JSON and BSON

JSON is a widely used data interchange format popular across many applications and technology stacks. BSON, the binary representation of JSON, is primarily used internally by MongoDB for efficient storage and data traversal.

Table of contents

What is JavaScript Object Notation?

JSON, or **J**ava**S**cript **O**bject **N**otation, is a human-readable format for data interchange, introduced in the early 2000s. Even though JSON is based on a subset of the JavaScript programming language standard, it's completely language-independent.

JSON *objects* are associative containers, wherein a string key is mapped to a *value* (which can be a number, string, boolean, array, an empty value – null, or even another object). Almost any programming language has support for this abstract data structure – objects in JavaScript, dictionaries in Python, hash tables in Java and C#, associative arrays in C++, and so on.

What does JSON data look like?

JSON objects are structured in human-readable formats, which are also easy for applications to read.

```
{
  "_id": 1,
  "name": { "first" : "John", "last" : "Backus" },
  "contribs": [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ],
  "awards": [
    {
      "award": "W.W. McDowell Award",
      "year": 1967,
      "by": "IEEE Computer Society"
    }, {
      "award": "Draper Prize",
      "year": 1993,
      "by": "National Academy of Engineering"
    }
  ]
}
```

Notice that the JSON file consists of key-value pairs separated by commas and key and value pairs are indicated using colon (:). The JSON object (document) starts and ends with curly braces. You can use any of the supported data types. The above example shows strings (in double quotes), numbers, and arrays (inside square brackets).

As JavaScript became the leading language for web development, JSON began to take on a life of its own. By virtue of being both human- and machine-readable, and comparatively simple to implement support in other languages, JSON quickly moved beyond the web page and into software everywhere.

Today, JSON shows up in many different cases:

- APIs
- Configuration files

- Log messages
- Database storage

The MongoDB-JSON connection

MongoDB was designed from its inception to be a database focused on delivering a great development experience. JSON's ubiquity made it the obvious choice for representing data structures in MongoDB's document data model.

It is easier to build applications using technology stacks like MEAN and MERN, as developers can use a single programming language (JavaScript) from end to end.

However, there are several issues that make JSON less than ideal for usage inside of a database.

1. JSON only supports a limited number of basic data types. Most notably, JSON lacks support for datetime and binary data.
2. JSON objects and properties don't have fixed length which makes traversal slower.
3. JSON does not provide metadata and type information, taking longer to retrieve documents.

To make MongoDB JSON-first but still high-performance and general purpose, BSON was invented to bridge the gap: a binary representation to store data as JSON documents, optimized for speed, space, and efficiency. It's not dissimilar from other binary interchange formats like Protocol Buffers, or Thrift, in terms of approach.

Binary JSON document

BSON stands for “Binary JSON,” and that's exactly what it was invented to be. A BSON file is a binary representation of the corresponding JSON file. BSON's binary-encoded serialization format encodes type and length information as well, which allows it to be traversed much more quickly compared to JSON.

BSON adds some additional data types (non-JSON-native), like dates and binary data, without which MongoDB would have been missing some valuable support.

BSON files

The following are some example JSON objects and their corresponding Binary JSON representations.

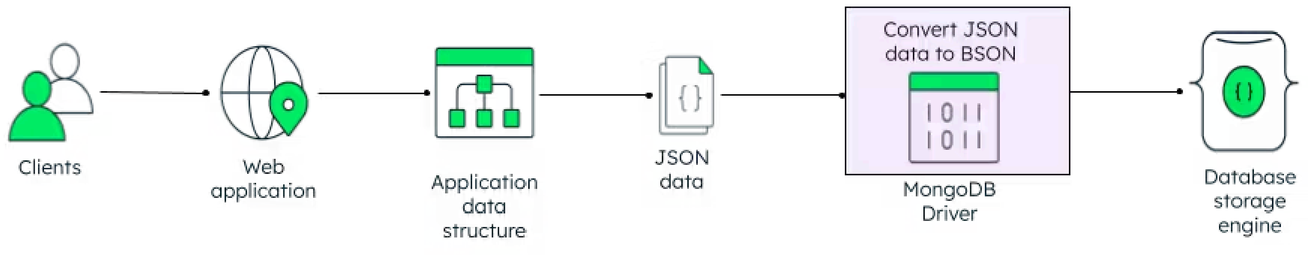
```
{"hello": "world"} →
\x16\x00\x00\x00           // total document size
\x02                       // 0x02 = type String
hello\x00                  // field name
\x06\x00\x00\x00world\x00  // field value
\x00                       // 0x00 = type EOO ('end of object')

{"BSON": ["awesome", 5.05, 1986]} →
\x31\x00\x00\x00
 \x04BSON\x00
 \x26\x00\x00\x00
 \x02\x30\x00\x08\x00\x00\x00awesome\x00
 \x01\x31\x00\x33\x33\x33\x33\x33\x33\x14\x40
 \x10\x32\x00\xc2\x07\x00\x00
 \x00
 \x00
```

You can learn more about the BSON grammar in the BSON specification.

Does MongoDB use BSON or JSON?

MongoDB stores data in BSON format both internally and over the network. Anything you can represent in JSON can be natively stored in MongoDB and retrieved just as easily in JSON.



When using the MongoDB driver for your favorite programming language, you work with the native data structures for that language. Your application needs to convert the native data structure (for example, a JavaScript object or POJO) to JSON. The MongoDB driver then takes care of converting the data from JSON to BSON and back when querying the database.

Unlike systems that store JSON as string-encoded values or binary-encoded blobs, MongoDB uses BSON to offer powerful indexing and querying features on top of the web's most popular data format.

For example, MongoDB allows developers to query and manipulate objects by specific keys inside the JSON/BSON document, even in nested documents many layers deep into a record, and create high-performance indexes on those same keys and values.

Firstly, BSON files may contain datetime or binary objects that are not natively representable in pure JSON.

Second, each programming language has its own object semantics. JSON objects have ordered keys, for instance, while Python dictionaries (the closest native data structure that's analogous to JavaScript objects) are unordered, while differences in numeric and string data types can also come into play. Third, BSON supports a variety of numeric types that are not native to JSON, and many languages represent these differently.

EJSON

EJSON or Extended JSON is a JSON-compatible way to represent BSON values in MongoDB. As JSON supports only a subset of the types supported by BSON, MongoDB adds certain extensions to the JSON format, namely, canonical mode and relaxed mode. MongoDB provides methods like serialize, deserialize, parse, and stringify for EJSON.

Parse JSON

When an application writes or updates data into MongoDB using the shell, API, or MongoDB Atlas, the data is parsed into BSON format by the MongoDB driver. Parsing involves identifying and interpreting the JavaScript objects or JSON structure, mapping each field to the right BSON data type, and converting the values. When an application requests data from MongoDB, the driver converts the data into JSON string before sending it through the server.

Check your driver documentation to make sure you understand how to best access MongoDB BSON-backed data in your language.

JSON vs BSON

	JSON	BSON
Encoding	UTF-8 string	Binary
Data Support	String, boolean, number, array, object, null	String, boolean, number (integer, float, long, decimal128...), array, null, date, BinData
Readability	Human and machine	Machine only

JSON and BSON are indeed close cousins by design. BSON is designed as a binary representation of JSON data, with specific extensions for broader applications, and optimized for data storage and traversal. Just like JSON, BSON supports embedding objects and arrays.

One particular way in which BSON differs from JSON is in its support for some more advanced types of data. JSON does not, for instance, differentiate between integers (which are round numbers) and floating-point numbers (which have decimal precision to various degrees).

Most server-side programming languages have more sophisticated numeric types (standards include integer, regular precision floating point number – a.k.a. “float” – double-precision floating point – a.k.a. “double” – and boolean values), each with its own optimal usage for efficient mathematical operations.

Schema flexibility and data governance

One of the big attractions for developers using databases with JSON and BSON data models is the dynamic and flexible schema they provide when compared to the rigid, tabular data models used by relational databases.

Firstly, MongoDB documents are polymorphic – fields can vary from document to document within a single collection (analogous to tables in a relational database). This flexibility makes it easier to model data of any structure and adapt the model as requirements change.