Exploring Temporal

Steve Harrison
https://www.linkedin.com/in/stevenanthonyharrison/

2 December 2022

Temporal is a new ECMAScript spec that provides a native modern date/time API, including timezone support.

What’s wrong with Date?

https://maggiepint.com/2017/04/09/fixing-javascript-date-getting-started/

- No support for time zones other than the user’s local time and UTC
- Parser behaviour so unreliable it is unusable
- Date object is mutable
- DST behaviour is unpredictable
- Computation APIs are unwieldy
- No support for non-Gregorian calendars

Enter Temporal

```
const birthday = Temporal.PlainMonthDay.from('12-15');

const birthdayIn2030 = birthday.toPlainDate({ year: 2030 });

birthdayIn2030.dayOfWeek; // => 7
```

When can you use it?

Temporal is in Stage 3, a final draft before the spec gets approved.

ECMAScript Spec Stages

Can you use it now?

Yes, but polyfills are not prod-ready yet.

@js-temporal/polyfill

https://github.com/js-temporal/temporal-polyfill

54.1kB minified + gzipped

```
import { Temporal, Intl, toTemporalInstant } from '@js-temporal/polyfill';
Date.prototype.toTemporalInstant = toTemporalInstant;
```

temporal-polyfill

https://github.com/fullcalendar/temporal

17.4kB minified + gzipped

```
import { Temporal } from 'temporal-polyfill'

console.log(Temporal.Now.zonedDateTimeISO().toString())
```

Try Temporal in DevTools now

https://tc39.es/proposal-temporal/docs/

ISO 8601 formatting

Created in 1988; last updated in 2022.

Date-times `'2022-11-26T13:38:52.249Z'`

Year-Month-Day

Hours:Minutes:Seconds.Milliseconds

Z = UTC time

Durations `'P3DT4H59M'`

3 days, 4 hours and 59 minutes

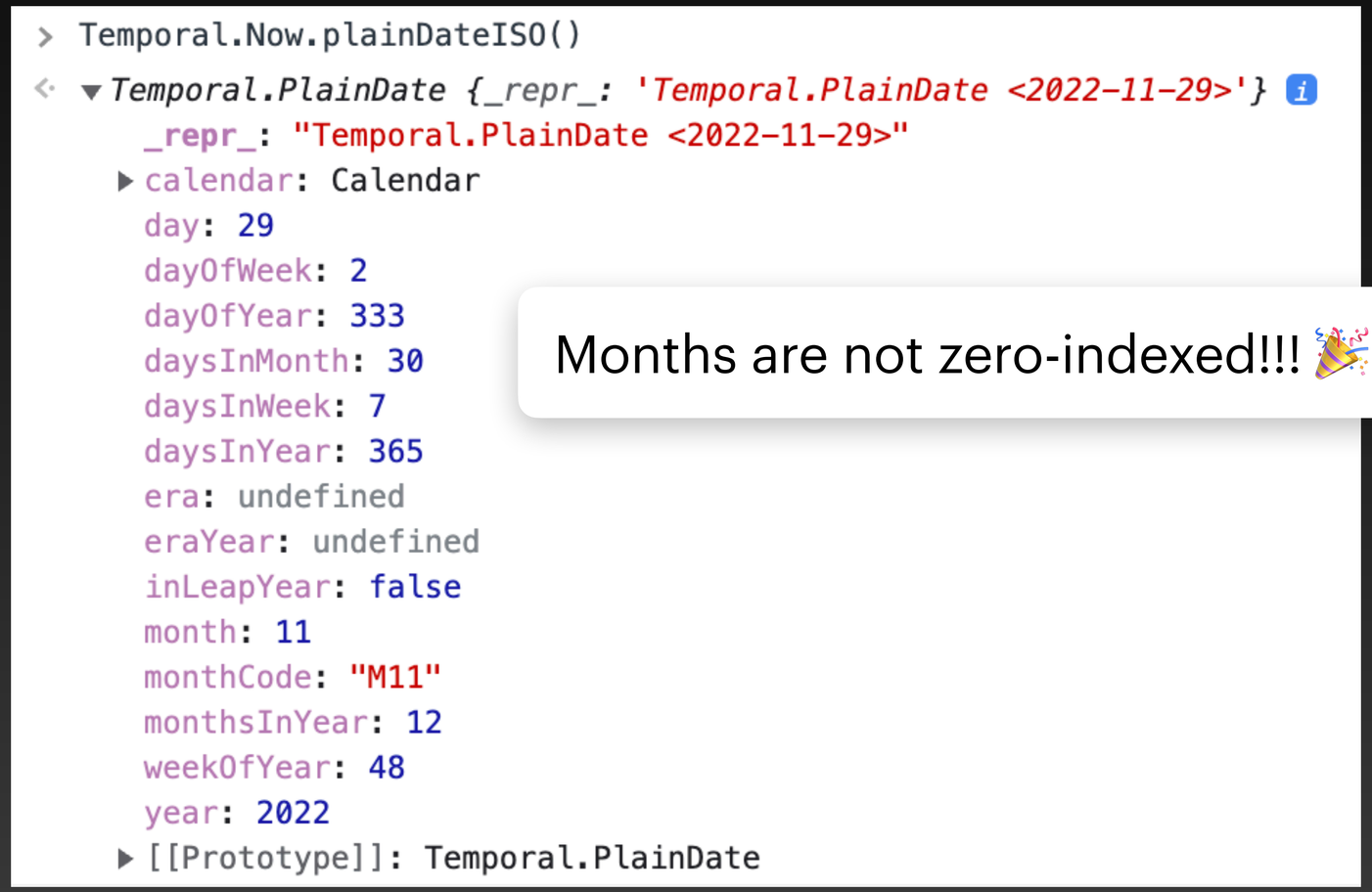
Basics

Temporal.Now

```
// Current date in ISO 8601 format
Temporal.Now.plainDateISO().toString();
'2022-11-27'

// Current date and time in ISO 8601 format
Temporal.Now.plainDateTimeISO().toString();
'2022-11-27T03:04:33.236673232'
```

Basics

Temporal.Now



Basics

Temporal.Now

```
const timeStamp = Temporal.Now.instant();

// Timestamp in Milliseconds
timeStamp.epochMilliseconds;

// Convenience methods for common units!
timeStamp.epochSeconds;
timeStamp.epochNanoseconds;
timeStamp.epochMicroseconds;
```

Basics

Temporal.PlainDate

```
new Temporal.PlainDate(2006, 8, 24)

Temporal.PlainDate.from('2006-08-24');

Temporal.PlainDate.from({ year: 2006, month: 8, day: 24 });
```

Basics

Temporal.PlainDateTime

```
new Temporal.PlainDateTime(2006, 8, 24, 10, 23, 30)

Temporal.PlainDateTime.from('2006-08-24T10:23:30');

Temporal.PlainDateTime.from({
  year: 2006,
  month: 8,
  day: 24,
  hour: 10,
  minute: 23,
  second: 30,
  millisecond: 0,
  microsecond: 0,
  nanosecond: 0
}); // => 2006-08-24T10:23:30.0000000
```

Basics

Temporal.PlainTime

```
new Temporal.PlainTime(13, 37)  // => 13:37:00

Temporal.PlainTime.from('03:24:30');

Temporal.PlainTime.from({
  hour: 19,
  minute: 39,
  second: 9,
  millisecond: 68,
  microsecond: 346,
  nanosecond: 205
}); // => 19:39:09.068346205
```

Basics

Temporal.ZonedDateTime

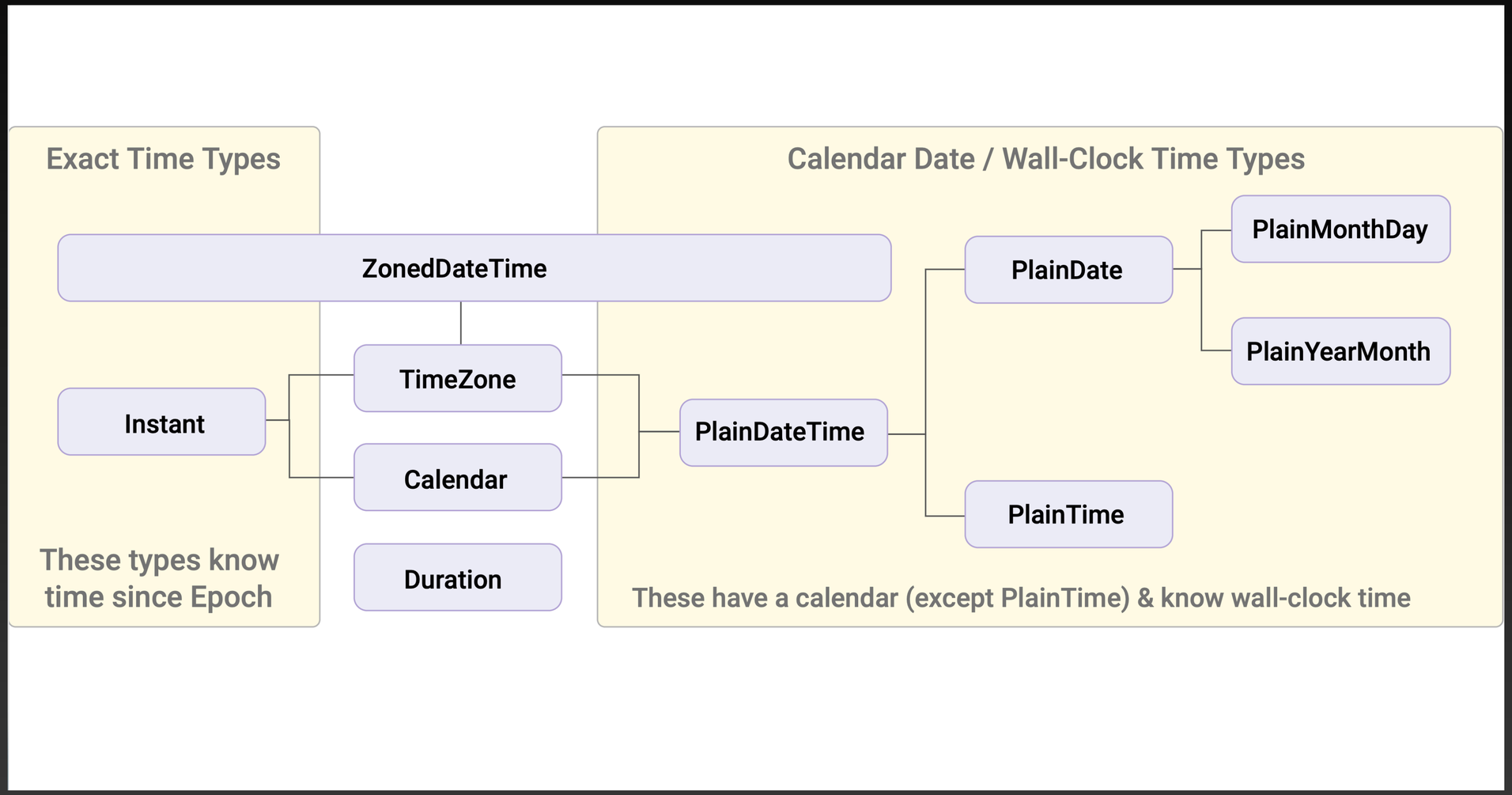
```
new Temporal.ZonedDateTime(818299470000000000n, 'Africa/Cairo');

Temporal.ZonedDateTime.from('1995-12-07T03:24:30+02:00[Africa/Cairo]');

Temporal.ZonedDateTime.from({
    timeZone: 'Africa/Cairo',
    year: 1995,
    month: 12,
    day: 7,
    hour: 3,
    minute: 24,
    second: 30,
    millisecond: 0,
    microsecond: 0,
    nanosecond: 0
});  // => 1995-12-07T03:24:30.0000000+02:00[Africa/Cairo]
```

Basics

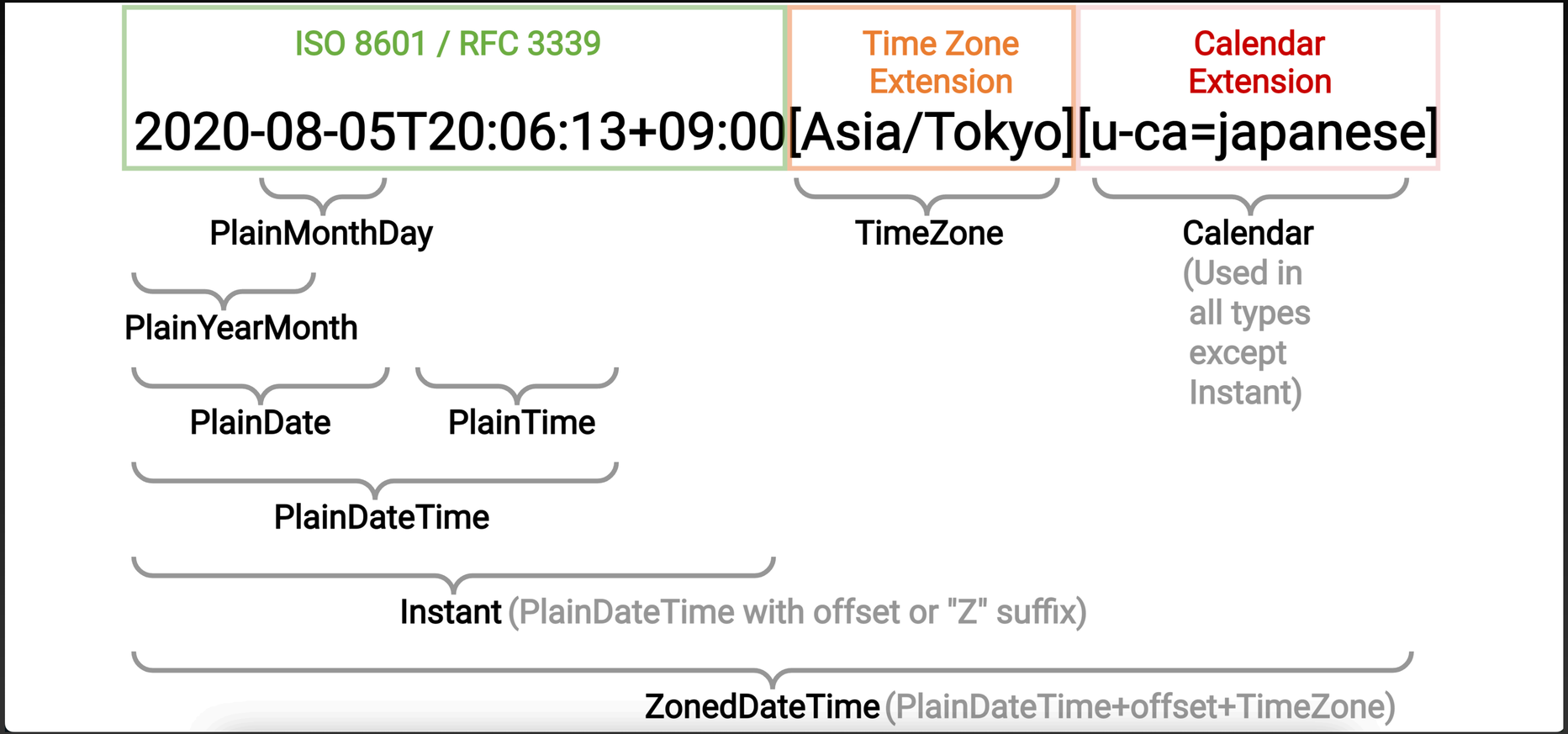
Temporal.Duration

```
new Temporal.Duration(0, 0, 0, 40);

Temporal.Duration.from('P40D');

Temporal.Duration.from({
  years: 0,
  months: 0,
  days: 40,
  hours: 0,
  minutes: 0
});
```

Types Summary



Temporal <> ISO 8601



So what can we do with these types?

Combine plain types

```
const date = Temporal.PlainDate.from('2020-05-14');
const noonOnDate =
date.toPlainDateTime(Temporal.PlainTime.from({ hour: 12 }));

'2020-05-14T12:00:00'
```

Combine plain types

```
const birthday = Temporal.PlainMonthDay.from('12-15');

const birthdayIn2030 = birthday.toPlainDate({ year: 2030 });

birthdayIn2030.dayOfWeek; // => 7
```

Operations

```
Temporal.PlainDateTime.from('1995-12-07')
  .add({ years: 20, months: 4, nanoseconds: 500 })
  .round({ smallestUnit: 'hour' })
  .subtract({ hours: 10 })
// => 2016-04-06T14:00:00
```

Comparisons

```
const dt1 = Temporal.PlainDateTime.from('2019-01-31T15:30');
const dt2 = Temporal.PlainDateTime.from('2019-01-31T15:30');

dt1.equals(dt2); // => true


dt1.until('2022-01-01');
   // => P1065DT8H30M
```

Example: Flight Times

```
const departure =
Temporal.ZonedDateTime.from('2020-03-08T11:55:00+08:00[Asia/
Hong_Kong]');
const flightTime = Temporal.Duration.from({ minutes: 775 });
const arrival = departure.add(flightTime).withTimeZone('America/
Los_Angeles');

'2020-03-08T09:50:00-07:00[America/Los_Angeles]'
```

Example: Flight Times

```
const calculatedFlightTime = departure.until(arrival);

'PT12H55M'
```

Example: Time in LA right now

```
Temporal.Now.zonedDateTimeISO()
  .withTimeZone('America/Los_Angeles')
```

Formatting & parsing dates

To/from ISO 8601 string

```
Temporal.ZonedDateTime.from('2022-02-28T11:06:00.092121729+08
:00[Asia/Shanghai][u-ca=chinese]').toString();
  // => "2022-02-28T11:06:00.092121729+08:00[Asia/Shanghai]
[u-ca=chinese]"

Temporal.PlainDate.from('2022-02-28').toString();
date.toString();
  // => "2022-02-28"

Temporal.Duration.from('P1DT12H30M').toString();
  // => "P1DT12H30M"
```

Locale-aware formatting

Intl.DateTimeFormat

```
Temporal.ZonedDateTime.from('2022-02-28T11:06:00.092121729+08:00[Asia/Shanghai][u-ca=chinese]').toLocaleString('zh-CN', { calendar: 'chinese' });
'正月28日 GMT+8 11:06:00'

new Intl.DateTimeFormat('en-AU').format(date);
'27/11/2022'

new Intl.DateTimeFormat('en-US').format(date);
'11/27/2022'

new Intl.DateTimeFormat('en-AU', {
  day: 'numeric',
  weekday: 'long',
  month: 'long',
  year: '2-digit'
}).format(date);
'Sunday, 27 November 22'
```

https://tc39.es/ecma402/#sec-datetimeformat-abstracts

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"
[[DayPeriod]]	"dayPeriod"	"narrow", "short", "long"
[[Hour]]	"hour"	"2-digit", "numeric"
[[Minute]]	"minute"	"2-digit", "numeric"
[[Second]]	"second"	"2-digit", "numeric"
[[FractionalSecondDigits]]	"fractionalSecondDigits"	1𝔽, 2𝔽, 3𝔽
[[TimeZoneName]]	"timeZoneName"	"short", "long", "shortOffset", "longOffset", "shortGeneric", "longGeneric"

Locale-aware formatting

```
new Intl.DateTimeFormat('en-AU')
  .formatToParts(Temporal.Now.instant())
```

```
(13) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
  0: {type: 'day', value: '01'}
  1: {type: 'literal', value: '/'}
  2: {type: 'month', value: '12'}
  3: {type: 'literal', value: '/'}
  4: {type: 'year', value: '2022'}
  5: {type: 'literal', value: ', '}
  6: {type: 'hour', value: '6'}
  7: {type: 'literal', value: ':'}
  8: {type: 'minute', value: '46'}
  9: {type: 'literal', value: ':'}
  10: {type: 'second', value: '57'}
  11: {type: 'literal', value: ' '}
  12: {type: 'dayPeriod', value: 'pm'}
  length: 13
  [[Prototype]]: Array(0)
```

Locale-aware formatting

Intl.DurationFormat (Stage 3)

```
Temporal.Duration.from('PT1H46M40S')
  .toLocaleString('fr-FR', { style: 'long' });
// => '1 heure, 46 minutes et 40 secondes'
```

Relative time formatting

```
new Intl.RelativeTimeFormat('en', { numeric: 'auto' }).format(0, 'month')
'this month'

new Intl.RelativeTimeFormat('en', { style: 'short' }).format(1, 'month')
'in 1 mo.'

new Intl.RelativeTimeFormat('es').format(-2.14, 'day')
'hace 2,14 días'
```

To/from from custom formats

Temporal does not support this.

Options:

Write your own code
(e.g. RegExp for parsing)

Use an existing library like date-fns
(e.g. convert to legacy Date)

Converting to/from regular dates

```
new Date(Temporal.ZonedDateTime.from('2020-01-01T00:00:01.001[Asia/
Tokyo]').epochMilliseconds)


new Date('2022-02-22T14:23:21.911Z').toTemporalInstant();
```

date-fns relative time formatting

Distance to the base date	Result
Previous 6 days	last Sunday at 04:30 AM
Last day	yesterday at 04:30 AM
Same day	today at 04:30 AM
Next day	tomorrow at 04:30 AM
Next 6 days	Sunday at 04:30 AM
Other	12/31/2017

Aside: Intl

ECMAScript's Internationalisation API

Intl.DateTimeFormat
Intl.RelativeTimeFormat
Intl.DurationFormat (coming soon: in Stage 3)
Intl.Collator
Intl.ListFormat
Intl.NumberFormat
Intl.PluralRules
Intl.Segmenter

Aside: Intl

ECMAScript's Internationalisation API

```
new Intl.NumberFormat('de-DE', { style: 'currency',
currency: 'EUR' }).format(number);
// → "123.456,79 €"

new Intl.PluralRules('ar-EG').select(18);
// → 'many'
```

You can also convert timezones with Intl

```
Intl.DateTimeFormat('en-AU', { timeZone:
'America/Los_Angeles', weekday: 'short',
hour: 'numeric', minute: 'numeric', second:
'numeric' }).format(new Date())

'Sat 7:43:08 am'
```

...if you just need to format the date.

Other things you can do in Temporal

More complex

Overflow behaviour

```
Temporal.ZonedDateTime.from({
  timeZone: 'Europe/Paris',
  year: 2001,
  month: 13,
  day: 1
}, { overflow: 'constrain' })
// => 2001-12-01T00:00:00+01:00[Europe/Paris]

Temporal.ZonedDateTime.from({
  timeZone: 'Europe/Paris',
  year: 2001,
  month: 13,
  day: 1
}, { overflow: 'reject' })
// => throws RangeError
```

More info

https://github.com/steveharrison/steveharrison.github.io/blob/master/temporal/index.md