RequestParameterEncoding

Request parameter encoding

How to set everything to UTF-8 with Cocoon and CForms (with Ajax and Dojo)

The best for internationalization is to handle everything in UTF-8, since this is probably the most intelligent encoding available out there. Everything means server side (Backend, XML), HTTP Requests/Responses and client side with forms and dojo.io.bind.

1. Sending all pages in UTF-8

You need to configure Cocoon's serializers to UTF-8. The XML serializer (`<serialize type="xml" />`) and the HTML serializer (`<serialize type="html" />`) need to be configured. To support all browsers, you must state the encoding to be used for the body and also include a meta tag in the html: `<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">`. This is very important, since the browser will then send form requests encoded in UTF-8 (and browsers normaly don't mention the encoding in the request, so you have to assume they are doing it right). Here is the configuration for the serializer components for your sitemaps that will do that:

```
<serializer name="xml" mime-type="text/xml"
  src="org.apache.cocoon.serialization.XMLSerializer">
  <encoding>UTF-8</encoding>
</serializer>

<serializer name="html" mime-type="text/html; charset=UTF-8"
  src="org.apache.cocoon.serialization.HTMLSerializer">
  <encoding>UTF-8</encoding>

  <!-- the following common doctype is only included for completeness, it has no impact on encoding -->
  <doctype-public>-//W3C//DTD HTML 4.01 Transitional//EN</doctype-public>
  <doctype-system>http://www.w3.org/TR/html4/loose.dtd</doctype-system>
</serializer>
```

2. AJAX Requests with CForms/Dojo

If you use CForms with ajax enabled, Cocoon will make use of dojo.io.bind() under the hood, which creates XMLHttpRequests that POST the form data to the server. Here Dojo decides the encoding by default, which does not match the browser's behaviour of using the charset defined in the META tag. But you can easily tell Dojo which formatting to use for all dojo.io.bind() calls, just include that in the top of your HTML pages, before dojo.js is included:

```
<script>djConfig = { bindEncoding: "utf-8" };</script>
```

You might already have other djConfig options, then simply add the `bindEncoding` property to the hash map.

3. Decoding incoming requests: Servlet Container

When the browser sends stuff to your server, eg. form data, the `ServletRequest` will be created by your servlet container, which needs to decode the parameters correctly into Java Strings. If there is the encoding specified in the HTTP request header, he will use that, but unfortunately this is typically not the case. When the browser sends a form post, he will only say `application/x-www-form-urlencoded` in the header. So you have to assume the encoding here, and the right thing to assume is the encoding of the page you originally sent to the browser.

The servlet standard says that the default encoding for incoming requests should be ISO-8859-1 (Jetty is not according to the standard here, it assumes UTF-8 by default). So to make sure UTF-8 is used for the parameter decoding, you have to tell your servlet that encoding explicitly. This is done by calling S`ervletRequest.setCharacterEncoding()`. To do that for all your requests, you can use a servlet filter like this one: SetCharacterEncodingFilter.

Then you add the filter to the web.xml:

```
<filter>
  <filter-name>SetCharacterEncoding</filter-name>
  <filter-class>filters.SetCharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
</filter>

<!-- either mapping to URL pattern -->

<filter-mapping>
  <filter-name>SetCharacterEncoding</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- or mapping to your Cocoon servlet (the servlet-name might be different) -->

<filter-mapping>
  <filter-name>SetCharacterEncoding</filter-name>
  <servlet-name>CocoonBlocksDispatcherServlet</servlet-name>
</filter-mapping>
```

Since the filter element was added in the servlet 2.3 specification, you need at least 2.3 in your web.xml, but using the current 2.4 version is better, it's the standard for Cocoon webapplications. For 2.4 you use a XSD schema:

```
<web-app version="2.4"
         xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
```

For 2.3 you need to modify the DOCTYPE declaration in the web.xml:

```
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
```

4. Setting Cocoon's encoding (especially CForms)

To tell Cocoon to use UTF-8 internally, you have to set 2 properties:

```
org.apache.cocoon.containerencoding=utf-8
org.apache.cocoon.formencoding=utf-8
```

They need to be in some `*.properties` file under `META-INF/cocoon/properties` in one of your blocks.

5. XML Files

This is normally not a problem, since the standard encoding for XML files is UTF-8. However, they should always start with the following instruction, which should force your XML Editor to save them in UTF-8 (it looks like most of them do that, so there should not be a problem here).

```
<?xml version="1.0" encoding="UTF-8"?>
```

6. Special Transformers

The standard XSLT Transformers and others are working on SAX events, which are not serialized, thus encoding is not a problem. But there are some special transformers that pass stuff on to another library that does include serialization and might need a hint to use the correct encoding. One problem is for example the NekoHTMLTransformer: https://issues.apache.org/jira/browse/COCOON-2063.

If you think there might be a transformer doing things wrong in your pipeline, add a `TeeTransformer` between each step, outputting the XML between the transformers into temp1.xml, temp2.xml and so on to look for the place where your umlaute and special characters are messed up.

7. Your own XML serializing Sources

If you have your own Source implementation that needs to serialize XML, make sure it will do that in UTF-8 as well. A good idea is to use Cocoon's XML serializer, since we already configured that one to UTF-8 above. Sample code that does that is here: UseCocoonXMLSerializerCode

Older documentation

Basics

If your Cocoon application needs to read request parameters that could contain *special* characters, i.e. characters outside of the first 128 ASCII characters, you'll need to pay attention to what encoding is used.

Normally a browser will send data to the server using the same encoding as the page containing the submitted form (or whatever). So if the pages are serialized using UTF-8, the browser will submit form data using UTF-8. The user can change the encoding, but it's quite safe to assume he/she won't do that (have you ever done it?).

In my browser this is the case, it is set in the preferences to ISO-8859-1 and he encodes form parameters with that, regardless of the UTF-8 content type of the page containing the form. I can't remember when I did set this property... So what to do with this case? This means, it could be any encoding. – AlexanderKlimetschek

After doing some tests with popular browsers, I've noticed that usually browsers will not let the server know what encoding they used to encode the parameters, so we need to make sure ourselves that the encoding used when serializing pages corresponds to the encoding used when decoding request parameters.

First of all, check in the sitemap what encoding is used when serializing HTML pages:
<encoding>UTF-8</encoding>

```
<map:serializer logger="sitemap.serializer.html" mime-type="text/html"
       name="html" pool-grow="4" pool-max="32" pool-min="4"
       src="org.apache.cocoon.serialization.HTMLSerializer">
  <buffer-size>1024</buffer-size>
  <encoding>UTF-8</encoding>
</map:serializer>
```

In the example above, UTF-8 is the encoding used. This is a widely supported Unicode encoding, so it is often a good choice.

The HTML serializer will automatically insert a <meta> tag into the HTML page's HEAD element specifying the encoding. Most browsers apparently require this. The HTML serializer will however only do this if your page already contains a HEAD (or head) element, so make sure it has one. The <meta> element inserted by the serializer will then look as follows:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

Mozilla (tested with 1.4), netscape 7.1 and Internet Explorer 6 will not respond to the setting of this meta tag, whereas they do respond to the http response header "Content-Type". So you may have to subclass the HTMLSerializer and let it add this header in order to get Mozilla and IE working.

– Someone added this last paragraph here. Good advice (haven't found time to verify it yet though), but if this is the case we should fix this in Cocoon. Patches welcome in bugzilla. (BrunoDumon).

*– I can confirm it and the effect is obvious when using a recent Tomcat (> 4.1.27): Bug #26997. But AFAIK the above must read 'will not respond to the setting of this meta tag **if** the encoding/charset in the "Content-Type" header is set' and Cocoon's problem is, that it does not set the encoding/charset and the recent Tomcats sets it to default ISO-8859-1. (JoergHeinicke)*

– _But you can make Cocoon set the header by configuring the serializer with the correct mime-type information:

```
<map:serializer name="html" mime-type="text/html; charset=utf-8"
       src="org.apache.cocoon.serialization.HTMLSerializer"
       logger="sitemap.serializer.html"
       pool-grow="4" pool-max="32" pool-min="4">
  <buffer-size>1024</buffer-size>
  <encoding>UTF-8</encoding>
</map:serializer>
```

The first `charset=utf-8` is needed for the HTTP header whereas `<encoding>UTF-8</encoding>` seems to be responsible for the encoding only of the document's content. (Volkmar W. Pogatzki)_

By default, if the browser doesn't explicitely mention the encoding, a servlet container will decode request parameters using the ISO-8859-1 encoding (independent of the platform on which the container is running). So in the above case where UTF-8 was used when serializing, we would be facing problems.

Note: Jetty uses [UTF-8 as default for decoding form parameters|http://docs.codehaus.org/display/JETTY/International+Characters+and+Character+Encodings]! So you have to use the `SetCharacterEncodingFilter` (see below) to set the encoding for Jetty to ISO-8859-1 if this is what the browser sends. --AlexanderKlimetschek

The encoding to use when decoding request parameters can be configured in the web.xml by supplying init parameters called "form-encoding" and "container-encoding" to the Cocoon servlet. The container-encoding parameter indicates according to what encoding the container tried to decode the request parameters (normally ISO-8859-1), and the form-encoding parameter indicates the actual encoding. Here's an example of how to specify the parameters in the web.xml:

```
<init-param>
  <param-name>container-encoding</param-name>
  <param-value>ISO-8859-1</param-value>
</init-param>
<init-param>
  <param-name>form-encoding</param-name>
  <param-value>UTF-8</param-value>
</init-param>
```

For Java-insiders: what Cocoon actually does internally is apply the following trick to get a parameter correctly decoded: suppose "value" is a string containing a request parameter, then Cocoon will do:

```
value = new String(value.getBytes("ISO-8859-1"), "UTF-8");
```

So it recodes the incorrectly decoded string back to bytes and decodes it using the correct encoding.

Locally overriding the form-encoding

Cocoon is ideally suited for publishing to different kinds of devices, and it may well be possible that for certain devices, it is required to use different encodings. In this case, you can redefine the form-encoding for specific pipelines using the SetCharacterEncodingAction.

To use it, first of all make sure the action is declared in the map:actions element of the sitemap:

```
<map:action name="set-encoding" src="org.apache.cocoon.acting.SetCharacterEncodingAction"/>
```

and then call the action at the required location as follows:

```
<map:act type="set-encoding">
  <map:parameter name="form-encoding" value="some-other-encoding"/>
</map:act>
```

Problems with components using the original HttpServletRequest (JSPGenerator, ...)

Some components such as the JSPGenerator use the original HttpServletRequest object, instead of the Cocoon Request object. In that case, the correct decoding of request parameters will not happen (that is, if for example the JSP page itself would read request parameters).

One possible solution would be to patch these components to use a wrapper class that delegates all calls to the HttpServletRequest object, except for the getParameter or getParameterValues methods, which should be delegated to Cocoon's Request object.

There's an easier solution that can be applied right away if your servlet container supports the Servlet 2.3 specification. Starting from 2.3, the Servlet specification allows to explicitely set the encoding to be used for decoding request parameters, though this has to happen before the first request data is read. Since Cocoon reads request parameters itself (such as cocoon-reload), this would require modification of the CocoonServlet. But it can also be done using a servlet filter. Tomcat 4 contains just such a filter in its "examples" webapp. Look for the file jakarta-tomcat/webapps/examples/WEB-INF/classes/filters/SetCharacterEncodingFilter.java. Compile it (with servlet.jar in the classpath), put it in a jar (using correct package and such) and put the jar in your webapps WEB-INF/lib directory.

Now modify your webapp's web.xml file to include the following (after the display-name and description elements, but before the servlet element):

```
<filter>
  <filter-name>Set Character Encoding</filter-name>
  <filter-class>filters.SetCharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>Set Character Encoding</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Since the filter element is new in the servlet 2.3 specification, you might need to modify the DOCTYPE declaration in the web.xml:

```
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
```

Of course, when using a servlet filter to set the encoding, you should not supply the form-encoding init parameter anymore in the web.xml. You could still supply the container-encoding parameter, though its value will now have to be the same as the encoding supplied to the filter. This will allow you to override the form-encoding using the SetCharacterEncodingAction, though only for the Cocoon Request object.

Using a servlet filter also has the advantage that it will work for any servlet. Suppose your webapp consists of multiple servlets, with Cocoon being only one of them. Sometimes the processing could start in another servlet (which sets the character encoding correctly) and then be forwarded to Cocoon, while other times the processing could start immediately in the Cocoon servlet. It would then be impossible to know in Cocoon whether the request parameter encoding needs to be corrected or not.

Operating System Preliminaries

Working with non-english characters may also pose problems depending on the operations system settings, as the JVM seems to detect the system language. If, e.g., german umlauts should be correctly processed with Cocoon on Linux, it is required to set the LANG environment variable to de like this:

```
export LANG=de
```

The remark in this last paragraph won't have any influence on request parameter decoding, though it might help for other things (reading text files, communication with database, ...) – BrunoDumon

(That's one of several ways of setting the JVM locale, see also SettingTheJvmLocale).

Just came across this today: http://www.w3.org/TR/REC-html40/interact/forms.html#adef-accept-charset This looks like related stuff we should investigate. (and test the current browsers for) – MarcPortier

More readings

- cocoon's defaults form-encoding and seerialize-encoding MarcPortier proposal to remove inconsitencies in the way Cocoon handles the encoding of serialized text and request-parameter decoding. This is a good summary of the thread.

What about file names for uploaded files?

I saw some reports on users list about file names being *wrongly* encoded, i.e. though everything else on a form works, the file name of an uploaded file is wrong when *special* characters were used. I never tested it though. (JoergHeinicke)

The Bug 24289 - "MultipartParser cannot handle multibyte character in uploaded file name correctly" might explain it. (JoergHeinicke)