FIX TRADING COMMUNITY™
INDUSTRY-DRIVEN • INDEPENDENT • NEUTRAL

Encoding FIX using JSON

Release Candidate 1 – User Guide

Document Revision 0.4.0 - November 17, 2016

THIS DOCUMENT IS A RELEASE CANDIDATE FOR A PROPOSED FIX TECHNICAL STANDARD. A RELEASE CANDIDATE HAS BEEN APPROVED BY THE GLOBAL TECHNICAL COMMITTEE AS AN INITIAL STEP IN CREATING A NEW FIX TECHNICAL STANDARD. POTENTIAL ADOPTERS ARE STRONGLY ENCOURAGED TO BEGIN WORKING WITH THE RELEASE CANDIDATE AND TO PROVIDE FEEDBACK TO THE GLOBAL TECHNICAL COMMITTEE AND THE WORKING GROUP THAT SUBMITTED THE PROPOSAL. THE FEEDBACK TO THE RELEASE CANDIDATE WILL DETERMINE IF ANOTHER REVISION AND RELEASE CANDIDATE IS NECESSARY OR IF THE RELEASE CANDIDATE CAN BE PROMOTED TO BECOME A FIX TECHNICAL STANDARD DRAFT.

© Copyright 2016 FIX Protocol Limited

DISCLAIMER

THE INFORMATION CONTAINED HEREIN AND THE FINANCIAL INFORMATION EXCHANGE PROTOCOL (COLLECTIVELY, THE "FIX PROTOCOL") ARE PROVIDED "AS IS" AND NO PERSON OR ENTITY ASSOCIATED WITH THE FIX PROTOCOL MAKES ANY REPRESENTATION OR WARRANTY, EXPRESS OR IMPLIED, AS TO THE FIX PROTOCOL (OR THE RESULTS TO BE OBTAINED BY THE USE THEREOF) OR ANY OTHER MATTER AND EACH SUCH PERSON AND ENTITY SPECIFICALLY DISCLAIMS ANY WARRANTY OF ORIGINALITY, ACCURACY, COMPLETENESS, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SUCH PERSONS AND ENTITIES DO NOT WARRANT THAT THE FIX PROTOCOL WILL CONFORM TO ANY DESCRIPTION THEREOF OR BE FREE OF ERRORS. THE ENTIRE RISK OF ANY USE OF THE FIX PROTOCOL IS ASSUMED BY THE USER.

NO PERSON OR ENTITY ASSOCIATED WITH THE FIX PROTOCOL SHALL HAVE ANY LIABILITY FOR DAMAGES OF ANY KIND ARISING IN ANY MANNER OUT OF OR IN CONNECTION WITH ANY USER'S USE OF (OR ANY INABILITY TO USE) THE FIX PROTOCOL, WHETHER DIRECT, INDIRECT, INCIDENTAL, SPECIAL OR CONSEQUENTIAL (INCLUDING, WITHOUT LIMITATION, LOSS OF DATA, LOSS OF USE, CLAIMS OF THIRD PARTIES OR LOST PROFITS OR REVENUES OR OTHER ECONOMIC LOSS), WHETHER IN TORT (INCLUDING NEGLIGENCE AND STRICT LIABILITY), CONTRACT OR OTHERWISE, WHETHER OR NOT ANY SUCH PERSON OR ENTITY HAS BEEN ADVISED OF, OR OTHERWISE MIGHT HAVE ANTICIPATED THE POSSIBILITY OF, SUCH DAMAGES.

DRAFT OR NOT RATIFIED PROPOSALS (REFER TO PROPOSAL STATUS AND/OR SUBMISSION STATUS ON COVER PAGE) ARE PROVIDED "AS IS" TO INTERESTED PARTIES FOR DISCUSSION ONLY. PARTIES THAT CHOOSE TO IMPLEMENT THIS DRAFT PROPOSAL DO SO AT THEIR OWN RISK. IT IS A DRAFT DOCUMENT AND MAY BE UPDATED, REPLACED, OR MADE OBSOLETE BY OTHER DOCUMENTS AT ANY TIME. THE FPL GLOBAL TECHNICAL COMMITTEE WILL NOT ALLOW EARLY IMPLEMENTATION TO CONSTRAIN ITS ABILITY TO MAKE CHANGES TO THIS SPECIFICATION PRIOR TO FINAL RELEASE. IT IS INAPPROPRIATE TO USE FPL WORKING DRAFTS AS REFERENCE MATERIAL OR TO CITE THEM AS OTHER THAN "WORKS IN PROGRESS". THE FPL GLOBAL TECHNICAL COMMITTEE WILL ISSUE, UPON COMPLETION OF REVIEW AND RATIFICATION, AN OFFICIAL STATUS ("APPROVED") OF/FOR THE PROPOSAL AND A RELEASE NUMBER.

No proprietary or ownership interest of any kind is granted with respect to the FIX Protocol (or any rights therein).

Copyright 2013-2016 FIX Protocol Ltd., all rights reserved.

Encoding FIX using JSON by FIX Protocol Ltd. is licensed under a Creative Commons Attribution-NoDerivatives 4.0 International License.

Based on a work at https://github.com/FIXTradingCommunity/fix-json-encoding-spec

Document History

Revision	Date	Author	Revision Comments
0.1.0	2016-04-14	Don Mendelson	Initial draft.
0.2.0	2016-09-12	Mike Gatny, Connamara Systems	Updated according to Working Group feedback.
0.3.0	2016-11-04	Mike Gatny, Connamara Systems	Recommend EP206 for sub-millisecond date/time precision.
0.4.0	2016-11-17	Mike Gatny, Connamara Systems	Updated *Name* vs *SymbolicName* rationale according to GTC feedback.

© Copyright, 2016, FIX Protocol, Limited

Table of Contents

© Copyright, 2016, FIX Protocol, Limited

1 Introduction

1.1 Objective

The objective of this user guide is to support development of web applications that require FIX semantics. JSON is another alternative to existing FIX encodings, including tag=value, FIXML, FAST, SBE and Google Protocol Buffers.

JSON encoding of FIX is optimized for operations in a web browser without the need for other software distribution to clients.

JSON encoding is also optimized for conversion to and from other FIX encodings. It is therefore a goal of this encoding not to discard any information that would be useful during such conversions.

It is not a goal to specify a template or schema format here, however it is a goal not to preclude the use of one.

On the scale of human-to-server web interactions, JSON provides acceptable performance. However, is not a goal to optimize JSON encoding to execute at very low latency like FIX binary encodings.

© Copyright, 2016, FIX Protocol, Limited

1.2 JavaScript and JSON

JavaScript is the predominant language for developing client-side logic in web applications. JavaScript is supported by all popular web browsers. A built-in feature of the language is JavaScript Object Notation (JSON). JSON syntax is identical to the code for creating JavaScript objects. It minimizes development effort while avoiding the complexity and greed for resources of XML parsers and the like.

An important characteristic of JSON is that it is self-describing. Elements have readable symbolic names. Therefore, message handlers can process message elements without looking up metadata in a data dictionary or schema.

JSON wire format can readily be parsed and encoded in other programming languages. There are several popular libraries for that purpose in Java, C#, C++, and so forth. These JSON implementations interoperate with JavaScript.

1.3 JSON Elements

JSON is so simple that the standard's authors foresaw no need for versioning since it is never expected to change. To summarize:

- An **array** is an ordered list of values.
- An **object** is a collection of name/value pairs. Names are *strings*.
- A **value** can be an *object*, *array*, *number*, *string*, *boolean*, or *null*.
- **Numbers** are signed. There is no syntactic distinction between integer and floating point values. In practice, most implementations store all numbers as double precision binary floating point.
- **Strings** are Unicode, with a few rules for escaping special values.
- **Booleans** are *true* or *false*.
- Arbitrary levels of *object* nesting are allowed.

1.4 Standards References

JavaScript is formalized as Standard ECMA-262 ECMAScript® 2015 Language Specification.

JSON is standardized by Standard ECMA-404 The JSON Data Interchange Format. The JSON standard is normative for this user guide.

1.5 Issues for Mapping JSON to FIX

This user guide provides standardized solutions to the following issues.

© Copyright, 2016, FIX Protocol, Limited

1.5.1 Dates and Times

JSON has no explicit provision for encoding dates or times. However, most languages/platforms that support JSON also support conversion of date/time to and from strings in ISO 8601 format (e.g. the JavaScript **Date** object).

Another potential issue is that languages/platforms only support millisecond precision (e.g. the JavaScript **Date** object), while FIX timestamps may require microsecond or nanosecond precision. Given that any timestamps captured on the client side are limited by PC clock precision, millisecond precision should be sufficient for web applications. When finer than millisecond precision is required, applications should adhere to the recommendations of FIX.5.0 SP2 EP206: Clock Synchronization Data Types Enhancements.

1.5.2 Decimal Representation

JSON does not provide a numeric data type that is suitable for storing prices, quantities, etc. Furthermore, since most implementations (including JavaScript) store all JSON numeric values using a binary floating point data type, attempts to represent decimals as scaled integers, as was done with FIX binary encodings, are unfortunately pointless.

1.5.3 Enumerations

Enumerations of valid values are needed for codes in FIX fields, but JSON has no special syntax for enumerations.

Languages/platforms that support JSON may also lack support for enumerations (e.g. JavaScript). Although it may be possible in such cases to emulate an enumeration with an associative array of symbolic names and values, deserialization of a code in JSON does not automatically associate to its symbolic name, and serialized strings or numbers are not constrained to valid values.

1.5.4 No tags

JSON does not have a built-in feature equivalent to FIX field tags or component IDs. This user guide suggests using symbolic names directly to avoid lookup by tag. The reduction in processing is offset by longer messages on the wire. This seems an acceptable trade-off since a web UI is unlikely to capture very large numbers of fields per message.

© Copyright, 2016, FIX Protocol, Limited

1.5.5 No Templates

JSON serialization and deserialization are not controlled by an external template or schema, only by an object that is being serialized. Each object is *sui generis*; JSON grew out of JavaScript, which does not have classes that objects must conform to, as realized by Java, C# and C++. It does have a prototype feature, but JavaScript objects are quite malleable. Properties and functions can be added on the fly. Nevertheless, it is possible to generate JSON objects corresponding to messages defined by the FIX Repository or FIX Orchestra.

1.6 Constraint

This user guide will depend only on standard JavaScript features that are implemented in most browsers and the JSON standard, without dependency on third-party frameworks.

2 Field Encoding

2.1 Data Types of Values

This user guide specifies that all the semantic data types of FIX protocol should be mapped to JSON *string* values in order to maximize the ability of applications to simply display the data.

Application logic must be tailored to handle data with proper semantics if it does anything with the data other than simply display it. Since there is no feature to convey the semantic type in-band, the actual FIX type must be referenced at development time. It is recommended that applications use either FIX Repository or FIX Orchestra for this purpose.

JSON does not provide a numeric data type that is suitable for storing prices, quantities, etc. Furthermore, most implementations (e.g. JavaScript) store all JSON numeric values using a binary floating point data type. Using JSON *string* values to represent FIX protocol numeric types circumvents this issue, and allows applications to choose the most appropriate data type provided by their language/platform (e.g. the Java *BigDecimal* type).

© Copyright, 2016, FIX Protocol, Limited

2.2 Names

Names of fields must be encoded exactly as they are spelled and capitalized in the FIX Repository.

The field's *Name* is used instead of its *Tag* number to enable applications to display a human-readable form with little or no logic and without requiring a data dictionary in the browser. However, for user-defined fields, the *Tag* number may be used instead of the *Name*.

The field's *Value* is used instead of its *SymbolicName* to optimize for interoperability with and convertibility to/from the other standard encodings of FIX (e.g. FIXML, SBE, GPB).

2.3 Field Encoding

Fields are encoded in accordance with the JSON standard as name/value pairs. Values must be serialized as JSON *strings*.

Example of a FIX field encoded as a JSON name/value pair:

FIX tag=value Encoding	JSON Encoding
`31=47.50`	`"LastPx": "47.50"`
`54=1`	`"Side": "1"`

2.3.1 Byte order

Since all JSON values, including numbers, are serialized as their string equivalent, there is no issue with byte order (endianness).

3 Message Structure

3.1 Field Presence

Although JSON does have a special value for *null*, it need not be used for a non-populated optional FIX field. Like FIX *tag=value* encoding, optional fields that are not populated are simply not serialized on the wire.

3.2 Field Order

Like FIX *tag=value* encoding, order of fields within a message or repeating group entry is not significant. All fields are accessed by name.

3.3 Message Framing

Each message is serialized as a JSON object, contained by opening and closing braces. A message may contain other JSON objects, specifically, repeating groups (see below).

© Copyright, 2016, FIX Protocol, Limited

Since this encoding is designed for use with web protocols, message framing is generally handled by the session layer protocol, e.g. HTTP or websockets. In these cases, no additional framing protocol is needed.

For cases where an additional framing protocol *is* needed, applications may use *FIX Simple Open Framing Header* (SOFH).

© Copyright, 2016, FIX Protocol, Limited

3.4 Header, Body, and Trailer

Every JSON message must have top-level fields named “Header”, “Body”, and “Trailer”:

```
{
    "Header": {},
    "Body": {},
    "Trailer": {}
}
```

This structure serves the goal of *not* discarding information that is useful when converting to/from other FIX encodings.

JSON encoding does not include a “CheckSum” field since it is unlikely to be useful at best, and likely to be incorrect at worst (e.g. if copied over from another FIX encoding).

3.5 Message Type

To identify a message on the wire, every JSON message should have a “MsgType” field in the “Header” sub-object. The value should be a valid *Value* of the “MsgType” field as defined in the FIX repository:

Example for a “NewOrderSingle” message:

```
{
    "Header": {
        "MsgType": "D"
    }
}
```

© Copyright, 2016, FIX Protocol, Limited

3.7 Repeating Groups

A repeating group is serialized as an array of JSON objects, each containing the fields that belong to a repeating group entry. A JSON array is surrounded by square brackets, and each entry is separated by a comma. Each entry is its own JSON object. Because some of the fields may be optional, not all entries are required to contain the same fields.

The count of entries is implicit to the array structure. There is no explicit *NumInGroup* field in the JSON encoding.

The name of a repeating group is the name of the associated *NumInGroup* field name as it appears in the FIX Repository.

Example of a “NoMDEntries” group with two entries:

```
{
    "Header": {
        "MsgType": "W"
    },
    "Body": {
        "NoMDEntries": [
            { "MDEntryType": "0", "MDEntryPx": "2179.75", "MDEntrySize": "175" },
            { "MDEntryType": "1", "MDEntryPx": "2180.25", "MDEntrySize": "125" }
        ]
    }
}
```

3.7.1 Empty group

An empty repeating group may be serialized to simplify conversion to/from other FIX encodings.

3.7.2 Nested groups

A JSON object for a repeating group entry may contain other objects to represent nested repeating groups.

© Copyright, 2016, FIX Protocol, Limited

4 Sample Messages

MarketDataSnapshotFullRefresh

```
{
  "Header": {
    "BeginString": "FIXT.1.1",
    "MsgType": "W",
    "MsgSeqNum": "4567",
    "SenderCompID": "SENDER",
    "TargetCompID": "TARGET",
    "SendingTime": "20160802-21:14:38.717"
  },
  "Body": {
    "SecurityIDSource": "8",
      "SecurityID": "ESU6",
      "MDReqID": "789",
      "NoMDEntries": [
        { "MDEntryType": "0", "MDEntryPx": "1.50", "MDEntrySize": "75", "MDEntryTime": "21:14:38.688" },
        { "MDEntryType": "1", "MDEntryPx": "1.75", "MDEntrySize": "25", "MDEntryTime": "21:14:38.688" }
      ]
  },
  "Trailer": {
  }
}
```

5 Unsupported Features

5.1 Metadata

This user guide only specifies wire format. There is no provision for external or internal metadata features.

5.2 Versioning

Explicit versioning is not supported by the JSON encoding of FIX. Generally, this is unnecessary since web sites serve client-side code all with page contents. Since the server controls encoding on both sides, they should always be internally consistent.

© Copyright, 2016, FIX Protocol, Limited