Tapestry5HowToStreamAnExistingBinaryFile

This example is very similar to the examples on Streaming a Dynamic PDF or a Dynamic chart. Except in this case the file already exists somewhere. I'll also show some utility classes I made up to assist in the process.

ImagePage.java

This is the page class in your Tapestry 5 application

```
import java.io.InputStream;

import com.mycompany.myapp.wui.tapestry.services.util.JPEGAttachment;

import org.apache.tapestry.StreamResponse;
 public class ImagePage {
        public StreamResponse onSubmit() {
                //Note that you could provide an absolute path here, like H:\\LOLCATZ.MP3
                InputStream input = ImagePage.class.getResourceAsStream("HelloKitty.jpg");
                return new JPEGAttachment(input);
        }
}
```

JPEGAttachment.java

```
import java.io.InputStream;

public class JPEGAttachment extends AttachmentStreamResponse {
        public JPEGAttachment(InputStream is, String args) {
                super(is, args);
                this.contentType = "image/jpeg";
                this.extension = "jpg";
        }

        public JPEGAttachment(InputStream is) {
                super(is);
                this.contentType = "image/jpeg";
                this.extension = "jpg";
        }
}
```

AttachmentStreamResponse.java

This is a util class for Attachments. The goal here is to have a "Save As" dialog pop up on the client browser.

```
import java.io.IOException;
import java.io.InputStream;

import org.apache.tapestry.StreamResponse;
import org.apache.tapestry.services.Response;

public class AttachmentStreamResponse implements StreamResponse {
        private InputStream is = null;

        /**
         * This can be changed to something obscure, so that it is more likely to trigger a "Save As" dialog,
although there
         * is no guarantee.
         *
         * ex: application/x-download
         *
         * See http://www.onjava.com/pub/a/onjava/excerpt/jebp_3/index3.html
         */
        protected String contentType = "text/plain";

        protected String extension = "txt";

        protected String filename = "default";

        public AttachmentStreamResponse(InputStream is, String filenameIn) {
                this.is = is;
                if (filenameIn != null) {
                        this.filename = filenameIn;
                }
        }

        public AttachmentStreamResponse(InputStream is) {
                this.is = is;
        }

        public String getContentType() {
                return contentType;
        }

        public InputStream getStream() throws IOException {
                return is;
        }

        public void prepareResponse(Response arg0) {
                arg0.setHeader("Content-Disposition", "attachment; filename=" + filename + ((extension == null)
? "" : ("." + extension)));
                arg0.setHeader("Expires", "0");
                arg0.setHeader("Cache-Control", "must-revalidate, post-check=0, pre-check=0");
                arg0.setHeader("Pragma", "public");
                //We can't set the length here because we only have an Input Stream at this point. (Although
we'd like to.)
                //We can only get size from an output stream.  arg0.setContentLength(.length);
        }
}
```

InlineStreamResponse.java

This file is not used in the example, but is for files you intend to display inline in the browser (No "Save As" dialog box):

```
import java.io.IOException;
import java.io.InputStream;

import org.apache.tapestry.StreamResponse;
import org.apache.tapestry.services.Response;

public class InlineStreamResponse implements StreamResponse {
        private InputStream is = null;

        protected String contentType = "text/plain";// this is the default

        protected String extension = "txt";

        protected String filename = "default";

        public InlineStreamResponse(InputStream is, String... args) {
                this.is = is;
                if (args != null) {
                        this.filename = args[0];
                }
        }

        public String getContentType() {
                return contentType;
        }

        public InputStream getStream() throws IOException {
                return is;
        }

        public void prepareResponse(Response arg0) {
                arg0.setHeader("Content-Disposition", "inline; filename=" + filename
                                + ((extension == null) ? "" : ("." + extension)));
        }
}
```

Other file types

Now, all you have to do for other file types is create a class like this:

For a JPG inline

```
import java.io.InputStream;

public class JPEGInline extends InlineStreamResponse {
        public JPEGInline(InputStream is, String... args) {
                super(is, args);
                this.contentType = "image/jpeg";
                this.extension = "jpg";
        }
}
```

For an XLS attachment

```
public class XLSAttachment extends AttachmentStreamResponse {
        public XLSAttachment(InputStream is, String args) {
                super(is, args);
                this.contentType = "application/vnd.ms-excel";
                this.extension = "xls";
        }
}
```

For a PDF attachement

```
import java.io.InputStream;

public class PDFAttachment extends AttachmentStreamResponse {
        public PDFAttachment(InputStream is, String args) {
                super(is, args);
                this.contentType = "application/pdf";
                this.extension = "pdf";
        }

        public PDFAttachment(InputStream is) {
                super(is);
                this.contentType = "application/pdf";
                this.extension = "pdf";
        }
}
```

Calling the event handler from a page template

The event handler can be called using an actionLink e.g. <t:actionLink id="streamPDF">download pdf</t:actionlink> but the event handler can can also be called from an expansion, for example if you want to embed a dynamically generated/retrievd image in a page - ${embedImage()}. When the page renders the event handler will be called

Creating a page for streaming files

If you rather prefer a dedicated page for streaming content (say for example *UploadStore*), you can also use the *onActivate()* method to return a *StreamRes ponse*. Here is an example:

```
public class UploadStore {
        @Inject
        private LinkSource linkSource;
        public String getUploadedFile(String uuid) {
                return linkSource.createPageRenderLink(UploadStore.class.getSimpleName(), false, new Object[]
{uuid});
        }

        public StreamResponse onActivate(final String uuid) {
                return new StreamResponse() { ... }
        }
}
public class SomePage {
        @InjectPage
        private UploadStore uploadStore;

        public String getUploadedImage() {
                return uploadStore.getUploadedFile("SOMEUUID");
        }
}
```

A simple **** in you *SomePage.tml* template should then do the job.