**Code First Conventions:**

We have seen how EF code-first creates db tables from domain classes in the previous section. Here, we will learn about default code-first conventions.

**What is Convention?**

Convention is a set of default rules to configure a conceptual model automatically based on domain class definitions when working with Code First. Code-First conventions are defined in [*System.Data.Entity.ModelConfiguration.Conventions*](http://msdn.microsoft.com/en-us/library/system.data.entity.modelconfiguration.conventions%28v=vs.103%29.aspx) namespace.

Let's see an overview of the various Code First conventions.

**Type Discovery:**

In the previous section, we created context class with DbSet properties for the classes that we want to be part of the model. Code First will create tables for classes included as DbSet properties as we have seen the previous section. Code First also includes any referenced types included in these classes, even if the referenced types are defined in a different assembly.

For example, the following Student entity class includes reference of Teacher class. However, context class does not include Teacher as DbSet property.

public class Student

{

public Student()

{

}

public int StudentID { get; set; }

public string StudentName { get; set; }

public DateTime DateOfBirth { get; set; }

public byte[] Photo { get; set; }

public decimal Height { get; set; }

public float Weight { get; set; }

**public Teacher Teacher { get; set; }**

public Standard Standard { get; set; }

}

public class Teacher

{

public Teacher()

{

}

public int TeacherId { get; set; }

public string TeacherName { get; set; }

}

The following is a context class which does not include Teacher as a DbSet property (entity set)

namespace EF\_Code\_First\_Tutorials

{

public class SchoolContext: DbContext

{

public SchoolContext(): base()

{

}

public DbSet<Student> Students { get; set; }

public DbSet<Standard> Standards { get; set; }

}

}

So, even Teacher is not included as entity set in context class, Code First will include it in conceptual model and create a db table for it as shown below.

Code First also includes derived classes even if context class only includes base class as a DbSet property.

So the conventions for the type discovery are:

1. Code First includes types defined as a DbSet property in context class.
2. Code First includes reference types included in entity types even if they are defined in different assembly.
3. Code First includes derived classes even if only base class is defined as DbSet property.

**Primary Key Convention:**

In the previous section, we have seen that Code First automatically creates Primary Key in each table. The default convention for primary key is that Code First would create a primary key for a property if the property name is Id or <class name>Id (NOT case sensitive). The data type of a primary key property can be anything, but if the type of the primary key property is numeric or GUID it will be configured as an identity column.

If you have defined key property other than Id or <ClassName>Id then ModelValidationException will be thrown. For example, the consider the following Standard class.

public class Standard

{

public Standard()

{

}

**public int StdId { get; set; }**

public string StandardName { get; set; }

public IList<Student> Students { get; set; }

}

As you can see in the above code, Standard class is defined with **StdId** key property. Entity Framework will throw following exception for this.

*'System.Data.Entity.ModelConfiguration.ModelValidationException' occurred in EntityFramework.dll  
EntityType 'Standard' has no key defined. Define the key for this EntityType.*

If you want to define StdId as primary key then you have to use DataAnnotations or Fluent API to configure it as primary key. We will see it how to do it later in these tutorials.

**Relationship Convention:**

Code First infer relationship between the two entities using navigation property. This navigation property can be simaple reference type or collection type. For example, we defined Standard navigation property in Student class and ICollection<Student> navigation property in Standard class. So Code First automatically created one-to-many relationship between Standards and Students db table by inserting Standard\_StandardId foreign key column in the Students table.

public class Student

{

public Student()

{

}

public int StudentID { get; set; }

public string StudentName { get; set; }

public DateTime DateOfBirth { get; set; }

public byte[] Photo { get; set; }

public decimal Height { get; set; }

public float Weight { get; set; }

//Navigation property

**public Standard Standard { get; set; }**

}

public class Standard

{

public Standard()

{

}

public int StandardId { get; set; }

public string StandardName { get; set; }

//Collection navigation property

**public IList<Student> Students { get; set; }**

}

The above entities created the following relationship using Standard\_StandardId foreign key.

Thus, default code first convention for relationship automatically inserts foreign key with **<navigation property Name>\_<primary key property name of navigation property type>** e.g. Standard\_StandardId.

**Foreign key Convention:**

We have seen above that Code First automatically inserts foreign key when it encounter navigation property. It is recommanded to include a foreign key property on the dependent end of a relationship. Consider the following example.

public class Student

{

public Student()

{

}

public int StudentID { get; set; }

public string StudentName { get; set; }

public DateTime DateOfBirth { get; set; }

public byte[] Photo { get; set; }

public decimal Height { get; set; }

public float Weight { get; set; }

//Foreign key for Standard

**public int StandardId { get; set; }**

**public Standard Standard { get; set; }**

}

public class Standard

{

public Standard()

{

}

public int StandardId { get; set; }

public string StandardName { get; set; }

**public IList<Student> Students { get; set; }**

}

As you can see in the above code, Student class includes foreign key StandardId, which is key property in Standard class. So now, Code First will create StandardId column in Students class instead of Standard\_StandardId column as shown below.

Please notice that StandardId foreign key is not null in the above figure because int data type is not nullable.

Code First infers the multiplicity of the relationship based on the nullability of the foreign key. If the property is nullable then the relationship is registered as null, otherwise the relationship is registered as NOT NULL. Modify data type of StandardId property from int to Nullable<int> in the Student class above to create nullable foreign key column in the Students table.

**Complex type Convention:**

Code First creates complex type for the class which does not include key property and also primary key is not registered using data annotations or the fluent API.

This was an overview of code first conventions. These conventions can be override using DataAnnotation or Fluent API. Also, you can define custom conventions for your application using Entity Framework 6.0 onwards.