

The Multilingual WWW

Table of Contents

<u>The Multilingual World Wide Web.....</u>	<u>1</u>
<u>Introduction The World Wide Web has enjoyed explosive growth in recent years, and there are now millions of</u>	
<u>1. Requirements for multilingual applications.....</u>	<u>2</u>
<u>1.1 Data Representation Requirements.....</u>	<u>2</u>
<u>1.2 Data Manipulation Issues.....</u>	<u>2</u>
<u>1.3 Data Display Requirements.....</u>	<u>3</u>
<u>1.4 Data Input Requirements.....</u>	<u>4</u>
<u>1.5 Summary Of Requirements.....</u>	<u>4</u>
<u>2. The WWW As A Multilingual Application.....</u>	<u>4</u>
<u>2.1 Data Representation Requirements for the WWW.....</u>	<u>4</u>
<u>2.2 Data Manipulation Requirements for the WWW.....</u>	<u>5</u>
<u>2.3 Data Display Requirements for the WWW.....</u>	<u>5</u>
<u>2.4 Data Input Requirements for the WWW.....</u>	<u>6</u>
<u>3. Issues Facing Multilingual Support in the WWW.....</u>	<u>6</u>
<u>3.1 MIME Issues.....</u>	<u>6</u>
<u>3.2 SGML Issues.....</u>	<u>7</u>
<u>3.3 Parsing Issues Facing Browsers.....</u>	<u>8</u>
<u>3.4 Browser Presentation Issues.....</u>	<u>8</u>
<u>3.5 Server Issues.....</u>	<u>8</u>
<u>4. Possible Solutions to the Issues.....</u>	<u>9</u>
<u>4.1 MIME and HTTP Resolution.....</u>	<u>9</u>
<u>4.2 Solutions to SGML's Problems.....</u>	<u>10</u>
<u>4.3 Parsing.....</u>	<u>11</u>
<u>4.4 Browsers.....</u>	<u>12</u>
<u>4.5 Servers.....</u>	<u>12</u>
<u>5. Recommendations.....</u>	<u>12</u>
<u>5.1 Changes to the SGML Declaration.....</u>	<u>13</u>
<u>5.2 Changes to HTML.....</u>	<u>14</u>
<u>5.3 Changes to Clients.....</u>	<u>14</u>
<u>5.4 Changes to Servers.....</u>	<u>15</u>
<u>6. Discussion.....</u>	<u>15</u>
<u>Appendices.....</u>	<u>15</u>
<u>A-1. The SGML Processing Model.....</u>	<u>15</u>
<u>A-2. Conversion Servers.....</u>	<u>15</u>
<u>A-3. Implementing Unicode.....</u>	<u>16</u>
<u>A-4. Problems With Unicode.....</u>	<u>16</u>
<u>A-5. Problems Facing ERCS.....</u>	<u>16</u>
<u>A-6. A Simple VM for use in Conversion.....</u>	<u>16</u>
<u>Glossary Of Terms.....</u>	<u>16</u>
<u>Bibliography.....</u>	<u>17</u>

The Multilingual World Wide Web

[Introduction](#)

[1. Requirements for multilingual applications](#)

[1.1 Data Representation Requirements](#)

[1.2 Data Manipulation Issues](#)

[1.3 Data Display Requirements](#)

[1.4 Data Input Requirements](#)

[1.5 Summary Of Requirements](#)

[2. The WWW As A Multilingual Application](#)

[2.1 Data Representation Requirements for the WWW](#)

[2.2 Data Manipulation Requirements for the WWW](#)

[2.3 Data Display Requirements for the WWW](#)

[2.4 Data Input Requirements for the WWW](#)

[3. Issues Facing Multilingual Support in the WWW](#)

[3.1 MIME Issues](#)

[3.2 SGML Issues](#)

[3.3 Parsing Issues Facing Browsers](#)

[3.4 Browser Presentation Issues](#)

[3.5 Server Issues](#)

[4. Possible Solutions to the Issues](#)

[4.1 MIME and Multilingual Data Resolution](#)

[4.2 Solutions to SGML's Problems](#)

[4.3 Parsing](#)

[4.4 Browsers](#)

[4.5 Servers](#)

[5. Recommendations](#)

[5.1 Changes to the SGML Declaration](#)

[5.2 Changes to HTML](#)

[5.3 Changes to Clients](#)

[5.4 Changes to Servers](#)

[6. Discussion](#)

[Appendices](#)

[A-1. The SGML Processing Model](#)

[A-2. Conversion Servers](#)

[A-3. Implementing Unicode](#)

[A-4. Problems With Unicode](#)

[A-5. Problems Facing ERCS](#)

[A-6. A Simple VM for use in Conversion](#)

[Glossary Of Terms](#)

[Bibliography](#)

[Table Of Contents](#)

Introduction

The World Wide Web has enjoyed explosive growth in recent years, and there are now millions of people using it all around the world. Despite the fact that the Internet, and the World Wide Web, span the globe,

there is, as yet, no well-defined way of handling documents that contain multiple languages, character sets, or encodings thereof. Rather, ad hoc solutions abound, which, if left unchecked, could lead to groups of users

applications, data is primarily manipulated in its encoded form, and in others, the entire application is built around [wide characters](#). ANSI C contains a definition of wide characters, and a group of functions defined in order to support the locale concept.

Most of the tradeoffs are quite obvious, but some of the most important are listed below, for completeness:

Memory usage.

Obviously, using 16 bits for all data can be inefficient in cases where most data can be represented using 8 bits.

Backward compatibility

Encoded strings can generally be treated as normal C strings, whereas wide strings cannot because they could contain bytes with a value of 0, which is the string terminator in C. It should be noted that as object oriented programming languages become more popular, this argument will lose its meaning: strings will be just another object.

Synchronisation

In many wide character representations, there is a one-to-one mapping between characters codes and the integers used to hold them. This means that the data tends to be self-synchronous, whereas encoded strings are generally not (in other words, you can move through a wide string by simply incrementing a counter, whereas encoded strings often require actual parsing of the data stream). The benefit of self-synchronicity is often understated.

1.3 Data Display Requirements

A visually-oriented multilingual application must obviously be able to present multilingual data to the reader in a sensible manner. The most commonly cited problem here is font mapping, however, languages around the world have different writing directions as well, and some languages have mixed writing directions, which should also be handled in the appropriate manner. Another point worthy of mention is that there is a very common misconception that if one can simply map one-to-one from a character code to a glyph image, nothing more is required. Sadly, this is not true: typographers can point to cases where a single character has multiple glyph images, even within the roman languages, and cases abound in other languages. This means that for any system attempting even low quality typographic output, glyph selection processing, rather than simple character code to glyph mapping, is required. See the [Character Glyph Model](#) paper for a more details.

Note that in the above "*in a sensible manner*" does not necessarily mean "*be able to render in its native form*". Transliteration (or even transcription) represents another possibility. For example, if some Japanese text was sent to a person that could not read Japanese, the display system could conceivably map the Japanese text into something like the following:

```
Nihongo, tokuni Kanji, wa totemo muzukashii desu.
```

possibly with some extra text indicating that this is Japanese. A rather interesting system for performing conversions is the [C3](#) system being developed in Europe. Another possibility is machine translation (which is becoming more viable year by year). While it is generally accepted that neither automatic transliteration, nor machine translation will ever be *perfect*, the author has seen systems on PC's that can generate quite understandable output.

Besides text, things like dates, money, weights and measures, and other such data should also be displayed in the end-user's native format by default. There should be a way to override such formatting if desired.

1.4 Data Input Requirements

As with display, the user should be able to input data in the manner he is accustomed to. Again, things like dates, weights and measures, and money should be represented in their native format, and text rendering should obey the rules of the language it is written in.

It should be noted that there are many different keyboard types, and many different keyboarding techniques. For example, in Japan, it is quite common for a user to go through a QWERTY (Romanji) to Hiragana to Kanji conversion process for each phrase typed (and often for single characters as well). In addition, cursor movements, and text flow, are affected by the directionality of the language being input. A truly multilingual application that requires input should take all such things into consideration.

1.5 Summary Of Requirements

A multilingual application must be able to:

1. Support the character sets and encodings used to represent the information being manipulated.
2. Present the data meaningfully if the application is required to display the data.
3. Manipulate multilingual data internally.
4. Allow multilingual input where required

Each of these entails quite a lot of detail, and a number of issues to be resolved, but as the [Plan 9](#) architects have proved, careful thought, well applied, can result in truly multilingual systems.

2. The WWW As A Multilingual Application

While most people view the WWW as a large collection of data, servers, and browsers, the author prefers to look at it as a single large application. Each component of the WWW is really nothing more than an object in the most dynamic, and fastest growing, distributed application on the Internet today. Just as all components within a given application can communicate with one another, the authors' overriding goal for a multilingual WWW can be stated as:

From an end-user perspective, no matter where a link leads, the browser will be able to cope intelligently with the data received. From a system viewpoint, all clients and servers should be able to at least communicate.

By looking at the WWW as a single application, it is quite obvious that the needs of the WWW are almost exactly the same as those in the previous section. In the following, we will discuss each category of requirements as it applies to the WWW, and present unresolved issues facing the WWW today.

2.1 Data Representation Requirements for the WWW

One of the most common mistakes in discussions about the WWW is to load the term "World Wide Web" with the meaning "HTTP and HTML". This is, in fact, very far from the truth. Not only are many different protocols supported, but many different data formats are supported as well (postscript, images, plain text, audio, etc). As such, "objects" within the WWW (servers, browsers, etc.), can communicate in a wide variety of ways.

While the above is certainly true, this document concerns itself primarily with HTTP and HTML, simply because they are the core (for the moment anyway), and represent a tractible problem set. Some parts of this discussion could possibly be applied to other protocols and data formats (such as plain text files retrieved via FTP), but that is left to the discretion of the reader).

The basic requirements for multilingual data representation in the WWW can be informally listed as:

1. HTTP and HTML must support every character set and encoding that is registered with IANA for use within MIME bodies.
2. Where internationalization requires alterations to either the HTML or HTTP specifications, the effect on existing clients and servers should be reduced to a minimum.
3. ISO 8859–1 will be the only character set browsers will be *required* to process.
4. HTML must be a conforming SGML application.
5. As much as possible, HTTP message content should conform to the requirements for MIME bodies.
6. Encodings of character sets should be space efficient to avoid wasting bandwidth.

where the first and second items are probably of the greatest importance. There is some conflict in the above. See [Issues Facing Multilingual Support in the WWW](#) for a discussion of problems in the WWW.

2.2 Data Manipulation Requirements for the WWW

While the primary data manipulation required is HTML parsing, this is certainly not the *only* processing required. The following is a list of perceived data manipulation requirements for the WWW.

Transmittal

An HTML document must be transmitted via HTTP or some other protocol. This implies that the character set and encoding of the HTML document must be legal within the protocol

HTML Parsing

A browser must be able to parse HTML documents according to the rules set forth in the SGML standard. (ISO 8879:1986).

HTML Indexing

Somewhat related to the above, HTML documents, and in particular, the data contained in the <HEAD> element, need to be indexed to aid in searching.

Data conversion

Data needs to be converted to and from HTML for a variety of reasons. For example, [DynaWeb\(tm\)](#) converts from SGML to HTML, and there are a large number of LaTeX, RTF, Frame etc. filters. WWW support for multilingual data needs to make this as easy as possible.

Character Set conversion

It seems very likely that as more character sets and encodings are used within the WWW, some conversion between them will be needed.

2.3 Data Display Requirements for the WWW

A common misconception is that Mosaic or Netscape nee Mozilla (and other such GUI-based browsers), represent the only display technology we need to support. This is in fact very far from the truth. Other important browser classes are the TTY-based browsers, and those supporting people with sight disabilities. HTML must be made independent of a given presentation technology, or there is a risk of pages being created that cannot be accessed by everyone. Already this is apparent in ISMAP pages, and in tags that do not use the ALT attribute,

While the above is relatively independent of the multilingual issue, it is important to note that if a person with sight-disabilities visited a Japanese site, and that persons' browser had no Japanese support, the effect would be similar to browsing an EUC HTML page in a Latin1 browser: you would get garbage output at worst, and be prompted to save to a file at best.

2.3.1 HTML Presentation Requirements

Given the above, the basic requirement is that the browser be able to present multilingual HTML data to the end user. In GUI based browsers, this will usually boil down to a font-handling problem. For TTY based and speech synthesis systems, it will probably boil down to transcription, transliteration or translation. Note that for GUI based browsers, it is a simple extension to an existing mechanism, whereas other systems must become much more complicated.

2.3.2 Localization Requirements

In addition to the HTML presentation requirements, there are data display requirements for things like dates, time, weights, and other such things. There is a large amount of variation in the format and units used, and it seems very desirable to present them all to the end user in the manner that he is most accustomed to.

2.4 Data Input Requirements for the WWW

In a truly multilingual WWW, an end user should be able to input data into a form in the manner he is most accustomed to, using his language of choice. He should then be able to send the data to a server, any server, and expect the server to at least accept the data, even if it cannot necessarily process it. This also applies to URLs: it should be possible to specify an object on the network in a way that is natural for end users, using their natural language.

In addition, the date, time, unit etc. requirement applies equally well on the input side: the user should be able to input time, dates, weights, and other such things, in the manner he is most accustomed to, and in his language of choice.

3. Issues Facing Multilingual Support in the WWW

From reading the previous sections outlining the requirements for multilingual applications, and for a multilingual WWW, it should be (somewhat *painfully!*) obvious that the WWW falls quite short of the mark. Below the more important issues are outlined.

3.1 MIME Issues

One of the problems with representing multilingual documents in the WWW is that the character set parameter does not actually define a *character set*, but rather an *encoding* of a character set, as [Dan Conolly notes](#) in the HTML working group mailing list.

In addition, recent discussions regarding the MIME specification have stated that for the `text/*` data types, all line breaks must be indicated by a CRLF pair, which is defined as a pair of octets with the values 13 and 10 respectively. This implies that certain encodings cannot be used within the `text/*` data types if the WWW is to be strictly MIME conformant.

Please note that this issue has already been resolved (or at least for HTTP). See [Solution to the MIME problems](#) for more details.

3.2 SGML Issues

SGML does *not* define the representation of characters. ISO 8879:1986 defines a code set as a set of bit combinations of equal size, a character as an atom of information, a character repertoire as a group of characters, and a character set as a mapping of a character repertoire onto a code set such that each character is represented by a unique bit combination within the code set. An SGML parser deals with *characters*, as defined above, and hence, is logically independent of the physical representation of the data. There is often an *internal* representation of characters that could be quite different to that used in data storage.

Below are outlined some of the problems encountered when processing multilingual SGML data.

Case Sensitivity and Unification Issues

It is quite common for Asian character sets have different character codes representing the character. In addition, there are half-width (8 bit) *hankaku* and full-width (16 bit) *zenkaku* in Japan, that represent the same character at different sizes. For the most part, one wants to *unite* these in markup, but not in data, and for some languages, case is simply meaningless. It seems impossible to meet all possible requirements using the standard SGML mechanisms.

Lexical Equivalence of Character Combinations

Some languages such as Thai cannot be handled directly by SGML, even if the SGML declaration is altered (though, it is possible to perform some application-specific normalisation within, or before the entity manager).

The problem is that there is no way to specify the lexical equivalence of character combinations (though there is a rather English-centered NAMECASE for specifying the case sensitivity of certain matches). In Thai, and other languages, one can have combinations of characters, the order of which is not defined. For example, in Thai, a [*consonant+vowel+tone mark*] is lexically equivalent to a [*consonant+tone mark+vowel*] combination made up of the same three characters.

Large Character Sets

In many Asian countries, the character sets include many thousands of characters, even in the minimal standard sets (which generally consist of between 2,500 and 16,000 characters). This implies a significant overhead in categorization table size etc.

Variations in Character Sets

In Japan at least, it is relatively common for there to be some slight variation between character sets used on different machines. For example, one systems' SJIS might not be exactly the same as others.

Multibyte Character Encodings

An SGML parser *requires* that all bit combinations representing character codes be the same size, or in the colloquial, fixed-width characters. SGML does not limit the number of bits however, so fixed-width multibyte encodings like UCS-2 present no problems whatsoever.

While SGML offers some support for shifting in and out of character sets and other such things, it tends to be somewhat error prone, and certainly non-intuitive when actually used. For example, when dealing with multiple character sets, as ISO 2022 allows you to do, there must be some specific entry into, and exit from the various character sets in order for the markup to be correctly recognised. (See Appendix E.3 of The SGML Handbook for some discussion of ISO 2022 handling).

Numeric Character References

Numeric character references can often produce surprising results. The reason is simply that numeric character references are resolved in the document character set, so if the document character set is translated in some way, the numeric character references no longer reference the character desired.

The SGML processing model is described more fully in [The SGML Processing Model](#).

3.3 Parsing Issues Facing Browsers

In theory, an HTML parser should parse HTML in full accordance with parsing rules of SGML. This is actually not the case: most parsers understand just enough to be **Mosaic and Netscape Compatible(tm)**, and indeed, they are often *bugward* compatible.

Irrespective of the above, HTML parsers face most of the same problems outlined in the previous section. Also, HTML parsers are required to be very fast, which leads to optimisations such as hard-coding tag recognition, complicating the multilingual data issue. Another constraint is that HTML parsers are often directly connected to the display subsystem, further complicating the parsing model.

When one combines the above, and the fact that there is such a large (and *growing!*) number of character sets and encodings, it seems that most parsers will be limited to a small subset of all extant, and all possible character sets and encodings (in the above, *parser* has a slightly different meaning to the SGML meaning).

3.4 Browser Presentation Issues

The vast majority of browsers in the WWW today have no support at all for multilingual data presentation. Besides the obvious font handling issues, there are additional problem with directionality. Very few browsers do anything at all with regard to either of these these two issues. [Mosaic 110n](#) is an exception to the rule but has numerous other failings.

In addition to the obvious problems, the WWW browsers available today are uniformly incomplete in their support of input and display methods for dates, time, weight, and other such things, in a manner that seems natural to the end user. Even localised products fail here. The primary reason is that there is no strictly defined way of representing such information in the WWW. Rather, it is left to the authors' discretion.

Finally, forms support is very much touch and go. If a user in Japan is sent a form from the USA, and he fills a text field with Japanese characters, then submits the form, the URL being sent to the server might look something like:

```
http://foo/bar?baz=%B5%6B%85%9A
```

depending on the particular browser. The problem actually arises only after the receiving program decodes the URL: it has no way of knowing what character set or encoding is in effect, and hence, might very well crash, or fail to recognise the text following the question mark, or indeed, the URL as a whole.

3.5 Server Issues

For servers, the problems generally fall into 2 categories: data received, and data sent.

The biggest problem with data received by a server, is that the vast majority of such data arrives via the forms mechanism. As was pointed out earlier, there is no standard way to specify the character set or encoding for

the data sent. This could lead to spectacular errors, and possibly, even outright server crashes.

On the output side, the biggest problem is what to do when a client requests data in a character set or encoding that the server does not understand. The HTTP specification has this to say:

```
If a resource is only available in a character set other than the
defaults, and that character set is not listed in the Accept-
Charset field, it is only acceptable for the server to send the
entity if the character set can be identified by an appropriate
charset parameter on the media type or within the format of the
media type itself.
```

The wording is somewhat vague, In particular *appropriate charset parameter* could be read as either "one of the IANA-registered values" or as "a string", because the HTTP standard allows arbitrary tokens as values (See [4.1 MIME and HTTP Resolution](#) for the syntax).

It should be noted that few servers actually support the `charset` parameter on the MIME type for HTML. It also appears that once sites with documents using multiple character sets do appear, managing the character set information could potentially become an considerable burden.

4. Possible Solutions to the Issues

Now that the requirements, and the issues have been explained, it is time to turn to solving the problems discussed. Below are a series of possible solutions to each of the problems raised above.

4.1 MIME and HTTP Resolution

While the issue with MIME is, as yet, unresolved, the issue with MIME-like bodies in HTTP has been resolved. Basically, MIME conformance has been sacrificed in order to allow data to be transported via HTTP in the form most natural. For example, HTTP allows the message bodies to be encoded using UCS-2, whereas MIME does not. The latest HTTP specification can be found at <http://www.ics.uci.edu/pub/ietf/http/>, but the relevant fragment is quoted below:

```
HTTP redefines the canonical form of text media to allow multiple
octet sequences to indicate a text line break. In addition to the
preferred form of CRLF, HTTP applications must accept a bare CR or
LF alone as representing a single line break in text media.
Furthermore, if the text media is represented in a character set
which does not use octets 13 and 10 for CR and LF respectively (as
is the case for some multi-byte character sets), HTTP allows the
use of whatever octet sequence(s) is defined by that character set
to represent the equivalent of CRLF, bare CR, and bare LF. It is
assumed that any recipient capable of using such a character set
will know the appropriate octet sequence for representing line
breaks within that character set.
```

many thanks to Roy Fielding for his bravery in taking this stance. Legal values for the `charset` parameter of the MIME HTML type are:

```
charset = "US-ASCII"
          | "ISO-8859-1" | "ISO-8859-2" | "ISO-8859-3"
```

"ISO-8859-4"	"ISO-8859-5"	"ISO-8859-6"
"ISO-8859-7"	"ISO-8859-8"	"ISO-8859-9"
"ISO-2022-JP"	"ISO-2022-JP-2"	"ISO-2022-KR"
"UNICODE-1-1"	"UNICODE-1-1-UTF-7"	"UNICODE-1-1-UTF-8"
token		

4.2 Solutions to SGML's Problems

In this section, some possible solutions to SGMLs' problems with handling multilingual data are examined.

4.2.1 Multibyte Character Encodings

An SGML parser *requires* that all bit combinations representing character codes be the same size, or in the colloquial, fixed-width characters. SGML does not limit the number of bits however, so fixed-width multibyte encodings like UCS-2 present no problems whatsoever.

It has been widely accepted that the SGML model for anything other than such encodings, is somewhat unworkable. Rather, the generally agreed upon technique (recommended by SGML WG8), is to map to a fixed-width representation in, or below, the entity manager. This model is implemented in James Clarks' [sp](#) SGML parser (see [The SGML Processing Model](#) for an overview of the SGML processing model).

4.2.2 Large Character Sets

Given fixed-width characters, we are then faced with the problem of manipulating them. As noted, an SGML parser works only on the abstract characters, so the parser per se has no problems at all. Rather, the implementation needs to make sure the data path (number of bits making up a character) is wide enough, and the implementation also needs to be able to classify each character.

The classification, when done in a brute force manner, requires 2 bytes per character (1 bit for each character class, and a few extra bits for padding to a byte boundary). So if we have a character set containing 65536 characters, we'd need tables 131072 bytes long to store the classification data. Given modern operating system designs, this is not a great overhead, but there are ways of reducing this considerably. For example, in large character sets, large ranges of characters belong to the same character classes, so table sizes can be reduced considerably at the cost of a conditional.

4.2.3 Case Sensitivity and Unification Issues

As stated before, the whole issue of case has an entirely different meaning in a multilingual world. While it is certainly possible to provide some mappings in the SGML declaration, it is probably impossible to handle all cases without modifying SGML somewhat.

4.2.4 Lexical Equivalence of Character Combinations

It is possible to handle languages in which multiple character combinations have lexical equivalence in the normaliser as one would shifted encodings, or to simply require that a canonical ordering, such as that defined in the Unicode standard, be used. These are roughly equivalent in the sense that some normalisation is taking place at some point. Solving this inside SGML would require changes to the standard, or multiple definitions of what would appear on the screen to be identical markup.

4.2.5 Variations in character sets

SGML does not allow a single bit pattern to represent more than one character. Provided the variations do not violate this rule, and that the variations occur within the range of values making up the character set, the SGML parser will treat variations "correctly".

4.2.6 Numeric Character Entity References

Numeric character references are evaluated based on the document character set. For numeric character references to remain invariant across encodings, the document character set must either be invariant, or be a strict superset (ie. the characters that appear in the first document character set must appear in exactly the same place in the second). For example, `@` has the same meaning in ISO 646 and ISO 8859-1, but `℣` does not have the same meaning in JIS X 0208 and ISO 10646. See [The SGML Processing Model](#) for a general overview of the SGML processing model.

The best work the author has seen regarding these issues is the [Extended Reference Concrete Syntax](#) by Rick Jelliffe of Allette Systems Australia. In that proposal, the author discusses many of these issues, and offers a fairly comprehensive solution. This seminal work appears to be gathering support, and is currently being offered to SGML Open for adoption.

4.3 Parsing

Provided that all encodings are mapped onto a fixed-width representation in, or before the entity manager, there should be few parsing problems. Full SGML parsing capability, while ultimately very desirable, is not necessary in the WWW today.

On the other hand, requiring the browser to map all character encodings onto a fixed width representation raises its own problems. There are a large number of character sets and encodings, and indeed, the set is really infinite. In an earlier section, a list of requirements was outlined, and among those requirements, was the desire to be able to handle data, no matter where it came from. If we want the browser to handle translation, that means that each browser needs to be able to translate a potentially infinite set of character sets and encodings to a fixed-width representation. There are 4 ways to deal with this:

1. Have all the translation code built into the browser. This is obviously close to impossible, especially in the face of new character sets and encodings, which are still appearing, and the fact that character sets and encodings change over time.
2. Have a VM built into the client such that the client can download a translation program (if it doesn't have one locally) to perform the translation. This implies that the tables will be widely replicated, leading to a possibly significant overhead.
3. Use Larry Masinters' conversion server idea ([Conversion Servers](#)).
4. Simply require all data to be in a common character set and encoding.

Item number 4 above is simply recognising that some mapping is required, and moving the responsibility for performing such mappings out of the client.

Quite obviously, if we need to convert to a fixed-width internal form, it makes sense to use the *same* internal format which raises the question of *which* fixed-width representation.

4.4 Browsers

Browser localisation is somewhat beyond the scope of this paper, as it is necessarily product-specific. Suffice it to say that things like hard-coding resource strings, and use of 8 bit wide characters, is not recommended.

The problem with representing units of measurement and whatnot will need to be addressed eventually. For input, one can extend the legal values of the TYPE attribute of <INPUT> to cover the desired input types. For output (ie. simple display), tags could be added to the HTML specification.

In order to cleanly handle multilingual data in forms, it is necessary to send the form data as `multipart/form-data` (defined in [Larry Masinters' file upload proposal](#)) or something much like it.

4.5 Servers

As mentioned above, the best way to solve the problems with data being sent to the server is to simply use (and perhaps later, require use of), the `multipart/form-data` type. This requires some client side changes.

In order to behave correctly in the face of the `Accept-Charset` parameter in HTTP specification, servers should know the character set and encoding of the documents they serve. For many sites, there will be a single character set and encoding used for all documents (such as ASCII, or Latin-1), but for other sites, there might be a multitude of character sets and encodings. In order to maintain such information (and information regarding security etc.), it seems that many servers will implement some method of attaching attributes to the data files.

Given that a server does know the character sets and encodings of it's documents, when a request comes in from a client specifying something entirely different (for example, the data is in EUC, and the request is for Latin-1), the question of correct behaviour arises.

One obvious solution would be for all clients and servers to share a lingua franca, which was suggested by the author in December of last year (see [The HyperMail HTTP-WG Archives](#)). In such a case, the above scenario never arises, because the client and server always have some character set and encodings in common.

Another solution is to send a redirect to the client telling it to visit a [conversion server](#) which will translate to the desired character set and encoding.

Finally, it is also possible for the server to perform the conversion on the fly using something like the VM architecture defined in [A Simple VM for use in Conversion](#).

5. Recommendations

With the background and explanations above, we now turn to specific recommendations that would allow multilingual data processing in the WWW. Backward compatibility has been kept in mind.

5.1 Changes to the SGML Declaration

The MIME charset parameter tells us how to map from the byte stream to characters (the data storage character set). The question then is how to map the data storage character set to a document character set the SGML parser can use. We have 2 choices:

1. Map the stream of characters in the data storage character set to a single document character set.
2. Assume that the data storage character set *is* the document character set, and derive a suitable SGML declaration.

The major stumbling block to the second is that numeric character entities still present a problem, because there will not be a common document character set. Hence, if the document was translated to a different data storage character set and encoding, and retransmitted, the derived SGML declaration (document character set) would differ, thereby (possibly) rendering numeric character entities invalid, or nonsensical, unless numeric character references are also translated.

It is therefore recommended that the HTML Working Group adopt an SGML declaration defining a single document character set that all data storage character sets can be mapped into. This means that the document character set must contain all the characters that data storage character sets may contain. A prime candidate for this is [-//SGML Open:1995//SYNTAX Extended \(ISO 8859-1 repertoire\)](#), or something very much like it. This basically uses ISO 10646 for the document character set, and allows only ISO 8859-1 in markup.

By requiring that the document character set (not the data storage character set) be ISO 10646 we are requiring that numeric character references be evaluated in terms of it. Hence, if we have a document in ISO 8859-1, with a numeric character reference "ß", then it has the meaning "the character represented by 223 in the ISO 10646 code set", *not* "the character represented by 223 in the ISO-8859-1 code set". In other words, numeric character references will be invariant across data storage character sets and encodings.

Given the [Unicode Consortium Mapping Tables](#), it is trivial to create (logical, or actual) translators to Unicode.

While it is very desirable, it seems that requiring actual Unicode support in client-side parsers is not feasible. As such, when a client receives an HTML document, it should be required to perform an *implicit* conversion only, and so long as the ESIS of the parsed document is exactly the same as that of the document parsed using the canonical model (where character equality is at the purely abstract character level), the parser should be regarded as conforming. In other words, current ISO 8859-1 (and other) parsers only need numeric character reference handling changed to be 100% conformant.

It should be noted that in addition to the data storage character set, and the document character set, SGML has the notion of *system character set*, or in other words, the character set the SGML system uses. Mapping the document character set to the system character set is allowed, though problems arise when the system character set does not contain all the characters found in the document character set. This problem is not solved in SGML.

Given the above, we have a firm basis for supporting multilingual documents, and a firm definition of conformance at the parser level. In addition, we also have a clean (if somewhat unorthodox) way of handling numeric character entities.

5.2 Changes to HTML

It is recommended that the following elements, or something much like them, be added to HTML in the near future.

<LOCALE>

This should occur only in the <HEAD> element, and have the following model:

```
<!ELEMENT locale - O>
<!ATTLIST locale
    LOCALE #CDATA (...) #REQUIRED>
```

<DATE>

This represents a date. The date format should be that defined in RFC 1123.

<TIME>

This represents a time in HH:MM:SS format.

<WEIGHT>

This represents a weight. The content should be a weight specified in metric using the standard abbreviations as a suffix.

<NUMBER>

The content of this element should be a single numeric quantity, with no punctuation. Formatting would be decided upon by the display system.

<CURRENCY>

The content of this element could be a number containing no punctuation.

It is further recommended that the value of the `inputType` parameter entity on <INPUT> be changed to:

```
<!ENTITY % InputType "(TEXT | PASSWORD | CHECKBOX |
    RADIO | SUBMIT | RESET |
    IMAGE | HIDDEN | DATE |
    TIME | WEIGHT | NUMBER | CURRENCY )">
```

In addition to the LOCALE element, each of the recommended tags should have a LOCALE attribute to allow the global locale to be overridden as necessary. The <INPUT> tag should also have this added.

When no locale information has been specified in the document, the default should be the locale of the client receiving the document. The locale names could possibly be the values used in the LANG attribute of HTML 3.0.

5.3 Changes to Clients

Ideally, all clients should be truly multilingual, Unicode-based systems. However, this is not likely to eventuate in the near future, so instead, groups of goals are itemised. Only the first group is required in the short term.

Near future (within 6 months to a year)

1. All clients should send an `Accept-Charset` parameter if the preferred character set is anything other than ISO-8859-1.
2. All clients should be altered to evaluate numeric character references in terms of Unicode (the system characters set).
3. Clients should send form data via message bodies rather than as name/value pairs attached to the end of a URL. Larry Masinters' `multipart/form-data` might be the best content

type.

4. Clients should accept the new tags, and especially, the new INPUT element types.

Future

1. Clients should be able to parse Unicode in either UTF-8 or UCS-2 format.
2. Clients should alter display and input characteristics based on the LOCALE effects.
3. Display via fonts, translation etc.

5.4 Changes to Servers

The ideal would be for servers to be capable of providing multilingual data in any of the formats the client requests, though this, like requiring the same of clients, is probably unreasonable. Instead, the author believes that ideally, clients and servers should both be able to process Unicode, thereby providing a *lingua franca*. Servers should only need to convert to Unicode in either UTF-8, or UCS-2.

The above has been met with strong objections, so as with client changes, server changes are organised by priority, or time frame.

Near future (*within 6 months*)

1. Servers should correctly label the content type of the data sent. In particular, the charset parameter must be set for data in an encoding other than US-ASCII and ISO-8859-1
2. Servers should be altered to accept the multipart/form-data data type.

Future

1. Servers should be able to translate into Unicode.
2. Conversion servers should be set up.
3. Glyph image/font servers should be set up to allow full Unicode display on systems without the necessary fonts.

6. Discussion

Appendices

The following are a series of informational appendices.

A-1. The SGML Processing Model

* This section is under construction. *

A-2. Conversion Servers

Larry Masinter had a very good short-term solution to the problem of document exchange. In this basic model, not all servers can perform the required translation from the data storage character set and encoding of the document, to the character set and encoding desired by the client.

Rather, when a server get's such a request, it sends a redirect to the client, telling it to connect to a a proxy server which *can* perform the translation. This is obviously somewhat inefficient in terms of network use, but has a redeeming value of not requiring all servers to perform character set translations. It seems quite likely that, at least initially, there will not be a great demand for character set translation, so this service should suffice as an intermediary step.

A-3. Implementing Unicode

```
*****
* This section is under construction. *
*****
```

A-4. Problems With Unicode

```
*****
* This section is under construction. *
*****
```

A-5. Problems Facing ERCS

```
*****
* This section is under construction. *
*****
```

A-6. A Simple VM for use in Conversion

```
*****
* This section is under construction. *
*****
```

Glossary Of Terms

Some parts of this glossary have been directly quoted from The SGML Handbook, or from file called i18ngloss.txt.

Alphabet

A set of symbols which can be used to represent the phonetics of a language in a written form. The correspondence between sounds in the language, and symbols, is often inexact.

Bit combination

A series of binary digits.

Character

In SGML terms, an atom of information with an individual meaning within a character repertoire. Characters are further classified into [graphic characters and control characters](#). [Characters are usually part of an alphabet](#).

Character Code

AN integer which uniquely identifies a character within a character repertoire.

Character Repertoire

A set of characters used together. Meanings are defined for each character, and possibly for [control](#)

[sequences](#)

Character Set/Coded Character Set

A mapping of a character repertoire onto a [code set](#), such that each character from the character repertoire is represented by a number (bit combination) in the code set.

Character Set Encoding

A mapping from a sequence of bits to a sequence of characters from a character repertoire. An intermediary step is usually that of converting the bit sequence into a sequence of character codes, which are then used to identify the characters.

Code Set

A set of unique integers. SGML defines this as a set of [bit combinations of equal size...](#)

Control Character

A character that has some processing semantics attached which affects the processing of character following.

Control Sequence

A sequence of characters that begins with a control character, and which affects the processing of characters following. An example is an escape sequence.

Graphics Character

A character that is not a control character. Graphic characters have one or more associated [glyphs](#)

Glyph

An abstract form which represents one or more [glyph images](#), and which is used to visually depict encoded character data.

Glyph Image

The actual, concrete image of a glyph representation having been rasterized or otherwise imaged onto some display surface.

MIME

Multipurpose Internet Mail Extensions: a Internet standard for mail message content specified in [rfc1521](#) and [rfc1522](#).

SGML

The Standard Generalised Markup Language, and ISO standard (ISO 8879:1986) used for document processing.

Wide Character

A representation of a character inside a computer system consisting of a bit combination of 16 or more bits that are used to hold a character code

Bibliography

The Plan 9 Papers

<ftp://netlib.att.com/>

East Asian Character Set Issues:

A Proposal For An Extended Reference Concrete Syntax

Rick Jelliffe

ricko@allette.com.au

Allette Systems

Sydney, Australia

The SGML Handbook

Oxford University Press

Written by Charles Goldfarb
ISBN 0-19-853737-9

The Unicode Standard, Version 1.1

Version 1.0, Volume 1, ISBN 0-201-56788-1
Version 1.0, Volume 2, ISBN 0-201-60845-6

Using Unicode with MIME

D. Goldsmith

<http://ds.internic.net/rfc/rfc1641>

UTF-7: A Mail Safe Transformation Format of Unicode

D. Goldsmith and M. Davis

<http://ds.internic.net/rfc/rfc1642>

MIME (Multipurpose Internet Mail Extensions) Part 1

N. Borenstein and N. Freed

<http://ds.internic.net/rfc/rfc1521.ps>

MIME (Multipurpose Internet Mail Extensions) Part 2

K. Moore

<http://ds.internic.net/rfc/rfc1522.txt>

Hypertext Transfer Protocol -- HTTP/1.0

T. Berners-Lee, R. T. Fielding, H. Frystyk Nielsen