Extend TimeZoneName Option Proposal for Stage 3

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

TC39 May 2021 Meeting
May 25-26, 2021

Slide: https://docs.google.com/presentation/d/1N4QoCxFVM4ZKr9gDnaDDnrHb-5_rPTy-wydp-f90xBM
Repo: https://github.com/tc39/proposal-intl-extend-timezonename/

Motivation / Scope

Extend the timeZoneName option in Intl.DateTimeFormat object to support more formatted options.

Add four new values to TimeZoneName option in Intl.DateTimeFormat: “short”, “long”, “shortOffset”, “longOffset”, “shortGeneric”, “longGeneric”

1.1 Abstract Operations For DateTimeFormat Objects

Several DateTimeFormat algorithms use values from the following table, which provides internal slots, property names and allowable values for the components of date and time formats:

Table 1: Components of date and time formats

Internal Slot	Property	Values
[[Weekday]]	"weekday"	"narrow", "short", "long"
[[Era]]	"era"	"narrow", "short", "long"
[[Year]]	"year"	"2-digit", "numeric"
[[Month]]	"month"	"2-digit", "numeric", "narrow", "short", "long"
[[Day]]	"day"	"2-digit", "numeric"
[[Hour]]	"hour"	"2-digit", "numeric"
[[Minute]]	"minute"	"2-digit", "numeric"
[[Second]]	"second"	"2-digit", "numeric"
[[TimeZoneName]]	"timeZoneName"	"short", "long", "shortOffset", "longOffset", "shortWall", "longWall"

Sample Usage in English Locale

```
d8> let timeZoneNames = ["short", "long",
"shortOffset", "longOffset", "shortGeneric",
"longGeneric"];

d8> timeZoneNames.forEach(function(timeZoneName) {
print((new Date()).toLocaleTimeString("en",
{timeZoneName}))});
9:27:14 AM PST
9:27:14 AM Pacific Standard Time
9:27:14 AM GMT-8
9:27:14 AM GMT-08:00
9:27:14 AM PT
9:27:14 AM Pacific Time
```

Sample Usage in Traditional Chinese Locale (zh-CN)

```
d8> let timeZoneNames = ["short", "long",
"shortOffset", "longOffset", "shortGeneric",
"longGeneric"];

d8> timeZoneNames.forEach(function(timeZoneName) {
print((new Date()).toLocaleTimeString("zh-Hant",
{timeZoneName}))});
上午9:27:27 [PST]
上午9:27:27 [太平洋標準時間]
上午9:27:27 [GMT-8]
上午9:27:27 [GMT-08:00]
上午9:27:27 [PT]
上午9:27:27 [太平洋時間]
```

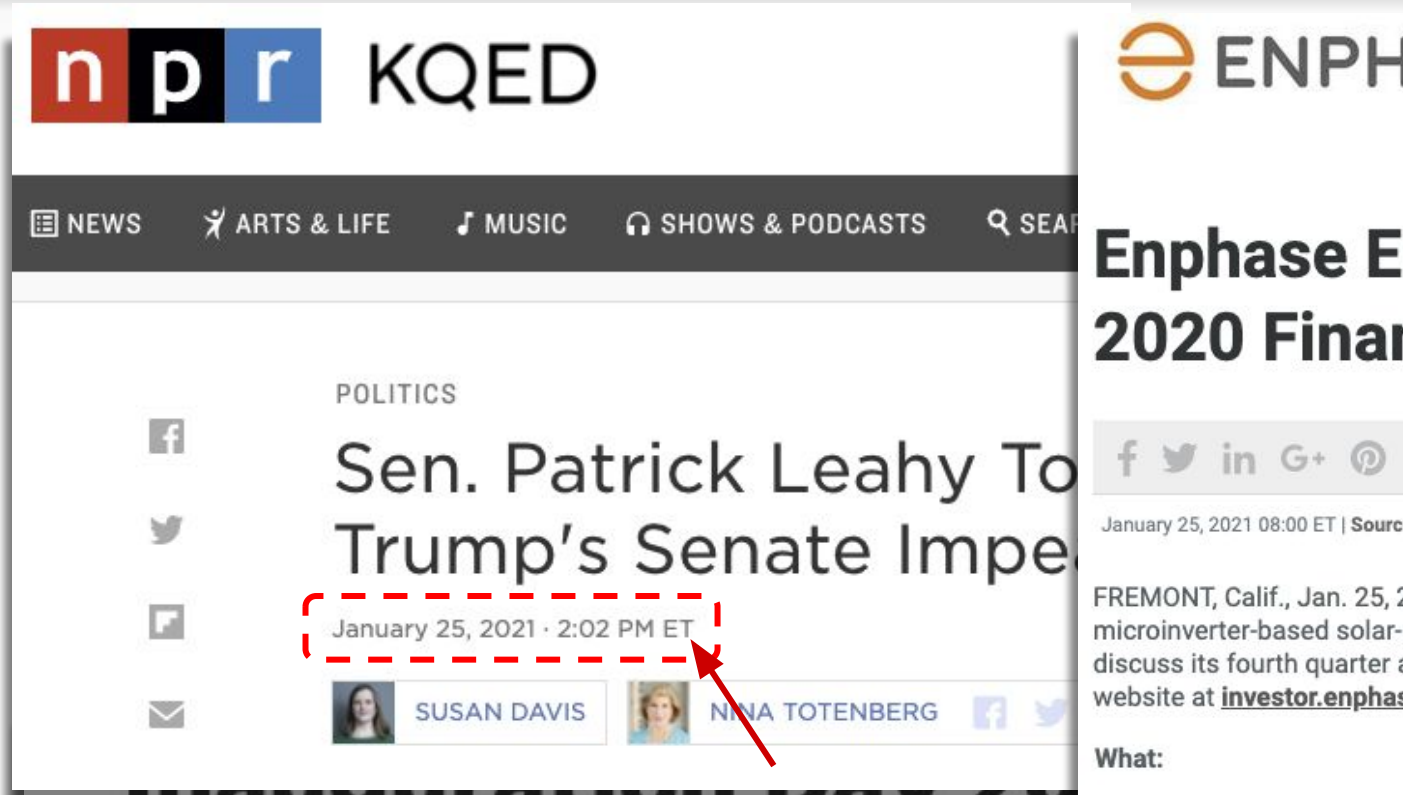
ECMA-402 Stage 2 & 3 Requirements

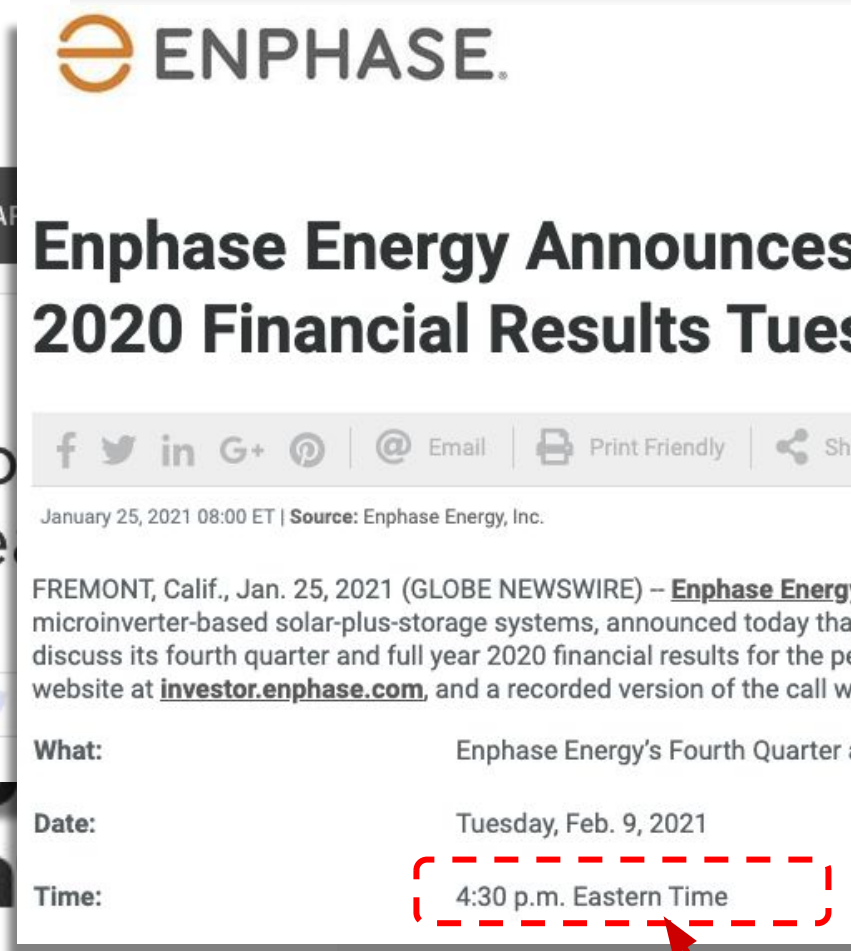
- Prior Art - ICU/ICU4J and many others
- Difficult to Implement in Userland
- Broad Appeal

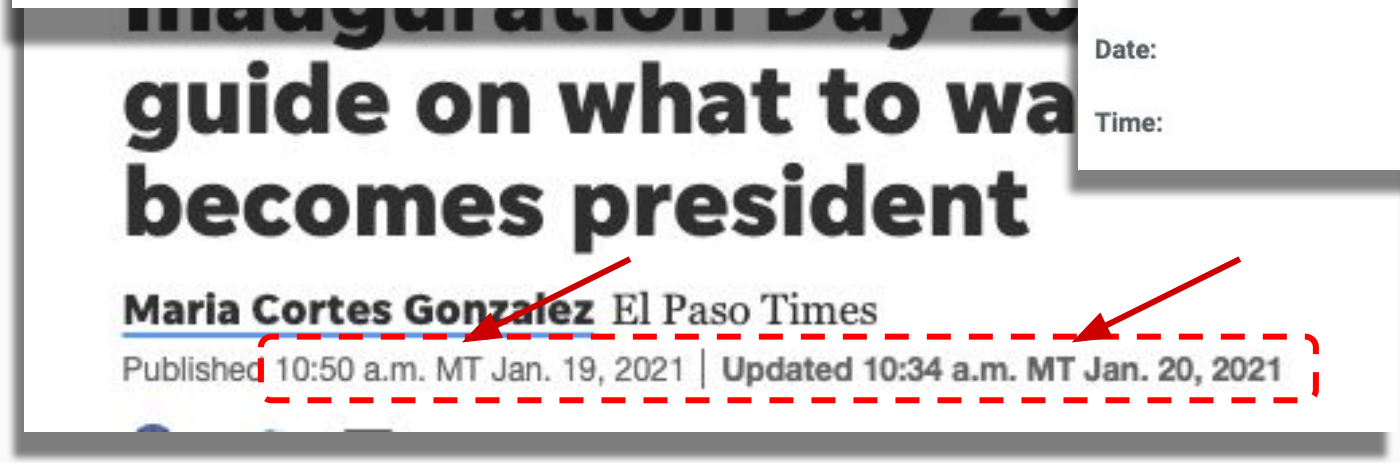
- Payload Mitigation

Examples of “Wall Time” Usage

Broad Appeal







Concerns about Data Size Increase

- **shortOffset & longOffset**:
  - # of items in 476 locales: 263
  - Total Bytes in UTF8: 1,826 bytes
  - Compressed Size: 392 bytes
- **shortGeneric**:
  - # of items in 476 locales: 332
  - Total Bytes in UTF8: 1,719 bytes
  - Compressed Size: 311 bytes
- **longGeneric**:
  - # of items in 476 locales: 10,047
  - Total Bytes in UTF8: 278,103 bytes
  - Compressed Size: 69,526 bytes

Note: Several other possible values of timeZoneName were removed from earlier proposal after we exam the size impact during Stage 0

History

- Advanced to Stage 1 in TC39 2021-01 Meeting
- ECMA402 2021-04 Monthly Meeting
  - Rname shortGMT/longGMT to shortOffset/longOffset
- Advanced to Stage 2 in TC39 2021-04 Meeting
- ECMA402 2021-05 Monthly Meeting
  - Rename shortWall/longWall to shortGeneric/longGeneric
  - Support to bring to TC39 for Stage 3 Advancement
-

1 DateTimeFormat Objects

1.1 Abstract Operations For DateTimeFormat Objects

Several DateTimeFormat algorithms use values from the following table, which provides internal slots, property names and allowable values for the components of date and time formats:

Table 1: Components of date and time formats

Internal Slot	Property	Values
[[Weekday]]	"weekday"	"narrow", "short", "long"
[[Era]]	"era"	"narrow", "short", "long"
[[Year]]	"year"	"2-digit", "numeric"
[[Month]]	"month"	"2-digit", "numeric", "narrow", "short", "long"
[[Day]]	"day"	"2-digit", "numeric"
[[Hour]]	"hour"	"2-digit", "numeric"
[[Minute]]	"minute"	"2-digit", "numeric"
[[Second]]	"second"	"2-digit", "numeric"
[[TimeZoneName]]	"timeZoneName"	"short", "long", "shortOffset", "longOffset", "shortGeneric", "longGeneric"

FormatDateTimePattern changes

1.1.2 FormatDateTimePattern (*dateTimeFormat*, *patternParts*, *x*, *rangeFormatOptions*)

The FormatDateTimePattern abstract operation is called with arguments *dateTimeFormat* (which must be an object initialized as a DateTimeFormat), *patternParts* (which is a list of Records as returned by PartitionPattern), *x* (which must be a Number value), and *rangeFormatOptions* (which is a range pattern Record as used in [[rangePattern]] or **undefined**), interprets *x* as a time value as specified in es2022, 20.4.1.1, and creates the corresponding parts according *pattern* and to the effective locale and the formatting options of *dateTimeFormat* and *rangeFormatOptions*. The following steps are taken:

1. Let *x* be TimeClip(*x*).
2. If *x* is **NaN**, throw a **RangeError** exception.
3. Let *locale* be *dateTimeFormat*.[[Locale]].
4. Let *nfOptions* be OrdinaryObjectCreate(**null**).
5. Perform ! CreateDataPropertyOrThrow(*nfOptions*, **"useGrouping"**, **false**).
6. Let *nf* be ? Construct(%NumberFormat%, « *locale*, *nfOptions* »).
7. Let *nf2Options* be OrdinaryObjectCreate(**null**).
8. Perform ! CreateDataPropertyOrThrow(*nf2Options*, **"minimumIntegerDigits"**, 2).
9. Perform ! CreateDataPropertyOrThrow(*nf2Options*, **"useGrouping"**, **false**).
10. Let *nf2* be ? Construct(%NumberFormat%, « *locale*, *nf2Options* »).
11. Let *fractionalSecondDigits* be *dateTimeFormat*.[[FractionalSecondDigits]].
12. If *fractionalSecondDigits* is not **undefined**, then
    a. Let *nf3Options* be OrdinaryObjectCreate(**null**).
    b. Perform ! CreateDataPropertyOrThrow(*nf3Options*, **"minimumIntegerDigits"**, *fractionalSecondDigits*).
    c. Perform ! CreateDataPropertyOrThrow(*nf3Options*, **"useGrouping"**, **false**).
    d. Let *nf3* be ? Construct(%NumberFormat%, « *locale*, *nf3Options* »).
13. Let *tm* be ToLocalTime(*x*, *dateTimeFormat*.[[Calendar]], *dateTimeFormat*.[[TimeZone]]).
14. Let *result* be a new empty List.
15. For each Record { [[Type]], [[Value]] } *patternPart* in *patternParts*, do
    a. Let *p* be *patternPart*.[[Type]].
    b. If *p* is **"literal"**, then
        i. Append a new Record { [[Type]]: **"literal"**, [[Value]]: *patternPart*.[[Value]] } as the last element of the list *result*.
    c. Else if *p* is equal to **"fractionalSecondDigits"**, then
        i. Let *v* be *tm*.[[Millisecond]].
        ii. Let *v* be floor(*v* × $10^{(fractionalSecondDigits - 3)}$).
        iii. Let *fv* be FormatNumeric(*nf3*, *v*).
        iv. Append a new Record { [[Type]]: **"fractionalSecond"**, [[Value]]: *fv* } as the last element of *result*.
    d. Else if *p* is equal to **"dayPeriod"**, then
        i. Let *f* be the value of *dateTimeFormat*'s internal slot whose name is the Internal Slot column of the matching row.
        ii. Let *fv* be a String value representing the day period of *tm* in the form given by *f*; the String value depends upon the implementation and the effective locale of *dateTimeFormat*.
        iii. Append a new Record { [[Type]]: *p*, [[Value]]: *fv* } as the last element of the list *result*.
    e. Else if *p* is equal to **"timeZoneName"**, then
        i. Let *f* be *dateTimeFormat*.[[TimeZoneName]].
        ii. Let *v* be *dateTimeFormat*.[[TimeZone]].
        iii. Let *fv* be a String value representing *v* in the form given by *f*; the String value depends upon the implementation and the effective locale of *dateTimeFormat*. The String value may also depend on the value of the [[InDST]] field of *tm* if *f* is **"short"**, **"long"**, **"shortOffset"**, or **"longOffset"**. If the implementation does not have a localized representation of *f*, then use the String value of *v* itself.
        iv. Append a new Record { [[Type]]: *p*, [[Value]]: *fv* } as the last element of the list *result*.
    f. Else if *p* matches a Property column of the row in Table 1, then
        i. If *rangeFormatOptions* is not **undefined**, let *f* be the value of *rangeFormatOptions*'s field whose name matches *p*.
        ii. Else, let *f* be the value of *dateTimeFormat*'s internal slot whose name is the Internal Slot column of the matching row.
        iii. Let *v* be the value of *tm*'s field whose name is the Internal Slot column of the matching row.
        iv. If *p* is **"year"** and *v* ≤ 0, let *v* be 1 - *v*.
        v. If *p* is **"month"**, increase *v* by 1.
        vi. If *p* is **"hour"** and *dateTimeFormat*.[[HourCycle]] is **"h11"** or **"h12"**, then
            1. Let *v* be *v* modulo 12.
            2. If *v* is 0 and *dateTimeFormat*.[[HourCycle]] is **"h12"**, let *v* be 12.
        vii. If *p* is **"hour"** and *dateTimeFormat*.[[HourCycle]] is **"h24"**, then
            1. If *v* is 0, let *v* be 24.
        viii. If *f* is **"numeric"**, then

FormatDateTimePattern changes

1.1.2 FormatDateTimePattern (*dateTimeFormat*, *patternParts*, *x*, *rangeFormatOptions*)

The FormatDateTimePattern abstract operation is called with arguments *dateTimeFormat* (which must be an object initialized as a DateTimeFormat), *patternParts* (which is a list of Records as returned by PartitionPattern), *x* (which must be a Number value), and *rangeFormatOptions* (which is a range pattern Record as used in [[rangePattern]] or **undefined**), interprets *x* as a time value as specified in es2022, 20.4.1.1, and creates the corresponding parts according *pattern* and to the effective locale and the formatting options of *dateTimeFormat* and *rangeFormatOptions*. The following steps are taken:

1. Let *x* be TimeClip(*x*).
2. If *x* is **NaN**, throw a **RangeError** exception.
3. Let *locale* be *dateTimeFormat*.[[Locale]].
4. Let *nfOptions* be OrdinaryObjectCreate(**null**).
5. Perform ! CreateDataPropertyOrThrow(*nfOptions*, **"useGrouping"**, **false**).

iii. Append a new Record { [[Type]]: *p*, [[Value]]: *fv* } as the last element of the list *result*.

e. Else if *p* is equal to **"timeZoneName"**, then

i. Let *f* be *dateTimeFormat*.[[TimeZoneName]].

ii. Let *v* be *dateTimeFormat*.[[TimeZone]].

iii. Let *fv* be a String value representing *v* in the form given by *f*; the String value depends upon the implementation and the effective locale of *dateTimeFormat*. The String value may also depend on the value of the [[InDST]] field of *tm* if *f* is **"short"**, **"long"**, **"shortOffset"**, or **"longOffset"**. If the implementation does not have a localized representation of *f*, then use the String value of *v* itself.

iv. Append a new Record { [[Type]]: *p*, [[Value]]: *fv* } as the last element of the list *result*.

f. Else if *p* matches a Property column of the row in Table 1, then

i. Append a new Record { [[Type]]: **"literal"**, [[Value]]: *patternPart*.[[Value]] } as the last element of the list *result*.

c. Else if *p* is equal to **"fractionalSecondDigits"**, then

i. Let *v* be *tm*.[[Millisecond]].

ii. Let *v* be floor(*v* × $10^{(fractionalSecondDigits - 3)}$).

iii. Let *fv* be FormatNumeric(*nf3*, *v*).

iv. Append a new Record { [[Type]]: **"fractionalSecond"**, [[Value]]: *fv* } as the last element of *result*.

d. Else if *p* is equal to **"dayPeriod"**, then

i. Let *f* be the value of *dateTimeFormat*'s internal slot whose name is the Internal Slot column of the matching row.

ii. Let *fv* be a String value representing the day period of *tm* in the form given by *f*; the String value depends upon the implementation and the effective locale of *dateTimeFormat*.

iii. Append a new Record { [[Type]]: *p*, [[Value]]: *fv* } as the last element of the list *result*.

e. Else if *p* is equal to **"timeZoneName"**, then

i. Let *f* be *dateTimeFormat*.[[TimeZoneName]].

ii. Let *v* be *dateTimeFormat*.[[TimeZone]].

iii. Let *fv* be a String value representing *v* in the form given by *f*; the String value depends upon the implementation and the effective locale of *dateTimeFormat*. The String value may also depend on the value of the [[InDST]] field of *tm* if *f* is **"short"**, **"long"**, **"shortOffset"**, or **"longOffset"**. If the implementation does not have a localized representation of *f*, then use the String value of *v* itself.

iv. Append a new Record { [[Type]]: *p*, [[Value]]: *fv* } as the last element of the list *result*.

f. Else if *p* matches a Property column of the row in Table 1, then

i. If *rangeFormatOptions* is not **undefined**, let *f* be the value of *rangeFormatOptions*'s field whose name matches *p*.

ii. Else, let *f* be the value of *dateTimeFormat*'s internal slot whose name is the Internal Slot column of the matching row.

iii. Let *v* be the value of *tm*'s field whose name is the Internal Slot column of the matching row.

iv. If *p* is **"year"** and *v* ≤ 0, let *v* be 1 - *v*.

v. If *p* is **"month"**, increase *v* by 1.

vi. If *p* is **"hour"** and *dateTimeFormat*.[[HourCycle]] is **"h11"** or **"h12"**, then

1. Let *v* be *v* modulo 12.
2. If *v* is 0 and *dateTimeFormat*.[[HourCycle]] is **"h12"**, let *v* be 12.

vii. If *p* is **"hour"** and *dateTimeFormat*.[[HourCycle]] is **"h24"**, then

1. If *v* is 0, let *v* be 24.

viii. If *f* is **"numeric"**, then

BasicFormatMatches Changes

1.1.1 BasicFormatMatcher (*options*, *formats*)

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

1. Let *removalPenalty* be 120.
2. Let *additionPenalty* be 20.
3. Let *longLessPenalty* be 8.
4. Let *longMorePenalty* be 6.
5. Let *shortLessPenalty* be 6.
6. Let *shortMorePenalty* be 3.
7. Let *offsetPenalty* be 1.
8. Let *bestScore* be -**Infinity**.
9. Let *bestFormat* be **undefined**.
10. Assert: Type(*formats*) is List.
11. For each element *format* of *formats*, do
    a. Let *score* be 0.
    b. For each property name *property* shown in Table 1, do
        i. If *options* has a field [[<*property*>]], let *optionsProp* be *options*.[[<*property*>]]; else let *optionsProp* be **undefined**.
        ii. If *format* has a field [[<*property*>]], let *formatProp* be *format*.[[<*property*>]]; else let *formatProp* be **undefined**.
        iii. If *optionsProp* is **undefined** and *formatProp* is not **undefined**, decrease *score* by *additionPenalty*.
        iv. Else if *optionsProp* is not **undefined** and *formatProp* is **undefined**, decrease *score* by *removalPenalty*.
        v. Else if *property* is **"timeZoneName"**, then
            1. If *optionsProp* is **"short"** or **"shortGeneric"**, then
                a. If *formatProp* is **"shortOffset"**, decrease *score* by *offsetPenalty*.
                b. Else if *formatProp* is **"longOffset"**, decrease *score* by (*offsetPenalty* + *shortMorePenalty*).
                c. Else if *optionsProp* is **"short"** and *formatProp* is **"long"**, decrease *score* by *shortMorePenalty*.
                d. Else if *optionsProp* is **"shortGeneric"** and *formatProp* is **"longGeneric"**, decrease *score* by *shortMorePenalty*.
                e. Else if *optionsProp* ≠ *formatProp*, decrease *score* by *removalPenalty*.
            2. Else if *optionsProp* is **"shortOffset"** and *formatProp* is **"longOffset"**, decrease *score* by *shortMorePenalty*.
            3. Else if *optionsProp* is **"long"** or **"longGeneric"**, then
                a. If *formatProp* is **"longOffset"**, decrease *score* by *offsetPenalty*.
                b. Else if *formatProp* is **"shortOffset"**, decrease *score* by (*offsetPenalty* + *longLessPenalty*).
                c. Else if *optionsProp* is **"long"** and *formatProp* is **"short"**, decrease *score* by *longLessPenalty*.
                d. Else if *optionsProp* is **"longGeneric"** and *formatProp* is **"shortGeneric"**, decrease *score* by *longLessPenalty*.
                e. Else if *optionsProp* ≠ *formatProp*, decrease *score* by *removalPenalty*.
            4. Else if *optionsProp* is **"longOffset"** and *formatProp* is **"shortOffset"**, decrease *score* by *longLessPenalty*.
            5. Else if *optionsProp* ≠ *formatProp*, decrease *score* by *removalPenalty*.
        vi. Else if *optionsProp* ≠ *formatProp*, then
            1. If *property* is **"fractionalSecondDigits"**, then
                a. Let *values* be « $\mathbf{1}_{\mathbb{F}}$, $\mathbf{2}_{\mathbb{F}}$, $\mathbf{3}_{\mathbb{F}}$ ».
            2. Else,
                a. Let *values* be « **"2-digit"**, **"numeric"**, **"narrow"**, **"short"**, **"long"** ».
            3. Let *optionsPropIndex* be the index of *optionsProp* within *values*.
            4. Let *formatPropIndex* be the index of *formatProp* within *values*.
            5. Let *delta* be max(min(*formatPropIndex* - *optionsPropIndex*, 2), -2).
            6. If *delta* = 2, decrease *score* by *longMorePenalty*.
            7. Else if *delta* = 1, decrease *score* by *shortMorePenalty*.
            8. Else if *delta* = -1, decrease *score* by *shortLessPenalty*.
            9. Else if *delta* = -2, decrease *score* by *longLessPenalty*.
    c. If *score* > *bestScore*, then
        i. Let *bestScore* be *score*.
        ii. Let *bestFormat* be *format*.
12. Return *bestFormat*.

iv. Else if *optionsProp* is not **undefined** and *formatProp* is **undefined**, decrease *score* by *removalPenalty*.
v. Else if *property* is **"timeZoneName"**, then
1. If *optionsProp* is **"short"** or **"shortGeneric"**, then
a. If *formatProp* is **"shortOffset"**, decrease *score* by *offsetPenalty*.
b. Else if *formatProp* is **"longOffset"**, decrease *score* by (*offsetPenalty* + *shortMorePenalty*).
c. Else if *optionsProp* is **"short"** and *formatProp* is **"long"**, decrease *score* by *shortMorePenalty*.
d. Else if *optionsProp* is **"shortGeneric"** and *formatProp* is **"longGeneric"**, decrease *score* by *shortMorePenalty*.
e. Else if *optionsProp* ≠ *formatProp*, decrease *score* by *removalPenalty*.
2. Else if *optionsProp* is **"shortOffset"** and *formatProp* is **"longOffset"**, decrease *score* by *shortMorePenalty*.
3. Else if *optionsProp* is **"long"** or **"longGeneric"**, then
a. If *formatProp* is **"longOffset"**, decrease *score* by *offsetPenalty*.
b. Else if *formatProp* is **"shortOffset"**, decrease *score* by (*offsetPenalty* + *longLessPenalty*).
c. Else if *optionsProp* is **"long"** and *formatProp* is **"short"**, decrease *score* by *longLessPenalty*.
d. Else if *optionsProp* is **"longGeneric"** and *formatProp* is **"shortGeneric"**, decrease *score* by *longLessPenalty*.
e. Else if *optionsProp* ≠ *formatProp*, decrease *score* by *removalPenalty*.
4. Else if *optionsProp* is **"longOffset"** and *formatProp* is **"shortOffset"**, decrease *score* by *longLessPenalty*.
5. Else if *optionsProp* ≠ *formatProp*, decrease *score* by *removalPenalty*.
vi. Else if *optionsProp* ≠ *formatProp*, then

c. Else if *optionsProp* is **"long"** and *formatProp* is **"short"**, decrease *score* by *longLessPenalty*.
d. Else if *optionsProp* is **"longGeneric"** and *formatProp* is **"shortGeneric"**, decrease *score* by *longLessPenalty*.
e. Else if *optionsProp* ≠ *formatProp*, decrease *score* by *removalPenalty*.
4. Else if *optionsProp* is **"longOffset"** and *formatProp* is **"shortOffset"**, decrease *score* by *longLessPenalty*.
5. Else if *optionsProp* ≠ *formatProp*, decrease *score* by *removalPenalty*.
vi. Else if *optionsProp* ≠ *formatProp*, then

1. If *property* is **"fractionalSecondDigits"**, then
a. Let *values* be « **1**$_F$, **2**$_F$, **3**$_F$ ».
2. Else,
a. Let *values* be « **"2-digit"**, **"numeric"**, **"narrow"**, **"short"**, **"long"** ».
3. Let *optionsPropIndex* be the index of *optionsProp* within *values*.
4. Let *formatPropIndex* be the index of *formatProp* within *values*.
5. Let *delta* be max(min(*formatPropIndex* - *optionsPropIndex*, 2), -2).
6. If *delta* = 2, decrease *score* by *longMorePenalty*.
7. Else if *delta* = 1, decrease *score* by *shortMorePenalty*.
8. Else if *delta* = -1, decrease *score* by *shortLessPenalty*.
9. Else if *delta* = -2, decrease *score* by *longLessPenalty*.
c. If *score* > *bestScore*, then
i. Let *bestScore* be *score*.
ii. Let *bestFormat* be *format*.
12. Return *bestFormat*.

References

- V8 Prototype: https://chromium-review.googlesource.com/c/v8/v8/+/2757899
- Mozilla
- JSC
- Reviewers:
  - Philip Chimento @ptomato
  - Rick Button @rickbutton
- Editors:

Entrance Criteria / Acceptance Signifies For Stage 3

Entrance Criteria:

- Complete spec text **DONE**
- Designated reviewers have signed off on the current spec text
- 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/proposal-intl-extend-timezonename/
- 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/FrankYFTang/proposal-intl-extend-timezonename/