Member Photo Management Servlets 2.0 Component Specification

1. Design

This component will provide the servlets which are used for member photo management: one servlet is used to upload member photo and other to remove member photos. This component will use the Member Photo Manager component as a backend.

MemberPhotoUploadServlet and MemberPhotoRemovalServlet classes provide the functionalities to upload the member photo and remove member photo.

Please note that this is a version 2.0, which is built on top of the previous version. Version 2.0 Changes will be color-coded as follows:

- All new additions are in red
- Changes to existing functionality is in blue
- No change is simply shown as black.

1.1 Design Patterns

- **Strategy Pattern**: The MemberInformationRetriever interface and its implementations are used as a strategy in MemberPhotoRemovalServlet.
- **MVC Pattern** is used since JSP pages will be used as Views, Servlets as the Controllers and the data being manipulated (i.e. image data) as Model

1.2 Industry Standards

Http Servlet, JSP

1.3 Required Algorithms

Here we will discuss the aspects of the JSP pages that the developers will have to provide when developing/testing the servlets.

Upload_Image.jsp:
User can upload a photo by specifying the following information:

- member id
- photo image file
- action value indicating the image should be previewed, cropped or committed.

Preview_Image.jsp
This is the result of the upload when the action is “preview” and this page will show the user the image that they have just uploaded in its original size. Note that this page will be JCrop enabled in the sense that the user will be able to crop the given image. The user will then be able to have this image cropped by the servlet. Note that the cropping action will have to send the following information to the servlet from JCrop:

- X, Y, X1, Y1 of the cropped rectangle

Commit_Cropped_Image.jsp
This is the page that allows the user to preview the cropped image. The user will also be able to submit this image to be saved.

Remove_Image.jsp
User will be able to remove a given image from the server using this page. It will allow member and web site administrator to remove particular member photo.
This servlet will handle the submission of the form with such fields:

- member id
- removal reason explanation

Member_Photo_List.jsp
This page will allow the user to request a paged list if member photos. It will allow web site administrator to view members' photos and will enable the request containing the following parameters:

- pageNo – the page number, which starts 1. Optional, default to 1
- pageSize – the page size, which should be positive. Optional, default to 10.

Member_Photo_List_view.jsp
This will be the jsp page, which take the data that the servlet returns and which will then render the paged photos.

Error.jsp
This is a general error page, which all the JSP pages will delegate to if there was an error (or if the servlet has reported errors)

1.4 Component Class Overview

- **MemberPhotoUploadServlet**: This class extends the HttpServlet, and it allows web site members to upload their photos.

  --- Version 2.0 changes
  1. Added the ability to process and validate GIF file formats
  2. Added the max byte size check for the image being uploaded when validating (RS 1.2.1.1)
  3. Added the configurable error code for the validation error for exceeded image file size (RS 1.2.1.1)
  4. The submitted flag parameter is not used anymore. Instead each action has its own string name.
- **MemberPhotoRemovalServlet**: This class extends the HttpServlet, and it allows web site administrators to remove particular member photos.
- **MemberInformationRetriever**: This interface defines the contract to retrieve the member information.
- **MemberInformation**: This data entity class contains the member information.
- **MemberPhotoListServlet**: This class extends the HttpServlet, and it allows web site administrators to list/view particular member photos. The listing is paged based on the input parameters or specific defaults are assumed (see method doc)
- The MemberPhotoManager is used to get the photos information from persistence, and get the image file in order to allow the user to view the file(s)
- It will validated the submitted form data, if validation fails, it will forward to a configured validation error url, and if everything goes well, it will forward to a configured success url.

1.5 Component Exception Definitions

- **MemberPhotoUploadException**: This exception extends the ServletException, and it's thrown from MemberPhotoUploadServlet if any unexpected error occurs, it's used to wrap the underlying exceptions.
- **MemberPhotoRemovalException**: This exception extends the ServletException, and it's thrown from MemberPhotoRemovalServlet if any unexpected error occurs, it's used to wrap the underlying exceptions.

- **MemberInformationRetrievalException**: This exception is thrown from MemberInformationRetriever interface and its implementations if they fail to retrieve the member information.
- **MemberPhotoListingException**: This exception extends the ServletException, and it's thrown from MemberPhotoListServlet if any unexpected error occurs, it's used to wrap the underlying exceptions.

NOTE: A string is empty if its length is 0 after being trimmed.

1.6 Thread Safety

The MemberPhotoUploadServlet and MemberPhotoRemovalServlet will be used to serve multiple requests from different threads, so they have to be thread-safe during the service time. Their instance variables will be injected by Spring Framework, and are immutable afterwards. The injection is done before they are used to serve user requests, so they are thread-safe during the service time.

The implementations of MemberInformationRetriever interface are required to be thread-safe. The MemberInformation class is mutable, but its instances will never be shared across multiple threads in this design, so it won't affect the thread-safety of this design.

All dependent TCS components are either thread-safe themselves or used in thread-safe way in this design.

There are no changes to thread safety considerations in version 2.0

2. Environment Requirements

2.1 Environment

- Java 1.5

2.2 TopCoder Software Components

- **Base Exception 2.0**: Used as the base exception for the custom exceptions in this design.
- **Member Photo Manager 1.0**: Used to manage member photo in persistence.
- **Email Engine 3.0**: Used to send notification email.
- **Document Generator 2.1**: Used to generate email body from template in this design.
- **File Upload 2.2**: Used to extract uploaded file from servlet request.
- **Logging Wrapper 2.0**: This is used for logging errors and actions in the servlet implementations.

2.3 Third Party Components

- **Spring Framework 2.0** - Used to inject properties to servlets, and it's also used to provide transactional support for this design.
- **JCrop 0.9.8+ -** On the preview page, user will be able to draw a frame to crop the image with certain aspect ratio using JCrop. http://deepliquid.com/content/Jcrop.html

3. Installation and Configuration

3.1 Package Name

com.topcoder.web.memberphoto.servlet

3.2 Configuration Parameters

The email body template file used in the MemberPhotoRemovalServlet can contain the following variables:

Variable Name	Variable Description
%memberName%	Represents the member name.
%photoRemovalReason%	Represents the reason to remove the photo.
%memberId%	Represents the member id.
%memberHandle%	Represents the member handle.

3.3 Dependencies Configuration

None.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

See demo.

4.3 Demo(updated)

The MemberPhotoUploadServlet and MemberPhotoRemovalServlet classes should be configured as beans using the Spring Framework in order to inject the properties' values into them. So actually they cannot be deployed as http servlet directly in the servlet container, we ought to create a separate dispatch http servlet to read the beans configuration for these two classes, and then dispatch the requests to them according to the requested URLs.

4.3.1 Upload Photo

Assume the request URL configured for MemberPhotoUploadServlet is: /uploadPhoto.

The following properties are configured for the MemberPhotoUploadServlet class.

The configured validationErrorForwardUrl is /uploadPhotoValidationError, the configured previewForwardUrl is /previewPhoto, and the configured submittedRedirectUrl is /userProfile?memberId=${memberId}, the configured submittedActionParameterName is action, The configured photoImagePreviewDirectory is c:\preview, and the configured photoImageSubmittedDirectory is c:\submitted. The configured JCrop data for width and height would be: width and height for the input parameter names.

4.3.1.1 Request 1

User submits a form, which will issue a POST request with memberId (= 1), photo image file, and action (= "preview") data to the /uploadPhoto on the server. The submitted photo image file will be resized and cropped properly, and then a file named 1.jpeg is stored in c:\submitted directory on the server side. The member photo information will also be stored in persistence. Then user will be redirected to the /userProfile?memberId=1 on the server. Note that we assume that information from JCrop would be submitted with the witdh and height of the box to be cropped as well as the box cooredinates.

Note the memberId should also be in the session under the configured key.

```
MemberPhotoUploadServlet bean =
TestHelper.genUploadServletBean("demo1");

MockHttpServletRequest request = TestHelper.prepareRequest();
MockHttpServletResponse response = new MockHttpServletResponse();

request.setParameter("action", "preview");
request.setParameter("member_id", "1");

MockHttpSession session = new MockHttpSession();
session.setAttribute("member_id_session_key", new Long(1));
request.setSession(session);

bean.doPost(request, response);
assertTrue("File should exist.", new File(TEST_FILES +
"submitted/1.jpeg").exists());
```

4.3.1.2 Request 2

User submits a form, which will issue a POST request with memberId (= 1), text file, and action (= “preview”) data to the /uploadPhoto on the server. Validation error will occurs, as user is required to submit a PNG, GIF, or JPEG image file. Then user will be forwarded to /uploadPhotoValidationError.

Note the memberId should also be in the session under the configured key.

```
MemberPhotoUploadServlet bean =
TestHelper.genUploadServletBean("demo2");

MockHttpServletRequest request = TestHelper.prepareRequest();
MockHttpServletResponse response = new MockHttpServletResponse();

request.setParameter("action", "preview");
request.setParameter("member_id", "1");

MockHttpSession session = new MockHttpSession();
session.setAttribute("member_id_session_key", new Long(1));
request.setSession(session);

bean.doPost(request, response);
assertEquals("error message should be the same.", "The image
format is not supported.",
request.getAttribute("photo_image_error_name").toString());
```

4.3.1.3 Request 3

User submits a form, which will issue a POST request with memberId (= 1), photo image file data to the /uploadPhoto on the server. The submitted photo image file will be resized and cropped properly, and then a file named 1.jpeg is stored in c:\preview directory on the server side. Then user will be forward to the /previewPhoto on the server.

Note the memberId should also be in the session under the configured key.

```
MemberPhotoUploadServlet bean =
TestHelper.genUploadServletBean("demo3");

MockHttpServletRequest request = TestHelper.prepareRequest();
MockHttpServletResponse response = new MockHttpServletResponse();

request.setParameter("action", "crop");
```

```
request.setParameter("member_id", "1");

MockHttpSession session = new MockHttpSession();
session.setAttribute("member_id_session_key", new Long(1));
request.setSession(session);

bean.doPost(request, response);
assertTrue("File should exist.", new File(TEST_FILES +
"preview/1.jpeg").exists());
```

4.3.2 *Remove Photo*

Assume the request URL configured for MemberPhotoRemovalServlet is: /removePhoto.

The following properties are configured for the MemberPhotoRemovalServlet class.

The configured validationErrorForwardUrl is /removePhotoValidationError, and the configured successResultForwardUrl is /removePhotoSuccess.
The configured photoImageDirectory is c:\submitted.
And the configured emailBodyTemplateFileName contains the following content:

```
Hello %memberName%,

You photo is removed because: %photoRemovalReason%.
Your id is: %memberId%.
Your handle is: %memberHandle%.

Regards.
```

4.3.2.1 Request 1

User submits a form, which will issue a POST request with memberId (= 1), removal reason (= “for security reason”) to the /removePhoto on the server. The image file 1.jpeg will be removed from the c:\submitted directory on the server, and its related information will also be removed from the persistence. Then a notification email will be sent to the member to notify him/her that his/her photo is removed. Then user will be forwarded to the /removePhotoSuccess.

Assume the memberName and memberHandle for member (with memberId = 1) are “SuperMan” and “SimpleMan”. Then the received email body will be:

```
Hello SuperMan,

You photo is removed because: for security reason.
Your id is: 1.
Your handle is: SimpleMan.

Regards.
```

Note the user who submits the form should have administrator role.

```
MemberPhotoRemovalServlet bean =
TestHelper.genRemovalServletBean("demo4");

MockHttpServletRequest request = new MockHttpServletRequest();
MockHttpServletResponse response = new MockHttpServletResponse();

request.setParameter("member_id", "1");
request.setParameter("removal_reason", "this is removal reason.");

MockHttpSession session = new MockHttpSession();
```

```
session.setAttribute("is_admin_session_key", true);
request.setSession(session);
bean.doPost(request, response);
```

4.3.2.2 Request 2

User submits a form, which will issue a POST request with memberId (= 1), removal reason to the /removePhoto on the server. If the user is not an administrator, validation error occurs, and user will be forwarded to /removePhotoValidationError on the server.

```
MemberPhotoRemovalServlet bean =
TestHelper.genRemovalServletBean("demo5");

MockHttpServletRequest request = new MockHttpServletRequest();
MockHttpServletResponse response = new MockHttpServletResponse();

request.setParameter("member_id", "1");
request.setParameter("removal_reason", "this is removal reason.");

MockHttpSession session = new MockHttpSession();
session.setAttribute("is_admin_session_key", false);
  request.setSession(session);

  bean.doPost(request, response);
  assertEquals("error message should be the same.", "The user is
   not an administrator.",
  request.getAttribute("not_admin_error_name").toString());
```

4.3.3 List user photos

Assume the request URL configured for MemberPhotoRemovalServlet is: /listPhotos. The following properties are configured for the MemberPhotoListServlet class.

The configured validationErrorForwardUrl is /ListPhotosValidationError, and the configured successResultForwardUrl is /listPhotosSuccess.
The configured photoListName is “member_photos_list”. The configured pageNumberParameterName is “pageNumber” and finally the configured pageSizeParameterName is “pageSize”

4.3.3.1 Request 1

User submits a form, which will issue a POST request with memberId (= 1), page number (= “2”) and page size (=”5”) to the /listPhotos on the server. All the image files for this user that fit the paging requested (i.e. 5 images starting with the 6th image in a sequence) will be provided in the request information (as simply links) Then user will be forwarded to the /listPhotosSuccess.

Note the user who submits the form should have administrator role.

```
MemberPhotoListServlet bean =
TestHelper.genRemovalServletBean("demo5");

MockHttpServletRequest request = new MockHttpServletRequest();
MockHttpServletResponse response = new MockHttpServletResponse();

request.setParameter("member_id", "1");
request.setParameter("pageNumber", "2");
request.setParameter("pageSize", "5");
```

```
MockHttpSession session = new MockHttpSession();
session.setAttribute("is_admin_session_key", true);
request.setSession(session);
bean.doPost(request, response);
```

5 Future Enhancements

MemberInformationRetriever implementations can be added.