Member Photo Manager 2.0 Component Specification

1. Design

All changes performed when synchronizing documentation with the version 1.0 of the source code of this component and when fixing some existing documentation issues are marked with **purple**.

All changes made in the version 2.0 are marked with **blue**.

All new items in the version 2.0 are marked with **red**.

A web site needs to allow users to upload their photos for display in their member profile. This component provides JPA entities and manager class for management of the member photos uploaded to the web site. It will be used by other components which will provide the servlets which will be used to process user actions.

Design considerations

The component provides entity classes corresponding to the attached class diagram in the attached TCUML file: Image and MemberImage. These classes follow JavaBeans conventions and are valid JPA entities. PagedResult<T> also follows JavaBeans conventions, but is not a JPA entity.

The component provides MemberPhotoManager manager class implementing the MemberPhotoManager interface shown in the attached class diagram in the attached TCUML file. The interface itself also is provided in this component. This class is implemented in such a way that Spring IoC (Inversion of Control) container can be used to inject dependencies into its objects.

The clients of this component most likely will handle transactions using declarative transaction handling provided by Spring framework. So this component does not deal with transaction, i.e. methods of manager class assume that they are called inside of transaction.

Component Usage:

Members are allowed to submit photos to the TopCoder site to be displayed. Currently this process is handled by hand. This component will be used in automated photo upload solution to be deployed at TopCoder site.

Other design considerations:

This design can perform ***logging*** (logging can be turned on or off) with standard logging (only exceptions) or detailed logging.

In the version 2.0 the following changes were performed:

- Added “id”, “removed” and auditing properties to MemberImage entity. Note that MemberImage#memberId is not an ID of this JPA entity anymore.
- MemberImage entities are now removed logically (by setting removed property to “true”).
- Added getMemberPhotos() method to MemberPhotoManager and MemberPhotoDAO and their implementations. This method can be used for retrieving the last added/updated member images with optional pagination.

1.1 Design Patterns

- **Strategy pattern** – *Dependency Injection (or IoC - Inversion of Control)* inject the required configuration EntityManager objects (using setter based injection or constructors based injection) that are used strategically by JPAMemberPhotoDAO. Also MemberPhotoDAO is used strategically by the MemberPhotoManagerImpl.
- **Delegation pattern –** MemberPhotoManagerImpl delegates its operations to the configured MemberPhotoDAO.
- **DAO Pattern** – this component provides a MemberPhotoDAO interface and a JPA implementation, which follows this pattern.
- **DTO pattern** – all the entities from this design are Data Transfer Objects for the MemberPhotoDAO.

1.2 Industry Standards

- JPA, XML, JavaBeans, IoC, JPQL

1.3 Required Algorithms

There are no complicated algorithms in this design.

The section below is just a design consideration related to logging used in this design.

1.3.1 Logging

The Log field can be initialized using:

`Log log = LogManager.getLog( logName )` if a logName is provided

or:

`Log log = LogManager.getLog()` if no logName is provided

Logging is used in this way:

```
logger.log( Level.DEBUG, “Some internal details” );
```

Logging can be turned on or off using:

`setLog(newLog)`method. If newLog is null logging is disabled. To enable logging, a valid value should be provided.

There are two types of logging that can be set using: setVerboseLogging(verboseLogging:boolean) method:

1. ***Standard log*** that logs only the exceptions:
   if(getVerboseLogging()==**false**):
   - Any exception (Level.ERROR level). The exception name and stack trace should be logged.
2. ***Detailed log*** that logs exceptions and other details:

   if(getVerboseLogging()==**true**):
   - Entering the method and timestamp (Level.DEBUG level);
   - Method arguments (Level.DEBUG level);
   - Time spent in the method (Level.DEBUG level);
   - Return value (Level.DEBUG level).

   Any exception (Level.ERROR level). The exception name and stack trace should be logged.

1.3.2 DB schema

The following database schema can be used with this component:

```
~~CREATE SEQUENCE SQ_image;~~

CREATE TABLE image (
  image_id                SERIAL8,
  file_name               VARCHAR(50) NOT NULL
);

CREATE TABLE member_image (
  member_image_id         SERIAL8,
  member_id               INT8 NOT NULL,
  image_id                INT8 NOT NULL,
  removed                 BOOLEAN NOT NULL,
  created_by              VARCHAR(50) NOT NULL,
  created_date            DATETIME YEAR TO FRACTION NOT NULL,
  updated_by              VARCHAR(50) NOT NULL,
  updated_date            DATETIME YEAR TO FRACTION NOT NULL
);

ALTER TABLE image
```

```
        ADD CONSTRAINT ( PRIMARY KEY (image_id) CONSTRAINT PK_image )
;
ALTER TABLE member_image
        ADD CONSTRAINT ( PRIMARY KEY (member_image_id) CONSTRAINT PK_member_image )
;

ALTER TABLE member_image
        ADD CONSTRAINT ( FOREIGN KEY(image_id)
        REFERENCES image(image_id) CONSTRAINT FK_member_image_1 )
;
```

1.3.3 ORM mapping file

The following ORM mapping file can be used with this component:

```
<?xml version="1.0" encoding="UTF-8" ?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
    http://java.sun.com/xml/ns/persistence/orm_1_0.xsd" version="1.0">

    <description>
        OR Mapping file for the TopCoder Java Generic Member Photo Manager 2.0
    </description>

~~    <sequence-generator name="SEQ" />~~

    <entity class="com.topcoder.web.memberphoto.entities.Image">
        <table name="image" />
~~        <sequence-generator name="SEQ" sequence-name="SQ_image" initial-value="1" allocation-
size="1" />~~
        <attributes>
            <id name="id">
                <column name="image_id" nullable="false" />
                <generated-value />
            </id>
            <basic name="fileName">
                <column name="file_name" nullable="false" />
            </basic>
        </attributes>
    </entity>

    <entity class="com.topcoder.web.memberphoto.entities.MemberImage">
        <table name="member_image" />
        <attributes>
            <id name="id">
                <column name="member_image_id" nullable="false" />
                <generated-value />
            </id>
            <basic name="memberId">
                <column name="member_id" nullable="false" />
            </basic>
            <basic name="removed">
                <column name="removed" nullable="false" />
            </basic>
            <basic name="createdBy">
                <column name="created_by" nullable="false" />
            </basic>
            <basic name="createdDate">
                <column name="created_date" nullable="false" />
            </basic>
            <basic name="updatedBy">
                <column name="updated_by" nullable="false" />
            </basic>
            <basic name="updatedDate">
                <column name="updated_date" nullable="false" />
            </basic>
            <one-to-one name="image" target-entity="com.topcoder.web.memberphoto.entities.Image">
                <join-column name="image_id" referenced-column-name="image_id" nullable="false" />
                <cascade>
                    <cascade-merge />
                </cascade>
            </one-to-one>
        </attributes>
    </entity>

</entity-mappings>
```

1.4 Component Class Overview

1.4.1 com.topcoder.web.memberphoto.entities

Image <<class, @Entity>>
This class represents the Image java bean. An Image can contain an image identifier and a file name.
This is a simple java bean (with a default no-arg constructor and for each property, a corresponding getter/setter method).
Any attribute in this bean is OPTIONAL so NO VALIDATION IS PERFORMED here.
This class is Serializable (implements Serializable).
The IoC should handle the validity if the passed values in setters.
Annotations:
@Entity - this annotation should be used to mark this java bean as an entity and to ease its use with JPA.
Thread safety:
This class contains only mutable fields so therefore it is not thread safe.

MemberImage <<class, @Entity>>
This class represents the MemberImage java bean. An MemberImage can contain a member identifier and an image.
This is a simple java bean (with a default no-arg constructor and for each property, a corresponding getter/setter method).
Any attribute in this bean is OPTIONAL so NO VALIDATION IS PERFORMED here.
This class is Serializable (implements Serializable).
The IoC should handle the validity if the passed values in setters.
Annotations:
@Entity - this annotation should be used to mark this java bean as an entity and to ease its use with JPA.
Thread safety:
This class contains only mutable fields so therefore it is not thread safe.
Changes in 2.0:
Added id, removed and auditing properties.

1.4.2 com.topcoder.web.memberphoto.manager

MemberPhotoManager <<interface>>
This interface represents the MemberPhotoManager interface.
This interface defines the specific methods available for the MemberPhotoManager interface: retrieve member photo, save member photo and remove member photo.
Thread safety:
Implementations of this interface should be thread safe.
Changes in 2.0:
Added byUser parameter to saveMemberPhoto() and removeMemberPhoto() methods (to be used for auditing).
Added getMemberPhotos() method for retrieving recently added/updated member images.

MemberPhotoDAO <<interface>>
This interface represents the MemberPhotoDAO layer interface.
This is a pluggable strategy. This is a pluggable strategy supporting any kind of the concrete implementation (like different databases, JPA, XML based persistence and so on).
This interface defines the specific methods available for the MemberPhotoDAO interface: retrieve member photo, save member photo and remove member photo.
Thread safety:
Implementations of this interface should be thread safe.
Changes in 2.0:

Added byUser parameter to saveMemberPhoto() and removeMemberPhoto() methods (to be used for auditing).
Added getMemberPhotos() method for retrieving recently added/updated member images.

MemberPhotoManagerImpl <<class>>
This class is a default implementation of MemberPhotoManager interface and implements the following operations: retrieve member photo, save member photo and remove member photo.
Its fields can be initialized or set using setters or using various constructors.
This design extensively reference operations from MemberPhotoDAO (delegates to its provided implementation).
Logging Wrapper 2.0 is used to perform the needed logging operations (see 1.3.1. section from CS for more details).
Thread safety:
This class can be considered as thread safe because its fields are injected once by Spring Framework and it ensure that will not be changed afterwards and after the injection the Spring Framework will provide thread safety.
Used in a multithread environment, the received arguments could be changed by external threads but we can assume this will not happen and that the component will be used from the Spring Framework will in a thread safe manner. Also MemberPhotoDAO implementations are required to be thread safe so thread safety should not be a concern here.
Changes in 2.0:
Added byUser parameter to saveMemberPhoto() and removeMemberPhoto() methods (to be used for auditing).
Added getMemberPhotos() method for retrieving recently added/updated member images.

1.4.3 com.topcoder.web.memberphoto.manager.persistence

JPAMemberPhotoDAO <<class>>
This class is a JPA realization of the MemberPhotoDAO interface.
This class implements the methods available for the MemberPhotoDAO interface: retrieve member photo, save member photo and remove member photo.
JPA Hibernate is used to perform the needed operations with the persistence.
Spring IoC (inversion of control) container is used to inject dependencies into this object.
The clients of this component most likely will handle transactions using declarative transaction handling provided by Spring framework. So this component does not deal with transaction, i.e. methods of manager class assume that they are called inside of transaction.
Thread safety:
This class can be considered as thread safe because its fields are injected once by Spring Framework and it ensure that will not be changed afterwards and after the injection the Spring Framework will provide thread safety. Also thread safety depend on the configured EntityManager.
Used in a multithread environment, the received arguments could be changed by external threads but we can assume this will not happen and that the component will be used from the Spring Framework will in a thread safe manner.
Under such conditions this class becomes thread-safe.
Changes in 2.0:
Added byUser parameter to saveMemberPhoto() and removeMemberPhoto() methods (to be used for auditing).
Added getMemberPhotos() method for retrieving recently added/updated member images.
Updated getMemberPhoto() and removeMemberPhoto() methods to support logical removal of MemberImage entities.
Updated saveMemberPhoto() and removeMemberPhoto() methods to support MemberImage auditing fields.

1.5 Component Exception Definitions

MemberPhotoManagementException <<exception>>
This exception is the base exception for all exceptions raised from operations performed in this design.
This exception wraps exceptions raised from persistence, from usage of the JPA Hibernate utilities or used TopCoder components.

Thread safety:
This exception is not thread safe because parent exception is not thread safe.
The application should handle this exception in a thread-safe manner.

MemberPhotoNotFoundException <<exception>>
This exception signals an issue if the member photo does not exists for the given member id.
Has a member id argument in each constructor and a getter for this property.
Thread safety:
This exception is not thread safe because parent exception is not thread safe.
The application should handle this exception in a thread-safe manner.

MemberPhotoDAOException <<exception>>
This exception is the exception raised from operations performed over the persistence.
This exception wraps exceptions raised from persistence and from usage of the JPA Hibernate utilities.
Thread safety:
This exception is not thread safe because parent exception is not thread safe.
The application should handle this exception in a thread-safe manner.

1.6 Thread Safety

This component is technically thread-safe due to the following reason:

- Even the java beans are not thread safe they will be used in a thread safe manner by the Spring Framework.
- The ***classes*** of this design are thread safe:
  - ➢ They contain mutable fields but the Spring Framework ensures that they will be initialized only once using the injection;
  - ➢ After the injection, the Spring Framework ensures that it will provide the necessary thread safety for the performed operations.

This component does not deal with transaction, i.e. methods of manager class assume that they are called inside of transaction.

Thread safety of the component was not changed in the version 2.0.

2. Environment Requirements

2.1 Environment

- Java 1.5 is required for development
- Java 1.5 is required for compilation
- Solaris 7
- RedHat Linux 9
- Windows 2000
- Windows 2003
- Spring Framework
- Hibernate 3.6 as JPA implementation: implementation: http://www.hibernate.org
- JBoss 4.0.2
- Spring 3.0.5 – http://www.springsource.org/node/2880
- Informix 11.0

2.2 TopCoder Software Components

- **Logging Wrapper 2.0** – provides a standard logging API with a pluggable back-end logging implementation.
- **Base Exception 2.0** – is used as base exception for custom exceptions created in this design. TopCoder standard for all custom exceptions.

2.3 Third Party Components

- **None**

3. Installation and Configuration

3.1 Package Name

com.topcoder.web.memberphoto.entities – represents package where the Image and MemberImage entities created in this design are placed.

com.topcoder.web.memberphoto.manager – represents the package where the member photo manager interface and its default implementation are placed. Also member photo DAO interface and PagedResult entity are placed here.

com.topcoder.web.memberphoto.manager.persistence – represents the package where the member photo DAO JPA implementation is placed.

3.2 Configuration Parameters

- None. All needed configurations are performed using constructors or using setters. IoC is responsible for setting the needed configurations.

3.3 Dependencies Configuration

- None.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Setup Informix database:

  a. Run test_files/DBSetup.sql.

  test_files/DropDB.sql can be used to drop tables.

  b. Update test_files/META-INF/persistence.xml if needed.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

- Follow the demo section.

4.3 Demo

The following demo shows the usage of the Member Photo Manager component:
First let's assume the following images and member images can be found in the database:

Images

instance	id	fileName
image1	1	Handle1.jpg
image2	2	Handle2.jpg
image3	3	Handle3.jpg

MemberImages (creation auditing fields are omitted for brevity)

instance	memberid	image instance	removed	updatedBy	updatedDate
memberImage1	222222333	image1	1	admin	2011-14-01 12:00:00
memberImage2	222333444	image2	0	admin	2011-14-01 12:00:00
memberImage3	222444555	image3	0	admin	2011-14-01 12:00:00

```
import com.topcoder.web.memberphoto.entities.Image;
import com.topcoder.web.memberphoto.entities.MemberImage;
import com.topcoder.web.memberphoto.manager.MemberPhotoManager;
import com.topcoder.web.memberphoto.manager.MemberPhotoManagerImpl;
import com.topcoder.web.memberphoto.manager.MemberPhotoDAO;
import com.topcoder.web.memberphoto.manager.persistence.JPAMemberPhotoDAO;
import com.topcoder.util.Log;
import com.topcoder.util.LogManager;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;


public class MemberPhotoManagerDemo {

        public static void main(String[] args) {

         // configure the EntityManager:
         EntityManagerFactory emf = Persistence.createEntityManagerFactory("memberPhotoManager");
         EntityManager entityManager = emf.createEntityManager();

         // create a MemberPhotoDAO using the obtained EntityManager:
         MemberPhotoDAO memberPhotoDAO = new JPAMemberPhotoDAO(entityManager);

         // create a MemberPhotoManager using the default constructor:
         MemberPhotoManager memberPhotoManager = new MemberPhotoManagerImpl(memberPhotoDAO);

         // create a MemberPhotoManager using the logName="MemberPhoto":
         String logName = "MemberPhoto";
         memberPhotoManager = new MemberPhotoManagerImpl(logName);

         // create a MemberPhotoManager using the obtained MemberPhotoDAO:
         memberPhotoManager = new MemberPhotoManagerImpl(memberPhotoDAO);

         // create a MemberPhotoManager using the obtained MemberPhotoDAO
         // and the logName="MemberPhoto":
         memberPhotoManager = new MemberPhotoManagerImpl(logName, memberPhotoDAO);

         // we will use the member images and images mentioned above:

         // 1. get image for memberImage3 using it's id:
          Image image = memberPhotoManager.getMemberPhoto(222444555);

        //the obtained image should be image 3 and should look like:
        /*
```

instance	id	fileName
image3	3	Handle3.jpg

```
        */

          // 1.a. get image for member with ID = 222222333:
         image = memberPhotoManager.getMemberPhoto(222222333);
         // image must be null since the corresponding member image is removed

         // 2. save member photo for member with ID=222555666 using image
         image = new Image();
         image.setFileName("NewHandle4.jpg");
         entityManager.getTransaction().begin();
         memberPhotoManager.saveMemberPhoto(222555666, image, "testUser");
          entityManager.getTransaction().commit();

        // the following Image instance must be persisted:
        /*
```

instance	id	filename
image4	4	NewHandle4.jpg

```
        */
        // additionally the following MemberImage instance must be persisted
        // (creation auditing fields are omitted for brevity):
        /*
```

instance	memberid	image instance	removed	updatedBy	updatedDate
memberImage4	222555666	image4	0	testUser	2011-14-01 14:21:48

```
        */
```

```
 // 3. save member photo for member with ID=222555777 using file name
entityManager.getTransaction().begin();
memberPhotoManager.saveMemberPhoto(222555777, "PicForHandle5.jpg", "testUser");
 entityManager.getTransaction().commit();

// the following Image instance must be persisted:
/*
```

instance	id	fileName
image5	5	PicForHandle5.jpg

```
*/
// additionally the following MemberImage instance must be persisted
// (creation auditing fields are omitted for brevity):
/*
```

instance	memberid	image instance	removed	updatedBy	updatedDate
memberImage5	222555777	image5	0	testUser	2011-14-01 14:21:49

```
*/

 // 4. remove member photo for member with ID=222333444
entityManager.getTransaction().begin();
memberPhotoManager.removeMemberPhoto(222333444, "testUser");
 entityManager.getTransaction().commit();

// after this call only removed property of MemberImage instance
// must be changed in the persistence (together with auditing properties):
/*
```

instance	memberid	image instance	removed	updatedBy	updatedDate
memberImage2	222333444	image2	1	testUser	2011-14-01 14:21:50

```
*/

 // 5. retrieve the first page of member photos, with 2 photos on the page:
PagedResult<MemberImage> pagedResult = memberPhotoManager.getMemberPhotos(1, 2);
// pagedResult.getTotalRecords() must be 3
// pagedResult.getRecords().size() must be 2
// pagedResult.getRecords().get(0).getMemberId() must be 222555777
// pagedResult.getRecords().get(0).isRemoved() must be false
// pagedResult.getRecords().get(0).getUpdatedBy() must be "testUser"
// pagedResult.getRecords().get(0).getImage().getId() must be 5
// pagedResult.getRecords().get(0).getImage().getFileName() must be "PicForHandle5.jpg"
// pagedResult.getRecords().get(1).getMemberId() must be 222555666
// pagedResult.getRecords().get(1).isRemoved() must be false
// pagedResult.getRecords().get(1).getUpdatedBy() must be "testUser"
// pagedResult.getRecords().get(1).getImage().getId() must be 4
// pagedResult.getRecords().get(1).getImage().getFileName() must be "NewHandle4.jpg"

// we can enable detailed (using true) or standard (using false)
// logging:
((MemberPhotoManagerImpl) memberPhotoManager).setVerboseLogging(true);
// now detailed logging is performed (if logging is enabled)

// we can turn on (using a valid value) or off (null) the logging:
((MemberPhotoManagerImpl) memberPhotoManager).setLog(null);
// now logging is not performed any more

// we can turn on the logging using a valid value in this way:
Log log = LogManager.getLog("newMemberPhotoLog");
((MemberPhotoManagerImpl) memberPhotoManager).setLog(log);
 // now logging is performed (enabled)

    }
}
```

5. Future Enhancements

- More generic image management solution (for example more types of uploaded images than just member photos).