

# Basic types & data formats

## Basic types

JSON supports four simple data types: string, number, integer, boolean; which are all represented in JSON schema.

Say we were expecting a JSON structure containing all three, i.e. something like the following:

```
{
  myString: "lorem ipsum",
  myNumber: 1234.56,
  myInteger: 123456,
  myBoolean: true
}
```

This would be represented by the schema

```
{
  type: "object",
  properties: {
    myString: {
      type: "string"
    },
    myNumber: {
      type: "number"
    },
    myInteger: {
      type: "integer"
    },
    myBoolean: {
      type: "boolean"
    }
  }
}
```

Additionally, JSON supports two complex types, 'object' and 'array'. We'll cover those types later.

In all cases where we're dealing with a basic type, we will supply a title within the specification . This is intended to be used as the label for the control.

```
{
  type: "string",
  title: "Property title"
}
```

Property title

## Formats

An optional 'format' property can be included to provide information about the format of the expected data. Using the type and format properties, we can determine an appropriate form control to render:

Type	Control
string	a text input
number	a numeric input
integer	a numeric input without decimals
boolean	a checkbox or toggle

The client implementation can decide on the best component to use. For example, in iOS, booleans are normally toggles, whereas on websites they're typically checkboxes.

Example:

String label

```
<tw-field
  name="stringProperty"
  model="Example"
  field="
    {
      'type': 'string',
      'title': 'String label',
      'placeholder': 'Please enter string'
    }">
</tw-field>
```

Number control

```
<tw-field
  name="numberProperty"
  model="123"
  field="
    {
      'type': 'number',
      'title': 'Number control',
      'placeholder': 'Please enter number'
    }">
</tw-field>
```

Boolean control

☐ Please choose

```
<tw-field
  name="booleanProperty"
  model="false"
  field="
    {
      'type': 'boolean',
      'title': 'Boolean control',
      'placeholder': 'Please choose'
    }">
</tw-field>
```

The format property is optional, it can contain a range of values. For some of these you may just render a standard text input, but you can optionally customise your presentation further, or use this information to trigger alternative keyboard layouts on mobile devices. At minimum you should support date and file controls as users will not be able to enter these manually.

Type	Format	Description
string	n/a	a text input
string	number	a text input with numeric keyboard
string	date	a date input returning an iso8601 date string e.g. 2000-01-01
string	date-time	a date and time selector returning an iso8601 date string <b>(unused)</b>
string	phone	a input for a telephone number
string	uri	a text input with custom keyboard or uri input
string	email	a text input custom keyboard or email input
string	base64url	a file upload that converts files to a base64 encoded string
number	n/a	a number input
integer	n/a	a number input that only allows integers <b>(unused)</b>
boolean	n/a	a checkbox

Example:

#### Date control



  

```
<tw-field
  name="dateProperty"
  model="2017-12-01"
  field="
    {
      "type": "string",
      "format": "date",
      "title": "Date control"
    }">
</tw-field>
```

#### Phone control

```
<tw-field
  name="phoneProperty"
  model=""
  field="
    {
      "type": "string",
      "format": "phone",
      "title": "Phone control"
    }">
</tw-field>
```

  
**Upload control**  
Choose file...  


```
<tw-field
  name="base64urlProperty"
  model=""
  field="
    {
      "type": "string",
      "format": "base64url",
      "title": "Upload control",
      "placeholder": "Choose file..."
    }">
</tw-field>
```

The list of formats may be extended in future, but with the exception of the above it will always be ok to fall back to the default control dictated by the type.

For example, if in future we added a new field with {type: "string", format: "credit-card"}, we may use that information to supply a special treatment on that control, but rendering the default text control will still provide a perfectly adequate fallback. New types will not be introduced.

### Enumerated values

Additionally, a property may have a limited range of possible values. In JSON schema this is represented by the 'enum' property, an array of the allowed values. Unfortunately this is too limited for our user interfaces so we include a 'values' property with richer objects:

```
{
  type: "number",
  enum: [1, 2]
  values: [
    {value: 1, label: "One"},
    {value: 2, label: "Two"}
  ]
}
```

The client may decide to render different controls based on how many values are possible. For example:

- 1 => User has no option so simply set the value and do not render a control
- 2 to 4 => radio buttons or selectable list
- 4 to 12 => a drop down selection box
- 13 or more => a searchable combo box

Example:

### Select control

One

```
<tw-field
  name="selectProperty"
  model="1"
  field="
  {
    "type": "number",
    "title": "Select control",
    "placeholder": "Please choose",
    "values": [
      {
        "value": 1,
        "label": "One"
      },
      {
        "value": 2,
        "label": "Two"
      },
      {
        "value": 3,
        "label": "Three"
      },
      {
        "value": 4,
        "label": "Four"
      }
    ]
  }
  ">
</tw-field>
```

### Radio control

☐ One

☒ Two

```
<tw-field
  name="radioProperty"
  model="2"
  field="
  {
    "type": "string",
    "title": "Radio control",
    "values": [
      {
        "value": "1",
        "label": "One"
      },
      {
        "value": "2",
        "label": "Two"
      }
    ]
  }
  ">
</tw-field>
```

However, we may also provide a suggested control type.

### Control type

In most cases, the above works to determine the control type used. However, we may also supply an additional 'control' property. This is not part of JSON schema, it is safe to ignore and simply rely on type and format, but supporting these additional options it will help to craft a high quality user experience.


Type	Control	Description
string	password	the data is a password, obfuscate the characters the user has entered
string	textarea	the expected data is likely to be a longer piece of text with line breaks, render a multi-line text control.

We may also supply one of the following control types along with enumerated values to help you decide on the best selection type:

Type	Values	Control	Description
string	...	select	render a select control regardless of how many values there are
string	...	radio	render a set of radio buttons regardless of how many values there are

Example:

#### Select override

One 


```
<tw-field
  name="selectProperty"
  model="1"
  field="
    {
      "type": "string",
      "title": "Select override",
      "control": "select",
      "values": [
        {
          "value": "1",
          "label": "One"
        },
        {
          "value": "2",
          "label": "Two"
        }
      ]
    }
  ">
</tw-field>
```

#### Password override

.....

```
<tw-field
  name="passwordProperty"
  model="qwerty"
  field="
    {
      "type": "string",
      "control": "password",
      "title": "Password override",
      "placeholder": "Choose password..."
    }
  ">
</tw-field>
```

#### Textarea override

Lorem Ipsum
 

```
<tw-field
  name="textareaProperty"
  model="Lorem Ipsum"
  field="
    {
      "type": "string",
      "control": "textarea",
      "title": "Textarea override"
    }
  ">
</tw-field>
```

## Enumerations & presentation

In some cases we supply additional presentational information along with the 'values' array, to assist in creating the best possible user experience. All of these pieces of additional information are strictly optional, it will always be possible to complete the form without access to them. The values that can appear are:

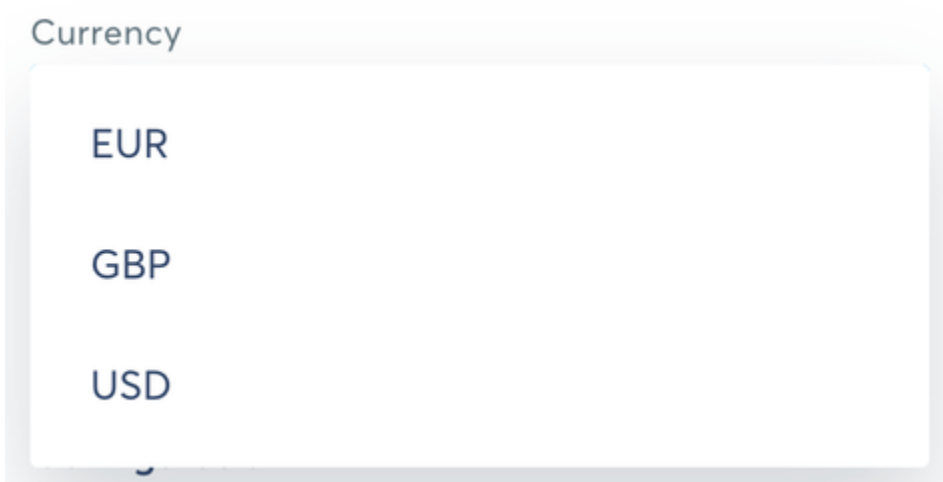
- **imgUrl** - the path to an image to show alongside the option, example: user avatar
- **note** - shorter supplemental text, typically shown after the label
- **secondary** - longer secondary text, typically shown as a second line under the option
- **currency** - meta data for when we're dealing with currencies. Typically used to show a flag associated with that currency
- **country** - meta-data for when we're dealing with countries. Typically used to show the flag of that country.

- **searchable** - additional string of searchable text that is not shown to the user. Example: when selecting a currency, we may include the names of the countries that use the currencies as searchable text.

For example, say we had a schema which required a property 'currency', this could be specified with the following additional information:

```
currency: {
  type: "string"
  title: "Currency",
  values: [{
    value: "GBP",
    label: "GBP",
    currency: "GBP",
    note: "Great British Pound",
    searchable: "England, Wales, Scotland, Northern Ireland"
  }, {
    value: "EUR",
    label: "EUR",
    currency: "EUR",
    note: "Euro",
    searchable: "France, Germany, Estonia"
  }, {
    value: "USD",
    label: "USD",
    currency: "USD",
    note: "United States Dollar",
    searchable: "United States"
  }]
}
```

A basic implementation would render a simple control utilising just the label and value.



However, a more sophisticated control could enhance the experience with the additional supplied information:



## Currency



EUR Euro



GBP Great British Pound



USD United States Dollar