DataSqueezer

NOTE: This is outdated information that applies only to Tapestry 4.

How Tapestry Deals objects in HTML

One of the features that really makes Tapestry stand out is "objects is objects". In order to support this very cool feature, Tapestry must represent an object using text when the object is part of an HTML page. The DataSqueezer has the job of outputing text for objects on the way out the door, and input objects from text on the way in the door.

The line marked <<<<< problem is an example of using this feature.

```
    <a href="#" jwcid="@DirectLink"
         parameters="ognl:components.table.tableRow"   <<<<<<<< problem
         listener="ognl:listeners.viewObject">Details</a>
```

The DataSqueezer works by examining the class and interface hierachy of the Object looking for a registred implementation of the ISqeezeAdaptor interface. The Serializable interface is registred with the DataSqueezer.

DataSqueezer can use the Serializeable Interface

The ISqueezeAdaptor implementation for the Serializable interface does the following:

Serializable Object Outbound to an HTML page

1. The data is serialized to a byte stream.
2. The byte stream is Base64 encoded.
3. The Base64 encoded data is sent out the door

Serializable Object Inbound from an HTML request

1. The Base64 text is decoded into a byte stream.
2. The byte stream is read using an ObjectInputStream.
3. The object is handed to the application.

Another DataSqueezer tactic

A Base64 encoded byte stream can be quite a bit of text. So, another route you can go is registering your own DataSqueezer that does what ever you want. For instance, the following example does this for objects rarely changing configuration stored in a database. On the way out, the ISqueezeAdaptor encodes information needed by an ORM to load it from the database including the class and object identifier. On the way in, the ISqueezeAdaptor decodes the class and object id from the string and asks the ORM to load the object.

Tapestry takes care of calling the DataSqueezer so all your application ever does is reference an object.

There are a couple of tricks to constructing a DataSqueezer ISqueezeAdaptor.

- They are registered in the method org.apache.tapestry.engine.AbstractEngine#createDataSqueezer()
- You have to find an unused prefix.
- You must make the prefix part of the outbound String.

Example Data Squeezer

```
    public DataSqueezer createDataSqueezer() {
        ISqueezeAdaptor keySqueezer = new ISqueezeAdaptor() {

            private static final String PREFIX = "k";

            public void register(DataSqueezer squeezer) {
                squeezer.register(PREFIX, EnterpriseObject.class, this);
            }

            public String squeeze(DataSqueezer squeezer, Object data)
                    throws IOException {
                ObjectFactory factory = ((Global) getGlobal())
                        .getObjectFactory();
                StringBuffer sb = new StringBuffer();
                sb.append(PREFIX).append(factory.getKey(data));
                return sb.toString();
            }

            public Object unsqueeze(DataSqueezer squeezer, String string)
                    throws IOException {
                ObjectFactory factory = ((Global) getGlobal())
                        .getObjectFactory();
                return factory.getObject(string.substring(1));
            }
        };
        ISqueezeAdaptor[] adapt = {keySqueezer};
        return new DataSqueezer(getResourceResolver(), adapt);
    }
```

Example PropertySelection using Data Squeezer

This example shows how create a DataSqueezer enabled PropertySelection. DataObject is the base class for all the squeezeable objects, and DataObject Adaptor is the squeeze adaptor that uses a orm service to load and save DataObjects. In contrast to the first example, the adaptor class is not anonymous because it is used both here and in the engine class (not shown).

```
package foo.web;

import foo.orm.DataObject;
import foo.orm.DataObjectAdaptor;

import java.util.ArrayList;
import java.util.List;

import org.apache.tapestry.form.IPropertySelectionModel;

public class EntitySelectionModel implements IPropertySelectionModel
{
    private static class Entry
    {
            DataObject option;
        String value;
        String label;

        Entry(DataObject option, String value, String label)
        {
                this.option = option;
            this.value = value;
            this.label = label;
        }
    }

    private static final int LIST_SIZE = 20;

    private DataObjectAdaptor adaptor;
    private List<Entry> entries = new ArrayList<Entry>(LIST_SIZE);

    public EntitySelectionModel(DataObjectAdaptor adaptor)
    {
```

```
            this.adaptor= adaptor;
            entries.add(new Entry(null, "", ""));
    }

    public void add(List<? extends DataObject> entities)
    {
            for (DataObject entity : entities)
            {
                    String value = adaptor.squeeze(null, entity);
                    String label = entity.toString();
                    entries.add(new Entry(entity, value, label));
            }
    }

    public int getOptionCount()
    {
        return entries.size();
    }

    public Object getOption(int index)
    {
        return entries.get(index).option;
    }

    public String getLabel(int index)
    {
        return entries.get(index).label;
    }

    public String getValue(int index)
    {
            return entries.get(index).value;
    }

    public Object translateValue(String value)
    {
        if (value.equals(""))
            return null;

        return adaptor.unsqueeze(null, value);
    }
}
```

See Also

CayenneTapestrySqueezer HibernateTapestrySqueezer