
Private Sub ReportFetchCount(ByVal result As IEnumerable, ByVal

```

```

entityType As String)
    Dim entitiesRetrieved As Integer = (CType(result,
ICollection)).Count
    WriteMessage(String.Format("Retrieved {0} entities of type " &
entityType, entitiesRetrieved))
    End Sub

    Private Sub WriteMessage(ByVal msg As String)
        _msgNumber += 1
        _statusStringBuilder.Append(String.Format("{0}",
_msgNumber.ToString("D4")) & " " & msg & Environment.NewLine)
        _statusTextBlock.Text = _statusStringBuilder.ToString()

        _statusMsg_ScrollView.ScrollToVerticalOffset(_statusMsg_ScrollView.Scr
lableHeight)
    End Sub
#End Region ' Methods to Display Status

#Region "Private Fields"
    Private _employees As New ObservableCollection(Of Employee) ()
    Private _orders As New ObservableCollection(Of Order) ()
    Private _mgr As DomainModelEntityManager
    Private _msgNumber As Integer = 0
    Private _statusStringBuilder As New StringBuilder()
#End Region ' Private Fields

End Class

```

When the page loads, we create an instance of a DevForce EntityManager (which processes data transfer requests and maintains a local cache of retrieved and newly created data). Next we call a method, *ConfigureDataForm()*, to configure the DataForm's Add and Delete behaviors, set its ItemsSource, and to establish a handler for its CurrentItemChanged event. We'll want to know about that event so we can refresh the Orders grid whenever the user navigates to a different Employee.

Next, we call *ConfigureDataGrid()* to set the ItemsSource for the DataGrid.

Finally, we log in. In our sample app we're logging in with dummy credentials and not doing any actual authentication; but the login is still required before any Silverlight data transfer can take place. Also, since, in Silverlight, all interactions with the server are asynchronous, we'll wait to load data into the app until the callback -- to method *LoggedIn()* -- is received from the server-side login operation. The asynchronous request for data gets made by the *LoadData()* method.

One more important point about the code is worth noting. In the handler for the DataForm's *CurrentItemChanged* event, we wish to refresh the Orders DataGrid to show the Orders associated with the Employee to which the end user has just navigated. We can get the list of those orders as `((Employee)_employeeDataForm.CurrentItem).Orders`. But note that this is a data retrieval request; and again, all data retrieval involving a remote server is handled asynchronously in Silverlight. So we need to set up an event handler to perform any actions that need to be done once the data arrives. And since the Orders list for any given Employee is a different list than the one for any other Employee, we don't

set this handler up until we actually need to retrieve a particular Employee's list. We also remove the handler in the handler itself, once it has done its work.

Note also that, until the Orders data actually returns from the server, ((Employee)_employeeDataForm.CurrentItem).Orders is only a *pending Entity list*, rather than one containing data. It is so flagged via its `IsPendingEntityList` property.

But remember: DevForce caches data retrieved from remote servers so you don't have to keep going back and back to those servers unnecessarily. So the second time we need the Orders for a particular Employee, we no longer have to make a trip to the server for the data. In that case, the `IsPendingEntityList` flag of ((Employee)_employeeDataForm.CurrentItem).Orders is false when we test it, and we can bypass the setting up of the event handler and repopulate the `_orders` list directly.

We also added a bit of event logging machinery into this solution so you can see the flow in a scrollable status window at the bottom of the page.

Here's what the page looks like after the application has been launched, login has occurred, and the initial data has been retrieved.

NorthwindIB Employees with Orders

The screenshot shows a web application interface. At the top, there is a form with several fields: `DirectReports` (containing `IdeaBlade.EntityModel.RelatedEntityList`1[DomainModel.Employee]`), `Manager` (containing `DomainModel.Employee`), `Orders` (containing `IdeaBlade.EntityModel.RelatedEntityList`1[DomainModel.Order]`), `Address` (containing "507 - 20th Ave. E. Apt. 2A"), `BirthDate` (containing "Wednesday, December 08, 1948"), and `City` (containing "Seattle"). Below the form is a data grid with columns: `Customer`, `SalesRep`, `OrderDetails`, `FreightCost`, `OrderDate`, `OrderID`, and `RequiredDate`. The grid contains 9 rows of data. At the bottom of the application, there is a status window with the following text: "0004 Now viewing Employee Davolio, Nancy Lynn", "0005 Retrieved 9 entities of type Employee", and "0006 Retrieved 124 entities of type Order".

Customer	SalesRep	OrderDetails	FreightCost	OrderDate	OrderID	RequiredDate
DomainModel.Customer	DomainModel.Employee	IdeaBlade.EntityModel.RelatedEntityList`1[DomainModel.OrderDetail]	142.5100	7/17/1996 12:00:00 AM	10258	8/14/1996 12:00:00 AM
DomainModel.Customer	DomainModel.Employee	IdeaBlade.EntityModel.RelatedEntityList`1[DomainModel.OrderDetail]	136.5400	8/1/1996 12:00:00 AM	10270	8/29/1996 12:00:00 AM
DomainModel.Customer	DomainModel.Employee	IdeaBlade.EntityModel.RelatedEntityList`1[DomainModel.OrderDetail]	26.9300	8/7/1996 12:00:00 AM	10275	9/4/1996 12:00:00 AM
DomainModel.Customer	DomainModel.Employee	IdeaBlade.EntityModel.RelatedEntityList`1[DomainModel.OrderDetail]	76.8300	8/20/1996 12:00:00 AM	10285	9/17/1996 12:00:00 AM
DomainModel.Customer	DomainModel.Employee	IdeaBlade.EntityModel.RelatedEntityList`1[DomainModel.OrderDetail]	1.3500	8/28/1996 12:00:00 AM	10292	9/25/1996 12:00:00 AM
DomainModel.Customer	DomainModel.Employee	IdeaBlade.EntityModel.RelatedEntityList`1[DomainModel.OrderDetail]	21.1800	8/29/1996 12:00:00 AM	10293	9/26/1996 12:00:00 AM
DomainModel.Customer	DomainModel.Employee	IdeaBlade.EntityModel.RelatedEntityList`1[DomainModel.OrderDetail]	63.7900	9/12/1996 12:00:00 AM	10304	10/10/1996 12:00:00 AM
DomainModel.Customer	DomainModel.Employee	IdeaBlade.EntityModel.RelatedEntityList`1[DomainModel.OrderDetail]	7.5600	9/16/1996 12:00:00 AM	10306	10/14/1996 12:00:00 AM
DomainModel.Customer	DomainModel.Employee	IdeaBlade.EntityModel.RelatedEntityList`1[DomainModel.OrderDetail]	24.6900	9/20/1996 12:00:00 AM	10311	10/4/1996 12:00:00 AM
DomainModel.Customer	DomainModel.Employee	IdeaBlade.EntityModel.RelatedEntityList`1[DomainModel.OrderDetail]	74.1600	9/25/1996 12:00:00 AM	10314	10/23/1996 12:00:00 AM

Because we turned on the autogeneration of fields in the `DataForm` and columns in the `DataGrid`, we got a lot of bang for very little coding buck. On the other hand, we got a few things we probably didn't want, as well. The `DirectReports` property of the `Employee` returns a collection of other `Employee`s, which the `DataForm` isn't quite smart enough to display in a meaningful manner; so it simply notes that an `IdeaBlade.EntityModel.RelatedEntityList<Employee>` was returned. The value displayed for the `Orders` property has a similar explanation. The `Manager` property returns a single `Employee`, but again

the DataForm doesn't know how to display that very well, so we'll need to do a little more work to get what we want. We have the same issues in the DataGrid.

You can learn about the many powerful capabilities of the DataForm and DataGrid from many sources that you can find on the web, so we won't go into them further here. In the code sample that accompanies this article, we have included an alternative startup page (MainPageIntermediate) wherein we've gone a little bit further down the road configuring these two controls. You can launch the alternative page by adjusting the line of code in App.Xaml.cs that calls the RootVisual:

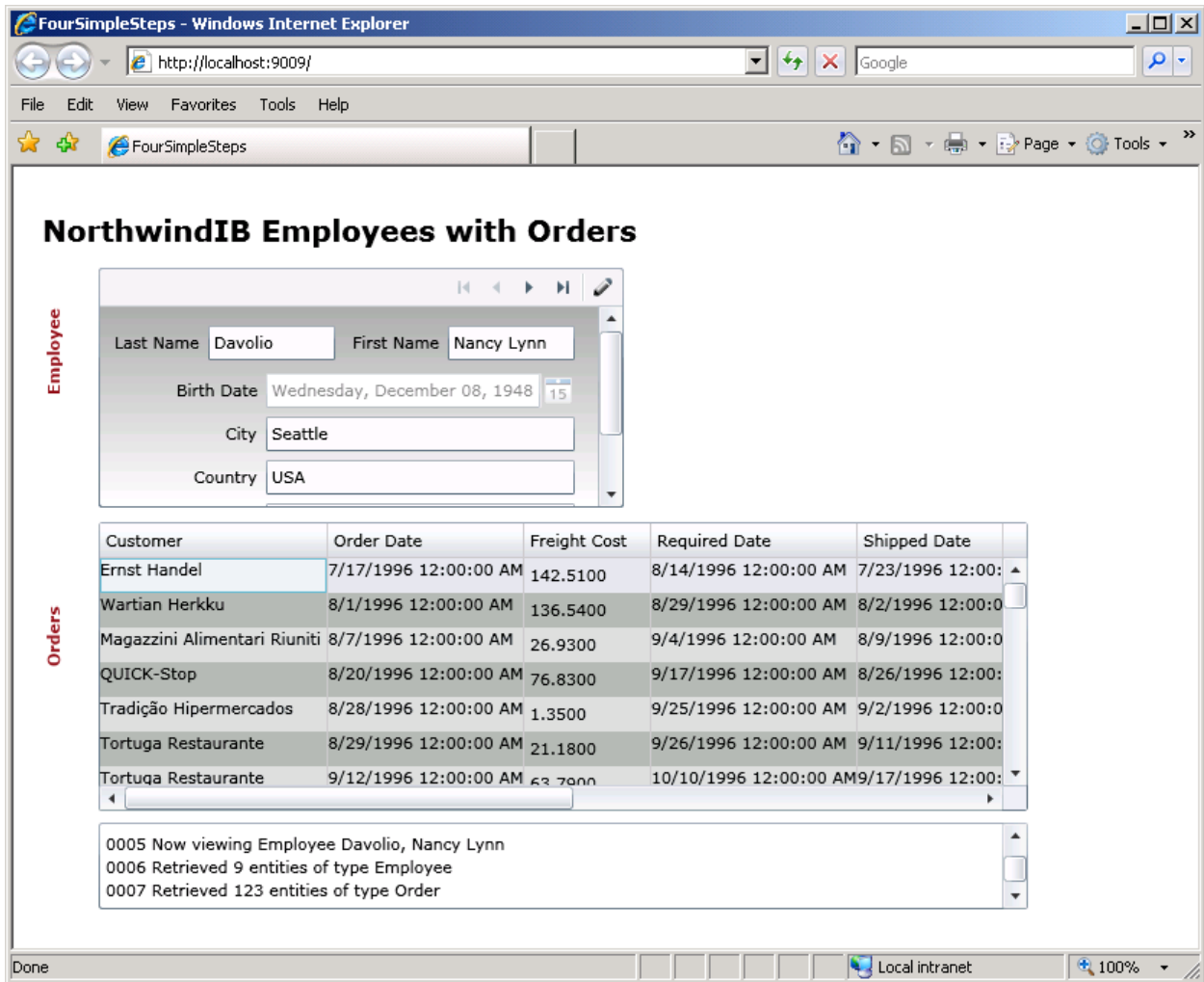
C#


```
private void Application_Startup(object sender, StartupEventArgs e) {  
    this.RootVisual = new MainPage ();  
}
```

VB

```
Private Sub Application_Startup(  
    ByVal o As Object, ByVal e As StartupEventArgs) Handles Me.Startup  
    Me.RootVisual = New MainPage ()  
End Sub
```

The alternative page looks like this:



In addition to taking greater control of the data selection and layout, this version includes date picker controls for all of the date properties and a dropdown list for the Order.Customer. These are edit-mode controls, so you don't see them in the above picture. You can use them by clicking the edit mode button on the DataForm , or by double-clicking in the Customer cell, or any date property cell, on the DataGrid.

You can examine the code and XAML for MainPageIntermediate in the Visual Studio solution that accompanies this article.

Conclusion

So there you have it: an n-tier Silverlight application in four simple steps. With DevForce and Silverlight, you can now build data-intensive Rich Internet Applications with a great user experience and seamless deployment. By freeing you from the task of writing and maintaining the extensive plumbing and infrastructure code necessary to support an n-tier model, DevForce allows you to focus on building the application you and your customers need and want.

DevForce ships with plenty of documentation, code samples, and tutorials to help you address the full spectrum of development tasks; and a world-class team of engineers will help you over the rough spots. Try it today!