**FBOLinx.Web**

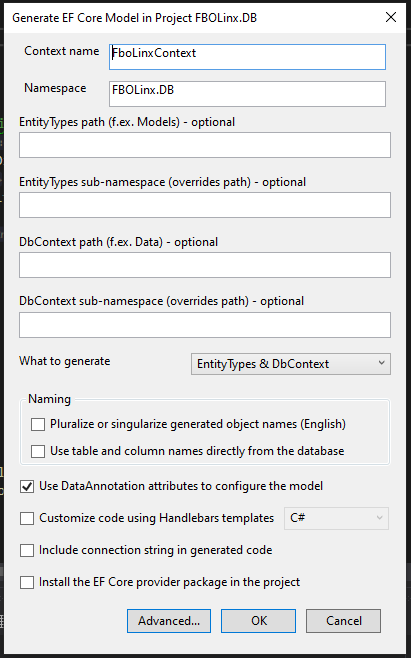
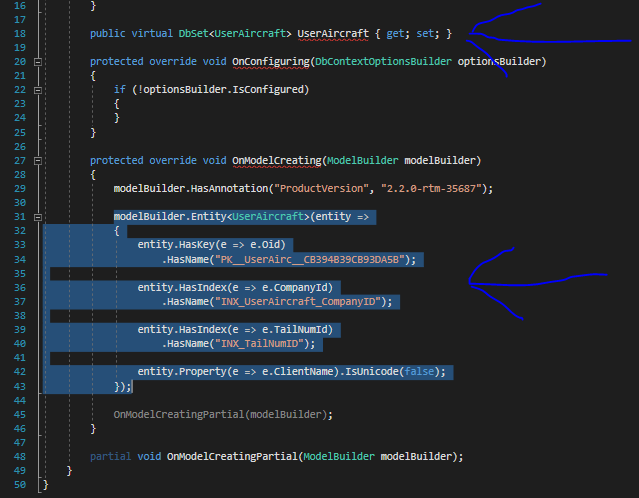
***Development Walkthroughs – Entities and DTO***

# Development Walkthroughs

Adding a new table to the database

1. Add your new table to the paragon\_test\_Local (development environment) database as well as the paragon\_test (test environment) database.
   1. Ensure your table has a primary key with the name OID. This can be of type “int” or “bigint” depending on how much data you anticipate going through this table.
2. Create a SQL script to create or alter the table.
   1. Ensure the script has a *use [Paragon\_Test]* line at the top so the script changes to the right DB on execution.
3. Save the script above in the FBOLinx.Web/SQL/<BranchName> folder of the repository where <BranchName> is your current branch name (usually the ID of your ClickUp task).

Adding a DB entity to FBOLinx.DB project

1. Add the entity to the FBOLinx.DB project
   1. Right-click on the FBOLinx.DB project and select EF Core Power Tools >> Reverse Engineer.
      1. Select your database connection. If you don’t have one setup, just add a new one to be used in the future.
   2. Choose the table you wish to import into the Entity Framework
      1. Ensure the table has a primary key. If it does not, you will see a red warning symbol and thus a primary key must be added to the table before you can use it on this step.
   3. On the next step, ensure your settings match the screenshot below:
      1. 
   4. Two classes will be added to the root of the project:
      1. FboLinxContext.cs
      2. <YourTableName>.cs
   5. Take the highlighted sections of the newly created FboLinxContext.cs file and move them to the FboLinxContext.cs file in the “Context” folder.
      1. 
   6. Delete the FboLinxContext.cs file that was created (the one in the root of Fuelerlinx.DB).
   7. Move the <YourTableName>.cs file to the “Models” folder.
   8. Have your new table’s class inherit from FBOLinxBaseEntityModel<int> (or a something other than int if you used a different type for your primary key).
   9. Remove the Oid property from your class (this is handled in the BaseEntityModel

Add an entity specification to Fuelerlinx.DB

1. Right-click on the “Specifications” folder in FBOLinx.DB and add a new folder called <YourTableName>
   1. Within this newly created folder, add a new class called <YourTableName>Specification.cs
2. Have your newly created Specification class inherit from BaseSpecification<DB.Models.<YourNewEntityClass>>, ISpecification<DB.Models.< YourNewEntityClass>>
3. Add a new constructor for every common query that would be run against your table that is not a direct select by OID/primary key.
   1. An example would be searching for CustomerAircraft records by customerID:

public CustomerAircraftByCustomerIdSpecification(int customerId, List<string> tailNumberList) : base(x => x.CustomerId == customerId && tailNumberList.Contains(x.TailNumber))

{

}

Creating the entity DTO in FBOLinx.ServiceLayer

**Note:** The DTO (Data Transfer Object) is used as a medium between the database and the API. This will be the actual class seen by the clients using your API. It is recommended to change property names/types within these classes if you feel that the table has limitations (i.e. poorly named columns for what the data represents).

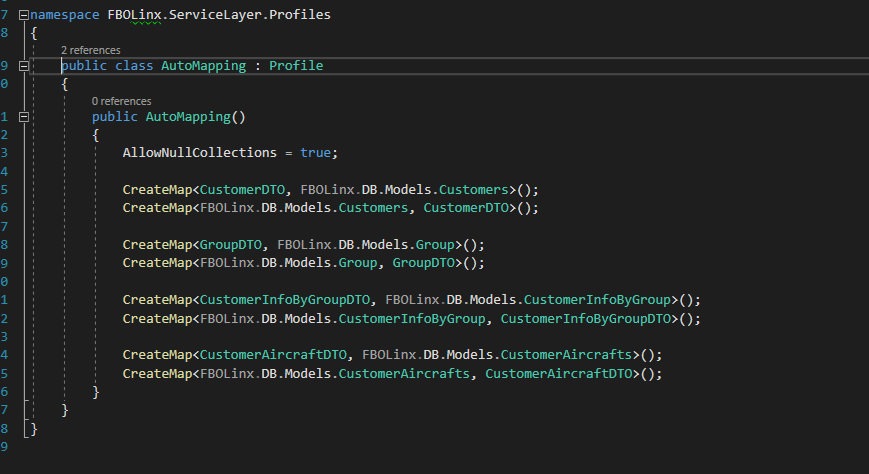
1. Right-click on the FBOLinx.ServiceLayer/DTO folder and add a new class called <YourNewEntityClass>DTO.cs
   1. *Example: CustomerAircraftDTO.cs*
2. Add the properties from <YourNewEntityClass> into the DTO. Make any naming changes you see fit.
3. Inherit from BaseEntityModelDTO and IEntityModelDTO. Make the generic type DB.Models.<YourNewEntityClass>.
4. Override the two inherited functions if you need to do additional work when mapping the properties between the entity and the DTO:
   1. ConvertToEntity function:
      1. Populate the passed in “result” entity with the values of your current DTO. This is handled by the AutoMapper by default in the base class but can be overridden if needed.
   2. CastFromEntity function:
      1. Populate the current instance of the DTO with the passed in entity. This is also handled by the AutoMapper by default in the base class but can be overridden if needed.

Add the New Entity Service to the FBOLinx.ServiceLayer Project

1. Right-click on the FBOLinx.ServiceLayer/EntityServices folder and add a new class called <YourNewEntityName>EntityService.cs
2. Have your new class inherit from BaseEntityService<T, TDTO> and IEntityService<T, TDTO>, where T is DB.Models.<YourNewEntityName> and TDTO is your new DTO from the previous section.
3. Add a constructor that takes in IMapper and FboLinxContext.
4. Add in any other entity service methods you may need. A simple search on the table should be done with new Specifications from the second section above.
5. Register your service in FBOLinx.Web/ServiceConfiguration.cs.

Add the New Entity Service to the FBOLinx.ServiceLayer Project

1. Add your mapping for the DTO >> Entity and Entity >> DTO to the AutoMapping class.
   1. This can be found in FBOLinx.Web/ServiceLayer/Profiles/AutoMapping.cs.
2. Use the CreateMap<> method to add your mapping. If you require special mapping cases (i.e. the DTO and Entity properties do not have the same name) then please refer to the AutoMapper document.
   1. <https://docs.automapper.org/en/stable/>

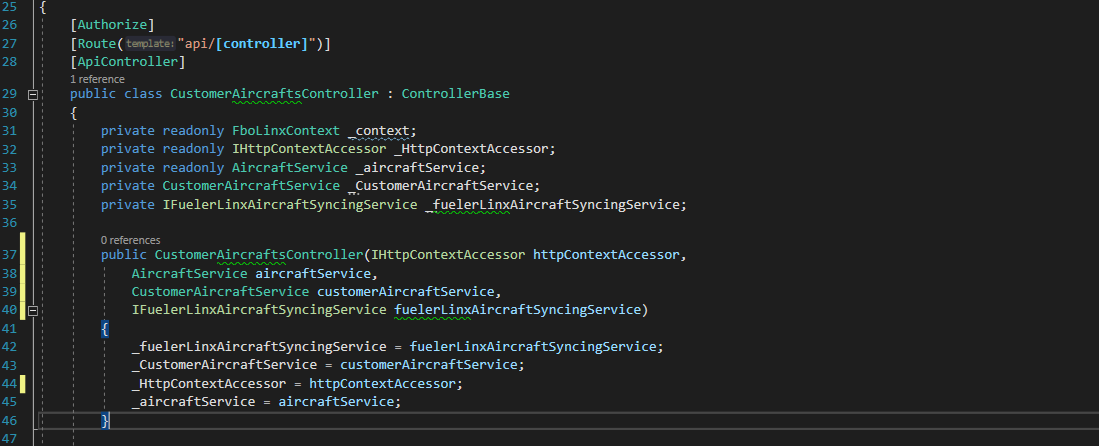


Add Controller to FBOLinx.Web/Controllers

**Controller**

FBOLinx.WebI>Controllers

1. Add a controller that matches your new entity followed by Controller.
   1. <YourNewEntityName>Controller.cs
2. Add the entity service to the controller along with any other dependencies. IHttpContextAccessor is generally needed as well for header access.

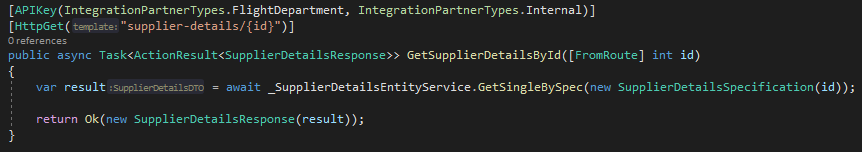


1. If the entity will be altered via the client UI or through an integration then add CRUD operations for your entity.

**The following are screenshots/samples from the FuelerLinx.API project as FBOLinx.Web’s controllers are still improperly using the DBContext directly.**

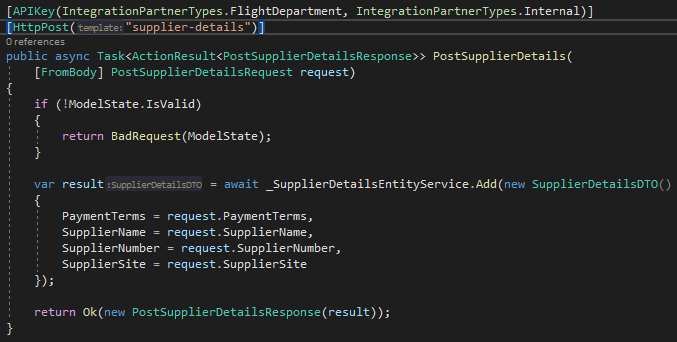
**Get**

For the GET response use the conditions created in the project specification

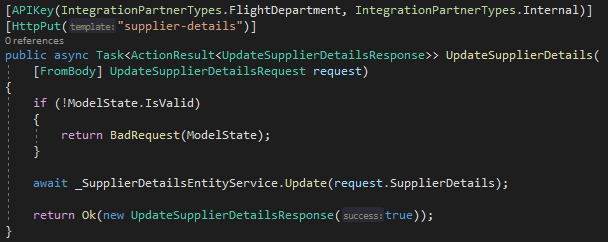


**Post**

Posting the requested items from the project DTO

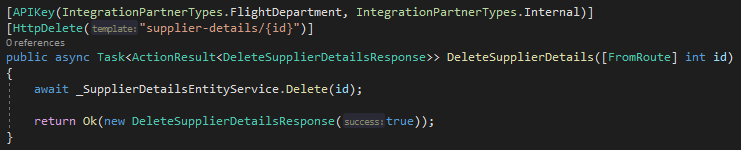


**Update**



**Delete**

Deletes record based on ID



Merging Project into Develop and Deploying via TeamCity

1. Through GitHub, merge the project into the develop branch.
   1. Note that the “develop” branch is locked every second-Thursday for the production release. It is usually re-opened the following Tuesday depending on hotfixes and patches.
2. Log into TeamCity <http://13.86.127.50:8111/> and confirm that your latest merge is showing for FBOLinx.Web and that a build has been started/queue’d.