JSON
JavaScript Object Notation

Ανάπτυξη Διαδικτυακών Συστημάτων & Εφαρμογών

Τμ. Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων

ΔιΠαΕ

Αντώνης Σιδηρόπουλος

Data Interchange

- The key idea in Ajax.
- An alternative to page replacement.
- Applications delivered as pages.
- How should the data be delivered?

History of Data Formats

- Ad Hoc
- Database Model
- Document Model
- Programming Language Model

JSON

- JavaScript Object Notation
- Minimal
- Textual
- Subset of JavaScript

JSON

- A Subset of ECMA-262 Third Edition.
- Language Independent.
- Text-based.
- Light-weight.
- Easy to parse.

JSON Is Not...

- JSON is not a document format.
- JSON is not a markup language.
- JSON is not a general serialization format.
  - No cyclical/recurring structures.
  - No invisible structures.
  - No functions.

History

- 1999 ECMAScript Third Edition
- 2001 State Software, Inc.
- 2002 JSON.org
- 2005 Ajax
- 2006 **RFC** 4627

Languages

- ActionScript
- C / C++
- C#
- Cold Fusion
- Delphi
- E
- Erlang
- Java
- Lisp
- Perl
- Objective-C
- Objective CAML
- PHP
- Python
- Rebol
- Ruby
- Scheme
- Squeak

Object Quasi-Literals

- JavaScript
- Python
- NewtonScript

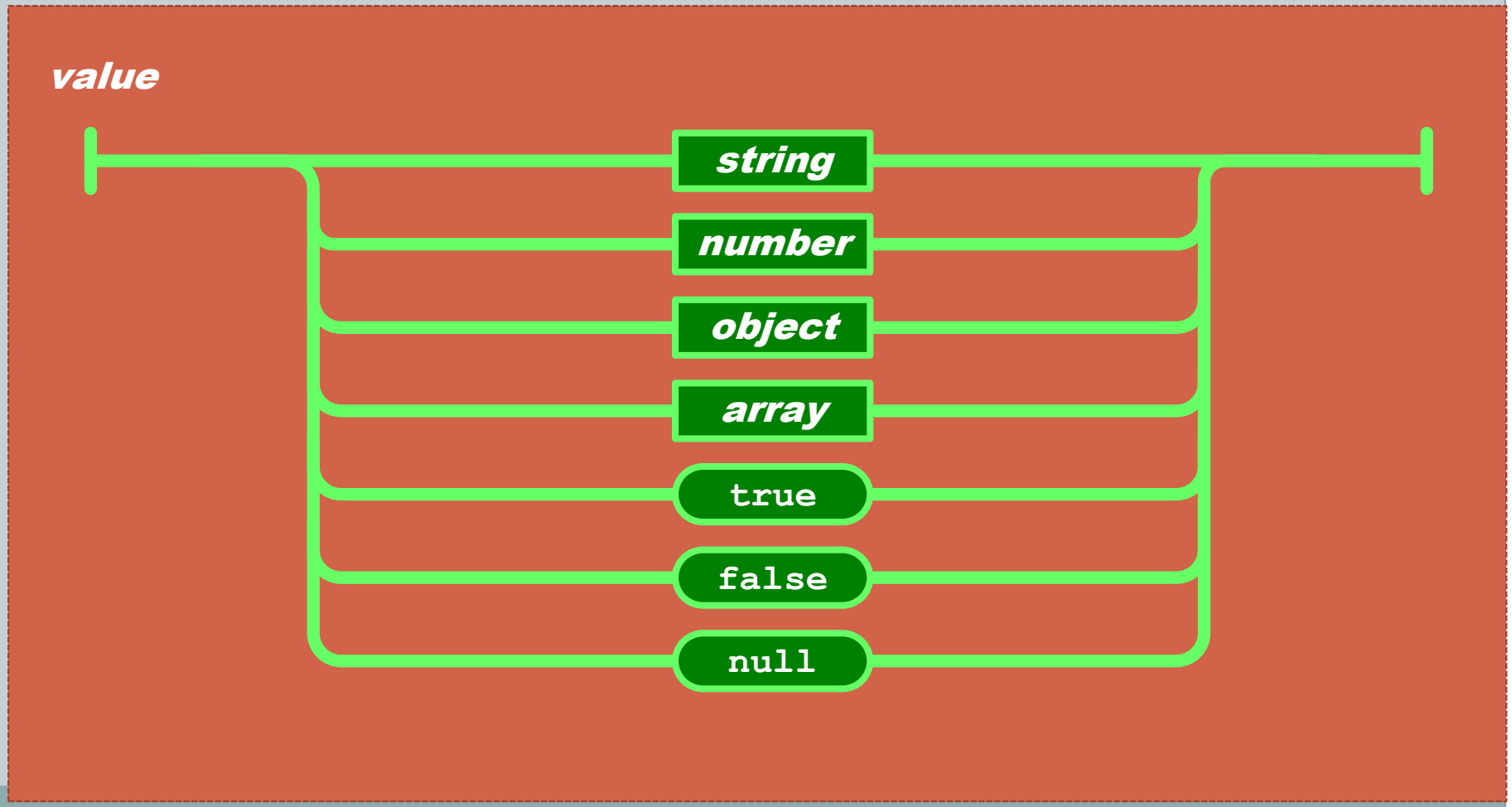
Values

- Strings
- Numbers
- Booleans

- Objects
- Arrays

- `null`

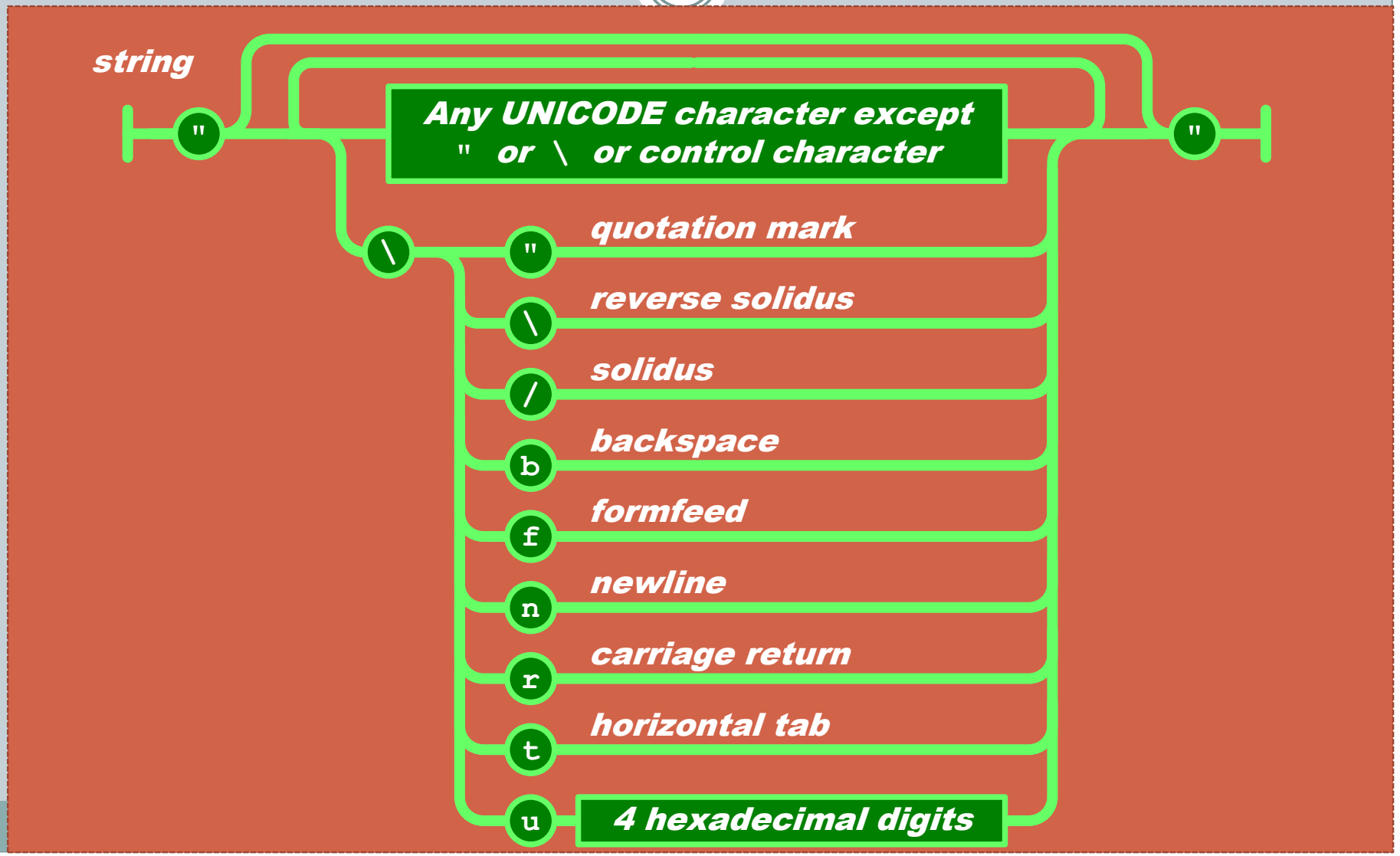
Value



Strings

- Sequence of 0 or more Unicode characters
- No separate character type
  - A character is represented as a string with a length of 1
- Wrapped in "double quotes"
- Backslash escapement

String

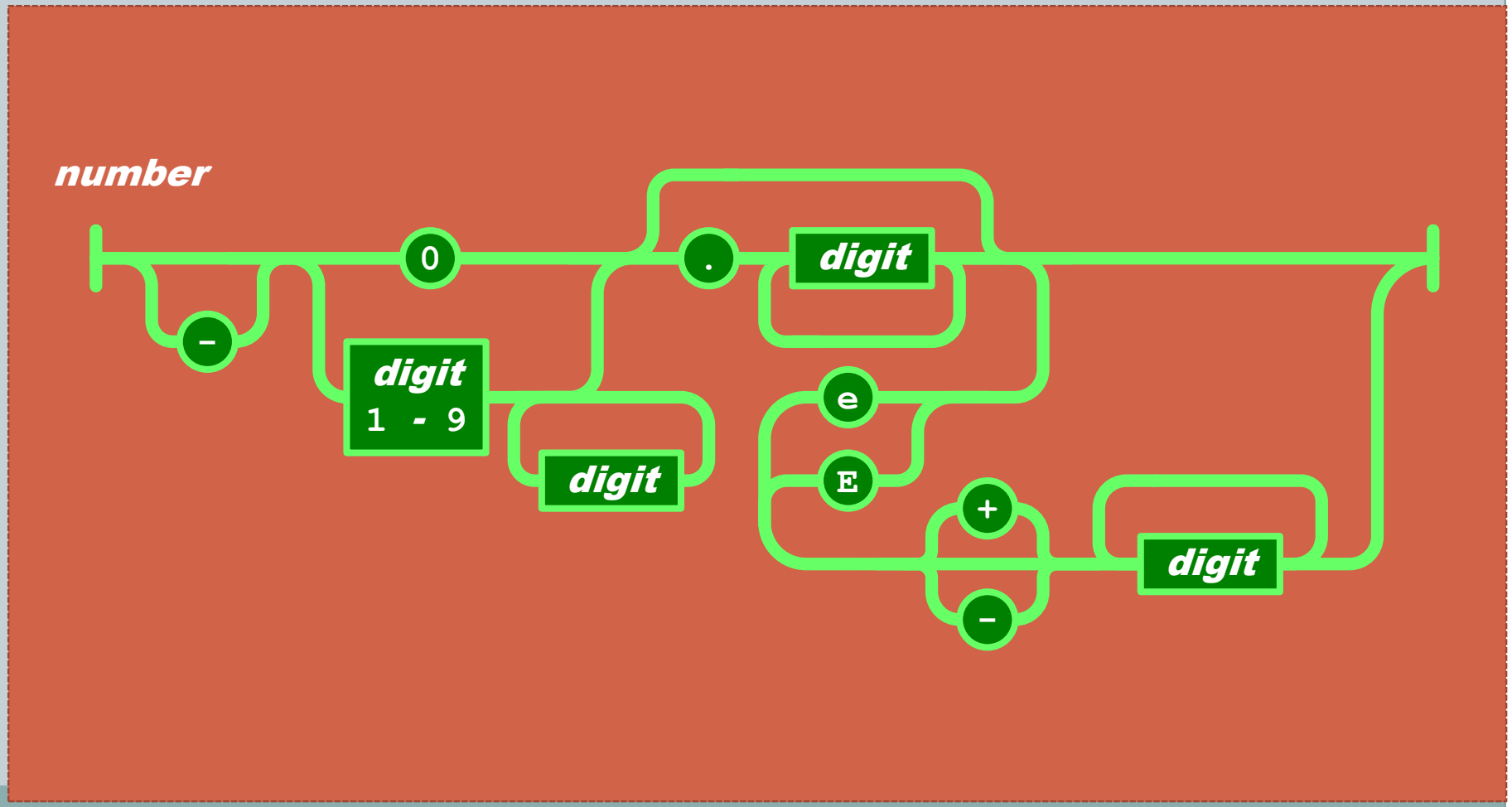

string
"
Any UNICODE character except " or \ or control character
\
"
quotation mark
\
reverse solidus
/
solidus
b
backspace
f
formfeed
n
newline
r
carriage return
t
horizontal tab
u
4 hexadecimal digits
"

Numbers

- Integer
- Real
- Scientific

- No octal or hex
- No **`NaN`** or **`Infinity`**
  - Use **`null`** instead

Number



Booleans

- `true`
- `false`

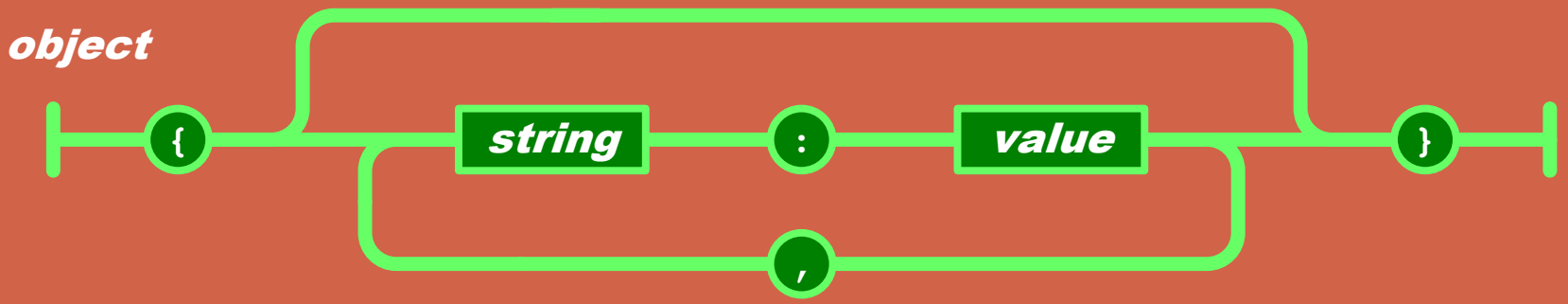
`null`

- A value that isn't anything

Object

- Objects are unordered containers of key/value pairs
- Objects are wrapped in { }
- , separates key/value pairs
- : separates keys and values
- Keys are strings
- Values are JSON values
  - struct, record, hashtable, object

Object



Object


```
{"name":"Jack B. Nimble","at large":
true,"grade":"A","level":3,
"format":{"type":"rect","width":1920,
"height":1080,"interlace":false,
"framerate":24}}
```

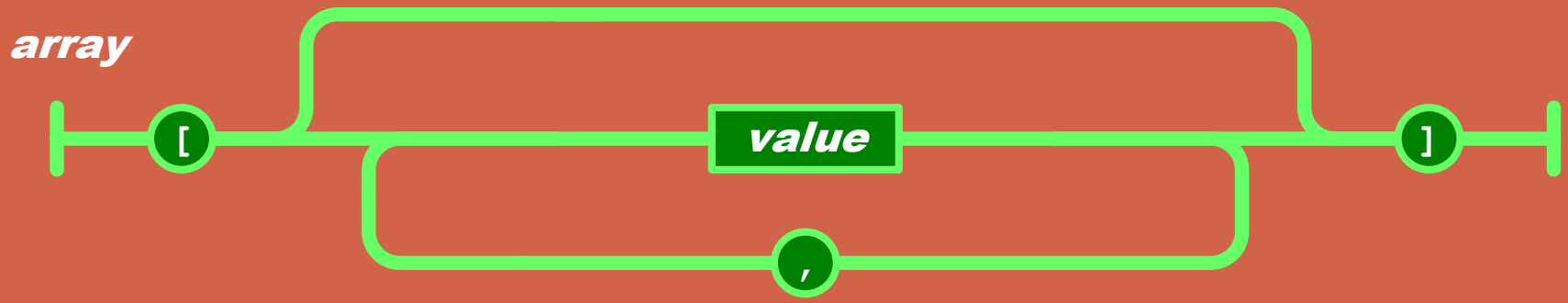
Object

```
{
    "name":        "Jack B. Nimble",
    "at large":    true,
    "grade":       "A",
    "format":      {
        "type":        "rect",
        "width":       1920,
        "height":      1080,
        "interlace":   false,
        "framerate":   24
    }
}
```

Array

- Arrays are ordered sequences of values
- Arrays are wrapped in `[]`
- `,` separates values
- JSON does not talk about indexing.
  - An implementation can start array indexing at 0 or 1.

Array



Array

Array with days:

```
["Sunday", "Monday", "Tuesday",
  "Wednesday", "Thursday", "Friday",
  "Saturday"]
```

Array of arrays:

```
[
  [0, -1, 0],
  [1, 0, 0],
  [0, 0, 1]
]
```

Array

Array of mixed:

```
[
  [0, -1, 0],
  "test",
  [0, 0, 1],
  {"name": “George”, “surname”: “Papadopoulos”},
  false
]
```

MIME Media Type

APPLICATION/JSON

Character Encoding

- Strictly UNICODE.
- Default: UTF-8.
- UTF-16 and UTF-32 are allowed.

Versionless

- JSON has no version number.
- No revisions to the JSON grammar are anticipated.
- JSON is very stable.

Rules

- A JSON decoder must accept all well-formed JSON text.
- A JSON decoder may also accept non-JSON text.
- A JSON encoder must only produce well-formed JSON text.
- *Be conservative in what you do, be liberal in what you accept from others.*

Supersets

- YAML is a superset of JSON.
  - A YAML decoder is a JSON decoder.
- JavaScript is a superset of JSON.
  - A JavaScript compiler is a JSON decoder.
- New programming languages based on JSON.

JSON is the X in Ajax

JSON in Ajax

- HTML Delivery.

- JSON data is built into the page.

```
<html>...
<script>
var data = { ... JSONdata ... };
</script>...
</html>
```

JSON in Ajax

- XMLHttpRequest
  - Obtain `responseText`
  - Parse the `responseText`

```
responseData = eval( '(' + responseText + ')');

responseData = responseText.parseJSON();
```

JSON in Ajax

- Is it safe to use **`eval`** with XMLHttpRequest?
- The JSON data comes from the same server that vended the page. **`eval`** of the data is no less secure than the original html.
- If in doubt, use ***`string`*`.parseJSON`** instead of **`eval`**.

Some features that make *it* well-suited for data transfer

- It's simultaneously human- and machine-readable format;
- It has support for Unicode, allowing almost any information in any human language to be communicated;
- The self-documenting format that describes structure and field names as well as specific values;
- The strict syntax and parsing requirements that allow the necessary parsing algorithms to remain simple, efficient, and consistent;
- The ability to represent the most general computer science data structures: records, lists and trees.

JSON Looks Like Data

- JSON's simple values are the same as used in programming languages.
- No restructuring is required: JSON's structures look like conventional programming language structures.
- JSON's object is record, struct, object, dictionary, hash, associate array…
- JSON's array is array, vector, sequence, list…

Arguments against JSON

- JSON Doesn't Have Namespaces.
- JSON Has No Validator.
- JSON Is Not Extensible.
- JSON Is Not XML.

JSON Doesn't Have Namespaces

- Every object is a namespace. Its set of keys is independent of all other objects, even exclusive of nesting.
- JSON uses context to avoid ambiguity, just as programming languages do.

Namespace

- **http://www.w3c.org/TR/REC-xml-names/**

- In this example, there are three occurrences of the name title within the markup, and the name alone clearly provides insufficient information to allow correct processing by a software module.

```
<section>
    <title>Book-Signing Event</title>
    <signing>
        <author title="Mr" name="Vikram Seth" />
        <book title="A Suitable Boy" price="$22.95" />
    </signing>
    <signing>
        <author title="Dr" name="Oliver Sacks" />
        <book title="The Island of the Color-Blind"
              price="$12.95" />
    </signing>
</section>
```

Namespace

```
{"section": {
    "title": "Book-Signing Event",
    "signing": [
        {
            "author": { "title": "Mr", "name": "Vikram Seth" },
            "book": { "title": "A Suitable Boy",
                      "price": "$22.95" }
        }, {
            "author": { "title": "Dr", "name": "Oliver Sacks" },
            "book": { "title": "The Island of the Color-Blind",
                      "price": "$12.95" }
        }
    ]
}}
```

- `section.title`
- `section.signing[0].author.title`
- `section.signing[1].book.title`

No need to parse !!

```
var o = {"section": {
    "title": "Book-Signing Event",
    "signing": [
        {
            "author": { "title": "Mr", "name": "Vikram Seth" },
            "book": { "title": "A Suitable Boy",
                      "price": "$22.95" }
        }, {
            "author": { "title": "Dr", "name": "Oliver Sacks" },
            "book": { "title": "The Island of the Color-Blind",
                      "price": "$12.95" }
        }
    ]
}};
```

- `o.section.title`
- `o.section.signing[0].author.title`
- `o.section.signing[1].book.title`

JSON Has No Validator

- Being well-formed and valid is not the same as being correct and relevant.
- Ultimately, every application is responsible for validating its inputs. This cannot be delegated.
- A YAML validator can be used.

JSON is Not Extensible

- It does not need to be.
- It can represent any non-recurrent data structure as is.
- JSON is flexible. New fields can be added to existing structures without obsoleting existing programs.

Data Interchange

- JSON is a simple, common representation of data.
- Communication between servers and browser clients.
- Communication between peers.
- Language independent data interchange.

show_allusers_json.php

```php
<?php
require "../internal/dbconnect.php";
$stmt = $mysqli->prepare('SELECT * FROM customer');
$stmt->execute();
$res = $stmt->get_result();
$r = $res->fetch_all(MYSQLI_ASSOC);
print json_encode($r, JSON_PRETTY_PRINT);
?>
```

JSON vs. XML

```
[
    {
        "ID": 11,
        "Fname": "Nikos",
        "Lname": "Papadopoulos",
        "Address": "Egnatia 130",
        "Phone": "302310222222",
        "uname": "nikos",
        "passwd_enc": "*00A51F3F48415C7D4E8908980D443C29C69B60C9",
        "is_admin": 1
    },
    {
        "ID": 10,
        "Fname": "Kostas",
        "Lname": "Chatzigeorgiou",
        "Address": "Ag. Dimitriou 150",
        "Phone": "302310111111",
        "uname": "kostas",
        "passwd_enc": "*00A51F3F48415C7D4E8908980D443C29C69B60C9",
        "is_admin": 0
    }
]
```

show_allusers_xml.php

```
<?php
require "../internal/dbconnect.php";
$stmt = $mysqli->prepare('SELECT * FROM customer');
$stmt->execute();
$res = $stmt->get_result();
$r = $res->fetch_all(MYSQLI_ASSOC);
print xmlrpc_encode($r);
?>
```

JSON vs. XML

```
<?xml version="1.0" encoding="utf-8"?>
<params>
<param>
 <value>
  <array>
   <data>
    <value>
     <struct>
      <member>
       <name>ID</name>
       <value>
        <int>11</int>
       </value>
      </member>
      <member>
       <name>Fname</name>
       <value>
        <string>Nikos</string>
       </value>
      </member>
      <member>
       <name>Lname</name>
```

www.JSON.org