Working with dates in Javascript: The Temporal proposal

Note:

The Temporal code is not ready for production, it is still possible (but not likely) that it'll change in incompatible ways before the final version is added to the ECMAScript specification, and because of that, it shouldn't be used in production until it reaches stage 4 in the TC39 process. However, you can test how it works now and decide if you want to use it in future projects.

One of the most infuriating things to do in Javascript is date manipulation. The default Javascript Date object is hard to work with. This has lead to a proliferation of third-party libraries like Moment.js, Luxon, date-fns and even the Javascript Intl API for some types of data manipulation.

For this post we'll concentrate in the Temporal Proposal, currently at stage 3 in the TC39 process. The idea is that `Temporal` will sit alongside the `Date` object and provide better date/time handling functionality without breaking existing code that uses the `Date` object.

According to the Temporal documentation:

> Date has been a long-standing pain point in ECMAScript. This is a proposal for Temporal, a global Object that acts as a top-level namespace (like Math), that brings a modern date/time API to the ECMAScript language. For a detailed look at some of the problems with Date, and the motivations for Temporal, see: Fixing JavaScript Date.
>
> Temporal fixes these problems by:
>
> - Providing easy-to-use APIs for date and time computations

> - First-class support for all time zones, including DST-safe arithmetic
> - Dealing only with objects representing fixed dates and times
> - Parsing a strictly specified string format
> - Supporting non-Gregorian calendars

SO with all the information out of the way, let's look at some code. I wrote these demos using Node; so the first step, as usual, is to require the code or import it if you're working with ESM Imports.

```
const { Temporal } = require('proposal-temporal');
```

Date and Date/Time objects

The first step in the research is to get the date right now. Note that this will provide the ISO calendar-based dates and date/time combinations using methods in the `Temporal.now` object.

If you run your own tests with this code your results will be different because the timing of now will be different than mine :)

```
const date = Temporal.now.plainDateISO();
// returns the date in ISO 8601 date format
date.toString();

const date2 = Temporal.now.plainDateTimeISO();
// returns the time in ISO 8601 format
date2.toString();
```

We can also get the time based on our time zone by using `zonedDateTimeISO` as the calendar instead of plain ISO as in the previous examples.

```
const date2 = Temporal.now.zonedDateTimeISO();
date2.toString();
```

Using a time zone as a parameter we can specify different times for our object. We

cam use individual variables to contain the different date and time strings.

```
const dateWithTimezone4 = Temporal.now.plainDateTimeISO('America/Los_Ange
console.log(`Local Time: ${dateWithTimezone4.toString()}`);

const dateWithTimezone = Temporal.now.plainDateTimeISO('America/Chicago')
console.log('America/Chicago'`Chicago Time: ${dateWithTimezone.toString()
console.log(`New York Time: $'America/New_York'`New York Time: ${dateWithT
console.log(`Tokyo Time: ${dateWithTimezone3.to'Asia/Tokyo'`Tokyo Time: $
console.log(`Santiago, Chile Time: ${dateWithTimezone5.toSt'Chile/Continer
```

Or we can use a function to output the results we want. The function uses a for of ot iterate over a list of iterables from our `citiesToCheck` generated with object.entries

We then log to console the name of the city and the result of running `Temporal.now.zonedDateTimeISO` for the corresponding time zone.

```
const citiesToCheck = {
  'Los Angeles': 'America/Los_Angeles',
  'Chicago': 'Am'America/Los_Angeles'w York': 'America/New_York',
  'Tokyo': 'Asia/Tokyo',
  'Santiago, Chile': 'Chile/Continental',
};

for (co': '[name, timeZone] of Object.entries(citiesToCheck)) {
  console.log(`${name}: ${Temporal.now.zonedDateTimeISO(timeZone)}`);
}
```

We can use something like this to keep a world clock-like application. Every time you reload the page the times for all your cities and time zones will be updated or you could run it so it updates once a second.

Building dates and date/times from custom parts

So far we've worked with dates based on the current time. But we can also build

specific dates.

We can build the `Temporal.PlainDate` objects from an object containing key/value pairs for each element (year, month and day) or from an ISO 8601 date.

```
const instant = Temporal.PlainDate.from({ year: 1974, month: 3, day: 14 }
instant.toString();
// Result = '1974-09-25'

const instant2 = Temporal.PlainDate.from(Temporal.now.plainDateTimeISO())
instant2.toString();
// Result = 2021-05-07
```

We can further refine it by breaking out date and time elements.

```
const instant3 = Temporal.PlainDate.from(
  Temporal.now.plainDateTimeISO('Chile/Continental')
);

const instant3a = Temporal.PlainTime.from(
  Temporal.now.plainDateTimeISO('Chile/Continental')
);

instant3.toString();
'Chile/Continental'// Result = '2021-05-07'();
// Result = '11:49:24.125564123'
'11:49:24.125564123'
```

The last item I found interesting when playing with dates is that we can find out what day of the week a given date falls in that specific date.

We create a `Temporal.PlainDate` object with the date we want to work with. Then we creaate a 2-dimensional array with the days of the week in one array and the specific day we want expressed as `instant.dayOfWeek -1`. This will return the correct day of the week for our instant date.

```
// BUILDING TEMPORAL DATES FROM STRINGS AND OBJECTS
```

```
const instant = Temporal.PlainDate.from({ year: 1974, month: 3, day: 14 }
instant.toString();
// Result = '1974-03-14'

['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sun
  instant.dayOfWeek - 1
];
'Monday'// Result = 'Thursday'
```

Time Stamps

Another area of interest is timestamp generation. We can use these are hashes for files or to measure time since the Unix Epoch. This is particularly useful when working with code.

To generate the time stamp, use the `Temporal.now.instant()`, which will be the end time of the comparison, and then choose the precision you want to work with either milliseconds (`epochMilliseconds`) or seconds (`epochSeconds`). This will measure how many of your chose units have ocurred between the Unix Epoch (1970-01-01) and the value of `now.instant`

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

// Timestamp in Milliseconds
timeStamp.epochMilliseconds;
// Timestamp in Seconds
timeStamp.epochSeconds;
```

Integrating with `Intl.DateTimeFormat`

There is currently work in progress to integrate the Temporal proposal with ECMA 402's Intl.DateTimeFormat().

The current discussion is happening in a Github issue that I think is worth

keeping track of. For now, there's no integrated API for developers to experiment with.

Notes and closing

There is plenty more you can do with the Temporal Proposal. If you're interested in tracking its evolution to stage 4 and publication as part of ECMAScript, the best place to do it is the Github Repository, particularly the cookbook, the issue tracker and the specification itself if you can understand the way specifications are written

Links and Resources

- Dates and Times in Javascript — Igalia
- ISO 8601 — Wikipedia
- Javascript Internationalization API — MDN
- Intl.DateTimeFormat — MDN
- Temporal Propopsal — TC39
- Temporal Proposal Cookbook — TC39