Working with Relationships in ObjectStateManager

You’ve seen the `RelationshipEntry` in `ObjectStateManager`. As with `EntityEntry` types, the debugger doesn’t provide a lot of information, especially critical information, to help you identify which entities the relationship is for.

You can access this information using `CurrentValues`, which returns `EntityKeys` in the first and second index positions. Because a relationship has two ends, each set has only two fields containing the `EntityKey` of the entity on each end of the relationship.

Although you will also find the `EntityKeys` in the `OriginalValues` (unless the relationship is `Added`), the `OriginalValues` are not truly viable. The property exists because it is there for all `EntityStateObjects`, but you should not rely on it for `RelationshipEntries`. Stick with the `CurrentValues`.

Because the `RelationshipEntry` describes a relationship between two entities, the `EntityKeys` found within the `CurrentValues` will match up with `EntityKeys` of `ObjectStateEntries` in the context. Figure 17-6 shows a `RelationshipEntry` that defines the relationship between a `Customer` and a `Reservation`.

![Figure 17-6. A RelationshipEntry defining a relationship between two entities using EntityKeys](image)

Object Services uses the values in the `RelationshipEntry` to determine how to build graphs with the entities in the context. It also uses this information to build commands that involve foreign keys when `SaveChanges` is called. If you need to work with code generically, you can take advantage of the `RelationshipEntries` as well.

**RelationshipEntry EntityState**

When a new relationship is created between entities, a `RelationshipEntry` is created and its `EntityState` is `Added`. 
The EntityState of a RelationshipEntry that is created of a graph being added to the context is Unchanged. This is also true for related entities that were returned from a query.

When a relationship is removed (e.g., you remove a Reservation from a Customer’s Reservations collection), the existing RelationshipEntry’s EntityState becomes Deleted.

If you change a relationship (e.g., move a payment from one reservation to another), the existing relationship is marked Deleted and a new relationship is created with its EntityState set to Added.

A RelationshipEntry will never have a Modified EntityState.

Inspecting the RelationshipEntries

You can filter the RelationshipEntries using the IsRelationship property. Then you can start digging into the values for each end of the current state of the relationship and the original state. Example 17-22 uses the GetObjectStateEntries overload to return entries from an already populated context, regardless of their EntityState. Then it filters for only those that are relationships.

Example 17-22. Inspecting the RelationshipEntry objects

C#  
For Each relEntry In _
   (From ose In context.ObjectStateManager.GetObjectStateEntries() Where ose.IsRelationship)
   Dim currRelEndA As EntityKey = relEntry.CurrentValues(0)
   Dim currRelEndB As EntityKey = relEntry.CurrentValues(1)
Next

C# foreach (var relEntry in
   (from ose in context.ObjectStateManager.GetObjectStateEntries()
      where ose.IsRelationship
      select ose))
{
   EntityState currRelEndA = relEntry.CurrentValues[0];
   EntityState currRelEndB = relEntry.CurrentValues[1];
}

Figure 17-7 shows the last value, the second end of the original values of the relationship, which, as promised, is an EntityKey. You can see that the EntityKey is for a CustomerType. The second CurrentValues item contains an EntityKey for the Customer entity, which is attached to this CustomerType in this particular relationship.

The EntityKeys provide the common thread throughout the graph. Now, given an entity, you can take its key, query the RelationshipEntries to find the relationships that it is part of, and from those relationships, find the other ends.
Locating relationships for an entity

You can search relationship entries for an EntityKey to see which relationships a particular entity is involved in. Example 17-23 is a handy extension method for ObjectStateEntry that searches a relationship for a given EntityKey. If the EntityKey does not exist, the result will be null. If it does exist, rather than just returning a Boolean of True, the result will be the ordinal representing the position (0 or 1) of the EntityKey in CurrentValues. This way, not only do you know that the entity is involved in that relationship, but also you can use the ordinal to retrieve the EntityKey of the related item. Notice in the example that it's necessary to cast the item to an EntityKey.

Example 17-23. Finding relationships with a particular entity

```csharp
public static Nullable<int> KeyIsinRelationship(this ObjectStateEntry relatedEnd, EntityKey eKey)
{
    //check currentvalues 0 and 1 for this entity key
    if (((EntityKey)(relatedEnd.CurrentValues[0])) == eKey)
        return 0;
    else if (((EntityKey)(relatedEnd.CurrentValues[1])) == eKey)
        return 1;
    else
        return null;
}
```
If you had a **Customer** entity in the context, you could use the extension method to help you find all of the entities that are related to the **Customer**. Recall that the **ApplyPropertyChanges** method that you learned about in Chapter 9 only affects scalar values. If you were writing a method to apply changes throughout a graph and you needed the method to be generic, you could use this technique to discover additional entities in the graph that should have changes applied as well.

### Why an Extension Method?

Why am I creating extension methods rather than regular methods? In the cases discussed in this section, it’s for discoverability. By creating an extension method specifically for an **ObjectStateEntry**, you can find that method very easily. Even if you didn’t know it was there, IntelliSense would show it to you and make you aware. But there is a tendency to overuse extension methods because they are handy and fun to write. So, be prepared to justify using it, even if you need to make that justification only to yourself.

Example 17-24 iterates through the **RelationshipEntries** looking for a relationship that contains a particular entity. This example excludes **Deleted** entries. It then grabs the **EntityKey** of the other related end and gets the related entity from the context.

Remember that the purpose of this code is to be generic, which is why you see **EntityObject** being used rather than a particular entity type.

**Example 17-24. Using the KeysInRelationship method to find the other end of a relationship**

```csharp
Dim osm = context.ObjectStateManager
For Each relEntry In _
    (From entry In osm.GetObjectStateEntries _
        (EntityState.Unchanged Or EntityState.Added) _
        Where entry.IsRelationship)
    Dim otherKey As EntityKey
    Dim otherEntity As EntityObject
    Dim pmtOrdinal = relEntry.KeyIsInRelationship(entityKeyToFind)
    If pmtOrdinal.HasValue Then
        'get entitykey of other end
        otherKey = CType(relEntry.CurrentValues(1), EntityKey)
    Else
        otherKey = CType(relEntry.CurrentValues(0), EntityKey)
    End If
    'get the entity on other end
    otherEntity = CType(context.GetObjectByKey(otherKey), EntityObject)
End If
Next
```

```vbnet
Dim osm = context.ObjectStateManager
For Each relEntry In _
    (From entry In osm.GetObjectStateEntries _
        (EntityState.Unchanged Or EntityState.Added) _
        Where entry.IsRelationship)
    Dim otherKey As EntityKey
    Dim otherEntity As EntityObject
    Dim pmtOrdinal = relEntry.KeyIsInRelationship(entityKeyToFind)
    If pmtOrdinal.HasValue Then
        'get entitykey of other end
        otherKey = CType(relEntry.CurrentValues(1), EntityKey)
    Else
        otherKey = CType(relEntry.CurrentValues(0), EntityKey)
    End If
    'get the entity on other end
    otherEntity = CType(context.GetObjectByKey(otherKey), EntityObject)
End If
Next
```
where entry.IsRelationship select entry))
{
  EntityKey otherKey = null;
  EntityObject otherEntity = null;
  var pmtOrdinal = relEntry.KeyIsinRelationship(PaymentKey);
  if (pmtOrdinal.HasValue)
  {
    if (pmtOrdinal == 0)
    //get entitykey of other end
    
    otherKey = (EntityKey)(relEntry.CurrentValues[1]);
    else
    otherKey = (EntityKey)(relEntry.CurrentValues[0]);
    otherEntity = (EntityObject)(context.GetObjectByKey(otherKey));
  }
}

Building graphs directly with the RelationshipManager

It is possible to get your hands on an instance of the RelationshipManager to build graphs on the fly, creating RelationshipEntries directly in your code.

The RelationshipManager’s entry point is through the IEntityWithRelationships interface. Every EntityObject implements this interface, and any custom objects that you build will need to implement it as well if you want to have relationships managed by Object Services.

The entity does not need to be attached to anObjectContext to get the RelationshipManager.

To get the IEntityRelationship view of an existing entity, cast the entity to IEntityRelationship. From there, you can get a RelationshipManager associated specifically with your entity.

Example 17-25 gets a RelationshipManager for an existing instance of a Payment object, represented by the variable pmt.

Example 17-25. Getting the RelationshipManager for an instance of a Payment

```csharp
Dim pmtRelMgr = CType(pmt, IEntityWithRelationships).RelationshipManager
```

```vbnet
var pmtRelMgr = (IEntityWithRelationships)pmt.RelationshipManager;
```

Once you have the RelationshipManager, the next step is to get a reference to the other end of the relationship that you want to add. To do this, you need to identify which association and which end of the association you want to work with. Unfortunately, you won’t be able to do this in a strongly typed way. You’ll have to use a string to specify the association’s name.

In Example 17-26, the goal is to add a Reservation to the Payment used in Example 17-25, so you’ll need to work with the FK_Payments_Reservations association and add it to the “Reservations” end.
Some of the tricks that RelationshipManager performs do not require the ObjectContext. This is handy to know if you are building generic code without the aid of the ObjectContext. Check out the MSDN Entity Framework forum post titled “Remove Associations from Entity,” which shows how to use IRelatedEnd with reflection to strip related data from an entity. (When reading this forum thread, which I started, you’ll also see that I learned this lesson the hard way, too.)

RelatedEnd has an Add method, which is the final call you’ll need to make. Example 17-26 shows how you can add the existing Reservation entity to the RelatedEnd. This will create a new relationship between the Payment entity and the Reservation entity.

Example 17-26. Creating a relationship on the fly using the RelationshipManager created in Example 17-25

```csharp
Dim resRelEnd As IRelatedEnd = 
    pmtRelMgr.GetRelatedEnd(FK_Payments_Reservations, Reservations)
resRelEnd.Add(myReservation)
```

This method of building graphs works exactly the same as if you had called pmt.Reservation=myReservation. If neither object is attached to the context, you will still get a graph; however, no RelationshipEntry will be created in the context. If only one of the entities is attached to the context, the other one will be pulled in and given the appropriate EntityState (Attached or Added).

```
RelatedEnd also has a Remove method, so you can deconstruct graphs as well.
```

ObjectStateManager and SavingChanges

One of the most useful places to take advantage of the ObjectStateManager is in the ObjectContext.SavingChanges event handler. You saw some examples of using the SavingChanges event in Chapter 10, where you used GetObjectStateEntries to find Modified and Added entries, and then to do some last-minute work on particular types.

```
The other events, Propertychanged/Changing and AssociationChanged,
do not have access to the ObjectContext or its ObjectStateManager, so
you won’t include this type of functionality in those event handlers.
```