

Alabama Code Camp 8

ADO.NET Entity Framework Version 4

Presenter: Sven Aelterman, Web/Technology Specialist, Troy University

Table of Contents

Table of Contents	1
Demo 1: Getting Started with ADO.NET EF4	2
Demo 2: Model-First Design	6
Demo 3: Using ADO.NET EF v4 in a Disconnected Scenario	9
Demo 4: POCO Entities	11

Demo 1: Getting Started with ADO.NET EF4

Start code base: EF01 Begin

Step	Action Code	Notes
1	In Domain project, add new ADO.NET Entity Data Model.	ADO.NET EntityObject Generator is to customize the code generation process.
	Use existing database: Server: <i>(local)</i> ; Database: <i>ALCodeCamp8</i> Save connection string in <i>App.Config</i> (default name) Select all tables, except <i>sysdiagrams</i> . Model Namespace: <i>CompanyModel</i>	No longer need to modify generated names if pluralize/singularize option is selected
	Modify name of entity set <i>Persons</i> to <i>People</i>	
	Copy connection string from Domain <i>App.Config</i> to <i>Web.Config</i> .	
	Add code to <i>DataAccess.CustomerData.GetContactById</i> : <pre>using (ALCodeCamp8Entities db = new ALCodeCamp8Entities()) { CustomerContact cc = (from contacts in db.CustomerContacts where contacts.PersonId == contactId select contacts).SingleOrDefault(); if (cc != null && !cc.PersonReference.IsLoaded) cc.PersonReference.Load(); if (cc != null && !cc.CustomerReference.IsLoaded) cc.CustomerReference.Load(); return cc; }</pre>	Use code snippet. Using *Reference properties is only one way to eager load. To avoid additional database roundtrip, use Include extension method (Demo 3)
	Add reference to <i>System.Data.Entity</i> to <i>DataAccess</i> project.	

Step	Action Code	Notes
	Set UI as startup project. Run application to test.	
	Examine ASPX code in default.aspx.cs Note that access to <i>FirstName</i> and <i>LastName</i> properties goes through the <i>Person</i> entity reference, but <i>Position</i> property does not.	
	Set base type of <i>Employee</i> and <i>CustomerContact</i> entities to <i>Person</i> .	
	Set <i>Person</i> class to <i>Abstract</i> = true.	
	Remove <i>PersonId</i> property from <i>Employee</i> and <i>CustomerContact</i> entities.	
	Remove associations between <i>Person</i> and <i>Employee</i> and <i>Person</i> and <i>CustomerContact</i> .	
	Set mapping for <i>PersonId</i> field on <i>People</i> and <i>CustomerContacts</i> tables to <i>PersonId</i> property.	
	Modify data access code to select <i>People</i> of type <i>CustomerContact</i> only: <pre>CustomerContact cc = (from contacts in db.People.OfType<CustomerContact>() where contacts.PersonId == contactId select contacts).SingleOrDefault(); //if (cc != null && !cc.PersonReference.IsLoaded) cc.PersonReference.Load();</pre>	<i>People</i> , <i>CustomerContact</i> and <i>Employee</i> are now part of the same <i>EntitySet</i> .
	Remove <i>Person</i> navigation property from <i>FirstName</i> and <i>LastName</i> property access in default.aspx.cs: NameLabel.Text = cc.FirstName + ' ' + cc.LastName;	
	Run and demonstrate application.	
	Add update button: <pre><p> <asp:Button runat="server" ID="UpdateButton" Text="Update Contact" OnClick="UpdateButton_Click" /></p></pre>	Use code snippets.

Step	Action Code	Notes
	<p>Add update button event handler:</p> <pre>protected void UpdateButton_Click(object sender, EventArgs e) { // Somehow, the ID has to be cached, // or we have to try to do a lookup "server-side" with values CustomerContact cc = Session["CustomerContact"] as CustomerContact; if (cc != null) { // Note: this is about the poorest way to update... and for demo purposes only // (should split first and last into 2 boxes if updates are required) cc.FirstName = NameLabel.Text.Substring(0, NameLabel.Text.IndexOf(' ')); cc.LastName = NameLabel.Text.Substring(cc.FirstName.Length + 1); cc.Position = PositionLabel.Text; cc.CustomerId = int.Parse(CompanyDropDownList.SelectedValue); // Imagine this is where we are calling the WCF service CustomerData data = new CustomerData(); data.UpdateCustomerContact(cc); } }</pre>	
	<p>Review <i>CustomerData.UpdateCustomerContact</i> method. Uncomment <i>ApplyCurrentValues</i> statement.</p>	
	Run and demo update functionality.	Use Profiler to show only modified properties are sent.
	Execute <i>SalesByCustomer</i> stored procedure to show output.	
	Add <i>SalesList.aspx</i> web form to UI.	
	Add <i>GridView</i> control.	

Step	Action Code	Notes
	Import <i>DataAccess</i> namespace: <code>using DataAccess;</code>	
	Bind <i>GridView</i> to output from <i>CustomerData</i> : <code>CustomerData data = new CustomerData();</code> <code>SalesGridView.DataSource = data.SalesByCustomer();</code> <code>SalesGridView.DataBind();</code>	Use code snippet
	Generate <i>SalesByCustomer</i> method.	
	In Entity Data Model, Update Model from Database to add <i>SalesByCustomer</i> stored procedure.	
	Add function import for <i>SalesByCustomer</i> stored procedure and create new complex type. Complex type name: <i>CustomerWithSales</i>	
	In method <i>CustomerData.SalesByCustomer</i> : <code>return new ALCodeCamp8Entities().SalesByCustomer().ToList();</code>	Use code snippet.
	Change return value of <i>CustomerData.SalesByCustomer</i> from object to <i>List<CustomerWithSales></i> .	
	Run UI with <i>SalesList.aspx</i> as the start page.	

Demo 2: Model-First Design

Start code base: EF02 Begin

Step	Action Code	Notes
	Add new model to <i>Domain</i> project. Choose Empty Model.	
	Drag an Entity item from the toolbox to the designer surface.	
	Rename existing property <i>Id</i> to <i>PersonId</i> . Ensure Entity Key property = true, Type = Int32, Store Generated Pattern = Identity.	
	Change Entity Set Name to <i>People</i> and Name to <i>Person</i> .	
	Add scalar properties: <ul style="list-style-type: none">• <i>FirstName</i>• <i>LastName</i>• <i>MiddleInitial</i> (nullable)• <i>Email</i> (nullable)	
	Set maximum field lengths on string type properties (50, 50, 1, 70).	
	Set Abstract = true for <i>Person</i> entity.	

Step	Action Code	Notes
	<p>Add <i>CustomerContact</i> and <i>Employee</i> entities to diagram.</p> <ul style="list-style-type: none"> • Set Base Type to <i>Person</i>. • Delete <i>Id</i> properties. • Set entity names. • Employee scalar properties: <i>SSN</i>, 9, not null; <i>DateOfBirth</i>, date, not null • <i>CustomerContact</i> scalar properties: <i>Position</i>, 50, not null; <i>CustomerId</i>, int, not null 	<p>Notice we add foreign keys also.</p>
	<p>Add <i>Customer</i> entity to diagram.</p> <ul style="list-style-type: none"> • Set entity name and entity set name. • Change <i>Id</i> property to <i>CustomerId</i>. • Set Store Generated Pattern to Identity. • Scalar property: <i>CompanyName</i>, 50, not null 	
	<p>Add Association to diagram between <i>CustomerContact</i> and <i>Customer</i> entity.</p> <p>Click first on <i>Customer</i>, then on <i>CustomerContact</i> (direction of one-to-many association).</p>	
	<p>Set Referential Constraint:</p> <ul style="list-style-type: none"> • Principal: <i>Customer</i> • Dependent: <i>CustomerContact</i> • Keys: <i>CustomerId</i> 	
	<p>Examine model properties.</p>	
	<p>Right-click model designer surface, Generate Database from Model.</p> <ul style="list-style-type: none"> • Create new connection, point to <i>ALCodeCamp8Demo2</i> database. 	
	<p>Save DDL as <i>CompanyModel.edmx</i>. Accept warning (there is no MSL or SSDL).</p>	

Step	Action Code	Notes
	Execute the script on the <i>ALCodeCamp8Demo2</i> database.	
	Execute the INSERT script on the <i>ALCodeCamp8Demo2</i> database.	
	Add connection string to <i>Web.Config</i> .	
	Run and demo application.	

Demo 3: Using ADO.NET EF v4 in a Disconnected Scenario

Start code base: EF01 Final

Step	Action Code	Notes
1	Open EF01 Final solution.	
	Discuss code generation from EDMX file.	
	Set Code Generation Strategy to None.	
	Add <i>ADO.NET EntityObject Generator</i> .	Discuss T4, time allowing
	Remove T4 template.	Want to go back to default: Set Code Generation Strategy to default on EDMX file.
	Add <i>ADO.NET Self-Tracking Entities Generator</i> .	
	Open one generated type class and examine setter property.	
	Modify <i>DataAccess.CustomerData.GetContactById</i> : <pre> db.ContextOptions.LazyLoadingEnabled = false; CustomerContact cc = (from contacts in db.People.OfType<CustomerContact>().Include("Customer") where contacts.PersonId == contactId select contacts).SingleOrDefault(); //if (cc != null && !cc.PersonReference.IsLoaded) cc.PersonReference.Load(); //if (cc != null && !cc.CustomerReference.IsLoaded) cc.CustomerReference.Load(); </pre>	STEs do not support lazy loading, in addition, there are no *Reference properties.

Step	Action Code	Notes
	Start tracking changes to the entity before returning from <i>GetContactById</i> : <code>if (cc != null) cc.StartTracking();</code>	
	Stop tracking changes to the entity when entering <i>UpdateCustomerContact</i> : <code>cc.StopTracking();</code>	
	Modify <i>UpdateCustomerContact</i> to remove call to database and replace with <i>ApplyChanges</i> : <code>context.People.ApplyChanges(cc);</code>	
	Run application and demonstrate functionally equivalent.	

Demo 4: POCO Entities

Start code base: EF04 Begin

Step	Action Code	Notes
1	Show references in Domain: no <i>System.Data.Entity</i> reference	
	Create Entity Data Model in <i>DataAccess</i> project. Same as Demo 1, but exclude <i>Sales</i> and <i>Employees</i> table.	
	Set code generation on EDMX file to None.	
	Modify the model like Demo 1: <ul style="list-style-type: none"> • <i>CustomerContact</i> inherits from <i>Person</i> • Set <i>Person</i> abstract • Remove association between <i>Person</i> and <i>CustomerContact</i> • Remove <i>PersonId</i> property • Map <i>PersonId</i> field to <i>PersonId</i> property 	
	Add new class <i>Customer</i> to <i>Domain</i> project.	
	Add scalar and navigation properties to <i>Customer</i> class. Note that the property names need to match the model. Property names do not have to match field names. <pre>public class Customer { public int CustomerId { get; set; } public string CompanyName { get; set; } public List<CustomerContact> CustomerContacts { get; set; } }</pre>	Use code snippet.

Step	Action Code	Notes
	Add existing classes <i>CustomerContact</i> and <i>Person</i> to <i>Domain</i> project.	Note that <i>Person</i> class is abstract.
	In <i>DataAccess</i> project, create new class <i>CompanyContext</i> <pre data-bbox="283 467 1159 553">public class CompanyContext : System.Data.Objects.ObjectContext { }</pre>	
	For every entity, create an <i>ObjectSet</i> read-only property: <pre data-bbox="283 626 1188 683">public ObjectSet<Domain.Customer> Customers { get; private set; } public ObjectSet<Domain.Person> People { get; private set; }</pre>	Use code snippet.
	Create a public constructor that initializes each <i>ObjectSet</i> : <pre data-bbox="283 751 1035 927">public CompanyContext() : base("name=ALCodeCamp8Entities", "CompanyModel") { Customers = CreateObjectSet<Domain.Customer>(); People = CreateObjectSet<Domain.Person>(); }</pre>	Use code snippet. Must call the base constructor with the name of the Entity SQL connection string and the name of the model. Note that <i>CustomerContacts</i> is initialized as an <i>ObjectSet</i> of <i>Person</i> .
	Copy connection string from <i>App.Config</i> to <i>Web.Config</i> .	
	Build solution and run application. Demonstrate that retrieving info works, but saving does not because <ul data-bbox="331 1300 751 1365" style="list-style-type: none"> • Change tracking is not enabled • The relationship is not fixed up. 	

Step	Action Code	Notes
	After adding the <i>CustomerContact</i> object to the context, force its state to modified: <code>context.ObjectStateManager.ChangeObjectState(cc, System.Data.EntityState.Modified);</code>	Alternative 1
	In default.aspx.cs, Change code that sets new <i>CustomerId</i> : <pre>// Determine if the customer ID was changed int NewCustomerId = int.Parse(CompanyDropDownList.SelectedValue); if (cc.CustomerId != NewCustomerId) { cc.CustomerId = NewCustomerId; // Have to set Customer entity to null to avoid issues with referential integrity cc.Customer = null; }</pre>	Use code snippet
	Open SQL Server Profiler, create new trace: Exclude Application Name "Report Server" and "%Management Studio%"	
	Run and demo updates.	
	Get original values of <i>CustomerContact</i> and <i>ApplyCurrentValues</i> : <code>CustomerContact OriginalValues = context.People.OfType<CustomerContact>().SingleOrDefault(p => (p.PersonId == cc.PersonId));</code> <code>context.People.ApplyCurrentValues(cc);</code>	Alternative 2 Use code snippet.
	Run and demo updates. Indicate difference between the UPDATE statements in the Profiler.	