Protocol Schema Documentation

JSON Schema Specification for Binary Protocol Parsing

Schema ID: http://example.com/schemas/protocol.json
Version: Draft-07 Compatible

Table of Contents

1. Overview
2. Root Schema Properties
3. Protocol Options
4. Message Specification
5. Field Specifications
6. Constraint Specifications
7. Data Extraction
8. Offset Configurations
9. Field Decoding
10. Semantics Specifications
11. Usage Examples

1. Overview

Purpose: This JSON schema defines a comprehensive protocol specification system for parsing and interpreting binary data formats. It supports complex message structures, field definitions, constraints, and semantic annotations.

Key Features

- **Mode Support:** Binary protocol parsing
- **Hierarchical Structure:** Nested objects and arrays
- **Flexible Offsets:** Multiple offset calculation methods
- **Rich Constraints:** Numeric ranges, string validation, enums
- **Semantic Annotations:** Message types, checksums, monotonicity
- **Bit-level Operations:** Support for bit fields and bit objects

2. Root Schema Properties

The root Protocol object defines the fundamental properties of a protocol specification.

Property	Type	Required	Description
`id`	string	**Required**	Unique identifier for the protocol
`name`	string	**Required**	Human-readable name of the protocol
`description`	string	**Required**	Detailed description of the protocol purpose and usage
`creator`	string	**Required**	Identifier of the protocol creator or organization
`lastModifiedTimestamp`	number	**Required**	Unix timestamp of last modification
`createdTimestamp`	number	**Required**	Unix timestamp of protocol creation
`messageSpec`	MessageSpec	**Required**	Specification of the message structure
`options`	ProtocolOptions	**Required**	Protocol-specific configuration options

3. Protocol Options

Protocol options define how the protocol should be processed, supporting binary mode.

Binary Protocol Options

Property	Type	Description
`type`	"binary"	Indicates binary protocol processing
`hexMode`	boolean	Whether to display/input data in hexadecimal format

4. Message Specification

The MessageSpec defines the structure and version of protocol messages.

Property	Type	Description
`version`	string	Version identifier for the message specification
`spec`	ObjectSpec	Root object specification defining the message structure

ObjectSpec Structure

ObjectSpec defines a structured object with nested fields and child specifications.

Property	Type	Description
`id`	string	Unique identifier within the specification
`name`	string	Human-readable name for the object
`children`	AbstractFieldSpec[]	Array of field specifications within this object
`description`	string	Optional description of the object's purpose
`type`	"ObjectSpec"	Type discriminator

5. Field Specifications

Field specifications define different types of data fields within protocol messages.

Abstract Field Specification

Base specification that all field types inherit from:

Field Types:

- `ObjectFieldSpec` - Nested object field
- `FieldSpec` - Basic data field
- `SwitchCaseFieldSpec` - Conditional field based on another field's value
- `BitObjectFieldSpec` - Bit-level object field
- `BitFieldSpec` - Individual bit field
- `ArrayFieldSpec` - Array of elements

FieldSpec (Basic Field)

Property	Type	Description
`hexMode`	boolean	Whether to display this field in hexadecimal
`dataExtraction`	DataExtractionSpec	How to extract data for this field
`fieldDecoding`	FieldDecodingSpec	How to decode the extracted data
`constraints`	ConstraintSpec[]	Validation constraints for the field
`semantics`	SemanticsSpec	Semantic meaning of the field

ArrayFieldSpec

Defines arrays of objects or bit objects with configurable sizing.

Property	Type	Description
`fieldItem`	BitObjectSpec | ObjectSpec	Specification for array elements

Property	Type	Description
`startOffset`	OffsetConfig	Where the array begins in the message
`arraySize`	ArraySizeConfig	How to determine array size

SwitchCaseFieldSpec

Provides conditional field definitions based on another field's value.

Property	Type	Description
`pathOfFieldToSwitchOn`	string	Path to the field whose value determines the case
`caseOptions`	object	Map of field values to their corresponding field specifications

6. Constraint Specifications

Constraints define validation rules for field values.

Constraint Types

Available Constraint Types:

- `NumericRangeConstraintSpec` - Numeric value ranges
- `StringSizeConstraintSpec` - String length validation
- `StringCharRulesConstraintSpec` - Character whitelist/blacklist
- `StringRegexConstraintSpec` - Regular expression matching
- `EnumSpec` - Enumerated values

NumericRangeConstraintSpec

Validates numeric values against specified ranges.

Range Types:

- `Between` - Value between min and max
- `LowerThan` - Value less than specified
- `GreaterThan` - Value greater than specified
- `EqualTo` - Value equal to specified

StringCharRulesConstraintSpec

Property	Type	Description
`rulesType`	"whiteList" | "blackList"	Whether to allow or block specified characters
`abc`	string[]	Array of allowed/blocked character categories
`charList`	string[]	Array of specific allowed/blocked characters

EnumSpec

Defines a set of valid key-value pairs for enumerated fields.

Property	Type	Description
`options`	object[]	Array of {key, label} pairs defining valid values

7. Data Extraction

DataExtractionSpec defines how to extract raw bytes from the message for field processing.

Property	Type	Description
`startOffset`	OffsetConfig	Where to start extracting data
`endOffset`	OffsetConfig	Where to stop extracting data

Note: The combination of startOffset and endOffset determines the exact bytes that will be processed for this field.

8. Offset Configurations

Offset configurations provide flexible ways to calculate positions within messages.

Offset Types

Available Offset Types:

- `RelativeToMessageStartOffsetConfig` - Fixed offset from message start
- `RelativeToMessageEndOffsetConfig` - Fixed offset from message end
- `RelativeToFieldOffsetConfig` - Relative to another field's position
- `FieldSizeOffsetConfig` - Based on field size
- `CharDelimiterOffsetConfig` - Until specific character delimiter
- `StoredSizeInAnotherFieldOffsetConfig` - Size stored in another field
- `DynamicArrayOffsetConfig` - For dynamic array positioning
- `StaticArrayOffsetConfig` - For static array positioning
- `EmptyOffsetConfig` - No offset calculation

Key Offset Configurations

RelativeToMessageStartOffsetConfig

Simple fixed offset from the beginning of the message.

Property	Type	Description
`value`	number	Byte offset from message start (0-based)

CharDelimiterOffsetConfig

Continues until a specific character is encountered.

Property	Type	Description
`delimiter`	string	Character that marks the end of the field

DynamicArrayOffsetConfig

For arrays where size is determined by another field's value.

Property	Type	Description
`arraySizeFieldPath`	string	Path to field containing array size
`itemSize`	number	Size of each array item in bytes

9. Field Decoding

Field decoding specifications define how to interpret extracted raw bytes.

Decoding Types

Available Decoding Types:

- `StringFieldDecodingSpec` - Text string decoding
- `NumberFieldDecodingSpec` - Numeric value decoding
- `RawFieldDecodingSpec` - Raw bytes (no decoding)

NumberFieldDecodingSpec

Decodes numeric values with various format options.

Property	Type	Description
`signed`	boolean	Whether the number is signed or unsigned
`endianness`	"LE" | "BE"	Little Endian or Big Endian byte order
`numberType`	"int" | "decimal"	Integer or decimal number interpretation

StringFieldDecodingSpec

Decodes text strings using specified character encoding.

Property	Type	Description
`charset`	string	Character encoding (e.g., "UTF-8", "ASCII")

10. Semantics Specifications

Semantic specifications provide meaning and interpretation context for fields.

Semantic Types

Available Semantic Types:

- `none` - No special semantics.
- `ChangingMonotonicitySemanticsSpec` - For values that change monotonically (e.g., sequence counters).
- `MessageLengthSemanticsSpec` - Describes a field that specifies the message length.
- `ChecksumSemanticsSpec` - Defines a checksum field for data integrity.
- `MessageTypeSemanticsSpec` - Identifies the message type or command.
- `message-destination` - Marks the field as the message destination address.
- `message-source` - Marks the field as the message source address.
- `gap` - Marks a field as a gap or padding, not a data field.

ChangingMonotonicitySemanticsSpec

For fields whose values are expected to change monotonically over a sequence of messages (e.g., a sequence counter).

Property	Type	Description
`monotonicityType`	enum	The type of monotonicity: `increasing` , `non-decreasing` , `decreasing` , or `non-increasing` .

MessageTypeSemanticsSpec

Used to identify the type of message, often used in switch-case logic to parse different message structures.

Property	Type	Description
`options`	MessageTypeOption[]	An array of possible message types, where each option is a key-label pair.

MessageLengthSemanticsSpec

Specifies a field that contains the length of the message or part of the message.

Property	Type	Description
`includeHeaderSize`	boolean	If true, the length value includes the size of the header. If false, it's only the payload length.
`options`	MessageLengthOption[]	An array of possible length options, typically used for validation or display.

ChecksumSemanticsSpec

Defines a field used for data integrity verification.

Property	Type	Description
`endianness`	"LE" | "BE"	The byte order of the checksum value.
`algorithm`	string	The checksum algorithm used (e.g., "CRC32", "SUM").
`dstFieldStartOffset`	number	The starting byte offset of the data range over which the checksum is calculated.
`dstFieldEndOffset`	number	The ending byte offset of the data range over which the checksum is calculated.

11. Usage Examples

This section is intended for concrete JSON examples demonstrating the use of this schema to define a protocol. A complete JSON object following the protocol schema would be shown here to illustrate how to define a simple protocol, such as Modbus or a custom IoT sensor format.

Example: Skeleton of a Simple Binary Protocol

```
{
  "id": "proto-001",
  "name": "Simple IoT Sensor Protocol",
  "description": "A basic binary protocol for IoT sensor readings.",
  "creator": "Example Corp",
  "lastModifiedTimestamp": 1672531200,
  "createdTimestamp": 1672531200,
  "options": {
    "type": "binary",
    "hexMode": true
  },
  "messageSpec": {
    "version": "1.0",
    "spec": {
      "id": "msg-root",
      "name": "SensorMessage",
      "type": "ObjectSpec",
      "children": [
        // Field definitions would go here...
        {
          "id": "field-header",
          "name": "Header",
          "type": "FieldSpec",
          // ... and so on
        }
      ]
    }
  }
}
```