AEP 101 - Optional Fields

Discussed in AVRO-519

Arguments in Favor

- Large sets of very sparse optional fields have a variety of uses, e.g. tags
- Some other serialization systems have direct support for optional fields
- Avro can handle missing fields by providing a default value however this can be inefficient for sparse sets of options and more complex since the default value doubles as the "empty" marker
- Avro can handle optional fields as a vector of a union of all the optional fields. While this works, it makes quickly determining whether or not a particular fields is present complex and inefficient. Annotations can be used to decrease the API complexity but that argues for making optional a top level concept.
- Support for fast "select" of sparse optional fields from a top level record would benefit certain applications.

Proposal

An additional field attribute:

- "optional" - with values "true"/"false" (where "false" is assumed)

For the encoding, any record which includes optional fields would be prefixed by an presence map which would be a sequence of int8 x* where:

- `x > 0`: the lower 7 bits are presence bits for the next 7 optional fields (low bit first)
- `-128 < x < 0`: the next present field is position x + 135 (as x runs from 0 to -127 and the first 7 must be empty otherwise we would use the x > 0 encoding)
- `x == -128`: no optional fields present in the next 134 optional fields
- `x = 0`: end of sequence

further, if the map has covered all the options, the end-of-sequence marker can be elided. For example, a type with 3 optional fields would require only a single byte.

This will permit encoding at 8/7 of a bit per present entry (worst case) and at a cost of 8/134 (0.06) bits/entry per all but last not-present (7.5 bytes / 1000 optional fields).

This encoding is backward compatible as well as schema's which do not contain optional elements do not have the presence map and the encoding is therefore identical. Backward
compatibility can be maintained by simply using the default value for not-present fields.

Language APIs

Efficient support could include either an explicit presence test or a function which returns the value or default value (if the field is not present).