

[Show pagesource](#)[Old revisions](#)[Recent changes](#)[Search](#)

Trace: » json_encoding_and_decoding

JSON encoding and decoding

The following built-in functions facilitate handling JSON data, as specified in [Douglas Crockford's RFC 4627](#).

[Ticket #212](#) proposes that we remove JSON support from ES4.

Encoding

■ `Object.prototype.toJSONString([pretty])`

Returns a String containing the JSON representation of an object. The `[[Prototype]]` internal property is not used when finding members. An object is serialized as a sequence of comma separated pairs wrapped in curly braces. During the course of the serialization, the `toJSONString` methods of the values will be used to obtain their JSON textual representations.

Values which are not directly represented in JSON (such as `undefined`, functions, and unknown types) will be skipped. The value `null` is serialized as the unquoted string `null`.

There needs to be a better spec for what should be skipped, as the only non-Object types in ES are `null` and `undefined`. I understand `function` could be in the skipped set but "unknown types" doesn't mean anything. It would be better to have an exhaustive list of the types that should be included, and why. For example, if I subclass `Object` for my "Point" class, should it be included, even if it has no interesting methods?

— [Lars T Hansen](#) 2007/09/24 21:14

We also need to have a notion of encodable and unencodable names. This came up before, and IIRC we decided that only "plain" names could be encoded (presumably, anything represented as string or uint, and a `Name` object if its namespace is empty/public but not otherwise).

— [Lars T Hansen](#) 2007/09/24 21:23

A new `EncodingError` instance will be thrown if `this` contains a cyclical structure. Recurring objects that do not cause cycles are allowed, but will produce a complete text for each occurrence.

If the optional `pretty` parameter is `true`, then linefeeds are inserted after each `{` and `,` and before `}`, and multiples of 4 spaces are inserted to indicate the level of nesting, and one space will be inserted after `::`. Otherwise, no whitespace is inserted between the tokens.

■ `Array.prototype.toJSONString([pretty])`

Returns a String containing the JSON representation of an array. An array is serialized as a sequence of comma separated values wrapped in square braces. During the course of the serialization, the `toString` methods of the values will be used to obtain their JSON textual representations.

Values which are not directly represented in JSON (such as `undefined`, functions, and unknown types) will be skipped. This means that holes in arrays, and array `length`, are not preserved. The value `null` is serialized as the unquoted string `null`.

A new `EncodingError` instance will be thrown if `this` contains a cyclical structure. Recurring objects that do not cause cycles are allowed, but will produce a complete text for each occurrence.

If the optional `pretty` parameter is `true`, then linefeeds are inserted after each `[` and `,` and before `]`, and multiples of 4 spaces are inserted to indicate the level of nesting. Otherwise, no whitespace is inserted between the tokens.

■ `Date.prototype.toJSONString()`

Returns a String containing the representation of a date. A `Date` object is serialized as an [ISO](#) date string in double quotes.

■ `String.prototype.toJSONString()`

Returns a String containing the JSON representation of a string. A `String` is serialized as a quoted string with backslash escapement.

■ `Number.prototype.toJSONString()`

Returns a String containing the JSON representation of a number. A finite number is serialized as an unquoted radix 10 string. Non-finite numbers are serializes as the unquoted string `null`.

■ `Boolean.prototype.toJSONString()`

Returns a String containing the JSON representation of a boolean value, an unquoted string `true` or `false`.

■ `EncodingError` is a new exception type. It contains a `name` property with value "EncodingError" and a `message` (an implementation-defined string).

Decoding

■ `String.prototype.parseJSON([filter])`

Returns the value represented by the string. A `syntaxError` is thrown if the string was not a strict JSON text.

The optional `filter` function will be called for each value found. It is passed each key and value. Its return value will be used instead of the original value in the final structure.

For example, this will transform all members whose keys contain the substring 'date' into `Date` objects.

```
myData = text.parseJSON(function (key, value) {
    return key.indexOf('date') >= 0 ? new Date(value) : value;
});
```

Is the filter called also for each array element (with the key being the array index), or just for fields of encoded objects?

—  [Lars T Hansen](#) 2007/09/27 00:52

Comments/Discussion

Both `toJSONString` methods should throw an error upon discovering a cycle or a join point, I think. Doug should correct me here if the join point check is too stringent. Cycle detection at a minimum is necessary to avoid recursive runaway DoS attacks.

—  [Brendan Eich](#) 2006/09/22 20:56

I suggest that both `toJSONString` methods should accept an optional filtering predicate that takes two parameters, an object and a property name, and return a boolean. If the predicate returns false (or perhaps, a false value) the property with that name will not be serialized for that object.

The use case for this is data structures that are deserialized, processed, and reserialized, but where the processing may add temporary data to the nodes that we do not want to be saved (and which may in fact not be valid JSON data).

—  [Lars T Hansen](#) 2006/09/26 03:24

I like Lars's suggestion for a filter function parameter for `toJSONString`. A simpler alternative would be an array of keynames.

Another useful option would be `prettyprint`, which would insert spaces and linebreaks. This is not necessary, but it would be friendly. How far off of minimal to we want to go?


`toJSONString` must look for cycles. The question is what it should do if it finds them. I see three reasonable alternatives:

1. When a recurring member is seen, then skip it, effectively deleting it from the serialization.
2. throw
3. return undefined

Do we want to choose one, or leave it to the implementation? The JSON standard is met by any of the three choices.

—  [Douglas Crockford](#) 2006/09/26 15:50

I took a shot at capturing this with a config object, which keeps the default function call simple (empty of parameters) but let's one specify more complex behaviour in a clear fashion. Is this interface too atypical though, I wonder?

—  [Iain Lamb](#) 2006/09/26 16:49

The array-of-names filter is probably adequate in practice.

—  [Lars T Hansen](#) 2006/09/27 03:04

Here's a slightly different proposal: get rid of `error` since it may be sufficient to use `filter`, get rid of the object argument and just use optional arguments. But I've also made the `filter` argument a little more complex. Here's a pseudo-type (since there's no way to specify optional arguments other than with rest-args):


```
toJSONString(/* optional */ pretty : Boolean, /* optional */ filter : ([String], function(String):Boolean, RegExp))
```

—  [Dave Herman](#) 2006/09/27 11:11


I think the array of names should be a white list, not black list. That makes it more useful in honoring protocol contracts.

```
toJSONString(/* optional */ filter : [String], /* optional */ pretty : Boolean)
```

—  [Douglas Crockford](#) 2006/10/02 11:13

In  [Mozilla bug #340987](#), Erik Arvidsson points out that this proposal needs `String.prototype.toJSONString` too.

—  [Brendan Eich](#) 2006/10/06 11:02

See  [Bob Ippolito's simplejson Python package](#), which Robert Sayre recommended to the `es4-discuss` list for its `object_hook` optional parameter. This hook, if provided, is called with the deserialized object (dict in Python terms), and whatever it returns replaces that object in its parent during deserialization. Iain, do I have that right? Comments?

—  [Brendan Eich](#) 2006/10/11 17:02

I updated to reflect what is currently offered in `json.js`. This is the API that people are actually using. The only complaint is that the methods are not DONTENUM.

—  [Douglas Crockford](#) 2006/11/01 08:23

The `filter` option in `parseJSON` is just a general purpose map function. Perhaps it would be more generally useful to pull it out of `parseJSON` and make it a general object method. So instead of

```
myObject = myString.parseJSON(f);
```

you would write

```
myObject = myString.parseJSON().filter(f);
```

The advantage is that `.filter` could be used for other purposes.

—  Douglas Crockford 2007/01/10 09:33

See [static generics](#) – `Array.filter` (available also on array instances via `Array.prototype.filter`) is proposed, after JS1.6 in Firefox, but it works only on indexed (non-negative property identifiers from 0 up to but not including the value of the instance's `length` property, and not including holes). It does not map over all, or all enumerable, properties of any object. Also, you can suppress elements from the instance by returning `false` from the filter function. This means that `Array.filter`, like `Array.map`, allocates a new array into which elements passing the filter are stored.

The advantage of a filter parameter `f` to `parseJSON` is that it avoids making the new array result that `Array.filter` creates. I think this is a compelling reason to keep the `filter` parameter. On the other hand, this proposal does not allow `f` to suppress a key/value pair, while `Array.filter` does. Unifying the two notions of filtering to allow suppression may be worthwhile. Do you know of real-world use cases for filtering “out” whole properties (key/value pairs), rather than transforming values as this proposal allows?

—  Brendan Eich 2007/01/11 15:02

The advantage of the pair is that transformations can be made based on the key. For example, if there is a convention that all date fields will include ‘date’ in their keys, then they can be easily identified and transformed.

```
myData = text.parseJSON(function (key, value) {  
  return key.indexOf('date') >= 0 ? new Date(value) : value;  
});
```

—  Douglas Crockford 2007/01/12 10:58

I see the benefit of separate `key` and `value` parameters to the callback, and didn't mean to question it – sorry for any confusion. As you noted earlier, the callback here is a `map` function, and it could be abstracted. It's not a filter in the Lisp or proposed ES4 `Array.filter` sense. But making it an optional callback parameter to `parseJSON` avoids an extra array or object allocation that would be imposed if we separated the transformation step: `text.parseJSON().map(function (key, value) {...})` would allocate twice. Sounds like we agree on avoiding that cost by keeping the callback parameter to `parseJSON`.

The question I asked above remains: should there be a way to suppress properties? Is that something real-world JSON decoders ever do?

—  Brendan Eich 2007/01/13 09:30

JSON does not encode cyclical structures, so implementations must take some action when presented with one. The `json.js` library simply loops until reaching stack or memory exhaustion. That is not an ideal coping strategy. It is highly recommended that implementations adopt more useful strategies, such as throwing an exception on noticing a cycle.

—  Douglas Crockford 2007/04/24 08:15

I just received a bug report:

```
var x = '{"__proto__": 3}'.parseJSON();
```

```
x.__proto__ === 3  
false
```

```
x.__proto__ === Object.prototype  
true
```

It seems that an implementation is allowed to add readonly properties to empty objects. We should have language in the standard to explicitly disallow this.

—  Douglas Crockford 2007/09/23 00:21

From ES3 Chapter 16: “An implementation may provide additional types, values, objects, properties, and functions beyond those described in this specification. This may cause constructs (such as looking up a variable in the global scope) to have implementation-defined behaviour instead of throwing an error (such as `ReferenceError`).”

The `__proto__` property in SpiderMonkey is not read-only – it can be written, but attempts to create cycles throw errors, and attempts to store primitive values are no-ops. [IIRC](#) other implements support read-only `__proto__`.

—  Brendan Eich 2007/09/24 01:06

Right. I am suggesting that that language in Chapter 16 be changed to disallow things like `__proto__`.

—  Douglas Crockford 2007/09/24 19:13

Could you make a specific proposal?

In this case I wonder who was bugged by `__proto__`. Was it a real-world bug, or a problem found by someone working on a fuzz-testcase generator or some such?

—  Brendan Eich 2007/09/24 20:42

proposals/json_encoding_and_decoding.txt · Last modified: 2007/09/27 00:54 by Iars

Show pagesource

Old revisions


Login

Index

Back to top

 [RSS](#)  [XML FEED](#)

 [LICENSED](#)

 [DONATE](#)

 [POWERED](#)

 [XHTML 1.0](#)

 [CSS](#)

 [DOUWIKI](#)