Internationalization (i18n) Activity

Making the World Wide Web worldwide!

Internationalization techniques: Developing specifications

This page provides checklists for specification developers, editors and reviewers who want to take account of internationalization issues while developing their spec. Where a checklist item is followed by a link, click on that for more information. The page also lists links to useful resources on the W3C Internationalization Activity site and elsewhere that may help.

Use of this checklist doesn't remove the need for a formal review, but warns of potential issues and requirements, at an early stage, that might otherwise be overlooked. Therefore, the later review should yield few, if any, nasty surprises. It also provides a useful set of points for reviewers to follow.

The checklist is currently still in development, and will be subject to regular changes. If you have a question or issue with some advice, please raise a Github issue.

This page is generated from the document Internationalization Best Practices for Spec Developers. It is just one of several techniques indexes, each of which focus on a particular type of user.

Collapse all • Expand all

Language

In this section

Language basics

How to's

Language basics

In Internationalization Best Practices for Spec Developers.

Background reading

Why use the language attribute?

Describes why it is useful to use the `lang` or `xml:lang` attribute to label language in web pages.

Use cases for language information in web annotations

Description of use cases for annotations that illustrate the differences between text-processing and metadata types of language declaration.

HTTP headers, meta elements and language information

How the distinction between text-processing

Best practices checklist

NA It should be possible to associate a language with any piece of natural language text that will be read by a user. *more*

NA Where possible, there should be a way to label natural language changes in inline text. *more*

Consider whether it is useful to express the intended linguistic audience of a resource, in addition to specifying the language used for text processing. *more*

A language declaration that indicates the text-processing language for a range of text must associate a single language value with a specific range of text. *more*

Use the HTML `lang` and XML `xml:lang` language attributes where appropriate, rather than creating a new attribute or mechanism. *more*

A metadata-type language declaration that indicates the intended use of the resource, rather than the language of a specific range of text, may be associated with multiple language values. *more*

language and language metadata plays out in HTML5.

Go to: top of this section • top of this page • techniques home page

Defining language values

How to's

Defining language values

In Internationalization Best Practices for Spec Developers.

BCP 47

The IETF specification that indicates how to create language subtags and how to match them.

Background reading

Language tags in HTML and XML

An overview of how to create language tags using BCP 47.

Best practices checklist

NA Values for language declarations must use BCP 47. *more*

Refer to BCP 47, not to RFC 5646. *more*

Be specific about what level of conformance you expect for language tags. The word "valid" has special meaning in BCP 47. Generally "well-formed" is a better choice.

Reference BCP47 for language tag matching.

Go to: top of this section • top of this page • techniques home page

Declaring language at the resource level

How to's

Declaring language at the resource level

In Internationalization Best Practices for Spec Developers.

Background reading

Use cases for

Best practices checklist

The specification should indicate how to define the default text-processing language for the resource as a whole. *more*

Content within the resource should inherit the language of the text-processing declared at the resource level, unless it is specifically overridden.

Consider whether it is necessary to have separate declarations to indicate the text-processing language versus metadata about the expected use of the

language information in web annotations

Description of use cases for annotations that illustrate the differences between text-processing and metadata types of language declaration.

HTTP headers, meta elements and language information

How the distinction between text-processing language and language metadata plays out in HTML5.

resource. *more*

If there is only one language declaration for a resource, and it has more than one language tag as a value, it must be possible to identify the default text-processing language for the resource. *more*

See also
Defining language values.

Go to: top of this section • top of this page • techniques home page

Establishing the language of a content block

How to's

Establishing the language of a content block

In Internationalization Best Practices for Spec Developers.

Best practices checklist

By default, blocks of content should inherit any text-processing language set for the resource as a whole. *more*

It should be possible to indicate a change in language for blocks of content where the language changes. *more*

See also
Defining language values.

Go to: top of this section • top of this page • techniques home page

Establishing the language of inline runs

How to's

- Establishing the language
  In Internationalization Best Practices for Spec Developers.

Best practices checklist

NA It should be possible to indicate language for spans of inline text where the language changes. *more*

See also
Defining language values.

Go to: top of this section • top of this page • techniques home page

Text direction

In this section

Basic requirements

Best practices checklist

NA It must be possible to indicate base direction for each individual paragraph-level item of natural language text that will be read by someone. *more*

NA It must be possible to indicate base direction changes for embedded runs of inline bidirectional text for all natural language text that will be read by someone. *more*

NA Annotating right-to-left text must require the

minimum amount of effort for people who work natively with right-to-left scripts. *more*

Go to: top of this section • top of this page • techniques home page

Background information

Best practices checklist

Do not assume that direction can be determined from language information. *more*

Go to: top of this section • top of this page • techniques home page

Handling direction in markup

How to's

- Estimation algorithms
  In Additional Requirements for Bidi in HTML & CSS.

Best practices checklist

NA The spec should indicate how to define a default base direction for the resource as a whole, ie. set the overall base direction. *more*

The default base direction, in the absence of other information, should be LTR. *more*

Values for the default base direction should include left-to-right, right-to-left, and auto. *more*

The content author must be able to indicate parts of the text where the base direction changes. At the block level, this should be achieved using attributes or metadata, and should not rely on Unicode control characters.

It must be possible to also set the direction for content fragments to `auto`. This means that the base direction will be determined by examining the content itself.

If the overall base direction is set to `auto` for plain text, the direction of content paragraphs should be determined on a paragraph by paragraph basis.

To indicate the sides of a block of text where relative to the start and end of its contained lines, you should use 'before' and 'after' (maybe block-start/block-end – the terminology is changing), rather than 'top' and

'bottom'.

To indicate the start/end of a line you should use 'start' and 'end' rather than 'left' and 'right'.

Provide dedicated attributes for control of base direction and bidirectional overrides; do not rely on the user applying style properties to arbitrary markup to achieve bidi control.

It must be possible to indicate spans of inline text where the base direction changes. If markup is available, this is the preferred method. Otherwise your specification must require that Unicode control characters are recognized by the receiving application, and correctly implemented.

It must be possible to also set the direction for a span to auto. This means that the base direction will be determined by examining the content itself. A typical approach here would be to set the direction based on the first strong directional character outside of any markup. *more*

If users use Unicode bidirectional control characters, the RLI/LRI/FSI with PDI characters must be supported by the application and recommended (rather than RLE/LRE with PDF) by the spec.

Use of RLM/LRM should be appropriate, and expectations of what those controls can and cannot do should be clear in the spec. *more*

Provide dedicated attributes for control of base direction and bidirectional overrides; do not rely on the user applying style properties to arbitrary markup to achieve bidi control.

Allow bidi attributes on all inline elements in markup that contain text.

Provide attributes that allow the user to (a) create an embedded base direction or (b) override the bidirectional algorithm altogether; the attribute should allow the user to set the direction to LTR or RTL in either of these two scenarios.

Go to: top of this section • top of this page • techniques home page

Characters

In this section

Choosing a definition of 'character'

How to's

Perceptions of Characters

In W3C Recommendation, Character Model for the World Wide Web.

See also
Defining 'string'.

Best practices checklist

√ Specifications, software and content *MUST NOT* require or depend on a one-to-one correspondence between characters and the sounds of a language. *more*

√ Specifications, software and content *MUST NOT* require or depend on a one-to-one mapping between characters and units of displayed text. *more*

√ Protocols, data formats and APIs *MUST* store, interchange or process text data in logical order. *more*

√ Independent of whether some implementation uses logical selection or visual selection, characters selected *MUST* be kept in logical order in storage. *more*

√ 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. *more*

√ 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. *more*

√ Specifications, software and content *MUST NOT* require or depend on a one-to-one relationship between characters and units of physical storage. *more*

√ When specifications use the term 'character' the specifications *MUST* define which meaning they intend. *more*

√ Specifications *SHOULD* use specific terms, when available, instead of the general term 'character'. *more*

Go to: top of this section • top of this page • techniques home page

Defining a Reference Processing Model

How to's

W3 Digital Encoding of Characters
In W3C Recommendation, Character Model for the World Wide Web.

See also
Including and excluding character ranges.

Best practices checklist

? Textual data objects defined by protocol or format specifications *MUST* be in a single character encoding. *more*

All specifications that involve processing of text *MUST* specify the processing of text according to the Reference Processing Model described by the rest of the recommendations in this list. *more*

X Specifications *MUST* define text in terms of Unicode characters, not bytes or glyphs. *more*

√ For their textual data objects specifications *MAY* allow use of any character encoding which can be transcoded to a Unicode encoding form. *more*

√ 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: (a) 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, (b) All processing *MUST* take place on this sequence of Unicode characters, (c) 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. *more*

NA 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. *more*

Go to: top of this section • top of this page • techniques home page

Including and excluding character ranges

How to's

Digital Encoding of Characters
In W3C Recommendation, Character Model for the World Wide Web.

See also
Using the Private Use Area.

Best practices checklist

√ Specifications *SHOULD NOT* arbitrarily exclude code points from the full range of Unicode code points from U+0000 to U+10FFFF inclusive. *more*

√ Specifications *MUST NOT* allow code points above U+10FFFF. *more*

√ Specifications *SHOULD NOT* allow the use of codepoints reserved by Unicode for internal use. *more*

√ Specifications *MUST NOT* allow the use of surrogate code points. *more*

√ Specifications *SHOULD* exclude compatibility characters in the syntactic elements (markup, delimiters, identifiers) of the formats they define. *more*

Go to: top of this section • top of this page • techniques home page

Using the Private Use Area

How to's

- Private use code points
  In W3C Recommendation, Character Model for the World Wide Web.

See also
Including and excluding character ranges.

Best practices checklist

√ Specifications *MUST NOT* require the use of private use area characters with particular assignments. *more*

√ Specifications *MUST NOT* require the use of mechanisms for defining agreements of private use code points. *more*

√ Specifications and implementations *SHOULD NOT* disallow the use of private use code points by private agreement. *more*

Specifications *MAY* define markup to allow the transmission of symbols not in Unicode or to identify specific variants of Unicode characters. *more*

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. *more*

Go to: top of this section • top of this page • techniques home page

Choosing character encodings

How to's

- Choice and identification of code points
  In W3C Recommendation, Character Model for the World Wide Web.

Background reading

- Document character set
  What is the 'Document

Best practices checklist

X Specifications *MUST* either specify a unique character encoding, or provide character encoding identification mechanisms such that the encoding of text can be reliably identified. *more*

When designing a new protocol, format or API, specifications *SHOULD* require a unique character encoding. *more*

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. *more*

Character Set' for XML and HTML, and how does it relate to the encodings I use for my documents?	X	When a unique character encoding is required, the character encoding *MUST* be UTF-8, UTF-16 or UTF-32. *more*
		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*. *more*
		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. *more*
		Character encodings that are not in the IANA registry *SHOULD NOT* be used, except by private agreement. *more*
	NA	If an unregistered character encoding is used, the convention of using 'x-' at the beginning of the name *MUST* be followed. *more*
	X	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). *more*
	X	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. *more*

Go to: top of this section • top of this page • techniques home page

Identifying character encodings

How to's **Best practices checklist**

Choice and identification of code points

In W3C Recommendation, Character Model for the World Wide Web.

√ Specifications *MUST NOT* propose the use of heuristics to determine the encoding of data. *more*

NA Specifications *MUST* define conflict-resolution mechanisms (e.g. priorities) for cases where there is multiple or conflicting information about character encoding. *more*

Go to: top of this section • top of this page • techniques home page

Designing character escapes

How to's

Character escaping

In W3C Recommendation, Character Model for the World Wide Web.

Best practices checklist

√ Specifications should provide a mechanism for escaping characters, particularly those which are invisible or ambiguous. *more*

Specifications *SHOULD NOT* invent a new escaping mechanism if an appropriate one already exists. *more*

The number of different ways to escape a character *SHOULD* be minimized (ideally to one). *more*

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. *more*

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. *more*

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. *more*

Go to: top of this section • top of this page • techniques home page

Storing text

How to's

Visual rendering and logical order

In W3C Recommendation, Character Model for the World Wide Web.

Best practices checklist

√ Protocols, data formats and APIs *MUST* store, interchange or process text data in logical order. *more*

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. *more*

Go to: top of this section • top of this page • techniques home page

Specifying sort and search functionality

How to's

Units of collation

In W3C Recommendation, Character Model for the World Wide Web.

Best practices checklist

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. *more*

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. *more*

Software that allows users to sort or search text *SHOULD* allow the user to select alternative rules for collation units and ordering. *more*

Specifications and implementations of sorting and searching algorithms *SHOULD* accommodate text that contains any character in Unicode. *more*

Go to: top of this section • top of this page • techniques home page

Converting to a Common Unicode Form

How to's

Best practices checklist

Converting to a Common Unicode Form

In W3C Working Draft, Character Model for the World Wide Web: String Matching and Searching.

Specifications of text-based formats and protocols *MAY* specify that all or part of the textual content of that format or protocol is normalized using Unicode Normalization Form C (NFC). *more*

NA

Specifications that do not normalize *MUST* document or provide a health-warning if canonically equivalent but disjoint Unicode character sequences represent a security issue. *more*

Specifications and implementations *MUST NOT* assume that content is in any particular normalization form. *more*

NA

Specifications *MUST* specify that string matching takes the form of "code point-by-code point" comparison of the Unicode character sequence, or, if a specific Unicode character encoding is specified, code unit-by-code unit comparison of the sequences. *more*

NA

Specifications that define a regular expression syntax *MUST* provide at least Basic Unicode Level 1 support per *Unicode Technical Standard #18: Unicode Regular Expressions* and *SHOULD* provide Extended or Tailored (Levels 2 and 3) support. *more*

X

Specifications of text-based formats and protocols that, as part of their syntax definition, require that the text be in normalized form *MUST* define string matching in terms of normalized string comparison and *MUST* define the normalized form to be NFC. *more*

A normalizing text-processing component which receives suspect text *MUST NOT* perform any normalization-sensitive operations unless it has first either confirmed through inspection that the text is in normalized form or it has re-normalized the text itself. Private agreements *MAY*, however, be created within private systems which are not subject to these rules, but any externally observable results *MUST* be the same as if the rules had been obeyed. *more*

Specifications of text-based languages and protocols *SHOULD* define precisely the construct boundaries necessary to obtain a complete definition of full-normalization. These definitions *SHOULD* include at least the boundaries between markup and

character data as well as entity boundaries (if the language has any include mechanism) , *SHOULD* include any other boundary that may create denormalization when instances of the language are processed, but *SHOULD NOT* include character escapes designed to express arbitrary characters. *more*

Where operations can produce denormalized output from normalized text input, specifications of API components (functions/methods) that implement these operations *MUST* define whether normalization is the responsibility of the caller or the callee. Specifications *MAY* state that performing normalization is optional for some API components; in this case the default *SHOULD* be that normalization is performed, and an explicit option *SHOULD* be used to switch normalization off. Specifications *SHOULD NOT* make the implementation of normalization optional. *more*

Specifications that define a mechanism (for example an API or a defining language) for producing textual data object *SHOULD* require that the final output of this mechanism be normalized. *more*

Go to: top of this section • top of this page • techniques home page

Handling Case Folding

How to's

Handling Case Folding

In W3C Working Draft, Character Model for the World Wide Web: String Matching and Searching.

Best practices checklist

√ Case sensitive matching is *RECOMMENDED* as the default for new protocols and formats. *more*

Because the "simple" case-fold mapping removes information that can be important to forming an identity match, the "Common plus Full" (or "Unicode C+F") case fold mapping is *RECOMMENDED* for Unicode case-insensitive matching. *more*

ASCII case-insensitive matching *MUST* only be applied to vocabularies that are restricted to ASCII. Unicode case-insensitivity *MUST* be used for all other vocabularies. *more*

If the vocabulary is not restricted to ASCII or permits user-defined values that use a broader range of

Unicode, ASCII case-insensitive matching *MUST NOT* be required. *more*

The Unicode C+F case-fold form is *RECOMMENDED* as the case-insensitive matching for vocabularies. The Unicode C+S form *MUST NOT* be used for string identity matching on the Web. *more*

Specifications and implementations that define string matching as part of the definition of a format, protocol, or formal language (which might include operations such as parsing, matching, tokenizing, etc.) *MUST* define the criteria and matching forms used. These *MUST* be one of: (a) Case-sensitive (b) Unicode case-insensitive using Unicode case-folding C+F (c) ASCII case-insensitive. *more*

Specifications *SHOULD NOT* specify case-insensitive comparison of strings. *more*

Specifications that specify case-insensitive comparison for non-ASCII vocabularies *SHOULD* specify Unicode case-folding C+F. *more*

Specifications *MAY* specify ASCII case-insensitive comparison for portions of a format or protocol that are restricted to an ASCII-only vocabulary. *more*

Specifications and implementations *MUST NOT* specify ASCII-only case-insensitive matching for values or constructs that permit non-ASCII characters. *more*

Go to: top of this section • top of this page • techniques home page

Defining 'string'

How to's

String concepts

In W3C Recommendation, Character Model for the World Wide Web.

Best practices checklist

Specifications *SHOULD NOT* define a string as a 'byte string'. *more*

The 'character string' definition *SHOULD* be used by most specifications. *more*

See also

Indexing strings and Choosing a

definition of 'character'.

Go to: top of this section • top of this page • techniques home page

Indexing strings

How to's

String indexing

In W3C Recommendation, Character Model for the World Wide Web.

See also
Defining 'string'.

Best practices checklist

NA The character string is *RECOMMENDED* as a basis for string indexing. *more*

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. *more*

Grapheme clusters *MAY* be used as a basis for string indexing in applications where user interaction is the primary concern. *more*

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. *more*

The use of byte strings for indexing is *NOT RECOMMENDED*. *more*

Specifications that need a way to identify substrings or point within a string *SHOULD* provide ways other than string indexing to perform this operation. *more*

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. *more*

Specifications of APIs *SHOULD NOT* specify single characters or single 'units of encoding' as argument or return types. *more*

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. *more*

Go to: top of this section • top of this page • techniques home page

Referring to Unicode characters

Best practices checklist

Use U+XXXX syntax to represent Unicode code points in the specification. *more*

Go to: top of this section • top of this page • techniques home page

Referencing the Unicode Standard

How to's

- Referencing the Unicode Standard and ISO/IEC 10646
  In W3C Recommendation, Character Model for the World Wide Web.

Best practices checklist

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. *more*

X 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. *more*

X All generic references to the Unicode Standard *MUST* refer to the latest version of the Unicode Standard available at the date of publication of the containing specification. *more*

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

Go to: top of this section • top of this page • techniques home page

Resource identifiers

Basics

How to's

- Internationalized Resource Identifiers (IRIs)

Best practices checklist

? Resource identifiers must permit the use of characters outside those of plain ASCII. *discussion*

X Specifications *MUST* define when the conversion from IRI references to URI references (or subsets thereof) takes place, in accordance with Internationalized Resource Identifiers (IRIs). *more*

Go to: top of this section • top of this page • techniques home page

Markup & syntax

In this section

Defining elements and attributes

Best practices checklist

√ Do not define attribute values that will contain user readable content. Use elements for such content. *more*

NA If you do define attribute values containing user readable content, provide a means to indicate directional and language information for that text separately from the text contained in the element.

Provide a way for authors to annotate arbitrary inline

content using a `span`-like element or construct.
more

Go to: top of this section • top of this page • techniques home page

Defining identifiers

Best practices checklist

Identifiers should be case-sensitive.

Go to: top of this section • top of this page • techniques home page

Working with plain text

Best practices checklist

√ Avoid natural language text in elements that only allow for plain text and in attribute values.

Provide a span-like element that can be used for any text content to apply information needed for internationalization. *more*

Go to: top of this section • top of this page • techniques home page

Typographic support

In this section

Enabling vertical text display

Best practices checklist

NA It should be possible to render text vertically for languages such as Japanese, Chinese, Korean, Mongolian, etc.

Vertical text must support line progression from LTR (eg. Mongolian) and RTL (eg. Japanese)

Go to: top of this section • top of this page • techniques home page

Setting box positioning coordinates when text direction varies

Best practices checklist

NA Box positioning coordinates must take into account whether the text is horizontal or vertical. *more*

Go to: top of this section • top of this page • techniques home page

Miscellaneous

Best practices checklist

NA Line heights must allow for characters that are taller than English.

Box sizes must allow for text expansion in translation.

Ruby text alongside base text should be supported for CJK text.

Line wrapping should take into account the special rules needed for non-Latin scripts. *more*

Avoid specifying presentational tags, such as `b` for bold, and `i` for italic. *more*

Go to: top of this section • top of this page • techniques home page

Local dates, times and formats

In this section

Working with time

Best practices checklist

?

When defining calendar and date systems, be sure to allow for dates prior to the common era, or at least define handling of dates outside the most common range.

When defining time or date data types, ensure that the time zone or relationship to UTC is always defined.

Provide a health warning for conversion of time or date data types that are "floating" to/from incremental types, referring as necessary to the *Time Zones WG Note*. *more*

Allow for leap seconds in date and time data types. *more*

Use consistent terminology when discussing date and time values. Use 'floating' time for time zone independent values.

Keep separate the definition of time zone from time zone offset.

Use IANA time zone IDs to identify time zones. Do not use offsets or LTO as a proxy for time zone.

Use a separate field to identify time zone.

When defining rules for a "week", allow for culturally specific rules to be applied. *more*

When defining rules for week number of year, allow for culturally specific rules to be applied.

When non-Gregorian calendars are permitted, note that the "month" field can go to 13 (undecimber).

Go to: top of this section • top of this page • techniques home page

Designing forms

Best practices checklist

NA When defining email field validation, allow for EAI (smtputf8) names.

Go to: top of this section • top of this page • techniques home page

Working with numbers

Best practices checklist

NA When parsing user input of numeric values, allow for digit shaping (non-ASCII digits).

When formatting numeric values for display, allow for culturally sensitive display, including the use of non-ASCII digits (digit shaping).

Go to: top of this section • top of this page • techniques home page

In this section

Providing for content negotiation based on language

How to's

Providing for content negotiation based on language
In Internationalization Best Practices for Spec Developers.

Best practices checklist

NA In a multilingual environment it must be possible for the user to receive text in the language they prefer. This may depend on implicit user preferences based on the user's system or browser setup, or on user settings explicitly negotiated with the user.

Go to: top of this section • top of this page • techniques home page

Links in this document:

Last changed 2016-01-27 16:47 GMT.