Asynchronous JSON parser implementation in Node.JS (in details)

By Nicu Micleușanu

Douglas Crockford

Douglas Crockford is known for:

- JSLint
- JSMin

- JSON

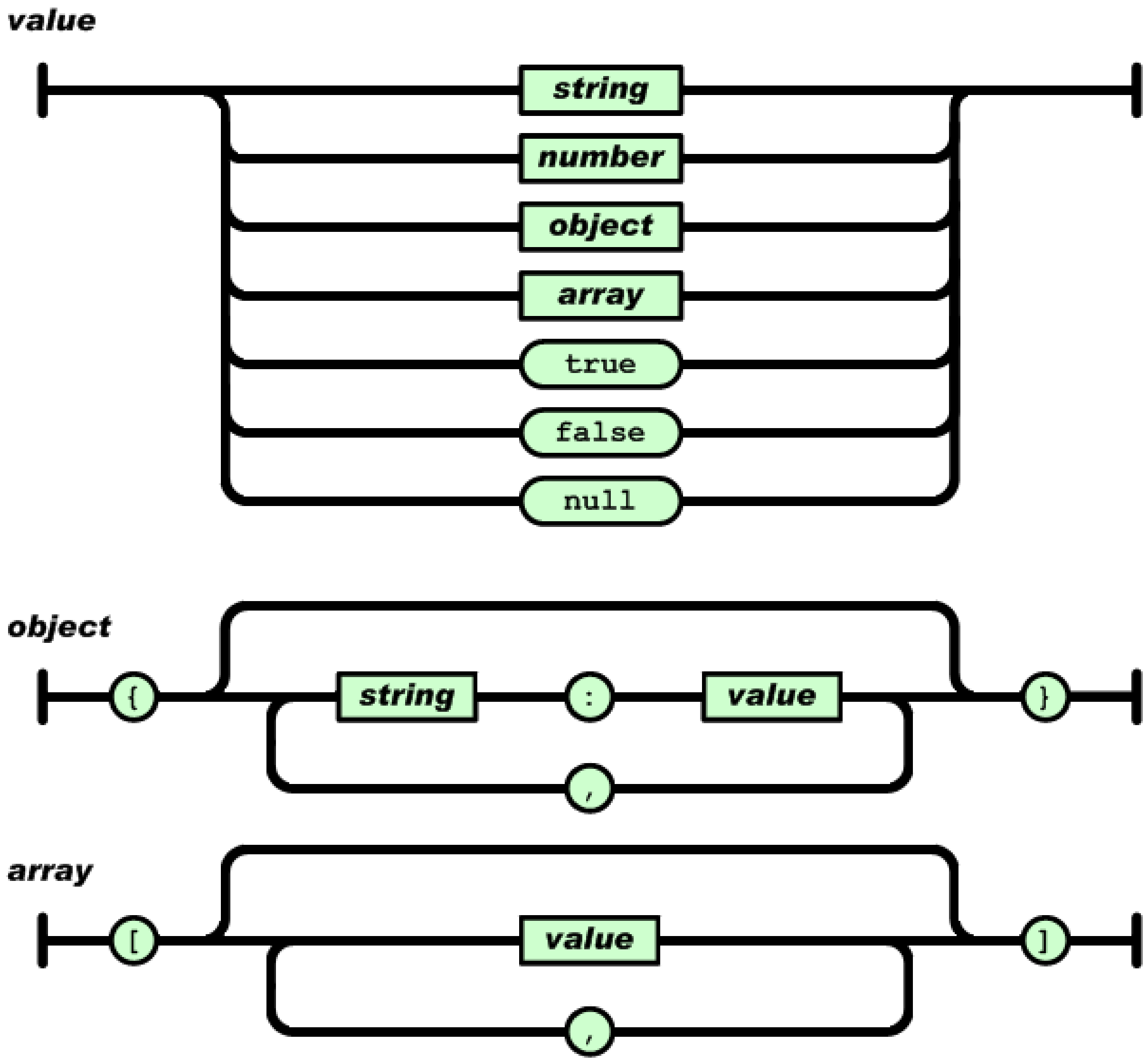
- TC39 member

- JavaScript: The Good Parts (book)

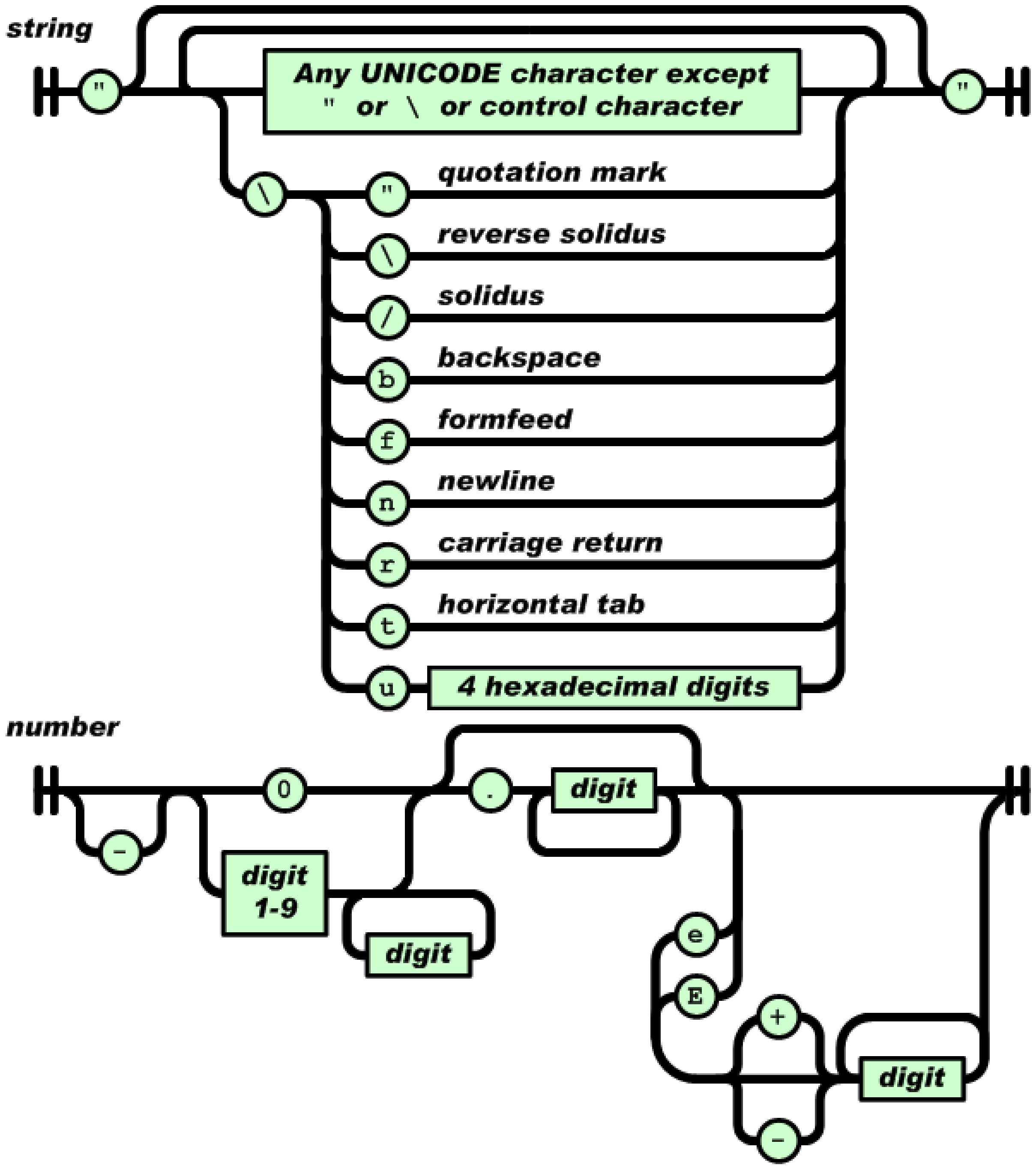
// TODO: add something more

JSON

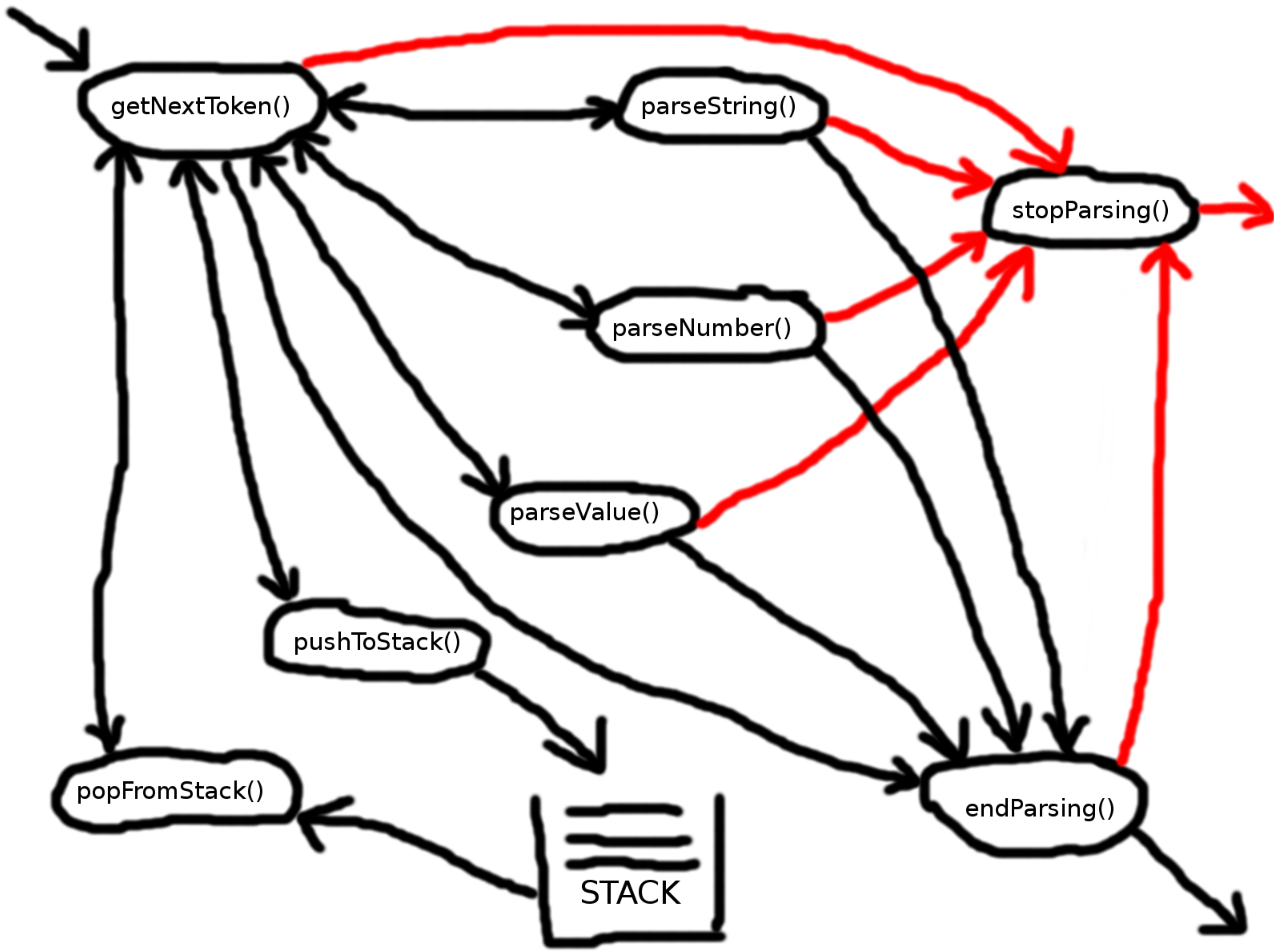
JavaScript Object Notation



JSON


string
"
Any UNICODE character except
" or \ or control character
"
\
"
quotation mark
\
reverse solidus
/
solidus
b
backspace
f
formfeed
n
newline
r
carriage return
t
horizontal tab
u
4 hexadecimal digits
number
-
0
digit
1-9
digit
.
digit
e
E
+
-
digit

Let’s write a JSON parser!



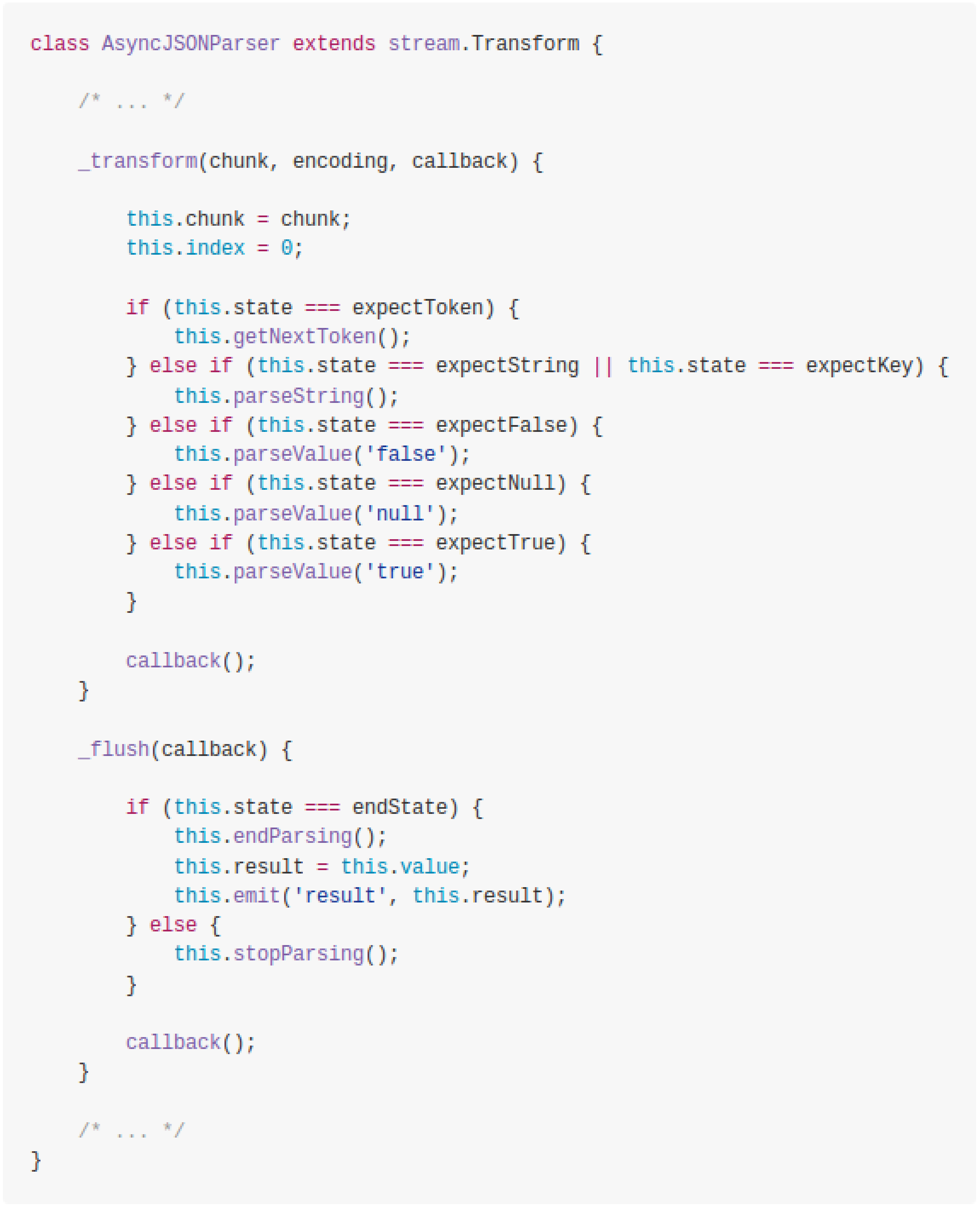
Parser setup

```
'use strict';

const stream = require('stream');

const errorState = Symbol('errorState');
const expectToken = Symbol('expectToken');
const expectString = Symbol('expectString');
const expectNumber = Symbol('expectNumber');
const expectFalse = Symbol('expectFalse');
const expectNull = Symbol('expectNull');
const expectTrue = Symbol('expectTrue');
const expectKey = Symbol('expectKey');
const endState = Symbol('endState');

const empty = Symbol('empty');

const toString = (value) => {

    return Object.prototype.toString.call(value);
};

const isArray = (value) => {

    return (toString(value) === '[object Array]');
};

const isObject = (value) => {

    return (toString(value) === '[object Object]');
};

class AsyncJSONParser extends stream.Transform { /* ... */ }

module.exports = AsyncJSONParser;
```

Parser constructor

```
class AsyncJSONParser extends stream.Transform {

    constructor() {

        super();

        this._writableState.objectMode = false;
        this._readableState.objectMode = true;

        this.chunk = null;
        this.container = null;
        this.index = 0;
        this.key = empty;
        this.result = null;
        this.stack = [];
        this.state = expectToken;
        this.value = empty;

        this.number = {
            digits: false,
            exponent: '',
            first: true,
            fraction: '',
            integer: '',
            point: false,
            power: false,
            sign: true
        };

        this.string = {
            escape: false,
            hex: '',
            unicode: false
        };
    }

    /* ... */
}
```

Stream Transform methods implementation



```
class AsyncJSONParser extends stream.Transform {

    /* ... */

    _transform(chunk, encoding, callback) {

        this.chunk = chunk;
        this.index = 0;

        if (this.state === expectToken) {
            this.getNextToken();
        } else if (this.state === expectString || this.state === expectKey) {
            this.parseString();
        } else if (this.state === expectFalse) {
            this.parseValue('false');
        } else if (this.state === expectNull) {
            this.parseValue('null');
        } else if (this.state === expectTrue) {
            this.parseValue('true');
        }

        callback();
    }

    _flush(callback) {

        if (this.state === endState) {
            this.endParsing();
            this.result = this.value;
            this.emit('result', this.result);
        } else {
            this.stopParsing();
        }

        callback();
    }

    /* ... */
}
```

Get tokens (strings)

```
class AsyncJSONParser extends stream.Transform {

    /* ... */

    getNextToken() {

        this.skipWhitespace();

        let current = this.chunk[this.index];

        // (") check for string value or object key
        if (current === 0x22) {

            if (this.value === empty) {

                if (isObject(this.container) && this.key === empty) {
                    this.state = expectKey;
                } else {
                    this.state = expectString;
                }

                this.index++;
                this.value = '';
                this.parseString();
            } else {
                this.stopParsing();
            }
        }

        /* ... */
    }

    /* ... */
}
```

Get tokens (false, null, true)

```
class AsyncJSONParser extends stream.Transform {

    /* ... */

    getNextToken() {

        /* ... */

        // (f) check for "false" value
        if (current === 0x66) {

            if (this.value === empty) {
                this.state = expectFalse;
                this.value = '';
                this.parseValue('false');
            } else {
                this.stopParsing();
            }

        // (n) check for "null" value
        } else if (current === 0x6E) {

            if (this.value === empty) {
                this.state = expectNull;
                this.value = '';
                this.parseValue('null');
            } else {
                this.stopParsing();
            }


        // (t) check for "true" value
        } else if (current === 0x74) {

            if (this.value === empty) {
                this.state = expectTrue;
                this.value = '';
                this.parseValue('true');
            } else {
                this.stopParsing();
            }
        }

        /* ... */
    }

    /* ... */
}
```

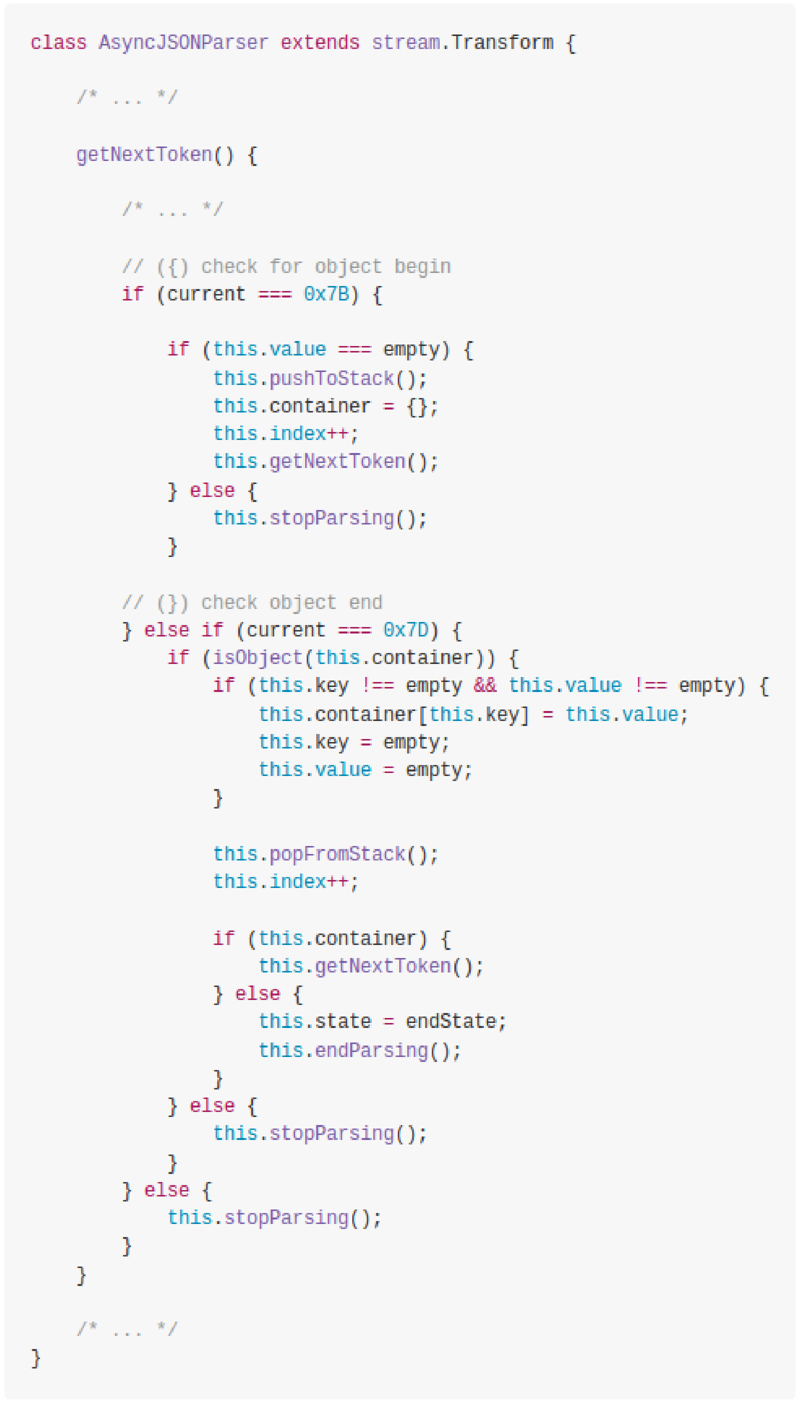
Get tokens (numbers, ‘,’, ‘:’)

```
class AsyncJSONParser extends stream.Transform {

    /* ... */

    getNextToken() {

        /* ... */

        // (-|0-9) check for number value
        if (current === 0x2D || (current > 0x2F && current < 0x3A)) {
            if (this.value === empty) {
                this.state = expectNumber;
                this.value = '';
                this.parseNumber();
            } else {
                this.stopParsing();
            }

        // (,) check for comma delimiter
        } else if (current === 0x2C) {
            if (this.container && this.value !== empty) {
                if (isArray(this.container)) {
                    this.container.push(this.value);
                    this.index++;
                    this.value = empty;
                    this.getNextToken();
                } else if (this.key === empty) {
                    this.stopParsing();
                } else {
                    this.container[this.key] = this.value;
                    this.key = empty;
                    this.value = empty;
                    this.getNextToken();
                }
            } else {
                this.stopParsing();
            }

        // (:) check for delimiter between object keys and values
        } else if (current === 0x3A) {
            if (isObject(this.container) && this.value !== empty) {
                this.key = this.value;
                this.value = empty;
                this.index++;
                this.getNextToken();
            } else {
                this.stopParsing();
            }
        }

        /* ... */
    }

    /* ... */
}
```

Get tokens (arrays)

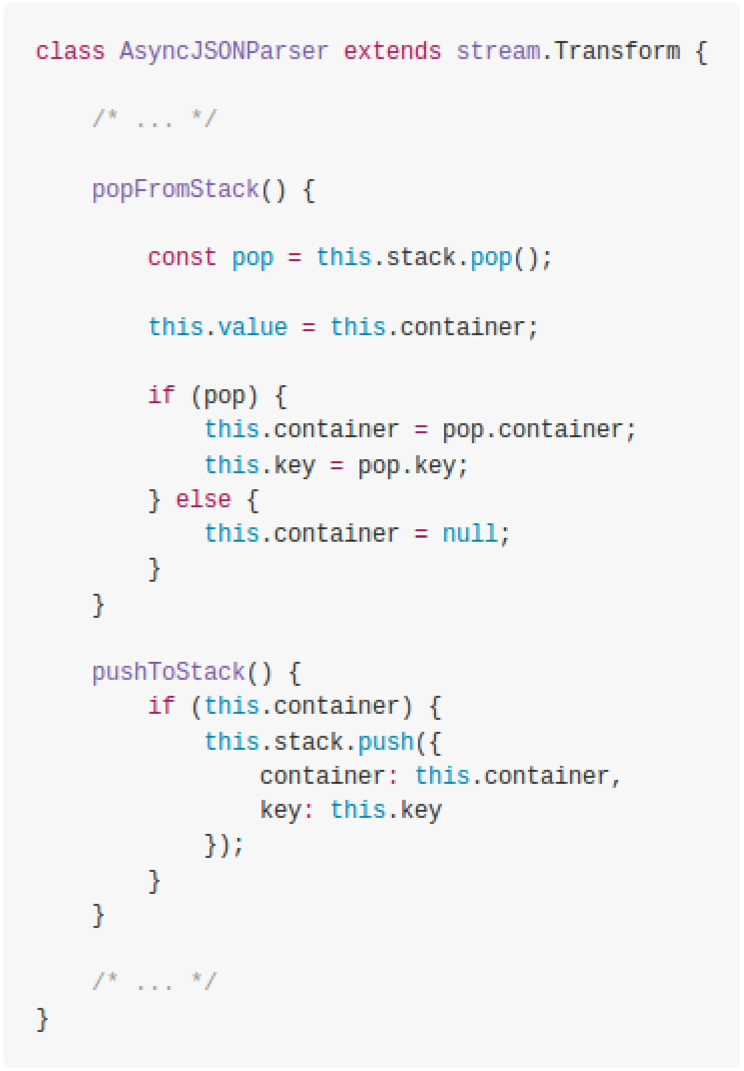
```
class AsyncJSONParser extends stream.Transform {

    /* ... */

    getNextToken() {

        /* ... */

        // ([) check for array begin
        if (current === 0x5B) {

            if (this.value === empty) {
                this.pushToStack();
                this.container = [];
                this.index++;
                this.getNextToken();
            } else {
                this.stopParsing();
            }

        // (]) check for array end
        } else if (current === 0x5D) {
            if (isArray(this.container)) {
                if (this.value !== empty) {
                    this.container.push(this.value);
                }

                this.popFromStack();
                this.index++;

                if (this.container) {
                    this.getNextToken();
                } else {
                    this.state = endState;
                    this.endParsing();
                }
            } else {
                this.stopParsing();
            }
        }

        /* ... */
    }

    /* ... */
}
```

Get tokens (objects)



```
class AsyncJSONParser extends stream.Transform {

    /* ... */

    getNextToken() {

        /* ... */

        // ({) check for object begin
        if (current === 0x7B) {

            if (this.value === empty) {
                this.pushToStack();
                this.container = {};
                this.index++;
                this.getNextToken();
            } else {
                this.stopParsing();
            }

        // (}) check object end
        } else if (current === 0x7D) {
            if (isObject(this.container)) {
                if (this.key !== empty && this.value !== empty) {
                    this.container[this.key] = this.value;
                    this.key = empty;
                    this.value = empty;
                }

                this.popFromStack();
                this.index++;

                if (this.container) {
                    this.getNextToken();
                } else {
                    this.state = endState;
                    this.endParsing();
                }
            } else {
                this.stopParsing();
            }
        } else {
            this.stopParsing();
        }
    }

    /* ... */
}
```

Stack operations (push, pop)



```
class AsyncJSONParser extends stream.Transform {

    /* ... */

    popFromStack() {

        const pop = this.stack.pop();

        this.value = this.container;

        if (pop) {
            this.container = pop.container;
            this.key = pop.key;
        } else {
            this.container = null;
        }
    }

    pushToStack() {
        if (this.container) {
            this.stack.push({
                container: this.container,
                key: this.key
            });
        }
    }

    /* ... */
}
```

Sync parsing

```
class AsyncJSONParser extends stream.Transform {

    /* ... */

    // Synchronous method for parsing, equivalent to JSON.parse()
    static parse(input) {

        const parser = new AsyncJSONParser();

        // Check for the type of the input to stringify it or not
        if (Buffer.isBuffer(input) || typeof input === 'string') {
            parser.end(input);
        } else {
            parser.end(String(input));
        }

        return parser.result;
    }

    /* ... */
}
```

The End

https://github.com/JSMD