Deleted: <sp>

Ecma/TC39/2012/NN

Deleted: <sp><sp><sp><sp> ... [1]

Working Draft Standard ECMA-XXX

1st Edition / Draft 23 February 2012

ECMAScript Internationalization API Specification

Standard

Deleted: <sp>

Deleted:

Deleted: 2011

Deleted: <sp><sp>

Contents

Page

Deleted: 2011

Deleted: <sp><sp>

Deleted: 2011

Deleted: <sp><sp>

Deleted: 2011

Deleted: <sp><sp>

Deleted: 2011

ECMAScript Internationalization API Specification

1 Scope

This Standard defines the application programming interface for ECMAScript objects that support programs that need to adapt to the linguistic and cultural conventions used by different human languages and countries.

2 Conformance

A conforming implementation of the ECMAScript Internationalization API must conform to the ECMAScript Language Specification, 5.1 edition or successor, and must provide and support all the objects, properties, functions, and program semantics described in this specification.

A conforming implementation of the ECMAScript Internationalization API is permitted to provide additional objects, properties, and functions beyond those described in this specification. In particular, a conforming implementation of the ECMAScript Internationalization API is permitted to provide properties not described in this specification, and values for those properties, for objects that are described in this specification. A conforming implementation is not permitted to add optional arguments to the functions defined in this specification.

3 Normative References

The following referenced documents are required 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.

ECMA-262, ECMAScript Language Specification, 5.1 edition or successor

ISO/IEC 10646:2003: Information Technology – Universal Multiple-Octet Coded Character Set (UCS) plus Amendment 1:2005, plus additional amendments and corrigenda, or successor

ISO 4217:2008, Codes for the representation of currencies and funds, or successor

The Unicode Standard, Version 4.1.0, or successor

Unicode Technical Standard 35, Unicode Locale Data Markup Language, version 2.0.1 or successor

IETF BCP 47:
- RFC 5646, Tags for Identifying Languages, or successor
- RFC 4647, Matching of Language Tags, or successor

IETF RFC 6067, BCP 47 Extension U, or successor

Throughout this document, the phrase "ES5, *x*", where *x* is a sequence of numbers separated by periods, may be used as shorthand for "ECMAScript Language Specification, 5.1 edition, subclause *x*".

4 Overview

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

4.1 Internationalization, Localization, and Globalization

Internationalization of software means designing it such that it supports or can be easily adapted to support the needs of users speaking different languages and having different cultural expectations, and enables

Deleted: <sp><sp>

Deleted: Globalization

Deleted: Globalization API

Deleted: Globalization

Deleted: Globalization

Deleted: Globalization

Deleted: references

Moved (insertion) [1]

Moved (insertion) [2]

Moved up [2]: The Unicode Standard, Version 4.1.0, or successor

Deleted: <#>IANA Language Subtag Registry, file date 2011-08-25 or later

Deleted: Unicode Technical Standard 35, Unicode Locale Data Markup Language, version 2.0.1 or successor

Moved up [1]: ISO 4217:2008, Codes for the representation of currencies and funds, or successor

Deleted: API.

Deleted: <#>Globalization ... [2]

Deleted: 2011

worldwide communication between them. Localization then is the actual adaptation to a specific language and culture. Globalization of software is commonly understood to be the combination of internationalization and localization. Globalization starts at the lowest level by using a text representation that supports all languages in the world, and using standard identifiers to identify languages, countries, time zones, and other relevant parameters. It continues with using a user interface language and data presentation that the user understands, and finally often requires product-specific adaptations to the user's language, culture, and environment.

The ECMAScript Language Specification lays the foundation by using Unicode for text representation and by providing a few language-sensitive functions, but gives applications little control over the behavior of these functions. The ECMAScript Internationalization API builds on this by providing a set of customizable language-sensitive functionality. The API is useful even for applications that themselves are not internationalized, as even applications targeting only one language and one region need to properly support that one language and region. However, the API also enables applications that support multiple languages and regions, even concurrently, as may be needed in server environments.

4.2 API Overview

The ECMAScript Internationalization API is designed to complement the ECMAScript Language Specification by providing key language-sensitive functionality. The API can be added to an implementation of the ECMAScript Language Specification, 5.1 edition or successor.

The ECMAScript Internationalization API provides three key pieces of language-sensitive functionality that are required in most applications: String comparison (collation), number formatting, and date and time formatting. While the ECMAScript Language Specification provides functions for this basic functionality (String.prototype.localeCompare, Number.prototype.toLocaleString, Date.prototype.toLocaleString, Date.prototype.toLocaleDateString, and Date.prototype.toLocaleTimeString), it leaves the actual behavior of these functions largely up to implementations to define. The Internationalization API Specification provides additional functionality, control over the language and over details of the behavior to be used, and a more complete specification of required functionality.

Applications can use the API in two ways:

1. Directly, by using the constructors Collator, NumberFormat, or DateTimeFormat to construct an object, specifying a list of preferred languages and options to configure the behavior of the resulting object. The object then provides a main function (compare or format), which can be called repeatedly. It also provides a resolvedOptions function, which the application can use to find out the exact configuration of the object.

2. Indirectly, by using the functions of the ECMAScript Language Specification mentioned above, which are respecified in this specification to accept the same arguments as the Collator, NumberFormat, and DateTimeFormat constructors and produce the same results as their compare or format methods.

To support the handling of BCP 47 language tags, LocaleList objects assist with validation and canonicalization of these tags and negotiation against the available locales in an implementation.

The Intl object is used to package all functionality defined in the ECMAScript Internationalization API to avoid name collisions.

4.3 Implementation Dependencies

Due to the nature of internationalization, the API specification has to leave several details implementation dependent:

- *The set of locales that an implementation supports with adequate localizations:* Linguists estimate the number of human languages to around 6000, and the more widely spoken ones have variations based on regions or other parameters. Even large locale data collections, such as the Common Locale Data Repository, cover only a subset of this large set. Implementations targeting resource-constrained devices may have to further reduce the subset.

Deleted: <sp><sp>

Deleted: Globalization

Deleted: an initial

Deleted: Globalization

Deleted: This initial version of the

Deleted: Globalization

Deleted: Globalization

Deleted: gives applications

Deleted: Globalization

Deleted: Globalization

Deleted: globalization

Deleted: *are supported*

Deleted: 2011

- *The exact form of localizations such as format patterns:* In many cases locale-dependent conventions are not standardized, so different forms may exist side by side, or they vary over time. Different internationalization libraries may have implemented different forms, without any of them being actually wrong. In order to allow this API to be implemented on top of existing libraries, such variations have to be permitted.
- *Subsets of Unicode:* Some operations, such as collation, operate on strings that can include characters from the entire Unicode character set. However, both the Unicode standard and the ECMAScript standard allow implementations to limit their functionality to subsets of the Unicode character set. In addition, locale conventions typically don't specify the desired behavior for the entire Unicode character set, but only for those characters that are relevant for the locale. While the Unicode Collation Algorithm combines a default collation order for the entire Unicode character set with the ability to tailor for local conventions, subsets and tailorings still result in differences in behavior.

5 Notational Conventions

This standard uses a subset of the notational conventions of the ECMAScript Language Specification, 5.1 edition:

- Algorithm conventions, including the use of abstract operations, as described in ES5, 5.2.
- Internal properties, as described in ES5, 8.6.2.
- The List specification type, as described in ES5, 8.8.

NOTE As described in the ECMAScript Language Specification, algorithms are used to precisely specify the required semantics of ECMAScript constructs, but are not intended to imply the use of any specific implementation technique. Internal properties are used to define the semantics of object values, but are not part of the API. They are defined purely for expository purposes. An implementation of the API must behave as if it produced and operated upon internal properties in the manner described here.

In addition, this standard uses variable-named internal properties: The notation "[[<*name*>]]" denotes an internal property whose name is given by the variable *name*, which must have a String value.

EXAMPLE If a variable *s* has the value `"a"`, then [[<*s*>]] denotes the [[a]] internal property.

6 Identification of Locales, Time Zones, and Currencies

This clause describes the String values used in the ECMAScript Internationalization API to identify locales, currencies, and time zones.

6.1 Case Sensitivity and Case Mapping

The String values used to identify locales, currencies, and time zones are interpreted in a case-insensitive manner, treating the Unicode Basic Latin characters "A" to "Z" (U+0041 to U+005A) as equivalent to the corresponding Basic Latin characters "a" to "z" (U+0061 to U+007A). No other case folding equivalences are applied. When mapping to upper case, a mapping shall be used that maps characters in the range "a" to "z" (U+0061 to U+007A) to the corresponding characters in the range "A" to "Z" (U+0041 to U+005A) and maps no other characters to the latter range.

EXAMPLES "ß" (U+00DF) must not match or be mapped to "SS" (U+0053, U+0053). "ı" (U+0131) must not match or be mapped to "I" (U+0049).

6.2 Language Tags

The ECMAScript Internationalization API identifies locales using language tags as defined by IETF BCP 47 (RFCs 5646 and 4647 or their successors), which may include extensions such as those registered through RFC 6067. Their canonical form is specified in RFC 5646 section 4.5 or its successor.

Deleted: <sp><sp>

Deleted: globalization

Deleted: the ECMAScript Language Specification, 5.1 edition

Deleted: the ECMAScript Language Specification, 5.1 edition

Deleted: Globalization

Deleted: Implementations shall interpret the

Deleted: described

Deleted: this clause as

Deleted: Localized

Deleted: , which might introduce other

Deleted: , shall not be used.

Deleted: lower case or

Deleted: A

Deleted: Z

Deleted: 0041

Deleted: 005A

Deleted: a

Deleted: z

Deleted: 0061

Deleted: 007A), or vice versa

Deleted: Globalization

Deleted: 2011

All well-formed BCP 47 language tags, as specified in RFC 5646 section 2.2.9 or successor, are valid for use with the APIs defined by this standard. However, the set of locales and thus language tags that an implementation supports with adequate localizations is implementation dependent. The constructors Collator, NumberFormat, and DateTimeFormat map the language tags used in requests to locales supported by their respective implementations.

6.2.1 Unicode Locale Extension Sequences

This standard uses the term "Unicode locale extension sequence" for any substring of a language tag that starts with a separator "-" and the singleton "u" and includes the maximum sequence of following non-singleton subtags and their preceding "-" separators.

6.2.2 IsWellFormedLanguageTag(locale)

The IsWellFormedLanguageTag abstract operation verifies that the locale argument (which must be a String value) represents a well-formed BCP 47 language tag as specified in RFC 5646 section 2.1, or successor. It returns true if locale can be generated from the ABNF grammar in that section, starting with Language-Tag, false otherwise. Terminal value characters in the grammar are interpreted as the Unicode equivalents of the ASCII octet values given.

6.2.3 CanonicalizeLanguageTag (locale)

The CanonicalizeLanguageTag abstract operation returns the canonical and case-regularized form of the locale argument (which must be a String value that is a well-formed BCP 47 language tag as verified by the IsWellFormedLanguageTag abstract operation). It takes the steps specified in RFC 5646 section 4.5, or successor, to bring the language tag into canonical form, and to regularize the case of the subtags, but does not take the steps to bring a language tag into "extlang form" and to reorder variant subtags.

The specifications for extensions to BCP 47 language tags, such as RFC 6067, may include canonicalization rules for the extension subtag sequences they define that go beyond the canonicalization rules of RFC 5646 section 4.5. Implementations are allowed, but not required, to apply these additional rules.

6.2.4 DefaultLocale ()

The DefaultLocale abstract operation returns a String value representing the well-formed (6.2.2) and canonicalized (6.2.3) BCP 47 language tag for the host environment's current locale.

6.3 Currency Codes

The ECMAScript Internationalization API identifies currencies using 3-letter currency codes as defined by ISO 4217. Their canonical form is upper case.

All well-formed 3-letter ISO 4217 currency codes are allowed. However, the set of combinations of currency code and language tag for which localized currency symbols are available is implementation dependent. Where a localized currency symbol is not available, the ISO 4217 currency code is used for formatting.

6.3.1 IsWellFormedCurrencyCode (currency)

The IsWellFormedCurrencyCode abstract operation verifies that the currency argument (which must be a String value) represents a well-formed 3-letter ISO currency code. The following steps are taken:

1. Let *c* be ToString(*currency*).
2. Let *normalized* be the result of mapping *c* to upper case as described in 6.1.
3. If the string length of *normalized* is not 3, return false.
4. If *normalized* contains any character that is not in the range "A" to "Z" (U+0041 to U+005A), return **false**.
5. Return **true**.

Deleted: <sp><sp>

Deleted: Implementations shall accept all

Deleted: it expects to

Deleted: it expects to

Deleted: . Implementations are allowed, but not required, to also canonicalize each extension subtag sequence within the tag according to the canonicalization specified by the standard registering the extension, such as RFC 6067 section 2.1.1

Deleted: NOTE . RFC 5646 section 4.5 also provides steps to bring a language tag into "extlang form", and allows the reordering of variant subtags. CanonicalizeLanguageTag does not take these steps.

Deleted: Globalization

Deleted: Implementations shall accept all

Deleted: . The

Deleted: it expects to

Deleted: *currencyCodeRE*

Deleted: creating a new object

Deleted: if by

Deleted: expression `new RegExp()`, where `RegExp`

Deleted: the standard built-in constructor with

Deleted: name, with

Deleted: arguments `"^[`

Deleted: –

Deleted: `]{3}$"` and `"i"`.

Deleted: <#>Return the result of calling the test method of *currencyCodeRE* with the argument *currency*.

Deleted: 2011

6.4 Time Zone Names

The ECMAScript Internationalization API defines a single time zone name, "UTC", which identifies the UTC time zone.

The Intl.DateTimeFormat constructor allows this time zone name; if the time zone is not specified, the host environment's current time zone is used. Implementations shall support UTC and the host environment's current time zone (if different from UTC) in formatting.

7 Requirements for Standard Built-in ECMAScript Objects

Unless specified otherwise in this document, the objects described in this standard are subject to the generic requirements and restrictions specified for standard built-in ECMAScript objects in the ECMAScript Language Specification 5.1 edition, introduction of clause 15, or successor.

8 The Intl Object

The Intl object is a single object that has some named properties, all of which are constructors.

The value of the [[Prototype]] internal property of the Intl object is the built-in Object prototype object specified by the ECMAScript Language Specification.

The Intl object does not have a [[Construct]] internal property; it is not possible to use the Intl object as a constructor with the new operator.

The Intl object does not have a [[Call]] internal property; it is not possible to invoke the Intl object as a function.

8.1 Constructor Properties of the Intl Object

Each of the properties of the Intl object is a constructor. The behavior of these constructors is specified in the following clauses: LocaleList (9), Collator (11), NumberFormat (12), and DateTimeFormat (13).

9 LocaleList Objects

LocaleList objects represent lists of language tags identifying locales. They can be used in two ways:

- To represent a language priority list, as described in RFC 4647, section 2.3, or successor. Algorithms interpreting a LocaleList object in this sense treat the list as ordered in descending order of priority.
- To represent a set of locales, such as those supported by an application or by the implementation of an object described in this specification. Algorithms interpreting a LocaleList object in this sense treat the list as unordered.

LocaleList objects have the properties of a generic array-like object: A length property and other properties whose names are array indices, as defined in ES5, 15.4. The value of the length property is numerically greater than the name of every property inserted during construction whose name is an array index. As LocaleList objects are extensible, this invariant is not guaranteed to be maintained after construction.

The LocaleList constructor is a property of the Intl object.

9.1 The Intl.LocaleList Constructor

9.1.1 Initializing an Object as a LocaleList

The abstract operation InitializeLocaleList accepts the argument *localeList*, which must be an object, and the optional argument *locales*. It initializes *localeList* as a LocaleList object by taking the following steps:

```
1. If locales is not provided or is undefined, then
    a. Let seen be a new List containing the String returned by the DefaultLocale abstract operation.
```

Deleted: <sp><sp>

Deleted: This version of the

Deleted: Globalization

Deleted:).

Deleted: <#>The Globalization Object ... [3]

Deleted: constructor properties of the Globalization object and their prototypes shall have the properties and behavior

Deleted: Internal Properties of the Globalization

Deleted: Globalization

Deleted: a [[currentHostLocale

Deleted: , whose value is a String value representing the well-formed

Deleted: 6.2.1)

Deleted: canonicalized (6.2.2) BCP 47 language tag for the host environment's current locale.

Deleted: the ECMAScript Language Specification, 5.1 edition, 15.4.

Deleted: Globalization object. Behavior common to all constructor properties of the Globalization object is specified in 7.1.

Deleted: Called

Deleted: Function

Deleted: When Globalization.LocaleList is called with a this value that is not an object whose constructor property is Globalization.LocaleList itself, it creates and initializes a new LocaleList object. Thus the function call Globalization.LocaleList(...) is equivalent to the object creation expression new Globalization.LocaleList(...) with the same arguments. ... [4]

Moved down [3]: <#>Let *index* be 0.

Deleted: <#>Let *seen* be the result of creating a new object as if by the expression `new Object()` where `Object` is the standard built-in constructor with that name. ... [5]

Deleted: *value*

Deleted: performs

Deleted: 2011

```
2.  Else
    a.  Let seen be a new empty List.
    b.  Let O be ToObject(locales).
    c.  Let lenValue be the result of calling the [[Get]] internal method of O with the argument "length".
    d.  Let len be ToUint32(lenValue).
    e.  Let k be 0.
    f.  Repeat, while k < len
        i.  Let Pk be ToString(k).
        ii.  Let kPresent be the result of calling the [[HasProperty]] internal method of O with argument Pk.
        iii.  If kPresent is true, then
            1.  Let kValue be the result of calling the [[Get]] internal method of O with argument Pk.
            2.  If the type of kValue is not String or Object, then throw a TypeError exception.
            3.  Let tag be ToString(kValue).
            4.  If the result of calling the abstract operation IsWellFormedLanguageTag, passing tag as the
                argument, is false, then throw a RangeError exception.
            5.  Let tag be the result of calling the abstract operation CanonicalizeLanguageTag, passing
                tag as the argument.
            6.  If tag is not an element of seen, then append tag as the last element of seen.
        iv.  Increase k by 1.
3.  Let index be 0.
4.  Repeat for each element tag of seen, in list order
    a.  Call the [[DefineOwnProperty]] internal method of localeList with arguments ToString(index), Property
        Descriptor {[[Value]]: tag, [[Writable]]: false, [[Enumerable]]: true, [[Configurable]]: false}, and true.
    b.  Increase index by 1.
5.  Call the [[DefineOwnProperty]] internal method of localeList with arguments "length", Property Descriptor
    {[[Value]]: index, [[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false}, and true.
6.  Set the [[initializedLocaleList]] internal property of localeList to true.
```

NOTE Non-normative summary: The function interprets the *locales* argument as an array and copies its elements into *localeList*, validating the elements as well-formed language tags and canonicalizing them, and omitting duplicates. It tags this object as an initialized locale list.

NOTE If an object has been previously initialized using the Intl.LocaleList constructor, and the canonicalization of the previously used *locales* differs from the canonicalization of the *locales* used now, then either step 1.a or step 5 will fail.

9.1.2 The Intl.LocaleList Constructor Called as a Function

When Intl.LocaleList is called as a function rather than as a constructor, it accepts the optional argument *locales* and takes the following steps:

```
1.  If this is the Intl object or undefined, then
    a.  Return the result of creating a new object as if by the expression new Intl.LocaleList(locales),
        where Intl.LocaleList is the standard built-in constructor defined in 9.1.3.
2.  Let obj be the result of calling ToObject passing the this value as the argument.
3.  Call the InitializeLocaleList abstract operation with arguments obj and locales.
4.  Return obj.
```

9.1.3 The Intl.LocaleList Constructor Used in a new Expression

When `Intl.LocaleList` is called as part of a **`new`** expression, it is a constructor. It accepts the optional argument *locales*, and initializes the properties of the newly constructed object by calling the InitializeLocaleList abstract operation (9.1.1), passing the newly constructed object and *locales* as arguments.

The [[Prototype]] internal property of the newly constructed object is set to the original Intl.LocaleList prototype object, the one that is the value of Intl.LocaleList.prototype (9.2.1).

The [[Extensible]] internal property of the newly constructed object is set to true.

Deleted: <sp><sp>

Deleted: *value*

Deleted: Replace

Deleted: with

Deleted: Let *duplicate* be

Deleted: result

Moved (insertion) [3]

Deleted: calling

Deleted: HasProperty

Deleted: *seen* with argument *tag*. ... [6]

Deleted: *tag*, **true**, and **true**. ... [7]

Deleted: and *desc*

Deleted: Apply Array.prototype.forEach

Deleted: with

Deleted: *cb*

Deleted: Let *desc* be

Deleted: `Object()`

Deleted: `Object`

Deleted: with that name.

Deleted: Call

Deleted: [[Put]] internal method

Deleted: *desc* with

Deleted: arguments "

Deleted: ", *index*, and **true**

Deleted: Object.defineProperty function

Deleted: **this**, `"length"`,

Deleted: *desc*

Deleted: initial

Moved down [4]: The [[Extensible]] internal property of the newly constructed object is set to true.

Deleted: <#>new Globalization.LocaleList () ... [8]

Deleted: Globalization

Deleted: 2011

9.2 Properties of the Intl.LocaleList Constructor

Besides the internal properties and the length property (whose value is 1), the Intl.LocaleList constructor has the following properties:

9.2.1 Intl.LocaleList.prototype

The value of Intl.LocaleList.prototype is the built-in Intl.LocaleList prototype object (9.3).

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

9.3 Properties of the Intl.LocaleList Prototype Object

The Intl.LocaleList prototype object is itself an Intl.LocaleList instance as specified in 1.1, whose properties are set as if it had been constructed by the expression **`new Intl.LocaleList([])`**.

9.3.1 Intl.LocaleList.prototype.constructor

The initial value of Intl.LocaleList.prototype.constructor is the built-in Intl.LocaleList constructor.

9.4 Properties of Intl.LocaleList Instances

Intl.LocaleList instances inherit properties from the Intl.LocaleList prototype object.

Intl.LocaleList instances and other objects that have been successfully initialized as a LocaleList have an [[initializedLocaleList]] internal property whose value is **true**.

Intl.LocaleList instances and other objects that have been successfully initialized as a LocaleList also have the following properties.

9.4.1 length

The length property of an Intl.LocaleList object is a data property whose value is the number of array index properties added to the object by the InitializeLocaleList abstract operation.

The length property has the attributes { [[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false }.

9.4.2 Properties With Array Index Names

A LocaleList object has properties whose names are array indices from 0 to (length - 1). The value of each of these properties is a String value representing a well-formed language tag. The values are unique within a LocaleList object.

These properties initially have the attributes { [[Writable]]: false, [[Enumerable]]: true, [[Configurable]]: false }.

10 Locale and Parameter Negotiation

The constructors for the objects providing locale sensitive services, Collator, NumberFormat, and DateTimeFormat, use a common pattern to negotiate the requests represented by the localeList and options arguments against the actual capabilities of their implementations. The common behavior is described here in terms of internal properties describing the capabilities and of abstract operations using these internal properties.

10.1 Internal Properties of Service Constructors

The constructors Intl.Collator, Intl.NumberFormat, and Intl.DateTimeFormat have the following internal properties:

Deleted: <sp><sp>
Deleted: Globalization
Deleted: initial
Deleted: Globalization
Deleted: a
Deleted: object,
Deleted: internal
Deleted: using
Deleted: In the following descriptions of functions that are properties of the LocaleList prototype object, the phrase "this LocaleList object" refers to the object that is the this value for the invocation of the function. ... [9]
Deleted:
Deleted: Globalization
Moved down [10]: <#>.
Moved down [11]: extension sequences must be ignored by the algorithm. Information about such subsequences is returned separately.
Deleted: <#>[[IndexOfMatchFor]] (loc ... [10]
Moved down [5]: It uses the fallback mechanism of RFC 4647, section 3.4. T ... [11]
Moved down [6]: Information about su ... [13]
Moved down [14]: . Two algorithms a ... [25]
Moved down [15]: . The following ste ... [26]
Deleted: The internal method returns a ... [14]
Moved down [7]: The following steps ... [15]
Moved down [16]: <#>Let *matcher* be t ... [28]
Moved down [17]: <#>Else
Deleted: <#>Let *extensionMatch* be null. ... [16]
Moved down [8]: <#>Let *availableIndex* be -1.
Deleted: <#>Repeat while *i* is less than th ... [17]
Moved down [9]: <#>Increase *i* by 1.
Moved down [12]: The following steps ... [21]
Moved down [13]: . The steps taken a ... [23]
Deleted: <#>Repeat while the value of the ... [12]
Deleted: <#>If *availableIndex* does not eq ... [18]
Deleted: <#>Call the [[Put]] internal meth ... [19]
Deleted: The internal method returns a ... [20]
Deleted: <#>If the constructor of ... [22]
Deleted: <#>[[SupportedLocalesOf]] ... [24]
Deleted: the input list
Deleted: <#>If *options* is provided and no ... [27]
Deleted: <#>Return the result of calling th ... [29]
Deleted: <#>Return the result of calling th ... [30]
Deleted:
Deleted: this
Deleted: a numeric value that is one gr ... [31]
Deleted: every property whose name is an
Deleted: initially
Deleted: 2011

- [[availableLocales]] is a LocaleList object with BCP 47 language tags identifying the locales for which the implementation provides the functionality of the constructed objects. The list must include the value returned by the DefaultLocale abstract operation (6.2.4). Language tags on the list must not have a Unicode locale extension sequence.
- [[relevantExtensionKeys]] is an array of keys of the language tag extensions defined in Unicode Technical Standard 35 that are relevant for the functionality of the constructed objects.
- [[sortLocaleData]] and [[searchLocaleData]] (for Intl.Collator) and [[localeData]] (for Intl.NumberFormat and Intl.DateTimeFormat) are objects that have properties for each locale contained in [[availableLocales]]. The value of each of these properties must be an object that has properties for each key contained in [[relevantExtensionKeys]]. The value of each of these properties must be a non-empty array of those values defined in Unicode Technical Standard 35 for the given key that are supported by the implementation for the given locale, with the first element providing the default value.

EXAMPLE An implementation of DateTimeFormat might include the language tag "th" in its [[availableLocales]] internal property, and must (according to 13.2.3) include the key "ca" in its [[relevantExtensionKeys]] internal property. For Thai, the "buddhist" calendar is usually the default, but an implementation might also support the calendars "gregory", "chinese", and "islamicc" for the locale "th". The [[localeData]] internal property would therefore at least include {"th": {ca: ["buddhist", "gregory", "chinese", "islamicc"]}}.

NOTE Implementations should include in [[availableLocales]] locales that can serve as fallbacks in the algorithm used to resolve locales (see 10.2.4). For example, implementations shouldn't just provide a "de-DE" locale; they should include a "de" locale that can serve as a fallback for requests such as "de-AT" and "de-CH". For locales that in current usage would include a script subtag (such as Chinese locales), old-style language tags without script subtags should be included such that, for example, requests for "zh-TW" and "zh-HK" lead to output in traditional Chinese rather than the default simplified Chinese.

10.2 Abstract Operations

Where the following abstract operations take an *availableLocales* argument, it must be an object that has been initialized as a LocaleList. It represents a set of locales; the ordering of the locales within *availableLocales* is irrelevant.

10.2.1 IndexOfMatch (availableLocales, locale)

The IndexOfMatch abstract operation compares the provided argument *locale*, which must be a String value with a well-formed and canonicalized BCP 47 language tag, against the locales in *availableLocales* and returns the index of the best available match. It uses the fallback mechanism of RFC 4647, section 3.4. The following steps are taken:

```
1. Let indexOf be the standard built-in function object defined in ES5, 15.4.4.14.
2. Let candidate be locale.
3. Repeat
    a. Let index be the result of calling the [[Call]] internal method of indexOf with availableLocales as the this
       value and an argument list containing the single item candidate.
    b. If index ≠ -1, then return index.
    c. Let pos be the character index of the last occurrence of "-" (U+002D) within candidate. If that character
       does not occur, return -1.
    d. If pos ≥ 2 and the character "-" occurs at index pos-2 of candidate, then decrease pos by 2.
    e. Let candidate be the substring of candidate from position 0 to position pos-1.
```

10.2.2 LookupMatch (availableLocales, requestedLocales)

The LookupMatch abstract operation compares *requestedLocales*, which must be a LocaleList object representing a BCP 47 language priority list, against the locales in *availableLocales* and determines the best available language to meet the request. The following steps are taken:

```
1. Let i be 0.
2. Let len be the result of calling the [[Get]] internal method of requestedLocales with argument "length".
3. Let availableIndex be -1.
```

Deleted: <sp><sp>

Deleted: of

Deleted: [[currentHostLocale]] internal property of the Globalization object

Deleted: 7.2).

Deleted: which

Deleted: 12.3.3

Moved (insertion) [5]

Moved (insertion) [12]

Moved (insertion) [8]

Deleted: 2011

```
4. Repeat while i < len and availableIndex = -1:
    a. Let locale be the result of calling the [[Get]] internal method of requestedLocales with argument ToString(i).
    b. Let noExtensionsLocale be the String value that is locale with all Unicode locale extension sequences removed.
    c. Let availableIndex be the result of calling the IndexOfMatch abstract operation with arguments availableLocales and noExtensionsLocale.
    d. Increase i by 1.
5. Let result be the result of creating a new object as if by the expression new Object() where Object is the standard built-in constructor with that name.
6. If availableIndex ≠ -1, then
    a. Let availableLocale be the result of calling the [[Get]] internal method of availableLocales with argument ToString(availableIndex).
    b. Set the [[locale]] internal property of result to availableLocale.
    c. If locale and noExtensionsLocale are not the same String value, then
        i. Let extension be the String value consisting of the first substring of locale that is a Unicode locale extension sequence.
        ii. Let extensionIndex be the character position of the initial "-" of the first Unicode locale extension sequence within locale.
        iii. Set the [[extension]] internal property of result to extension.
        iv. Set the [[extensionIndex]] internal property of result to extensionIndex.
7. Else
    a. Set the [[locale]] internal property of result to the value returned by the DefaultLocale abstract operation.
8. Return result.
```

NOTE The algorithm is based on the Lookup algorithm described in RFC 4647 section 3.4, but options specified through Unicode locale extension sequences are ignored in the lookup. Information about such subsequences is returned separately. The abstract operation returns an object with a locale property, whose value is the language tag of the selected locale, which must be an element of *availableLocales*. If the language tag of the request locale that led to the selected locale contained a Unicode locale extension sequence, then the returned object also contains an extension property whose value is the first Unicode locale extension sequence, and an extensionIndex property whose value is the index of the first Unicode locale extension sequence within the request locale language tag.

10.2.3 BestFitMatch (availableLocales, requestedLocales)

The BestFitMatch abstract operation compares *requestedLocales*, which must be a LocaleList object representing a BCP 47 language priority list, against the locales in *availableLocales* and determines the best available language to meet the request. The algorithm is implementation dependent, but should produce results that a typical user of the requested locales would perceive as at least as good as those produced by the LookupMatch abstract operation. Options specified through Unicode locale extension sequences must be ignored by the algorithm. Information about such subsequences is returned separately. The abstract operation returns an object with a locale property, whose value is the language tag of the selected locale, which must be an element of *availableLocales*. If the language tag of the request locale that led to the selected locale contained a Unicode locale extension sequence, then the returned object also contains an extension property whose value is the first Unicode locale extension sequence, and an extensionIndex property whose value is the index of the first Unicode locale extension sequence within the request locale language tag.

10.2.4 ResolveLocale (availableLocales, requestedLocales, options, relevantExtensionKeys, localeData)

The ResolveLocale abstract operation compares a BCP 47 language priority list *requestedLocales* against the locales in *availableLocales* and determines the best available language to meet the request. Two algorithms are available to match the locales: the Lookup algorithm described in RFC 4647 section 3.4, and an implementation dependent best-fit algorithm. Independent of the locale matching algorithm, options specified through Unicode locale extension sequences are negotiated separately, taking the caller's relevant extension keys and locale data as well as client-provided options into consideration. The abstract operation returns an object with a locale property whose value is the language tag of the selected locale, and properties for each key in *relevantExtensionKeys* providing the selected value for that key.

The following steps are taken:

Deleted: <sp><sp>

Moved (insertion) [9]

Moved (insertion) [10]

Moved (insertion) [6]

Moved (insertion) [11]

Deleted: a set of available

Deleted: expects that the availableLocales argument is a LocaleList object. It

Deleted: 2011

```
1.  If requestedLocales is undefined, then
    a.  Let requestedLocales be the result of creating a new LocaleList object as if by the expression new
        Intl.LocaleList() where Intl.LocaleList is the standard built-in constructor defined in 9.1.3.
2.  Else
    a.  Let requestedLocales be ToObject(requestedLocales).
    b.  If requestedLocales does not have an [[initializedLocaleList]] internal property with value true, then
        i.  Let requestedLocales be the result of creating a new LocaleList object as if by the expression new
            Intl.LocaleList(requestedLocales), where Intl.LocaleList is the standard built-in
            constructor defined in 9.1.3.
3.  Let matcher be the value of the [[localeMatcher]] internal property of options.
4.  If matcher equals "lookup" then
    a.  Let r be the result of calling the LookupMatch abstract operation with arguments availableLocales and
        requestedLocales.
5.  Else
    a.  Let r be the result of calling the BestFitMatch abstract operation with arguments availableLocales and
        requestedLocales.
6.  Let foundLocale be the value of the [[locale]] internal property of r.
7.  Let extension be the value of the [[extension]] internal property of r.
8.  If extension is not undefined, then
    a.  Let extensionIndex be the value of the [[extensionIndex]] internal property of r.
    b.  Let split be the standard built-in function object defined in ES5, 15.5.4.14.
    c.  Let extensionSubtags be the result of calling the [[Call]] internal method of split with extension as the this
        value and an argument list containing the single item "-".
    d.  Let extensionSubtagsLength be the result of calling the [[Get]] internal method of extensionSubtags with
        argument "length".
9.  Let result be the result of creating a new object as if by the expression new Object() where Object is the
    standard built-in constructor with that name.
10. Set the [[dataLocale]] internal property of result to foundLocale.
11. Let supportedExtension be "-u".
12. Let i be 0.
13. Let len be the result of calling the [[Get]] internal method of relevantExtensionKeys with argument "length".
14. Repeat while i < len:
    a.  Let key be the result of calling the [[Get]] internal method of relevantExtensionKeys with argument
        ToString(i).
    b.  Let foundLocaleData be the result of calling the [[Get]] internal method of localeData with the argument
        foundLocale.
    c.  Let keyLocaleData be the result of calling the [[Get]] internal method of foundLocaleData with the
        argument key.
    d.  Let value be the result of calling the [[Get]] internal method of keyLocaleData with argument "0".
    e.  Let supportedExtensionAddition be "".
    f.  Let indexOf be the standard built-in function object defined in ES5, 15.4.4.14.
    g.  If extensionSubtags is not undefined, then
        i.  Let keyPos be the result of calling the [[Call]] internal method of indexOf with extensionSubtags as
            the this value and an argument list containing the single item key.
        ii. If keyPos ≠ -1 then
            1.  If keyPos + 1 < extensionSubtagsLength and the length of the result of calling the [[Get]]
                internal method of extensionSubtags with argument ToString(keyPos +1) is greater than 2,
                then
                a.  Let requestedValue be the result of calling the [[Get]] internal method of
                    extensionSubtags with argument ToString(keyPos + 1).
                b.  Let valuePos be the result of calling the [[Call]] internal method of indexOf with
                    keyLocaleData as the this value and an argument list containing the single item
                    requestedValue.
                c.  If valuePos ≠ -1, then
                    i.  Let value be requestedValue.
                    ii. Let supportedExtensionAddition be the concatenation of "-", key, "-",
                        and value.
            2.  Else
```

Deleted: <sp><sp>

Deleted: the constructor of ...equestedLocales is not ... [32]

Deleted: with…e the result of creating a new LocaleL ... [33]

Moved (insertion) [18]

Deleted: [[…ookupMatch]] internal method of…abs ... [34]

Moved (insertion) [17]

Moved up [18]: <#>Else

Deleted: [[…estFitMatch]] internal method of …abs ... [35]

Deleted: result…alue of calling …he [[Get…ocale]] ... [36]

Deleted: result…alue of calling …he [[Get…xtensio ... [37]

Deleted: result…alue of calling …he [[Get]] internal ... [38]

Deleted: split …[Call]] internal method of *split* with ... [39]

Deleted: "-".

Deleted: Call…et the [[Put…ataLocale]] internal ... [40]

Deleted: is less then the value of the length property of *relevantExtensions*

Deleted: element at index *i*…esult of *relevantExtensi* ... [41]

Deleted: element at index 0

Deleted: indexOf …[Call]] internal method of *index* ... [42]

Deleted: does not equal… -1 and ... [43]

Deleted: less…reater than or equal to the value of the ... [44]

Deleted: element at index *keyPos* + 1 of *extensionSu* ... [45]

Deleted: method of…ith *keyLocaleData* with…s the ... [46]

Deleted: *supportedExtension*…*upportedExtensionAd* ... [47]

Deleted: If *options* has an own property with name *key*, and the

Deleted: 2011

```
            a.  Let valuePos be the result of calling the [[Call]] internal method of indexOf with
                keyLocaleData as the this value and an argument list containing the single item
                "true".
            b.  If valuePos ≠ -1, then
                   i.  Let value be "true".
      h.  Let optionsValue be the value of the [[<key>]] internal property of options.
      i.  If optionsValue is not undefined, and the result of calling the [[Call]] internal method of indexOf with
          keyLocaleData as the this value and an argument list containing the single item optionsValue is not -1, then
             i.  If optionsValue is not equal to value, then
                   1. Let value be optionsValue.
                   2. Let supportedExtensionAddition be "".
      j.  Set the [[<key>]] internal property of result to value.
      k.  Append supportedExtensionAddition to supportedExtension.
      l.  Increase i by 1.
15. If the length of supportedExtension is greater than 2, then
      a.  Let preExtension be the substring of foundLocale from position 0 to position extensionIndex -1.
      b.  Let postExtension be the substring of foundLocale from position extensionIndex to the end of the string.
      c.  Let foundLocale be the concatenation of preExtension, supportedExtension, and postExtension.
16. Set the [[locale]] internal property of result to foundLocale.
17. Return result.
```

10.2.5 LookupSupportedLocales (availableLocales, requestedLocales)

The LookupSupportedLocales abstract operation returns the subset of the provided BCP 47 language priority list *requestedLocales* for which *availableLocales* has a matching locale when using the BCP 47 Lookup algorithm. Locales appear in the same order in the returned list as in *requestedLocales*. The following steps are taken:

```
1.  Let len be the result of calling the [[Get]] internal method of requestedLocales with the argument "length".
2.  Let subset be a new empty List.
3.  Let k be 0.
4.  Repeat while k < len
      a.  Let locale be the result of calling the [[Get]] internal method of requestedLocales with argument
          ToString(k).
      b.  Let noExtensionsLocale be the String value that is locale with all Unicode locale extension sequences
          removed.
      c.  Let availableIndex be the result of calling the IndexOfMatch abstract operation with arguments
          availableLocales and noExtensionsLocale.
      d.  If availableIndex ≠ -1, then append locale to the end of subset.
      e.  Increment k by 1.
5.  Let subsetArray be an Array object whose elements are the same values in the same order as the elements of subset.
6.  Return a new LocaleList object created as if by the expression new Intl.LocaleList(subsetArray), where
    Intl.LocaleList is the standard built-in constructor defined in 9.1.3.
```

10.2.6 BestFitSupportedLocales (availableLocales, requestedLocales)

The BestFitSupportedLocales abstract operation returns the subset of the provided BCP 47 language priority list *requestedLocales* for which *availableLocales* has a matching locale when using the Best Fit Match algorithm. Locales appear in the same order in the returned list as in *requestedLocales*. The steps taken are implementation dependent.

10.2.7 SupportedLocales (availableLocales, requestedLocales, options)

The SupportedLocales abstract operation returns the subset of the provided BCP 47 language priority list *requestedLocales* for which *availableLocales* has a matching locale. Two algorithms are available to match the locales: the Lookup algorithm described in RFC 4647 section 3.4, and an implementation dependent best-fit algorithm. Locales appear in the same order in the returned list as in *requestedLocales*. The following steps are taken:

```
1.  Let requestedLocales be ToObject(requestedLocales).
```

- **Deleted:** <sp><sp>
- **Deleted:** method of
- **Deleted:** with
- **Deleted:** named *key*
- **Deleted:** -1, then let *value* be the value of the property named *key* of *options*.
- **Deleted:** Call
- **Deleted:** [[Put]]
- **Deleted:** method
- **Deleted:** with the arguments *key*,
- **Deleted:** , and **true**
- **Deleted:** the value of
- **Deleted:** property
- **Deleted:** result of calling the
- **Deleted:** method
- **Deleted:** with arguments
- **Deleted:** and
- **Deleted:** result of calling the
- **Deleted:** method
- **Deleted:** with argument
- **Deleted:** Call
- **Moved (insertion) [7]**
- **Deleted:** Put
- **Deleted:** result
- **Deleted:** "locale", *foundLocale*,
- **Moved (insertion) [13]**
- **Moved (insertion) [14]**
- **Moved (insertion) [15]**
- **Deleted:** 2011

```
2. If requestedLocales does not have an [[initializedLocaleList]] internal property with value true, then
    a. Let requestedLocales be a new LocaleList object created as if by the expression new
       Intl.LocaleList(requestedLocales), where Intl.LocaleList is the standard built-in constructor
       defined in 9.1.3.
3. If options is not undefined, then
    a. Let options be ToObject(options).
    b. Let matcher be the result of calling the [[Get]] internal method of options with argument
       "localeMatcher".
    c. If matcher is not undefined, then
        i. Let matcher be ToString(matcher).
        ii. If matcher is not equal to "lookup" or "best fit", then throw a RangeError exception.
4. If matcher is undefined or equals "best fit" then
    a. Return the result of calling the BestFitSupportedLocales abstract operation with arguments
       availableLocales and requestedLocales.
5. Else
    a. Return the result of calling the LookupSupportedLocales abstract operation with arguments
       availableLocales and requestedLocales.
```

Deleted: <sp><sp>

Deleted: .

Moved (insertion) [16]

Deleted: <#>Return *result*.

10.2.8 GetGetOption (options)

The GetGetOption abstract operation returns a function that extracts a property value from the provided options object, converts it to the required type, checks whether it is one of a list of allowed values, and fills in a fallback value if necessary.

When the GetGetOption abstract operation is called with argument *options*, the following steps are taken.

```
1. Let getOption be a function which, when called with arguments property, type, values, and fallback, takes the
   following steps:
    a. Let value be the result of calling the [[Get]] internal method of options with argument property.
    b. If value is neither undefined nor null then
        i. If type equals "boolean" then let value be ToBoolean(value).
        ii. If type equals "string" then let value be ToString(value).
        iii. If type equals "number" then let value be ToNumber(value).
        iv. If values is not undefined, then
            1. If the result of calling the indexOf method of values with argument value is -1, then throw a
               RangeError exception.
        v. Return value.
    c. Else return fallback.
2. Return getOption.
```

10.2.9 GetGetNumberOption (options)

The GetGetNumberOption abstract operation returns a function that extracts a property value from the provided options object, converts it to a Number value, checks whether it is in the allowed range, and fills in a fallback value if necessary.

When the GetGetNumberOption abstract operation is called with argument *options*, the following steps are taken.

```
1. Let getNumberOption be a function which, when called with arguments property, minimum, maximum, and fallback,
   takes the following steps:
    a. Let value be the result of calling the [[Get]] internal method of options with argument property.
    b. If value is neither undefined nor null then
        i. Let value be ToNumber(value).
        ii. If value is NaN or less than minimum or greater than maximum, throw a RangeError exception.
        iii. Return floor(value).
    c. Else return fallback.
2. Return getNumberOption.
```

Deleted: 2011

ecma INTERNATIONAL

11 Collator Objects

The Intl.Collator constructor is a property of the Intl object. Behavior common to all service constructor properties of the Intl object is specified in 10.1.

11.1 The Intl.Collator Constructor

11.1.1 Initializing an Object as a Collator

The abstract operation InitializeCollator accepts the argument *collator*, which must be an object, and the optional arguments *localeList* and *options*. It initializes *collator* as a Collator object. The computation uses the abstract operations ResolveLocale (10.2.4) and GetGetOption (10.2.8).

Several steps in the algorithm use values from the following table, which associates Unicode locale extension keys, property names, types, and allowable values:

Table 1 – Collator options settable through both extension keys and options properties

Key	Property	Type	Values
kb	backwards	"boolean"	
kc	caseLevel	"boolean"	
kn	numeric	"boolean"	
kh	hiraganaQuaternary	"boolean"	
kk	normalization	"boolean"	
kf	caseFirst	"string"	"upper", "lower", "false"

The following steps are taken:

1. If *options* is not provided or is **undefined**, then
    a. Let *options* be the result of creating a new object as if by the expression `new Object()` where `Object` is the standard built-in constructor with that name.
2. Else
    a. Let *options* be ToObject(*options*).
3. Let *getOption* be the result of calling the GetGetOption abstract operation with argument *options*.
4. Let *u* be the result of calling *getOption* with arguments `"usage"`, `"string"`, `["sort", "search"]`, and `"sort"`.
5. Set the [[usage]] internal property of *collator* to *u*.
6. If *u* is equal to `"sort"`, then let *localeData* be the value of the [[sortLocaleData]] internal property of Intl.Collator; else let *localeData* be the value of the [[searchLocaleData]] internal property of Intl.Collator.
7. Let *opt* be the result of creating a new object as if by the expression `new Object()` where `Object` is the standard built-in constructor with that name.
8. Let *matcher* be the result of calling *getOption* with the arguments `"localeMatcher"`, `"string"`, `["lookup", "best fit"]`, and `"best fit"`.
9. Set the [[localeMatcher]] internal property of *opt* to *matcher*.
10. For each row in Table 1, except the header row, do:
    a. Let *key* be the name given in the Key column of the row.
    b. Let *value* be the result of calling *getOption*, passing as arguments the name given in the Property column of the row, the string given in the Type column of the row, and, if the Values column of the row contains strings, an array containing those strings.
    c. If the string given in the Type column of the row is `"boolean"` and *value* is not **undefined**, then
        i. Let *value* be ToString(*value*).
    d. Set the [[<*key*>]] internal property of *opt* to *value*.
11. Let *relevantExtensionKeys* be the value of the [[relevantExtensionKeys]] internal property of Intl.Collator.
12. Let *r* be the result of calling the ResolveLocale abstract operation with the [[availableLocales]] internal property of Intl.Collator, the *localeList* argument, *opt*, *relevantExtensionKeys*, and *localeData*.
13. Set the [[locale]] internal property of *collator* to the value of the [[locale]] internal property of *r*.

Deleted: <sp><sp>

Deleted: Globalization

Deleted: Globalization

Deleted: 7.1 and

Deleted: Called

Deleted: Function ... [48]

Deleted: is called with a this value that is not

Deleted: whose constructor property is Globalization.Collator itself, it creates

Deleted: new

Deleted: Thus the function call Globalization.Collator(...) is equivalent to the object creation expression new Globalization.Collator(...) with the same arguments. ... [49]

Deleted: <#>If *localeList* is not provided or is **undefined**, then let *localeList* be the result of creating a new LocaleList object as if by the expression `new Globalization.LocaleList()` where `Globalization.LocaleList` is the standard built-in constructor with that name.

Deleted: let

Deleted: the newly constructed object

Deleted: Call

Deleted: Put

Deleted: method

Deleted: with arguments `"localeMatcher"`,

Deleted: , and **true**

Deleted: Call

Deleted: [[Put]] internal method of *opt*, passing as arguments the name

Deleted: Key

Deleted: ,

Deleted: , and **true**.

Deleted: *relevantExtensions*

Deleted: *relevantExtensions*

Deleted: the newly constructed object

Deleted: 2011

```
14. Let i be 0.
15. Let len be the result of calling the [[Get]] internal method of relevantExtensionKeys with argument "length".
16. Repeat while i < len:
    a. Let key be the result of calling the [[Get]] internal method of relevantExtensionKeys with argument ToString(i).
    b. If key is equal to "co", then
        i. Let property be "collation".
        ii. Let value be the value of the [[co]] internal property of r.
        iii. If value is null, then let value be "default".
    c. Else use the row of Table 1 that contains the value of key in the Key column:
        i. Let property be the name given in the Property column of the row.
        ii. Let value be the value of the [[<key>]] internal property of r.
        iii. If the name given in the Type column of the row is "boolean", then let value be the result of
             comparing value with "true".
    d. Set the [[<property>]] internal property of collator to value.
    e. Increase i by 1.
17. Let s be the result of calling getOption with arguments "sensitivity", "string", and ["base",
    "accent", "case", "variant"].
18. If s is undefined, then
    a. If u is equal to "sort", then let s be "variant".
    b. Else
        i. Let dataLocale be the value of the [[dataLocale]] internal property of r.
        ii. Let dataLocaleData be the result of calling the [[Get]] internal operation of localeData with
            argument dataLocale.
        iii. Let s be the result of calling the [[Get]] internal operation of dataLocaleData with argument
             "sensitivity".
19. Set the [[sensitivity]] internal property of collator to s.
20. Let ip be the result of calling getOption with arguments "ignorePunctuation", "boolean", undefined, and
    false.
21. Set the [[ignorePunctuation]] internal property of collator to ip.
22. Set the [[boundCompare]] internal property of collator to undefined.
23. Set the [[initializedCollator]] internal property of collator to true.
```

11.1.2 The Intl.Collator Constructor Called as a Function

When Intl.Collator is called as a function rather than as a constructor, it accepts the optional arguments *localeList* and *options* and takes the following steps:

```
1. If this is the Intl object or undefined, then
    a. Return the result of creating a new object as if by the expression new Intl.Collator(localeList,
       options), where Intl.Collator is the standard built-in constructor defined in 11.1.3.
2. Let obj be the result of calling ToObject passing the this value as the argument.
3. Call the InitializeCollator abstract operation with arguments obj, localeList, and options.
4. Return obj.
```

11.1.3 The Intl.Collator Constructor Used in a new Expression

When `Intl.Collator` is called as part of a **`new`** expression, it is a constructor. It accepts the optional arguments *localeList* and *options*, and initializes the properties of the newly constructed object by calling the InitializeCollator abstract operation (1.1.1), passing the newly constructed object, *localeList*, and *options* as arguments.

The [[Prototype]] internal property of the newly constructed object is set to the original Intl.Collator prototype object, the one that is the value of Intl.Collator.prototype (11.2.1).

The [[Extensible]] internal property of the newly constructed object is set to true.

Deleted: <sp><sp>

Deleted: *relevantExtensions*

Deleted: is less than

Deleted: element at index *i*

Deleted: *relevantExtensions*.

Deleted: co

Deleted: named *key*

Deleted: named *property*

Deleted: the newly constructed object

Deleted: value of the

Deleted: property of the value of the property named *dataLocale* of *localeData*.

Deleted: the newly constructed object

Deleted: the newly constructed object

Deleted: initial

Moved down [19]: The [[Extensible]] internal property of the newly constructed object is set to true.

Deleted: 2011

11.2 Properties of the Intl.Collator Constructor

Besides the internal properties and the length property (whose value is 2), the Intl.Collator constructor has the following properties:

11.2.1 Intl.Collator.prototype

The value of Intl.Collator.prototype is the built-in Intl.Collator prototype object (11.3).

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

11.2.2 Intl.Collator.supportedLocalesOf (requestedLocales [, options])

When the supportedLocalesOf method of Intl.Collator is called, the following steps are taken:

1. Let *availableLocales* be the value of the [[availableLocales]] internal property of Intl.Collator.
2. Return the result of calling the SupportedLocales abstract operation with arguments *availableLocales*, *requestedLocales*, and *options*.

11.2.3 Internal Properties

The value of the [[availableLocales]] internal property is implementation dependent within the constraints described in 10.1.

The value of the [[relevantExtensionKeys]] internal property is an array that must include the element "co", may include any or all of the elements "kb", "kc", "kn", "kh", "kk", "kf", and must not include any other elements.

NOTE Unicode Technical Standard 35 describes ten locale extension keys that are relevant to collation: "co" for collator usage and specializations, "ka" for alternate handling, "kb" for backward second level weight, "kc" for case level, "kn" for numeric, "kh" for hiragana quaternary, "kk" for normalization, "kf" for case first, "ks" for collation strength, and "vt" for variable top. Collator, however, requires that the usage is specified through the usage property of the options object, alternate handling through the ignorePunctuation property of the options object, and the strength through the sensitivity property of the options object. The "co" key in the language tag is supported only for collator specializations, and the key "vt" is not supported in this version of the Internationalization API. Support for the remaining keys is implementation dependent.

The values of the [[sortLocaleData]] and [[searchLocaleData]] internal properties are implementation dependent within the constraints described in 10.1 and the following additional constraints:

- The first element of [[sortLocaleData]][locale].co and [[searchLocaleData]][locale].co must be null for all locale values.
- The values "standard" and "search" must not be used as elements in any [[sortLocaleData]][locale].co and [[searchLocaleData]][locale].co array.
- [[searchLocaleData]][locale] must have a sensitivity property with a String value equal to "base", "accent", "case", or "variant" for all locale values.

11.3 Properties of the Intl.Collator Prototype Object

The Intl.Collator prototype object is itself an Intl.Collator instance as specified in 11.4, whose internal properties are set as if it had been constructed by the expression `new Intl.Collator()`.

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

11.3.1 Intl.Collator.prototype.constructor

The initial value of Intl.Collator.prototype.constructor is the built-in Intl.Collator constructor.

Deleted: <sp><sp>

Deleted: Globalization

Deleted: initial

Deleted: Globalization

Deleted: Globalization

Deleted: [[SupportedLocalesOf]] internal method of

Deleted: with the arguments

Moved down [20]: <#>Internal Properties

Deleted: initial

Deleted: initial

Deleted: Globalization

Deleted: initial

Deleted: a

Deleted: object,

Deleted: using

Deleted: Globalization

Deleted:

Deleted: Globalization

Deleted: 2011

11.3.2 Compare (collator, x, y)

When the Compare abstract operation is called with arguments *collator* (which must be a Collator object), *x* and *y*, it returns a Number other than NaN that represents the result of a locale-sensitive String comparison of *x* (converted to a String) with *y* (converted to a String). The two Strings are *X* and *Y*. The two Strings are compared in an implementation-defined fashion. The result is intended to order String values in the sort order specified by the effective locale and collation options computed during construction of *collator*, and will be negative, zero, or positive, depending on whether *X* comes before *Y* in the sort order, the Strings are equal under the sort order, or *X* comes after *Y* in the sort order, respectively.

The sensitivity of *collator* is interpreted as follows:

- base: Only strings that differ in base letters compare as unequal. Examples: a ≠ b, a = á, a = A, ぁ = あ.
- accent: Only strings that differ in base letters or accents compare as unequal. Examples: a ≠ b, a ≠ á, a = A, ぁ = あ.
- case: Only strings that differ in base letters or case compare as unequal. Examples: a ≠ b, a = á, a ≠ A, ぁ = あ.
- variant: Strings that differ in base letters, accents, case, or width compare as unequal. Examples: a ≠ b, a ≠ á, a ≠ A, ぁ ≠ あ.

If the collator is set to ignore punctuation, then strings that differ only in punctuation compare as equal.

For the interpretation of options settable through extension keys, see Unicode Technical Standard 35.

Before performing the comparison, the following steps are performed to prepare the Strings:

1. Let *X* be ToString(*x*).
2. Let *Y* be ToString(*y*).

The Compare abstract operation with any given *collator* argument, if considered as a function of the remaining two arguments *x* and *y*, is a consistent comparison function (as defined in ES5, 15.4.4.11, or successor) on the set of all Strings.

The actual return values are implementation-defined to permit implementers to encode additional information in the value, but the method is required to define a total ordering on all Strings and to return 0 when comparing Strings that are considered canonically equivalent by the Unicode standard.

NOTE 1 It is recommended that the Compare abstract operation be implemented following Unicode Technical Standard 10, Unicode Collation Algorithm, using tailorings for the effective locale and collation options of *collator*.

NOTE 2 Applications should not assume that the behavior of the Compare abstract operation for Collator instances with the same resolved options will remain the same for different versions of the same implementation.

11.3.3 Intl.Collator.prototype.compare

This named accessor property returns a function that compares two strings according to the sort order of this Collator. The function is bound to this Collator, so that it can be passed directly to Array.prototype.sort.

The value of the [[Get]] attribute is a function that takes the following steps:

1. If this Collator does not have an [[initializedCollator]] internal property with value **true**, then throw a **TypeError** exception.
2. If the [[boundCompare]] internal property of this Collator is **undefined**, then:
   a. Let *that* be **this**.
   b. Let *bc* be a function that takes the arguments *x* and *y* and performs the following steps:
      i. Return the result of calling the Compare abstract operation with arguments *that*, *x*, and *y*.
   c. Set the [[boundCompare]] internal property of this Collator to *bc*.
3. Return the value of the [[boundCompare]] internal property of this Collator.

Deleted: <sp><sp>

Deleted: <#>Globalization.Collator.prototype.compare (x, y)

Deleted: compare method

Deleted: two

Deleted: this Collator object

Deleted: the

Deleted: compare method of

Deleted: Collator object

Deleted: the ECMAScript Language Specification, 5.1 edition

Deleted: The compare method itself is not directly suitable as an argument to Array.prototype.sort because it must be invoked as the method of a Collator obj ... [50]

Deleted: compare method

Deleted: Unicode Technical Standard #10, Unicode Collation Algorithm,

Deleted: this Collator object

Deleted: 3

Deleted: compare methods of

Deleted: Globalization

Deleted: 2011

The [[Set]] attribute is undefined.

11.3.4 Intl.Collator.prototype.resolvedOptions

This named accessor property provides access to the locale and collation options computed during initialization of the object.

The value of the [[Get]] attribute is a function that returns a new object with properties locale, usage, sensitivity, ignorePunctuation, collation, as well as those properties shown in Table 1 whose keys are included in the [[relevantExtensionKeys]] internal property of Intl.Collator. Each property has the value of the corresponding internal property of this Collator object (see 11.4). If the accessor is called on an object that does not have an [[initializedCollator]] internal property with value true, then a TypeError is thrown.

The [[Set]] attribute is undefined.

11.4 Properties of Intl.Collator Instances

Intl.Collator instances inherit properties from the Intl.Collator prototype object.

Intl.Collator instances and other objects that have been successfully initialized as a Collator have an [[initializedCollator]] internal property whose value is **true**.

Objects that have been successfully initialized as a Collator also have several internal properties that are computed by the constructor:

- [[locale]] is a String value with the language tag of the locale whose localization is used for collation.
- [[usage]] is one of the String values "sort" or "search", identifying the collator usage.
- [[sensitivity]] is one of the String values "base", "accent", "case", or "variant", identifying the collator's sensitivity.
- [[ignorePunctuation]] is a Boolean value, specifying whether punctuation should be ignored in comparisons.
- [[collation]] is a String value with the "type" given in Unicode Technical Standard 35 for the collation, except that the values "standard" and "search" are not allowed, while the value "default" is allowed.

Objects that have been successfully initialized as a Collator also have the following internal properties if the key corresponding to the name of the internal property in Table 1 is included in the [[relevantExtensionKeys]] internal property of Intl.Collator:

- [[backwards]] is a Boolean value, specifying whether backward second level weight is used.
- [[caseLevel]] is a Boolean value, specifying whether case level adjustment is used.
- [[numeric]] is a Boolean value, specifying whether numeric sorting is used.
- [[hiraganaQuaternary]] is a Boolean value, specifying whether hiragana characters receive special treatment at quaternary level.
- [[normalization]] is a Boolean value, specifying whether strings should be normalized before comparison.
- [[caseFirst]] is a String value; allowed values are specified in Table 1.

Finally, objects that have been successfully initialized as a Collator have a [[boundCompare]] internal property that caches the function returned by the compare method (11.3.3).

12 NumberFormat Objects

The NumberFormat constructor is a property of the Intl object. Behavior common to all service constructor properties of the Intl object is specified in 10.1.

Deleted: <sp><sp>

Deleted:

Deleted:). The [[Set]] attribute is undefined

Deleted:

Deleted: Collator instances

Deleted: Globalization

Deleted: Globalization

Deleted: 7.1 and

Deleted: 2011

12.1 The Intl.NumberFormat Constructor

12.1.1 Initializing an Object as a NumberFormat

The abstract operation InitializeNumberFormat accepts the argument *numberFormat*, which must be an object, and the optional arguments *localeList* and *options*. It initializes *numberFormat* as a NumberFormat object. The computation uses the abstract operations ResolveLocale (10.2.4), GetGetOption (10.2.8), GetGetNumberOption (10.2.9), and CurrencyDigits (defined below).

The following steps are taken:

1. If *options* is not provided or is **undefined**, then
   a. Let *options* be the result of creating a new object as if by the expression `new Object()` where `Object` is the standard built-in constructor with that name.
2. Else
   a. Let *options* be ToObject(*options*).
3. Let *getOption* be the result of calling the GetGetOption abstract operation with argument *options*.
4. Let *getNumberOption* be the result of calling the GetGetNumberOption abstract operation with argument *options*.
5. Let *opt* be the result of creating a new object as if by the expression `new Object()` where `Object` is the standard built-in constructor with that name.
6. Let *matcher* be the result of calling *getOption* with the arguments `"localeMatcher"`, `"string"`, `["lookup", "best fit"]`, and `"best fit"`.
7. Set the [[localeMatcher]] internal property of *opt* to *matcher*.
8. Let *localeData* be the value of the [[localeData]] internal property of Intl.NumberFormat.
9. Let *r* be the result of calling the ResolveLocale abstract operation with the [[availableLocales]] internal property of Intl.NumberFormat, the *localeList* argument, *opt*, the [[relevantExtensionKeys]] internal property of Intl.NumberFormat, and *localeData*.
10. Set the [[locale]] internal property of *numberFormat* to the value of the [[locale]] internal property of *r*.
11. Set the [[numberingSystem]] internal property of *numberFormat* to the value of the [[nu]] internal property of *r*.
12. Let *dataLocale* be the value of the [[dataLocale]] internal property of *r*.
13. Let *s* be the result of calling *getOption* with the arguments `"style"`, `"string"`, `["decimal", "percent", "currency"]`, and `"decimal"`.
14. Set the [[style]] internal property of *numberFormat* to *s*.
15. Let *c* be the result of calling *getOption* with the arguments `"currency"` and `"string"`.
16. If *c* is not undefined and the result of calling the IsWellFormedCurrencyCode abstract operation with argument *c* is false, then throw a **RangeError** exception.
17. If *s* is equal to `"currency"` and *c* is **undefined**, throw a **TypeError** exception.
18. If *s* is equal to `"currency"` then
    a. Let *c* be the result of converting *c* to upper case as specified in 6.1.
    b. Set the [[currency]] internal property of *numberFormat* to *c*.
19. Let *cd* be the result of calling *getOption* with the arguments `"currencyDisplay"`, `"string"`, `["code", "symbol", "name"]`, and `"symbol"`.
20. If *s* is equal to `"currency"` then set the [[currencyDisplay]] internal property of *numberFormat* to *cd*.
21. Let *mnid* be the result of calling *getNumberOption* with the arguments `"minimumIntegerDigits"`, 1, 21, and 1.
22. Set the [[minimumIntegerDigits]] internal property of *numberFormat* to *mnid*.
23. If *s* is equal to `"currency"` then let *mnfdDefault* be CurrencyDigits(*c*); else let *mnfdDefault* be 0.
24. Let *mnfd* be the result of calling *getNumberOption* with the arguments `"minimumFractionDigits"`, 0, 20, and *mnfdDefault*.
25. Set the [[minimumFractionDigits]] internal property of *numberFormat* to *mnfd*.
26. If *s* is equal to `"currency"` then let *mxfdDefault* be max(*mnfd*, CurrencyDigits(*c*)); else if *s* is equal to `"percent"` then let *mxfdDefault* be max(*mnfd*, 0); else let *mxfdDefault* be max(*mnfd*, 3).
27. Let *mxfd* be the result of calling *getNumberOption* with the arguments `"maximumFractionDigits"`, *mnfd*, 20, and *mxfdDefault*.
28. Set the [[maximumFractionDigits]] internal property of *numberFormat* to *mxfd*.
29. Delete the [[minimumSignificantDigits]] and [[maximumSignificantDigits]] internal properties of *numberFormat*.
30. If *options* has at least one of the properties minimumSignificantDigits and maximumSignificantDigits, then:
    a. Let *mnsd* be the result of calling *getNumberOption* with the arguments `"minimumSignificantDigits"`, 1, 21, and 1.

Deleted: <sp><sp>

Deleted: Called as a Function

Deleted: When Globalization.

Deleted: is called with a this value that is not

Deleted: whose constructor property is Globalization.NumberFormat itself, it creates

Deleted: new

Deleted: Thus the function call Globalization.NumberFormat(...) is equivalent to the object creation expression new Globalization.NumberFormat(...) with the same arguments. ... [51]

Deleted: <#>If *localeList* is not provided or is **undefined**, then let *localeList* be the result of creating a new LocaleList object as if by the expression `new Globalization.LocaleList()` where `Globalization.LocaleList` is the standard built-in constructor with that name.

Deleted: let

Deleted: Call the [[Put]] internal method of *opt* with arguments "

Deleted: ",

Deleted: , and **true**

Deleted: DateTimeFormat

Deleted: the newly constructed object

Deleted: the newly constructed object

Deleted: the newly constructed object

Deleted: set the [[currency]] internal property of the newly constructed object to

Deleted: , converted

Deleted: the newly constructed object

Deleted: the newly constructed object

Deleted: the newly constructed object

Deleted: the newly constructed object

Deleted: the newly constructed object

Deleted: 2011

b. Let *mxsd* be the result of calling *getNumberOption* with the arguments `"maximumSignificantDigits"`, *mnsd*, 21, and 21.
c. Set the [[minimumSignificantDigits]] internal property of *numberFormat* to *mnsd*, and the [[maximumSignificantDigits]] internal property of *numberFormat* to *mxsd*.
31. Let *g* be the result of calling *getOption* with the arguments `"useGrouping"`, `"boolean"`, **undefined**, and **true**.
32. Set the [[useGrouping]] internal property of *numberFormat* to *g*.
33. Let *dataLocaleData* be the result of calling the [[Get]] internal method of *localeData* with argument *dataLocale*.
34. Let *patterns* be the result of calling the [[Get]] internal method of *dataLocaleData* with argument `"patterns"`.
35. Let *patterns* be the result of calling the [[Get]] internal method of *patterns* with argument *s*.
36. Set the [[positivePattern]] internal property of *numberFormat* to the result of calling the [[Get]] internal method of *patterns* with the argument `"positivePattern"`.
37. Set the [[negativePattern]] internal property of *numberFormat* to the result of calling the [[Get]] internal method of *patterns* with the argument `"negativePattern"`.
38. Set the [[initializedNumberFormat]] internal property of *numberFormat* to **true**.

When the CurrencyDigits abstract operation is called with an argument *currency* (which must be an upper case String value), the following steps are taken:

1. If the ISO 4217 currency and funds code list contains *currency* as an alphabetic code, then return the minor unit value corresponding to the *currency* from the list; else return 2.

12.1.2 The Intl.NumberFormat Constructor Called as a Function

When Intl.NumberFormat is called as a function rather than as a constructor, it accepts the optional arguments *localeList* and *options* and takes the following steps:

1. If **this** is the Intl object or **undefined**, then
   a. Return the result of creating a new object as if by the expression new `Intl.NumberFormat(localeList, options)`, where `Intl.NumberFormat` is the standard built-in constructor defined in 12.1.3.
2. Let *obj* be the result of calling ToObject passing the **this** value as the argument.
3. Call the InitializeNumberFormat abstract operation with arguments *obj*, *localeList*, and *options*.
4. Return *obj*.

12.1.3 The Intl.NumberFormat Constructor Used in a new Expression

When Intl.NumberFormat is called as part of a `new` expression, it is a constructor. It accepts the optional arguments *localeList* and *options*, and initializes the properties of the newly constructed object by calling the InitializeNumberFormat abstract operation (12.1.1), passing the newly constructed object, *localeList*, and *options* as arguments.

The [[Prototype]] internal property of the newly constructed object is set to the original Intl.NumberFormat prototype object, the one that is the value of Intl.NumberFormat.prototype (12.2.1).

The [[Extensible]] internal property of the newly constructed object is set to true.

12.2 Properties of the Intl.NumberFormat Constructor

Besides the internal properties and the length property (whose value is 2), the Intl.NumberFormat constructor has the following properties:

12.2.1 Intl.NumberFormat.prototype

The value of Intl.NumberFormat.prototype is the built-in Intl.NumberFormat prototype object (12.3).

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

Deleted: <sp><sp>
Deleted: the newly constructed object
Deleted: the newly constructed object
Deleted: the newly constructed object
Deleted: value of the
Deleted: property of the property named *dataLocale* of *localeData*.
Deleted: value
Deleted: property named *s*
Deleted: the newly constructed object
Deleted: value
Deleted: property of *patterns*.
Deleted: the newly constructed object
Deleted: value
Deleted: property of *patterns*.
Deleted: is expected to
Deleted: a
Deleted: initial
Moved (insertion) [4]
Deleted: The [[Extensible]] internal property of the newly constructed object is set to true.
Deleted: Globalization
Deleted: initial
Deleted: Globalization
Deleted: 2011

12.2.2 Intl.NumberFormat.supportedLocalesOf (requestedLocales [, options])

When the supportedLocalesOf method of Intl.NumberFormat is called, the following steps are taken:

1. Let *availableLocales* be the value of the [[availableLocales]] internal property of Intl.NumberFormat.
2. Return the result of calling the SupportedLocales abstract operation with arguments *availableLocales*, *requestedLocales*, and *options*.

12.2.3 Internal Properties

The value of the [[availableLocales]] internal property is implementation dependent within the constraints described in 10.1.

The value of the [[relevantExtensionKeys]] internal property is ["nu"].

NOTE Unicode Technical Standard 35 describes two locale extension keys that are relevant to number formatting, "nu" for numbering system and "cu" for currency. Intl.NumberFormat, however, requires that the currency of a currency format is specified through the currency property in the options objects.

The value of the [[localeData]] internal property is implementation dependent within the constraints described in 10.1 and the following additional constraints:

- [[localeData]][locale] must have a patterns property for all locale values. The value of this property must be an object, which must have properties with the names of the three number format styles: `"decimal"`, `"percent"`, and `"currency"`. Each of these properties in turn must be an object with the properties positivePattern and negativePattern. The value of these properties must be string values that contain a substring `"{number}"`; the values within the currency property must also contain a substring `"{currency}"`. The pattern strings must not contain digits as specified by the Unicode Standard.

12.3 Properties of the Intl.NumberFormat Prototype Object

The Intl.NumberFormat prototype object is itself an Intl.NumberFormat instance as specified in 12.4, whose internal properties are set as if it had been constructed by the expression `new Intl.NumberFormat()`.

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

12.3.1 Intl.NumberFormat.prototype.constructor

The initial value of Intl.NumberFormat.prototype.constructor is the built-in Intl.NumberFormat constructor.

12.3.2 Intl.NumberFormat.prototype.format (value)

Returns a String value representing the result of calling ToNumber(value) according to the effective locale and the formatting options of this NumberFormat object. The computation uses the abstract operations ToRawPrecision and ToRawFixed (defined below).

The computations rely on String values and locations within numeric strings that are implementation and locale dependent (“ILD”) or implementation, locale, and numbering system dependent (“ILND”). The ILD and ILND Strings mentioned must not contain digits as specified by the Unicode Standard.

The following steps are taken:

1. If this NumberFormat object does not have an [[initializedNumberFormat]] internal property with value **true**, then throw a **TypeError** exception.
2. Let *x* be ToNumber(*value*).
3. Let *negative* be **false**.
4. If the result of isFinite(*x*) is **false**, then

Deleted: <sp><sp>

Deleted: Globalization

Deleted: [[SupportedLocalesOf]] internal method of

Deleted: with the arguments

Moved (insertion) [20]

Moved down [21]: <#>Internal Properties

Deleted: initial

Deleted: initial

Deleted: initial

Deleted: ocaleData

Deleted: objects

Deleted: which

Deleted: a

Deleted: object,

Deleted: using

Deleted: Globalization

Deleted: Globalization

Deleted: Globalization

Deleted: 2011

Deleted: <sp><sp>

```
        a.  If x is NaN, then let n be an ILD String value indicating the NaN value.
        b.  Else
                i.  Let n be an ILD String value indicating infinity.
                ii. If x < 0, then let negative be true.
5.  Else
        a.  If x < 0, then
                i.  Let negative be true.
                ii. Let x be -x.
        b.  If the value of the [[style]] internal property of this NumberFormat object is "percent", let x be 100 × x.
        c.  If the [[minimumSignificantDigits]] and [[maximumSignificantDigits]] internal properties of this
            NumberFormat object are present, then
                i.  Let n be the result of calling ToRawPrecision, passing as arguments x and the values of the
                    [[minimumSignificantDigits]] and [[maximumSignificantDigits]] internal properties of this
                    NumberFormat object.
        d.  Else
                i.  Let n be the result of calling ToRawFixed, passing as arguments x and the values of the
                    [[minimumIntegerDigits]], [[minimumFractionDigits]], and [[maximumFractionDigits]] internal
                    properties of this NumberFormat object.
        e.  If the value of the [[numberingSystem]] internal property of this NumberFormat object matches one of the
            values in the “Numbering System” column of Table 2 below, then
                i.  Let digits be a String value containing exactly the 10 digits specified in the “Digits” column of
                    Table 2 in the row containing the value of the [[numberingSystem]] internal property.
                ii. Replace each digit in n with the value of digits[digit].
        f.  Else use an implementation dependent algorithm to map n to the appropriate representation of n in the given
            numbering system.
        g.  If n contains the character ".", then replace it with an ILND String representing the decimal separator.
        h.  If the value of the [[useGrouping]] internal property of this NumberFormat object is true, then insert an
            ILND String representing a grouping separator into an ILND set of locations within the integer part of n.
6.  If negative is true, then let result be the value of the [[negativePattern]] internal property of this NumberFormat
    object; else let result be the value of the [[positivePattern]] internal property of this NumberFormat object.
7.  Replace the substring "{number}" within result with n.
8.  If the value of the [[style]] internal property of this NumberFormat object is "currency", then:
        a.  Let currency be the value of the [[currency]] internal property of this NumberFormat object.
        b.  If the value of the [[currencyDisplay]] internal property of this NumberFormat object is "code", then let cd
            be currency.
        c.  Else if the value of the [[currencyDisplay]] internal property of this NumberFormat object is "symbol",
            then let cd be an ILD string representing currency in short form. If the implementation does not have such a
            representation of currency, then use currency itself.
        d.  Else if the value of the [[currencyDisplay]] internal property of this NumberFormat object is "name", then
            let cd be an ILD string representing currency in long form. If the implementation does not have such a
            representation of currency, then use currency itself.
        e.  Replace the substring "{currency}" within result with cd.
9.  Return result.
```

When the ToRawPrecision abstract operation is called with arguments *x* (which must be a finite non-negative number), *minPrecision*, and *maxPrecision* (both must be integers between 1 and 21) the following steps are taken:

Deleted: expected to

Deleted: positive

Deleted: expected to

```
1.  Let p be maxPrecision.
2.  If x = 0, then
        a.  Let m be the String consisting of p occurrences of the character "0".
        b.  Let e be 0.
3.  Else
        a.  Let e and n be integers such that $10^{p–1} \le n < 10^p$ and for which the exact mathematical value of $n \times 10^{e–p+1} –$
            $x$ is as close to zero as possible. If there are two such sets of e and n, pick the e and n for which $n \times 10^{e–p+1}$ is
            larger.
        b.  Let m be the String consisting of the digits of the decimal representation of n (in order, with no leading
            zeroes).
4.  If $e \ge p$ then
```

Deleted: 2011

```
        a. Return the concatenation of $m$ and $e$-$p$+1 occurrences of the character "0".
5. If $e = p$-1 then
        a. Return $m$.
6. If $e \geq 0$ then
        a. Let $m$ be the concatenation of the first $e$+1 characters of $m$, the character ".", and the remaining $p$–($e$+1) characters of $m$.
7. If $e < 0$ then
        a. Let $m$ be the concatenation of the String "0.", –($e$+1) occurrences of the character "0", and the string $m$.
8. Let period be the result of calling the indexOf method of $m$ with argument ".".
9. If period > 0 and maxPrecision > minPrecision then
        a. Let cut be maxPrecision – minPrecision.
        b. Repeat while cut > 0 and the last character of $m$ is "0":
                i. Remove the last character from $m$.
                ii. Decrease cut by 1.
        c. If the last character of $m$ is ".", then
                i. Remove the last character from $m$.
10. Return $m$.
```

When the ToRawFixed abstract operation is called with arguments *x* (which must be a finite non-negative number), *minInteger* (which must be an integer between 1 and 21), *minFraction*, and *maxFraction* (which must be integers between 0 and 20) the following steps are taken:

```
1. Let $f$ be maxFraction.
2. Let $n$ be an integer for which the exact mathematical value of $n \div 10^f – x$ is as close to zero as possible. If there are two such $n$, pick the larger $n$.
3. If $n$ = 0, let $m$ be the String "0". Otherwise, let $m$ be the String consisting of the digits of the decimal representation of $n$ (in order, with no leading zeroes).
4. If $f \neq 0$, then
        a. Let $k$ be the number of characters in $m$.
        b. If $k \leq f$, then
                i. Let $z$ be the String consisting of $f$+1–$k$ occurrences of the character "0".
                ii. Let $m$ be the concatenation of Strings $z$ and $m$.
                iii. Let $k$=$f$+1.
        c. Let $a$ be the first $k$–$f$ characters of m, and let $b$ be the remaining $f$ characters of $m$.
        d. Let $m$ be the concatenation of the three Strings $a$, ".", and $b$.
        e. Let int be the number of characters in $a$.
5. Else let int be the number of characters in $m$.
6. Let cut be maxFraction – minFraction.
7. Repeat while cut > 0 and the last character of $m$ is "0":
        a. Remove the last character from $m$.
        b. Decrease cut by 1.
8. If the last character of $m$ is ".", then
        a. Remove the last character from $m$.
9. If int < minInteger, then
        a. Let $z$ be the String consisting of minInteger–int occurrences of the character "0".
        b. Let $m$ be the concatenation of Strings $z$ and $m$.
10. Return $m$.
```

Table 2 – Numbering systems with simple digit mappings

Numbering System	Digits
arab	U+0660 to U+0669
arabext	U+06F0 to U+06F9
beng	U+09E6 to U+09EF
deva	U+0966 to U+096F

Deleted: <sp><sp>

Deleted: expected to

Deleted: positive

Deleted: expected to

Deleted: both expected to

Deleted: 2011

fullwide	U+FF10 to U+FF19
gujr	U+0AE6 to U+0AEF
guru	U+0A66 to U+0A6F
hanidec	U+3007, U+4E00, U+4E8C, U+4E09, U+56DB, U+4E94, U+516D, U+4E03, U+516B, U+4E5D
khmr	U+17E0 to U+17E9
knda	U+0CE6 to U+0CEF
laoo	U+0ED0 to U+0ED9
latn	U+0030 to U+0039
mlym	U+0D66 to U+0D6F
mong	U+1810 to U+1819
mymr	U+1040 to U+1049
orya	U+0B66 to U+0B6F
tamldec	U+0BE6 to U+0BEF
telu	U+0C66 to U+0C6F
thai	U+0E50 to U+0E59
tibt	U+0F20 to U+0F29

12.3.3 Intl.NumberFormat.prototype.resolvedOptions

This named accessor property provides access to the locale and formatting options computed during initialization of the object.

The value of the [[Get]] attribute is a function that returns a new object with properties locale, numberingSystem, style, currency, currencyDisplay, minimumIntegerDigits, minimumFractionDigits, maximumFractionDigits, minimumSignificantDigits, maximumSignificantDigits, and useGrouping, each with the value of the corresponding internal property of this NumberFormat object (see 12.4). If the accessor is called on an object that does not have an [[initializedNumberFormat]] internal property with value true, then a TypeError is thrown.

The [[Set]] attribute is undefined.

12.4 Properties of Intl.NumberFormat Instances

Intl.NumberFormat instances inherit properties from the Intl.NumberFormat prototype object.

Intl.NumberFormat instances and other objects that have been successfully initialized as a NumberFormat have an [[initializedNumberFormat]] internal property whose value is **true**.

Objects that have been successfully initialized as a NumberFormat also have several internal properties that are computed by the constructor:

- [[locale]] is a String value with the language tag of the locale whose localization is used for formatting.
- [[numberingSystem]] is a String value with the "type" given in Unicode Technical Standard 35 for the numbering system used for formatting.
- [[style]] is one of the String values "decimal", "currency", or "percent", identifying the number format style used.
- [[currency]] is a String value with the currency code identifying the currency to be used if formatting with the "currency" style.
- [[currencyDisplay]] is one of the String values "code", "symbol", or "name", specifying whether to display the currency as an ISO 4217 alphabetic currency code, a localized currency symbol, or a localized currency name if formatting with the "currency" style.

Deleted: <sp><sp>

Deleted: Globalization

Deleted:

Deleted:). The [[Set]] attribute is undefined

Deleted:

Deleted: 2011

- [[minimumIntegerDigits]] is a non-negative integer Number value indicating the minimum integer digits to be used. Numbers will be padded with leading zeroes if necessary.
- [[minimumFractionDigits]] and [[maximumFractionDigits]] are non-negative integer Number values indicating the minimum and maximum fraction digits to be used. Numbers will be rounded or padded with trailing zeroes if necessary.
- [[minimumSignificantDigits]] and [[maximumSignificantDigits]] are positive integer Number values indicating the minimum and maximum fraction digits to be shown. Either none or both of these properties are defined; if they are, they override minimum and maximum integer and fraction digits – the formatter uses however many integer and fraction digits are required to display the specified number of significant digits.
- [[useGrouping]] is a Boolean value indicating whether a grouping separator should be used.
- [[positivePattern]] and [[negativePattern]] are String values as described in 12.2.3.

13 DateTimeFormat Objects

The Intl.DateTimeFormat constructor is a property of the Intl object. Behavior common to all service constructor properties of the Intl object is specified in 10.1.

13.1 The Intl.DateTimeFormat Constructor

13.1.1 Initializing an Object as a DateTimeFormat

The abstract operation InitializeDateTimeFormat accepts the argument *dateTimeFormat*, which must be an object, and the optional arguments *localeList* and *options*. It initializes *dateTimeFormat* as a DateTimeFormat object. The computation uses the abstract operations ResolveLocale (10.2.4), GetGetOption (10.2.8), ToDateTimeOptions, BasicFormatMatch, and BestFitFormatMatch (defined below). Several algorithms use values from the following table, which provides property names and allowable values for the components of date and time formats:

Table 3 – Components of date and time formats

Property	Values
weekday	"narrow", "short", "long"
era	"narrow", "short", "long"
year	"2-digit", "numeric"
month	"2-digit", "numeric", "narrow", "short", "long"
day	"2-digit", "numeric"
hour	"2-digit", "numeric"
minute	"2-digit", "numeric"
second	"2-digit", "numeric"
timeZoneName	"short", "long"

The following steps are taken:

1. Let *options* be the result of calling the ToDateTimeOptions abstract operation with arguments *options*, **true**, and **false**.
2. Let *getOption* be the result of calling the GetGetOption abstract operation with argument *options*.
3. Let *opt* be the result of creating a new object as if by the expression `new Object()` where `Object` is the standard built-in constructor with that name.
4. Let *matcher* be the result of calling *getOption* with the arguments `"localeMatcher"`, `"string"`, `["lookup", "best fit"]`, and `"best fit"`.
5. Set the [[localeMatcher]] internal property of *opt* to *matcher*.
6. Let *localeData* be the value of the [[localeData]] internal property of Intl.DateTimeFormat.

Deleted: <sp><sp>

Deleted: Globalization

Deleted: Globalization

Deleted: 7.1 and

Deleted: Called

Deleted: Function ... [52]

Deleted: is called with a this value that is not

Deleted: whose constructor property is Globalization.DateTimeFormat itself, it creates

Deleted: a new

Deleted: Thus the function call Globalization.DateTimeFormat(...) is equivalent to the object creation expression new Globalization.DateTimeFormat(...) with the same arguments. ... [53]

Deleted: <#>If *localeList* is not provided or is **undefined**, then let *localeList* be the result of creating a new LocaleList object as if by the expression `new Globalization.LocaleList()` where `Globalization.LocaleList` is the standard built-in constructor with that name.

Deleted: Call

Deleted: Put]] internal method of *opt* with arguments "

Deleted: ",

Deleted: , and **true**

Deleted: 2011

```
7.  Let r be the result of calling the ResolveLocale abstract operation with the [[availableLocales]] internal property of
    Intl.DateTimeFormat, the localeList argument, opt, the [[relevantExtensionKeys]] internal property of
    Intl.DateTimeFormat, and localeData.
8.  Set the [[locale]] internal property of dateTimeFormat to the value of the [[locale]] internal property of r.
9.  Set the [[calendar]] internal property of dateTimeFormat to the value of the [[ca]] internal property of r.
10. Set the [[numberingSystem]] internal property of dateTimeFormat to the value of the [[nu]] internal property of r.
11. Let dataLocale be the value of the [[dataLocale]] internal property of r.
12. Let tz be the result of calling the [[Get]] internal method of options with argument "timeZone".
13. If tz is not undefined, then
    a.  Let tz be ToString(tz).
    b.  Convert tz to upper case as described in 6.1.
    c.  If tz is not equal to "UTC", then throw a RangeError exception.
14. Set the [[timeZone]] internal property of dateTimeFormat to tz.
15. Let hr12 be the result of calling getOption with the arguments "hour12" and "boolean".
16. Let dataLocaleData be the result of calling the [[Get]] internal method of localeData with argument dataLocale.
17. If hr12 is undefined, then let hr12 be the result of calling the [[Get]] internal method of dataLocaleData with
    argument "hour12".
18. Set the [[hour12]] internal property of dateTimeFormat to hr12.
19. Let opt be the result of creating a new object as if by the expression new Object() where Object is the standard
    built-in constructor with that name.
20. For each row of Table 3, except the header row, do:
    a.  Let prop be the name given in the Property column of the row.
    b.  Let value be the result of calling getOption, passing as arguments the name given in the Property column of
        the row, "string", and an array with the strings given in the Values column of the row.
    c.  Set the [[<prop>]] internal property of opt to value.
21. Let formats be the result of calling the [[Get]] internal method of dataLocaleData with argument "formats".
22. Let match be the result of calling getOption with the arguments "formatMatch", "string", ["basic",
    "best fit"], and "best fit".
23. If match equals "basic" then
    a.  Let bestFormat be the result of calling BasicFormatMatch with opt and formats.
24. Else
    a.  Let bestFormat be the result of calling BestFitFormatMatch with opt and formats.
25. For each row in Table 3, except the header row, do
    a.  Let prop be the name given in the Property column of the row.
    b.  Let pDesc be the result of calling the [[GetOwnProperty]] internal method of bestFormat with argument
        prop.
    c.  If pDesc is not undefined, then
        i.  Let p be the result of calling the [[Get]] internal method of bestFormat with argument prop.
        ii. Set the [[<prop>]] internal property of dateTimeFormat to p.
26. Let pattern be the result of calling the [[Get]] internal method of bestFormat with argument "pattern".
27. Set the [[pattern]] internal property of dateTimeFormat to pattern.
28. Set the [[initializedDateTimeFormat]] internal property of dateTimeFormat to true.
```

When the ToDateTimeOptions abstract operation is called with arguments *options*, *date*, and *time*, the following steps are taken:

```
1.  If options is not supplied or is undefined, then let options be null, else let options be ToObject(options).
2.  Let create be the standard built-in function object defined in ES5, 15.2.3.5.
3.  Let options be the result of calling the [[Call]] internal method of create with undefined as the this value and an
    argument list containing the single item options.
4.  Let needDefault be true.
5.  If date equals true, then
    a.  For each of the property names "weekday", "year", "month", "day":
        i.  If the result of calling the [[Get]] internal method of options with the property name is not
            undefined, then let needDefault be false.
6.  If time equals true, then
    a.  For each of the property names "hour", "minute", "second":
        i.  If the result of calling the [[Get]] internal method of options with the property name is not
            undefined, then let needDefault be false.
```

Deleted: <sp><sp>
Deleted: the newly constructed object
Deleted: the newly constructed object
Deleted: the newly constructed object
Deleted: the newly constructed object
Deleted: value of the
Deleted: property of the property named *dataLocale* of *localeData*.
Deleted: the newly constructed object
Deleted: Call
Deleted: [[Put
Deleted: *opt*, passing as arguments the name given in the Property column of the row, *value*, and **true**.
Deleted: <#>Let *formats* be the value of the formats property of the property named *dataLocale* of *localeData*.
Deleted: *p*
Deleted: value of the property of *bestFormat* with the
Deleted: *p*
Deleted: set
Deleted: internal property
Deleted: newly constructed object
Deleted: the name given in
Deleted: Property column
Deleted: row to *p*.
Deleted: the newly constructed object
Deleted: the value of the
Deleted: *bestFormat*
Deleted: .
Deleted: built-in function Object.
Deleted: 2011

```
7.  If needDefault and date both equal true, then
    a.  For each of the property names "year", "month", "day":
        i.  Call the [[DefineOwnProperty]] internal method of options with the property name, Property
            Descriptor {[[Value]]: "numeric", [[Writable]]: true, [[Enumerable]]: true, [[Configurable]]:
            true}, and true.
8.  If needDefault and time both equal true, then
    a.  For each of the property names "hour", "minute", "second":
        i.  Call the [[DefineOwnProperty]] internal method of options with the property name, Property
            Descriptor {[[Value]]: "numeric", [[Writable]]: true, [[Enumerable]]: true, [[Configurable]]:
            true}, and true.
9.  Return options.
```

When the BasicFormatMatch abstract operation is called with two arguments options and formats, the following steps are taken:

```
1.  Let removalPenalty be 120.
2.  Let additionPenalty be 20.
3.  Let longLessPenalty be 8.
4.  Let longMorePenalty be 6.
5.  Let shortLessPenalty be 6.
6.  Let shortMorePenalty be 3.
7.  Let bestScore be -Infinity.
8.  Let bestFormat be undefined.
9.  Let i be 0.
10. Let len be the result of calling the [[Get]] internal method of formats with argument "length".
11. Repeat while i < len:
    a.  Let format be the result of calling the [[Get]] internal method of formats with argument ToString(i).
    b.  Let score be 0.
    c.  For each property shown in Table 3:
        i.   Let optionsProp be the result of calling the [[Get]] internal method of options with argument
             property.
        ii.  Let formatPropDesc be the result of calling the [[GetOwnProperty]] internal method of format
             with argument property.
        iii. If formatPropDesc is not undefined, then
             1. Let formatProp be the result of calling the [[Get]] internal method of format with argument
                property.
        iv.  If optionsProp is undefined and formatProp is not undefined, then decrease score by
             additionPenalty.
        v.   Else if optionsProp is not undefined and formatProp is undefined, then decrease score by
             removalPenalty.
        vi.  Else
             1. Let values be the array ["2-digit", "numeric", "narrow", "short",
                "long"].
             2. Let optionsPropIndex be the index of optionsProp within values.
             3. Let formatPropIndex be the index of formatProp within values.
             4. Let delta be max(min(formatPropIndex - optionsPropIndex, 2), -2).
             5. If delta = 2, decrease score by longMorePenalty.
             6. Else if delta = 1, decrease score by shortMorePenalty.
             7. Else if delta = -1, decrease score by shortLessPenalty.
             8. Else if delta = -2, decrease score by longLessPenalty.
    d.  If score > bestScore, then
        i.  Let bestScore be score.
        ii. Let bestFormat be format.
    e.  Increase i by 1.
12. Return bestFormat.
```

When the BestFitFormatMatch abstract operation is called with two arguments options and formats, it performs implementation dependent steps, which should return a set of component representations that a

Deleted: <sp><sp>

Deleted: Put

Deleted: Put

Moved (insertion) [22]

Moved (insertion) [23]

Moved (insertion) [24]

Moved up [24]: <#>Let *shortLessPenalty* be 6.

Moved up [23]: <#>Let *longMorePenalty* be 6.

Moved up [22]: <#>Let *longLessPenalty* be 8.

Deleted: value

Deleted: length property

Deleted: :

Deleted: is less than

Deleted: element at index *i*

Deleted: .

Deleted: value

Deleted: property named *property*

Deleted: value of the property named

Deleted: of *format*

Deleted: equals

Deleted: equals

Deleted: equals

Deleted: equals

Deleted: is greater than

Deleted: 2011

typical user of the selected locale would perceive as at least as good as the one returned by BasicFormatMatch.

13.1.2 The Intl.DateTimeFormat Constructor Called as a Function

When Intl.DateTimeFormat is called as a function rather than as a constructor, it accepts the optional arguments *localeList* and *options* and takes the following steps:

```
1. If this is the Intl object or undefined, then
     a. Return the result of creating a new object as if by the expression new
        Intl.DateTimeFormat(localeList, options), where Intl.DateTimeFormat is the standard
        built-in constructor defined in 13.1.3.
2. Let obj be the result of calling ToObject passing the this value as the argument.
3. Call the InitializeDateTimeFormat abstract operation with arguments obj, localeList, and options.
4. Return obj.
```

13.1.3 The Intl.DateTimeFormat Constructor Used in a new Expression

When Intl.DateTimeFormat is called as part of a **new** expression, it is a constructor. It accepts the optional arguments *localeList* and *options*, and initializes the properties of the newly constructed object by calling the InitializeDateTimeFormat abstract operation (1.1.1), passing the newly constructed object, *localeList*, and *options* as arguments.

The [[Prototype]] internal property of the newly constructed object is set to the original Intl.DateTimeFormat prototype object, the one that is the value of Intl.DateTimeFormat.prototype (13.2.1).

The [[Extensible]] internal property of the newly constructed object is set to true.

13.2 Properties of the Intl.DateTimeFormat Constructor

Besides the internal properties and the length property (whose value is 2), the Intl.DateTimeFormat constructor has the following properties:

13.2.1 Intl.DateTimeFormat.prototype

The value of Intl.DateTimeFormat.prototype is the built-in Intl.DateTimeFormat prototype object (13.3).

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

13.2.2 Intl.DateTimeFormat.supportedLocalesOf (requestedLocales [, options])

When the supportedLocalesOf method of Intl.DateTimeFormat is called, the following steps are taken:

```
1. Let availableLocales be the value of the [[availableLocales]] internal property of Intl.DateTimeFormat.
2. Return the result of calling the SupportedLocales abstract operation with arguments availableLocales,
   requestedLocales, and options.
```

13.2.3 Internal Properties

The value of the [[availableLocales]] internal property is implementation dependent within the constraints described in 10.1.

The value of the [[relevantExtensionKeys]] internal property is ["ca", "nu"].

NOTE Unicode Technical Standard 35 describes three locale extension keys that are relevant to date and time formatting, "ca" for calendar, "tz" for time zone, and implicitly "nu" for the numbering system of the number format used for numbers within the date format. DateTimeFormat, however, requires that the time zone is specified through the timeZone property in the options objects.

Deleted: <sp><sp>

Deleted: initial

Moved (insertion) [19]

Deleted: The [[Extensible]] internal property of the newly constructed object is set to true.

Deleted: Globalization

Deleted: initial

Deleted: Globalization

Deleted: Globalization

Deleted: [[SupportedLocalesOf]] internal method of

Deleted: with the arguments

Moved (insertion) [21]

Deleted: <#>Internal Properties

Deleted: initial

Deleted: initial

Deleted: 2011

The value of the [[localeData]] internal property is implementation dependent within the constraints described in 10.1 and the following additional constraints:

- [[localeData]][locale] must have an hour12 property with a Boolean value for all locale values.
- [[localeData]][locale] must have a formats property for all locale values. The value of this property must be an array of objects, each of which has a subset of the properties shown in Table 3, where each property must have one of the values specified for the property in Table 3. Multiple objects in an array may use the same subset of the properties as long as they have different values for the properties. The following subsets must be available for each locale:
- weekday, year, month, day, hour, minute, second
- weekday, year, month, day
- year, month, day
- year, month
- month, day
- hour, minute, second
- hour, minute

  Each of the objects must also have a pattern property, whose value is a String value that contains for each of the date and time format component properties of the object a substring starting with "{", followed by the name of the property, followed by "}".

EXAMPLE An implementation might include the following object as part of its English locale data: {hour: "numeric", minute: "2-digit", second: "2-digit", pattern: "{hour}:{minute}:{second}"}.

13.3 Properties of the Intl.DateTimeFormat Prototype Object

The Intl.DateTimeFormat prototype object is itself an Intl.DateTimeFormat instance as specified in 13.4, whose internal properties are set as if it had been constructed by the expression **`new Intl.DateTimeFormat()`**.

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

13.3.1 Intl.DateTimeFormat.prototype.constructor

The initial value of Intl.DateTimeFormat.prototype.constructor is the built-in Intl.DateTimeFormat constructor.

13.3.2 Intl.DateTimeFormat.prototype.format ([date])

Returns a String value representing the result of calling ToNumber(date) according to the effective locale and the formatting options of this DateTimeFormat object. The computation uses the abstract operation ToLocalTime (defined below). The result of calling ToNumber(date) is interpreted as a time value as specified in ES5, 15.9.1.1, or successor.

1. If this DateTimeFormat object does not have an [[initializedDateTimeFormat]] internal property with value **true**, then throw a **TypeError** exception.
2. If *date* is omitted let *y* be **`Date.now()`**; else let *y* be ToNumber(*date*).
3. If *y* is not a finite Number, then throw a **RangeError** exception.
4. Let *locale* be the value of the [[locale]] internal property of this DateTimeFormat object.
5. Let *nf* be the result of creating a new NumberFormat object as if by the expression **`new Intl.NumberFormat([locale])`** where **`Intl.NumberFormat`** is the standard built-in constructor defined in 12.1.3.
6. Let *nf2* be the result of creating a new NumberFormat object as if by the expression **`new Intl.NumberFormat([locale], {minimumIntegerDigits: 2})`** where **`Intl.NumberFormat`** is the standard built-in constructor defined in 12.1.3.
7. Let *tm* be the result of calling ToLocalTime with *y*, the value of the [[calendar]] internal property of this DateTimeFormat object, and the value of the [[timeZone]] internal property of this DateTimeFormat object.
8. Let *result* be the value of the [[pattern]] internal property of this DateTimeFormat object.
9. For each row of Table 3, except the header row, do:

Deleted: <sp><sp>
Deleted: initial
Deleted: ocaleData
Deleted: a
Deleted: object,
Deleted: using
Deleted: Globalization
Deleted: Globalization
Deleted: Globalization
Deleted: Globalization
Deleted: Globalization
Deleted: with that name.
Deleted: Globalization
Deleted: Globalization
Deleted: with that name.
Deleted: 2011

```
a. If this DateTimeFormat object has an internal property with the name given in the Property column of the row, then:
    i. Let p be the name given in the Property column of the row.
    ii. Let f be the value of the [[<p>]] internal property of this DateTimeFormat object.
    iii. Let v be the value of the [[<p>]] internal property of tm.
    iv. If p equals "month", then increase v by 1.
    v. Let format be the standard built-in function object defined in 12.3.2.
    vi. If f equals "numeric", then
        1. Let fv be the result of calling the [[Call]] internal method of format with nf as the this value and an argument list containing the single item v.
    vii. Else if f equals "2-digit", then
        1. Let fv be the result of calling the [[Call]] internal method of format with nf2 as the this value and an argument list containing the single item v.
        2. If the length of fv is greater than 2, let fv be the substring of fv containing the last two characters.
    viii. Else if f equals "narrow", "short", or "long", then let fv be an implementation, locale, and calendar dependent String value representing f in the desired form. If p equals "month", then the String value may also depend on whether this DateTimeFormat object has a day property. If p equals "timeZoneName", then the String value may also depend on the value of the isDST property of tm.
    ix. Replace the substring of result that consists of "{", p, and "}", with fv.
10. Return result.
```

When the abstract operation ToLocalTime is called with arguments date, calendar, and timeZone, the following steps are taken:

```
1. Apply calendrical calculations on date for the given calendar and time zone to produce weekday, era, year, month, day, hour, minute, second, and inDST values. The calculations should use best available information about the specified calendar and time zone. If the calendar is "gregory", then the calculations must match the algorithms specified in ES5, 15.9.1, except that calculations are not bound by the restrictions on the use of best available information on time zones for local time zone adjustment and daylight saving time adjustment imposed by ES5, 15.9.1.7 and 15.9.1.8.
2. Return an object with internal properties [[weekday]], [[era]], [[year]], [[month]], [[day]], [[hour]], [[minute]], [[second]], and [[inDST]], each with the corresponding calculated value.
```

NOTE It is recommended that implementations use the time zone information of the IANA Time Zone Database.

13.3.3 Intl.DateTimeFormat.prototype.resolvedOptions

This named accessor property provides access to the locale and formatting options computed during initialization of the object.

The value of the [[Get]] attribute is a function that returns a new object with properties locale, calendar, numberingSystem, timeZone, hour12, weekday, era, year, month, day, hour, minute, second, and timeZoneName, each with the value of the corresponding internal property of this DateTimeFormat object (see 13.4). If the accessor is called on an object that does not have an [[initializedDateTimeFormat]] internal property with value true, then a TypeError is thrown.

The [[Set]] attribute is undefined.

NOTE In this version of the ECMAScript Internationalization API, the timeZone property will remain undefined if no timeZone property was provided in the options object provided to the Intl.DateTimeFormat constructor. However, applications should not rely on this, as future versions may return a String value identifying the host environment's current time zone instead.

13.4 Properties of Intl.DateTimeFormat Instances

Intl.DateTimeFormat instances inherit properties from the Intl.DateTimeFormat prototype object.

Deleted: <sp><sp>

Deleted: named *p*

Deleted: named *p*

Deleted: let

Deleted: format

Deleted: *nf*

Deleted: format

Deleted: *nf2*

Deleted: result of calling the

Deleted: method

Deleted: with *argument* length-2

Deleted: are expected to

Deleted: the ECMAScript Language Specification, 5.1 edition

Deleted: the ECMAScript Language Specification, 5.1 edition

Deleted: ,

Deleted: ,

Deleted: ,

Deleted: ,

Deleted: ,

Deleted: ,

Deleted: ,

Deleted: ,

Deleted: ,

Deleted: IANA Time Zone Database.

Deleted: Globalization

Deleted:

Deleted:). The [[Set]] attribute is undefined

Deleted: Globalization

Deleted:

Deleted: 2011

Intl.DateTimeFormat instances and other objects that have been successfully initialized as a DateTimeFormat have an [[initializedDateTimeFormat]] internal property whose value is **true**.

Objects that have been successfully initialized as a DateTimeFormat also have several internal properties that are computed by the constructor:

- [[locale]] is a String value with the language tag of the locale whose localization is used for formatting.
- [[calendar]] is a String value with the "type" given in Unicode Technical Standard 35 for the calendar used for formatting.
- [[numberingSystem]] is a String value with the "type" given in Unicode Technical Standard 35 for the numbering system used for formatting.
- [[timeZone]] is either the String value "UTC" or undefined.
- [[hour12]] is a Boolean value indicating whether 12-hour format (true) or 24-hour format (false) should be used.
- [[weekday]], [[era]], [[year]], [[month]], [[day]], [[hour]], [[minute]], [[second]], [[timeZoneName]] are each either undefined, indicating that the component is not used for formatting, or one of the String values given in Table 3, indicating how the component should be presented in the formatted output.
- [[pattern]] is a String value as described in 13.2.3.

14 Locale Sensitive Functions of the ECMAScript Language Specification

The ECMAScript Language Specification, edition 5.1 or successor, describes several locale sensitive functions. An ECMAScript implementation that implements this Internationalization API shall implement these functions as described here.

14.1 Properties of the String Prototype Object

14.1.1 String.prototype.localeCompare (that [, localeList [, options]])

When the **localeCompare** method is called with arguments *that*, *localeList*, and *options*, the following steps are taken:

1. Let *collator* be the result of creating a new object as if by the expression **new Intl.Collator(localeList, options)** where **Intl.Collator** is the standard built-in constructor defined in 13.1.3.
2. Return the result of calling the Compare abstract operation with arguments *collator*, **this**, and *that*.

14.2 Properties of the Number Prototype Object

14.2.1 Number.prototype.toLocaleString ([localeList [, options]])

When the **toLocaleString** method is called with arguments *localeList* and *options*, the following steps are taken:

1. Let *numberFormat* be the result of creating a new object as if by the expression **new Intl.NumberFormat(localeList, options)** where **Intl.NumberFormat** is the standard built-in constructor defined in 12.1.3.
2. Let *format* be the standard built-in function object defined in 12.3.2.
3. Return the result of calling the [[Call]] internal method of *format* with *numberFormat* as the **this** value and an argument list containing the single item **this**.

14.3 Properties of the Date Prototype Object

The properties of the Date prototype object described here use the abstract operation ToDateTimeOptions, specified in 1.1.1.

Deleted: <sp><sp>

Field Code Changed

Deleted: implementations

Deleted: implement

Deleted: Globalization

Deleted: Globalization

Deleted: Globalization

Deleted: with that name.

Deleted: compare method of *collator*

Deleted: Globalization

Deleted: Globalization

Deleted: with that name.

Deleted: format

Deleted: with

Field Code Changed

Deleted: 2011

14.3.1 Date.prototype.toLocaleString ([localeList [, options]])

When the `toLocaleString` method is called with arguments *localeList* and *options*, the following steps are taken:

1. Let *options* be the result of calling the ToDateTimeOptions abstract operation with arguments *options*, **true**, and **true**.
2. Let *dateTimeFormat* be the result of creating a new object as if by the expression `new Intl.DateTimeFormat(localeList, options)` where `Intl.DateTimeFormat` is the standard built-in constructor defined in 13.1.3.
3. Let *format* be the standard built-in function object defined in 13.3.2.
4. Return the result of calling the [[Call]] internal method of *format* with *dateTimeFormat* as the **this** value and an argument list containing the single item **this**.

14.3.2 Date.prototype.toLocaleDateString ([localeList [, options]])

When the `toLocaleDateString` method is called with arguments *localeList* and *options*, the following steps are taken:

1. Let *options* be the result of calling the ToDateTimeOptions abstract operation with arguments *options*, **true**, and **false**.
2. Let *dateFormat* be the result of creating a new object as if by the expression `new Intl.DateTimeFormat(localeList, options)` where `Intl.DateTimeFormat` is the standard built-in constructor defined in 13.1.3.
3. Let *format* be the standard built-in function object defined in 13.3.2.
4. Return the result of calling the [[Call]] internal method of *format* with *dateTimeFormat* as the **this** value and an argument list containing the single item **this**.

14.3.3 Date.prototype.toLocaleTimeString ([localeList [, options]])

When the `toLocaleTimeString` method is called with arguments *localeList* and *options*, the following steps are taken:

1. Let *options* be the result of calling the ToDateTimeOptions abstract operation with arguments *options*, **false**, and **true**.
2. Let *timeFormat* be the result of creating a new object as if by the expression `new Intl.DateTimeFormat(localeList, options)` where `Intl.DateTimeFormat` is the standard built-in constructor defined in 13.1.3.
3. Let *format* be the standard built-in function object defined in 13.3.2.
4. Return the result of calling the [[Call]] internal method of *format* with *dateTimeFormat* as the **this** value and an argument list containing the single item **this**.

References

ECMA-262, ECMAScript Language Specification
http://www.ecma-international.org/publications/standards/Ecma-262.htm

ISO 4217:2008, Codes for the representation of currencies and funds
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=46121

ISO/IEC 10646:2003: Information Technology – Universal Multiple-Octet Coded Character Set (UCS) plus Amendment 1:2005
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39921
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=40755

The Unicode Standard
http://www.unicode.org/versions/latest

Deleted: <sp><sp>

Deleted: `Globalization`

Deleted: `Globalization`

Deleted: with that name.

Deleted: format

Deleted: with

Deleted: `Globalization`

Deleted: `Globalization`

Deleted: with that name.

Deleted: format

Deleted: *dateFormat*

Deleted: `Globalization`

Deleted: `Globalization`

Deleted: with that name.

Deleted: format

Deleted: *timeFormat*

Deleted: 2011

Unicode Technical Standard 10, Unicode Collation Algorithm
http://unicode.org/reports/tr10/

Unicode Technical Standard 35, Unicode Locale Data Markup Language
http://unicode.org/reports/tr35/

Unicode Common Locale Data Repository
http://cldr.unicode.org/

IETF BCP 47:

- RFC 5646, Tags for Identifying Languages
  http://tools.ietf.org/html/rfc5646
- RFC 4647, Matching of Language Tags
  http://tools.ietf.org/html/rfc4647

IETF RFC 6067, BCP 47 Extension U
http://tools.ietf.org/html/rfc6067

IANA Time Zone Database
http://www.iana.org/time-zones

Deleted: <sp><sp>

Deleted: 2011