**Contents**

A reference navigation property and some scalar queries return the **null entity** when there is **no entity to return**. The null entity (aka, the **nullo**) is a special version of an entity class that **represents the missing object**.

In principle, every *Order* should have a parent *Customer*. What if a particular *noCustOrder* doesn't have a parent *Customer*? What should its *Customer* navigation property return?

Should it return *null* (*Nothing* in VB)? Then the statement *noCustOrder.Customer.CompanyName* will throw a *NullReferenceException* because the *null* value doesn't have a *CompanyName* property.

Should we wrap every reference navigation in a giant *try/catch* block? Or should we follow every entity navigation with a test for null? That might be worse than catching an exception. In some situations - in UI code for example - it is difficult to check whether *noCustOrder* has a *Customer*, let alone do something about it.

Fortunately an entity reference navigation neither returns a *null* value nor throws an exception. Instead, when the entitymanager can't find the *Customer*, it returns the *Customer* null entity.

# The null entity

The **null entity** represents the "entity not found". The null entity (known informally as the **nullo**) is a sentinel object that looks and behaves, for the most part, like a real entity instance.

In most respects it **is** a real entity

- it has a specific type (e.g., *Customer*)
- it has the properties, methods, and events of its type.
- it belongs to an *EntityManager*

The null entity has four important differences:

- You can tell its the null entity because *noCustOrder.Customer.EntityAspect.*IsNullEntity returns *true*
- You cannot set a nullo property.
- You cannot create, delete or save a nullo.
- There is only one nullo per type in a given *EntityManager*; you can't have two *Customer* nullos in the same *EntityManager*.

# Actions returning a nullo

Among the ways to get a null entity are

- a reference navigation that cannot find the parent entity
- a scalar query that fails to find the requested entity
- ask the *EntityManager* for it

## Reference navigation

A reference navigation property returns a single entity. The *Order.Customer* is a reference navigation property returning the order's parent *Customer*. Because *noCustOrder* lacks a parent *Customer*, the statement *noCustOrder.Customer* returns a *Customer* nullo.

We don't worry about null entities with collection navigations. A collection navigation property returns a list. The list is never null. It is an empty list if there are no related entities. The statement *emptyOrder.OrderDetails* returns an empty list of *OrderDetail*.

## Scalar query

A scalar query returns a single object. In the following example, the developer executes a scalar query, requesting that the query return either the first entity found or the null entity if there is no match. There is no *Customer* with a zero ID.

```
cust = manager.Customers
            .Where(c => c.CustomerID == 0) // doesn't exist
           .FirstOrNullEntity();
```

```
cust = (From c in manager.Customers  _
       Where c = 0 _' doesn't exist
      Select c) _
       .FirstOrNullEntity()
```

Note that query would have returned *null* - the *cust* value would be *null* - if the developer executed the query with *FirstOrDefault*.

### Get from manager

Although you can't create a nullo, you can ask an *EntityManager* to give you the nullo of a particular type.

```
nullo = manager.GetNullEntity<Customer>(); // the Customer nullo
```

```
nullo = manager.GetNullEntity(Of Customer)() ' the Customer nullo
```

## Nullos belong to an EntityManager

A null entity always belongs to a particular *EntityManager*, the **same *EntityManager*** as the entity which produced it. The nullo from *noCustOrder.Customer* belongs to *noCustOrder*'s manager.

While there can be only one *Customer* nullo per *EntityManager* there can be two nullo *Customer* instances ... if there are two *EntityManagers*.

The requirement that a null entity belong to cache has an unexpected implication. What if the *noCustOrder* is not in cache ... if it is detached?

Then the statement *noCustOrder.Customer* returns the *null* value. You won't get a nullo from a detached entity. A detached entity can't return a nullo because a **null entity** always belongs to a particular *EntityManager* and a detached entity doesn't have an *EntityManager*. DevForce doesn't know what *EntityManager* to use. All it can do is return *null*.

## The default nullo

Every entity class defines its own null entity.

You don't have to do anything special. By default, its simple data properties return the default value for the property's type. An *int* returns 0. A *string* returns the empty string. A property returning a native .NET object type returns a null object of that type; a *nullable int* returns *null*.

Navigation properties are different.

- A nullo's reference navigation property returns another nullo. The statement *noCustOrder.Customer.Region* returns the *Region* nullo. This makes it safe to chain nullos as in *noCustOrder.Customer.Region.Country.CountryName*.
- A collection navigation property returns an empty collection. The statement *noCustOrder.Customer.Orders* returns an empty list of *Orders*.

ComplexType properties return a real *ComplexType* object with property values like a nullo.

## Configure the nullo

The *CompanyName* property of the *Customer* nullo returns an empty string. Suppose you prefer that it return a canned value such as "N/A" or "[none]". You can change the property values of a nullo if you don't like the defaults by overriding the *UpdateNullEntity* method in the partial class.