Glossary

IO

- **Encoder**/**Decoder**: Avro specifies two different encodings: Binary and JSON. See http://avro.apache.org/docs/current/spec.html#Encodings for the details of these encodings.
- **block**: Array and Maps are encoded as a series of blocks, with a "count" long at the beginning of each block (and optionally "size"). Used for reading and writing data structures that don't fit into memory. May also refer to the "blocks" in a file object container.
- **`DatumReader`**/**`DatumWriter`**
- **`DataFileReader`**/**`DataFileWriter`**
- **Projection**: The ability to select a subset of data from an Avro schema by specifying an "expected" schema with the objects you'd like to read. Can possibly avoid the overhead of deserialization of all columns when you only want a few.

IPC

- **requestor**
- **responder**
- **server**
- **transceiver**

API Styles

Different API styles are possible with Avro, and correspond to different in-memory representations for Avro data.

- **generic** All avro records are represented by a generic attribute/value data structure. This style is most useful for systems which dynamically process datasets based on user-provided scripts. For example, a program may be passed a data file whose schema has not been previously seen by the program and told to sort it by the field named "city".
- **specific**: Each Avro record corresponds to a different kind of object in the programming language. For example, in Java, C and C++, a specific API would generate a distinct class or struct definition for each record definition. This style is used for programs written to process a specific schema. RPC systems typically use this.
- **reflect** Avro schemas are generated via reflection to correspond to existing programming language datastructures. This may be useful when converting an existing codebase to use Avro with minimal modifications.

Many Avro terms of art are defined in the specification.