EJSON

EJSON is an extension of JSON to support more types. It supports all JSON-safe types, as well as:

- **Date** (JavaScript `Date`)
- **Binary** (JavaScript `Uint8Array` or the result of `EJSON.newBinary`)
- **Special numbers** (JavaScript `NaN`, `Infinity`, and `-Infinity`)
- **Regular expressions** (JavaScript `RegExp`)
- **User-defined types** (see `EJSON.addType`. For example, `Mongo.ObjectID` is implemented this way.)

All EJSON serializations are also valid JSON. For example an object with a date and a binary buffer would be serialized in EJSON as:

```
{
  "d": { "$date": 1358205756553 },
  "b": { "$binary": "c3VyZS4=" }
}
```

Meteor supports all built-in EJSON data types in publishers, method arguments and results, Mongo databases, and `Session` variables.

{% apibox "EJSON.parse" %}

{% apibox "EJSON.stringify" %}

{% apibox "EJSON.fromJSONValue" %}

{% apibox "EJSON.toJSONValue" %}

{% apibox "EJSON.equals" %}

{% apibox "EJSON.clone" %}

{% apibox "EJSON.newBinary" %}

Buffers of binary data are represented by `Uint8Array` instances on JavaScript platforms that support them. On implementations of JavaScript that do not support `Uint8Array`, binary data buffers are represented by standard arrays containing numbers ranging from 0 to 255, and the `$Uint8ArrayPolyfill` key set to `true`.

{% apibox "EJSON.isBinary" %}

{% apibox “EJSON.addType” %}

The factory function passed to the `EJSON.addType` method should create an instance of our custom type and initialize it with values from an object passed as the first argument of the factory function. Here is an example:

```
class Distance {
  constructor(value, unit) {
    this.value = value;
    this.unit = unit;
  }

  // Convert our type to JSON.
  toJSONValue() {
    return {
      value: this.value,
      unit: this.unit
    };
  }

  // Unique type name.
  typeName() {
    return 'Distance';
  }
}

EJSON.addType('Distance', function fromJSONValue(json) {
  return new Distance(json.value, json.unit);
});

EJSON.stringify(new Distance(10, 'm'));
// Returns '{"$type":"Distance","$value":{"value":10,"unit":"m"}}'
```

When you add a type to EJSON, Meteor will be able to use that type in:

- publishing objects of your type if you pass them to publish handlers.
- allowing your type in the return values or arguments to methods.
- storing your type client-side in Minimongo.
- allowing your type in `Session` variables.

Instances of your type must implement `typeName` and `toJSONValue` methods, and may implement `clone` and `equals` methods if the default implementations are not sufficient.

{% apibox “EJSON.CustomType#typeName” %} {% apibox “EJSON.CustomType#toJSONValue” %}

For example, the `toJSONValue` method for `Mongo.ObjectID` could be:

```
function () {
  return this.toHexString();
}
```

{% apibox “EJSON.CustomType#clone” %}

If your type does not have a `clone` method, `EJSON.clone` will use `toJSONValue` and the factory instead.

{% apibox “EJSON.CustomType#equals” %}

The `equals` method should define an equivalence relation. It should have the following properties:

- *Reflexivity* - for any instance `a`: `a.equals(a)` must be true.
- *Symmetry* - for any two instances `a` and `b`: `a.equals(b)` if and only if `b.equals(a)`.
- *Transitivity* - for any three instances `a`, `b`, and `c`: `a.equals(b)` and `b.equals(c)` implies `a.equals(c)`.

If your type does not have an `equals` method, `EJSON.equals` will compare the result of calling `toJSONValue` instead.