

Entity Framework – Code First Approach – Part 4

This will show walkthrough of following concepts

- 1- DataAnnotations
- 2- Validations using DataAnnotations

This tutorial is prepared with Visual Studio 2013 + SQL Server.

Pre-requisite: Entity Framework – Code First Approach – Part 3

Version	Last updated	Comments	Modified By
V1.0	29-04-2016		Bilal Shahzad

Introduction

- 1- The **System.ComponentModel.DataAnnotations** namespace provides attribute classes that are used to define metadata for our model classes. We can add these attributes on our properties/classes to add some features in them. We can use these attributes (without using Entity Framework). Although Entity Framework has extended this namespace and provided “MaxLength” & “MinLength” attributes.
- 2- The **System.ComponentModel.DataAnnotations.Schema** namespace includes attributes that impacts the schema of the database. This comes with “Entity Framework” DLL.
- 3- Following screenshots are showing some attributes of both namespaces.

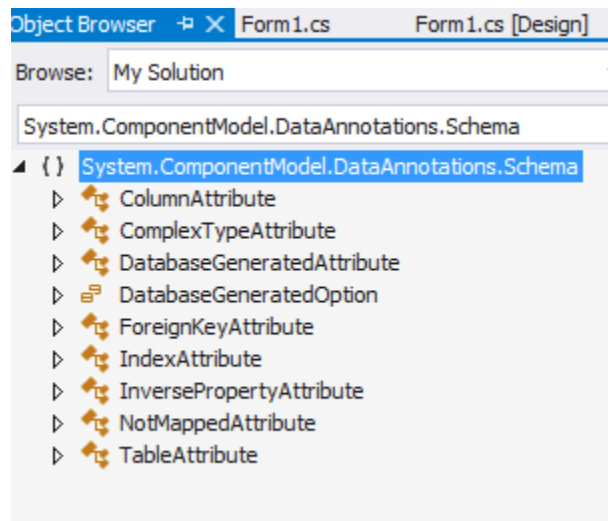
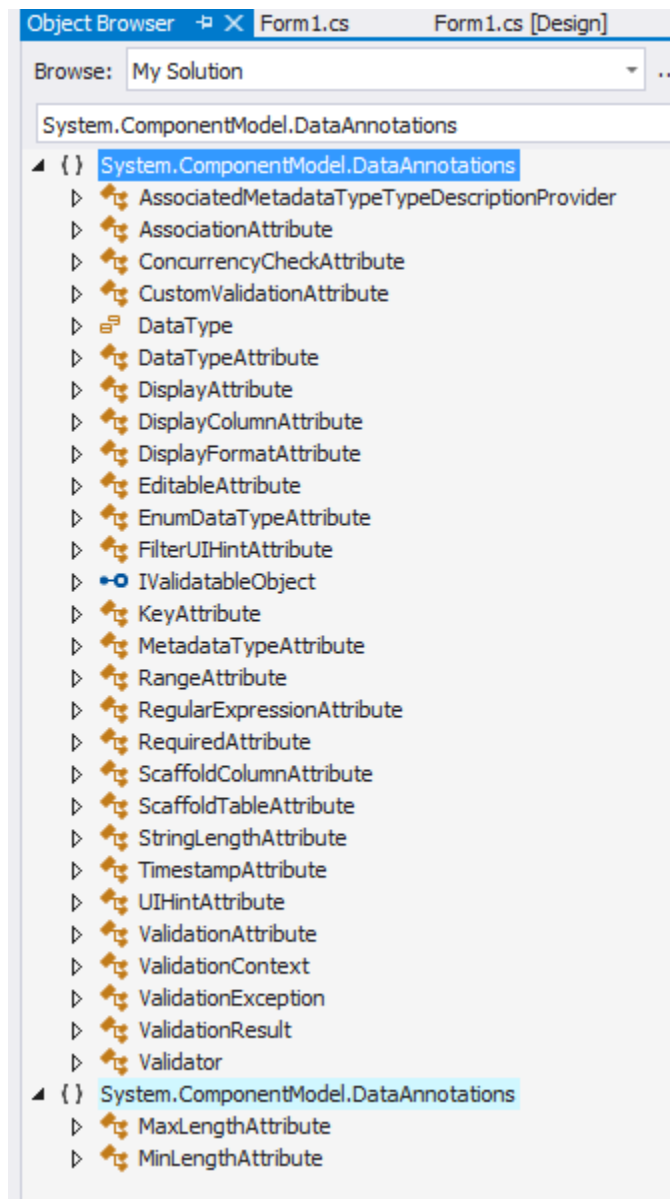
System.ComponentModel.DataAnnotations Attributes:

Attribute	Description
Key	Mark property as EntityKey which will be mapped to PK of the related table.
Timestamp	Mark the property as a non-nullable timestamp column in the database.
ConcurrencyCheck	ConcurrencyCheck annotation allows you to flag one or more properties to be used for concurrency checking in the database when a user edits or deletes an entity.
Required	The Required annotation will force EF (and MVC) to ensure that property has data in it.
MinLength	MinLength annotation validates property whether it has minimum length of array or string.
MaxLength	MaxLength annotation is the maximum length of property which in turn sets the maximum length of a column in the database
StringLength	Specifies the minimum and maximum length of characters that are allowed in a data field.

System.ComponentModel.DataAnnotations.Schema Attributes:

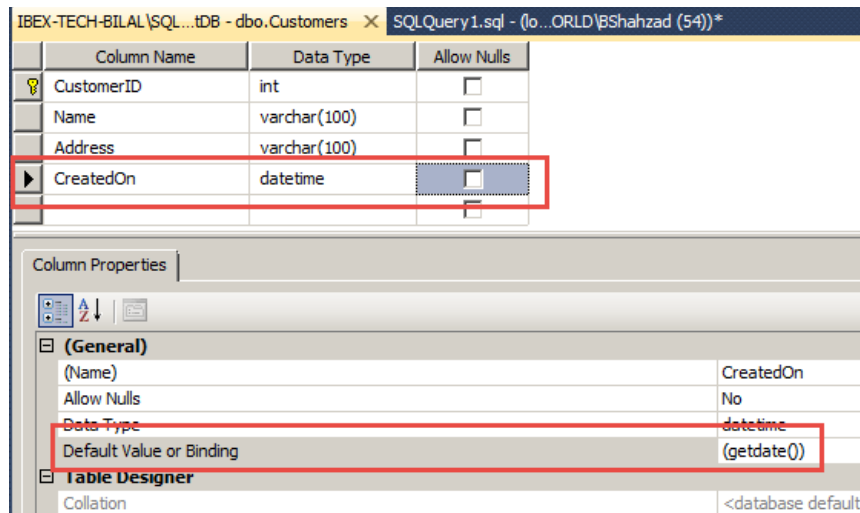
Attribute	Description
Table	Specify name of the DB table which will be mapped with the class
Column	Specify column name and datatype which will be mapped with the property
Index	Create an Index for specified column. (EF 6.1 onwards only)
ForeignKey	Specify Foreign key property for Navigation property
NotMapped	Specify that property will not be mapped with database
DatabaseGenerated	DatabaseGenerated attribute specifies that property will be mapped to computed column of the database table. So, the property will be read-only property. It can also be used to map the property to identity column (auto incremental column).
InverseProperty	InverseProperty is useful when you have multiple relationships between two classes.
ComplexType	Mark the class as complex type in EF.

- 4- In Visual Studio (Object Browser), we can browse these namespaces to see their members.

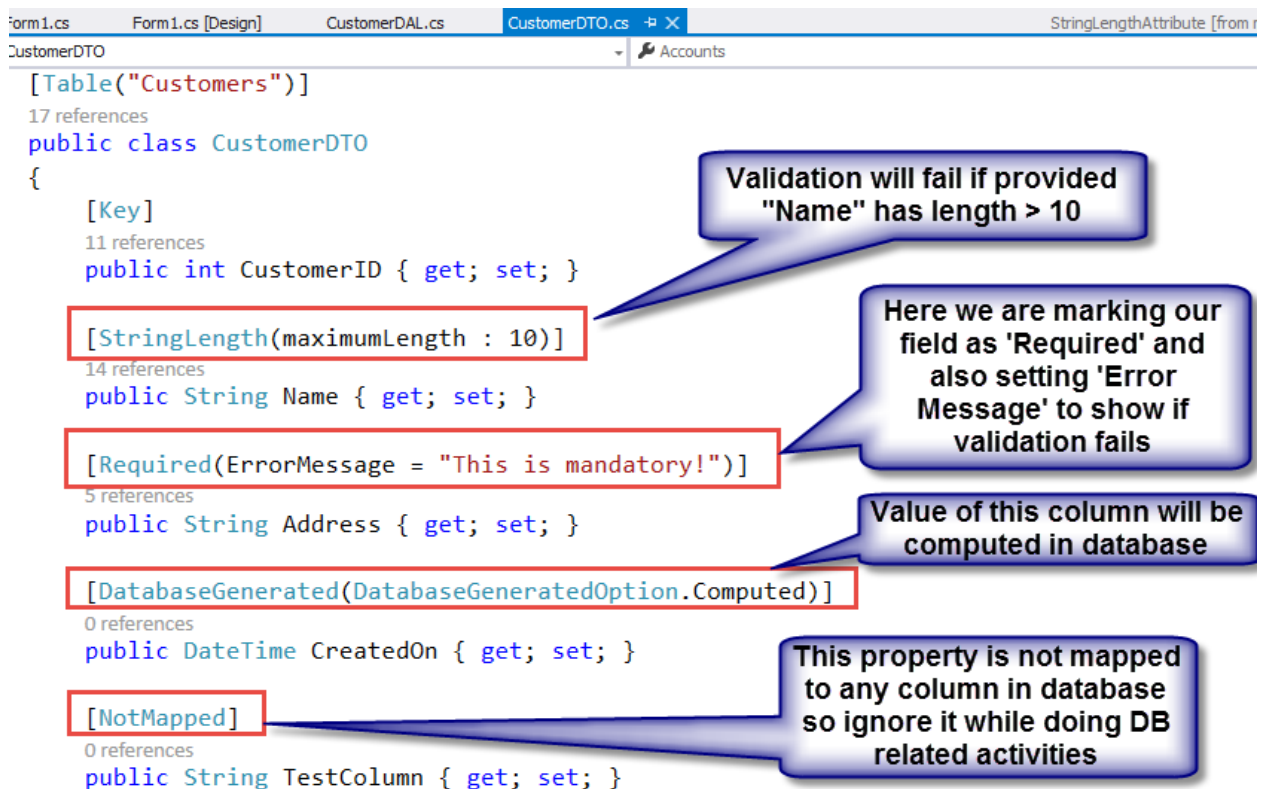


Step by Step Walkthrough

- 1- To validate our data, we can use custom validation (by applying IF conditions for example) or we can annotate our properties in our model to perform automatic validation. Here we are going to see how to do this and how to check if any validation is failed.
- 2- Let's add a column "CreatedOn" in our "dbo.Cstuomers" table. This column will automatically have current date & time whenever a record is inserted. We've set "getdate()" function in default value as shown in the screenshot.



- 3- Now let's make some changes in our "CustomerDTO" class by adding some annotations.



- 4- If “ErrorMessage” property is not set, a default error message (based on attribute type) is generated automatically.
- 5- Now when we’ll create an object of this class, we can validate using “Validator” class whether created instance is according to the decorated attributes or not. Let’s go to “Save” button click event handler and add some code as highlighted below. This logic of annotation & validation is independent from EF and can be used without EF.

```

private void btnSave_Click(object sender, EventArgs e)
{
    CustomerDTO dto = new CustomerDTO();
    //Get value from ID textbox
    int id = 0;
    Int32.TryParse(txtCustomerID.Text, out id);

    dto.CustomerID = id;
    dto.Name = txtName.Text;
    dto.Address = txtAddress.Text;

    var context = new ValidationContext(dto, serviceProvider: null, items: null);
    var results = new List<ValidationResult>();
    var isValid = Validator.TryValidateObject(dto, context, results, true);

    if (!isValid)
    {
        foreach (var validationResult in results)
        {
            Console.WriteLine(validationResult.ErrorMessage);
        }
    }
}

```

- 6- Even when we call “ctx.SaveChanges()” method, EF applies all this validation internally on all attached instances. It throws exception if any validation failure is found. But unfortunately it doesn’t give error message straight forwardly and we’ll have to write some code to extract error messages from that exception. Here we are appending all error messages in a string and then throwing exception of same type but with detailed message. In actual project, we can move all this conversion code into a function and can just call that function directly here.

```

public void SaveCustomer(CustomerDTO dto)
{
    try
    {
        using (var ctx = new MyDBContext()){
            ctx.Customers.Add(dto);
            ctx.SaveChanges();
        }
    }
    catch (DbEntityValidationException ex)
    {
        StringBuilder sb = new StringBuilder();
        foreach (var failure in ex.EntityValidationErrors)
        {
            sb.AppendFormat("{0} failed validation\n", failure.Entry.Entity.GetType());
            foreach (var error in failure.ValidationErrors)
            {
                sb.AppendLine(String.Format("- {0} : {1}", error.PropertyName, error.ErrorMessage));
            }
        }

        throw new DbEntityValidationException("Entity Validation Failed - errors follow:\n"
            + sb.ToString(), ex);
    }
}

```

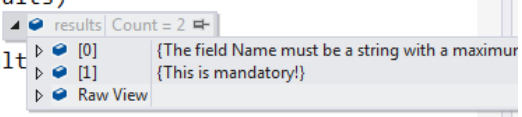
- 7- Run the project, try with different cases (e.g. by providing the name with length > 10, by leaving address empty etc.)
- 8- If you run the project and provide Name of length greater than 10 and leave address empty. You can see error messages while doing debugging.

```
int.TryParse(txtCustomerID.Text, out id);

dto.CustomerID = id;
dto.Name = txtName.Text;
dto.Address = txtAddress.Text;

var context = new ValidationContext(dto, serviceProvider: null, items: null)
var results = new List<ValidationResult>();
var isValid = Validator.TryValidateObject(dto, context, results, true);

if (!isValid)
{
    foreach (var validationResult in results)
    {
        Console.WriteLine(validationResult.ErrorMessage);
    }
}
```



- 9- Now you can check following link to get some more information
 - a. <https://msdn.microsoft.com/en-us/data/jj591583.aspx>

References

<http://www.entityframeworktutorial.net/code-first/dataannotation-in-code-first.aspx>

<https://msdn.microsoft.com/en-us/data/jj591583.aspx>

Fluent API

<http://www.entityframeworktutorial.net/code-first/configure-entity-mappings-using-fluent-api.aspx>