Intl.DisplayNames v2 for Stage 3

Frank Yung-Fong Tang / 譚永鋒
ftang@google.com

TC39 April 2021 Meeting
April 19-22, 2021

Slide: https://docs.google.com/presentation/d/1_BR2bq6gi_i9QjDDluv683cuO2AXNwZl-3hXC4gLl3M
Repo: https://github.com/tc39/intl-displaynames-v2

History of Intl.DisplayNames

- Aug 2020: v2 repo is created as Stage 0 to track additional feature requests (will be discussed later)
- Sep 2020: Advanced v2 to Stage 1 in TC39 meeting.
- Jan 2021: Advanced v2 **to Stage 2**
- April 2021: Proposed v2 to **Stage 3**

Motivation

To enable developers to get **human translation** of language, region, script, and other display names on the client.

Also, provide **a straightforward API** to provide the functionality to reduce developers to use **“work in some context but not other” work around**.

Change after ECMA402 Jan Stage 2 Discussion

Draft Spec https://tc39.es/intl-displaynames-v2/

- Calendar
- Unit
- **Spec out Dialect Support**
- **Add names for “Date Time Fields”**

Change to ECMA402 - Part 1

1.3.1 Intl.DisplayNames (*locales*, *options*)

When the **Intl.DisplayNames** function is called with arguments *locales* and *options*, the following steps are taken:

1. If NewTarget is **undefined**, throw a **TypeError** exception.
2. Let *displayNames* be ? OrdinaryCreateFromConstructor(NewTarget, `"%DisplayNamesPrototype%"`, « [[InitializedDisplayNames]], [[Locale]], [[Style]], [[Type]], [[Fallback]], [[DialectHandling]], [[Fields]] »).
3. Let *requestedLocales* be ? CanonicalizeLocaleList(*locales*).
4. Let *options* be ? ToObject(*options*).
5. Let *opt* be a new Record.
6. Let *localeData* be %DisplayNames%.[[LocaleData]].
7. Let *matcher* be ? GetOption(*options*, `"localeMatcher"`, `"string"`, « `"lookup"`, `"best fit"` », `"best fit"`).
8. Set *opt*.[[localeMatcher]] to *matcher*.
9. Let *r* be ResolveLocale(%DisplayNames%.[[AvailableLocales]], *requestedLocales*, *opt*, %DisplayNames%.[[RelevantExtensionKeys]]).
10. Let *style* be ? GetOption(*options*, `"style"`, `"string"`, « `"narrow"`, `"short"`, `"long"` », `"long"`).
11. Set *displayNames*.[[Style]] to *style*.
12. Let *type* be ? GetOption(*options*, `"type"`, `"string"`, « `"language"`, `"region"`, `"script"`, `"currency"`, `"calendar"`, `"dateTimeField"`, `"unit"`», **undefined**).
13. If *type* is **undefined**, throw a **TypeError** exception.
14. Set *displayNames*.[[Type]] to *type*.
15. Let *fallback* be ? GetOption(*options*, `"fallback"`, `"string"`, « `"code"`, `"none"` », `"code"`).
16. Set *displayNames*.[[Fallback]] to *fallback*.
17. Set *displayNames*.[[Locale]] to the value of *r*.[[Locale]].
18. Let *dataLocale* be *r*.[[dataLocale]].
19. Let *dataLocaleData* be *localeData*.[[<*dataLocale*>]].
20. Let *types* be *dataLocaleData*.[[types]].
21. Assert: *types* is a Record (see 1.4.3).
22. Let *typeFields* be *types*.[[<*type*>]].
23. Assert: *typeFields* is a Record (see 1.4.3).
24. If *type* is `"language"`, then
    a. Let *dialectHandling* be ? GetOption(*options*, `"dialectHandling"`, `"string"`, « `"dialectName"`, `"standardName"` », `"dialectName"`).
    b. Set *displayNames*.[[DialectHandling]] to *dialectHandling*.
    c. Let *typeFields* be *typeFields*.[[<*dialectHandling*>]].
    d. Assert: *typeFields* is a Record (see 1.4.3).
25. Let *styleFields* be *typeFields*.[[<*style*>]].
26. Assert: *styleFields* is a Record (see 1.4.3).
27. Set *displayNames*.[[Fields]] to *styleFields*.
28. Return *displayNames*.

Examples of Dialect Handling


```
d8> dn2 = new
Intl.DisplayNames("en",
{type: "language",
 dialectHandling: "dialectName"})
d8> dn2.of("en")
"English"
d8> dn2.of("en-GB")
"British English"
d8> dn2.of("en-US")
"American English"
d8> dn2.of("en-AU")
"Australian English"
d8> dn2.of("en-CA")
"Canadian English"
d8> dn2.of("zh")
"Chinese"
d8> dn2.of("zh-Hant")
"Traditional Chinese"
d8> dn2.of("zh-Hans")
"Simplified Chinese"
```

```
d8> dn3 = new
Intl.DisplayNames("en",
{type: "language",
 dialectHandling: "standardName"})
d8> dn3.of("en")
"English"
d8> dn3.of("en-GB")
"English (United Kingdom)"
d8> dn3.of("en-AU")
"English (Australia)"
d8> dn3.of("en-CA")
"English (Canada)"
d8> dn3.of("en-US")
"English (United States)"
d8> dn3.of("zh")
"Chinese"
d8> dn3.of("zh-Hant")
"Chinese (Traditional)"
d8> dn3.of("zh-Hans")
"Chinese (Simplified)"
```

1.1.1 CanonicalCodeForDisplayNames (*type*, *code*)

Change to ECMA402 - Part 2

The CanonicalCodeForDisplayNames abstract operation is called with arguments *type*, *code*. It verifies that the *code* argument r formed code according to the *type* argument and returns the case-regularized form of the *code*. The algorithm refers to UTS 35's and Locale Identifiers grammar. The following steps are taken:

1. If *type* is "language", then
    a. If *code* does not matches the unicode_language_id production, throw a **RangeError** exceptipon.
    b. If IsStructurallyValidLanguageTag(*code*) is **false**, throw a **RangeError** exception.
    c. Set *code* to CanonicalizeUnicodeLocaleId(*code*).
    d. Return *code*.
2. If *type* is "region", then
    a. If *code* does not matches the unicode_region_subtag production, throw a **RangeError** exceptipon.
    b. Let *code* be the result of mapping *code* to upper case as described in 6.1.
    c. Return *code*.
3. If *type* is "script", then
    a. If *code* does not matches the unicode_script_subtag production, throw a **RangeError** exceptipon.
    b. Let *code* be the result of mapping the first character in *code* to upper case, and mapping the second, third and fourth c lower case, as described in 6.1.
    c. Return *code*.
4. If *type* is "calendar", then
    a. If *code* does not match the Unicode Locale Identifier type nonterminal, throw a **RangeError** exception.
    b. Let *code* be the result of mapping *code* to lower case as described in 6.1.
    c. Return *code*.
5. If *type* is "unit", then
    a. If the result of IsWellFormedUnitIdentifier(*code*) is **false**, throw a **RangeError** exception.
    b. Return *code*.
6. If *type* is "dateTimeField", then
    a. If the result of IsValidDateTimeFieldCode(*code*) is **false**, throw a **RangeError** exception.
    b. Return *code*.
7. Assert: *type* is "currency".
8. If ! IsWellFormedCurrencyCode(*code*) is **false**, throw a **RangeError** exceptipon.
9. Let *code* be the result of mapping *code* to upper case as described in 6.1.
10. Return *code*.

Example of type: “calendar”

```
d8> dn = new
Intl.DisplayNames("en",
{type: "calendar"})
d8> dn.of("roc")
"Minguo Calendar"
d8> dn.of("persian")
"Persian Calendar"
d8> dn.of("gregory")
"Gregorian Calendar"
d8> dn.of("ethioaa")
"Ethiopic Amete Alem Calendar"
d8> dn.of("japanese")
"Japanese Calendar"
d8> dn.of("dangi")
"Dangi Calendar"
d8> dn.of("chinese")
"Chinese Calendar"
```

```
d8>  dn = new
Intl.DisplayNames("zh",
{type: "calendar"})
d8> dn.of("roc")
"民国纪年"
d8> dn.of("persian")
"波斯历"
d8> dn.of("gregory")
"公历"
d8> dn.of("ethioaa")
"埃塞俄比亚阿米特阿莱姆日历"
d8> dn.of("japanese")
"和历"
d8> dn.of("dangi")
"檀纪历"
8> dn.of("chinese")
"农历"
```

Example of type: “unit”

```
d8> dn1 = new Intl.DisplayNames("zh-Hant", {type: "unit"})
d8> dn1.of("meter")
"公尺"
d8> dn1.of("degree")
"角度"
d8> dn1.of("kilogram")
"公斤"

d8> dn2 = new Intl.DisplayNames("fr", {type: "unit"})
d8> dn2.of("meter")
"mètres"
d8> dn2.of("degree")
"degrés"
d8> dn2.of("kilogram")
"kilogrammes"
```

Change to ECMA402 - Part 3

1.2 IsValidDateTimeFieldCode (*field*)

The IsValidDateTimeFieldCode abstract operation is called with arguments *field*. It verifies that the *field* argument represents a valid date time field code. The following steps are taken:

1. If *field* is listed in Table 1, return **true**.
2. Return **false**.

Table 1: Codes For Date Time Field of DisplayNames

Code	Description
`"era"`	The field indicating the era, e.g., AD or BC in the Gregorian (Julian) calendar.
`"year"`	The field indicating the year.
`"quarter"`	The field indicating the quarter, e.g., Q2, 2nd quarter, etc.
`"month"`	The field indicating the month, e.g. Sep, September, etc.
`"weekOfYear"`	The field indicating the week number within the current year.
`"weekday"`	The field indicating the day of week, e.g. Tue, Tuesday, etc.
`"day"`	The field indicating the day in month.
`"dayPeriod"`	The field indicating the day period, either am/pm marker or others, e.g. noon, evening.
`"hour"`	The field indicating the hour in day.
`"minute"`	The field indicating the minute in hour.
`"second"`	The field indicating the second in minute.
`"timeZoneName"`	The field indicating the timezone name, e.g. PDT, Pacific Daylight Time, etc.

Example of type: “dateTimeField”

```
d8> dn = new Intl.DisplayNames("zh",
{type: "dateTimeField"})
d8> dn.of("era")
"纪元"
d8> dn.of("year")
"年"
d8> dn.of("month")
"月"
d8> dn.of("quarter")
"季度"
d8> dn.of("weekOfYear")
"周"
d8> dn.of("weekday")
"工作日"
d8> dn.of("dayPeriod")
"上午/下午"
d8> dn.of("day")
"日"
d8> dn.of("hour")
"小时"
d8> dn.of("minute")
"分钟"
d8> dn.of("second")
"秒"
```

```
d8> dn = new Intl.DisplayNames("es",
{type: "dateTimeField"})
d8> dn.of("era")
"era"
d8> dn.of("year")
"año"
d8> dn.of("month")
"mes"
d8> dn.of("quarter")
"trimestre"
d8> dn.of("weekOfYear")
"semana"
d8> dn.of("weekday")
"día de la semana"
d8> dn.of("dayPeriod")
"a. m./p. m."
d8> dn.of("day")
"día"
d8> dn.of("hour")
"hora"
d8> dn.of("minute")
"minuto"
d8> dn.of("second")
"segundo"
```

1.4.3 Internal slots

Change to ECMA402 - Part 4

The value of the [[AvailableLocales]] internal slot is implementation defined within the constraints described in 9.1.

The value of the [[RelevantExtensionKeys]] internal slot is « ».

The value of the [[LocaleData]] internal slot is implementation defined within the constraints described in 9.1 and the following additional constraints:

- [[LocaleData]].[[<*locale*>]] must have a [[types]] field for all locale values *locale*. The value of this field must be a Record, which must have fields with the names of one of the valid display name types: `"language"`, `"region"`, `"script"`, ~~and~~ `"currency"`, `"calendar"`, `"dateTimeField"`, and `"unit"`.
- The value of the field `"language"` must be a Record which must have fields with the names of one of the valid display name dialect handlings: `"dialectName"`, and `"standardName"`.
- The display name dialect handling fields under display name type `"language"` should contain Records which must have fields with the names of one of the valid display name styles: `"narrow"`, `"short"`, and `"long"`.
- The value of fields ~~`"language",`~~ `"region"`, `"script"`, ~~and~~ `"currency"`, `"calendar"`, `"dateTimeField"`, and `"unit"` must be a Record which must have fields with the names of one of the valid display name styles: `"narrow"`, `"short"`, and `"long"`.
- The display name styles fields under display name type `"language"` should contain Records, with keys corresponding to language codes according to `unicode_language_id` production. The value of these fields must be string values.
- The display name styles fields under display name type `"region"` should contain Records, with keys corresponding to region codes. The value of these fields must be string values.
- The display name styles fields under display name type `"script"` should contain Records, with keys corresponding to script codes. The value of these fields must be string values.
- The display name styles fields under display name type `"currency"` should contain Records, with keys corresponding to currency codes. The value of these fields must be string values.
- The display name styles fields under display name type `"calendar"` should contain Records, with keys corresponding to a String value with the "type" given in Unicode Technical Standard 35 for the calendar used for formatting. The value of these fields must be string values.
- The display name styles fields under display name type `"dateTimeField"` should contain Records, with keys corresponding to codes listed in Table 1. The value of these fields must be string values.
- The display name styles fields under display name type `"unit"` should contain Records, with keys corresponding to a well-formed core unit identifier as defined in UTS #35, Part 2, Section 6.. The value of these fields must be string values.

1.5.4 Intl.DisplayNames.prototype.resolvedOptions ( )

This function provides access to the locale and options computed during initialization of the object.

1. Let *displayNames* be **this** value.
2. If Type(*displayNames*) is not Object, throw a **TypeError** exception.
3. If *displayNames* does not have an [[InitializedDisplayNames]] internal slot, throw a **TypeError** exception.
4. Let *options* be ! ObjectCreate(%ObjectPrototype%).
5. For each row of Table 2, except the header row, in table order, do
    a. Let *p* be the Property value of the current row.
    b. Let *v* be the value of *displayNames*'s internal slot whose name is the Internal Slot value of the current row.
    c. If *v* is not **undefined**, then
        i. Perform ! CreateDataPropertyOrThrow(*options*, *p*, *v*).
6. Return *options*.

Table 2: Resolved Options of DisplayNames Instances

Internal Slot	Property
[[Locale]]	`"locale"`
[[Style]]	`"style"`
[[Type]]	`"type"`
[[Fallback]]	`"fallback"`
[[DialectHandling]]	`"dialectHandling"`

1.6 Properties of Intl.DisplayNames Instances

Intl.DisplayNames instances are ordinary objects that inherit properties from %DisplayNamesPrototype%.

Intl.DisplayNames instances have an [[InitializedDisplayNames]] internal slot.

Intl.DisplayNames instances also have several internal slots that are computed by the constructor:

- [[Locale]] is a String value with the language tag of the locale whose localization is used for formatting.
- [[Style]] is one of the String values `"narrow"`, `"short"`, or `"long"`, identifying the display names style used.
- [[Type]] is one of the String values `"language"`, `"region"`, `"script"`, ~~or~~ `"currency"`, `"calendar"`, `"dateTimeField"`, or `"unit"`, identifying the type of the display names requested.
- [[Fallback]] is one of the String values `"code"`, or `"none"`, identifying the fallback return when the system does not have the requested display name.
- [[DialectHandling]] is one of the String values `"dialectName"`, or `"standardName"`, identifying the display names dialect handling while and only while the [[Type]] is `"language"`.
- [[Fields]] is a Record (see 1.4.3) which must have fields with keys corresponding to codes according to [[Style]] ~~and~~, [[Type]], and [[DialectHandling]].

References

- **Reviewers**:
  - Shane Carr
  - Ujjwal Sharma
- **Editor**: Richard Gibson
- **V8 prototype**: https://chromium-review.googlesource.com/c/v8/v8/+/2335890
- **Mozilla**:
- **JSC**:

ECMA402 2021-04-09 Monthly Meeting

Shane Carr and Ujjwal Sharma sign up for Stage 3 reviewers

Agree to support champion to bring to TC39 April meeting for Stage 3 advancement.

Entrance Criteria / Acceptance Signifies For Stage 3

Entrance Criteria:

- Complete spec text **DONE**
- Designated reviewers have signed off on the current spec text **DONE**
- All ECMAScript editors have signed off on the current spec text
- All Entrance Criteria for State 2 **DONE** (see next slide)

Acceptance Signifies:

The solution is complete and no further work is possible without implementation experience, significant usage and external feedback.

Requesting the Committee Approval for advancement to Stage 3

Entrance Criteria / Acceptance Signifies For Stage 2

Entrance Criteria:

- Initial spec text **DONE** https://tc39.es/intl-displaynames-v2/
- All Entrance Criteria for State 1 **DONE** (see next slide)

Acceptance Signifies:

- Stage 1: “The committee expects to devote time to examining the problem space, solutions and cross-cutting concerns”
- **Stage 2**: “The committee expects the feature to be developed and eventually included in the standard”

Entrance Criteria For Stage 1

- Identified “champion” who will advance the addition: **DONE-@FrankYFTang**
- Prose outlining the problem or need and the general shape of a solution **DONE**
- Illustrative examples of usage **DONE**
- High-level API **DONE**
- Discussion of key algorithms, abstractions and semantics **DONE**
- Identification of potential “cross-cutting” concerns and implementation challenges/complexity **DONE**
- A publicly available repository for the proposal that captures the above requirements: **DONE**
- https://github.com/tc39/intl-displaynames-v2