Character Model for the World Wide Web 1.0: Fundamentals

W3C Recommendation 15 February 2005

This version:
http://www.w3.org/TR/2005/REC-charmod-20050215/
Latest version:
http://www.w3.org/TR/charmod/
Previous version:
http://www.w3.org/TR/2004/PR-charmod-20041122/
Editors:
Martin J. Dürst, W3C <duerst@w3.org>
François Yergeau (Invited Expert)
Richard Ishida, W3C <ishida@w3.org>
Misha Wolf (until Dec 2002), Reuters Ltd. <misha.wolf@reuters.com>
Tex Texin (Invited Expert), XenCraft <tex@XenCraft.com>

Please refer to the **errata** for this document, which may include some normative corrections.

See also **translations**.

Abstract

This Architectural Specification provides authors of specifications, software developers, and content developers with a common reference for interoperable text manipulation on the World Wide Web, building on the Universal Character Set, defined jointly by the Unicode Standard and ISO/IEC 10646. Topics addressed include use of the terms 'character', 'encoding' and 'string', a reference processing model, choice and identification of character encodings, character escaping, and string indexing.

For normalization and string identity matching, see the companion document *Character Model for the World Wide Web 1.0: Normalization* [CharNorm]. For resource identifiers, see the companion document *Character Model for the World Wide Web 1.0: Resource Identifiers* [CharIRI].

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document contains the *Character Model for the World Wide Web 1.0: Fundamentals* specification, and is a W3C Recommendation. It has been reviewed by W3C Members and other interested parties and has been endorsed by the Director. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document was developed as part of the W3C Internationalization Activity by the W3C Internationalization Core Working Group, with the help of the Internationalization Interest Group.

If you have comments on this document, send them to www-i18n-comments@w3.org (public archive). Last Call dispositions are available in a public version and a Members-only version. There is also an implementation report. Changes to this document since the Proposed Recommendation version are detailed in **E Changes since the Proposed Recommendation**.

This document was produced under the 24 January 2002 CPP as amended by the W3C Patent Policy Transition Procedure. The Working Group maintains a public list of patent disclosures relevant to this document; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) with respect to this specification should disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

Appendices

---

1 Introduction

1.1 Goals and Scope

The goal of the Character Model for the World Wide Web is to facilitate use of the Web by all people, regardless of their language, script, writing system, and cultural conventions, in accordance with the *W3C goal of universal access*. One basic prerequisite to achieve this goal is to be able to transmit and process the characters used around the world in a well-defined and well-understood way.

The main target audience of this specification is W3C specification developers. This specification and parts of it can be referenced from other W3C specifications. It defines conformance criteria for W3C specifications as well as other specifications.

Other audiences of this specification include software developers, content developers, and authors of specifications outside the W3C. Software developers and content developers implement and use W3C specifications. This specification defines some conformance criteria for implementations (software) and content that implement and use W3C specifications. It also helps software developers and content developers to understand the character-related provisions in W3C specifications.

The character model described in this specification provides authors of specifications, software developers, and content developers with a common reference for consistent, interoperable text manipulation on the World Wide Web. Working together, these three groups can build a more international Web.

Topics addressed in this part of the Character Model for the World Wide Web include use of the terms 'character', 'encoding' and 'string', a reference processing model, choice and identification of character encodings, character escaping, and string indexing.

Other parts of the Character Model address normalization and string identity matching ([CharNorm]) and Internationalized Resource Identifiers (IRI) conventions ([CharIRI]).

Topics as yet not addressed or barely touched include fuzzy matching, and language tagging. Some of these topics may be addressed in a future version of this specification.

At the core of the model is the Universal Character Set (UCS), defined jointly by the Unicode Standard [Unicode] and ISO/IEC 10646 [ISO/IEC 10646]. In this document, **Unicode** is used as a synonym for the Universal Character Set. The model will allow Web documents authored in the world's scripts (and on different platforms) to be exchanged, read, and searched by Web users around the world.

1.2 Background

This section provides some historical background on the topics addressed in this specification.

Starting with *Internationalization of the Hypertext Markup Language* [RFC 2070], the Web community has recognized the need for a character model for the World Wide Web. The first step towards building this model was the adoption of Unicode as the document character set for HTML.

The choice of Unicode was motivated by the fact that Unicode:

- is the only universal character repertoire available,
- provides a way of referencing characters independent of the encoding of the text,
- is being updated/completed carefully,
- is widely accepted and implemented by industry.

W3C adopted Unicode as the document character set for HTML in [HTML 4.0]. The same approach was later used for specifications such as XML 1.0 [XML 1.0] and CSS2 [CSS2]. W3C specifications and applications now use Unicode as the common reference character set.

When data transfer on the Web remained mostly unidirectional (from server to

browser), and where the main purpose was to render documents, the use of Unicode without specifying additional details was sufficient. However, the Web has grown:

- Data transfers among servers, proxies, and clients, in all directions, have increased.
- Characters outside the US-ASCII [ISO/IEC 646][MIME-charset] repertoire are being used in more and more places.
- Data transfers between different protocol/format elements (such as element/attribute names, URI components, and textual content) have increased.
- More and more APIs are defined, not just protocols and formats.

In short, the Web may be seen as a single, very large application (see [Nicol]), rather than as a collection of small independent applications.

While these developments strengthen the requirement that Unicode be the basis of a character model for the Web, they also create the need for additional specifications on the application of Unicode to the Web. Some aspects of Unicode that require additional specification for the Web include:

- Choice of Unicode encoding forms (UTF-8, UTF-16, UTF-32).
- Counting characters, measuring string length in the presence of variable-length character encodings and combining characters.
- Duplicate encodings of characters (e.g. precomposed vs decomposed).
- Use of control codes for various purposes (e.g. bidirectionality control, symmetric swapping, etc.).

It should be noted that such aspects also exist in various encodings, and in many cases have been inherited by Unicode in one way or another from these encodings.

The remainder of this specification presents additional requirements to ensure an interoperable character model for the Web, taking into account earlier work (from W3C, ISO and IETF).

The first few chapters of the Unicode Standard [Unicode] provide very useful background reading. The policies adopted by the IETF for on the use of character sets on the Internet are documented in [RFC 2277].

1.3 Terminology and Notation

Unicode code points are denoted as U+hhhh, where "hhhh" is a sequence of at least four, and at most six hexadecimal digits.

Text has been used for examples to allow them to be cut and pasted by the reader. Characters used will not appear as intended unless you have the appropriate font, but care has been taken to annotate the examples so that they remain understandable even if you do not. In some cases it is important to see

the result of an example, so images have been used; by clicking on the image it is possible to link to the text for these examples in **C Example text**.

2 Conformance

This section explains the conditions that specifications, software, and Web content have to fulfill to be able to claim conformance to this specification.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC 2119].

> **NOTE:** RFC 2119 makes it clear that requirements that use SHOULD are not optional and must be complied with unless there are specific reasons not to: "*This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.*"

This specification defines conformance criteria for specifications, for software, and for Web content. To aid the reader, all conformance criteria are preceded by '[X]' where 'X' is one of 'S' for specifications, 'I' for software implementations, and 'C' for Web content. These markers indicate the relevance of the conformance criteria and allow the reader to quickly locate relevant conformance criteria by searching through this document.

A specification conforms to this document if it:

1. does not violate any conformance criteria preceded by [S],
2. documents the reason for any deviation from criteria where the imperative is SHOULD, SHOULD NOT, or RECOMMENDED,
3. where applicable, requires implementations conforming to the specification to conform to this document,
4. where applicable, requires content conforming to the specification to conform to this document.

An implementation (software) conforms to this document if it does not violate any conformance criteria preceded by [I].

Content conforms to this document if it does not violate any conformance criteria preceded by [C].

> **NOTE:** Requirements placed on specifications might indirectly cause requirements to be placed on implementations or content that claim to conform to those specifications. Likewise, requirements placed on content may affect implementations designed to produce such content, and so on.

Where this specification places requirements on processing, it is to be understood as a way to specify the desired external behavior. Implementations

can use other means of achieving the same results, as long as observable behavior is not affected.

3 Perceptions of Characters

3.1 Introduction

The glossary entry in the Unicode Standard [Unicode 4.0] gives:

"*Character. (1) The smallest component of written language that has semantic value; refers to the abstract meaning and/or shape ...*"

The word 'character' is used in many contexts, with different meanings. Human cultures have radically differing writing systems, leading to radically differing concepts of a character. Such wide variation in end user experience can, and often does, result in misunderstanding. This variation is sometimes mistakenly seen as the consequence of imperfect technology. Instead, it derives from the great flexibility and creativity of the human mind and the long tradition of writing as an important part of the human cultural heritage. The alphabetic approach used by scripts such as Latin, Cyrillic and Greek is only one of several possibilities.

> **EXAMPLE:** A character in Japanese hiragana and katakana scripts corresponds to a syllable (usually a combination of consonant plus vowel).
>
> **EXAMPLE:** Korean Hangul combines symbols for individual sounds of the language into square blocks, each of which represents a syllable. Depending on the user and the application, either the individual symbols or the syllabic clusters can be considered to be characters.
>
> **EXAMPLE:** In Indic scripts each consonant letter carries an inherent vowel that is eliminated or replaced using semi-regular or irregular ways to combine consonants and vowels into clusters. Depending on the user and the application, either individual consonants or vowels, or the consonant or consonant-vowel clusters can be perceived as characters.
>
> **EXAMPLE:** In Arabic and Hebrew vowel sounds are typically not written at all. When they are written they are indicated by the use of combining marks placed above and below the consonantal letters.

The developers of specifications, and the developers of software based on those specifications, are likely to be more familiar with usages of the term 'character' they have experienced and less familiar with the wide variety of usages in an international context. Furthermore, within a computing context, characters are often confused with related concepts, resulting in incomplete or inappropriate specifications and software.

This section examines some of these contexts, meanings and confusions.

3.2 Units of aural rendering

In some scripts, characters have a close relationship to phonemes (a **phoneme** is a minimally distinct sound in the context of a particular spoken language), while in others they are closely related to meanings. Even when characters (loosely) correspond to phonemes, this relationship may not be simple, and there is rarely a one-to-one correspondence between character and phoneme.

> **EXAMPLE:** In the English sentence, "*They were too close to the door to close it.*" the same character 's' is used to represent both /s/ and /z/ phonemes.
>
> **EXAMPLE:** In the English language the phoneme /k/ of "*cool*" is like the phoneme /k/ of "*keel*".
>
> **EXAMPLE:** In many scripts a single character may represent a sequence of phonemes, such as the syllabic characters of Japanese hiragana.
>
> **EXAMPLE:** In many writing systems a sequence of characters may represent a single phoneme, for example 'th' and 'ng' in "*thing*".

C001 [S] [I] [C] Specifications, software and content MUST NOT require or depend on a one-to-one correspondence between characters and the sounds of a language.

3.3 Units of visual rendering

Visual rendering introduces the notion of a *glyph*. **Glyphs** are defined by ISO/IEC 9541-1 [ISO/IEC 9541-1] as "*a recognizable abstract graphic symbol which is independent of a specific design*". There is *not* a one-to-one correspondence between characters and glyphs:

- A single character can be represented by multiple glyphs (each glyph is then part of the representation of that character). These glyphs may be physically separated from one another.
- A single glyph may represent a sequence of characters (this is the case with ligatures, among others).
- A character may be rendered with very different glyphs depending on the context.
- A single glyph may represent different characters (e.g. capital Latin A, capital Greek A and capital Cyrillic A).

A set of glyphs makes up a **font**. Glyphs can be construed as the basic units of organization of the visual rendering of text, just as characters are the basic unit of organization of encoded text.

C002 [S] [I] [C] Specifications, software and content MUST NOT require or depend on a one-to-one mapping between characters and units of displayed text.

See the appendix **B Examples of Characters, Keystrokes and Glyphs** for

examples of the complexities of character to glyph mapping.

3.3.1 Visual Rendering and Logical Order

Some scripts, in particular Arabic and Hebrew, are written from right to left. Text including characters from these scripts can run in both directions and is therefore called bidirectional text. The Unicode Standard [Unicode] requires that characters be stored and interchanged in **logical order**, i.e. roughly corresponding to the order in which text is typed in via the keyboard or spoken (for a more detailed definition see [Unicode 4.0], Section 2.2). Logical ordering is important to ensure interoperability of data, and also benefits accessibility, searching, and collation.

C003 [S] [I] [C] Protocols, data formats and APIs MUST store, interchange or process text data in logical order.

In the presence of bidirectional text, two possible selection modes can be considered. The first is **logical selection mode**, which selects all the characters *logically* located between the end-points of the user's mouse gesture. Here the user selects from between the first and second letters of the second word to the middle of the number. Logical selection looks like this:

Visual display	عدد مارس ١٩٩٨
Logical order	ع د د <space> م ا ر س <space> ١ ٩ ٩ ٨

Logical selection resulting in discontiguous visual ranges

It is a consequence of the bidirectionality of the text that a single, continuous logical selection in memory results in a *discontinuous selection appearing on the screen*. This discontinuity makes some users prefer a **visual selection mode**, which selects all the characters *visually* located between the end-points of the user's mouse gesture. With the same mouse gesture as before, we now obtain:

Visual display	عدد مارس ١٩٩٨
Logical order	ع د د <space> م ا ر س <space> ١ ٩ ٩ ٨

Visual selection resulting in discontiguous logical ranges

In visual selection mode, as seen in the example above, a single visual selection range may result in *two or more* logical ranges, which may have to be accommodated by protocols, APIs and implementations. Other, related aspects of a user interface for bidirectional text include caret movement, behavior of backspace/delete keys, and so on.

Currently, most implementations provide logical selection, while only very few

provide visual selection.

C075 [I] Independent of whether some implementation uses logical selection or visual selection, characters selected MUST be kept in logical order in storage.

C004 [S] Specifications of protocols and APIs that involve selection of ranges SHOULD provide for discontiguous logical selections, at least to the extent necessary to support implementation of visual selection on screen on top of those protocols and APIs.

3.4 Units of input

In keyboard input, it is *not* always the case that keystrokes and input characters correspond one-to-one. A limited number of keys can fit on a keyboard. Some keyboards will generate multiple characters from a single keypress. In other cases ('dead keys') a key will generate no characters, but affect the results of subsequent keypresses. Many writing systems have far too many characters to fit on a keyboard and must rely on more complex **input methods**, which transform keystroke sequences into character sequences. Other languages may make it necessary to input some characters with special modifier keys. See **B Examples of Characters, Keystrokes and Glyphs** for examples of non-trivial input.

C005 [S] [I] Specifications and software MUST NOT require nor depend on a single keystroke resulting in a single character, nor that a single character be input with a single keystroke (even with modifiers), nor that keyboards are the same all over the world.

3.5 Units of collation

String comparison as used in sorting and searching is based on units which do not in general have a one-to-one relationship to encoded characters. Such string comparison can aggregate a character sequence into a single **collation unit** with its own position in the sorting order, can separate a single character into multiple collation units, and can distinguish various aspects of a character (case, presence of diacritics, etc.) to be sorted separately (multi-level sorting).

In addition, a certain amount of pre-processing may also be required, and in some languages (such as Japanese and Arabic) sort order may be governed by higher order factors such as phonetics or word roots. Collation methods may also vary by application.

> **EXAMPLE:** In traditional Spanish sorting, the character sequences 'ch' and 'll' are treated as atomic collation units. Although Spanish sorting, and to some extent Spanish everyday use, treat 'ch' as a single unit, current digital encodings treat it as two characters, and keyboards do the same (the user types 'c', then 'h').
>
> **EXAMPLE:** In some languages, the letter 'æ' is sorted as two consecutive collation units: 'a' and 'e'.

EXAMPLE: The sorting of text written in a bicameral script (i.e. a script which has distinct upper and lower case letters) is usually required to ignore case differences in a first pass; case is then used to break ties in a later pass.

EXAMPLE: Treatment of accented letters in sorting is dependent on the script or language in question. The letter 'ö' is treated as a modified 'o' in French, but as a letter completely independent from 'o' (and sorting after 'z') in Swedish. In German certain applications treat the letter 'ö' as if it were the sequence 'oe'.

EXAMPLE: In Thai the sequence 'เก' (U+0E40 U+0E01) must be sorted as if it were written 'กเ' (U+0E01 U+0E40). Reordering is typically done during an initial pre-processing stage.

EXAMPLE: German dictionaries typically sort 'ä', 'ö' and 'ü' together with 'a', 'o' and 'u' respectively. On the other hand, German telephone books typically sort 'ä', 'ö' and 'ü' as if they were spelled 'ae', 'oe' and 'ue'. Here the application is affecting the collation algorithm used.

C006 [S] [I] Software that sorts or searches text for users SHOULD do so on the basis of appropriate collation units and ordering rules for the relevant language and/or application.

C007 [S] [I] Where searching or sorting is done dynamically, particularly in a multilingual environment, the 'relevant language' SHOULD be determined to be that of the current user, and may thus differ from user to user.

C066 [S] [I] Software that allows users to sort or search text SHOULD allow the user to select alternative rules for collation units and ordering.

C008 [S] [I] Specifications and implementations of sorting and searching algorithms SHOULD accommodate text that contains any character in Unicode.

Note that this requires, as a minimum, that a collation algorithm does not break down if the text contains Unicode characters that are not covered by its rules. It does not necessarily require full implementation of complex algorithms for all scripts. One useful way of satisfying the requirement is to apply a default collation algorithm that covers all Unicode characters.

ISO/IEC 14651 [ISO/IEC 14651] and Unicode Technical Report #10, the Unicode Collation Algorithm [UTR #10], describe a model for collation that accommodates most languages and provide a default collation order. They are appropriate references for collation and provide implementation guidelines. The default collation order can be used in conjunction with rules tailored for a particular locale to ensure a predictable ordering and comparison of strings, whatever characters they include.

3.6 Units of storage

Computer storage and communication rely on units of physical storage and information interchange, such as bits and bytes (8-bit units, also called octets). A frequent error in specifications and implementations is the equating of characters with units of physical storage. The mapping between characters and such units of storage is actually quite complex, and is discussed in the next section, **4.1 Character Encoding**.

C009 [S] [I] Specifications, software and content MUST NOT require or depend on a one-to-one relationship between characters and units of physical storage.

3.7 Summary

The term character is used differently in a variety of contexts and often leads to confusion when used outside of these contexts. In the context of the digital representations of text, a **character** can be defined as a small logical unit of text. **Text** is then defined as sequences of characters. While such an informal definition is sufficient to create or capture a common understanding in many cases, it is also sufficiently open to create misunderstandings as soon as details start to matter. In order to write effective specifications, protocol implementations, and software for end users, it is very important to understand that these misunderstandings can occur.

This section, **3 Perceptions of Characters**, has discussed terms for units that do not necessarily overlap with the term 'character', such as phoneme, glyph, and collation unit. The next section, **4.1 Character Encoding**, lists terms that should be used rather than 'character' to precisely define **units of encoding** (code point, code unit, and byte).

C010 [S] When specifications use the term 'character' the specifications MUST define which meaning they intend.

C067 [S] Specifications SHOULD use specific terms, when available, instead of the general term 'character'.

4 Digital Encoding of Characters

4.1 Character Encoding

On the WWW, as in any computing environment, characters must be encoded to be of any use. To achieve text encoding, a large variety of character encodings have been devised. Character encodings can loosely be explained as mappings between the character sequences that users manipulate and the sequences of bits that computers manipulate.

Given the complexity of text encoding and the large variety of mechanisms for character encoding invented throughout the computer age, a more formal description of the encoding process is useful. The process of defining a text encoding can be described as follows (see Unicode Technical Report #17: Character Encoding Model [UTR #17] for a more detailed description):

1. A set of characters to be encoded is identified. The characters are pragmatically chosen to express text and to efficiently allow various text processes in one or more target languages. They may not correspond precisely to what users perceive as letters and other characters. The set of characters is called a **repertoire**.
2. Each character in the repertoire is then associated with a (mathematical, abstract) non-negative integer, the **code point** (also known as a **character number** or **code position**). The result, a mapping from the repertoire to the set of non-negative integers, is called a **coded character set (CCS)**.
3. To enable use in computers, a suitable base datatype is identified (such as a byte, a 16-bit unit of storage or other) and a **character encoding form (CEF)** is used, which encodes the abstract integers of a coded character set (CCS) into sequences of the **code units** of the base datatype. The character encoding form can be extremely simple (for instance, one which encodes the integers of the CCS into the natural representation of integers of the chosen datatype of the computing platform) or arbitrarily complex (a variable number of code units, where the value of each unit is a non-trivial function of the encoded integer).
4. To enable transmission or storage using byte-oriented devices, a **serialization scheme** or **character encoding scheme (CES)** is next used. A character encoding scheme is a mapping of the code units of a character encoding form (CEF) into well-defined sequences of bytes, taking into account the necessary specification of byte-order for multi-byte base datatypes and including in some cases switching schemes between the code units of multiple character encoding schemes (an example is ISO 2022). A character encoding scheme, together with the coded character sets it is used with, is called a **character encoding**, and is identified by a unique identifier, such as an IANA charset identifier. Given a sequence of bytes representing text and a character encoding identified by a `charset` identifier, one can in principle unambiguously recover the sequence of characters of the text.

   NOTE: See **4.4.2 Character encoding identification** for a discussion of the term 'charset' and further details on character encodings.

   NOTE: The term 'character encoding' is somewhat ambiguous, as it is sometimes used to describe the actual process of encoding characters and sometimes to denote a particular way to perform that process (as in "*this file is in the X character encoding*"). Context normally allows the distinction of those uses, once one is aware of the ambiguity.

   NOTE: Given a sequence of characters, a given 'character encoding' may not always produce the same sequence of bytes. In particular for encodings based on ISO 2022, there may be choices available during the encoding process.

In very simple cases, the whole encoding process can be collapsed to a single step, a trivial one-to-one mapping from characters to bytes; this is the case, for instance, for US-ASCII [ISO/IEC 646] and ISO-8859-1.

Text is said to be in a **Unicode encoding form** if it is encoded in UTF-8, UTF-16 or UTF-32.

4.2 Transcoding

Transcoding is the process of converting text from one character encoding to another. Transcoders work only at the level of character encoding and do not parse the text; consequently, they do not deal with character escapes such as numeric character references (see **4.6 Character Escaping**) and do not adjust embedded character encoding information (for instance in an XML declaration or in an HTML `meta` element).

NOTE: Transcoding may involve one-to-one, many-to-one, one-to-many or many-to-many mappings. In addition, the storage order of characters varies between encodings: some, such as the Unicode encoding forms, prescribe logical ordering, while others use visual ordering; among encodings that have separate diacritics, some prescribe that they be placed before the base character, some after. Because of these differences in sequencing characters, transcoding may involve reordering: thus XYZ may map to yxz.

EXAMPLE: This first example shows the transcoding of the Russian word 'Русский' meaning 'Russian' (language), from the UTF-16 encoding of Unicode to the ISO 8859-5 encoding:

UTF-16		ISO 8859-5	
Code unit	**Char. name (abbreviated)**	**Code unit**	**Char. name (abbreviated)**
0420	CAPITAL ER	C0	CAPITAL ER
0443	SMALL U	E3	SMALL U
0441	SMALL ES	E1	SMALL ES
0441	SMALL ES	E1	SMALL ES
043A	SMALL KA	DA	SMALL KA
0438	SMALL I	D8	SMALL I
0439	SMALL SHORT I	D9	SMALL SHORT I

EXAMPLE: This second example shows a much more complex case, where the Arabic word 'السلام', meaning 'peace', is transcoded from the visually-ordered, contextualized encoding IBM CP864 to the UTF-16 encoding of Unicode:

IBM CP864		UTF-16	
Code	**Char. name**	**Code**	**Char. name**

unit	(abbreviated)	unit	(abbreviated)
EF	FINAL MEEM	0627	ALEF
9E	MEDIAN LAM-ALEF	0644	LAM
D3	MEDIAN SEEN	0633	SEEN
E4	MEDIAN LAM	0644	LAM
C7	INITIAL ALEF	0627	ALEF
		0645	MEEM

Notice that the order of the characters has been reversed, that the single LAM-ALEF in CP864 has been converted to a LAM ALEF sequence in UTF-16, and that the contextual variants (initial, median or final) in the source encoding have been converted to generic characters in the target encoding.

4.3 Reference Processing Model

Many Internet protocols and data formats, most notably the very important Web formats HTML, CSS and XML, are based on text. In those formats, everything is text but the relevant specifications impose a structure on the text, giving meaning to certain constructs so as to obtain functionality in addition to that provided by **plain text** (text that is not in the context of markup or a programming language). HTML and XML are **markup languages**, defining documents entirely composed of text but with conventions allowing the separation of this text into **markup** and **character data**. Citing from the XML 1.0 specification [XML 1.0], section 2.4:

"*Text consists of intermingled character data and markup. [...] All text that is not markup constitutes the character data of the document.*"

For the purposes of this section, the important aspect is that everything is text, that is, a sequence of characters.

A **textual data object** is a whole text protocol message or a whole text document, or a part of it that is treated separately for purposes of external storage and retrieval. Examples include external parsed entities in XML and textual MIME entity bodies [MIME-entity].

C013 [S] [C] Textual data objects defined by protocol or format specifications MUST be in a *single* character encoding.

Note that this does not imply that character set switching schemes such as ISO 2022 cannot be used, since such schemes perform character set switching within a single character encoding.

Since its early days, the Web has seen the development of a **Reference**

Processing Model, first described for HTML in RFC 2070 [RFC 2070]. This model was later embraced by XML and CSS. It is applicable to any data format or protocol that is text-based as described above. The essence of the Reference Processing Model is the use of Unicode as a common reference. Use of the Reference Processing Model by a specification does not, however, require that implementations actually use Unicode. The requirement is only that the implementations behave as if the processing took place as described by the Model. Also, while this document uses the term Reference *Processing* Model and describes its properties in terms of processing, the model also applies to specifications that do not explicitly define a processing model.

C014[S] All specifications that involve processing of text MUST specify the processing of text according to the Reference Processing Model, namely:

1. Specifications MUST define text in terms of Unicode characters, not bytes or glyphs.
2. For their textual data objects specifications MAY allow use of any character encoding which can be transcoded to a Unicode encoding form.
3. Specifications MAY choose to disallow or deprecate some character encodings and to make others mandatory. Independent of the actual character encoding, the specified behavior MUST be the same *as if* the processing happened as follows:
    - The character encoding of any textual data object received by the application implementing the specification MUST be determined and the data object MUST be interpreted as a sequence of Unicode characters - this MUST be equivalent to transcoding the data object to some Unicode encoding form, adjusting any character encoding label if necessary, and receiving it in that Unicode encoding form.
    - All processing MUST take place on this sequence of Unicode characters.
    - If text is output by the application, the sequence of Unicode characters MUST be encoded using a character encoding chosen among those allowed by the specification.
4. If a specification is such that multiple textual data objects are involved (such as an XML document referring to external parsed entities), it MAY choose to allow these data objects to be in different character encodings. In all cases, the Reference Processing Model MUST be applied to all textual data objects.

NOTE: All specifications which define applications of the XML 1.0 specification [XML 1.0] automatically inherit this Reference Processing Model. XML is entirely defined in terms of Unicode characters and requires the UTF-8 and UTF-16 character encodings while allowing any other character encoding for parsed entities.

NOTE: When specifications choose to allow character encodings other than Unicode encoding forms, implementers should be aware that the correspondence between the characters of such encodings and Unicode characters may in practice depend on the software used for transcoding.

See the Japanese XML Profile [XML Japanese Profile] for examples of such inconsistencies.

C070 [S] Specifications SHOULD NOT *arbitrarily* exclude code points from the full range of Unicode code points from U+0000 to U+10FFFF inclusive.

C077 [S] Specifications MUST NOT allow code points above U+10FFFF.

Unicode contains some code points for internal use (such as noncharacters) or special functions (such as surrogate code points).

C079 [S] Specifications SHOULD NOT allow the use of codepoints reserved by Unicode for internal use.

C078 [S] Specifications MUST NOT allow the use of surrogate code points.

Excluding code points without good reason conflicts with the W3C goal of universal accessibility. Excluding code points would prevent some scripts from being used which may be important to a user community or communities. For example, without strong reasons to do so, decisions to exclude code points above the Basic Multilingual Plane or to limit code points to the US-ASCII or Latin-1 repertoire are inappropriate. Also, please note that the Unicode Standard requires software to not corrupt any code points.

Other examples of legitimate and non-arbitrary reasons to exclude characters can be seen in *Unicode in XML and other Markup Languages* [UXML], where the use of certain characters is discouraged for reasons such as:

- They are deprecated in the Unicode Standard.
- They cannot be supported without additional data.
- They are better handled by markup.
- They conflict with equivalent markup.

4.4 Choice and Identification of Character Encodings

Because encoded text *cannot* be interpreted and processed without knowing the encoding, it is vitally important that the character encoding (see **4.1 Character Encoding**) is known at all times and places where text is exchanged, stored or processed. In what follows we use 'character encoding' to mean either character encoding form (CEF) or character encoding scheme (CES) depending on the context. When text is transmitted or stored as a byte stream, for instance in a protocol or file system, specification of a CES is required to ensure proper interpretation. In contexts such as an API, where the environment (typically the processor architecture) specifies the byte order of multibyte quantities, specification of a CEF suffices.

C015 [S] Specifications MUST either specify a unique character encoding, or provide character encoding identification mechanisms such that the encoding of text can be reliably identified.

C016 [S] When designing a new protocol, format or API, specifications SHOULD require a unique character encoding.

C017 [S] When basing a protocol, format, or API on a protocol, format, or API that already has rules for character encoding, specifications SHOULD use rather than change these rules.

> **EXAMPLE:** An XML-based format should use the existing XML rules for choosing and determining the character encoding of external entities, rather than invent new ones.

4.4.1 Mandating a unique character encoding

Mandating a unique character encoding is simple, efficient, and robust. There is no need for specifying, producing, transmitting, and interpreting encoding tags. At the receiver, the character encoding will always be understood. There is also no ambiguity as to which character encoding to use if data is transferred non-electronically and later has to be converted back to a digital representation. Even when there is a need for compatibility with existing data, systems, protocols and applications, multiple character encodings can often be dealt with at the boundaries or outside a protocol, format, or API. The DOM [DOM Level 1] is an example of where this was done. The advantages of choosing a unique character encoding are greater when text sizes are small or the specification is close to the actual processing.

C018 [S] When a unique character encoding is required, the character encoding MUST be UTF-8, UTF-16 or UTF-32.

US-ASCII is upwards-compatible with UTF-8 (an US-ASCII string is also a UTF-8 string, see [RFC 3629]), and UTF-8 is therefore appropriate if compatibility with US-ASCII is desired. In other situations, such as for APIs, UTF-16 or UTF-32 may be more appropriate. Possible reasons for choosing one of these include efficiency of internal processing and interoperability with other processes.

> **NOTE:** The IETF Charset Policy [RFC 2277] specifies that on the Internet "*Protocols MUST be able to use the UTF-8 charset*".

> **NOTE:** The XML 1.0 specification [XML 1.0] requires all conforming XML processors to accept both UTF-16 and UTF-8.

4.4.2 Character encoding identification

The MIME Internet specification provides a good example of a mechanism for character encoding identification [MIME-charset][RFC 2978]. The MIME `charset` parameter definition is intended to supply sufficient information to uniquely decode the sequence of bytes of the received data into a sequence of characters. The values are drawn from the IANA charset registry [IANA].

> **NOTE:** Unfortunately, some charset identifiers do not represent a single,

unique character encoding. Instead, these identifiers denote a number of small variations. Even though small, the differences may be crucial and may vary over time. For these identifiers, recovery of the character sequence from a byte sequence is ambiguous. For example, the character encoded as 0x5C in Shift_JIS is ambiguous. This code point sometimes represents a YEN SIGN and sometimes represents a REVERSE SOLIDUS. See the [XML Japanese Profile] for more detail on this example and for additional examples of such ambiguous charset identifiers.

NOTE: The term **charset** derives from 'character set', an expression with a long and tortured history (see [Connolly] for a discussion).

C020 [S] Specifications SHOULD avoid using the terms 'character set' and 'charset' to refer to a character encoding, except when the latter is used to refer to the MIME `charset` parameter or its IANA-registered values. The term 'character encoding', or in specific cases the terms 'character encoding form' or 'character encoding scheme', are RECOMMENDED.

NOTE: In XML, the XML declaration or the text declaration contains the `encoding` pseudo-attribute which identifies the character encoding using the IANA charset.

The IANA charset registry is the official list of names and aliases for character encoding schemes on the Internet.

C021 [S] If the unique encoding approach is not taken, specifications SHOULD require the use of the IANA charset registry names, and in particular the names identified in the registry as 'MIME preferred names', to designate character encodings in protocols, data formats and APIs.

C022 [S] [I] [C] Character encodings that are not in the IANA registry SHOULD NOT be used, except by private agreement.

C023 [S] [I] [C] If an unregistered character encoding is used, the convention of using 'x-' at the beginning of the name MUST be followed.

C049 [I] [C] The character encoding of content SHOULD be chosen so that it maximizes the opportunity to directly represent characters (ie. minimizes the need to represent characters by markup means such as character escapes) while avoiding obscure encodings that are unlikely to be understood by recipients.

NOTE: Due to Unicode's large repertoire and wide base of support, a character encoding based on Unicode is a good choice to encode a document.

C034 [C] If facilities are offered for identifying character encoding, content MUST make use of them; where the facilities offered for character encoding identification include defaults (e.g. in XML 1.0 [XML 1.0]), relying on such defaults is sufficient to satisfy this identification requirement.

C024 [I] [C] Content and software that label text data MUST use one of the names required by the appropriate specification (e.g. the XML specification when editing XML text) and SHOULD use the MIME preferred name of a character encoding to label data in that character encoding.

C025 [I] [C] An IANA-registered `charset` name MUST NOT be used to label text data in a character encoding other than the one identified in the IANA registration of that name.

C026 [S] If the unique encoding approach is not chosen, specifications MUST designate at least one of the UTF-8 and UTF-16 encoding forms of Unicode as admissible character encodings and SHOULD choose at least one of UTF-8 or UTF-16 as required encoding forms (encoding forms that MUST be supported by implementations of the specification).

C027 [S] Specifications that require a default encoding MUST define either UTF-8 or UTF-16 as the default, or both if they define suitable means of distinguishing them.

C028 [S] Specifications MUST NOT propose the use of heuristics to determine the encoding of data.

Examples of heuristics include the use of statistical analysis of byte (pattern) frequencies or character (pattern) frequencies. Heuristics are bad because they will not work consistently across different implementations. Well-defined instructions of how to unambiguously determine a character encoding, such as those given in XML 1.0 [XML 1.0], Appendix F, are not considered heuristics.

C029 [I] *Receiving* software MUST determine the encoding of data from available information according to appropriate specifications.

C030 [I] When an IANA-registered `charset` name is recognized, receiving software MUST interpret the received data according to the encoding associated with the name in the IANA registry.

C031 [I] When no charset is provided receiving software MUST adhere to the default character encoding(s) specified in the specification.

Receiving software may recognize as many character encodings and as many charset names and aliases for them as appropriate.

A field-upgradeable mechanism may be appropriate for this purpose. Certain character encodings are more or less associated with certain languages (e.g. Shift_JIS with Japanese). Trying to support a given language or set of customers may mean that certain character encodings have to be supported. However, one cannot assume universal support for a favoured but non-required encoding. The character encodings that need to be supported may change over time. This document does not give any advice on which character encoding may be appropriate or necessary for the support of any given language.

Because of the layered Web architecture (e.g. formats used over protocols), there may be multiple and at times conflicting information about character encoding.

C035 [S] Specifications MUST define conflict-resolution mechanisms (e.g. priorities) for cases where there is multiple or conflicting information about character encoding.

C033 [I] Software MUST completely implement the mechanisms for character encoding identification and conflict resolution.

4.5 Private use code points

Certain ranges of Unicode code points are designated for private use: the Private Use Area (PUA) (U+E000-F8FF) and planes 15 and 16 (U+F0000-FFFFD and U+100000-10FFFD). These code points are guaranteed to never be allocated to standard characters, and are available for use by private agreement. However, private agreements do not scale on the Web. Code points from different private agreements may collide. Also, a private agreement, and therefore the meaning of the code points, can quickly become lost.

C073 [C] Publicly interchanged content SHOULD NOT use codepoints in the private use area.

> **NOTE:** A typical exception would be the use of the PUA to design and test the encoding of not yet encoded (e.g. historic or rare) scripts.

C076 [C] Content MUST NOT use a code point for any purpose other than that defined by its coded character set.

This prohibits, for example, the construction of fonts that misuse the codepoints in the ISO Latin 1 character set to represent different scripts, characters, or symbols than those actually encoded in iso-8859-1.

C038 [S] Specifications MUST NOT require the use of private use area characters with particular assignments.

C039 [S] Specifications MUST NOT require the use of mechanisms for defining agreements of private use code points.

C040 [S] [I] Specifications and implementations SHOULD NOT disallow the use of private use code points by private agreement.

As an example, XML does not disallow the use of private use code points.

C041 [S] Specifications MAY define markup to allow the transmission of symbols not in Unicode or to identify specific variants of Unicode characters.

> **EXAMPLE:** MathML (see [MathML2] section 3.2.9) defines an element `mglyph` for mathematical symbols not in Unicode.

> **EXAMPLE:** SVG (see [SVG] section 10.14) defines an element `altglyph` which allows the identification of specific display variants of Unicode characters.

C068 [S] Specifications SHOULD allow the inclusion of or reference to pictures and graphics where appropriate, to eliminate the need to (mis)use character-oriented mechanisms for pictures or graphics.

4.6 Character Escaping

Markup languages or programming languages often designate certain characters as **syntax-significant**, giving them specific functions within the language (e.g. '<' and '&' serve as markup delimiters in HTML and XML). As a consequence, these syntax-significant characters cannot be used to represent themselves in text in the same way as all other characters do, creating the need for a mechanism to "*escape*" their syntax-significance. There is also a need, often satisfied by the same or similar mechanisms, to express characters not directly representable in the character encoding chosen for a particular document or program (an instance of the markup or programming language).

Formally, a **character escape** is a syntactic device defined in a markup or programming language that allows one or more of:

1. expressing syntax-significant characters while disregarding their significance in the syntax of the language, or
2. expressing characters not representable in the character encoding chosen for an instance of the language, or
3. expressing characters in general, without use of the corresponding encoded characters.

Escaping a character means expressing it using such a syntactic device, appropriate to the format or protocol in which the character appears; **expanding a character escape** (or **unescaping**) means replacing it with the character that it represents.

> **EXAMPLE:** HTML and XML define 'Numeric Character References' which allow both the escaping of syntax-significance and the expression of arbitrary Unicode characters. Expressed as < or < the character '<' will not be parsed as a markup delimiter.

> **EXAMPLE:** The programming language Java uses "" to delimit strings. To express "" within a string, one may escape it as '\"'.

> **EXAMPLE:** XML defines 'CDATA sections' which allow escaping the syntax-significance of all characters between the CDATA section delimiters. CDATA sections prevent the expression of characters using numeric character references.

The following guidelines apply to the way specifications define character escapes.

- ***C042*** [S] Specifications SHOULD NOT invent a new escaping mechanism if an appropriate one already exists.
- ***C043*** [S] The number of different ways to escape a character SHOULD be minimized (ideally to one).

  A well-known counter-example is that for historical reasons, both HTML and XML have redundant decimal (&#ddddd;) and hexadecimal (&#xhhhh;) character escapes.
- ***C044*** [S] Escape syntax SHOULD require either explicit end delimiters or a fixed number of characters in each character escape. Escape syntaxes where the end is determined by any character outside the set of characters admissible in the character escape itself SHOULD be avoided.

  These character escapes are not clear visually, and can cause an editor to insert spurious line-breaks when word-wrapping on spaces. Forms like SPREAD's &UABCD; [SPREAD] or XML's &#xhhhh;, where the character escape is explicitly terminated by a semicolon, are much better.
- ***C045*** [S] Whenever specifications define character escapes that allow the representation of characters using a number, the number MUST represent the Unicode code point of the character and SHOULD be in hexadecimal notation.
- ***C046*** [S] Escaped characters SHOULD be acceptable wherever their unescaped forms are; this does not preclude that syntax-significant characters, when escaped, lose their significance in the syntax. In particular, if a character is acceptable in identifiers and comments, then its escaped form should also be acceptable.

The following guidelines apply to content developers, as well as to software that generates content:

- ***C047*** [I] [C] Escapes SHOULD only be used when the characters to be expressed are not directly representable in the format or the character encoding of the document, or when the visual representation of the character is unclear.

  NOTE: An example of when the visual representation of the character is unclear is the use of   to distinguish a non-breaking space from a normal space.
- ***C048*** [I] [C] Content SHOULD use the hexadecimal form of character escapes rather than the decimal form when there are both.

  NOTE: The hexadecimal form is preferred because character encoding standards (in particular Unicode) usually list character numbers as hexadecimal, making lookup easier.

5 Compatibility and Formatting Characters

This specification does not address the suitability of particular characters for use in markup languages, in particular formatting characters and compatibility equivalents. For detailed recommendations about the use of compatibility and formatting characters, see *Unicode in XML and other Markup Languages* [UXML].

C050 [S] Specifications SHOULD exclude compatibility characters in the syntactic elements (markup, delimiters, identifiers) of the formats they define.

6 Strings

6.1 String concepts

Various specifications use the notion of a 'string', sometimes without defining precisely what is meant and sometimes defining it differently from other specifications. The reason for this variability is that there are in fact multiple reasonable definitions for a string, depending on one's intended use of the notion; the term 'string' is used for all these different notions because these are actually just different views of the same reality: a piece of text stored inside a computer.

Byte string: A string viewed as a sequence of bytes representing characters in a particular character encoding. This corresponds to a character encoding scheme (CES). Text processing of a byte string is dependent on the particular encoding used. When the encoding changes the processing must also be changed to reflect the stucture of the new encoding. Such a change could require significant redesign of the functions or API used to process the byte strings as text. Therefore, this definition is only useful in specifications when the textual nature of a string is unimportant and the string is considered only as a piece of opaque data with a length in bytes (such as when copying a buffer).

C011 [S] Specifications SHOULD NOT define a string as a 'byte string'.

> **EXAMPLE:** This is a counter-example, illustrating one reason why considering strings as byte strings may be problematic. Consider text containing the character U+233B4 (a Chinese character meaning 'stump of tree') encoded as UTF-16 in big-endian byte order (UTF-16BE). The text will contain the bytes D8 4C DF B4. If one searches this text, considered as a byte string, for the character U+4CDF (another Chinese character meaning 'phoenix'), an erroneous match will be found on the bytes 4C DF that are the UTF-16BE representation of U+4CDF.

Code unit string: A string viewed as a sequence of code units representing characters in a particular character encoding. This corresponds to a character encoding form (CEF). A definition of a code unit string needs to include the size of the code units (e.g. 16 bits) and the character encoding used (e.g. UTF-16). Code unit strings are useful in APIs that expose a physical representation of string data based on reliable knowledge of the encoding forms that are likely candidates for implementation. Example: For the DOM [DOM Level 1], UTF-16 was chosen based on widespread implementation practice. In general, 'code unit string' is only useful if the implementation candidates are likely to be either UTF-16 or UTF-32.

Character string: A string viewed as a sequence of characters, each represented by a code point in Unicode [Unicode]. This is usually what programmers consider to be a string, although it may not match exactly what

most users perceive as characters. This is the highest layer of abstraction that ensures interoperability with very low implementation effort. The 'character string' definition of a string is generally the most useful. Good examples using this definition include the Production [2] of XML 1.0 [XML 1.0], the SGML declaration of HTML 4.0 [HTML 4.01], and the character model of RFC 2070 [RFC 2070].

C012 [S] The 'character string' definition SHOULD be used by most specifications.

EXAMPLE: Consider the string comprising the characters U+233B4 (a Chinese character meaning 'stump of tree'), U+2260 NOT EQUAL TO, U+0071 LATIN SMALL LETTER Q and U+030C COMBINING CARON, encoded in UTF-16 in big-endian byte order. The rows of the following table show the string viewed as a character string, code unit string and byte string, respectively:

Glyphs	𣎴		≠		q		◌̌	
Character string	U+233B4		U+2260		U+0071		U+030C	
Code unit string	D84C	DFB4	2260		0071		030C	
Byte string	D8	4C	DF	B4	22	60	00	71	03	0C

NOTE: It is also possible to view a string as a sequence of **grapheme clusters**. Grapheme clusters divide the text into units that correspond more closely than character strings to the user's perception of where character boundaries occur in a visually rendered text. A discussion of grapheme clusters is given at the end of Section 2.10 of the Unicode Standard, Version 4 [Unicode 4.0]; a formal definition is given in Unicode Standard Annex #29 [UTR #29]. The Unicode Standard defines *default* grapheme clustering. Some languages require tailoring to this default. For example, a Slovak user might wish to treat the default pair of grapheme clusters "ch" as a single grapheme cluster. Note that the interaction between the language of string content and the end-user's preferences may be complex.

6.2 String indexing

There are many situations where a software process needs to access a substring or to point within a string and does so by the use of **indices**, i.e. numeric "*positions*" within a string. Where such indices are exchanged between components of the Web, there is a need for an agreed-upon definition of string indexing in order to ensure consistent behavior. The requirements for string indexing are discussed in *Requirements for String Identity Matching* [CharReq], section 4. The two main questions that arise are: "*What is the unit of counting?*" and "*Do we start counting at 0 or 1?*".

The example in the previous section, **6.1 String concepts**, shows a string

viewed as a character string, code unit string and byte string, respectively, each of which involves different units for indexing.

Depending on the particular requirements of a process, the unit of counting may correspond to definitions of a string provided in section **6.1 String concepts**. In particular:

- ***C051*** [S] [I] The character string is RECOMMENDED as a basis for string indexing.

  (Example: the XML Path Language [XPath]).
- ***C052*** [S] [I] A code unit string MAY be used as a basis for string indexing if this results in a significant improvement in the efficiency of internal operations when compared to the use of character string.

  (Example: the use of UTF-16 in [DOM Level 1]).
- ***C071*** [S] [I] Grapheme clusters MAY be used as a basis for string indexing in applications where user interaction is the primary concern.

  See Unicode Standard Annex #29, Text Boundaries [UTR #29].

  C074 [S] Specifications that define indexing in terms of grapheme clusters MUST either: a) define grapheme clusters in terms of default grapheme clusters as defined in Unicode Standard Annex #29, Text Boundaries [UTR #29], or b) define specifically how tailoring is applied to the indexing operation.
- ***C072*** [S] [I] The use of byte strings for indexing is NOT RECOMMENDED.

It is noteworthy that there exist other, non-numeric ways of identifying substrings which have favorable properties. For instance, substrings based on string matching are quite robust against small edits; substrings based on document structure (in structured formats such as XML) are even more robust against edits and even against translation of a document from one human language to another.

C053 [S] Specifications that need a way to identify substrings or point within a string SHOULD provide ways other than string indexing to perform this operation.

C054 [I] [C] Users of specifications (software developers, content developers) SHOULD whenever possible prefer ways other than string indexing to identify substrings or point within a string.

Experience shows that more general, flexible and robust specifications result when individual characters are understood and processed as substrings, identified by a position before and a position after the substring. Understanding indices as boundary positions *between* the counting units also makes it easier to relate the indices resulting from the different string definitions.

C055 [S] Specifications SHOULD understand and process single characters as substrings, and treat indices as boundary positions *between* counting units, regardless of the choice of counting units.

C056 [S] Specifications of APIs SHOULD NOT specify single characters or single 'units of encoding' as argument or return types.

> **EXAMPLE:** The function `uppercase("ß")` cannot return the proper result (the two-character string 'SS') if the return type of the `uppercase` function is defined to be a single character. Note, also, that there is not necessarily a one-to-one mapping between characters and units of sound, input, etc. as described in **3 Perceptions of Characters**.

The issue of index origin, i.e. whether we count from 0 or 1, actually arises only after a decision has been made on whether it is the units themselves that are counted or the positions between the units.

C057 [S] When the positions between the units are counted for string indexing, starting with an index of 0 for the position at the start of the string is the RECOMMENDED solution, with the last index then being equal to the number of counting units in the string.

7 Referencing the Unicode Standard and ISO/IEC 10646

Specifications often need to make references to the Unicode Standard or International Standard ISO/IEC 10646. Such references must be made with care, especially when normative. The questions to be considered are:

- Which standard should be referenced?
- How to reference a particular version?
- When to use versioned vs. unversioned references?

ISO/IEC 10646 is developed and published jointly by ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission). The Unicode Standard is developed and published by the Unicode Consortium, an organization of major computer corporations, software producers, database vendors, national governments, research institutions, international agencies, various user groups, and interested individuals. The Unicode Standard is comparable in standing to W3C Recommendations.

ISO/IEC 10646 and the Unicode Standard define exactly the same coded character set (CCS) (same repertoire, same code points) and encoding forms. They are actively maintained in synchrony by liaisons and overlapping membership between the respective technical committees. In addition to the jointly defined CCS and encoding forms, the Unicode Standard adds normative and informative lists of character properties, normative character equivalence and normalization specifications, a normative algorithm for bidirectional text and a large amount of useful implementation information. In short, the Unicode Standard adds semantics to the characters that ISO/IEC 10646 merely enumerates. Conformance to the Unicode Standard implies conformance to ISO/IEC 10646, see [Unicode 4.0] Appendix C.

C062 [S] Since specifications in general need both a definition for their characters and the semantics associated with these characters, specifications

SHOULD include a reference to the Unicode Standard, whether or not they include a reference to ISO/IEC 10646.

By providing a reference to the Unicode Standard implementers can benefit from the wealth of information provided in the standard and on the Unicode Consortium Web site.

The fact that both ISO/IEC 10646 and the Unicode Standard are evolving (in synchrony) raises the issue of versioning: should a specification refer to a specific version of the standard, or should it make a generic reference, so that the normative reference is to the version current at the time of *reading* the specification? In general the answer is *both*.

C063 [S] A generic reference to the Unicode Standard MUST be made if it is desired that characters allocated after a specification is published are usable with that specification. A specific reference to the Unicode Standard MAY be included to ensure that functionality depending on a particular version is available and will not change over time.

An example would be the set of characters acceptable as Name characters in XML 1.0 [XML 1.0], which is an enumerated list that parsers must implement to validate names.

> **NOTE:** See http://www.unicode.org/unicode/standard/versions/#Citations for guidance on referring to specific versions of the Unicode Standard.

A generic reference can be formulated in two ways:

1. By explicitly including a *generic* entry in the bibliography section of a specification and simply referring to that entry in the body of the specification. Such a generic entry contains text such as "*... as it may from time to time be revised or amended*".
2. By including a *specific* entry in the bibliography and adding text such as "*... as it may from time to time be revised or amended*" at the point of reference in the body of the specification.

It is an editorial matter, best left to each specification, which of these two formulations is used. Examples of the first formulation can be found in the bibliography of this specification (see the entries for [ISO/IEC 10646] and [Unicode]). Examples of the latter, as well as a discussion of the versioning issue with respect to MIME `charset` parameters for UCS encodings, can be found in [RFC 3629] and [RFC 2781].

C064 [S] All *generic* references to the Unicode Standard [Unicode] MUST refer to the latest version of the Unicode Standard available at the date of publication of the containing specification.

C065 [S] All *generic* references to ISO/IEC 10646 [ISO/IEC 10646] MUST refer to the latest version of ISO/IEC 10646 available at the date of publication of the containing specification.

A References

A.1 Normative References

IANA
Internet Assigned Numbers Authority, *Official Names for Character Sets*. (See http://www.iana.org/assignments/character-sets.)

ISO/IEC 10646
ISO/IEC 10646:2003, *Information technology -- Universal Multiple-Octet Coded Character Set (UCS)*, as, from time to time, amended, replaced by a new edition or expanded by the addition of new parts. (See http://www.iso.org/iso/en/ISOOnline.openerpage for the latest version.)

MIME-entity
N. Freed, N. Borenstein, *Multipurpose Internet Mail Extensions (MIME). Part One: Format of Internet Message Bodies*, RFC 2045, November 1996, http://www.ietf.org/rfc/rfc2045.txt.

MIME-charset
Multipurpose Internet Mail Extensions (MIME). Part Two: Media Types, N. Freed, N. Borenstein, RFC 2046, November 1996, http://www.ietf.org/rfc/rfc2046.txt.

RFC 2119
S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC 2119. (See http://www.ietf.org/rfc/rfc2119.txt.)

Unicode
The Unicode Consortium, *The Unicode Standard, Version 4*, ISBN 0-321-18578-1, as updated from time to time by the publication of new versions. (See http://www.unicode.org/unicode/standard/versions for the latest version and additional information on versions of the standard and of the Unicode Character Database).

Unicode 3.2
The Unicode Consortium, *The Unicode Standard, Version 3.2.0* is defined by *The Unicode Standard, Version 3.0* (Reading, MA, Addison-Wesley, 2000. ISBN 0-201-61633-5), as amended by the *Unicode Standard Annex #27: Unicode 3.1* (see http://www.unicode.org/reports/tr27) and by the *Unicode Standard Annex #28: Unicode 3.2* (see http://www.unicode.org/reports/tr28).

Unicode 4.0
The Unicode Consortium. *The Unicode Standard, Version 4.0*, Reading, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1. (See http://www.unicode.org/versions/Unicode4.0.0/)

A.2 Other References

CharNorm
Martin J. Dürst, François Yergeau, Richard Ishida, Misha Wolf, Tex Texin, Addison Phillips *Character Model for the World Wide Web 1.0: Normalization*, W3C Working Draft. (See http://www.w3.org/TR/charmod-norm.)

CharIRI
Martin J. Dürst, François Yergeau, Richard Ishida, Misha Wolf, Tex Texin,

Character Model for the World Wide Web 1.0: Resource Identifiers, W3C Candidate Recommendation. (See http://www.w3.org/TR/charmod-resid.)

CharReq

Martin J. Dürst, *Requirements for String Identity Matching and String Indexing*, W3C Working Draft. (See http://www.w3.org/TR/WD-charreq.)

Connolly

D. Connolly, *Character Set Considered Harmful*, W3C Note. (See http://www.w3.org/MarkUp/html-spec/charset-harmful.)

CSS2

Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs, Eds., *Cascading Style Sheets, level 2* (CSS2 Specification), W3C Recommendation. (See http://www.w3.org/TR/REC-CSS2.)

DOM Level 1

Vidur Apparao et al., *Document Object Model (DOM) Level 1 Specification*, W3C Recommendation. (See http://www.w3.org/TR/REC-DOM-Level-1.)

HTML 4.0

Dave Raggett, Arnaud Le Hors, Ian Jacobs, Eds., *HTML 4.0 Specification*, W3C Recommendation, 18-Dec-1997 (See http://www.w3.org/TR/REC-html40-971218.)

HTML 4.01

Dave Raggett, Arnaud Le Hors, Ian Jacobs, Eds., *HTML 4.01 Specification*, W3C Recommendation. (See http://www.w3.org/TR/html401.)

ISO/IEC 646

ISO/IEC 646:1991, *Information technology -- ISO 7-bit coded character set for information interchange*. This standard defines an International Reference Version (IRV) which corresponds exactly to what is widely known as ASCII or US-ASCII. ISO/IEC 646 was based on the earlier standard ECMA-6. ECMA has maintained its standard up to date with respect to ISO/IEC 646 and makes an electronic copy available at http://www.ecma-international.org/publications/standards/Ecma-006.htm

ISO/IEC 9541-1

ISO/IEC 9541-1:1991, *Information technology -- Font information interchange -- Part 1: Architecture*. (See http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=17277 for the latest version.)

ISO/IEC 14651

ISO/IEC 14651:2000, *Information technology -- International string ordering and comparison -- Method for comparing character strings and description of the common template tailorable ordering* as, from time to time, amended, replaced by a new edition or expanded by the addition of new parts. (See http://www.iso.org/iso/en/ISOOnline.openerpage for the latest version.)

MathML2

David Carlisle, Patrick Ion, Robert Miner, Nico Poppelier, Eds., *Mathematical Markup Language (MathML) Version 2.0*, W3C Recommendation. (See http://www.w3.org/TR/MathML2.)

Nicol

Gavin Nicol, *The Multilingual World Wide Web*, Chapter 2: The WWW As A Multilingual Application. (See http://www.mind-to-

mind.com/library/papers/multilingual/multilingual-www.html.)

RFC 2070
F. Yergeau, G. Nicol, G. Adams, M. Dürst, *Internationalization of the Hypertext Markup Language*, IETF RFC 2070, January 1997. (See http://www.ietf.org/rfc/rfc2070.txt.)

RFC 2277
H. Alvestrand, *IETF Policy on Character Sets and Languages*, IETF RFC 2277, BCP 18, January 1998. (See http://www.ietf.org/rfc/rfc2277.txt.)

RFC 2978
N. Freed, J. Postel, *IANA Charset Registration Procedures*, IETF RFC 2978, BCP 19, October 2000. (See http://www.ietf.org/rfc/rfc2978.txt.)

RFC 3629
F. Yergeau, *UTF-8, a transformation format of ISO 10646*, IETF RFC 3629, STD 63, November 2003. (See http://www.ietf.org/rfc/rfc3629.txt.)

RFC 2781
P. Hoffman, F. Yergeau, *UTF-16, an encoding of ISO 10646*, IETF RFC 2781, February 2000. (See http://www.ietf.org/rfc/rfc2781.txt.)

SPREAD
SPREAD - Standardization Project for East Asian Documents Universal Public Entity Set. (See http://www.ascc.net/xml/resource/entities/index.html)

SVG
Jon Ferraiolo, 藤沢 淳 (FUJISAWA Jun), Dean Jackson, Ed., *Scalable Vector Graphics (SVG) 1.1 Specification*, W3C Recommendation. (See http://www.w3.org/TR/SVG.)

UTR #10
Mark Davis, Ken Whistler, *Unicode Collation Algorithm*, Unicode Technical Report #10. (See http://www.unicode.org/unicode/reports/tr10.)

UTR #17
Ken Whistler, Mark Davis, *Character Encoding Model*, Unicode Technical Report #17. (See http://www.unicode.org/unicode/reports/tr17.)

UTR #29
Mark Davis, *Text Boundaries*, Unicode Standard Annex #29. (See http://www.unicode.org/unicode/reports/tr29 for the latest version).

UXML
Martin Dürst and Asmus Freytag, *Unicode in XML and other Markup Languages*, Unicode Technical Report #20 and W3C Note. (See http://www.w3.org/TR/unicode-xml.)

XML 1.0
Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, Eds., *Extensible Markup Language (XML) 1.0*, W3C Recommendation. (See http://www.w3.org/TR/REC-xml.)

XML Japanese Profile
MURATA Makoto Ed., *XML Japanese Profile*, W3C Note. (See http://www.w3.org/TR/japanese-xml.)

XPath
James Clark, Steve DeRose, Eds, *XML Path Language (XPath) Version 1.0*, W3C Recommendation. (See http://www.w3.org/TR/xpath.)

B Examples of Characters, Keystrokes and Glyphs (Non-

Normative)

A few examples will help make sense all this complexity of text in computers (which is mostly a reflection of the complexity of human writing systems). Let us start with a very simple example: a user, equipped with a US-English keyboard, types "*Foo*", which the computer encodes as 16-bit values (the UTF-16 encoding of Unicode) and displays on the screen.

Keystrokes	Shift-f	o	o
Input characters	F	o	o
Encoded characters (byte values in hex)	0046	006F	006F
Display	Foo		

Example: Basic Latin

The only complexity here is the use of a modifier (Shift) to input the capital 'F'.

A slightly more complex example is a user typing 'çé' on a traditional French-Canadian keyboard, which the computer again encodes in UTF-16 and displays. We assume that this particular computer uses a fully composed form of UTF-16.

Keystrokes	¸	c	é
Input characters	ç		é
Encoded characters (byte values in hex)	00E7		00E9
Display	çé		

Example: Latin with diacritics

A few interesting things are happening here: when the user types the cedilla ('¸'), nothing happens except for a change of state of the keyboard driver; the cedilla is a **dead key**. When the driver gets the c keystroke, it provides a complete 'ç' character to the system, which represents it as a single 16-bit code unit and displays a 'ç' glyph. The user then presses the dedicated 'é' key, which results in, again, a character represented by two bytes. Most systems will display this as one glyph, but it is also possible to combine two glyphs (the base letter and the accent) to obtain the same rendering.

On to a Japanese example: our user employs a **romaji input method** to type '日本語' (U+65E5, U+672C, U+8A9E), which the computer encodes in UTF-16 and displays.

Keystrokes	n i h o n g o <space> <return>		
Input characters	日	本	語

Encoded characters (byte values in hex)	65E5	672C	8A9E
Display	日本語		

Example: Japanese

The interesting aspect here is input: the user types Latin characters, which are converted on the fly to kana (not shown here), and then to kanji when the user requests conversion by pressing <space>; the kanji characters are finally sent to the application when the user presses <return>. The user has to type a total of nine keystrokes before the three characters are produced, which are then encoded and displayed rather trivially.

A Persian example, using Arabic script, will show different phenomena:

Keystrokes	ل	ا	لا		ی	ی
Input characters	ل	ا	ل	ا	ی	ی
Encoded characters (byte values in hex)	0644	0627	0644	0627	06CC	06CC
Display	لالایی					

Example: Persian

Here the first two keystrokes each produce an input character and an encoded character, but the pair is displayed as a single glyph ('لا', a lam-alef ligature).

The next keystroke is a lam-alef, which some Arabic script keyboards have; it produces the same two characters which are displayed similarly, but this second lam-alef is placed to the *left* of the first one when displayed. The last two keystrokes produce two identical characters which are rendered by two different glyphs (a medial form followed to its left by a final form). We thus have 5 keystrokes producing 6 characters and 4 glyphs laid out right-to-left.

A final example in Tamil, typed with an ISCII keyboard, will illustrate some additional phenomena:

Keystrokes	ட	◌ா	ங	◌்	க	ே◌ா
Input characters	ட	◌ா	ங	◌்	க	ே◌ா
Encoded characters (byte values in hex)	0B9F	0BBE	0B99	0BCD	0B95	0BCB
Display	டாங்கோ					

Example: Tamil

Here input is straightforward, but note that contrary to the preceding accented Latin example, the virama diacritic ' ◌் ' (U+0BCD) is entered *after* the 'ங' (U+0B99) to which it applies. Rendering is interesting for the last two characters. The last one ' கௌ' (U+0BCC) clearly consists of two glyphs which *surround* the glyph of the next to last character 'க' (U+0B95).

C Example text (Non-Normative)

The following are textual versions of strings or characters used in image-based examples in this document. They are provided here for the benefit of those who want to cut and paste the text for their own testing.

1. Section: **3.3 Units of visual rendering**
   Example: ع د د <space> م ا ر س <space> ١٩ ٩٨
   Text: عدد مارس ١٩٩٨
2. Section: **6.1 String concepts**
   Example: 不≠q◌̌
   Text: □≠q□
3. Section: **B Examples of Characters, Keystrokes and Glyphs**
   Example: 日本語
   Text: 日本語
4. Section: **B Examples of Characters, Keystrokes and Glyphs**
   Example: لاغ لا
   Text: لاغ لا
5. Section: **B Examples of Characters, Keystrokes and Glyphs**
   Example: டாங்கோ
   Text: டாங்கோ

D List of conformance criteria (Non-Normative)

This is a list of the conformance criteria in this specification, in document order. This list can be used to check specifications, implementations, and content for conformance to this specification.

When doing so, the following points should be kept in mind:

- To ensure that you understand the meaning, read the whole document first. Use this list as a quick reference only after having first read the conformance criteria in context in the main body of the text.
- If the meaning of a conformance criterion in this list is still unclear after referring back to the surrounding text in the main body of the document, consider sending a comment to www-i18n-comments@w3.org (publicly archived).

- Not all conformance criteria apply to all specifications, implementations, or content. Before checking for actual conformance, applicability should be checked. As an example, C010 only applies to specifications. As another example, C002 applies to specifications, implementations, and content, but only if it deals with mapping between characters and units of displayed text.

C001	[S] [I] [C]	Specifications, software and content MUST NOT require or depend on a one-to-one correspondence between characters and the sounds of a language.
C002	[S] [I] [C]	Specifications, software and content MUST NOT require or depend on a one-to-one mapping between characters and units of displayed text.
C003	[S] [I] [C]	Protocols, data formats and APIs MUST store, interchange or process text data in logical order.
C075	[I]	Independent of whether some implementation uses logical selection or visual selection, characters selected MUST be kept in logical order in storage.
C004	[S]	Specifications of protocols and APIs that involve selection of ranges SHOULD provide for discontiguous logical selections, at least to the extent necessary to support implementation of visual selection on screen on top of those protocols and APIs.
C005	[S] [I]	Specifications and software MUST NOT require nor depend on a single keystroke resulting in a single character, nor that a single character be input with a single keystroke (even with modifiers), nor that keyboards are the same all over the world.
C006	[S] [I]	Software that sorts or searches text for users SHOULD do so on the basis of appropriate collation units and ordering rules for the relevant language and/or application.
C007	[S] [I]	Where searching or sorting is done dynamically, particularly in a multilingual environment, the 'relevant language' SHOULD be determined to be that of the current user, and may thus differ from user to user.
C066	[S] [I]	Software that allows users to sort or search text SHOULD allow the user to select alternative rules for collation units and ordering.
C008	[S] [I]	Specifications and implementations of sorting and searching algorithms SHOULD accommodate text that contains any character in Unicode.
C009	[S] [I]	Specifications, software and content MUST NOT require or depend on a one-to-one relationship between characters and units of physical storage.
C010	[S]	When specifications use the term 'character' the specifications MUST define which meaning they intend.
C067	[S]	Specifications SHOULD use specific terms, when available, instead of the general term 'character'.
C013	[S] [C]	Textual data objects defined by protocol or format specifications MUST be in a *single* character encoding.

C014 [S] All specifications that involve processing of text MUST specify the processing of text according to the Reference Processing Model, namely:

1. Specifications MUST define text in terms of Unicode characters, not bytes or glyphs.
2. For their textual data objects specifications MAY allow use of any character encoding which can be transcoded to a Unicode encoding form.
3. Specifications MAY choose to disallow or deprecate some character encodings and to make others mandatory. Independent of the actual character encoding, the specified behavior MUST be the same *as if* the processing happened as follows:
    - The character encoding of any textual data object received by the application implementing the specification MUST be determined and the data object MUST be interpreted as a sequence of Unicode characters - this MUST be equivalent to transcoding the data object to some Unicode encoding form, adjusting any character encoding label if necessary, and receiving it in that Unicode encoding form.
    - All processing MUST take place on this sequence of Unicode characters.
    - If text is output by the application, the sequence of Unicode characters MUST be encoded using a character encoding chosen among those allowed by the specification.
4. If a specification is such that multiple textual data objects are involved (such as an XML document referring to external parsed entities), it MAY choose to allow these data objects to be in different character encodings. In all cases, the Reference Processing Model MUST be applied to all textual data objects.

C070 [S] Specifications SHOULD NOT *arbitrarily* exclude code points from the full range of Unicode code points from U+0000 to U+10FFFF inclusive.

C077 [S] Specifications MUST NOT allow code points above U+10FFFF.

C079 [S] Specifications SHOULD NOT allow the use of codepoints reserved by Unicode for internal use.

C078 [S] Specifications MUST NOT allow the use of surrogate code points.

C015 [S] Specifications MUST either specify a unique character encoding, or provide character encoding identification mechanisms such that the encoding of text can be reliably identified.

C016	[S]	When designing a new protocol, format or API, specifications SHOULD require a unique character encoding.
C017	[S]	When basing a protocol, format, or API on a protocol, format, or API that already has rules for character encoding, specifications SHOULD use rather than change these rules.
C018	[S]	When a unique character encoding is required, the character encoding MUST be UTF-8, UTF-16 or UTF-32.
C020	[S]	Specifications SHOULD avoid using the terms 'character set' and 'charset' to refer to a character encoding, except when the latter is used to refer to the MIME `charset` parameter or its IANA-registered values. The term 'character encoding', or in specific cases the terms 'character encoding form' or 'character encoding scheme', are RECOMMENDED.
C021	[S]	If the unique encoding approach is not taken, specifications SHOULD require the use of the IANA charset registry names, and in particular the names identified in the registry as 'MIME preferred names', to designate character encodings in protocols, data formats and APIs.
C022	[S] [I] [C]	Character encodings that are not in the IANA registry SHOULD NOT be used, except by private agreement.
C023	[S] [I] [C]	If an unregistered character encoding is used, the convention of using 'x-' at the beginning of the name MUST be followed.
C049	[I] [C]	The character encoding of content SHOULD be chosen so that it maximizes the opportunity to directly represent characters (ie. minimizes the need to represent characters by markup means such as character escapes) while avoiding obscure encodings that are unlikely to be understood by recipients.
C034	[C]	If facilities are offered for identifying character encoding, content MUST make use of them; where the facilities offered for character encoding identification include defaults (e.g. in XML 1.0 [XML 1.0]), relying on such defaults is sufficient to satisfy this identification requirement.
C024	[I] [C]	Content and software that label text data MUST use one of the names required by the appropriate specification (e.g. the XML specification when editing XML text) and SHOULD use the MIME preferred name of a character encoding to label data in that character encoding.
C025	[I] [C]	An IANA-registered `charset` name MUST NOT be used to label text data in a character encoding other than the one identified in the IANA registration of that name.
C026	[S]	If the unique encoding approach is not chosen, specifications MUST designate at least one of the UTF-8 and UTF-16 encoding forms of Unicode as admissible character encodings and SHOULD choose at least one of UTF-8 or UTF-16 as required encoding forms (encoding forms that MUST be supported by implementations of the specification).
C027	[S]	Specifications that require a default encoding MUST define either UTF-8 or UTF-16 as the default, or both if they define

suitable means of distinguishing them.

C028 [S] Specifications MUST NOT propose the use of heuristics to determine the encoding of data.

C029 [I] *Receiving* software MUST determine the encoding of data from available information according to appropriate specifications.

C030 [I] When an IANA-registered `charset` name is recognized, receiving software MUST interpret the received data according to the encoding associated with the name in the IANA registry.

C031 [I] When no charset is provided receiving software MUST adhere to the default character encoding(s) specified in the specification.

C035 [S] Specifications MUST define conflict-resolution mechanisms (e.g. priorities) for cases where there is multiple or conflicting information about character encoding.

C033 [I] Software MUST completely implement the mechanisms for character encoding identification and conflict resolution.

C073 [C] Publicly interchanged content SHOULD NOT use codepoints in the private use area.

C076 [C] Content MUST NOT use a code point for any purpose other than that defined by its coded character set.

C038 [S] Specifications MUST NOT require the use of private use area characters with particular assignments.

C039 [S] Specifications MUST NOT require the use of mechanisms for defining agreements of private use code points.

C040 [S] [I] Specifications and implementations SHOULD NOT disallow the use of private use code points by private agreement.

C041 [S] Specifications MAY define markup to allow the transmission of symbols not in Unicode or to identify specific variants of Unicode characters.

C068 [S] Specifications SHOULD allow the inclusion of or reference to pictures and graphics where appropriate, to eliminate the need to (mis)use character-oriented mechanisms for pictures or graphics.

C042 [S] Specifications SHOULD NOT invent a new escaping mechanism if an appropriate one already exists.

C043 [S] The number of different ways to escape a character SHOULD be minimized (ideally to one).

C044 [S] Escape syntax SHOULD require either explicit end delimiters or a fixed number of characters in each character escape. Escape syntaxes where the end is determined by any character outside the set of characters admissible in the character escape itself SHOULD be avoided.

C045 [S] Whenever specifications define character escapes that allow the representation of characters using a number, the number MUST represent the Unicode code point of the character and SHOULD be in hexadecimal notation.

C046	[S]	Escaped characters SHOULD be acceptable wherever their unescaped forms are; this does not preclude that syntax-significant characters, when escaped, lose their significance in the syntax. In particular, if a character is acceptable in identifiers and comments, then its escaped form should also be acceptable.
C047	[I] [C]	Escapes SHOULD only be used when the characters to be expressed are not directly representable in the format or the character encoding of the document, or when the visual representation of the character is unclear.
C048	[I] [C]	Content SHOULD use the hexadecimal form of character escapes rather than the decimal form when there are both.
C050	[S]	Specifications SHOULD exclude compatibility characters in the syntactic elements (markup, delimiters, identifiers) of the formats they define.
C011	[S]	Specifications SHOULD NOT define a string as a 'byte string'.
C012	[S]	The 'character string' definition SHOULD be used by most specifications.
C051	[S] [I]	The character string is RECOMMENDED as a basis for string indexing.
C052	[S] [I]	A code unit string MAY be used as a basis for string indexing if this results in a significant improvement in the efficiency of internal operations when compared to the use of character string.
C071	[S] [I]	Grapheme clusters MAY be used as a basis for string indexing in applications where user interaction is the primary concern.
C074	[S]	Specifications that define indexing in terms of grapheme clusters MUST either: a) define grapheme clusters in terms of default grapheme clusters as defined in Unicode Standard Annex #29, Text Boundaries [UTR #29], or b) define specifically how tailoring is applied to the indexing operation.
C072	[S] [I]	The use of byte strings for indexing is NOT RECOMMENDED.
C053	[S]	Specifications that need a way to identify substrings or point within a string SHOULD provide ways other than string indexing to perform this operation.
C054	[I] [C]	Users of specifications (software developers, content developers) SHOULD whenever possible prefer ways other than string indexing to identify substrings or point within a string.
C055	[S]	Specifications SHOULD understand and process single characters as substrings, and treat indices as boundary positions *between* counting units, regardless of the choice of counting units.
C056	[S]	Specifications of APIs SHOULD NOT specify single characters or single 'units of encoding' as argument or return types.
C057	[S]	When the positions between the units are counted for string indexing, starting with an index of 0 for the position at the start

of the string is the RECOMMENDED solution, with the last index then being equal to the number of counting units in the string.

C062 [S] Since specifications in general need both a definition for their characters and the semantics associated with these characters, specifications SHOULD include a reference to the Unicode Standard, whether or not they include a reference to ISO/IEC 10646.

C063 [S] A generic reference to the Unicode Standard MUST be made if it is desired that characters allocated after a specification is published are usable with that specification. A specific reference to the Unicode Standard MAY be included to ensure that functionality depending on a particular version is available and will not change over time.

C064 [S] All *generic* references to the Unicode Standard [Unicode] MUST refer to the latest version of the Unicode Standard available at the date of publication of the containing specification.

C065 [S] All *generic* references to ISO/IEC 10646 [ISO/IEC 10646] MUST refer to the latest version of ISO/IEC 10646 available at the date of publication of the containing specification.

E Changes since the Proposed Recommendation (Non-Normative)

- A small number of links and references were updated in the references section.
- Minor editorial change to paragraph after C076 to clarify: "This prohibits, for example, the construction of fonts that misuse the repertoire encoded by iso-8859-1 to represent different scripts, characters, or symbols than what is actually encoded in iso-8859-1." changed to "This prohibits, for example, the construction of fonts that misuse the codepoints in the ISO Latin 1 character set to represent different scripts, characters, or symbols than those actually encoded in iso-8859-1.".

F Acknowledgements (Non-Normative)

Tim Berners-Lee and James Clark provided important details in the section on URIs. Asmus Freytag , Addison Phillips, and in early stages Ian Jacobs, provided significant help in the authoring and editing process. The W3C I18N WG and IG, as well as many others, provided many helpful comments and suggestions.