SendingDiferentContentTypes

Sometimes one needs to send files which cannot be created by a Tapestry template or stored as Assets in the webapp's path, such as PDF reports, imaages generated on the fly, etc.

The basic idea on how to do this is to get a hold of the servlet's output stream and pipe the file's bytes into it. This cannot be done by an IPage implementation, since by the time it is called, the ouput stream has already been configured for use by the tapestry engine for sending text into it.

So the way to do this, is to implement a IServiceEngine. A service is called early enough into the request processing that we can configure ourselves the OutputStream for our purposes.

The following code was taken from a response by Jií Mareš to Ender Shorty on the Tapestry mailing list, edited to complete a few details.

```
package some.package;

public class SendFile implements IEngineService
{
        protected WebResponse response;
        protected LinkFactory linkFactory;

        public void setResponse(WebResponse response) {
                this.response = response;
        }

        public void setLinkFactory(LinkFactory linkFactory) {
                this.linkFactory = linkFactory;
        }

        public String getName() { return "SendFile";        }

        public ILink getLink(boolean post, Object parameter) {
            // You might want to validate parameters
            // parameter is actually a Object[] with the parameters
            // given to the @ServiceLink component in the template
            // where the link to this service is given
            Map<String, Object> params = new HashMap<String, Object>();

            params.put(PARAM_NAME, ((Object[]) parameter)[0].toString());
            // Put as many parameters as you need to build/retrieve your file
            // They have to be encoded as strings. Use a different name for each,
            // of course

            // The 'true' at the end means the URL will have a jsessionid at the end, and
            // thus the service will have access to the session state. If your service
            // does not need it, you can set it to 'false'
            return linkFactory.constructLink(this, post, params, true);
        }

        public void service(IRequestCycle cycle) throws IOException {
                // Retrieve and parse your parameters
                // Long id = Long.parseLong(cycle.getParameter(PARAM_NAME));
                // Then use them to get a hold of your file's data metadata

                // You can set arbitrary headers, eg.
                // response.setHeader("Content-Disposition",
                //                "inline; attachment; filename=" + someFileName);
                try {
                  OutputStream os = response.getOutputStream(new ContentType(/* Your file's content-type */));
                  // Write your file's data to os
                } catch (IOException e) {
                  throw new ApplicationRuntimeException(e);
                }
        }
}
```

Once you have this, you need to tell Tapestry about your new Service implementation. Add the following to your META-INF/hivemodule.xml (or create it, if you didn't have one):

```
<!-- id is whatever you want -->
<service-point id="SendFileService" interface="org.apache.tapestry.engine.IEngineService">
   <invoke-factory>
     <construct class="some.package.SendFile">
       <set-object property="response" value="infrastructure:response"/>
       <set-object property="linkFactory" value="infrastructure:linkFactory"/>
       <!-- You can inject any other HiveMind service into your service
            or set other properties -->
     </construct>
   </invoke-factory>
</service-point>

<contribution configuration-id="tapestry.services.ApplicationServices">
   <!-- name is whatever you want. object must be "service:the name you used as id in service-point" -->
   <service name="send_file" object="service:SendFileService"/>
</contribution>
```

That's it!

To use it, you must give a link to the user pointing to the service. In some page in your application include

```
<a jwcid="@ServiceLink" service="send_file" parameters="ognl:{param1, param2, etc}">Download the file!</a>
```

where 'service' is the name defined in the contribution/service entry in the hivemodule.xml, and parameters are the ones your service needs, in regular ognl syntax. If there's only one you can omit the curly braces.