# Module 2 Lab D

# Using LINQ

Time 70 Minutes

##### Objective: Use the CrmServices client to retrieve, update and create records using the OrganizationServiceContext class and LINQ from and MVC Web Application

**Step 1: Creating the starter MVC project**

|  |  |
| --- | --- |
| **1** | Open Visual Studio 2015 Community Edition and create a new C# project selecting the ASP NET Web Application Template under the Web Category. For the **Location** Select the Lab D Starter folder and for it’s Name enter DynamicsLinq. **Deselect** the create directory for solution option |

|  |  |
| --- | --- |
| **2** | After clicking on the OK button change the Authentication Type to No Authentication and make sure the Host in the cloud checkbox is cleared as shown below. |
| **3** | After the Solution has been created use the Solution Explorer to add references to Microsoft.Xrm,Sdk,dll and Microsoft.Xrm.ToolingConnector.dll.  Browse to the Dynamics SDK Bin folder and add references to Microsoft.Xrm,Sdk,dll and Microsoft.Xrm.ToolingConnector.dll.    Also add reference to the System.Runtime.Serialization dll using the Assemblies tab. |

**Step 2: Using CrmSvcUtil.exe to create the Context and Entity classes and Building an MVC Application that used them.**

|  |  |
| --- | --- |
| **1** | Add the location of the CrmSvcUtil.exe to the system path environment variable. By opening up the settings app and searching for system    Click on the system button and then click on Advanced System Settings    Click on the environment Variables button    Under the System Variables list select to edit the **Path** environment variable.    If you installed the Dynamics 365 SDK in the location specified in a previous lab then you should add the following entry  C:\Program Files\Microsoft SDKs\SDK\Bin    to the list of paths. If you extracted the files elsewhere then use the path correct for your system.      Click OK on all the screens that follow to complete the edit.  Now that we have modified the path environment variable we will use CrmSvcUtil to generate Entity classes and a Context class. |
| **2** | Open up a command prompt and use the CD command to navigate to the project folder for this lab e.g. CD C:\QACRMDEV\LABS\Module2\Lab D\Starter\DynamicsLinq\Models |
| **3** | Execute the following command replacing the URL and credentials that are correct for your tenancy.  CrmSvcUtil.exe /url:https://qajuly17.api.crm11.dynamics.com/XRMServices/2011/Organization.svc /out:Entities.cs /username:paul@qajuly17.onmicrosoft.com /password:QaJuly17 /namespace:DynamicsLinq /serviceContextName:OrgServiceContext  you can get the correct URL for your tenancy by navigating to the settings area of Dynamics 365 selecting Customizations > Developer Resources |
| **5** | The output should look like this (it may take some time for the Entities.cs file to be generated) |
| **6** | Go back to the Visual Studio project and right click on the model folder add select to add an existing item navigate to the models folder and select the Entities.cs file generated by the CrmSvcUtil.exe |
| **7** | Build the solution and fix any errors |
| **8** | Edit the web.config file in the root folder of the solution **(NOT the one in the Views folder)** Below the AppSettings Areaadd the Connection used to connect to your Dynamics 365 **T**enancy. You can copy and paste the one from the previous lab e.g.  <add key="webpages:Enabled" value="false" />  <add key="ClientValidationEnabled" value="true" />  <add key="UnobtrusiveJavaScriptEnabled" value="true" />  </appSettings>  <connectionStrings>  <add name="CrmOnline" connectionString="Url=https://qajuly17.crm11.dynamics.com;Username=paul@qajuly17.onmicrosoft.com;Password=CabrelliJuly17;authtype=Office365" />  </connectionStrings>  <system.web> |
| **9** | Modify the Application\_Start() method in the Global.asax.cs file to set the security protocol to TLS 1.2 as per the previous labs. |
| **10** | Run the web application in debug mode (F5) and correct any errors that might be reported.  Stop the Web Application and then right click on the Controllers folder and Select Add.  Create a Controller Named AccountController, choose the option that will generate a controller that supports read/write operations |
| **9** | Within the AccountController class declare a variable named cnString Add a call to retrieve the connection string from the web.config file assigning the value to cnString and fix the missing using statement in the usual way.    string cnString = ConfigurationManager.ConnectionStrings["CrmOnline"].ConnectionString; |
| **10** | Within the index method of the Account controller class create an instance of the CrmServiceClient initialised with the connection string that was read from the Web.Config file.  // GET: Account  public ActionResult Index()  {  using (var crmSvc = new CrmServiceClient(cnString))  {  }  Fix the missing using statement in the usual way. |
| **11** | Within the using statement declare and initialise a variable named ctx of type OrgServiceContext and pass the crmSvc object to the class constructor as follows |
| **12** | Use the ctx object to return 10 Accounts records ordered by Account name and pass the data return to a view. |
| **13** | Right click on the Index Method and from the menu select add view |
| **14** | Using the diagram below to identify what values you should be setting for the View  Make sure you **deselect** “Reference script libraries”. |
| **15** | Open up the Index.cshtml file in the Views Account folder and you will see a very large number of elements that will render every attribute of an account entity. Replace the Index.cshtml file with the one in the assets folder in Module 2\LAB D\Starter. |
| **16** | Open the RouteConfig.cs file in the App\_Start folder and modify the defaults so that the index method of the Account Controller is the default path as shown below. |
|  | Hit the F5 key to run the app and correct any compile/runtime errors. You should see something similar to that shown below    Note that if you click on any of the Create, Edit, Details or Delete links an exception is thrown. We will fix these now. |

**Step 3: Adding Create, Update and Delete Functionality**

|  |  |
| --- | --- |
| **1** | Add the views Create.cshtml, Edit.cshtml, Details.cshtml and Delete.cshtml from the Assets folder to the Views\Account folder in the project. |
| **2** | Replace the code in the Create Methods with the code listed below. Then build the solution and correct any compile time errors.  // GET: Account/Create  public ActionResult Create()  {  Account acc = new Account();  return View(acc);  }  // POST: Account/Create  [HttpPost]  public ActionResult Create(Account newAccount)  {  using (var crmSvc = new CrmServiceClient(cnString))  {  OrgServiceContext ctx = new OrgServiceContext(crmSvc);  //add the new Account object to the context object  ctx.AddObject(newAccount);  // call the webservice to savechanges back to Dynamics 365  ctx.SaveChanges();  return RedirectToAction("Index");  }  } |
| **3** | Replace the code in the Edit methods with that shown below. Then build the solution and correct any compile time errors.  // GET: Account/Edit/5  public ActionResult Edit(Guid id)  {  using (var crmSvc = new CrmServiceClient(cnString))  {  OrgServiceContext ctx = new OrgServiceContext(crmSvc);  // retrive the Account the user is interested in editing  var account = ctx.AccountSet.Where(acc => acc.Id == id).Select(acc => acc);  // render the form that allows the user to edit the Account  return View(account.FirstOrDefault());  }  }  // POST: Account/Edit/5  [HttpPost]  public ActionResult Edit(Guid id, Account modifiedAccount)  {  // modifiedAccount has been posted back with all the changes made by the user  using (var crmSvc = new CrmServiceClient(cnString))  {  OrgServiceContext ctx = new OrgServiceContext (crmSvc);  // Get the context object to start tracking this entity for changes  ctx.Attach(modifiedAccount);  //mark it as being a modified version that should be saved  ctx.UpdateObject(modifiedAccount);  // save changes to Dynamics 365  ctx.SaveChanges();  return RedirectToAction("Index");  }  } |
| **4** | Replace the code in the Delete methods with that shown below. Then build the solution and correct any compile time errors.  // GET: Account/Delete/5  public ActionResult Delete(Guid id)  {  using (var crmSvc = new CrmServiceClient(cnString))  {  // Retrive the account record to be delete  OrgServiceContext ctx = new OrgServiceContext(crmSvc);  var account = ctx.AccountSet.Where(acc => acc.Id ==  id).Select(acc => acc);  // Reender the info and ask user to confirm delete  return View(account.FirstOrDefault());  }  }  // POST: Account/Delete/5  [HttpPost]  public ActionResult Delete(Account accToDelete)  {  // perform the delete  using (var crmSvc = new CrmServiceClient(cnString))  {  OrgServiceContext ctx = new OrgServiceContext(crmSvc);  //enable traking of entity  ctx.Attach(accToDelete);  // Mark the entity as one that should be delete  ctx.DeleteObject(accToDelete);  // send the request to the Organisation WebService  ctx.SaveChanges();  return RedirectToAction("Index");  }  } |
| **5** | Finally replace the Details Method with the following code    // GET: Account/Details/5  public ActionResult Details(Guid id)  {  using (var crmSvc = new CrmServiceClient(cnString))  {  OrgServiceContext ctx = new OrgServiceContext(crmSvc);  // get the Account record by it's Id  var account = ctx.AccountSet.Where(acc => acc.Id == id).Select(acc => acc);  //Render the Account Record Information  return View(account.FirstOrDefault());  }  } |
| **6** | Run the app in debug mode, Create and account called Aardvark, then edit the account in some way, click on the details to see the details of the account then delete it. You may need to increase the number of accounts taken by increasing the parameter to the Take extension method to 100 |

**Step 4: Adding Paging functionality**

|  |  |
| --- | --- |
| **1** | Add the Paging.cshml file in the assets folder to the Views/Account folder in the project .  Open the Paging.cshtml file in visual studio and you will see that there are links at the top that will be associated with paging through sets of Account records |
| **2** | Add the following method to the Account controller.  public ActionResult Paging(int id = 0)  {  int pageNumber;  // Make sure we're not attempting to navigate before the first page  if (id >= 0)  {  pageNumber = id;  }  else  {  pageNumber = 0;  }  // set the number of records to show per page  int pageSize = 5;  // based on the pagesize and page number identity how many  // records we need to skip  int recordsToSkip = pageNumber \* pageSize;  using (var crmSvc = new CrmServiceClient(cnString))  {  OrgServiceContext ctx = new OrgServiceContext(crmSvc);  // execute a request to retrieve the records required  var accounts = ctx.AccountSet.OrderBy(acc =>  acc.Name).Skip(recordsToSkip).Take(pageSize);    // assign to the ViewBag properties values that will be used  // to render  //the corrent links for the next and previous buttons  ViewBag.nextPageNumber = pageNumber + 1;  ViewBag.previousPageNumber = pageNumber - 1;  return View(accounts);  }  } |
| **3** | Notice that the method uses the Skip and Take Linq extension methods to retrieve pages of records and that the variables recordsToSkip and pageSize are used to provide the parameter values to these methods. |
| **4** | Modify the \_LayOuts.cshtml file in the Views/Shared folder |
| **5** | Add the following Actionlink entry in the nav bar area  <li>@Html.ActionLink("Accounts", "Paging", "Account")</li> |
| **6** | Run the app and then click on the Accounts button |
| **7** | You should be able to page through all the accounts |