Request for a second option when creating a JPA Controller From Entities

Issue #1

Currently the code generated by NetBeans for JPA controllers assumes an environment not managed by a container such as for a desktop or web application for which dependency injection is not normally used. However, when the JPA controller will be used in a managed container such as Payara the controller must be modified. I propose providing a second option in NetBeans to generate controllers that are coded for a container.

Assumption: There is a MySQL database named 'aquarium' with a single table named 'fish'.

Here is the JPA Controller created by NetBeans. The changes appear in the next block of code.

```
import com.kfstandard.entities.Fish;
import com.kfstandard.jpacontroller.exceptions.NonexistentEntityException;
import java.io.Serializable;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Query;
import javax.persistence.EntityNotFoundException;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;

public class FishJpaControllerByNB implements Serializable {

    public FishJpaControllerByNB(EntityManagerFactory emf) {
        this.emf = emf;
    }
    private EntityManagerFactory emf = null;

    public EntityManager getEntityManager() {
        return emf.createEntityManager();
    }

    public void create(Fish fish) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            em.persist(fish);
            em.getTransaction().commit();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

public void edit(Fish fish) throws NonexistentEntityException, Exception {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        fish = em.merge(fish);
        em.getTransaction().commit();
    } catch (Exception ex) {
        String msg = ex.getLocalizedMessage();
        if (msg == null || msg.length() == 0) {
            Integer id = fish.getId();
            if (findFish(id) == null) {
                throw new NonexistentEntityException("The fish with id " + id + " no longer exists.");
            }
        }
        throw ex;
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

public void destroy(Integer id) throws NonexistentEntityException {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        Fish fish;
        try {
            fish = em.getReference(Fish.class, id);
            fish.getId();
        } catch (EntityNotFoundException enfe) {
            throw new NonexistentEntityException("The fish with id " + id + " no longer exists.", enfe);
        }
        em.remove(fish);
        em.getTransaction().commit();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

public List<Fish> findFishEntities() {
    return findFishEntities(true, -1, -1);
}

public List<Fish> findFishEntities(int maxResults, int firstResult) {
    return findFishEntities(false, maxResults, firstResult);
}

private List<Fish> findFishEntities(boolean all, int maxResults, int firstResult) {
    EntityManager em = getEntityManager();
    try {
        CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
        cq.select(cq.from(Fish.class));
        Query q = em.createQuery(cq);
        if (!all) {
            q.setMaxResults(maxResults);
            q.setFirstResult(firstResult);
        }
        return q.getResultList();
    } finally {
        em.close();
    }
}

public Fish findFish(Integer id) {
    EntityManager em = getEntityManager();
    try {
        return em.find(Fish.class, id);
    } finally {
        em.close();
    }
}

public int getFishCount() {
    EntityManager em = getEntityManager();
    try {
        CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
        Root<Fish> rt = cq.from(Fish.class);
        cq.select(em.getCriteriaBuilder().count(rt));
        Query q = em.createQuery(cq);
        return ((Long) q.getSingleResult()).intValue();
    } finally {
        em.close();
    }
}

}
```

Now, here is the modified code to work in a container. Items in bold are my changes or additions. In the queries the try/finally was completely removed because they were only needed when calling upon now removed getEntityManager.

```
import com.kfwebstandard.entities.Fish;
import com.kfwebstandard.exceptions.NonexistentEntityException;
import java.io.Serializable;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.annotation.Resource;
import javax.enterprise.context.SessionScoped;
import javax.inject.Named;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import javax.persistence.EntityNotFoundException;
import javax.persistence.PersistenceContext;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import javax.transaction.HeuristicMixedException;
import javax.transaction.HeuristicRollbackException;
import javax.transaction.NotSupportedException;
import javax.transaction.RollbackException;
import javax.transaction.SystemException;
import javax.transaction.UserTransaction;
```

```
@Named
@SessionScoped
public class FishActionBeanJPA implements Serializable {

    @Resource
    private UserTransaction utx;

    @PersistenceContext(unitName = "fishiesPU")
    private EntityManager em;

    public FishActionBeanJPA() {
    }

    public void create(Fish fish) {
        try {
            utx.begin();
            em.persist(fish);
            utx.commit();
        } catch (NotSupportedException | SystemException | RollbackException | HeuristicMixedException |
                 HeuristicRollbackException | SecurityException | IllegalStateException ex) {
            Logger.getLogger(FishActionBeanJPA.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void edit(Fish fish) throws NonexistentEntityException, Exception {
        try {
            utx.begin();
            fish = em.merge(fish);
            utx.commit();
        } catch (IllegalStateException | SecurityException | HeuristicMixedException | HeuristicRollbackException |
                 NotSupportedException | RollbackException | SystemException ex) {
            String msg = ex.getLocalizedMessage();
            if (msg == null || msg.length() == 0) {
                Integer id = fish.getId();
                if (findFish(id) == null) {
                    throw new NonexistentEntityException("The fish with id " + id + " no longer exists.");
                }
            }
            throw ex;
        }
    }
```

```
public void destroy(Integer id) throws NonexistentEntityException {
    try {
        utx.begin();
        Fish fish;
        try {
            fish = em.getReference(Fish.class, id);
            fish.getId();
        } catch (EntityNotFoundException enfe) {
            throw new NonexistentEntityException("The fish with id " + id + " no longer exists.", enfe);
        }
        em.remove(fish);
        utx.commit();
    } catch (NotSupportedException | SystemException | RollbackException | HeuristicMixedException |
             HeuristicRollbackException | SecurityException | IllegalStateException ex) {
        Logger.getLogger(FishActionBeanJPA.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public List<Fish> findFishEntities() {
    return findFishEntities(true, -1, -1);
}

public List<Fish> findFishEntities(int maxResults, int firstResult) {
    return findFishEntities(false, maxResults, firstResult);
}

private List<Fish> findFishEntities(boolean all, int maxResults, int firstResult) {
    CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
    cq.select(cq.from(Fish.class));
    Query q = em.createQuery(cq);
    if (!all) {
        q.setMaxResults(maxResults);
        q.setFirstResult(firstResult);
    }
    return q.getResultList();
}

public Fish findFish(Integer id) {
    return em.find(Fish.class, id);
}
```

```
    public int getFishCount() {
        CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
        Root<Fish> rt = cq.from(Fish.class);
        cq.select(em.getCriteriaBuilder().count(rt));
        Query q = em.createQuery(cq);
        return ((Long) q.getSingleResult()).intValue();
    }

}
```

One issue to resolve is whether or not there should be a multi-catch when numerous exceptions, mostly from begin() and commit(), are thrown. The alternative is just a catch(Exception ex).

In a multi-table with joins you will find rollback() and its exceptions. In a previous generation of this code there was a named exception for rollback failures called RollbackFailureException but it is not present in this current code generator.

Issue #2

When entities are generated it is usual to request that a persistence.xml file is created. This file contains the connection information extracted from the NetBeans Database Connection. This file also needs to be modified when an application server container is used.

Examples:

Assumption: There is a MySQL database named ‘aquarium’ with a single table named ‘fish’.

The persistence.xml that is generated now is:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="com.kfwebstandard_ApacheJiraReport_war_1.1-SNAPSHOTPU" transaction-type="RESOURCE_LOCAL">
    <class>com.kfstandard.entities.Fish</class>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.jdbc.url"
              value="jdbc:mysql://localhost:3306/aquarium?zeroDateTimeBehavior=CONVERT_TO_NULL"/>
      <property name="javax.persistence.jdbc.user" value="fish"/>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.password" value="kfstandard"/>
    </properties>
  </persistence-unit>
</persistence>
```

What is required is:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
    <persistence-unit name="com.kfwebstandard_ApacheJiraReport_war_1.1-SNAPSHOTPU" transaction-type="JTA">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
        <jta-data-source>java:app/jdbc/myAquarium</jta-data-source>
        <class>com.kfwebstandard.entities.Fish</class>
        <exclude-unlisted-classes>true</exclude-unlisted-classes>
    </persistence-unit>
</persistence>
```

The persistence-unit name is based on the project name and is usually edited to something smaller.

The transaction-type needs to be JTA and not RESOURCE_LOCAL

There needs to be a jta-data-source and this is normally defined in glassfish-resources.xml/payara-resources.xml.

Tests show that the properties for the database connection can be left as they are ignored when using a container. My example has them removed.

Version number should be 2.2 and not 2.1.

Issue #3

If the persistence.xml file is edited to support a container as shown in issue #2 then when JPA controllers are generated the classes contain no code, just an empty class. If I use my modified persistence.xml file and then request a controller for the 'fish' table I get:

```
import java.io.Serializable;

public class FishJpaController implements Serializable {

}
```

As you can see the generator fails.