Contents

This code sample shows how to break out the domain model into its own assembly. For step-by-step instructions on how to do this for your application see Separate the model project.

- **Platform:** Silverlight
- **Language:** C#, VB
- **Download:** Separate model projects

Problem

It is generally good practice to decouple your domain model from the other parts of the application. This makes the architecture easier to understand, and encourages clean code as well as code reuse.

Solution

The SeparateModelProjects solution introduces two changes from the SimpleSilverlightApp solution:

1. It uses a more complex project structure, separating the Domain Model and Silverlight Domain Model each into their own assemblies. This structure is more desirable for a real-world app than the simpler two-project structure used in SimpleSilverlightApp, where all server pieces are tossed into the web project, and everything else into the Silverlight project.
2. It also introduces DevForce's support for binding to anonymous types. This is facilitated by the following code in the Page.Fetch() method, which uses the DevForce DynamicTypeConverter to render anonymous types returned in a query into something to which Silverlight controls can bind:

```
_entityManager.ExecuteQueryAsync(
  query,
  args => {
   if (args.Error != null) {
     WriteMessage(args.Error.Message);
    } else {
    // Special logic to handle the projection query -
    // can't bind to an anonymous type in Silverlight
    if (AnonymousFns.IsAnonymousType(args.EntityQuery.ElementType)) {
      dg.ItemsSource = DynamicTypeConverter.Convert(args.Results);
     } else {
      dg.ItemsSource = args.Results;
     }
     ReportFetchCount(args.Results);
    }
  },
 null);
```

```
    ' Special logic to handle the projection query -
    ' can't bind to an anonymous type in Silverlight
    _entityManager.ExecuteQueryAsync( _
      query, Sub(args)
               If args.Error IsNot Nothing Then
                 WriteMessage(args.Error.Message)
               Else
                 If AnonymousFns.IsAnonymousType( _
                   args.EntityQuery.ElementType) Then
                   dg.ItemsSource = DynamicTypeConverter.Convert(args.Results)
                 Else
                   dg.ItemsSource = args.Results
                 End If
                 ReportFetchCount(args.Results)
               End If
             End Sub, Nothing)
```

That code, in turn, makes it possible for the DataGrid on Page to bind to the results returned from this new query, added in the Queries region of the code behind for Page:

```
Queries.Add("Get Customer info as projection",
  _entityManager.Customers
```

```
    .Select(c => new { c.CustomerID, c.CompanyName }));
```

```
Queries.Add("Get Customer info as projection", _
  _entityManager.Customers.Select( _
    Function(c) New With {Key c.CustomerID, Key c.CompanyName}))
```

Running the application

Build and run the application (be sure that the **web project is set as the startup project**). First notice that a browser window opens to something like http://localhost:9009/Default.aspx. If you look in the Default.aspx file in your web project you'll see that it contains a Silverlight control with height and width set to fill the page, and whose source is a "XAP" file loaded from the server. This XAP file is a compressed archive containing your Silverlight application and all its dependencies and resources.

You'll first notice that you need to press the "Connect" button in this application - this creates a connection to the BOS and performs some other housekeeping to initialize communications but does this asynchronously, as required in a Silverlight application when communicating with a service. Remember that the BOS here is also hosted by the web application, at an address similar to http://localhost:9009/EntityService.svc.

Next we Login. This must also be done asynchronously, since the BOS must authenticate all users. Here, because we have not implemented an IEntityLoginManager, we are logged in as a guest.

We can now run the several queries provided in the ComboBox to fetch data to the Silverlight application. Fetches must also be performed asynchronously in Silverlight if they will go to the BOS to be fulfilled.

If you make any changes to grid data you can save those modifications by pressing the "Save" button. Here an asynchronous save operation is performed to send the changes to the BOS for saving to the database.

You can also use the "Logout" button to perform an asynchronous logout from the BOS, and the "Reset" button to both logout and disconnect.