Ecma/TC39/2011/048

Working Draft Standard ECMA-XXX

1st Edition / Draft 31 October 2011

ECMAScript Internationalization API Specification

Standard

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

Contents

Page

ECMAScript Globalization API Specification

1 Scope

This Standard defines the ECMAScript Globalization API.

2 Conformance

A conforming implementation of the ECMAScript Globalization 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 Globalization API is permitted to provide additional objects, properties, and functions beyond those described in this specification. In particular, a conforming implementation of the ECMAScript Globalization 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

The Unicode Standard, Version 4.1.0, or successor

IETF BCP 47:
- RFC 5646, Tags for Identifying Languages, or successor
- RFC 4647, Matching of Language Tags, or successor
- IANA Language Subtag Registry, file date 2011-08-25 or later

IETF RFC 6067, BCP 47 Extension U, or successor

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

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

4 Overview

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

4.1 Globalization

In software development, globalization is commonly understood to be the combination of internationalization and localization. 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 worldwide communication between them. Localization then is the actual adaptation to a specific language and culture. 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 Globalization API builds on this by providing an initial 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 Globalization 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.

This initial version of the ECMAScript Globalization 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 functionality (String.prototype.localeCompare, Number.prototype.toLocaleString, Date.prototype.toLocaleString and friends), the API described here gives applications control over the language and over details of the behavior to be used.

The standard usage pattern is that applications use 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.

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

4.3 Implementation Dependencies

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

- *The set of locales that are supported 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.
- *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 globalization 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.

4.4 Conventions

This standard makes use of internal properties, a convention introduced in the ECMAScript Language Specification. Internal properties are used by the specification to define the semantics of object values. These internal properties are not part of the ECMAScript language or the ECMAScript Globalization API. They are defined by this specification purely for expository purposes. An implementation of the ECMAScript Globalization API must behave as if it produced and operated upon internal properties in the manner described here. The names of internal properties are enclosed in double square brackets [[ ]]. When an algorithm uses an internal property of an object and the object does not implement the indicated internal property, a TypeError exception is thrown.

5 Identification of Locales, Time Zones, and Currencies

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

5.1 Case Sensitivity and Case Mapping

Implementations shall interpret the String values described in this clause as case-insensitive, 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). Localized case folding, which might introduce other equivalences, shall not be used. When mapping to lower case or upper case, a mapping shall be used that maps characters in the range "A" to "Z" (U+0041 to U+005A) to the corresponding characters in the range "a" to "z" (U+0061 to U+007A), or vice versa.

5.2 Language Tags

The ECMAScript Globalization 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.

Implementations shall accept all well-formed BCP 47 language tags, as specified in RFC 5646 section 2.2.9 or successor. However, the set of locales and thus language tags that an implementation supports with adequate localizations is implementation dependent.

5.2.1 IsWellFormedLanguageTag(locale)

The IsWellFormedLanguageTag abstract operation verifies that the locale argument (which it expects to 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.

5.2.2 CanonicalizeLanguageTag(locale)

The CanonicalizeLanguageTag abstract operation returns the canonical and case-regularized form of the locale argument (which it expects to 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. 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.

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.

5.3 Currency Codes

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

Implementations shall accept all well-formed 3-letter ISO 4217 currency codes. The set of combinations of currency code and language tag for which localized currency symbols are available is implementation dependent.

5.3.1 IsWellFormedCurrencyCode(currency)

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

1. Let *currencyCodeRE* be the result of creating a new object as if by the expression **`new RegExp()`**, where **`RegExp`** is the standard built-in constructor with that name, with the arguments **`"^[A-Z]{3}$"`** and **`"i"`**.
2. Return the result of calling the test method of *currencyCodeRE* with the argument *currency*.

5.4 Time Zone Names

This version of the ECMAScript Globalization API defines a single time zone name, "UTC", which identifies the UTC time zone.

Implementations shall support UTC and the host environment’s current time zone (if different from UTC).

6 The Globalization Object

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

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

NOTE The [[Extensible]] internal property is set to false for compatibility with the future module system in the ECMAScript Language Specification, 6 edition.

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

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

6.1 Constructor Properties of the Globalization Object

Each of the properties of the Globalization object is a constructor. The common behavior of these constructors is specified in this section; all remaining aspects are specified in the following clauses: LocaleList, Collator, NumberFormat, and DateTimeFormat.

6.1.1 Properties of the Constructors and Their Prototypes

Unless specified otherwise in this document, the constructor properties of the Globalization object and their prototypes shall have the properties and behavior specified for standard built-in ECMAScript objects in the ECMAScript Language Specification 5.1 edition, introduction of clause 15, or successor.

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

- [[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 of the [[currentHostLocale]] internal property of the Globalization object (6.2). Language tags on the list must not have a Unicode 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 Collator) and [[localeData]] (for NumberFormat and DateTimeFormat) are objects that have properties for each locale contained in [[availableLocales]]. The value of each of these properties must be an object which 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 10.3.3) 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 simple algorithm used to resolve locales (see 7.6.1). 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. Finally, implementations may want to include languages that can be adequately handled by the implementation of another language – for example, the deprecated language tag "sh" (Serbo-Croatian) might be treated as equivalent to "sr" (Serbian).

6.2 Internal Properties of the Globalization Object

The Globalization object has a [[currentHostLocale]] internal property, whose value is a String value representing the well-formed (5.2.1) and canonicalized (5.2.2) BCP 47 language tag for the host environment's current locale.

7 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. In this case, the list is ordered in descending order of priority.
- To represent an unordered set of locales, such as those supported by an application or by the implementation of an object described in this specification.

The LocaleList constructor is a property of the Globalization object. Behavior common to all constructor properties of the Globalization object is specified in 6.1.

7.1 The LocaleList Constructor Called as a Function

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.

7.2 The LocaleList Constructor

When Globalization.LocaleList is called with a this value that is an object whose constructor property is Globalization.LocaleList itself, it acts as a constructor: it initializes the object.

7.2.1 new Globalization.LocaleList(locales)

When the LocaleList constructor is called with one argument, it interprets the locales argument as an array and copies its elements into the newly constructed object, validating the elements as well-formed language tags using the abstract operation IsWellFormedLanguageTag (5.2.1), and omitting duplicates.

```
1. Let obj be the this value.
2. Let index be 0.
3. 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.
4. Let cb be a function that takes the argument value and performs the following steps:
    a. Let tag be ToString(value).
    b. If the result of calling the abstract operation IsWellFormedLanguageTag, passing tag as the
       argument, is false, then throw a ValueError exception.
    c. Replace tag with the result of calling the abstract operation CanonicalizeLanguageTag, passing tag
       as the argument.
    d. Let duplicate be the result of calling the [[HasProperty]] internal method of seen with argument tag.
    e. If duplicate is true, then return.
    f. Call the [[Put]] internal method of seen with arguments tag, true, and true.
    g. Let desc 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.
    h. Call the [[Put]] internal method of desc with the arguments "value", tag, and true.
    i. Call the [[Put]] internal method of desc with the arguments "enumerable", true, and true.
    j. Call the Object.defineProperty function with arguments obj, ToString(index), and desc.
    k. Increase index by 1.
5. Apply Array.prototype.forEach to locales with argument cb.
6. Let desc 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.
7. Call the [[Put]] internal method of desc with the arguments "value", index, and true.
8. Call the Object.defineProperty function with arguments this, "length", and desc.
```

The [[Prototype]] internal property of the newly constructed object is set to the original LocaleList prototype object, the one that is the initial value of LocaleList.prototype (7.3.1).

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

7.2.2 new Globalization.LocaleList()

When the LocaleList constructor is called with no argument, it behaves as if it had received the array [locale] as the first argument, where locale is the value of the [[currentHostLocale]] internal property of the Globalization object.

7.3 Properties of the LocaleList Constructor

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

7.3.1 Globalization.LocaleList.prototype

The initial value of Globalization.LocaleList.prototype is the built-in LocaleList prototype object (7.4).

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

7.4 Properties of the LocaleList Prototype Object

The LocaleList prototype object is itself a LocaleList object, whose internal properties are set as if it had been constructed using new LocaleList().

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.

7.4.1 Globalization.LocaleList.prototype.constructor

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

7.4.2 [[IndexOfMatchFor]] (locale)

The [[IndexOfMatchFor]] internal method compares the provided argument locale, which it expects to be a String value with a well-formed and canonicalized BCP 47 language tag, against the locales in this locale list 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. Repeat while the value of the length property of locale is greater than 0:
    a. Let index be the result of applying the Array.prototype.indexOf method to this LocaleList object with the argument list [locale].
    b. If index does not equal -1, then return index.
    c. Let pos be the result of calling the lastIndexOf method of locale with the argument "-".
    d. If pos does not equal -1, then
        i. If pos is greater or equal to 2 and the character at index pos-2 of locale equals "-", then decrease pos by 2.
        ii. Let locale be the result of calling the substring method of locale with arguments 0 and pos.
    e. Else let locale be "".
2. Return -1.
```

NOTE This algorithm uses only a simple tactic, stripping off subtags, for finding similar locales that may reasonably be used instead of a requested locale. Implementations can mimic the results of other tactics by listing additional locales as available, as discussed in the note of section 6.1.1.

7.4.3 [[Lookup]] (requestedLocales)

The [[Lookup]] internal method compares requestedLocales, which it expects to be a LocaleList object representing a BCP 47 language priority list, against the set of locales in this LocaleList object, and determines the best available language to meet the request. The algorithm is based on the Lookup algorithm described in RFC 4647 section 3.4, but options specified through Unicode extension sequences are ignored in the lookup. Information about such subsequences is returned separately. The internal method returns an object with a locale property, whose value is the language tag of the selected locale. If the language tag of the request locale that led to the selected locale contained a Unicode extension subsequence, then the returned object also contains an extension property whose value is the Unicode extension subsequence (starting with "-u-"), and an extensionIndex property whose value is the index of the Unicode extension subsequence within the request locale language tag.

The following steps are taken:

```
1. Let extensionMatch be null.
2. Let i be 0.
```

3. Let availableIndex be -1.
4. Repeat while i is less than the value of the length property of requestedLocales and availableIndex is -1:
    a. Let locale be the element at index i of requestedLocales.
    b. Let extensionMatch be the result of calling the match method of locale with the argument /-u(-([a-z0-9]{2,8}))+/.
    c. If extensionMatch is not null, then:
        i. Let extension be the value of the element at index 0 of extensionMatch.
        ii. Let extensionIndex be the value of the index property of extensionMatch.
        iii. Let locale be the result of calling the replace method of locale with arguments extension and "".
    d. Let availableIndex be the result of calling the [[IndexOfMatchFor]] method of this LocaleList object, passing locale as the argument.
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 does not equal -1, then:
    a. Call the [[Put]] internal method of result with the arguments "locale", the element at index availableIndex of this LocaleList object, and true.
    b. If extensionMatch is not null, then:
        i. Call the [[Put]] internal method of result with the arguments "extension", extension, and true.
        ii. Call the [[Put]] internal method of result with the arguments "extensionIndex", extensionIndex, and true.
7. Else
    a. Call the [[Put]] internal method of result with the arguments "locale", the value of the [[currentHostLocale]] internal property of the Globalization object, and true.
8. Return result.

7.4.4 [[SupportedLocalesOf]] (requestedLocales)

The [[SupportedLocalesOf]] internal method returns the subset of the provided BCP 47 language priority list for which this LocaleList object has a matching locale. Locales appear in the same order in the returned list as in the input list. The following steps are taken:

1. If the constructor of requestedLocales is not Globalization.LocaleList, then replace requestedLocales with a new LocaleList object as if by the expression new Globalization.LocaleList(requestedLocales), where Globalization.LocaleList is the standard built-in constructor with that name.
2. Let callback be a function that takes the argument locale and performs the following steps:
    a. Let locale be the result of calling the replace method of locale with the arguments /-u(-([a-z0-9]{2,8}))+/ and "".
    b. Let index be the result of calling the [[IndexOfMatchFor]] internal method of this LocaleList object, passing locale as the argument.
    c. If index does not equal -1, then return true, otherwise return false.
3. Let subset be the result of applying the Array.prototype.filter method to requestedLocales, passing the argument list [callback, this].
4. Return the result of creating a new object as if by the expression new Globalization.LocaleList(), where Globalization.LocaleList is the standard built-in constructor with that name, with the argument subset.

7.5 Properties of LocaleList Instances

LocaleList instances inherit properties from the LocaleList prototype object. LocaleList instances also have the following properties.

7.5.1 length

The length property of this LocaleList object is a data property whose value is a numeric value that is one greater than the largest numeric property name.

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

7.5.2 Properties With Numeric Names

A LocaleList object has properties with numeric names 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 }.

7.6 Abstract Operations With LocaleList Objects

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

The ResolveLocale abstract operation compares a BCP 47 language priority list against a set of available locales and determines the best available language to meet the request. The algorithm is based on the Lookup algorithm described in RFC 4647 section 3.4, but options specified through Unicode extension sequences are negotiated separately, taking the caller's relevant extension keys and locale data as well as client-provided options into consideration. The operation expects that the availableLocales argument is a LocaleList object. It 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.

NOTE The core Lookup algorithm is fairly simplistic; it relies on availableLocales being comprehensive. See the note in 6.1.1 for more information.

The following steps are taken:

```
1.  If the constructor of requestedLocales is not Globalization.LocaleList, then replace requestedLocales with a
    new LocaleList object as if by the expression new Globalization.LocaleList(requestedLocales), where
    Globalization.LocaleList is the standard built-in constructor with that name.
2.  Let lookup be the result of calling the [[Lookup]] internal method of availableLocales with the argument
    requestedLocales.
3.  Let foundLocale be the result of calling the [[Get]] internal method of result with the argument "locale".
4.  Let extension be the result of calling the [[Get]] internal method of result with the argument "extension".
5.  If extension is not undefined, then
        a.  Let extensionIndex be the result of calling the [[Get]] internal method of result with the argument
            "extensionIndex".
        b.  Let extensionSubtags be the result of calling the split method of extension with the argument "-".
6.  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.
7.  Call the [[Put]] internal method of result with the arguments "dataLocale", foundLocale, and true.
8.  Let supportedExtension be "-u".
9.  Let i be 0.
10. Repeat while i is less then the value of the length property of relevantExtensions:
        a.  Let key be the element at index i of relevantExtensions.
        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 element at index 0 of keyLocaleData.
        e.  If extensionSubtags is not undefined, then
                i.  Let keyPos be the result of calling the indexOf method of extensionSubtags with argument
                    key.
                ii. If keyPos does not equal -1 and keyPos + 1 is less than or equal to the value of the length
                    property of extensionSubtags, then
                        1. Let requestedValue be the element at index keyPos + 1 of extensionSubtags.
                        2. If the result of calling the indexOf method of keyLocaleData with the argument
                           requestedValue is not -1, then
                                a.  Let value be requestedValue.
            b. Let supportedExtension be the concatenation of supportedExtension, "-",
               key, "-", and value.
      f. If options has an own property with name key, and the result of calling the indexOf method of
         keyLocaleData with the value of the property key of options is not -1, then let value be the value of
         the property key of options.
      g. Call the [[Put]] internal method of result with the arguments key, value, and true.
      h. Increase i by 1.
11. If the value of the length property of supportedExtension is greater than 2, then
      a. Let preExtension be the result of calling the substring method of foundLocale with arguments 0 and
         extensionIndex.
      b. Let postExtension be the result of calling the substring method of foundLocale with argument
         extensionIndex.
      c. Let foundLocale be the concatenation of preExtension, supportedExtension, and postExtension.
12. Call the [[Put]] internal method of result with the arguments "locale", foundLocale, and true.
13. Return result.
```

8 Collator Objects

The Collator constructor is a property of the Globalization object. Behavior common to all constructor properties of the Globalization object is specified in 6.1.

8.1 The Collator Constructor Called as a Function

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

8.2 The Collator Constructor

When Globalization.Collator is called with a this value that is an object whose constructor property is Globalization.Collator itself, it acts as a constructor: it initializes the object.

8.2.1 new Globalization.Collator(localeList, options)

When the Collator constructor is called with two arguments, it computes its effective locale and its collation options from these arguments.

Several steps in the algorithm use values from the following table, which associates Unicode 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 localeList 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.
2. If options has the property usage, then let u be ToString(options.usage); else let u be "sort".
3. If u is not equal to "sort" or "search", throw a ValueError exception.
4. Set the [[usage]] internal property to u.
5. If u is equal to "sort", then let localeData be the value of the [[sortLocaleData]] internal property of Collator; else let localeData be the value of the [[searchLocaleData]] internal property of Collator.
6. 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.
7. For each row in table 1, except the header row, do:
    a. If options has a property with the name given in the Property column of the row, then:
        i. Let value be the result of calling the function with the name given in the Type column of the row, passing as argument the value of the property of options with the name given in the Property column of the row.
        ii. If the name given in the Type column of the row is String, and value is not one of the strings given in the Values column of the row, then throw a ValueError exception.
        iii. Call the [[Put]] internal method of opt, passing as arguments the name given in the Key column of the row, value, and true.
8. Let relevantExtensions be the value of the [[relevantExtensionKeys]] internal property of Collator.
9. Let r be the result of calling the ResolveLocale abstract operation with the [[availableLocales]] internal property of Collator, the localeList argument, relevantExtensions, and localeData.
10. Set the [[locale]] internal property to r.locale.
11. Let i be 0.
12. Repeat while i is less than the value of the length property of relevantExtensions:
    a. Let key be the element at index i of relevantExtensions.
    b. If key is equal to "co", then
        i. Let property be "collation".
        ii. Let value be the value of the property co 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 property named key of r.
        iii. If the name given in the Type column of the row, then let value be the result of comparing value with "true".
    d. Set the internal property named property of the newly constructed object to value.
    e. Increase i by 1.
13. If options has the property sensitivity, then
    a. Let s be ToString(options.sensitivity).
    b. If s is not equal to "base", "accent", "case", or "variant", throw a ValueError exception.
14. Else
    a. If u is equal to "sort", then let s be "variant".
    b. Else
        i. Let dataLocale be the value of the dataLocale property of r.
        ii. Let s be the value of the sensitivity property of the value of the property named dataLocale of localeData.
15. Set the [[sensitivity]] internal property to s.
16. If options has the property ignorePunctuation, then let ip be ToBoolean(options.ignorePunctuation); else let ip be false.
17. Set the [[ignorePunctuation]] internal property of the newly constructed object to ip.

The [[Prototype]] internal property of the newly constructed object is set to the original Collator prototype object, the one that is the initial value of Collator.prototype (8.3.1).

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

8.2.2 new Globalization.Collator(localeList)

When the Collator constructor is called with a single argument, it behaves as if it had received the object {usage: "sort"} as the second argument.

8.2.3 new Globalization.Collator()

When the Collator constructor is called with no argument, it behaves as if it had received the object new LocaleList() as the first argument and the object {usage: "sort"} as the second argument.

8.3 Properties of the Collator Constructor

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

8.3.1 Globalization.Collator.prototype

The initial value of Globalization.Collator.prototype is the built-in Collator prototype object (8.4).

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

8.3.2 Globalization.Collator.supportedLocalesOf(requestedLocales)

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

1. Let availableLocales be the value of the [[availableLocales]] internal property of Collator.
2. Return the result of calling the [[SupportedLocalesOf]] internal method of availableLocales with the argument requestedLocales.

8.3.3 Internal Properties

The initial value of the [[availableLocales]] internal property is implementation dependent within the constraints described in 6.1.1.

The initial 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 Globalization API. Support for the remaining keys is implementation dependent.

The initial values of the [[sortLocaleData]] and [[searchLocaleData]] internal properties are implementation dependent within the constraints described in 6.1.1 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 Boolean value for all locale values.

8.4 Properties of the Collator Prototype Object

The Collator prototype object is itself a Collator object, whose internal properties are set as if it had been constructed using new Collator().

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

8.4.1 Globalization.Collator.prototype.constructor

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

8.4.2 Globalization.Collator.prototype.compare(x, y)

When the compare method is called with two arguments 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 this Collator object, 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.

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 method of any given Collator object, if considered as a function of two arguments x and y, is a consistent comparison function (as defined in the ECMAScript Language Specification, 5.1 edition, 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 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 object.

NOTE 2 It is recommended that the compare method be implemented following Unicode Technical Standard #10, Unicode Collation Algorithm, using tailorings for the effective locale and collation options of this Collator object.

NOTE 3 Applications should not assume that the behavior of the compare methods of Collator instances with the same resolved options will remain the same for different versions of the same implementation.

8.4.3 Globalization.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 Collator. Each property has the value of the corresponding internal property of this Collator object (see 8.5). The [[Set]] attribute is undefined.

8.5 Properties of Collator Instances

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

Collator instances 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 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.

9 NumberFormat Objects

The NumberFormat constructor is a property of the Globalization object. Behavior common to all constructor properties of the Globalization object is specified in 6.1.

9.1 The NumberFormat Constructor Called as a Function

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

9.2 The NumberFormat Constructor

When Globalization.NumberFormat is called with a this value that is an object whose constructor property is Globalization.NumberFormat itself, it acts as a constructor: it initializes the object.

9.2.1 new Globalization.NumberFormat(localeList, options)

When the NumberFormat constructor is called with two arguments, it computes its effective locale and its formatting options from these arguments. The computation of the formatting options uses the abstract operations ToDigits and CurrencyDigits, which are defined below.

The effective locale is computed as follows:

1. If localeList 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.
2. 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.
3. Let r be the result of calling the ResolveLocale abstract operation with the [[availableLocales]] internal property of NumberFormat, the localeList argument, opt, the [[relevantExtensionKeys]] internal property of NumberFormat, and the [[localeData]] internal property of NumberFormat.
4. Set the [[locale]] internal property to r.locale.
5. Set the [[numberingSystem]] internal property to r.nu.

The formatting options are computed as follows:

1. If options has the property style, then let s be ToString(options.style); else let s be "decimal".
2. If s is not equal to "decimal", "currency", or "percent", throw a ValueError exception.
3. Set the [[style]] internal property to s.
4. If options has the property currency, then let c be ToString(options.currency); else let c be undefined.
5. If c is a String value but the result of calling the IsWellFormedCurrencyCode abstract operation with argument c is false, then throw a ValueError exception.
6. If s is equal to "currency" and c is undefined, throw a TypeError exception.
7. If s is equal to "currency" then set the [[currency]] internal property to c.
8. If options has the property currencyDisplay, then let cd be ToString(options.currencyDisplay); else let cd be "symbol".
9. If cd is not equal to "code", "symbol", or "name", throw a ValueError exception.
10. If s is equal to "currency" then set the [[currencyDisplay]] internal property to cd.
11. If options has the property minimumIntegerDigits then let mnid be ToDigits(options.minimumIntegerDigits, 0, 21); else let mnid be 1.
12. Set the [[minimumIntegerDigits]] internal property to mnid.
13. If options has the property minimumFractionDigits then let mnfd be ToDigits(options.minimumFractionDigits, 0, 20); else if s is equal to "currency" then let mnfd be CurrencyDigits(c); else let mnfd be 0.
14. Set the [[minimumFractionDigits]] internal property to mnfd.
15. If options has the property maximumFractionDigits then let mxfd be ToDigits(options.maximumFractionDigits, mnfd, 20); else if s is equal to "currency" then let mxfd be max(mnfd, CurrencyDigits(c)); else if s is equal to "percent" then let mxfd be max(mnfd, 0); else let mxfd be max(mnfd, 3).
16. Set the [[maximumFractionDigits]] internal property to mxfd.
17. Delete the [[minimumSignificantDigits]] and [[maximumSignificantDigits]] internal properties.
18. If options has at least one of the properties minimumSignificantDigits and maximumSignificantDigits, then:
    a. If options has the property minimumSignificantDigits then let mnsd be ToDigits(options.minimumSignificantDigits, 1, 21); else let mnsd be 1.
    b. If options has the property maximumSignificantDigits then let mxsd be ToDigits(options.maximumSignificantDigits, mnsd, 21); else let mxsd be 21.
    c. Set the [[minimumSignificantDigits]] internal property to mnsd, and the [[maximumSignificantDigits]] internal property to mxsd.
19. If options has the property useGrouping then let g be ToBoolean(options.useGrouping); else let g be true.
20. Set the [[useGrouping]] internal property to g.

When the ToDigits abstract operation is called with three arguments value, minimum, maximum (the latter two are expected to be Number values greater than or equal to 0), the following steps are taken:

1. Let v be ToNumber(value).
2. If v is NaN or less than minimum or greater than maximum, throw a RangeError exception.
3. Return floor(v).

When the CurrencyDigits abstract operation is called with an argument currency (which is expected to be a 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.

The [[Prototype]] internal property of the newly constructed object is set to the original NumberFormat prototype object, the one that is the initial value of NumberFormat.prototype (9.3.1).

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

9.2.2 new Globalization.NumberFormat(localeList)

When the NumberFormat constructor is called with a single argument, it behaves as if it had received the object {style: "decimal"} as the second argument.

9.2.3 new Globalization.NumberFormat()

When the NumberFormat constructor is called with no argument, it behaves as if it had received the object new LocaleList() as the first argument and the object {style: "decimal"} as the second argument.

9.3 Properties of the NumberFormat Constructor

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

9.3.1 Globalization.NumberFormat.prototype

The initial value of Globalization.NumberFormat.prototype is the built-in NumberFormat prototype object (9.4).

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

9.3.2 Globalization.NumberFormat.supportedLocalesOf(requestedLocales)

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

```
1. Let availableLocales be the value of the [[availableLocales]] internal property of NumberFormat.
2. Return the result of calling the [[SupportedLocalesOf]] internal method of availableLocales with the
   argument requestedLocales.
```

9.3.3 Internal Properties

The initial values of the [[availableLocales]] and [[localeData]] internal properties are implementation dependent within the constraints described in 6.1.1.

The initial 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. NumberFormat, however, requires that the currency of a currency format is specified through the currency property in the options objects.

9.4 Properties of the NumberFormat Prototype Object

The NumberFormat prototype object is itself a NumberFormat object, whose internal properties are set as if it had been constructed using new NumberFormat().

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

9.4.1 Globalization.NumberFormat.prototype.constructor

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

9.4.2 Globalization.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.

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. Let x be ToNumber(value).
2. If the result of isFinite(x) is false, then
    a. If x is NaN, then let result be an ILD String value indicating the NaN value.
    b. Else if x > 0, then let result be an ILD String value indicating positive infinity.
    c. Else let result be an ILD String value indicating negative infinity.
3. Else
    a. Let negative be false.
    b. If x equals -0, then let x be +0.
    c. If x < 0, then
        i. Let negative be true.
        ii. Let x be -x.
    d. If the value of the [[style]] internal property of this NumberFormat is "percent", let x be 100 * x.
    e. If the [[minimumSignificantDigits]] and [[maximumSignificantDigits]] internal properties of this NumberFormat are present, then
        i. To do: Let result be a String representation of x with at least [[minimumSignificantDigits]] and at most [[maximumSignificantDigits]] significant digits.
           If the complete representation of value would require more significant digits than the formatting options allow, the function rounds to the nearest value that can be represented with the permitted significant digits; if the given value falls midway, it rounds away from zero.
           NOTE With one allowed significant digit, 0,44 rounds to 0,4; 0,45 rounds to 0,5; -0,44 rounds to -0,4; -0,45 rounds to -0,5.
    f. Else
        i. To do: Let result be a String representation of x with at least [[minimumFractionDigits]] and at most [[maximumFractionDigits]] fraction digits, and at least [[minimumIntegerDigits]] integer digits.
           If the complete representation of value would require more fraction digits than the formatting options allow, the function rounds to the nearest value that can be represented with the permitted fraction digits; if the given value falls midway, it rounds away from zero.
           NOTE With one allowed fraction digit, 0,44 rounds to 0,4; 0,45 rounds to 0,5; -0,44 rounds to -0,4; -0,45 rounds to -0,5.
    g. If the value of the [[numberingSystem]] internal property of this NumberFormat matches one of the values in the “Numbering System” column of Table 2 below, then
        i. Let digits be the String value in the “Digits” column of Table 2 in the row containing the value of the [[numberingSystem]] internal property.
        ii. Replace each digit in result with the value of digits[digit].
    h. Else use an implementation dependent algorithm to map result to the appropriate representation of result in the given numbering system.
    i. If result contains the character ".", then replace it with an ILND String representing the decimal separator.
    j. If the value of the [[useGrouping]] internal property of this NumberFormat is true, then insert an ILND String representing a grouping separator into an ILND set of locations within the integer part of result.
    k. If negative is true, then insert one or two ILND Strings into result in ILND locations to indicate that the value is negative, possibly adding whitespace. The inserted Strings may also depend on whether the [[style]] internal property of this NumberFormat is "currency".
    l. Else insert one or two ILND Strings into result in ILND locations to indicate that the value is positive, possibly adding whitespace. The inserted Strings may also depend on whether the [[style]] internal property of this NumberFormat is "currency".
4. If the [[style]] internal property of this NumberFormat is "percent", then insert an ILND String indicating percentage into result in an ILND location, possibly adding whitespace.
5. Else if the [[style]] internal property of this NumberFormat is "currency", then:
    a. Let currency be the value of the [[currency]] internal property of this NumberFormat.
    b. If the value of the [[currencyDisplay]] internal property of this NumberFormat is "code", then let cd be currency.
    c. Else if the value of the [[currencyDisplay]] internal property of this NumberFormat 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 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. Insert cd into result in an ILD location, possibly adding whitespace.
6. Return result.

Table 2 – Numbering systems with simple digit mappings

Numbering System	Digits
arab	٠١٢٣٤٥٦٧٨٩
arabext	۰۱۲۳۴۵۶۷۸۹
beng	□□□□□□□□□□
deva	□□□□□□□□□□
fullwide	０１２３４５６７８９
guir	□□□□□□□□□□
guru	□□□□□□□□□□
hanidec	一二三四五六七八九
khmr	□□□□□□□□□□
knda	□□□□□□□□□□
laoo	□□□□□□□□□□
latn	0123456789
mlym	□□□□□□□□□□
mong	᠐᠑᠒᠓᠔᠕᠖᠗᠘᠙
mymr	□□□□□□□□□□
orya	□□□□□□□□□□
tamldec	□□□□□□□□□□
telu	□□□□□□□□□□
thai	□□□□□□□□□□
tibt	༠༡༢༣༤༥༦༧༨༩

9.4.3 Globalization.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 9.5). The [[Set]] attribute is undefined.

9.5 Properties of NumberFormat Instances

NumberFormat instances inherit properties from the NumberFormat prototype object. NumberFormat instances 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.
- [[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.

10 DateTimeFormat Objects

The DateTimeFormat constructor is a property of the Globalization object. Behavior common to all constructor properties of the Globalization object is specified in 6.1.

10.1 The DateTimeFormat Constructor Called as a Function

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

10.2 The DateTimeFormat Constructor

When Globalization.DateTimeFormat is called with a this value that is an object whose constructor property is Globalization.DateTimeFormat itself, it acts as a constructor: it initializes the object.

10.2.1 new Globalization.DateTimeFormat(localeList, options)

When the DateTimeFormat constructor is called with two arguments, it computes its effective locale and its formatting options from these arguments. The computation of the formatting options and effective locale uses the abstract operation ToBestMatch, which is 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 effective locale is computed as follows:

1. If localeList 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.
2. 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.
3. Let localeData be the value of the [[localeData]] internal property of DateTimeFormat.
4. Let r be the result of calling the ResolveLocale abstract operation with the [[availableLocales]] internal property of DateTimeFormat, the localeList argument, opt, the [[relevantExtensionKeys]] internal property of DateTimeFormat, and localeData.
5. Set the [[locale]] internal property to r.locale.
6. Set the [[calendar]] internal property to r.ca.
7. Set the [[numberingSystem]] internal property to r.nu.

The formatting options are computed as follows:

1. Let dataLocale be the value of the dataLocale property of r.
2. If options has the property timeZone, then let tz be ToString(options.timeZone).toUpperCase(); else let tz be undefined.
3. If s is a String value but not equal to "UTC", throw a ValueError exception.
4. Set the [[timeZone]] internal property to tz.
5. If options has the property hour12, then let hr12 be ToBoolean(options.hour12); else let hr12 be the value of the hour12 property of the property named dataLocale of localeData.
6. Set the [[hour12]] internal property to hr12.
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. For each row of table 3, except the header row, do:
    a. If options has a property with the name given in the Property column of the row, then:
        i. Let value be the result of calling ToString with the value of the property of options with the name given in the Property column of the row.
        ii. If value is not one of the strings given in the Values column of the row, then throw a ValueError exception.

```
            iii. Call the [[Put]] internal method of opt, passing as arguments the name given in the Property
                 column of the row, value, and true.
9.  Let formats be the value of the formats property of the property named dataLocale of localeData.
10. Let bestFormat be the result of calling ToBestMatch with opt and formats.
11. For each row in table 1, except the header row, do
        a. Let p be the value of the property of bestFormat with the name given in the Property column of the
           row.
        b. If p is not undefined, then set the internal property of the newly constructed object with the name
           given in the Property column of the row to p.
12. Set the [[pattern]] internal property to the value of the pattern property of bestFormat.
```

When the ToBestMatch 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 shortMorePenalty be 3.
4.  Let shortLessPenalty be 6.
5.  Let longMorePenalty be 6.
6.  Let longLessPenalty be 8.
7.  Let bestScore be -Infinity.
8.  Let bestFormat be undefined.
9.  Let i be 0.
10. Repeat while i is less than the value of the length property of formats:
        a. Let format be the element at index i of formats.
        b. Let score be 0.
        c. For each property shown in table 3:
              i. Let optionsProp be options[property].
             ii. Let formatProp be format[property].
            iii. If optionsProp === undefined and formatProp !== undefined, then score -= additionPenalty.
             iv. Else if optionsProp !== undefined and formatProp === undefined, then score -=
                 removalPenalty.
              v. 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 equals 2, decrease score by longMorePenalty.
                     6. Else if delta equals 1, decrease score by shortMorePenalty.
                     7. Else if delta equals -1, decrease score by shortLessPenalty.
                     8. Else if delta equals -2, decrease score by longLessPenalty.
        d. If score is greater than bestScore, then
              i. Let bestScore be score.
             ii. Let bestFormat be format.
        e. Increase i by 1.
11. Return bestFormat.
```

The [[Prototype]] internal property of the newly constructed object is set to the original DateTimeFormat prototype object, the one that is the initial value of DateTimeFormat.prototype (10.3.1).

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

10.2.2 new Globalization.DateTimeFormat(localeList)

When the DateTimeFormat constructor is called with a single argument, it behaves as if it had received the object {year: "numeric", month: "numeric", day: "numeric"} as the second argument.

10.2.3 new Globalization.DateTimeFormat()

When the DateTimeFormat constructor is called with no arguments, it behaves as if it had received the object new LocaleList() as the first argument and the object {year: "numeric", month: "numeric", day: "numeric"} as the second argument.

10.3 Properties of the DateTimeFormat Constructor

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

10.3.1 Globalization.DateTimeFormat.prototype

The initial value of Globalization.DateTimeFormat.prototype is the built-in DateTimeFormat prototype object (10.4).

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

10.3.2 Globalization.DateTimeFormat.supportedLocalesOf(requestedLocales)

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

1. Let availableLocales be the value of the [[availableLocales]] internal property of DateTimeFormat.
2. Return the result of calling the [[SupportedLocalesOf]] internal method of availableLocales with the argument requestedLocales.

10.3.3 Internal Properties

The initial value of the [[availableLocales]] internal property is implementation dependent within the constraints described in 6.1.1.

The initial 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.

The initial value of the [[ocaleData]] internal property is implementation dependent within the constraints described in 6.1.1 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: "numeric", second: "numeric", pattern: "{hour}:{minute}:{second}"}.

10.4 Properties of the DateTimeFormat Prototype Object

The DateTimeFormat prototype object is itself a DateTimeFormat object, whose internal properties are set as if it had been constructed using new DateTimeFormat().

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

10.4.1 Globalization.DateTimeFormat.prototype.constructor

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

10.4.2 Globalization.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.

```
1. If date is omitted let y be Date.now(); else let y be ToNumber(date).
2. If y is not a finite Number, then throw a RangeError exception.
3. Let locale be the value of the [[locale]] internal property of this DateTimeFormat object.
4. Let nf be the result of creating a new NumberFormat object as if by the expression new
   Globalization.NumberFormat(locale) where Globalization.NumberFormat is the standard built-in constructor
   with that name.
5. Let nf2 be the result of creating a new NumberFormat object as if by the expression new
   Globalization.NumberFormat(locale, {minimumIntegerDigits: 2}) where Globalization.NumberFormat is the
   standard built-in constructor with that name.
6. 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.
7. Let result be the value of the [[pattern]] internal property of this DateTimeFormat object.
8. For each row of table 3, except the header row, do:
    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 internal property named p of this DateTimeFormat object.
       iii. Let v be the value of the property named p of tm.
        iv. If p equals "month", then increase v by 1.
         v. If f equals "numeric", then let fv be the result of calling nf.format(v).
        vi. Else if f equals "2-digit", then
              1. Let fv be the result of calling nf2.format(v).
              2. If the length of fv is greater than 2, let fv be fv.substring(length-2).
       vii. 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.
      viii. Replace the substring of result that consists of "{", p, and "}", with fv.
9. 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 for years after
   1930 are expected to match the algorithms specified in the ECMAScript Language Specification, 5.1 edition,
```

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 the ECMAScript Language Specification, 5.1 edition, 15.9.1.7 and 15.9.1.8. For earlier years, calculations may switch to the Julian calendar at points that may depend on the locale.
2. Return an object with properties weekday, era, year, month, day, hour, minute, second, and inDST, each with the corresponding calculated value.

10.4.3 Globalization.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 10.5). The [[Set]] attribute is undefined.

10.5 Properties of DateTimeFormat Instances

DateTimeFormat instances inherit properties from the DateTimeFormat prototype object. DateTimeFormat instances 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 10.3.3.

11 Error Objects

This clause specifies error objects in addition to those described in the ECMAScript Language Specification, 5.1 edition, clause 15. It only applies to implementations that are based on a version of the ECMAScript Language Specification that does not itself specify the error objects described here.

11.1 Native Error Types Used in This Standard

One of the NativeError objects described below or in the ECMAScript Language Specification, 5.1 edition, 15.11.6, or successor, is thrown when a runtime error is detected. All of these objects share the same structure, as described in the ECMAScript Language Specification, 5.1 edition, 15.11.7, or successor.

11.1.1 ValueError

Indicates a string value is invalid. See [to do: insert subclause references].