

JSON

JavaScript Object Notation

JSON

From Wikipedia, the free encyclopedia

For similarly named people, see [J Son \(disambiguation\)](#).

JSON, (canonically pronounced /ˈdʒeɪsən/ *JAY-sən*^[1] sometimes **JavaScript Object Notation**), is an [open standard](#) format that uses [human-readable](#) text to transmit data objects consisting of [attribute–value pairs](#). It is the primary data format used for asynchronous browser/server communication ([AJAJ](#)), largely replacing [XML](#) (used by [AJAX](#)).

Although originally derived from the [JavaScript](#) scripting language, JSON is a [language-independent](#) data format. Code for [parsing](#) and generating JSON data is readily available in many programming languages.

<http://www.jsoneditoronline.org/>

<http://json.org/>

The screenshot shows a web browser window with the address bar displaying `godoc.org/encoding/json`. The browser's toolbar includes various icons for social media and utilities. The page header features the **GoDoc** logo and navigation links for [Home](#), [Index](#), and [About](#), along with a search bar. Below the header, a breadcrumb trail shows [Go: encoding/json](#) and links to [Index](#), [Examples](#), and [Files](#). The main content area is titled **package json** and contains the following text:

```
import "encoding/json"
```

Package json implements encoding and decoding of JSON objects as defined in [RFC 4627](#). The mapping between JSON objects and Go values is described in the documentation for the `Marshal` and `Unmarshal` functions.

See "JSON and Go" for an introduction to this package: https://golang.org/doc/articles/json_and_go.html

Encoder, Decoder
Marshal, Unmarshal

to JSON

from JSON

Encoder, Decoder
Marshal, Unmarshal

to JSON from JSON

Encoder, Decoder reader, writer

Marshal, Unmarshal []bytes

godoc.org/encoding/json

marks



func Marshal

```
func Marshal(v interface{}) ([]byte, error)
```

Marshal returns the JSON encoding of v.

godoc.org/encoding/json

marks M En O T G A 24 f Y T CNN reddit d Digg S PM Hawk J Android GO

func Unmarshal

```
func Unmarshal(data []byte, v interface{}) error
```

Unmarshal parses the JSON-encoded data and stores the result in the value pointed to by v.

godoc.org/encoding/json#Encoder

marks

type Encoder

```
type Encoder struct {  
    // contains filtered or unexported fields  
}
```

An Encoder writes JSON objects to an output stream.

godoc.org/encoding/json#Decoder

type Decoder

```
type Decoder struct {  
    // contains filtered or unexported fields  
}
```

A Decoder reads and decodes JSON objects from an input stream.

```
1 package main
2
3 import (
4     "encoding/json"
5     "fmt"
6 )
7
8 func main() {
9     jsonData := `
10     {
11     "name": "Todd McLeod"
12     }
13     `
14
15     var obj map[string]string
16
17     err := json.Unmarshal([]byte(jsonData), &obj)
18     if err != nil {
19         panic(err)
20     }
21     fmt.Println(obj)
22 }
23
```

Unmarshal

```
01 $ go run main.go
map[name:Todd McLeod]
01 $ _
```

```
1 package main
2
3 import (
4     "encoding/json"
5     "fmt"
6 )
7
8 func main() {
9     jsonData := `
10     {
11         "name": "Todd McLeod",
12         "age": 44
13     }
14 `
15
16     var obj map[string]interface{}
17
18     err := json.Unmarshal([]byte(jsonData), &obj)
19     if err != nil {
20         panic(err)
21     }
22     fmt.Println(obj)
23 }
24
```

Unmarshal

```
02 $ go run main.go
map[name:Todd McLeod age:44]
02 $ _
```



main.go x

```
1 package main
2
3 import (
4     "encoding/json"
5     "fmt"
6 )
7
8 func main() {
9     jsonData := `
10     [100, 200, 300.5, 400.1234]
11     `
12
13     var obj []float64
14
15     err := json.Unmarshal([]byte(jsonData), &obj)
16     if err != nil {
17         panic(err)
18     }
19     fmt.Println(obj)
20 }
21
```

Unmarshal

```
04 $ go run main.go
[100 200 300.5 400.1234]
04 $ _
```

```
1 package main
2
3 import (
4     "encoding/json"
5     "fmt"
6 )
7
8 func main() {
9     jsonData := `
10     100
11     `
12
13     var obj interface{}
14
15     err := json.Unmarshal([]byte(jsonData), &obj)
16     if err != nil {
17         panic(err)
18     }
19     fmt.Println(obj)
20
21     fmt.Printf("%T\n", obj)
22 }
```

Unmarshal

```
05 $ go run main.go
100
float64
05 $ _
```



```
1 package main
2 |
3 import (
4     "encoding/json"
5     "fmt"
6     "os"
7 )
8
9 type StockData struct {
10     Returns []float64 `json:"returns"`
11 }
12
13 func main() {
14     f, err := os.Open("data.json")
15     if err != nil {
16         panic(err)
17     }
18     defer f.Close()
19
20     var obj StockData
21     err = json.NewDecoder(f).Decode(&obj)
22     if err != nil {
23         panic(err)
24     }
25     fmt.Println(obj)
26 }
27
```

Unmarshal

```
14_struct $ go run main.go
{[1 2 3 4]}
14_struct $ _
```

Example

Code:

play

```
var jsonBlob = []byte(`[
    {"Name": "Platypus", "Order": "Monotremata"},
    {"Name": "Quoll",    "Order": "Dasyuromorphia"}
]`)

type Animal struct {
    Name string
    Order string
}

var animals []Animal
err := json.Unmarshal(jsonBlob, &animals)
if err != nil {
    fmt.Println("error:", err)
}
fmt.Printf("%+v", animals)
```

Unmarshal

Output:

```
[{Name:Platypus Order:Monotremata} {Name:Quoll Order:Dasyuromorphia}]
```

Example

Code:

play

```
type ColorGroup struct {
    ID      int
    Name     string
    Colors []string
}

group := ColorGroup{
    ID:      1,
    Name:    "Reds",
    Colors: []string{"Crimson", "Red", "Ruby", "Maroon"},
}

b, err := json.Marshal(group)
if err != nil {
    fmt.Println("error:", err)
}

os.Stdout.Write(b)
```

Marshal

Output:

```
{"ID":1,"Name":"Reds","Colors":["Crimson","Red","Ruby","Maroon"]}
```

[Example](#)

This example uses a Decoder to decode a stream of distinct JSON values.

Code:

[play](#)

```
const jsonStream = `
    {"Name": "Ed", "Text": "Knock knock."}
    {"Name": "Sam", "Text": "Who's there?"}
    {"Name": "Ed", "Text": "Go fmt."}
    {"Name": "Sam", "Text": "Go fmt who?"}
    {"Name": "Ed", "Text": "Go fmt yourself!"}
`

type Message struct {
    Name, Text string
}

dec := json.NewDecoder(strings.NewReader(jsonStream))
for {
    var m Message
    if err := dec.Decode(&m); err == io.EOF {
        break
    } else if err != nil {
        log.Fatal(err)
    }
    fmt.Printf("%s: %s\n", m.Name, m.Text)
}
```

Decoder

Output:

```
Ed: Knock knock.
Sam: Who's there?
Ed: Go fmt.
Sam: Go fmt who?
Ed: Go fmt yourself!
```

exercises

JSON

Create a struct, encode it as JSON, and write it out to stdout.

JSON

- Create a JSON file
- Create a program that reads that JSON file into a struct, then writes it to stdout.

JSON

Create a program which can read a csv file and write it as a JSON file.