Contents

- Multi-manager scenarios
  - Sandbox editor
  - Search manager
- Create a second EntityManager

Create **multiple *EntityManagers*** when you need to isolate one set of entities (and their changes) from another set of unmodified entities. The need typically arises when launching an editor in a "sandbox"; changes within the editor are provisional and confined to the editor scope.

Some applications only need a single *EntityManager* instance. That instance can hold in its cache every entity the application ever needs. The user queries, changes, and saves entities using that one manager. An ASP application should only require one manager.

However, we often see multiple *EntityManagers* in smart client and RIA applications. We tend to cache entities for longer periods - often the entire session - and soon discover that we need to maintain separate, isolated caches.

Entities are unique within a manager's cache - there can be only one *Customer* with Id=42 in a particular cache. If I change the customer's name from "Acme" to "Beta", that change is visible to every view referencing that customer. That isn't always desirable. We may want two copies of *Customer #42* that we use for different purposes.

Multiple managers help us work with multiple copies of the same entity and they provide entity isolation because they each have their own caches.

Multi-manager scenarios

Two multi-manager scenarios are common:

1. "Sandbox editor" - when you want to isolate the edited entities from the main set of entities. You don't want the changes to propagate to the main set until they are saved.
2. "Search manager" - a manager's cache can grow quite large as the user searches for items as when browing a product catalog. Once specific items are selected, the interim search results are no longer interesting and they hog memory. It's easy and harmless to throw them away if they are confined to a search-only manager that you can clear.

Sandbox editor

Customer #42 can appear twice in the same *application* if it is cached in *two separate managers*. Suppose the main manager 'A' supports a customer selection view and manager 'B' is the isolated manager within the "sandbox editor". Changing the customer's name in the editor from "Acme" to "Beta" effects the *Customer #42* in manager 'B' and is visible only within the editor. Users looking at the customer selection view will see the *Customer #42* in manager 'A' whose name remains "Acme".

If the user saves the name change in the editor, the stored name of *Customer #42* becomes "Beta". It's still "Acme" back in manager 'A' and in the customer selection view. The situation for managers 'A' and 'B' is analagous to two separate users who are looking at the same customers.

If the user should see a refreshed and current view of *Customer #42*, the developer must take steps to update that customer in manager 'A', perhaps through a messaging mechanism.

See the blog post *"Sandbox" Editors with ClientUI Contacts* for an example and discussion.

Take this a step further. Suppose the application can edit multiple customers at the same time. It might open an editor for *Customer #42* and another for *Customer #007*. Now there are three managers in play, each with their isolated caches. The user can toggle among them, making changes to #42, then #007, then back to #42, then canceling the #007 edit session, and finally saving changes to #42 which propagate back to the main customer selection view.

Multiple managers are handy when you have to juggle ongoing tasks like this.

Search manager

Imagine a shopping application in which manager 'A' holds products to purchase. The user, while searching for more items to purchase, is busily querying products, perhaps hundreds of them, most of which will not end up in the shopping cart. The volume could be substantial and the toll on local memory severe. When the user is ready to checkout, those unwanted products are dead weight.

You could try to identify and purge them from manager 'A', taking care to separate the product entities mentioned in the cart from the vast majority of unwanted products.

It *might* be easier to conduct the search using a manager 'B'. When the user picks an item, you copy it from 'B' to 'A' (using *ImportEntities*). When the shopping is over, you clear manager 'B'.

Create a second EntityManager

Create a second manager the same way you did the first. You have the same range of options.

Consider using the handy copy constructor that creates a new *EntityManager* configured like its source.

```
var secondManager = new EntityManager(firstManager);
```

```
Dim SecondManager = New EntityManager(FirstManager)
```

The second EntityManager will have the same settings and security context as the first EntityManager, but will contain no data.

The copy constructor is especially useful in server-side interceptors and methods, where DevForce provides a special "server-side" EntityManager. Since this EntityManager is not sub-typed, it will not contain any of your EntityQuery methods, but you can easily remedy this by constructing your sub-typed EntityManager from the server-side manager provided using the copy constructor.