The “right” JSON date format

994

246

I've seen so many different standards for the JSON date format:

```
"\"\\/Date(1335205592410)\\/\""          .NET JavaScriptSerializer
"\"\\/Date(1335205592410-0500)\\/\""     .NET DataContractJsonSerializer
"2012-04-23T18:25:43.511Z"               JavaScript built-in JSON object
"2012-04-21T18:25:43-05:00"              ISO 8601
```

Which one is the right one? Or best? Is there any sort of standard on this?

javascript json

edited Dec 5 '14 at 10:35
bluish
14.7k 19 94 151

asked Apr 23 '12 at 18:32
Kamyar Nazeri
13.7k 13 44 82

62 There is no date format in JSON, there's only strings a de-/serializer decides to map to date values. – user395760 Apr 23 '12 at 18:34

10 `strings` , `numbers` , `true` , `false` , `null` , `objects` and `arrays` – Russ Cam Apr 23 '12 at 18:44

10 However, *JavaScript built-in JSON object* and *ISO8601* contains all the information to be understand by human and computer and does not relies on the beginning of the computer era (1970-1-1). – poussma Jan 23 '13 at 13:58

7 Perhaps this should be changed to ask for the best convention, since the answers herein dismiss the question. – David Rivers Apr 19 '13 at 15:18

11 What you called "JavaScript built-in JSON object" is also ISO8601-compliant. Standard allows for specifying decimal fraction of a second and for two ways of describing time zone. See W3C's note. – skalee May 4 '13 at 18:21

13 Answers

```
2012-04-23T18:25:43.511Z
```

Here's why:

1. It's human readable but also succinct
2. It sorts correctly
3. It includes fractional seconds, which can help re-establish chronology
4. It conforms to ISO 8601
5. ISO 8601 has been well-established internationally for more than a decade
6. ISO 8601 is endorsed by W3C, RFC3339, and XKCD

That being said, every date library ever written can understand "milliseconds since 1970". So for easy portability, ThiefMaster is right.

edited Apr 23 at 6:06

acdcjunior
86.9k 23 211 205

answered Apr 11 '13 at 15:20

funroll
25.6k 7 44 54

26 This is also the preferred representations according to ECMA: `JSON.stringify({'now': new Date()}) "{"now":"2013-10-21T13:28:06.419Z"}"` – Steven Oct 21 '13 at 13:28

9 I would add another important reason to the list: it's locale-independent. If you had a date like 02-03-2014 you'd need additional information to know if it refers to the 3rd of Feb or the 2nd of March. It depends on whether US-format or other format is used. – Juanal Jul 2 '14 at 7:41

95 upvote for mentioning and linking xkcd :D @ajorquera I usually use momentjs for this. I've also seen issues with IE in this regard – fholzer Apr 29 '15 at 16:12

45 Regarding the second point, it does not sort correctly after the year 10000. We do have almost 8000 years to come up with a new format though, so it's probably not an issue. – Erfa Aug 19 '15 at 14:29

7 Actually, @Erfa, since `-` comes before digits in `ASCII`, it will sort just fine until the year 100,000. ;P – Ben Leggiero May 2 '16 at 18:23

JSON does not know anything about dates. What .NET does is a non-standard hack/extension.


14.2k 19 88 115


246k 63 486 568

2 Ugh, I'd expect an error... But at least firefox does stringify it... well, it's not part of the JSON standard so I wouldn't feed a Date object to a JSON serializer - it might not work in all browsers. Apparently it's a common idea for JSON serializers to use a `toJSON()` function if it exists on an unknown object. At least Firefox does that for Date objects and Date objects do have such a method. – ThiefMaster Apr 23 '12 at 18:40

3 stackoverflow.com/questions/10286385/… - let's see if someone knows why FF behaves like that. – ThiefMaster Apr 23 '12 at 18:47

8 If you go this route, make sure you don't need to deal with dates earlier than 1970! – Ben Dolman May 9 '13 at 21:14

8 As @BenDolman said, this "solution" doesn't deal appropriately with dates prior to Jan 1st, 1970 (Epoch). Also, there is a reason ISO8601 exists in the first place. Here on Earth we have things called "time zones." Where is that in milliseconds? JSON may not have a standard for dates, but dates exist outside of JSON, and there *is* a standard for that. funroll's answer is the correct one (see also: xkcd.com/1179). – JoeLinux Nov 14 '13 at 20:19

5 It is maybe also worth mentioning that (milli)seconds from 1970 isn't predictable for dates in the future because we have leap seconds. So I wouldn't use if for inter-process communication and data storage. It is however nice to use internally in a program since it can be stored in a single integer which gives you some performance benefits. – Brodie Garnet Nov 3 '15 at 13:56

37

There is no right format; The **JSON specification** does not specify a format for exchanging dates which is why there are so many different ways to do it.

The best format is arguably a date represented in ISO 8601 format (see Wikipedia); it is a well known and widely used format and can be handled across many different languages, making it very well suited for interoperability. If you have control over the generated json, for example, you provide data to other systems in json format, choosing 8601 as the date interchange format is a good choice.

If you do not have control over the generated json, for example, you are the consumer of json from several different existing systems, the best way of handling this is to have a date parsing utility function to handle the different formats expected.

edited Feb 27 '18 at 16:31


Basil Bourque
125k 35 424 596

answered Apr 23 '12 at 18:35


Russ Cam
106k 24 171 231

2 @mlissner but that's an *opinion* on which one is best. ISO-8601 is a standard, but it's not *the standard for JSON* (even though I'd be inclined to use it); for example, Microsoft decided not to use it (msdn.microsoft.com/en-us/library/…). The best practice is to stick with one (sensible) convention , whatever that is. As I stated in the answer, the best way of handling this is to define a date parsing utility function that can handle the expected formats. If you

The JSON schema specification actually says that dates that are verified by a schema must be in 8601 format. – gnasher729 Jan 16 '15 at 16:20

3 @gnasher729 do you have a link? – Russ Cam Jan 16 '15 at 21:56

@vallismortis - That is a draft specification for defining a schema for a given json structure exchanged between parties, not the format for dates in the json specification. I'm going to revise my answer as based on the comments, it doesn't appear I've made it clear enough – Russ Cam Jun 27 '15 at 0:32

24

From RFC 7493 (The I-JSON Message Format):

I-JSON stands for either Internet JSON or Interoperable JSON, depending on who you ask.

> Protocols often contain data items that are designed to contain timestamps or time durations. It is RECOMMENDED that all such data items be expressed as string values in ISO 8601 format, as specified in RFC 3339, with the additional restrictions that uppercase rather than lowercase letters be used, that the timezone be included not defaulted, and that optional trailing seconds be included even when their value is "00". It is also RECOMMENDED that all data items containing time durations conform to the "duration" production in Appendix A of RFC 3339, with the same additional restrictions.

edited May 20 '15 at 19:40

John Flatness
25.3k 5 66 74

answered Mar 27 '15 at 18:48

Bryan Larsen
6,509 6 44 37

2 This is also the format produced by `Date().toISOString()` and `Date().toJSON()`, with the limitation that `Date` doesn't track a timezone value, and hence always emits the timestamps in the UTC (`Z`) timezone. The value can be parsed using `new Date("...")` and `Date.parseDate` . – Søren Løvborg Sep 21 '15 at 9:06

12

Just for reference I've seen this format used:

```
Date.UTC(2017,2,22)
```

It works with **JSONP** which is supported by the `$.getJSON()` function. Not sure I would go so far as to recommend this approach... just

Kind of a pet hate, but many people believe that UTC is just the new name for GMT -- wrong! If your system does not implement leap seconds then you are using GMT (often called UTC despite being incorrect). If you do fully implement leap seconds you really are using UTC. Future leap seconds cannot be known; they get published by the IERS as necessary and require constant updates. If you are running a system that attempts to implement leap seconds but contains and out-of-date reference table (more common than you might think) then you have neither GMT, nor UTC, you have a wonky system pretending to be UTC.

These date counters are only compatible when expressed in a broken down format (y, m, d, etc). They are NEVER compatible in an epoch format. Keep that in mind.

answered Feb 27 '17 at 7:33

Tel
224 2 2

4 I wouldn't use that format, but the rest of the info you provided is very useful, thank you! – Robert Mikes Dec 25 '17 at 11:47

4

JSON itself has no date format, it does not care how anyone stores dates. However, since this question is tagged with javascript, I assume you want to know how to store javascript dates in JSON. You can just pass in a date to the `JSON.stringify` method, and it will use `Date.prototype.toJSON` by default, which in turns uses `Date.prototype.toISOString` (MDN on Date.toJSON):

```
const json = JSON.stringify(new Date());
const parsed = JSON.parse(json); //2015-10-26T07:46:36.611Z
const date = new Date(parsed); // Back to date object
```

I also found it useful to use the `reviver` parameter of `JSON.parse` (MDN on JSON.parse) to automatically convert ISO strings back to javascript dates whenever I read JSON strings.

```
const isoDatePattern = new RegExp(/\d{4}-[01]\d-[0-3]\dT[0-2]\d:[0-5]\d:[0-5]\d\.\d+
([+-][0-2]\d:[0-5]\d|Z)/);

const obj = {
 a: 'foo',
 b: new Date(1500000000000) // Fri Jul 14 2017, etc...
}
```

```
    }
    return value; // leave any other value as-is
});
console.log(parsed.b); // // Fri Jul 14 2017, etc...
```

edited May 23 at 10:38

answered May 8 at 13:04

Justus Romijn
11.9k 2 41 55

2

I believe that the best format for *universal interoperability* is not the ISO-8601 string, but rather the format used by EJSON:

```
{ "myDateField": { "$date" : <ms-since-epoch> } }
```

As described here: https://docs.meteor.com/api/ejson.html

Benefits

1. **Parsing performance:** If you store dates as ISO-8601 strings, this is great if you are expecting a date value under that particular field, but if you have a system which must determine value types without context, you're parsing every string for a date format.
2. **No Need for Date Validation:** You need not worry about validation and verification of the date. Even if a string matches ISO-8601 format, it may not be a real date; this can never happen with an EJSON date.
3. **Unambiguous Type Declaration:** as far as generic data systems go, if you wanted to store an ISO string **as a string** in one case, and a *real system date* in another, generic systems adopting the ISO-8601 string format will not allow this, mechanically (without escape tricks or similar awful solutions).

Conclusion

I understand that a human-readable format (ISO-8601 string) is helpful and more *convenient* for 80% of use cases, and indeed no-one should ever be told *not* to store their dates as ISO-8601 strings if that's what their applications understand, **but** for a universally accepted transport format which should guarantee certain values to *for sure* be dates, how can we allow for ambiguity and need for so much validation?

edited Feb 8 at 16:04

answered Feb 8 at 9:01

1

The prefered way is using `2018-04-23T18:25:43.511Z` ...

The picture below shows why this is the prefered way:

```
> const now = new Date();
< undefined
> now
< Thu Jan 31 2019 22:24:59 GMT+1100 (AEDT)
> now.toJSON;
< ƒ toJSON() { [native code] }
> now.toJSON();
< "2019-01-31T11:24:59.323Z"
> new Date("2019-01-31T11:24:59.323Z");
< Thu Jan 31 2019 22:24:59 GMT+1100 (AEDT)
>
```

So as you see Date has a native Method `toJSON` , which `return` in this format and can be easily converted to `Date` again...

answered Jan 31 at 11:33

Alireza

57.7k 14 191 126

2 Correct! The JSON Data Interchange Syntax does not specify the standard: ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf but in practice, ISO 8601 compatible formats are more desirable across platforms including JavaScript runtime. – Kamyar Nazeri Jan 31 at 14:45

0

In Sharepoint 2013, getting data in JSON there is no format to convert date into date only format, because in that date should be in ISO format

```
yourDate.substring(0,10)
```

This may be helpful for you

-4

There is only one correct answer to this and most systems get it wrong. Number of milliseconds since epoch, aka a 64 bit integer. Time Zone is a UI concern and has no business in the app layer or db layer. Why does your db care what time zone something is, when you know it's going to store it as a 64 bit integer then do the transformation calculations.

Strip out the extraneous bits and just treat dates as numbers up to the UI. You can use simple arithmetic operators to do queries and logic.

answered Dec 16 '15 at 23:56

Chad Wilson
129 3

Comments have been moved to chat. – Jon Clements ♦ Jun 7 '16 at 17:13

5 Now you have 2 problems: Which epoch should you choose, and which milliseconds should you count? Probably the most common choice is Unix time (1970-01-01T00:00:00 UTC and SI milliseconds except for those which fall within a leap second), but of course that makes future times undefined. – aij Sep 21 '16 at 18:51

2 So how do you represent microseconds? RFC3339 works fine with any precision, you'll have a reader that parses the timezone and gives you the right time stamp, and it's additional information. Calendar apps usually care about time zones. – gnasher729 Nov 4 '16 at 17:05

10 Timezone is not a UI concern, unless you don't mind missing your next flight. Flights are posted in local time and follow specific rules for DST changes. Losing the offset means losing important business information – Panagiotis Kanavos Dec 5 '16 at 12:00

1 Some further counterarguments include the ability to represent times before 1970 (assuming that particular epoch), and JSONs tendency to be somewhat human-readable. – Timo Dec 13 '16 at 16:36

-4

it is work for me with parse Server

```
{
    "ContractID": "203-17-DC0101-00003-10011",
    "Supplier":"Sample Co., Ltd",
    "Value":12345.80,
    "Curency":"USD",
```

edited Dec 15 '17 at 2:11

Rosdi Kasim
12.6k 16 87 125

answered Aug 24 '17 at 6:45

Kemal Karadag
150 1 4

-6

The following code has worked for me. This code will print date in **DD-MM-YYYY** format.

```
DateValue=DateValue.substring(6,8)+"-"+DateValue.substring(4,6)+"-
"+DateValue.substring(0,4);
```

else, you can also use:

```
DateValue=DateValue.substring(0,4)+"-"+DateValue.substring(4,6)+"-
"+DateValue.substring(6,8);
```

edited Jun 6 '18 at 12:56

Phil3992
767 5 16 36

answered Apr 10 '18 at 6:11

Aniket Kumar Shrivastava
1 3

-14

I think that really depends on the use case. In many cases it might be more beneficial to use a proper object model (instead of rendering the date to a string), like so:

```
{
"person" :
      {
 "name" : {
   "first": "Tom",
   "middle": "M",
  ...
}
 "dob" :  {
```

```
        "timeZone": "America/New_York"
    }
  }
}
```

Admittedly this is more verbose than RFC 3339 but:

- it's human readable as well
- it implements a proper object model (as in OOP, as far as JSON permits it)
- it supports time zones (not just the UTC offset at the given date and time)
- it can support smaller units like milliseconds, nanoseconds, ... or simply fractional seconds
- it doesn't require a separate parsing step (to parse the date-time string), the JSON parser will do everything for you
- easy creation with any date-time framework or implementation in any language
- can easily be extended to support other calendar scales (Hebrew, Chinese, Islamic ...) and eras (AD, BC, ...)
- it's year 10000 safe ;-) (RFC 3339 isn't)
- supports all-day dates and floating times (Javascript's `Date.toJSON()` doesn't)

I don't think that correct sorting (as noted by funroll for RFC 3339) is a feature that's really needed when serializing a date to JSON. Also that's only true for date-times having the same time zone offset.

edited Sep 6 '18 at 3:52
Mukus
3,236 2 29 48

answered Jan 1 '16 at 15:30
Marten
3,144 1 8 22

7 I doubt anyone will be using json in the year 10000, or even that by that time the year 10000 will still be year 10000. But if both of those things are still true by then, the format can simply be expanded to contain a 3 digit century component. So I'd say people can safely stick with RFC 3339, at least until the year 9900 – memory of a dream Mar 9 '16 at 12:15

7 @downvoters: According to Why is voting important? you should downvote if a `post contains wrong information, is poorly researched, or fails to communicate information` . Please explain for which one of these reasons you've downvoted this answer. – Marten Jun 7 '16 at 20:11

4 @Marten Two things. 1. You are never owed an explanation for downvotes, though I understand it can be helpful. 2. I did not downvote your answer, but I would guess that people don't like your answer because they think it is the wrong way to do this. That would qualify it as "Wrong information" since the

are quite unpredictable. There is no way to say if in 2025 "12:00 Moscow time" is still "9:00 UTC" like it is today, it has been changed a couple of times during the last 30 years. If you want to express a future local time you need true time zones. – Marten Aug 8 '17 at 7:45

protected by Machavity Feb 8 at 23:03

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?