Record & Tuple

for Stage 2

Robin Ricard & Rick Button
Bloomberg

Advisor: Daniel Ehrenberg
Igalia

A recap of last update

Equality Semantics

-0 === +0

Going with intermediary semantics for ==/===:

- The one used for Map keys/Set values comparison.
- A unification of +0 and -0.

Object.is compares to see if they are identical:
In that case +0 and -0 are different.

```
const s = new Set();
s.add(#[+0]);
s.has(#[-0]) === true;
s.add(#[NaN]);
s.has(#[NaN]) === true;

#[-0]  === #[+0]  // => true
#[NaN] === #[NaN] // => true

#[-0]  ==  #[+0]  // => true
#[NaN] ==  #[NaN] // => true

Object.is(#[-0],  #[+0])  === false
Object.is(#[NaN], #[NaN]) === true
```

Avoids “black-holing” structures if a NaN appears in any of them.

```
const measure = 42;

const computed = #{
    name: "Computed Measurement",
    value: pureComputeValue(measure),
};

assert(computed === computed);
// What if pureComputeValue returns NaN?
```

Avoids failing comparisons when the structure potentially has a -0 in it.

```
function isAtOrigin(c) {
    return c === #{x: 0, y: 0};
}

const coord = #{x: 0, y: 3};
const coord2 = #{
    x: coord.x * -4,
    y: coord.y - 3,
};

assert(isAtOrigin(coord));
// We expect this one to be true!
```

In general, we’re trying to make comparing records and tuples “trustworthy” for users and avoiding those subtle equality breakages helps in establishing this.

Still open for discussion!

- This is the equality we have in the Stage 2 spec
- This can change before we get to Stage 3
- The right decision will appear through more research:
    - Experimental implementations
    - Interviewing and surveying developers
    - Performance implications in implementations

State of the proposal

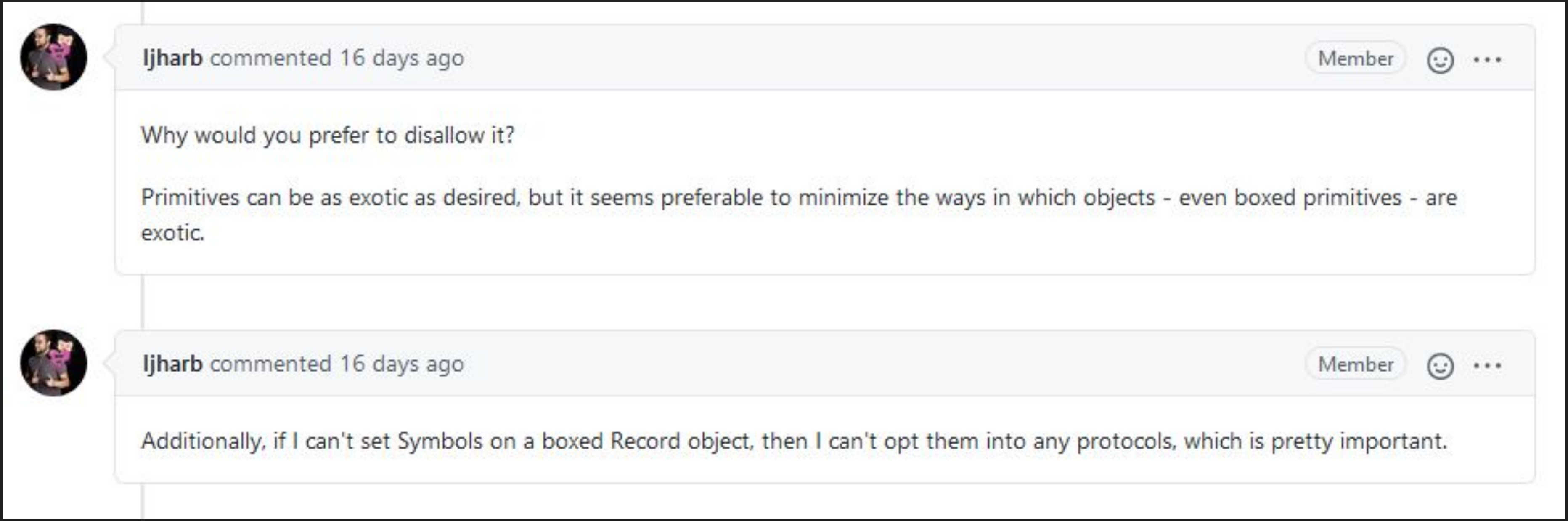
Ongoing Stage 3 Discussions

- Definitive equality semantics (#65)
- Names and exact semantics of Tuple.prototype methods (e.g. pushed) (#121)
- Syntax still open with a possibility to move to {| } and [|] (#10)
- Should the wrapper objects be extensible (#137)
- Should Record have a null prototype? (#71)
- Exact ToString behavior (#136)

Desire: “guarantee” string property access on Records will only return properties on the Record

Solution: Make exotic Record wrapper immutable

```
const wrapper = Object(#{ a: 1 });

wrapper.foo = "bar";

wrapper.foo // undefined
wrapper.a   // 1
```



ljharb commented 16 days ago · Member

Why would you prefer to disallow it?

Primitives can be as exotic as desired, but it seems preferable to minimize the ways in which objects - even boxed primitives - are exotic.

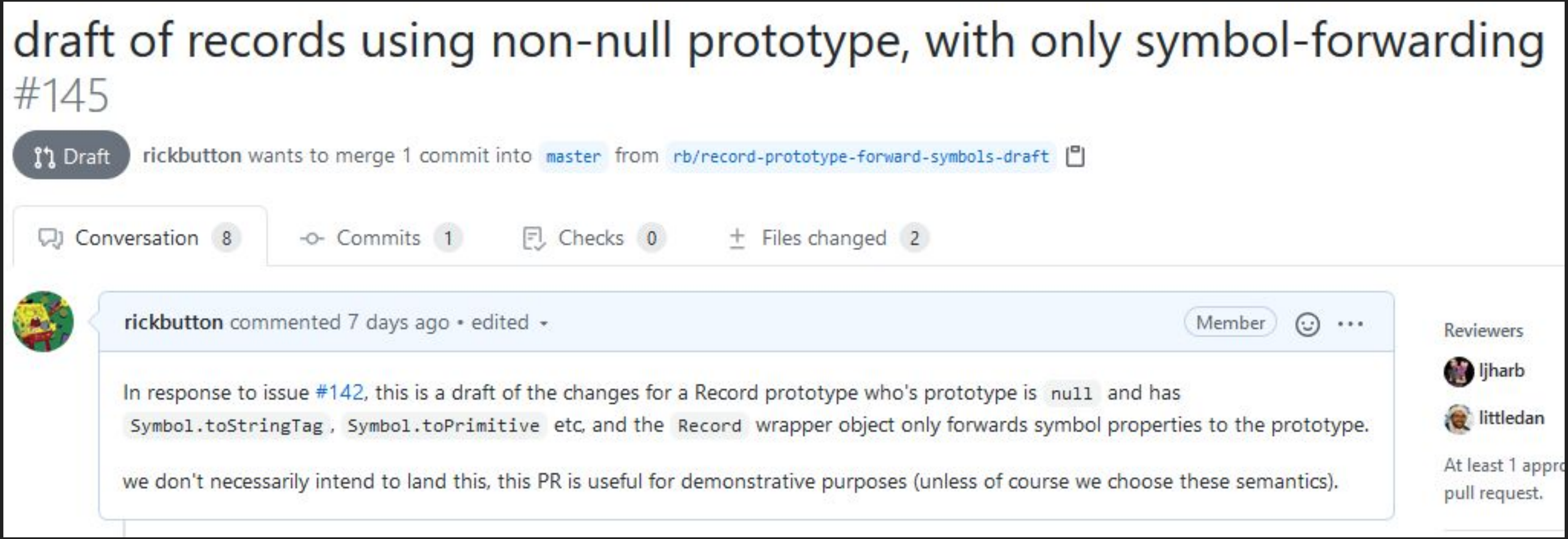
ljharb commented 16 days ago · Member

Additionally, if I can't set Symbols on a boxed Record object, then I can't opt them into any protocols, which is pretty important.

Desire: Opting Record into Symbol protocols, while preserving “string property guarantee”

Alternative: Make Record.prototype an Object with no prototype, rather than null, and only forward symbol properties to prototype

```
Record.prototype.foo = "bar";
const sym = Symbol();
Record.prototype[sym] = "sym";

const record = #{ a: 1 };


record.foo // undefined
record.sym // "sym"
```


draft of records using non-null prototype, with only symbol-forwarding #145

Draft rickbutton wants to merge 1 commit into `master` from `rb/record-prototype-forward-symbols-draft`

rickbutton commented 7 days ago • edited

In response to issue #142, this is a draft of the changes for a Record prototype who's prototype is `null` and has `Symbol.toStringTag`, `Symbol.toPrimitive` etc, and the `Record` wrapper object only forwards symbol properties to the prototype.

we don't necessarily intend to land this, this PR is useful for demonstrative purposes (unless of course we choose these semantics).

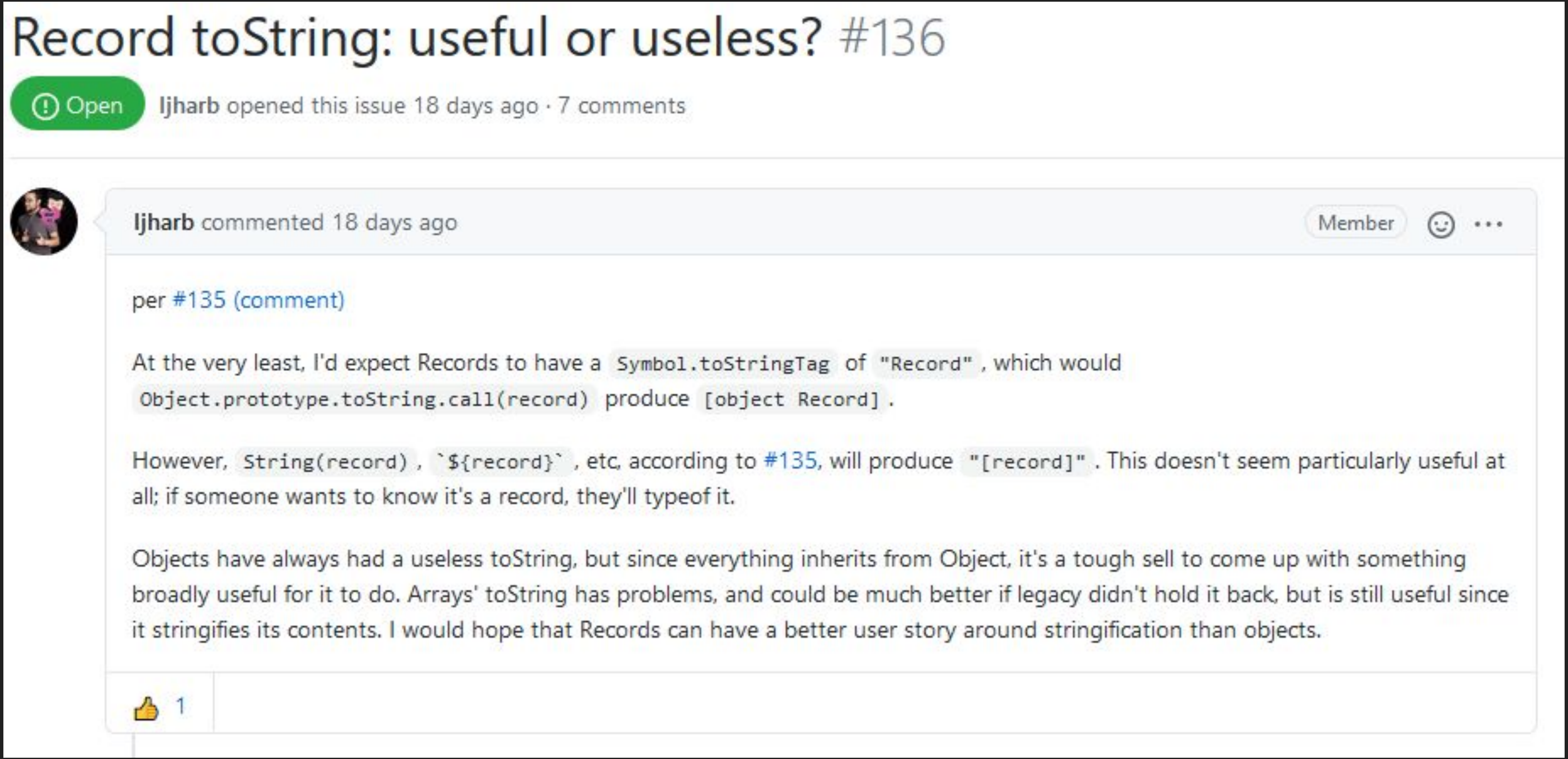
Reviewers

ljharb

littledan

At least 1 appro

pull request.



Record toString: useful or useless? #136

Open ljharb opened this issue 18 days ago · 7 comments

ljharb commented 18 days ago Member

per #135 (comment)

At the very least, I'd expect Records to have a `Symbol.toStringTag` of `"Record"`, which would `Object.prototype.toString.call(record)` produce `[object Record]`.

However, `String(record)`, `` `${record}` ``, etc, according to #135, will produce `"[record]"`. This doesn't seem particularly useful at all; if someone wants to know it's a record, they'll typeof it.

Objects have always had a useless toString, but since everything inherits from Object, it's a tough sell to come up with something broadly useful for it to do. Arrays' toString has problems, and could be much better if legacy didn't hold it back, but is still useful since it stringifies its contents. I would hope that Records can have a better user story around stringification than objects.

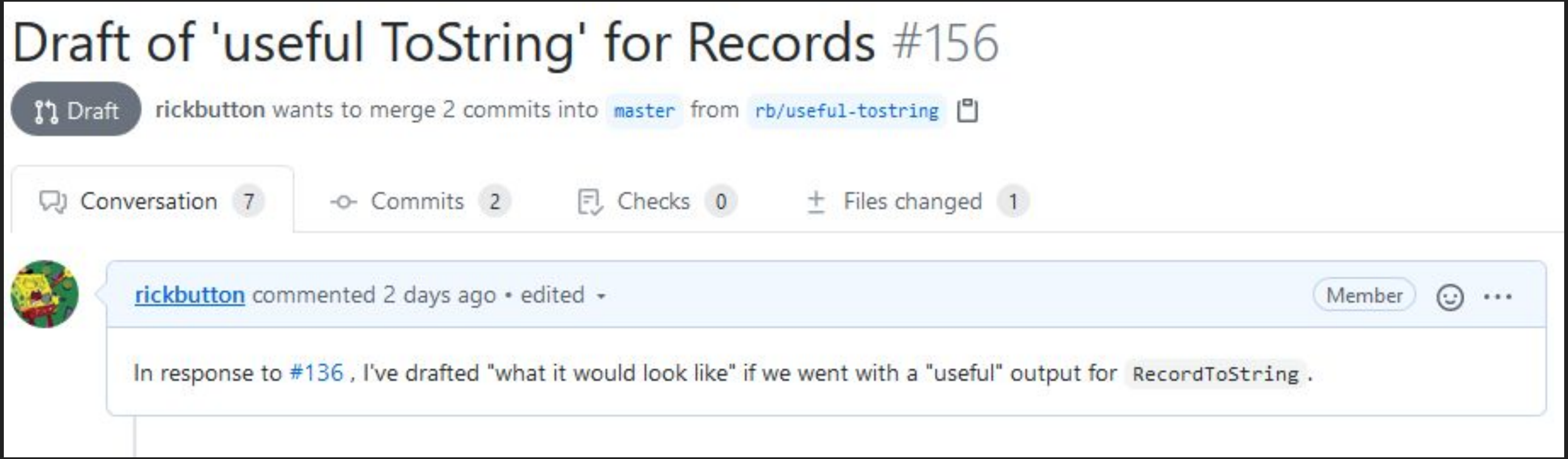
👍 1

Question: What should ToString produce for records?

Currently: [object Record]

Alternative: Something “more useful”

```
const record = #{ a: 1 };

const current = String(record);
asserts(current === "[object Record]");

// if alternative chosen
const alternative = String(record);
asserts(current === "#{ a: 1 }");
```



Draft of 'useful ToString' for Records #156

Draft rickbutton wants to merge 2 commits into `master` from `rb/useful-tostring`

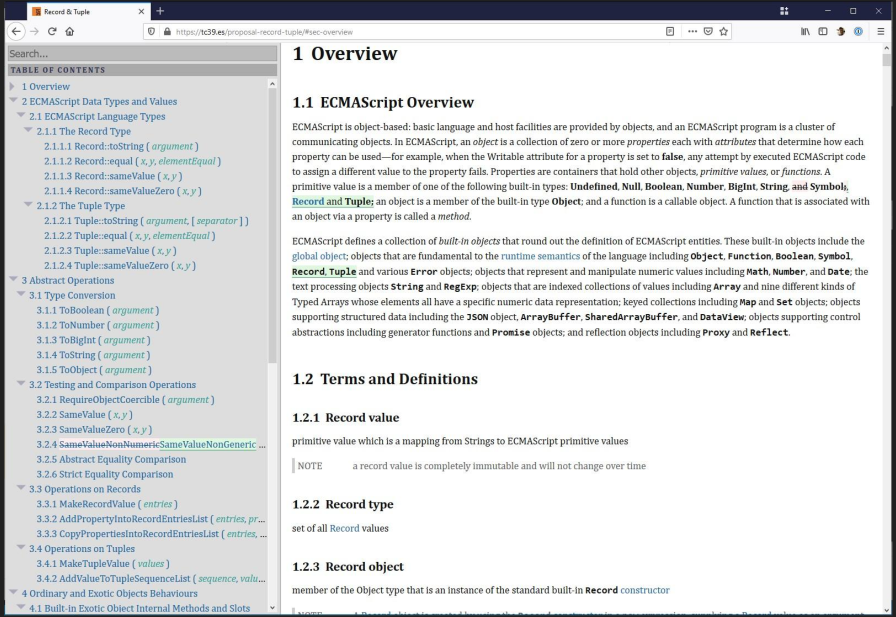
Conversation 7 · Commits 2 · Checks 0 · Files changed 1

rickbutton commented 2 days ago • edited · Member

In response to #136 , I've drafted "what it would look like" if we went with a "useful" output for `RecordToString` .

Record and Tuple Spec Text

https://tc39.es/proposal-record-tuple



Record & Tuple

https://tc39.es/proposal-record-tuple/#sec-overview

Search...

TABLE OF CONTENTS

1 Overview

1.1 ECMAScript Overview

ECMAScript is object-based: basic language and host facilities are provided by objects, and an ECMAScript program is a cluster of communicating objects. In ECMAScript, an *object* is a collection of zero or more *properties* each with *attributes* that determine how each property can be used—for example, when the Writable attribute for a property is set to **false**, any attempt by executed ECMAScript code to assign a different value to the property fails. Properties are containers that hold other objects, *primitive values*, or *functions*. A primitive value is a member of one of the following built-in types: **Undefined**, **Null**, **Boolean**, **Number**, **BigInt**, **String**, ~~and~~ **Symbol**~~;~~, **Record** and **Tuple;** an object is a member of the built-in type **Object**; and a function is a callable object. A function that is associated with an object via a property is called a *method*.

ECMAScript defines a collection of *built-in objects* that round out the definition of ECMAScript entities. These built-in objects include the global object; objects that are fundamental to the runtime semantics of the language including `Object`, `Function`, `Boolean`, `Symbol`, `Record`, `Tuple` and various `Error` objects; objects that represent and manipulate numeric values including `Math`, `Number`, and `Date`; the text processing objects `String` and `RegExp`; objects that are indexed collections of values including `Array` and nine different kinds of Typed Arrays whose elements all have a specific numeric data representation; keyed collections including `Map` and `Set` objects; objects supporting structured data including the `JSON` object, `ArrayBuffer`, `SharedArrayBuffer`, and `DataView`; objects supporting control abstractions including generator functions and `Promise` objects; and reflection objects including `Proxy` and `Reflect`.

1.2 Terms and Definitions

1.2.1 Record value

primitive value which is a mapping from Strings to ECMAScript primitive values

NOTE a record value is completely immutable and will not change over time

1.2.2 Record type

set of all Record values

1.2.3 Record object

member of the Object type that is an instance of the standard built-in `Record` constructor

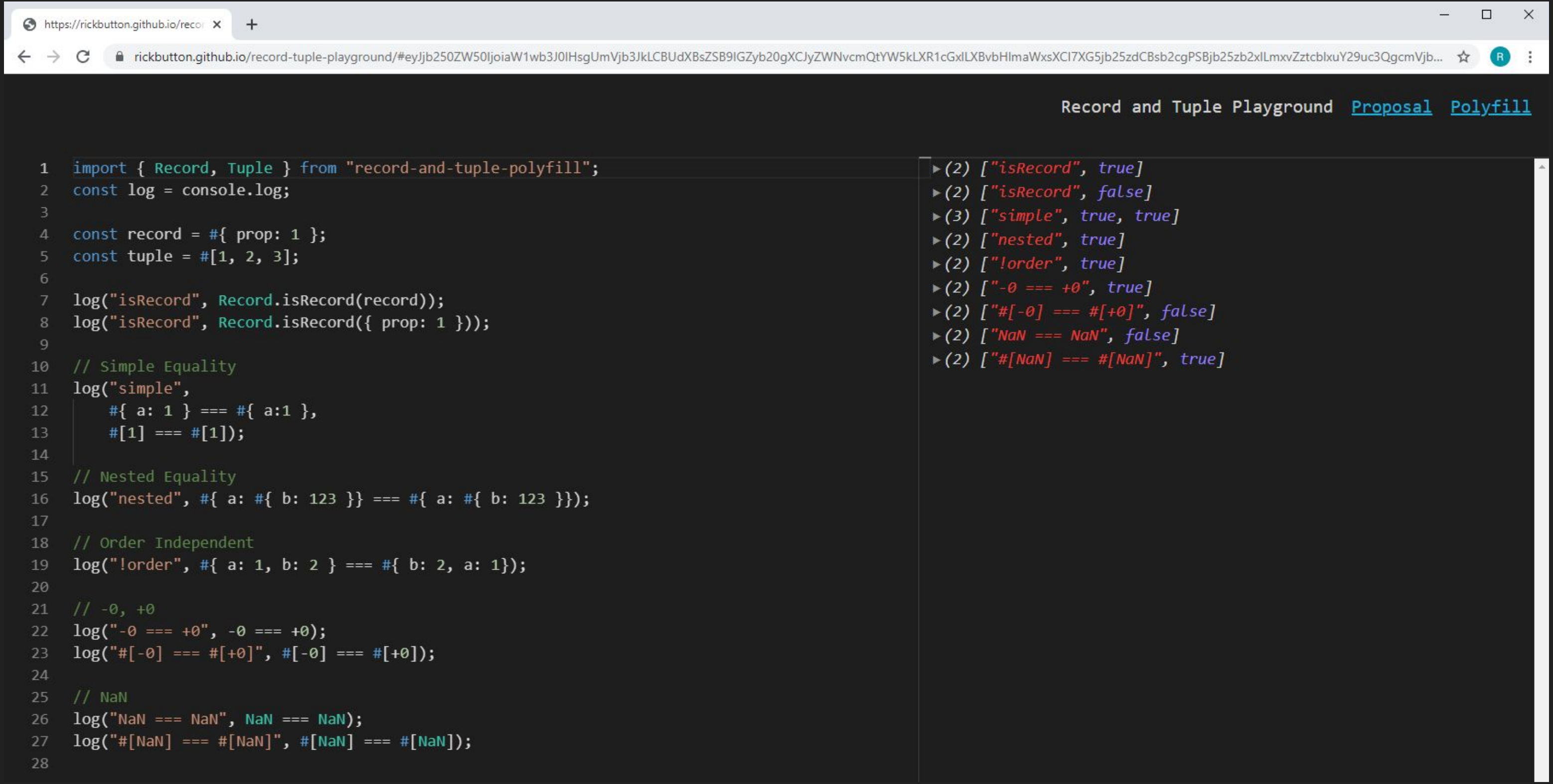
Notable sections:

- RecordEqual and TupleEqual
- Abstract Operations updated
- Record exotic object wrapper
- Tuple exotic object wrapper
- Record initializer syntax & semantics
- Tuple initializer syntax & semantics
- typeof unary expression
- Record & Tuple objects...
- … with the Tuple prototype

Record and Tuple Toy Implementation & Playground

https://github.com/bloomberg/record-tuple-polyfill

https://rickbutton.github.io/record-tuple-playground/



https://rickbutton.github.io/recor

rickbutton.github.io/record-tuple-playground/#eyJjb250ZW50IjoiaW1wb3J0IHsgUmVjb3JkLCBUdXBsZSB9IGZyb20gXCJyZWNvcmQtYW5kLXR1cGxlLXBvbHlmaWxsXCI7XG5jb25zdCBsb2cgPSBjb25zb2xlLmxvZztcblxuY29uc3QgcmVjb...

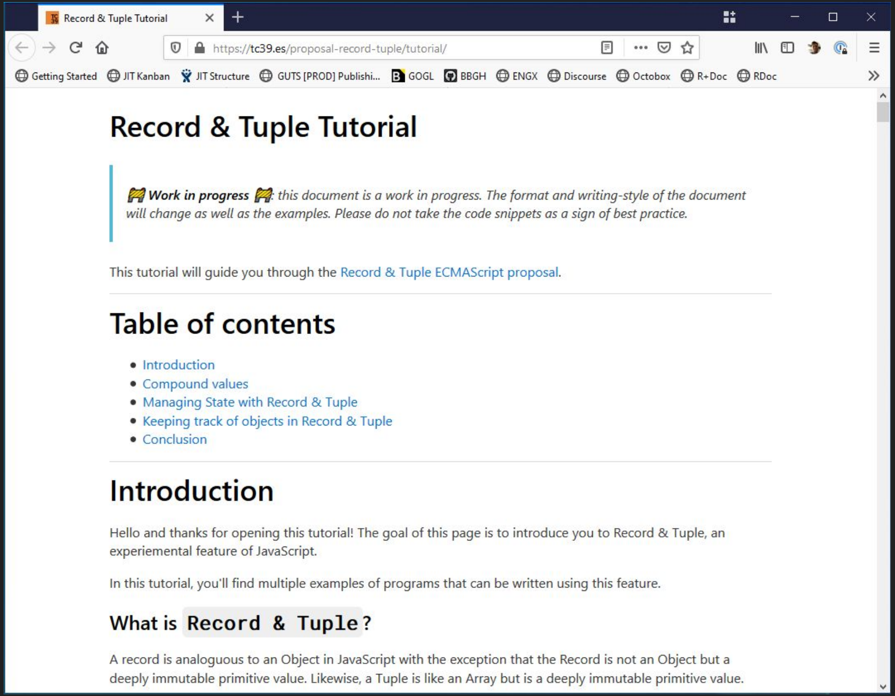
Record and Tuple Playground Proposal Polyfill

```js
import { Record, Tuple } from "record-and-tuple-polyfill";
const log = console.log;

const record = #{ prop: 1 };
const tuple = #[1, 2, 3];

log("isRecord", Record.isRecord(record));
log("isRecord", Record.isRecord({ prop: 1 }));

// Simple Equality
log("simple",
    #{ a: 1 } === #{ a:1 },
    #[1] === #[1]);

// Nested Equality
log("nested", #{ a: #{ b: 123 }} === #{ a: #{ b: 123 }});

// Order Independent
log("!order", #{ a: 1, b: 2 } === #{ b: 2, a: 1});

// -0, +0
log("-0 === +0", -0 === +0);
log("#[-0] === #[+0]", #[-0] === #[+0]);

// NaN
log("NaN === NaN", NaN === NaN);
log("#[NaN] === #[NaN]", #[NaN] === #[NaN]);
```

```
▶(2) ["isRecord", true]
▶(2) ["isRecord", false]
▶(3) ["simple", true, true]
▶(2) ["nested", true]
▶(2) ["!order", true]
▶(2) ["-0 === +0", true]
▶(2) ["#[-0] === #[+0]", false]
▶(2) ["NaN === NaN", false]
▶(2) ["#[NaN] === #[NaN]", true]
```

Record and Tuple Documentation Bits

https://tc39.es/proposal-record-tuple/tutorial/
https://tc39.es/proposal-record-tuple/cookbook/



Record & Tuple Tutorial

*🚧 **Work in progress** 🚧: this document is a work in progress. The format and writing-style of the document will change as well as the examples. Please do not take the code snippets as a sign of best practice.*

This tutorial will guide you through the Record & Tuple ECMAScript proposal.

Table of contents

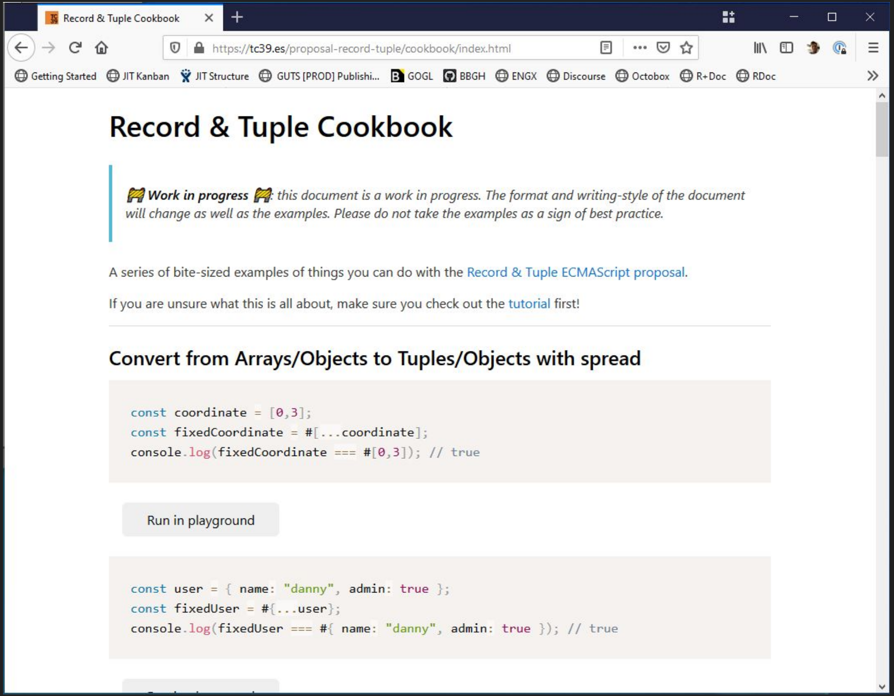
Introduction

Hello and thanks for opening this tutorial! The goal of this page is to introduce you to Record & Tuple, an experiemental feature of JavaScript.

In this tutorial, you'll find multiple examples of programs that can be written using this feature.

What is `Record & Tuple`?

A record is analoguous to an Object in JavaScript with the exception that the Record is not an Object but a deeply immutable primitive value. Likewise, a Tuple is like an Array but is a deeply immutable primitive value.



Record & Tuple Cookbook

*🚧 **Work in progress** 🚧: this document is a work in progress. The format and writing-style of the document will change as well as the examples. Please do not take the examples as a sign of best practice.*

A series of bite-sized examples of things you can do with the Record & Tuple ECMAScript proposal.

If you are unsure what this is all about, make sure you check out the tutorial first!

Convert from Arrays/Objects to Tuples/Objects with spread

```
const coordinate = [0,3];
const fixedCoordinate = #[...coordinate];
console.log(fixedCoordinate === #[0,3]); // true
```

Run in playground

```
const user = { name: "danny", admin: true };
const fixedUser = #{...user};
console.log(fixedUser === #{ name: "danny", admin: true }); // true
```

We also started reaching out to the W3C TAG for a preliminary review.
The review is now approved.

https://github.com/w3ctag/design-reviews/issues/518

Seeking Stage 2

- Last meeting’s open questions are now solved.
- Toy Implementation & Spec Text written.
- Positive feedback in framework outreach calls.

We are now seeking for Stage 2 and reviewers.

Stage 2?