ecma INTERNATIONAL

Ecma/TC39/2011/NN

1st Draft Standard ECMA-XXX

Edition / Date

ECMAScript Internationalization API Specification

Reference number
ECMA-123:2009

Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001

Contents

Page

Introduction

This Ecma Standard is based on several existing internationalization technologies, the most well known being ICU library for C and Java (open source) and Microsoft Windows APIs.

The development of this standard started in September 2010. It is not yet adopted by the Ecma General Assembly.

Initial implementations can be found in Chrome browser and in Amazon Kindle e-book reader.

ECMAScript internationalization API is a vibrant library and its evolution is not complete. Significant technical enhancement will continue with future editions of this specification.

This Ecma Standard has been adopted by the General Assembly of <month> <year>.

ECMAScript Internationalization API Specification

1 Scope

This Standard defines ECMAScript Internationalization API.

2 Conformance

A conforming implementation of ECMAScript Internationalization API must provide and support all the types, values, objects, properties, functions and semantics described in this specification.

A conforming implementation of ECMAScript i18n API is permitted to support objects, functions and parameters not described in this specification. In particular, a conforming implementation of ECMAScript i18n API is permitted to support optional parameters listed in TODO(cira): (add section for dateformat and nubmer format optional parameters) of this specification.

3 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Unicode Technical Standard 35, *Unicode Locale Data Markup Language*

ECMA-262, *ECMAScript Language Specification*, 6th edition

4 Overview

This section contains a non-normative overview of the ECMAScript i18n API.

ECMAScript i18n API enables development of locale (language and region) aware applications. It is based on current best practices in internationalization community and it's aim is to improve current built in capabilities of ECMAScript language and to fill in the gap between actual user needs and what is offered in current third party i18n libraries.

The goals are:

- Consistent, rich i18n API, similar to what modern OS/i18n libraries have today for Python, C++ and Java.
- Consistent i18n results (not 100% identical but comparable to the data collected in CLDR or locale support on major platforms).
- Support for multiple locales per application.
- Reuse of objects that are more expensive to create, like collators.
- Off-line processing.

4.1 API Overview

The following is an informal overview of ECMAScript i18n API—not all parts of the API are described. This overview is not part of the standard proper.

The API is based on top level LocaleInfo object that serves both as a namespace and as a locale descriptor. LocaleInfo holds information about users' language and region, and that information is used throughout the API.

Collation is locale aware sorting and it's one of unsolved problems in ECMAScript internationalization effort. Data set required for some of the Asian locales is prohibitively large so most i18n libraries avoid implementing collation. Proper collation is becoming increasingly important with online and mobile services – sorting contact lists, dictionaries, phonebooks, number lists etc. – so it needs to be solved.

Date, time and number formatting lets you format data in locale acceptable format, with more or less details, e.g. 11/08/11 or August, 11th 2011. It is also possible to get date related symbols like month, week, day and era names in various widths, e.g. J, Jan or January, which is important in mobile development.

4.2 Object construction

The following is an informal overview of how API objects are created — not all parts of construction process are described. This overview is not part of the standard proper.

LocaleInfo object is the only public object in the API and is constructed using new statement. Collator and formatter objects are private and are constructed using a factory methods on LocaleInfo instance. Both constructor and factory methods accept **settings** object which contains key/value pairs necessary for successful construction of the given object. LocaleInfo constructor may accept simple string parameter containing the proper language identifier.

Once object is constructed it becomes **immutable** and exposes **derive** method, other methods related to its domain, e.g. format(), and **options** property.

Derive method lets us clone given object and possibly overwrite some of the settings. It's a useful approach for objects that take large number of parameters, e.g. collators.

Options property contains resolved settings passed in during construction. Settings that were not recognised are skipped, and ones used for construction are canonicalized and possibly resolved to the best match on the given platform. Some key/value pairs are inferred if they were not passed in in settings object.

4.3 Definitions

For the purposes of this document, the following terms and definitions apply.

4.3.1
Language identifier
Identifies language and related data as spoken in given region. It is defined in TR#35, section 3: Unicode Language and Locale Identifiers.

Difference between locale and language identifier is described in TR#35, Appendix D: Unicode Language and Locale Ids.

4.3.2
Locale identifier
Consists of language identifier and extensions, e.g -u-co-phonebk.

4.3.3
Region identifier
Region identifier selects currency code and measurement units. It's orthogonal to locale identifier. Covered in ISO 3166 document and LDLM document.

4.3.4
Currency code
Identifies currency code used in currency formatting. Defined in ISO 4217, currency and funds name and code elements.

5 Application Programming Interface

5.1 Locale support

LocaleInfo class is a global object and acts as an aggregator for other classes. This approach avoids pollution of global namespace and makes it easy to pass locale data around.

5.1.1 The LocaleInfo constructor

When **LocaleInfo** is called as a part of **new** expression, it is a constructor: it initialises newly created object.

5.1.1.1 new LocaleInfo()

LocaleInfo object is created using default locale identifier – implementation specific value.

Created object has **options.localeID** set to default locale identifier, and **options.regionID** is set to inferred region identifier value – inferred from options.localeID. If the value of options.regionID can't be inferred it is set to **ZZ**.

The [[Prototype]] internal property of the newly constructed object is set to the original LocaleInfo prototype object, the one that is the initial value of **`LocaleInfo.prototype`** (5.1.2.1).

The [[Class]] internal property of the newly constructed object is set to **`"LocaleInfo"`**.

NOTE It is recommended to use **window.navigator.language** as default locale identifier in browser environment.

5.1.1.2 new LocaleInfo(localeID)

LocaleInfo object is created using a single string argument which is a valid LDLM locale identifier.

If localeID is an invalid LDLM locale identifier an **"Invalid locale identifier specified"** exception is thrown.

A best match between specified and supported locale identifier(s) is to be found. Only language identifier part is to be used for best match search, but original locale extensions should be preserved in the final result.

Created object has **options.localeID** set to best match language identifier with original extensions preserved, and **options.regionID** is set to inferred region identifier value – inferred from options.localeID. If the value of options.regionID can't be inferred it is set to **ZZ**.

Valid unicode extensions are defined in LDLM document, in section Key/Type definitions. This standard uses only -co- for collation and -cu- for currency specification. Other extensions are optional and should be ignored, but implementations are free to support them.

The [[Prototype]] internal property of the newly constructed object is set to the original LocaleInfo prototype object, the one that is the initial value of **`LocaleInfo.prototype`** (5.1.2.1).

The [[Class]] internal property of the newly constructed object is set to **`"LocaleInfo"`**.

5.1.1.2.1 Best match algorithm

Best match algorithm cannot fail and in the worst case is allowed to fall back to a default locale. In best case it selects exact match as a result. The actual implementation of the algorithm is implementation specific.

Example:

```
requested [A, B], supports {A, B}, pick A, order breaks tie.
requested [A, B], supports {A', B}, pick B. A' is a near perfect match.
requested [A, B], supports {A', B'}, implementation dependent.
```

5.1.1.3 new LocaleInfo(settings)

LocaleInfo object is created using settings parameter (5.1.1.3.1).

Created object has **options.localeID** set tp best match language identifier with original extensions preserved, and **options.regionID** is set to either **settings.regionID** or inferred region identifier value – inferred from options.localeID. If the value of options.regionID can't be inferred it is set to **ZZ**.

Valid unicode extensions are defined in LDLM document, in section Key/Type definitions. This standard uses only -co- for collation and -cu- for currency specification. Other extensions are optional and should be ignored, but implementations are free to support them.

The [[Prototype]] internal property of the newly constructed object is set to the original LocaleInfo prototype object, the one that is the initial value of **`LocaleInfo.prototype`** (5.1.2.1).

The [[Class]] internal property of the newly constructed object is set to **`"LocaleInfo"`**.

5.1.1.3.1 Settings parameter

Settings object helps avoid possible future changes to LocaleInfo constructor signature by encapsulating all parameters into one object.

Settings object has two properties, **localeID** and **regionID**.

Required localeID parameter is either a string (see 5.1.1.2) or a priority list of LDML locale identifiers. If any of the elements of the priority list is not a string or is not a valid LDLM identifier an "**Invalid locale identifier specified**" exception is thrown. Order in the priority list is used only for breaking ties between two matches (selecting item closer to the beginning of the list). Exact match should always be selected as the best match.

Optional regionID parameter is a string. Region identifier specifies region to be used for currency handling and selecting proper units of measurement. It is a two letter region code as defined in LDLM document, section "Language/Locale Field Definitions**"**. The value of "ZZ" means undefined or invalid territory.

5.1.2 Properties of LocaleInfo constructor

The value of the [[Prototype]] internal property of the LocaleInfo constructor is the Function prototype object.

Besides the internal properties, the LocaleInfo constructor has the following properties:

5.1.2.1 LocaleInfo.prototype

The initial value of **`LocaleInfo.prototype`** is the LocaleInfo prototype object (5.1.3).

This property has the attributes { [[Writable]]: **false**, [[Enumerable]]: **false**, [[Configurable]]: **false** }.

5.1.2.2 LocaleInfo.options

Each LocaleInfo object stores best matched locale identifier into **options.localeID** and user supplied regionID or inferred regionID into **options.regionID**.

Both properties are canonicalized to match LDML definition, as in section Language/Locale field definitions.

5.1.2.3 LocaleInfo.derive(settings)

This function creates a new LocaleInfo object, or returns a cached copy that has the same settings. It is a convenience method that makes creation of similar objects easier.

It takes **LocaleInfo.options** and **settings** and generates new settings. Finally it invokes LocaleInfo constructor with a newly created settings object.

All applicable properties from **settings** parameter override corresponding properties in **LocaleInfo.options** in the new settings object. To unset a property **x** one needs to set **settings.x** to **undefined** value.

5.1.3 Properties of the LocaleInfo Prototype Object

The LocaleInfo prototype object is itself a LocaleInfo object (its [[Class]] is **`"LocaleInfo"`**).

The value of the [[Prototype]] internal property of the LocaleInfo prototype object is the standard built-in Object prototype object.

In following descriptions of functions that are properties of the LocaleInfo prototype object, the phrase “this LocaleInfo object” refers to the object that is the **this** value for the invocation of the function.

5.1.3.1 LocaleInfo.prototype.constructor

The initial value of **`LocaleInfo.prototype.constructor`** is the **`LocaleInfo`** constructor.

5.1.3.2 LocaleInfo.prototype.collator (settings)

This function returns a collator object based on this LocaleInfo object and supplied settings. Content of the collator object is listed in 5.2.

Function returns either a new collator object or a cached copy.

5.1.3.3 LocaleInfo.prototype.numberFormat (settings)

This function returns a number formatter object based on this LocaleInfo object and supplied settings. Content of the number formatter object is listed in 5.3.

Function returns either a new number formatter object or a cached copy.

5.1.3.4 LocaleInfo.prototype.dateTimeFormat (settings)

This function returns a date time formatter object based on this LocaleInfo object and supplied settings. Content of the date time formatter object is listed in 5.4.

Function returns either a new date time formatter object or a cached copy.

5.2 Collation

5.2.1 Collator constructor

When **LocaleInfo.__Collator** is called as a part of **new** expression, it is a constructor: it initialises newly created object. This constructor is not part of the public API. Users of the API should call `LocaleInfo.prototype.collator()` instead.

5.2.1.1 new LocaleInfo.__Collator(settings, localeInfo)

This constructor creates a new LocaleInfo.__Collator object based on supplied collation settings and localeInfo object.

5.2.1.1.1 Settings parameter

Settings object helps avoid possible future changes to `LocaleInfo.prototype.collator(settings)` factory signature by encapsulating all parameters into one object.

Settings object has three properties -- **numeric**, **ignorePunctuation** and **sensitivity**.

Optional boolean-valued **numeric** property can have true, false or undefined values. If set to true, numbers in strings are treated as numbers not strings, in which case “9” < “12”. If set to false, numbers in strings are treated as plain strings, in which case “12” < “9”. If set to undefined then default locale preference should be taken (TODO – clarify why we need this, if we need it). If a non-boolean value is specified an “**Invalid numeric flag value is specified**” exception is thrown.

Optional boolean-valued **ignorePunctuation** property can have true, false and undefined values. If set to true, punctuation characters are ignored when comparing strings. If set to false punctuation characters are taken into account when comparing strings. Value of undefined tells us to use locale default setting (TODO – do we need undefined?). If a non-boolean value is specified an “**Invalid ignorePunctuation flag value is specified**” exception is thrown.

Required string-valued **sensitivity** property can have values as shown in table 1. If a non-string value, or a non-listed string value is specified an “**Invalid sensitivity value specified**” exception is thrown.

Table 1 – Sensitivity key/value pairs and examples

Value (string)	Description	Strength	Example (for sorting)
base	Only base letter differences	S = P	a < b, a == á, a == A
accent	Honors accents	S = S,	a < b, a < á, a == A
case	Case sensitive	S = P, CL = T	a < b, a == á, a < A
variants	Includes case, accents and width	S = Q, HL = T for ja locale	ja cases* a < b, a < á, a < A
default	Locale default		

Exact ordering of characters depends on locale, so a < A is illustrative.

* Illustrating Japanese exceptions (only for sensitivity: variants).

S = S (all locales)	ぁ == あ == ア
S = T (non-ja)	ぁ < あ < ア
S = T (ja)	ぁ < あ == ア
S = Q (ja)	ぁ < あ < ア

Collation strength (S) is explained in Unicode Technical Report #10 (UTS#10) document. Actual collation algorithm is implementation specific.

All default values are taken from **localeInfo** object.

5.2.1.2 new LocaleInfo.__Collator(localeInfo)

Default values for numeric, ignorePunctuation and sensitivity are taken from **localeInfo** object.

5.2.1.3 LocaleInfo.__Collator.options

This property contains recognised and resolved properties from the settings parameter.

If numeric property was specified as an input, **LocaleInfo.__Collator.options.numeric** is set to its value.

If ignorePunctuation property was specified as an input, **LocaleInfo.__Collator.options.ignorePunctuation** is set to its value.

LocaleInfo.__Collator.options.sensitivity is set to settings.sensitivity value, or "**default**" if empty constructor was used.

5.2.1.4 LocaleInfo.__Collator.derive(settings)

This function creates a new LocaleInfo.__Collator object, or returns a cached copy that has the same settings. It is a convenience method that makes creation of similar objects easier.

It takes **LocaleInfo.__Collator.options** and **settings** and generates new settings. Finally it invokes LocaleInfo.__Collator constructor with a newly created settings object.

All applicable properties from **settings** parameter override corresponding properties in **LocaleInfo.__Collator.options** in the new settings object. To unset a property **x** one needs to set **settings.x** to **undefined** value.

5.2.2 Properties of the LocaleInfo.__Collator Prototype Object

The LocaleInfo prototype object is itself a LocaleInfo object (its [[Class]] is **`"LocaleInfo.Collator"`**).

The value of the [[Prototype]] internal property of the LocaleInfo.__Collator prototype object is the standard built-in Object prototype object.

In following descriptions of functions that are properties of the LocaleInfo.__Collator prototype object, the phrase "this Collator object" refers to the object that is the **this** value for the invocation of the function.

5.2.2.1 LocaleInfo.__Collator.prototype.constructor

The initial value of **`LocaleInfo.__Collator.prototype.constructor`** is the **`LocaleInfo.__Collator`** constructor.

5.2.2.2 LocaleInfo.__Collator.prototype.compare (a, b)

This method compares strings **a** and **b** in a locale sensitive way. The result is intended to order String values in the sort order specified by the locale passed to the constructor, and will be negative, zero, or positive, depending on whether **a** comes before **b** in the sort order, the Strings are equal, or **a** comes after **b** in the sort order, respectively.

5.3 Number formatting

5.3.1 Number formatter constructor

When __**NumberFormat** is called as a part of **new** expression, it is a constructor: it initialises newly created object. This constructor is not part of the public API. Users of the API should call `LocaleInfo.prototype.numberFormat()` instead.

5.3.1.1 new LocaleInfo.__NumberFormat(settings, localeInfo)

The localeInfo parameter supplies locale info to the number formatter.

5.3.1.1.1 Settings parameter

Settings object helps avoid possible future changes to `LocaleInfo.prototype.numberFormat(settings)` factory signature by encapsulating all parameters into one object.

Settings object has four properties -- **currencyCode**, **style**, **pattern** and **skeleton**.

Optional string-valued **currencyCode** property defines currency code to be used when formatting currencies. Its value is defined by ISO 4217 standard. If specified it overrides any other currency hint, like locale identifier -u-cu- value or currency code inferred from the regionID.

Optional string-valued **style** property defines one of the supported number formatting styles – decimal, currency and percent. Scientific style is optional with value – scientific, and should be ignored if implementation doesn't support it. If invalid style value is specified an "**Invalid number style specified**" exception is thrown.

Optional string-valued **pattern** property defines a number format using a pattern. Pattern format is defined in UTS#35: Appendix G: Number Format Patterns.

Optional string-valued **skeleton** property defines a number format using a best match pattern that corresponds to the given skeleton. Best match algorithm is implementation specific. There is a ICU proposal for skeleton support – http://site.icu-project.org/design/formatting/numbers/skeleton. It it's not supported by an implementation it should be ignored.

If none of skeleton, pattern or style properties are specified an "**No number format style specified**" exception is thrown.

If more than one format descriptor is specified then the **skeleton** property overrides any of **pattern** or **style** properties. The **pattern** property overrides the **style** property.

5.3.1.2 new LocaleInfo.__NumberFormat(localeInfo)

Creates a new LocaleInfo.__NumberFormat object, with number format set to "**decimal**" style.

5.3.1.3 LocaleInfo.__NumberFormat.options

This property contains recognised and resolved properties from the settings parameter.

The **LocaleInfo.__NumberFormat.options.currencyCode** is set to either settings.currencyCode property value or to an inferred currency code from the localeInfo.options.regionID. The final value is canonicalized – 3-letter, uppercased ASCII currency code.

If skeleton property was specified as an input, **LocaleInfo.__NumberFormat.options.pattern** is set to the best match pattern. Other number format descriptors (style or pattern) are ignored.

If pattern property was specified as an input (but not skeleton), **LocaleInfo.__NumberFormat.options.pattern** is set to that pattern. The style property, if specified, is ignored.

If style property was specified as an input (but not skeleton or pattern), **LocaleInfo.__NumberFormat.options.style** is set to that style.

5.3.1.4 LocaleInfo.__NumberFormat.derive(settings)

This function creates a new LocaleInfo.__NumberFormat object, or returns a cached copy that has the same settings. It is a convenience method that makes creation of similar objects easier.

It takes **LocaleInfo.__NumberFormat.options** and **settings** and generates new settings. Finally it invokes LocaleInfo.__NumberFormat constructor with a newly created settings object.

All applicable properties from **settings** parameter override corresponding properties in **LocaleInfo.__NumberFormat.options** in the new settings object. To unset a property **x** one needs to set **settings.x** to **undefined** value.

5.3.2 Properties of the LocaleInfo.__NumberFormat Prototype Object

The LocaleInfo.__NumberFormat prototype object is itself a LocaleInfo.__NumberFormat object (its [[Class]] is **`"LocaleInfo.NumberFormat"`**).

The value of the [[Prototype]] internal property of the LocaleInfo.__NumberFormat prototype object is the standard built-in Object prototype object.

In following descriptions of functions that are properties of the LocaleInfo.__NumberFormat prototype object, the phrase “this NumberFormat object” refers to the object that is the **this** value for the invocation of the function.

5.3.2.1 LocaleInfo.__NumberFormat.prototype.constructor

The initial value of **`LocaleInfo.__NumberFormat.prototype.constructor`** is the **`LocaleInfo.__NumberFormat`** constructor.

5.3.2.2 LocaleInfo.__NumberFormat.prototype.format (value)

This method returns a string with **value** formatted using specified number format. If **value** is **NaN** method returns “**NaN**”. If **value** is a non-number an “**Invalid numeric value specified**” exception is thrown.

5.4 Date and time formatting

5.4.1 Date and time formatter constructor

When **__DateTimeFormat** is called as a part of **new** expression, it is a constructor: it initialises newly created object. This constructor is not part of the public API. Users of the API should call **`LocaleInfo.prototype.dateTimeFormat()`** instead.

5.4.1.1 new LocaleInfo.__DateTimeFormat(settings, localeInfo)

The localeInfo parameter supplies locale info to the date time formatter.

5.4.1.1.1 Settings parameter

Settings object helps avoid possible future changes to `LocaleInfo.prototype.dateTimeFormat(settings)` factory signature by encapsulating all parameters into one object.

Settings object has three properties -- **timeStyle**, **dateStyle** and **skeleton**.

Optional string-valued **timeStyle** property defines one of the supported time formatting styles – **short** and **long**. Support for **medium** and **full** styles is **optional**. Implementation that doesn't support optional values should ignore them. If invalid timeStyle value is specified an "**Invalid time style specified**" extension is thrown.

Optional string-valued **dateStyle** property defines one of the supported date formatting styles – **short** and **medium**. Support for **long** and **full** styles is **optional**. Implementation that doesn't support optional values should ignore them. If invalid dateStyle value is specified an "**Invalid date style specified**" extension is thrown.

Optional string-valued **skeleton** property defines a date time format using a best match pattern that corresponds to the given skeleton. Best match algorithm is implementation specific. If skeleton is not supported by an implementation it should be ignored. Skeleton format is specified in UTS#35: Appendix F: Date Format Patterns. See table 2 for supported skeletons.

If none of skeleton, timeStyle or dateStyle properties are specified an "**No date time format style specified**" exception is thrown.

If more than one format descriptor is specified then the **skeleton** property overrides any of **dateStyle** and **timeStyle** properties. The **dateStyle** and **timeStyle** can be specified together – the ordering of the two will be locale specific.

Various calendars are supported by Unicode locale identifier extension **-u-ca-**. Calendars that are not supported should be ignored.

Table 2 – Supported skeleton patterns

Skeleton	Description
yMd	Same as short dateStyle.
yyyyMMMMdEEEE	Same as long dateStyle, includes day of the week.
yyyyMMMMd	Same as long dateStyle, exclude day of the week, and include era if necessary.
yMMM	Year/month pattern.
MMMd	Month/day pattern.

5.4.1.2 new LocaleInfo.__DateTimeFormat(localeInfo)

Creates a new LocaleInfo.__DateTimeFormat object, with dateStyle set to "**short**", timeStyle set to "**short**" and locale defaults from localeInfo object.

5.4.1.3 LocaleInfo.__DateTimeFormat.options

This property contains recognised and resolved properties from the settings parameter.

If skeleton property was specified as an input, **LocaleInfo.__NumberFormat.options.pattern** is set to the best match pattern. Other number format descriptors (style or pattern) are ignored.

If dateStyle property was specified as an input (but not skeleton), **LocaleInfo.__DateTimeFormat.options.dateStyle** is set to that style.

If timeStyle property was specified as an input (but not skeleton), **LocaleInfo.__DateTimeFormat.options.timeStyle** is set to that style.

5.4.1.4 LocaleInfo.__DateTimeFormat.derive(settings)

This function creates a new LocaleInfo.__DateTimeFormat object, or returns a cached copy that has the same settings. It is a convenience method that makes creation of similar objects easier.

It takes **LocaleInfo.__DateTimeFormat.options** and **settings** and generates new settings. Finally it invokes LocaleInfo.__DateTimeFormat constructor with a newly created settings object.

All applicable properties from **settings** parameter override corresponding properties in **LocaleInfo.__DateTimeFormat.options** in the new settings object. To unset a property **x** one needs to set **settings.x** to **undefined** value.

5.4.2 Properties of the LocaleInfo.__DateTimeFormat Prototype Object

The LocaleInfo.__DateTimeFormat prototype object is itself a LocaleInfo.__DateTimeFormat object (its [[Class]] is `"LocaleInfo.DateTimeFormat"`).

The value of the [[Prototype]] internal property of the LocaleInfo.__DateTimeFormat prototype object is the standard built-in Object prototype object.

In following descriptions of functions that are properties of the LocaleInfo.__DateTimeFormat prototype object, the phrase "this DateTimeFormat object" refers to the object that is the **this** value for the invocation of the function.

5.4.2.1 LocaleInfo.__DateTimeFormat.prototype.constructor

The initial value of `LocaleInfo.__DateTimeFormat.prototype.constructor` is the `LocaleInfo.__DateTimeFormat` constructor.

5.4.2.2 LocaleInfo.__DateTimeFormat.prototype.format (date)

This method returns a string with **date** formatted using specified date time format. If **date** is missing method returns current time and date. If **date** is a non-Date an "**Invalid date specified**" exception is thrown.

5.4.2.3 LocaleInfo.__DateTimeFormat.prototype.getMonths (width)

This method returns an Array of months, translated to match current locale. The **width** parameter specifies the width of month names – **abbreviated** and **wide**. The value **narrow** is optional, and implementation can ignore it.

If width is omitted use width: **'wide'.** If width value is not supported an “**Invalid width specified**” exception is thrown.

Example of abbreviated names: [Jan, Feb, Mar,...]. Wide months: [January, February, March,...]. Narrow months: [J, F, M,...].

5.4.2.4 LocaleInfo.__DateTimeFormat.prototype.getWeekdays (width)

This method returns an Array of week days, translated to match current locale. The **width** parameter specifies the width of day names – **abbreviated** and **wide**. The value **narrow** is optional, and implementation can ignore it.

If width is omitted use width: **'wide'.** If width value is not supported an “**Invalid width specified**” exception is thrown.

5.4.2.5 LocaleInfo.__DateTimeFormat.prototype.getEras (width)

This method returns an Array of eras, translated to match current locale. The **width** parameter specifies the width of era names – **abbreviated** and **wide**. The value **narrow** is optional, and implementation can ignore it.

If width is omitted use width: **'wide'.** If width value is not supported an “**Invalid width specified**” exception is thrown.

5.4.2.6 LocaleInfo.__DateTimeFormat.prototype.getAmPm (width)

This method returns an Array of day periods, translated to match current locale. The **width** parameter specifies the width of day periods – **abbreviated** and **wide**. The value **narrow** is optional, and implementation can ignore it.

If width is omitted use width: **'wide'.** If width value is not supported an “**Invalid width specified**” exception is thrown.

Annex A
(normative)100

Annex title

Text text text

Annex B
(informative)
200
Annex title

Text text text

Bibliography (if any)

[1] Experimental statistics, US National Bureau of Standards Handbook 91, 1963

[2] Applied Regression Analysis, Draper and Smith, Wiley Edition 2

[3] Statistical Methods for Reliability Data, Meeker, Escobar, 1998, John Wiley & Sons Inc.