Contents

A ***ComplexType*** is a class composed of other entity data properties.

The ComplexType

A *ComplexType* is a way to represent complex data within an entity.

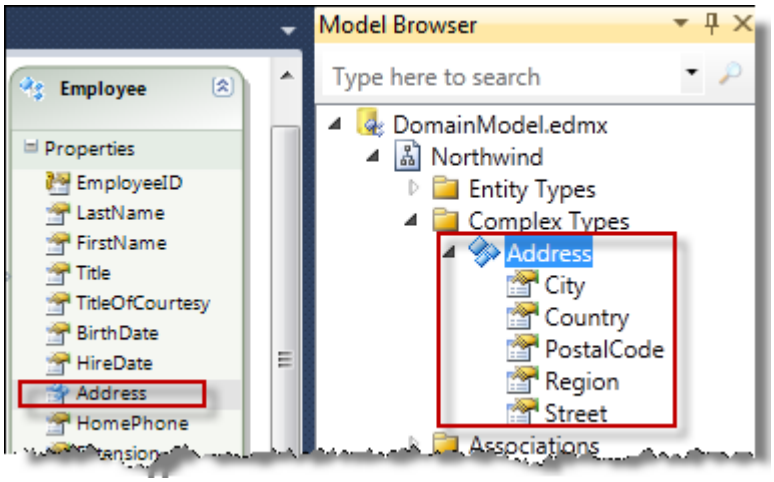
Consider the Northwind "Employee" table for example. The "Employee" table has *FirstName* and *LastName* columns. These become simple data properties of the *Employee* entity.

The "Employee" table also has *Address*, *City*, *Region*, *PostalCode* and *Country* columns. In a default mapping, the *Employee* entity would have five corresponding properties.

But conceptually, these five properties are a single thing, an "address". We'd like to treat them as an *Address* object. We'd like to write *anEmployee.Address* and expect to get an *Address* object in return. We'd like to be able to create a new *Address* object and assign it to *anEmployee.Address*. We might add validation rules to the *Address* class that constrain the *PostalCode* to values that are legitimate for the *Region*.

The *Address* in this example is a *ComplexType* and you can create it in the Entity Framework's EDM Designer. Julie Lerman tells how in a blog post and Matthieu Mezil shows how in his video.

Here is the Northwind *Employee* entity with an *Address* property as seen in the EDM Designer:



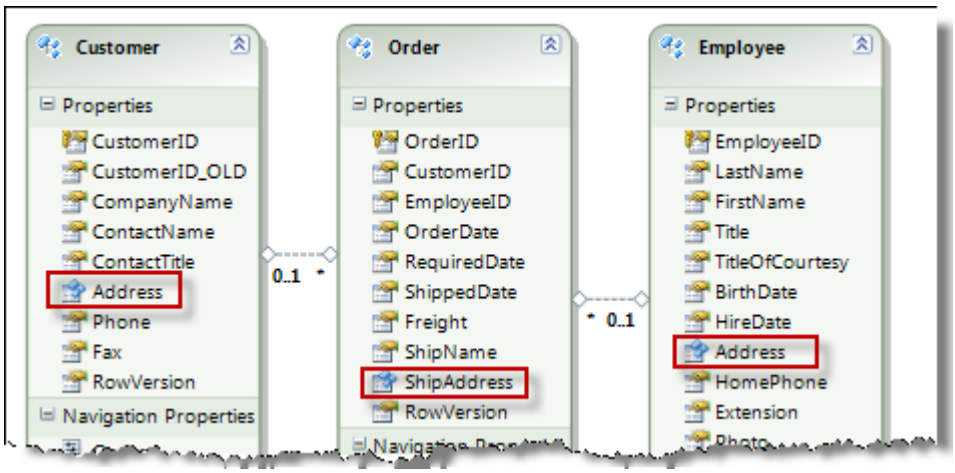
It is not an entity

A *ComplexType* is not an entity. It doesn't map to a complete table row. It lacks a unique key. It has no existence independent from the entity to which it belongs. A *ComplexType* is called a "**value type** because it is entirely defined by the values it holds, in much the way that the number "5" is a value type. There is no table of numbers and no row representing the value "5". In this example there is no "Address" table either.

ComplexType reuse

As it happens, those same five address columns appear in two other Northwind database tables: "Customer" and "Order".

You can re-map their address columns to the *Address* type as seen in this screenshot:



ComplexType classes

DevForce generates *ComplexType* classes into your code file as it does the entity classes. The properties of the generated *ComplexType* classes are similar to generated entity properties and they support notification, validation, and property interception as their entity cousins do. The generated *ComplexType* classes are *partial* so you can extend them with custom logic as you do your entity classes.

An instance has a parent entity

A *ComplexType* instance doesn't stand alone; it belongs to its parent entity. For example, the *Address* of the "Nancy Davolio" *Employee* belongs to the "Nancy Davolio" *Employee* entity. In DevForce , you can always ask a *ComplexType* for its parent.

```
Assert.AreSame(nancy, nancy.Address.ComplexAspect.ParentEntity);
```

```
Assert.AreSame(nancy, nancy.Address.ComplexAspect.ParentEntity)
```

This "belonging" is a necessary consequence of a *ComplexType*. The *City*, *Region*, and other properties of "Nancy's" address actually map to columns on the "Nancy" record in the *Employee* table. There is no *Address* table in this case; only a subset of the *Person* table's columns that we want to treat as an address.

It follows also that a data property that returns a *ComplexType* cannot be null. It's member properties can be null - *nancy.Address.City* may be null. But the *ComplexType* data property itself - *nancy.Address* - cannot be null.

Learn more

Learn more about how to create and work with *ComplexTypes* in the EDM designer.