

Record & Tuple

for Stage 2

Robin Ricard & Rick Button
Bloomberg

*Advisor: Daniel Ehrenberg
Igalia*

Rationale for Record & Tuple primitives

Or [what was presented for Stage 1](#) and a few more reasons why this is a good idea!

Compounding primitives (with strings)

```
const grid = {  
  "0:0": "player",  
  "3:5": "enemy",  
  "0:1": "wall",  
};
```

```
console.log("at 0:0", grid["0:0"]); // player  
console.log("at 0:0x0", grid["0:0x0"]); // undefined
```

Compounding primitives (with JSON strings)

```
const grid = {  
  '{"x":0,"y":0}': "player",  
  '{"x":3,"y":5}': "enemy",  
  '{"x":0,"y":1}': "wall",  
};
```

```
console.log("at 0:0x0", grid[JSON.stringify({  
  x: 0,  
  y: 0x0,  
})]); // player
```

```
console.log("at 0:0x0", grid[JSON.stringify({  
  y: 0x0,  
  x: 0,  
})]); // undefined
```

Compounding primitives with Tuple!

```
const grid = new Map([  
  [#[0, 0], "player"],  
  [#[3, 5], "enemy"],  
  [#[0, 1], "wall"],  
]);
```

```
console.log("at 0:0", grid.get(#[0, 0])); // player  
console.log("at 0:0", grid.get(#[0, 0x0])); // player
```

All about the equality!

```
function isAtOrigin(coordinate) {  
    return coordinate === #[0, 0];  
}
```

```
const c1 = #[1, 2];  
const c2 = #[c1[0] - 1, c1[1] * 2 - 4];  
console.log("c1 at origin?", isAtOrigin(c1)); // false  
console.log("c2 at origin?", isAtOrigin(c2)); // true
```

Same is possible with Record! (but keyed)

```
const grid = new Map([
  [{x:0, y:0}, "player"],
  [{x:3, y:5}, "enemy"],
  [{x:0, y:1}, "wall"],
]);

console.log("at 0:0", grid.get({x:0, y:0x0})); // player
console.log("at 0:0", grid.get({y:0, x:0x0})); // player

function isAtOrigin(coordinate) {
  return coordinate === {x:0, y:0};
}

const c1 = {x:1, y:2};
const c2 = {x: c1.x - 1, y: c1.y * 2 - 4 };
console.log("c1 at origin?", isAtOrigin(c1)); // false
console.log("c2 at origin?", isAtOrigin(c2)); // true
```

Deep immutability!

```
const record = #{  
  a: #{  
    foo: "bar",  
  },  
};  
func(record);  
// runtime guarantees that  
// record is entirely unchanged  
assert(record.a.foo === "bar");
```

```
const object = {  
  a: {  
    foo: "bar",  
  },  
};  
Object.freeze(object);  
func(object);  
// func is able to mutate object's  
// keys even if object is frozen  
//  
// side effects make the state  
// of the program after func()  
// harder to reason about
```


Deep immutability!

```
const record = #{  
  a: #{  
    foo: "bar",  
  },  
};  
func(record);  
// runtime guarantees that  
// record is entirely unchanged  
assert(record.a.foo === "bar");
```

```
const clonedObject = JSON.parse(  
  JSON.stringify(object)  
);  
func(clonedObject);  
// now func can have side effects on  
// clonedObject, object is untouched  
// but at what cost?  
assert(object.a.foo === "bar");
```

Why not frozen objects and/or userland classes?

- **Equality!**

- Objects & Arrays have equality by **identity**
- *Record* & *Tuple* have equality by **value**
- Equality is not only about `==` and `===` but also about Map keys/Set values

- **Deep immutability**

- Guaranteed immutable data structures
- Trustworthy (deep) equality
- No need for deep cloning with `JSON.parse(JSON.stringify(obj))`

- **Accessed the same as Objects & Arrays**

- Write functions that can access both Objects/Arrays and *Record* & *Tuple*
- Unlike userland immutable libraries, access *Record* & *Tuple* using the same notations as *Object*/*Array* instead of functions

Ecosystem split issue

```
const ProfileRecord = Immutable.Record({
  name: "Anonymous User",
  githubHandle: null,
});

function getGithubUrl(profile) {
  if (Immutable.Record.isRecord(profile)) {
    return `https://github.com/${
      profile.get("githubHandle")
    }`;
  }
  return `https://github.com/${
    profile.githubHandle
  }`;
}
```

```
const jobResult = Immutable.fromJS(
  ExternalLib.processJob(
    jobDescription.toJS()
  )
);
```

Equality Semantics



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

Open Github Issues

- 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](#))
- ... and more

Highlights

We have initial conclusions for all of these issues checked into the spec-text, and have sketched out alternatives in draft PRs.

Let's walk through the highlights!

Desire: Guarantee that
accesses to the Record
wrapper reflect what's in the
underlying Record

Solution: Make the Record
wrapper frozen

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

```
wrapper.foo = "bar";
```

```
wrapper.foo // undefined
```

```
wrapper.a   // 1
```

Desire: String property access on Records always accesses Record entries, not strings on the prototype.

Current: Record wrappers have a null prototype.



```
assert(Record.prototype === null);
```

Option: 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` 

 Conversation 8  Commits 1  Checks 0  Files changed 2



rickbutton commented 7 days ago • edited •

Member  ...

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).

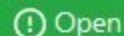
Reviewers

 ljharb

 littledan

At least 1 approved pull request.

Record toString: useful or useless? #136



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

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 ▾

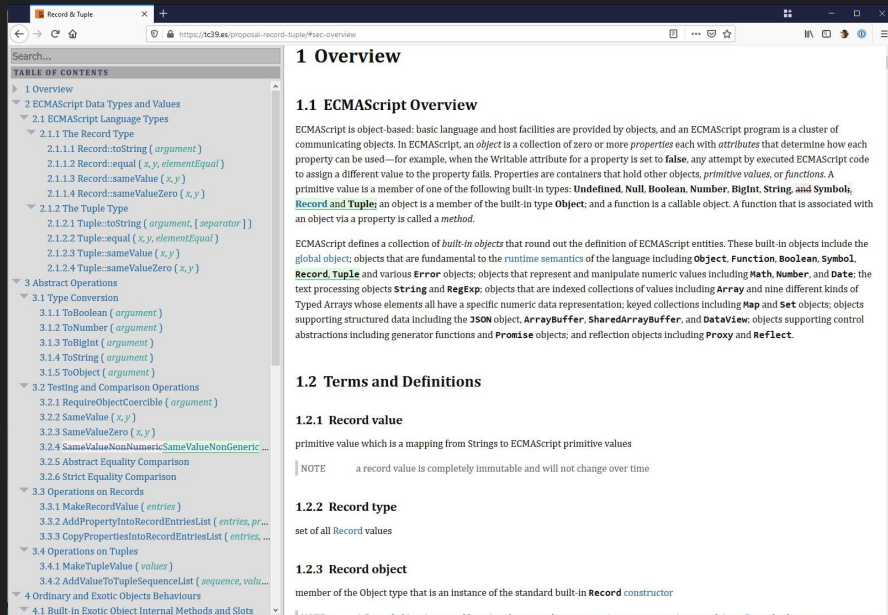
Member



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>



Notable sections:

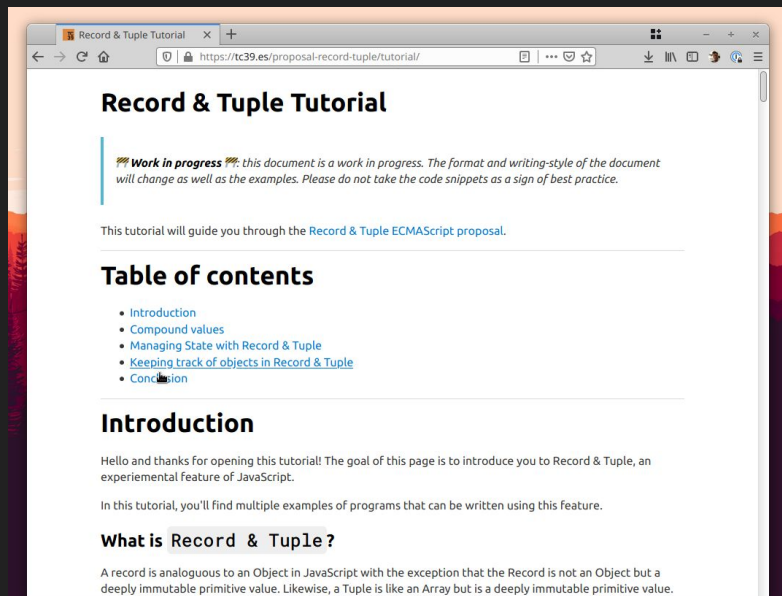
- [RecordEqual](#) and [TupleEqual](#)
- [Abstract Operations](#) updated
- [Record object](#) wrapper
- [Tuple object](#) wrapper
- [Record initializer](#) syntax & semantics
- [Tuple initializer](#) syntax & semantics
- [typeof unary expression](#)
- [Record](#) & [Tuple](#) objects...
- ... with the [Tuple prototype](#)

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



Record and Tuple Documentation Bits

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



The screenshot shows the 'Record & Tuple Tutorial' page in a web browser. The page title is 'Record & Tuple Tutorial'. Below the title is a warning: 'Work in progress' with a note that the document is a work in progress and its format and writing style may change. The tutorial's purpose is to guide the reader through the Record & Tuple ECMAScript proposal. A 'Table of contents' section lists: Introduction, Compound values, Managing State with Record & Tuple, Keeping track of objects in Record & Tuple, and Conclusion. The 'Introduction' section begins with a welcome message and states the goal of the page. It also mentions that multiple examples of programs can be found using this feature. The 'What is Record & Tuple?' section explains that a Record is analogous to an Object but is a deeply immutable primitive value, and a Tuple is like an Array but is also a deeply immutable primitive value.

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](#)
- [Compound values](#)
- [Managing State with Record & Tuple](#)
- [Keeping track of objects in Record & Tuple](#)
- [Conclusion](#)

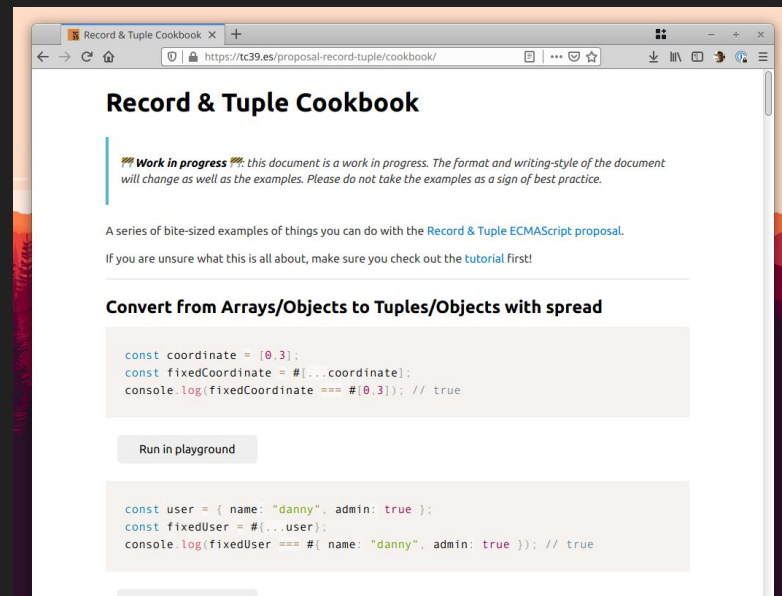
Introduction

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



The screenshot shows the 'Record & Tuple Cookbook' page in a web browser. The page title is 'Record & Tuple Cookbook'. It includes the same 'Work in progress' warning as the tutorial page. The page describes a series of bite-sized examples for the Record & Tuple ECMAScript proposal and suggests checking out the tutorial if the reader is unsure. A section titled 'Convert from Arrays/Objects to Tuples/Objects with spread' provides two code examples. The first example shows how to convert an array [0, 3] into a fixed coordinate object. The second example shows how to convert a user object into a fixed user object. Both examples use the spread operator (#) to create a new object with the same properties as the original, but as a deeply immutable primitive value. A 'Run in playground' button is provided for the first example.

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?