CayenneTapestrySqueezer

NOTE: This information is outdated and only applies to Tapestry 4, not Tapestry 5.

How To Get Cayenne to Work with Tapestry DataSqueezer

To fully benefit from the information below you should be familiar with the workings of the DataSqueezer and Cayenne. The key concept to remember during this explanation is that there must be ONE unique Datacontext for each users session in the Tapestry webapp. Now if you are not using the DataSqueezer, it suffices to simply put the Datacontext into your VisitObject. This results in a unique Datacontext for each users session. However there is an additional hurdle when using Tapestry's DataSqueezer to serialize and de-serialize Cayenne Dataobjects. The difficulty is that the ISqueezeAdaptor interface does not provide a mechanism for accessing the VisitObject, hence no access to the Datacontext, hence no way to de-serialize your objects.

The solution comes from the use of the ThreadLocal object. By creating a reference to a ThreadLocal object you can gain global access to the VisitObject which solves the problem mentioned above. Since code is worth a thousand words, we will take a step by step journey to getting Cayenne to work with the DataSqueezer.

Create UserContext

First create the class that allows you to bind variables to the current thread and will henceforth be used to provide global access to whatever the current visit class is:

```
public class UserContext {

 static ThreadLocal _visitLocal = new ThreadLocal();

 public static Object getVisit() {
     return (Visit) _visitLocal.get();
 }

 public static void setVisit(Object visit) {
     _visitLocal.set(visit);
 }
}
```

You should take note that everything here is static and so can be accessed from anywhere in your Tapestry application.

Create Custom Engine

Next we need to Create a Custom Engine to initialize the Usercontext with the VisitObject and to register our custom Squeeze adapter.

NOTE:

To register a custom engine, go to your foo.application file and change the engine-class attribute of your application to the following:

```
<application name="foo" engine-class="com.acme.superApp.CustomEngine">
```

If you already knew how to do this, this may seem like a silly step to mention but I couldn't find a single place in the TIA book where it tells you how to do this, nor could I find it on the net.

```
public class CustomEngine extends BaseEngine {

 protected void setupForRequest(RequestContext context)
 {
   if (getVisit() != null)
     UserContext.setVisit(getVisit());

   super.setupForRequest(context);
 }

 public DataSqueezer createDataSqueezer()
 {
   DataSqueezer defaultDataSqueezer = super.createDataSqueezer();
   DataSqueezeAdaptor customAdaptor = new DataSqueezeAdaptor();
   customAdaptor.register(defaultDataSqueezer);
   return defaultDataSqueezer;
 }
}
```

The setupforrequest method is where the Usercontext is initialized with the visit class currently registered with the Engine.

```
There is a scenario where initializing your userContext may need to be done in the createVisit method:

 protected Object createVisit(IRequestCycle cycle)
 {
   Visit visit = (Visit) super.createVisit(cycle);
   UserContext.setVisit(visit);
 }

The reason for having the VisitObject also initialized in the createVisit method is to account for when users
visit the site with a bookmark before logging in. When the user uses his/her bookmark, the visit will be null
at the time setupForRequest() is called.
```

Access Visit from Squeezer

Now because of this magic, you can now access the Visit class in your adaptors' unsqueeze method:

NOTE:

This is not a complete implementation of the squeezer, for a more complete implementation see CayenneDataObjectSqueezeAdaptor

```
public class DataSqueezeAdaptor implements ISqueezeAdaptor {

 private static final String prefix          = "D";
 private static final String SEPERATOR       = "_";

 public DataSqueezeAdaptor()
 {
   super();
 }

 public String squeeze(DataSqueezer squeezer, Object data) throws IOException
 {
   if (data instanceof DataObject)
     return prefix + data.getClass().getName() + SEPERATOR + DataObjectUtils.pkForObject((DataObject)data);

   return null;

 }

 public Object unsqueeze(DataSqueezer squeezer, String string)
     throws IOException
 {
    String classAndId = string.substring(prefix.length());
    int seperatorIndex = classAndId.indexOf(SEPERATOR);
    String className = classAndId.substring(0,seperatorIndex);
    int id = Integer.parseInt(classAndId.substring(seperatorIndex + 1, classAndId.length()-1));

   try
   {
     return DataObjectUtils.objectForPK(getDataContext(), Class.forName(className), id);
   } catch (Exception e)
   {
     return null;
   }
 }

 private Visit getVisit()
 {
   return (Visit) UserContext.getVisit();
 }

 public void register(DataSqueezer squeezer)
 {
   squeezer.register(prefix, IDataObject.class, this);
 }

 private DataContext getDataContext()
 {
   return getVisit().getDataContext();
 }
}
```