

International Characters and Character Encodings

International Characters and Character Encoding of Requests

Internet standards support for international character sets was historically weak, is now improving, but is still not perfect. The current standards situation is confusing, and has led to false expectations about what can be reliably achieved using today's browsers, protocols and servers.

Jetty fully supports and implements the current relevant standards and specifications, but this alone is not sufficient to make working with international characters easy or reliable in all situations. This FAQ explains the current standards, provides hints and tips for building and decoding Internationalised web pages (including ones with dynamic data) and explains how Jetty has anticipated the probable future direction of the standards.

The intended readership is people developing Servlet applications and their associated web pages. A basic knowledge of Java, HTML and of hexadecimal notation is assumed.

Unless otherwise stated, the information below applies to all current (August 2002) standards-conformant Web Servers, not just to Jetty.

Primer and Terminology

There are four groups whose standards and specifications affect character handling in web pages:

1. The Internet Engineering Task Force (IETF) manages the Requests for Comments (RFCs) which define the basic transportation protocols, including the HyperTextTransfer Protocol (HTTP) and Uniform Resource Locators (URLs) with which we are concerned.
2. The World Wide Web Consortium (W3C) manages the HTML and XHTML standards, with which web pages are built, and the XML standard, used in Jetty configuration.
3. The Internet Assigned Numbers Authority (IANA), which, among other duties, maintains a list of character encodings.
4. Sun Microsystems, Inc. owns the Servlet specification, which is expressed in Java APIs.
The Internet was initially designed and constructed using basic English characters encoded in the 7-bit US-ASCII character set. This provides 26 upper and lower case letters, 10 digits, and a variety of punctuation and other symbols - it encodes 95 'printing' characters. Today, these are still the only printable characters which can be used in HTTP.

A single byte (8 bits) has the capacity to represent up to 256 characters. There are several widely-used encodings which give the US-ASCII characters their normal values in the range 0-127, and a selection of other characters are assigned code values in the range 128-255. In many web browsers the encoding to use can either be specified by the web page designer or selected by the user. Some of these encodings are proprietary, others specified by consortia or international standards bodies.

The first approaches to supporting international characters involved selecting one of these 8-bit character sets, and attempting to ensure that the web page source, the browser, and any server using data from that browser were using the same character encoding.

There is a default character set ISO-8859-1, which supports most western European languages, and is currently the official 'default' content encoding for content carried by HTTP (but not for data included in URLs - see below). This is also the default for all Jetty versions except 4.0.x. Alternative encodings may be specified by defining the HTTP Content-Type header of the page to be something like "text/html; charset=ISO-8859-4". From a Servlet you can use `request.setContentType("text/html; charset=ISO-8859-4");`. Pages can then be composed using the desired literal

characters (e.g. accented letters from Europe or, with a different encoding selected, Japanese characters). This mechanism can be unreliable; the browser's user can select the encoding to be applied, which may be different from that intended by the servlet designer.

Today the Internet is converging on a single, common encoding - Unicode - in which can be represented all known written languages, as well as a wide range of symbols (e.g. mathematical symbols and decorative marks). Unicode requires a 16-bit integer value to represent each character. By design, the 95 printable US-ASCII characters have the same code values in Unicode; US-ASCII is a subset of Unicode. Most modern browsers can decode and display a wide range of the characters represented by Unicode - but it would be rare to find a browser capable of displaying all the Unicode characters.

Unicode is the only character encoding used in XML and is now the default in HTML, XHTML and in most Java implementations.

The Internet transmits data in groups of 8 bits, which the IETF usually call 'octets', but everyone else calls 'bytes'. When larger values have to be sent, such as the 16 bits needed for Unicode and some other international character encodings, there has to be a convention on how the 16 bits are packed into one or more octets. There are two standards commonly used to encode Unicode: UTF-8 and UTF-16.

UTF-16 is the 'brute-force' encoding for data transmission. The 16 bits representing the character value are placed in adjacent octets. Variants of UTF-16 place the octets in different orders.

UTF-8 is more common, and is recommended for most purposes. Characters with values less 0080 (hexadecimal notation) are transmitted as a single octet whose value is that of the character. Characters with values in the range 0080 to 07FF are encoded as two octets, whilst the (infrequently-used) Unicode characters with values between 0800 and FFFF are encoded into three octets. This encoding has the really useful property that a sequence of (7-bit) US-ASCII characters sent as bytes and then sent as Unicode UTF-8 octets produce identical octet streams - a US-ASCII byte stream is a valid UTF-8 octet stream and represents the same printing characters. This is not the case if other characters are being sent, or if UTF-16 is in use.

As well as having US-ASCII compatibility, UTF-8 is preferred because, in the majority of situations, it results in shorter messages than UTF-16.

Note that, when UTF-8 is specified, it not only defines the way in which the character code values are packed into octets, it also implicitly defines the character encoding in use as Unicode.

There is an international standard - ISO-10646 - which defines an identical character set to Unicode - but omits much essential additional information. For most purposes refer to Unicode rather than ISO-10646.

Characters included in HTTP requests

There are two places in which HTTP requests (from browsers to web servers) may include character data:

1. In the URL part of the first line of the HTTP request,
2. In the HTTP content area at the end of the HTTP message, resulting from an HTML `<form method='post'...> ... </form>` submission

Characters in the HTTP content part

Wherever possible, a POST method should be used when international characters are involved.

This is because the browser sends a HTTP `Content-Type` header which *can* help the web server determine the encoding of the content. The `Content-Type` header will tell the server the MIME-type encoding of the content (usually `application/x-www-form-urlencoded`) and also can *optionally* include the character encoding of the

content eg:

```
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
```

If both the MIME-type and the charset encoding information is sent in the POST HTTP header, the server can correctly decode the content.

Unfortunately, many browsers do not bother to send the charset information, leaving the web server to guess the correct encoding. For this reason, the Servlet API provides the [ServletRequest.setCharacterEncoding\(String\)](#) method to allow the webapp developer to control the decoding of the form content.

Jetty-6 uses a default of UTF-8 if no overriding character encoding is set on a request.

International characters in URLs

A typical URL looks like:

<http://www.my.site/dir/page.html>

When a form is sent, using the default GET method, the data values from the form are included in the URL, e.g.:

<http://www.my.site/dir/page.html?name=Durst&age=18>

It is important to note that only a very restricted sub-set of the US-ASCII printing characters are permitted in URLs.

Something like `name=Dürst` (with an umlaut) is illegal. It might work with some browser/server combinations (and might even deliver the expected value), but it should never be used.

The HTTP Specification provides an 'escape' mechanism, which permits arbitrary octet values to be included in the URL. The three characters %HH - where HH are hexadecimal characters - inserts the specified octet value into the URL. This has to be used for the US-ASCII characters which may not appear literally in URLs, and can be used for other octet values.

It is a common fallacy that this permits international characters to be reliably transmitted. **This is wrong.**

This is because the %HH escape permits the transmission of a sequence of octets, but has nothing to say about what character encoding is in use.

There is no provision in the HTTP protocol for the sender (the browser) to tell the receiver (the web server) what encoding has been used in the URI, and none of the specifications related to HTTP/HTML define a default encoding.

Thus, although any desired octet sequence can be placed in a URL, none of the standards tell the web server how to interpret that octet sequence.

The designers of web servers with Servlet APIs currently have a problem. They are presented with an octet stream of unspecified encoding, and yet have to deliver a Java String (a sequence of decoded characters) to the Servlet API.

Due to the lack of a standard, different browsers took different approaches to the character encoding used. Some use the encoding of the page and some use UTF-8. Some drafts were prepared by various standards bodies suggesting that UTF-8 would become the standard encoding. Older versions of jetty (eg 4.0.x series) used UTF-8 as the default in anticipation of a standard being adopted. As a standard was not forthcoming, jetty-4.1.x reverted to a default encoding of ISO-8859-1.

The W3C organization's HTML standard now recommends the use of UTF-8: <http://www.w3.org/TR/html40/appendi>

[x/notes.html#non-ascii-chars](#) and accordingly jetty-6 series uses a default of UTF-8.

If UTF-8 is not correct for your environment, you may use one of two jetty-specific methods to set the charset encoding of the query string in GET requests:

1. call `Request.setQueryEncoding(String)` **before reading any of the content or params.**
2. set the system property `org.mortbay.util.URI.charset` to the encoding you want to use.

Handling of International characters by browsers

There are many ways in which international characters can be displayed or placed into browsers for inclusion in HTTP requests. Some examples are:

- `<h3>Dürst</h3> <!-- Assumes specific character encoding -->`
- `<h3>Dürst</h3>`
- `<h3>Dürst</h3>`
- `<h3>Dürst;</h3>`
- `<input type='text' value='Dürst'>`
- `<script language='javascript'>name="D\u00FCrst";</script>`
- `<form action='/servlet?name=D%C3%BCrst'>...</form>`
- `...`

It is also possible to manipulate document text using the DOM APIs.

It is believed that, in all the above examples, all modern browsers (those supporting HTML 4) will treat the `&...;` encoding as representing Unicode characters. Earlier ones may not understand this encoding.

The first example, with the literal ü, should only be used if the character encoding can be relied upon, and if support for 'legacy' browsers (those not understanding the `&...;` encoding) is essential.

It is, of course, possible for users to enter characters using `<input..>` and `<textarea...>` elements via the operating system. Text can come from keyboards, and also from 'cut and paste' mechanisms. It appears that most browsers use their current (user-selectable) 'Encoding' setting (e.g. in MSIE: View..Encoding) to encode such characters. After the User has selected the encoding to use, it appears that many browsers will transmit the data characters in the request in that locally-defined encoding, rather than the one specified with the page.

Techniques for working with international characters

The only reliable, standards-supported way to handle international characters in a browser- and server-neutral way appears to be:

1. To specify the contents of the HTML page using `&...;` encoding and
2. To use the `<form method='post' ...>` method of submission.

Need for GET-method support

It is sometimes suggested that all forms can and should be submitted using the above POST method. There is, in fact, a valid need to use the default GET method.

To appreciate this need one must consider carefully a significant difference between submitting a form using POST and GET. When using GET the data values from the form are appended to the URI and form part of the visible 'address' seen at the top of the browser. It is also this entire string that is saved if the page is bookmarked by the browser, and may be moved using 'cut-and-paste' into an eMail, another web page etc..

It is possible that the dynamic data from the form is an integral part of the semantics of the 'page address' to be stored.

The address may be part of a map; one of the data values from the map may define the town on which the map view is to be centered - and this name may only be expressible in, say, Thai characters. The town name may have been entered or selected by the user - it was not a 'literal' in the HTML defining the page or in the URL.

Another common need is to 'bookmark' or sent by eMail the request string from a search engine request which has non-ASCII characters in the search.

There is not yet any standards-based way of constructing this dynamically-defined URL - there is no direct way to be certain what character encodings have been applied in constructing the URI-with-query string that the browser generates.

A work-around which has been suggested is to provide additional, known text fields alongside the unknown text. In the example above, the form with the dynamically-defined town name could also have a hidden field into which the generated page inserts 'known' text from the same character set (using the &...; encoding). When the request is eventually received by a servlet the octet contents of the known field are inspected (typically by using `request.getQueryString()`). If the characters of the 'tracer' field are the same as those injected into the page when it was generated (and the character code values encompass those of the unknown town) then there is an assumption that the encoding used was Unicode and that the town name as present in Java is accurate.

If the the actual encoding can be deduced from the 'tracer' octets, the Servlet API `request.setCharacterEncoding()` can be called (before calling any of the `.getParameter()` methods) to tell the web server which encoding to assume when decoding the query.

There is an obvious potential flaw in this 'tracer' technique - the browser may represent &...;-specified 'tracer' text with its Unicode values, yet may use the local keyboard/operating system encoding for locally-entered data. The author is not aware of any conclusive knowledge or evidence in this area.

An alternative work-around, which is more complex but might be more certain if GET has to be used, would be to use Javascript or the DOM interfaces to transfer the characters from the input fields to the query part of the URL string.

International characters in Jetty configuration files

Jetty configuration files use XML. If international characters need to be included in these files, the standard XML character encoding method should be used. Where the character has a defined abbreviation (such as `ü` for u-umlaut), that should be used. In other cases the hexadecimal description of the character's Unicode code value should be used. For example `¡` defines the Greek capital A letter. Use of the decimal form `¡` seems now to be unfashionable in W3C circles.

Future possibilities

It is to be hoped that something like the IRI scheme described in the Internet Draft will evolve into a standard that will be adopted by suppliers of web servers and browsers. It will probably also need changes to HTTP and/or the use of internationalised versions of the http and https protocols. As currently drafted, such a scheme would not, of its own solve the problem of dynamic data derived from form GET submissions. This will require changes to HTML4 or, more likely, extensions to a future version of XHTML.

The whole area of form data handling may be radically improved if the Xforms program is successful. This has defined an XML-based approach to forms and associated data and event handling and uses Unicode throughout. The Xforms 1.0 specification is currently (August 2002) at 'last call working draft' status, and a number of

experimental implementations, some using browser applets or plug-ins, have been announced.

Neither of these likely developments will improve the handling of international characters by 'today's' browsers, so designers of web services for the 'open' market seem likely to have to work within today's constraints for a long time.

Anyone interested in the full complexity of handling international characters and languages might like to read the W3C's Character Model (currently a working draft) and follow the W3C's International Activity.

originally contributed by Chris Hayes