

## Multiple datasources, PersistenceMapperMultiDb

In ECO, a single ecospace can use multiple databases to load/store the objects. A few restrictions apply:

- all embedded parts of an object must reside in a single database (this includes attributes and embedded single links, including inherited such).
- all databases must be supported by Eco (either using PersistenceMapperBdp, PersistenceMapperSql or some other custom made PersistenceMapper that derives from PersistenceMapperDb)

The databases can use either the standard ECO O/R mapping (using ECO\_ID columns etc) or use custom mapping (specified using an XML-file).

The key component to tap into this is the PersistenceMapperMultiDb. It acts as a kind of aggregator for all the databases that the ecospace is going to use. This can be used directly in an ecospace.

Ok, now you have a PersistenceMapperMultiDb, now you have to make it aggregate some databases. The new component has a collection property called "PersistenceMappers". This is a collection of Name/PersistenceMapper pairs. You need to add a PersistenceMapperBdp/Sql/whatever for each database you need to retrieve data from. You can use any combination of persistence mappers, depending on the databases you plan to connect to. Each database should be assigned a unique symbolic name (the name property in the collection). This can be any name, such as "customerdb". It does not have to have any relationship with the connection string of the database (but it usually makes sense to use something from the connection string).

By now you have a set of databases, each with a unique name. Let's call this name the "DatabaseName". Now you have to tell ECO where each class comes from (remember, each class must come from exactly one database. You can not have customers in multiple databases - unless you model them as different classes of course).

The location of an object can be specified in several different ways. If the class is "default mapped" using standard ECO mapping, you simply set the tagged value `Eco.Database` to the appropriate database name. If you want to use custom mapping (an XML-file), you have two options, either you can specify an XML-file for each database that you have custom mapping for, and then add a FileMappingProvider to each PersistenceMapper, and set the correct filenames.

**Important:** For each PersistenceMapper that specifies its own custom mapping, you have to set the boolean property `ImportConfig` in the collection of the PersistenceMapperMultiDb to `true`. You can also specify a compound mapping file where you specify the custom mapping for all databases in your system. If you do this, you must specify the database name for each class in the mapping file like this:

```
<ClassDef Name="LineItem" Database="Orders">
  ...
</ClassDef>
```

If you have a compound mapping file, then you have to set up a FileMappingProvider for the PersistenceMapperMultiDb (use the RuntimeMappingProvider property).

You can have associations between objects in different databases. The classes can be modelled in separate packages if you want to. If the association is 1--1 or 1--n you just have to make sure you set the "embedded" flag correctly on the side that exists on one of the databases. If the association is n--m then you have to specify the O/R mapping for the link-class, but this is just like for any custom mapped model.

I'll also mention a few words about transactions. ECO can operate in three different transaction modes when it uses multiple databases:

- No transactions - no transactions are used whatsoever, this is normally not something you want.
- Local transactions - ECO uses the local transactions available in each database. All update operations will be performed first, and then all databases will be committed (unless an exception has occurred during the update, then all databases will be rolled back. This can cause an inconsistency if the second commit fails as the first commit can not be rolled back anymore.
- Distributed transactions - ECO uses distributed transactions (also called two-phase commit) to control the behaviour of the update. If any of the databases fails to commit, all will be rolled back. This is currently only supported for PersistenceMapperSql since BDP does not support this. The default implementation uses the utility classes of the System.EnterpriseServices namespace (this is a wrapper for COM+ distributed transactions, or MSDTC). It is possible to implement support for another distributed transaction manager, but currently I don't know of any other. If anyone is interested in details, please let me know.

Last Modified By: Jonas Högström, den 18 april 2008

<http://www.capableobjects.com/apps/InstantKB13/KnowledgebaseArticle50007.aspx>

den 7 februari 2012