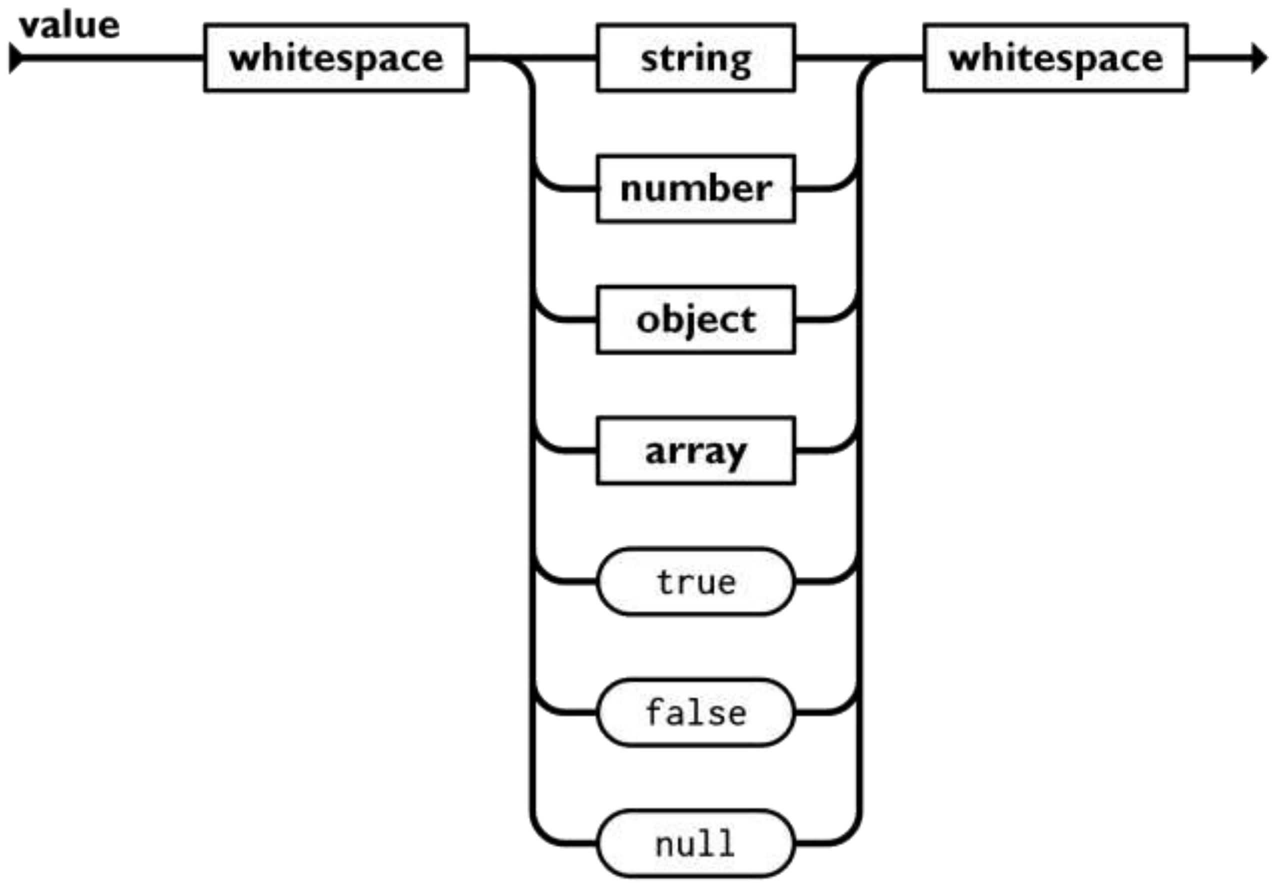
JSON Type serialization registry

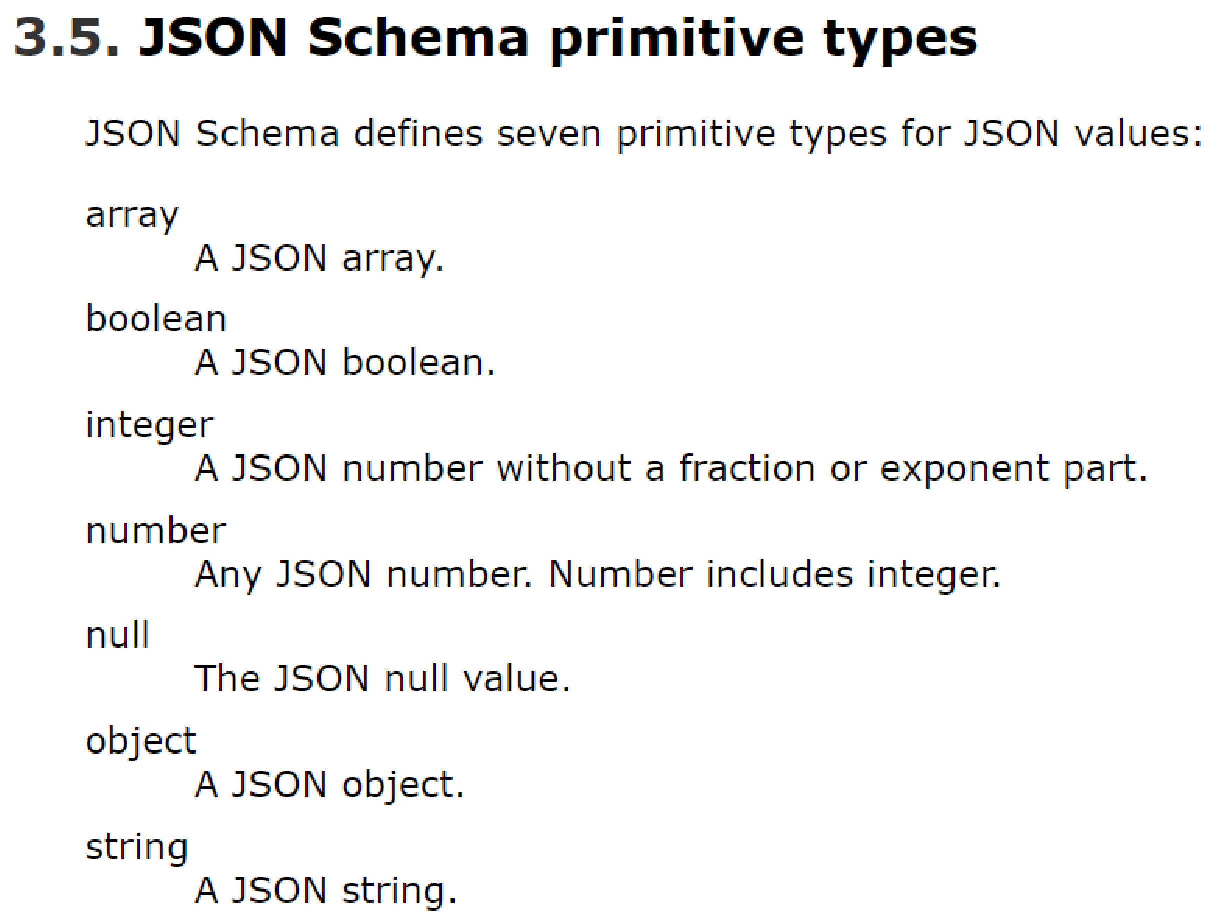
Darrel Miller

JSON is the primary format used by APIs…



But has a very limited type system

JSON Schema added a little

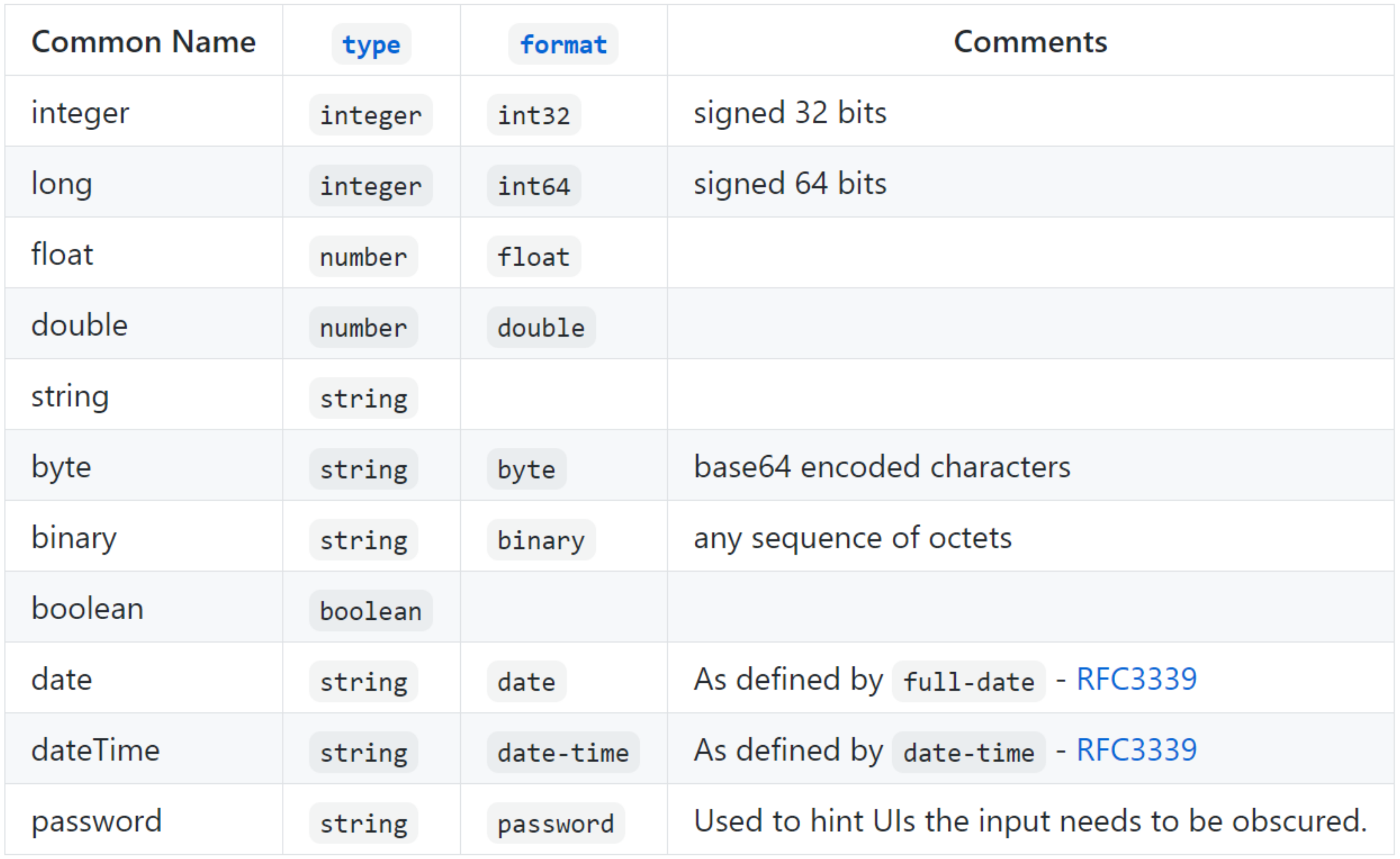


3.5. JSON Schema primitive types

JSON Schema defines seven primitive types for JSON values:

array	A JSON array.
boolean	A JSON boolean.
integer	A JSON number without a fraction or exponent part.
number	Any JSON number. Number includes integer.
null	The JSON null value.
object	A JSON object.
string	A JSON string.

OpenAPI Specification went further



Common Name	`type`	`format`	Comments
integer	`integer`	`int32`	signed 32 bits
long	`integer`	`int64`	signed 64 bits
float	`number`	`float`	
double	`number`	`double`	
string	`string`		
byte	`string`	`byte`	base64 encoded characters
binary	`string`	`binary`	any sequence of octets
boolean	`boolean`		
date	`string`	`date`	As defined by `full-date` - RFC3339
dateTime	`string`	`date-time`	As defined by `date-time` - RFC3339
password	`string`	`password`	Used to hint UIs the input needs to be obscured.

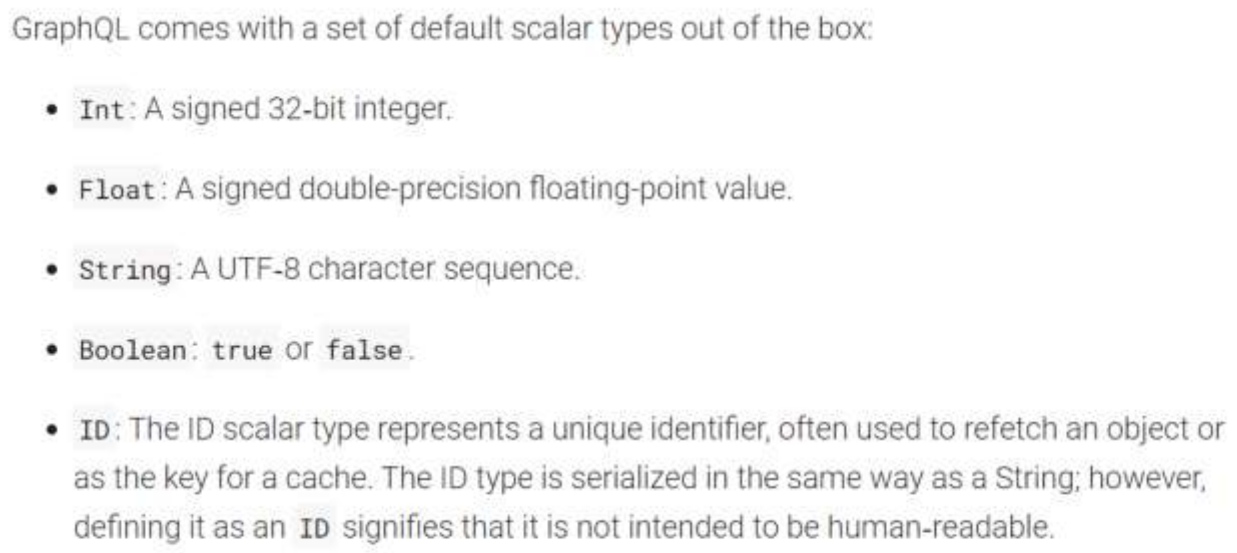
OData got carried away

3.3 Primitive Types

Structured types are composed of other structured types and primitive types. OData defines the following primitive types:

Type	Meaning
Edm.Binary	Binary data
Edm.Boolean	Binary-valued logic
Edm.Byte	Unsigned 8-bit integer
Edm.Date	Date without a time-zone offset
Edm.DateTimeOffset	Date and time with a time-zone offset, no leap seconds
Edm.Decimal	Numeric values with decimal representation
Edm.Double	IEEE 754 binary64 floating-point number (15-17 decimal digits)
Edm.Duration	Signed duration in days, hours, minutes, and (sub)seconds
Edm.Guid	16-byte (128-bit) unique identifier
Edm.Int16	Signed 16-bit integer
Edm.Int32	Signed 32-bit integer
Edm.Int64	Signed 64-bit integer
Edm.SByte	Signed 8-bit integer
Edm.Single	IEEE 754 binary32 floating-point number (6-9 decimal digits)
Edm.Stream	Binary data stream
Edm.String	Sequence of UTF-8 characters
Edm.TimeOfDay	Clock time 00:00-23:59:59.999999999999
Edm.Geography	Abstract base type for all Geography types
Edm.GeographyPoint	A point in a round-earth coordinate system
Edm.GeographyLineString	Line string in a round-earth coordinate system
Edm.GeographyPolygon	Polygon in a round-earth coordinate system
Edm.GeographyMultiPoint	Collection of points in a round-earth coordinate system
Edm.GeographyMultiLineString	Collection of line strings in a round-earth coordinate system
Edm.GeographyMultiPolygon	Collection of polygons in a round-earth coordinate system
Edm.GeographyCollection	Collection of arbitrary Geography values
Edm.Geometry	Abstract base type for all Geometry types
Edm.GeometryPoint	Point in a flat-earth coordinate system
Edm.GeometryLineString	Line string in a flat-earth coordinate system
Edm.GeometryPolygon	Polygon in a flat-earth coordinate system
Edm.GeometryMultiPoint	Collection of points in a flat-earth coordinate system
Edm.GeometryMultiLineString	Collection of line strings in a flat-earth coordinate system
Edm.GeometryMultiPolygon	Collection of polygons in a flat-earth coordinate system
Edm.GeometryCollection	Collection of arbitrary Geometry values

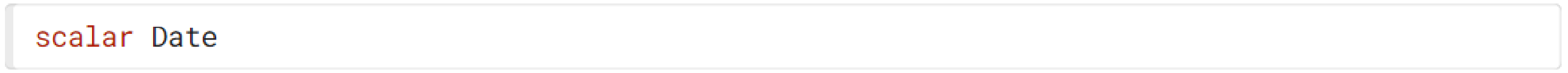
GraphQL kept things simple



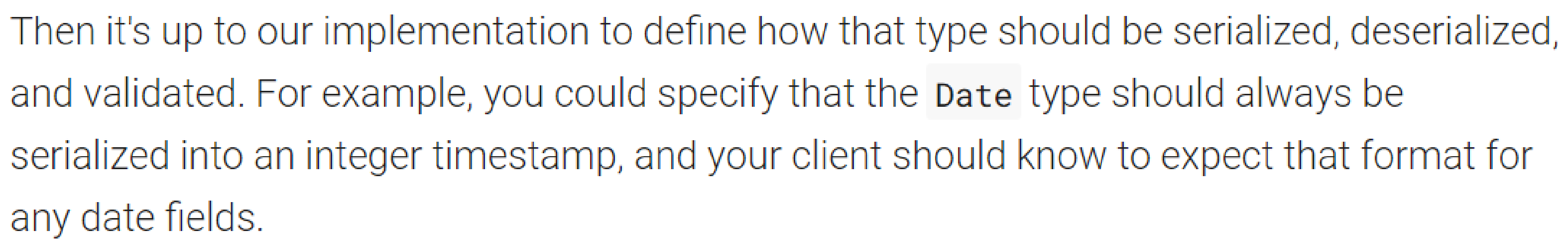
GraphQL comes with a set of default scalar types out of the box:

- `Int`: A signed 32-bit integer.
- `Float`: A signed double-precision floating-point value.
- `String`: A UTF-8 character sequence.
- `Boolean`: `true` or `false`.
- `ID`: The ID scalar type represents a unique identifier, often used to refetch an object or as the key for a cache. The ID type is serialized in the same way as a String; however, defining it as an `ID` signifies that it is not intended to be human-readable.

With some exercises left up to the implementer….

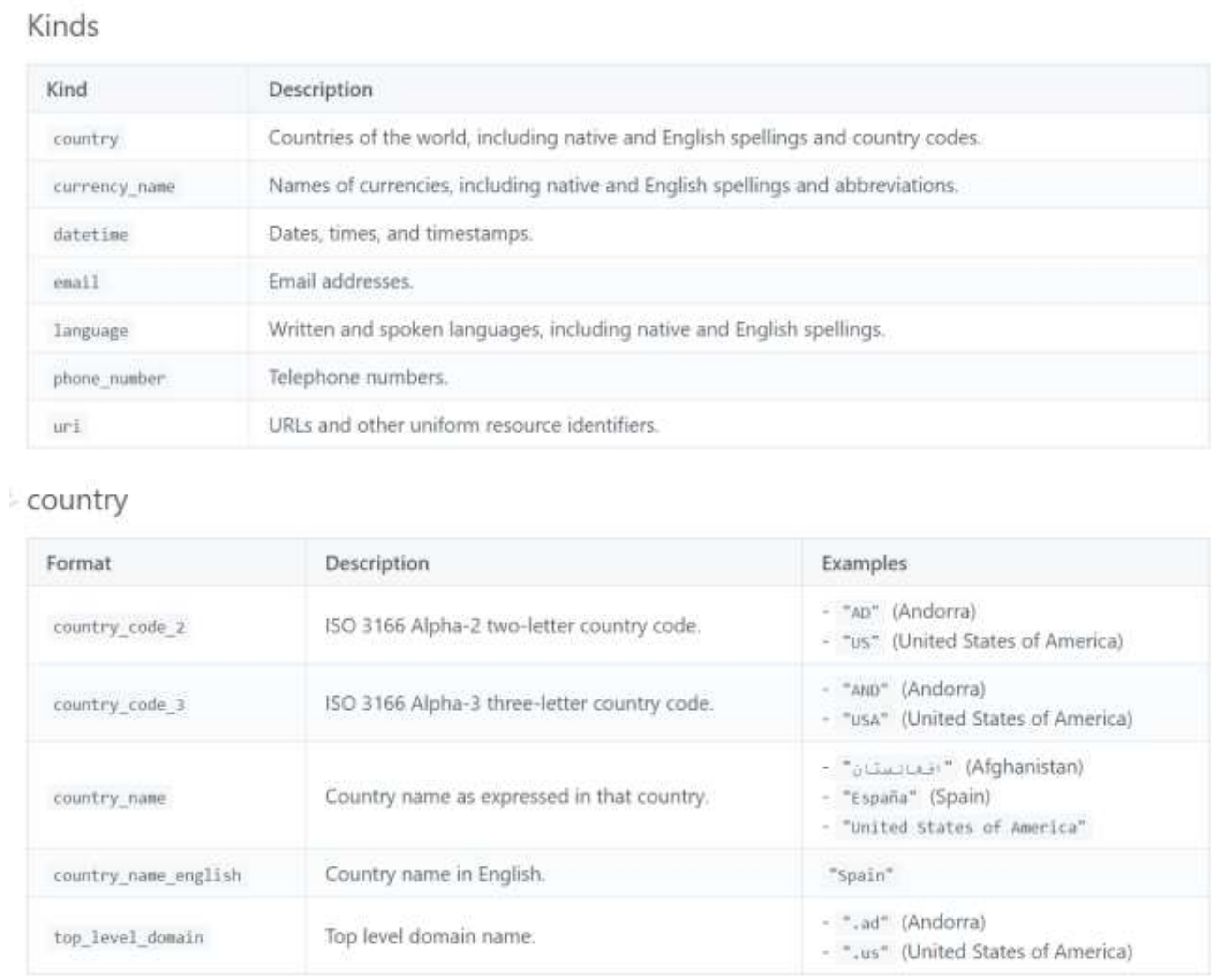


```
scalar Date
```



Then it's up to our implementation to define how that type should be serialized, deserialized, and validated. For example, you could specify that the `Date` type should always be serialized into an integer timestamp, and your client should know to expect that format for any date fields.

Some have gone beyond language primitives…



Kinds

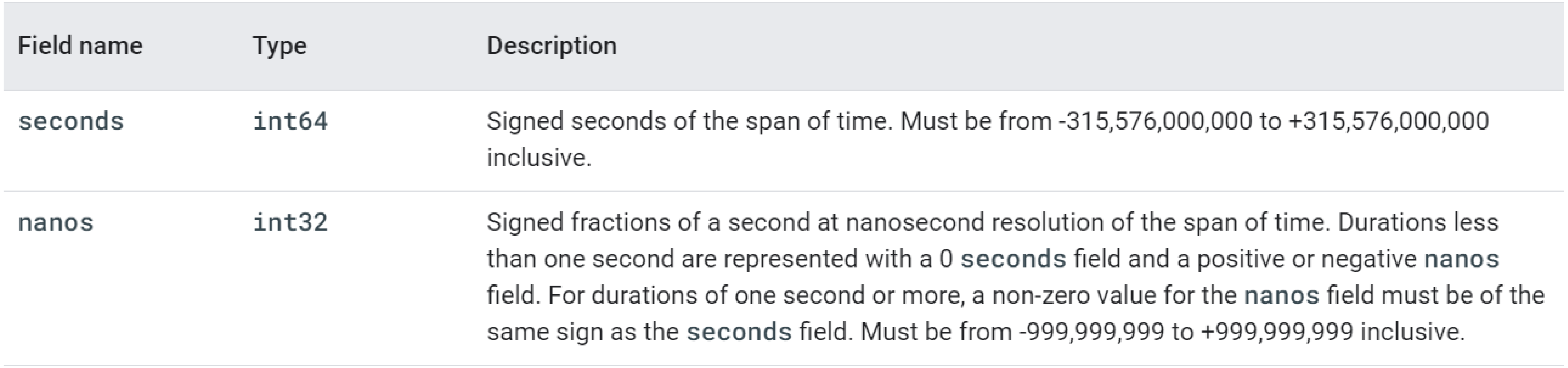
Kind	Description
`country`	Countries of the world, including native and English spellings and country codes.
`currency_name`	Names of currencies, including native and English spellings and abbreviations.
`datetime`	Dates, times, and timestamps.
`email`	Email addresses.
`language`	Written and spoken languages, including native and English spellings.
`phone_number`	Telephone numbers.
`uri`	URLs and other uniform resource identifiers.

country

Format	Description	Examples
`country_code_2`	ISO 3166 Alpha-2 two-letter country code.	- `"AD"` (Andorra) - `"US"` (United States of America)
`country_code_3`	ISO 3166 Alpha-3 three-letter country code.	- `"AND"` (Andorra) - `"USA"` (United States of America)
`country_name`	Country name as expressed in that country.	- `"افغانستان"` (Afghanistan) - `"España"` (Spain) - `"United States of America"`
`country_name_english`	Country name in English.	`"Spain"`
`top_level_domain`	Top level domain name.	- `".ad"` (Andorra) - `".us"` (United States of America)

https://docs.akita.software/docs/data-formats

Beyond primitives completely



Field name	Type	Description
`seconds`	`int64`	Signed seconds of the span of time. Must be from -315,576,000,000 to +315,576,000,000 inclusive.
`nanos`	`int32`	Signed fractions of a second at nanosecond resolution of the span of time. Durations less than one second are represented with a 0 `seconds` field and a positive or negative `nanos` field. For durations of one second or more, a non-zero value for the `nanos` field must be of the same sign as the `seconds` field. Must be from -999,999,999 to +999,999,999 inclusive.

https://developers.google.com/protocol-buffers/docs/reference/google.protobuf#google.protobuf.Duration

More than one right answer

```
{
    "linkset":
      [
        { "anchor": "http://example.net/bar",
          "next": [
                {"href": "http://example.com/foo1"}
          ]
        },
        { "anchor": "http://example.net/boo",
          "http://example.com/relations/baz" : [
                {"href": "http://example.com/foo2"}
          ]
        }
      ]
  }
```

https://www.ietf.org/archive/id/draft-wilde-linkset-07.txt

```
[
    {
      "href": "https://evertpot.com/",
      "rel": "author",
      "title": "Evert Pot"
    },
    {
      "href": "https://test.example/",
      "rel": "self"
    }
]
```

https://tools.ietf.org/html/draft-pot-json-link-02

JSON Type Serialization Registry

- Globally unique type identifier
- Reference to specification that defines JSON fragment
- Syntax (and semantics?)
- Standardized Schema format?

Why?

- Stop re-inventing JSON wheels
- Stop bikeshedding, support multiple.
- Needed because APIs share lots of semantics between consumer and provider
- Developers are not using media types to define payload semantics and serialization, so language based serializers define the behavior, which hurts interop.