HttpEntity

HttpEntity

The HttpEntity is an interface which represents the body of a request/response.

Description for HttpEntity

An entity that can be sent or received with an HTTP message. Entities can be found in some requests and in responses, where they are optional. Requests that use entities are POST/PUT, responses that don't use entities, are HEAD.

In some places, the JavaDoc distinguishes three kinds of entities, depending on where their content originates:

1. streamed: The content is received from a stream, or generated on the fly. In particular, this category includes entities being received from a connection. Streamed entities are generally not repeatable.
2. self-contained: The content is in memory or obtained by means that are independent from a connection or other entity. Self-contained entities are generally repeatable.
3. wrapping: The content is obtained from another entity.

This distinction is important for connection management with incoming entities. For entities that are created by an application and only sent using the HTTP components framework, the difference between streamed and self-contained is of little importance. In that case, it is suggested to consider non-repeatable entities as streamed, and those that are repeatable (without a huge effort) as self-contained.

Using Response Entities

Since an entity can represent both binary and character content, it has support for character encodings (to support the latter, ie. character content).

The entity is created when the request was successful, and used to read the response.

To read the content from the entity, you can either retrieve the input stream via the HttpEntity.getContent() method, which returns an InputStream, or you can supply an output stream to the HttpEntity.writeTo(OutputStream) method, which will return once all content has been written to the given stream.

When the entity was received as a result of a response, the methods getContentType() and getContentLength() methods are for reading the common headers Content-Type and Content-Length respectively (if they are available). Since the Content-Type header can contain a character encoding for text mime-types like text/plain or text/html, the getContentEncoding() method is used to read this information. If the headers aren't available, a length of -1 will be returned, and NULL for the content-type. If the Content-Type header is available, a [Header] object will be returned.

When creating an entity for a request, this meta data has to be supplied by the creator of the entity.

Other headers from the response are read using the getHeaders() methods from the response object.

EntityUtils

This is a utility class, that exposes 4 static methods to more easily read the content or information from an entity. Instead of reading the InputStream yourself, you can retrieve the whole content body in a String/byte array by using the methods from this class.

One of these static methods are for retrieving the content character set, where the other 3 are for reading the content itself.

<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="3ae3be0d-0179-46c5-8d95-087db999cbb6"><ac:plain-text-body><![CDATA[	byte[] toByteArray(HttpEntity)	Returns the content body in a byte[]	]]></ac:plain-text-body></ac:structured-macro>

String toString(HttpEntity)	Returns the content body in a String
String toString(HttpEntity, String defaultCharset)	Returns the content body in a String, using the character set supplied as the second argument if the character set wasn't set upon creation. This will be the case if you didn't specify it when creating the Entity, or when the response didn't supply the character set in the response headers

Entities Content

Repeatable Entities

An entity can be repeatable, meaning it's content can be read more than once. This is only possible with self contained entities (like ByteArrayEntity or StringEntity).

Consuming Content

When you are finished with an entity that relies on an underlying input stream, it's important to execute the consumeContent() method, so as to clean up any available content on the stream so the connection be released to any connection pools. If you don't do this, other requests can't reuse this connection, since there are still available information on the buffer.

Alernatively you can simple check the result of isStreaming(), and keep reading from the input stream until it returns false.

Self contained entities will always return false with isStreaming(), as there is no underlying stream it depends on. For these entities consumeContent() will do nothing, and does not need to be called.

Creating Entities

There are a few ways to create entities. You would want to do this when you implement custom responses, or send POST/PUT requests.

These classes include the following implementations provided by HttpComponents:

1. BasicHttpEntity
2. BufferedHttpEntity
3. ByteArrayEntity
4. EntityTemplate
5. FileEntity
6. InputStreamEntity
7. StringEntity
8. AbstractHttpEntity
9. HttpEntityWrapper

BasicHttpEntity

This is exactly as the name implies, a basic entity that represents an underlying stream. This is generally the entities received from HTTP responses.

This entity has an empty constructor. After constructor it represents no content, and has a negative content length.

You need to set the content stream, and optionally the length. You can do this with the setContent(java.io.InputStream) and setContentLength(long length) methods respectively.

Example:

```
numbers=off
 BasicHttpEntity myEntity = new BasicHttpEntity();
 myEntity.setContent(someInputStream);
 myEntity.setContentLength(340); // sets the length to 340
```

BufferedHttpEntity

This is an entity wrapper. It is constructed by supplying another entity. It reads the content from the supplied entity, and buffers it in memory.

This allows you to make a repeatable entity, from a non-repeatable entity. If the supplied entity is already repeated, calls are simply passed through to the underlying entity.

```
numbers=off
 BufferedHttpEntity myEntity = new BufferedHttpEntity(myNonRepeatableEntity);
```

ByteArrayEntity

This is a simple self contained repeatable entity, which receives it's content from a given byte array. This byte array is supplied to the constructor.

```
numbers=off
 String myData = "Hello world on the other side!!";
 ByteArrayEntity myEntity = new ByteArrayEntity(myData.getBytes());
```

EntityTemplate

This is an entity which receives it's content from a ContentProducer. Content producers are objects which produce their content on demand, by writing it out to an output stream. They are expected to be able produce their content every time they are requested to do so. So creating a EntityTemplate, you supply a reference to a content producer, which effectively creates a repeatable entity.

There are no standard ContentProducers in HttpComponents. It's basically just a convenience interface to allow wrapping up complex logic into an entity. To use this entity you will need to create a class that implements ContentProducer and override the writeTo(OutputStream) method. Inside this method you will write the full content body to the output stream.

If you for instance made a HTTP server, you would serve static files with the FileEntity, but running CGI programs can be done with a ContentProducer, inside which you run the program and supply the content as it becomes available. This way you don't need to buffer it in a string and then use a StringEntity or ByteArrayEntity. Like I mentioned, a convenience class.

```
numbers=off
   ContentProducer myContentProducer = new ContentProducer() {
     @Override
     public void writeTo(OutputStream out) throws IOException
     {
       out.write("ContentProducer rocks!".getBytes());
       out.write(("Time requested: " + new Date()).getBytes());
     }
   };
   HttpEntity myEntity = new EntityTemplate(myContentProducer);
```

FileEntity

This entity is reads it's content body from a file. Since this is mostly used to stream large files of different types, you need to supply the content type of the file, for instance, sending a zip you would supply the content type "application/zip", for XML "application/xml".

```
numbers=off
 File staticFile = new File("/path/to/a-class-software/httpcore-4.0-beta1.jar");
 HttpEntity entity = new FileEntity(staticFile, "application/java-archive");
```

InputStreamEntity

An entity that reads it's content from an input stream. It is constructed by supplied the input stream as well as the content length.

The content length is used to limit the amount of data read from the InputStream. If the length matches the content length available on the input stream, then all data will be sent. Alternatively a negative content length will read all data from the input stream, which is the same as supplying the exact content length, so the length is most often used to limit the length.

```
numbers=off
 InputStream instream = getSomeInputStream();
 InputStreamEntity myEntity = new InputStreamEntity(instream, 16);
```

StringEntity

Very simple entity. It's is a self contained, repeatable entity that retrieves it's data from a String object.

It has 2 constructors, one simply constructs with a given String object where the other also takes a character encoding for the data in the String.

```
numbers=off
 StringBuffer sb = new StringBuffer();
 Map<String, String> env = System.getenv();
 for (Entry<String, String> envEntry : env.entrySet())
 {
   sb.append(envEntry.getKey()).append(": ").append(envEntry.getValue()).append("\n");
 }

 // construct without a character encoding
 HttpEntity myEntity1 = new StringEntity(sb.toString());

 // alternatively construct with an encoding
 HttpEntity myEntity2 = new StringEntity(sb.toString(), "UTF-8");
```

AbstractHttpEntity

This is the base class for streaming and self contained entities. It provides all the commonly used attributes used by these entities, like contentEncoding, contentType and the chunked flag.

HttpEntityWrapper

This is the base class for creating wrapped entities. It contains an instance variable wrappedEntity which holds the instance to the wrapped entity.

BufferedHttpEntity is a subclass of HttpEntityWrapper.

Request entities - Sending a POST request with HttpClient

... to be completed ...