AVRO

**Introduction**

Apache Avro™ is a data serialization system.

Avro provides:

* Rich data structures.
* A compact, fast, binary data format.
* A container file, to store persistent data.
* Remote procedure call (RPC).
* Simple integration with dynamic languages. Code generation is not required to read or write data files nor to use or implement RPC protocols. Code generation as an optional optimization, only worth implementing for statically typed languages.

## Schema Declaration

A Schema is represented in [JSON](http://www.json.org/) by one of

### Primitive Types

The set of primitive type names is:

* null: no value
* boolean: a binary value
* int: 32-bit signed integer
* long: 64-bit signed integer
* float: single precision (32-bit) IEEE 754 floating-point number
* double: double precision (64-bit) IEEE 754 floating-point number
* bytes: sequence of 8-bit unsigned bytes
* string: unicode character sequence

Primitive types have no specified attributes.

Primitive type names are also defined type names. Thus, for example, the schema "string" is equivalent to:

{"type": "string"}

### Complex Types

Avro supports six kinds of complex types: records, enums, arrays, maps, unions and fixed.

Records:

For example, a linked-list of 64-bit values may be defined with:

{

"type": "record",

"name": "LongList",

"aliases": ["LinkedLongs"], // old name for this

"fields" : [

{"name": "value", "type": "long"}, // each element has a long

{"name": "next", "type": ["LongList", "null"]} // optional next element

]

}

#### **Enums**

Ex:

{ "type": "enum",

"name": "Suit",

"symbols" : ["SPADES", "HEARTS", "DIAMONDS", "CLUBS"]

}

#### **Arrays**

Arrays use the type name "array" and support a single attribute:

* items: the schema of the array's items.

For example, an array of strings is declared with:

{"type": "array", "items": "string"}

#### **Maps**

Maps use the type name "map" and support one attribute:

* values: the schema of the map's values.

Map keys are assumed to be strings.

For example, a map from string to long is declared with:

{"type": "map", "values": "long"}

#### **Unions**

Unions, as mentioned above, are represented using JSON arrays. For example, ["string", "null"] declares a schema which may be either a string or null.

Unions may not contain more than one schema with the same type, except for the named types record, fixed and enum. For example, unions containing two array types or two map types are not permitted, but two types with different names are permitted. (Names permit efficient resolution when reading and writing unions.)

Unions may not immediately contain other unions.

#### **Fixed**

Fixed uses the type name "fixed" and supports two attributes:

* name: a string naming this fixed (required).
* namespace, a string that qualifies the name;
* aliases: a JSON array of strings, providing alternate names for this enum (optional).
* size: an integer, specifying the number of bytes per value (required).

For example, 16-byte quantity may be declared with:

{"type": "fixed", "size": 16, "name": "md5"}