# Development-Which cache to use with EOs?

## Introduction

We won't talk about the EOF cache because you have no control on it. We'll talk here about the managed caches you implement to speed up access to the most used EOs also known as cache level 1.
First we will talk about ERXEnterpriseObjectCache and the issues you can encounter then we will show you wich library you can use and the constraints it implies.

## ERXEnterpriseObjectCache

The ERXEnterpriseObjectCache class is present for a while in Wonder. It's dedicated to EOs so you can't use it for any other object. It has the merit to be available and is sometimes the only solution. But there are several issues:

1. the most important issue is there is no eviction policy
2. the EOs in the cache can be updated in your back
3. the design is a bit old

As there is no eviction policy, the cache can only grow. Of course, you can clear the cache from time to time but it looks more like a bad workaround. And combined with the point 3 we'll see later, the cache can take a lot of memory at startup.

Regarding the second point, take a look at Chuck's response here to have a better idea of what's going on. I would add that the cache seems to be updated also when a notification is sent by another ec while the saveChanges() method is called. I just noticed that point when we trace the code or in the stack trace when there is an exception.

Finally, the design of the cache is a little bit outdated and some parameters are incorrectly named. For example, if you create a cache with an attribute key, everything is working magically. ERXEnterpriseObjectCache will fetch automatically the missing EO and returns the value to the client.

But, sometimes (often?) the key is not a simple attribute key but a compound key (I recommend to use ERXMultiKey for that purpose). In that case, there is no magic.

First, you have to understand that ERXEnterpriseObjectCache has no way to fetch the missing EOs. So you have to handle it manually.

Secondly you have to use the constructor with the parameter shouldFetchInitialValues set to true which is incorrectly named (more later). As a result, the EOs are entirely fetched when the cache is create and if there are thousands of records or more, no luck!

Why shouldFetchInitialValues is incorrectly named IMO? Because it implies 2 behaviors not reflected by the parameter name: one is to warm up the cache and the second implicit behavior is to tell ERXEnterpriseObjectCache to not fetch objects. When you look at the API, it's not obvious.

Of course, the temptation is high to enhance ERXEnterpriseObjectCache but if your need is to cache globalID rather than EOs, you can use an interesting library: Guava.

## Guava Cache

Guava provides a bunch of interesting classes (the collection classes are a must). But because it relies on java 6, it can't be included in Wonder so far.

What we will see here, is a subset of Guava: Guava cache.

Guava cache provides everything you need to manage efficiently your cache:

1. eviction policy of your choice
2. a coherent way to fill in the cache
3. tools to watch the cache

However, Guava cache doesn't replace ERXEnterpriseObjectCache if you absolutely need to cache EOs (except if you associate an editing context with your cache, it's worth a try). So instead of caching EOs, you put GlobalID. But you have to pay attention to the way you link objects together as we will see at the end.

Back to the advantages of Guava cache. The first one is the eviction policy. Basically, you can define a max size of the cache in terms of elements. You can also add a constraint on the weigh of the object. And there is also a timed eviction. Read the documentation to get all details on the eviction policies.

The second advantage is the way you can fill in the cache. The first step is to create a loader then associate the loader when you create the cache (see example below).

Finally Guava provides tools to track evicted objects (just implement a listener) and statistics.

The example below shows you the minimum to learn and build a working cache. The method load() of CarEntityLoader class is called when a car is not in the cache (var carCache).

```
public class BasicCacheManager {
```

```
   private final LoadingCache<String, EOGlobalID> carCache;

   private static class CarEntityLoader extends CacheLoader<ERXMultiKey, EOGlobalID> {
      @Override
      public EOGlobalID load(ERXMultiKey keyForCache) {
         String value1 = (String) keyForCache.keys()[0];
         String value2 = (String) keyForCache.keys()[1];

         EOQualifier qual = CarEntity.Keys.MODEL_KEY.eq(value1).and(CarEntity.Keys.TYPE_KEY.is(value2));
         EOEditingContext ec = aFactory.editingContext();
         CarEntity aCar;

         ec.lock();
         try {
            aCar = CarEntity.fetchCar(ec, qual);
            if (aCar == null) {
               aCar = CarEntity.createAndInsertCar(ec);
               aCar.setModel(value1);
               aCar.setType(value2);
               ec.saveChanges();
           }
         } finally {
            ec.unlock();
         }
         return aCar.permanentGlobalID();
      }
   }

   public NBCacheManager() {
      carCache = CacheBuilder.newBuilder()
                         .maximumSize(500)
                         .build(new CarEntityLoader());
   }

   public CarEntity carForModelAndType(final EOEditingContext ec, final String aModel, String aType) {
      String model = Preconditions.checkNotNull(aModel);
      String type = Preconditions.checkNotNull(aType);
      ERXMultiKey key4Cache = new ERXMultiKey(model, type);

      EOGlobalID aCarGlobalID = carCache.getUnchecked(key4Cache);
      return (CarEntity) ec.faultForGlobalID(aCarGlobalID, ec);
   }
}
```

Note if you need to warm up the cache, implement the getAll() method in your loader.

However, using globalID may have a major drawback because EOF sends a SQL request when it needs to resolve the fault. And if it resolves the fault each time you look up the cache, there is no gain.

So think twice how you will use your EOs. But often, you use a cache to retrieve quickly objects you add to relationships. In this case there are some good news.
First check your code if you use addObjectToBothSidesOfRelationshipWithKey() because EOF will resolve the fault. **Always**. If you can, avoid to call this method but call takeStoredValueForKey(). Even if you call takeStoredValueForKey(), if you look at the code in the ERXGenericRecord class, you'll see it will update the inverse relationship if you set the property er.extensions.ERXEnterpriseObject.updateInverseRelationships to true. Instead, set this property to false.

**To sum up:** if your cache contains EOs you use mainly to set relationship with other objects, and if your application tolerates to not update the inverse relationship, you can advantageously use Guava. And don't forget you can cache other objects which are not Enterprise Objects.