High Performance FIX for the Web Era

FIX Semantics Get a Bidirectional Push over the Web

By Don Mendelson, FIX Technical Architect

Silver Flash LLC donmendelson@silver-flash.net

September 2018

Updated 9/26/2018 5:06 PM

Contents

Abstract

FIX Protocol is by most measures wildly successful—it has spread internationally, and it covers the semantics of trading, pre-trade and post-trade phases of the financial industry. At the same time, the implementation of FIX was conceived in an era when Internet protocols had enabled communication between firms, but when the world wide web was still immature. As the web matured, the FIX technical protocol did not keep pace with the high-performance needs of trading, although that has been mitigated in recent years with binary encodings, a low-latency session layer, and adaptations to ensure security.

The Fintech revolution spawned by alternative payment mechanisms and the creation of cryptocurrencies and Initial Coin Offerings ("ICO") has produced a proliferation new web-based systems. Firms attempted to bridge the gap with REST APIs that software developers are familiar with, but a synchronous request/response architectural style is a poor fit for bidirectional communications needed for trading. Therefore, cryptocurrency venues are beginning to offer WebSocket-based APIs instead of RESTful, which provides the asynchronous messaging required for trading. However, the messaging semantics are different from one venue to another.

The most valuable aspect of FIX is its standardized business semantics. FIX messages need to be conveyed by underlying protocols with two-way asynchronous message push. Fortunately, FIX semantics can be conveyed by mature, familiar web-based protocols, meeting trading performance requirements without relying on proprietary or esoteric means. The Conga project proves that the valued and standardized FIX semantics can be readily brought into the modern Web technology stack without sacrificing latency or high transaction rates.

A Survey of FIX Use Cases and Technology Requirements

Trading

Semantics

The term *semantics* refers to the meaning or content of a message rather than its form. FIX has standardized many order and execution messages for different purposes. They fall into these categories:

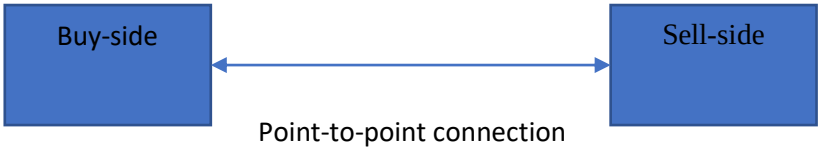
- Single General Order Handling—handle one order at a time
- Program Trading—define and execute lists of orders, such as stock baskets
- Order Mass Handling—enter or cancel multiple orders at a time, e.g. ladders
- Cross Orders—present a pre-matched buyer and seller
- Multileg Orders—spread trading, especially option strategies

Orders flow from buy-side (investors or fund managers) to sell-side (exchanges or firms that provide execution services) often making multiple hops along the way through various smart order routers, internal order matching, and multiple execution venues, and executions flow in the reverse direction. Order flows may also be categorized by the role of an order originator. Originally, program trading meant using a computer program to trade, and it was first popularized with basket trading. Now days, however, algorithms are commonly applied to any order flow.

Message Delivery Requirements

Trading sessions are private, so they are implemented with point-to-point communication protocols. One exception is drop copy, a flow of duplicate execution reports to a trader's risk guarantor. That may be thought of as a logical multicast although the copy is generally implemented at the application layer rather than session layer.

To maintain a correct accounting of securities positions, it is essential to have guaranteed delivery of execution reports. Errors could result in market risk if, for example, a position was not hedged when it was thought to be. Guaranteed delivery means if a communication break occurs, then missed messages must be persisted by the sender and retransmitted when communication is re-established. The cost of guaranteed delivery is delay. That cost is justified for execution reports, but it may not be for orders. In very fast-moving markets, the delay inherent in a retransmitted order message may miss a market opportunity or result in an unprofitable trade. Therefore, some traders prefer *idempotent* message delivery rather than recoverability. Idempotency means that a message is delivered at-most-once. That is, duplicate actions are prevented that would result in over-filled orders, but there are no retry



Point-to-point connection

Bidirectional flow

Optimistic delivery

attempts.

Performance Requirements

Tolerance for latency in order message delivery is dependent on trading strategy. Trades based on fundamental analysis are somewhat tolerant while the profits of high frequency trading are dependent on shaving latency to the minimum, often referred to as time or latency arbitrage colloquially. This has led to a technology arms race, sometimes aided by hardware acceleration.

Performance requirements are different across asset classes. Equities, listed derivatives, and FX trading require low latency and high transaction volumes. However, latency sensitive trading has come to fixed income and the on-venue swaps markets as high frequency trading firms applied their trading infrastructures to these asset classes.

It is the responsibility of venues to provide a level playing field for market participants. Nevertheless, they can provide a communication interface that eliminates unnecessary complexity and delays. An ideal system permits the clever to succeed without breaking rules. Factors that allow for low latency messaging include:

- Small message size achieved by eliminating redundant information and using an efficient wire format.
- Reduced duplication of effort performed by layers of the protocol stack.

Security Requirements

Market participants must have confidence that their communications are private, free from snooping by competitors, or sabotaged by malicious actors. Since traders are financially responsible for their actions, each participant in a session must be authenticated. That is, they must be able to prove who they claim to be. Additionally, non-repudiation of messages is highly desirable in communication protocols. This means that it can be proven that a message originated with an authenticated participant and that the message has not been altered.

Pre-Trade Flows

Semantics

These are the categories of pre-trade information flows:

- Indication
- Event Communication
- Quotation Negotiation
- Market Data
- Securities Reference Data
- Parties Reference Data
- Parties Action

These pre-trade message flows can be divided into three patterns of behavior.

- Reference data (Securities Reference Data, Parties Reference).

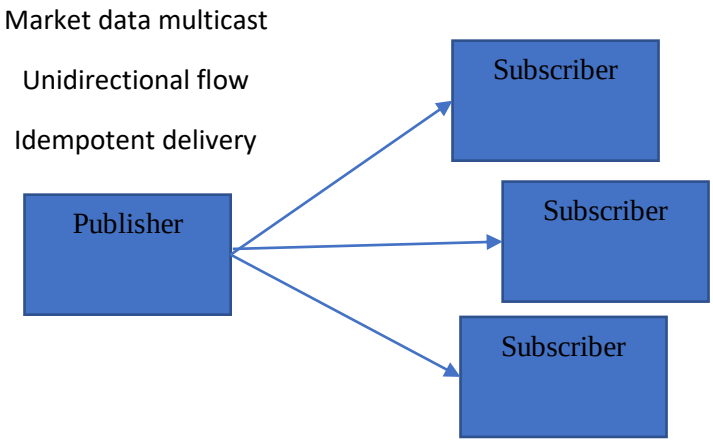
- Market data (Market data: trade, order book, trade volume, market highs and low; Venue and instrument trading status; Streaming quotes; some request for quote models).
- Quotation and Negotiation (Markets where quotes and responses to quotes are used to obtain an executable price between counterparties). Quotation and negotiation have messaging dynamics similar to trading.

Message Delivery and Performance Requirements

The message delivery requirements are quite different for reference data compared to the other flows. Reference data is usually less time sensitive, and if a query fails, it can be retried until it succeeds.

On the other hand, indications and quotes may be actionable, so their message delivery requirements are more like trading use cases described above.

Market data is highly time sensitive, and it is intimately tied to trading across the vast majority of asset classes covered by FIX: equities, listed derivatives, FX, fixed income, and swaps. Trading applications must minimize the latency of tick-to-trade, the elapsed time from receiving a market data message, deciding on strategy, and sending an order or cancel message. Market data is often transmitted on a multicast since all subscribers receive the same data. Protocols use message sequence numbers to detect gaps caused by lost data. However, multicast is one-way and thus, not recoverable in-band. Some services provide a side channel for recovering missed messages.



Post-Trade Flows

Semantics

These are the categories of post-trade information flows:

- Allocation
- Settlement Instruction
- Registration Instruction
- Trade Capture
- Confirmation
- Position Maintenance

- Collateral Management
- Margin Requirement Management
- Account Reporting

Message Delivery Requirements

Clearing flows require guaranteed message delivery since they impact security ownership and money transfers. However, since performance generally does not need to be real-time, direct connection is not always required. Instead, implementations sometimes employ a queueing service to act as a message-routing intermediary between parties.

Performance Requirements

Post-trade message flows do not have the low latency requirements of trading systems across all asset classes. Performance is usually measured in terms of transaction throughput.

Evolution of the FIX Protocol Stack

It is useful to analyze network protocols using the Open Systems Interconnection model (OSI). Although highly idealized, it helps think about separation of concerns.

Layer	Responsibilities
Application	Semantics, peer identity, synchronization
Presentation	Message syntax, framing, encryption
Session	Message dialogue, checkpointing, recovery
Transport	Connection, multiplexing, flow control, message segmentation and reassembly
Network	Address, routing
Data Link	Node-to-node datagram transfer
Physical	Electrical signaling, mechanical

The FIX 4 Protocol Stack

FIX 4.x is a monolithic protocol covering application semantics, message encoding, and session layer. It was quickly adopted, but the fact that FIX didn't initially isolate concerns eventually became an impediment to adaptation to technical change.

Layer	Protocol
Application	FIX 4.x
Presentation	
Session	
Transport	TCP
Network	IP
Data Link	LLC
Physical	Ethernet

At the application layer, FIX defines such things as how order quantities and execution quantities are accounted. The message definitions make trading concepts concrete. For example, the roles of trade participants are enumerated including executing broker, client, clearing firm, and so forth.

At the presentation layer, FIX originally consisted of tag value encoding. A message was encoded with ASCII characters. Each field consisted of a tag number followed by '=' and the character string value of the field, then delimited by an ASCII Ctrl-A control character. Tag numbers are standardized in FIX documentation.

In tag value encoding, all message structural features are performed bottom-up from the field level. Repeating group structures (arrays of records within a message) are inferred from the field that begins a record. There is no other group delimiter. This field-level technique extends to message framing (message delimiters). Messages begin with field BeginString (tag 8) and end with CheckSum (tag 10).

Nullness of an optional field is expressed in tag value encoding by lack of presence of the field in a message. It should be said that a strength of tag value encoding is its capacity for extension. New fields can later be added to a message type without breaking parsing of older messages. The obverse side of this coin is high variability of message layouts is detrimental to performance.

At the session layer, FIX 4.x is a point-to-point protocol with recoverable delivery guarantee in both directions (symmetrical). Every message in each direction has a sequence number contained by a standard message header. Both application and session messages consume a sequence number. Logon and logout messages initiate and terminate a session. A FIX session is persistent—a trader may suspend a session by logging out and then restart it later where it left off. If any messages were missed in a disconnect, they may be requested by sending a ResendRequest message for a range of application messages. Retransmitted messages have a PosDupe flag set in the message header. As a keep-alive, Heartbeat messages are exchanged during idle periods. Session messages use the same tag-value wire format as application messages.

One innovation of the FIX session protocol, in contrast to earlier financial industry protocols, is that it is asynchronous with optimistic message delivery. In other words, it does not depend on an acknowledgement of a message before sending the next. This promotes high performance. Rather than synchronous response, it detects errors and recovers asynchronously.

Theoretically, FIX is transport independent, but trading interfaces are almost always implemented atop a TCP/IP transport stack.

FIX 5 Protocol Stack

FIX 5 is much like FIX 4, but one improvement was made in the stack documentation—the session layer was isolated in a separate specification called FIXT. However, the session layer was still tightly bound to tag value encoding, and the StandardMessageHeader is common to both session and application messages.

One other change was made. Since users were reluctant to change FIX versions, FIXT allowed mixing of FIX application versions on a session. Although that change was intended to make FIX more flexible, it was one of the factors that led to a proliferation of incompatible interfaces.

Layer	**Protocol**
Application	FIX 5.x
Presentation	
Session	FIXT
Transport	TCP
Network	IP

Layer	Protocol
Data Link	LLC
Physical	Ethernet

Post-Trade Stacks

Typical Listed Derivatives Stack

The post-trade stack for listed futures and options typically uses XML as the message encoding. FIXML consists of XML Schemas (XSD) for message definitions that share the same semantics as tag value encoding. An advantage of using XML is that it is a broadly supported industry standard; parsers are provided on practically every computing platform. Like tag value encoding, XML messages consist of character strings, so they are not very low latency, and are highly variable in size and element order. However, XML is relatively easy for operations staff since messages are humanly readable.

Layer	Protocol
Application	FIX 5.x
Presentation	XML – FIXML Schema
Session and Transport	Queueing service, possibly proprietary

Typical Stack for Other Asset Classes

Buy-side to sell-side post-trade operations include confirm, affirm and allocation. Communication typically operates over the FIX 4 session layer.

Layer	Protocol
Application	FIX 4.2 or 4.4
Presentation	
Session	
Transport	TCP
Network	IP
Data Link	LLC
Physical	Ethernet

Typical Market Data Stack

One step that was taken towards high performance was the development of FIX FAST, an encoding optimized for market data dissemination. Its main goal was reduction of bandwidth utilization because at that time, network capacity was quite expensive. FAST had reasonably good latency characteristics, but that was a secondary goal.

Layer	Protocols

Layer	Protocols
Application	FIX 4.x or 5.x
Presentation (binary encodings)	FIX FAST
Session	(none—not recoverable in-band)
Transport	UDP Multicast

Not for Low Latency

In recent years, it became apparent that the FIX 4 and 5 protocol stacks were no longer fit-for-purpose for high performance trading, although they are still useful for other use cases.

- Tag value encoding is verbose, and its performance is non-deterministic due to variability in field order and message size.
- ASCII encoding requires unnecessary translation to and from native data types.
- The standard message header is highly redundant and verbose yet largely unnecessary. Message integrity features are redundant with features at the transport layer.
- Message framing is complex.
- In-band retransmissions can block real-time message flow.
- Delivery guarantees are not configurable; bidirectional recoverability may not be desirable.

The FIX High Performance Stack

In recent years, the FIX High Performance Working Group has responded to the demand for a fit-for-purpose protocol stack for trading. The good news for developers and operational staff is that the most valuable aspect of FIX, the application semantics, is retained. All the layers below that are optimized for low latency.

A guiding principal of the high-performance stack is that each layer should be isolated from the others to reduce inter-dependencies. This should allow protocols to be substituted and to allow development to respond to new requirements at each layer, in contrast to the monolithic FIX 4 protocol.

Layer	Protocols
Application	FIX 5.0SP2 semantics with optimizations
Presentation (binary encodings)	Message encodings: SBE, ASN.1, GPB Framing: Simple Open Framing Header
Session	FIXP
Transport	TCP (stream) or UDP (packet-oriented)

Binary Encodings

Simple Binary Encoding (SBE) has been developed as a FIX standard, and it has found acceptance in high performance use cases, especially in market data, but also order entry. Two other binary encodings are in release candidate status, FIX mappings to ASN.1 and Google Protocol Buffers.

The most fundamental improvement to performance is changing from ASCII characters on the wire to binary encodings. Binary values are what computers use internally. Constant translation between character data and native formats is a complete waste of resources from a performance perspective, resulting in unnecessary latency. The number of processor cycles devoted to those transformations is substantial for tag value and other character string encodings, such as XML or JSON.

Speed of encoding and decoding is not the only performance goal, even though binary encodings are two or three orders of magnitude faster than tag value. Just as important as speed is deterministic performance. In other words, very low variability. Traders want to be treated equally and want to have predictable results of their actions. Random snags in performance kills confidence in a system. In the extreme, they engender suspicions of favoritism.

One of the ways that SBE achieves deterministic performance is by controlling message layouts with templates. The position of a field in a message structure is largely predetermined, so not only are read and write operations very predictable, it is unnecessary to send much information about field position on the wire. Rather, parties exchange message schema files beforehand rather than figuring out operations on the fly, as is necessary with tag value encoding. Typically, developers generate highly optimized code for message encoders and decoders from the templates. Predefined message templates are also conducive to hardware acceleration.

Templates can be narrowly targeted for business use cases, and the best practice is that they should contain only the fields required for that use case. For example, a template can be defined for ExecutionReport due to a trade containing LastPx and LastQty versus a template for ExecutionReport for order booked, which would not contain trade price and quantity. Having one general template for both use cases would waste memory and CPU cycles for price and quantity in the booked case.

FIX Performance Session Layer

FIX Performance Session Layer (FIXP) is a FIX standard under development. It was specified to be independent of the presentation layer; session messages can be encoded in any binary encoding. It is also truly independent of the transport layer. FIXP can be used with either point-to-point or multicast transports, and they can either be stream or datagram (packet) oriented.

One way that FIXP achieves both layer independence and low latency is that unlike the original FIX protocol, the session layer imposes no message header on application messages. To the session layer, an application message is an opaque bag of bytes to send or receive; its internal structure is of no concern. Hence, no speed bump to crack the message in order to deliver it.

One issue that the FIXP standard addresses is setting appropriate message delivery guarantees. It is obvious that too low a guarantee can lead to data loss. It may be less obvious that over-promising can cause a performance hit without a benefit. Therefore, FIXP makes the level of guarantee negotiable at session start-up, and it can be independently set for flows in each direction. A typical trading use case is to send outbound executions over a recoverable flow while inbound orders are idempotent. This means

that executions are guaranteed to be delivered, with retransmissions if necessary, while order transmissions only make one attempt. The reason is that in a fast-moving market, there isn't time for recoverability; by the time a retransmission succeeds, the market may have moved. When FIXP performs retransmissions on a recoverable flow, it mitigates latency by imposing a flow control technique. Real time messages are not subject to back pressure, however.

FIXP specifies both point-to-point sessions between peers and also multicasts from one publisher to many subscribers. The latter is conducive to market data and reference data where all the receivers get the same information. The benefit is that you don't need to use totally different protocols for the two use cases.

FIXP also supports multiplexing of message flows over a shared transport. For example, multiple algorithms running for the benefit of the same trader can share a connection to an exchange. Each one has its own flow of orders inbound and executions out without blocking each other.

Recent Technology Trends and Applicability to FIX

As was stated earlier, the most valuable part of FIX is its stable business semantics while technologies come and go. The financial industry needs to periodically assess the suitability of new technologies for conveying FIX semantics.

JSON Encoding

JavaScript Object Notation (JSON) is pervasive in systems today because it is very conducive to client-side JavaScript code that runs in the context of web browsers. Web developers are so familiar with JavaScript and JSON that they are now common in server code, and JSON is popular across programming languages. It is fair to ask whether JSON should be used to encode FIX semantics. Indeed, it is possible, and we formed a FIX working group on JSON Encoding. So why isn't it more popular as a FIX encoding?

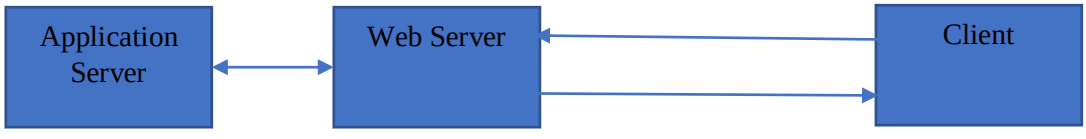
Some possible reasons:

- Like tag value and FIXML encodings, JSON is character based, not binary. Therefore, it does not have a clear performance advantage over established FIX tag value and FIXML encodings.
- JSON lacks the rigor of XML. In fact, that is its popularity driver. JSON is most commonly used without a schema for validation of messages, but the resulting variability can work against it in financial APIs. JSON itself lacks a precise specification like XML and other standards. Meanwhile, XML has a richer set of datatypes that can more closely map to FIX datatypes, and it has a complete schema specification and other supporting technologies such as Extensible Stylesheet Language Transformations (XSLT).
- The JSON specification contains the statement "JSON is agnostic about the semantics of numbers." This is shockingly sloppy for usage in the financial industry where numeric precision is of utmost importance.

Nevertheless, JSON may be suitable for client-side UI presentation since displayed values are ultimately converted to character strings anyway. The main use case that the FIX JSON Encoding Working Group tackled was translation from FIX tag value encoded messages to JSON for web display.

REST APIs

Representational State Transfer (REST) is a system architectural style, not a technology or standard. Nevertheless, it has become the dominant principle for the design of transactional, web-based systems. Although it is a theory or abstraction rather than a protocol (its proponents constantly remind you), common practice is intimately linked with web technologies, especially HTTP. In fact, a critique of REST is that implementations typically amount to RPCs over HTTP.



Synchronous

HTTP Request/Response

Negotiated Security

By no means is this a tutorial on the subject, but a few key points to know about REST:

- Each resource is identified by globally unique Uniform Resource Identifier (URI).
- Requests invoke operations on resources, returning a response. In practice, the operations are coupled to HTTP verbs POST, PUT, GET, and DELETE to create a new resource, update one, query a resource or list of resources, or to delete one. Because of the link to HTTP as a transport, there is no way for a server to push information to a client; the client must request it. The usual work-around is to poll for changes in a timer loop.
- An oft-stated principle of REST is called Hypermedia As The Engine Of Application State (HATEOAS). This abstraction means that the client need not know in advance all the possible responses to a request. The response itself is supposed to provide the next possible actions. When you create a resource, for example, the response should contain the URI of the new resource. In theory, APIs need not be versioned, and the resources and responses can be dynamically generated. Actual practice usually falls well short of theory, however. A major difficulty is how to test a highly dynamic interface—how do you even know what the test cases will be? As a practical matter, firms statically declare their APIs using Swagger (formally known as OpenAPI) and the like, contrary to the preached theory.
- Frequently, a returned resource is used to populate a web UI, but the style can also be used in a client/server mode without a UI (web services).
- The payload of operations to create or return a resource can theoretically be encoded in any format, but by far, the most common one is JSON.

Which FIX use cases, if any, are suitable for RESTful APIs? It works for cases when a client creates a resource or queries for resources and when information is not highly time sensitive. The information flows most conducive to this architectural style are pre-trade reference data and post-trade transactions.

REST is completely unsuitable, however, for applications that require asynchronous or event-driven flows of information. Foremost, market data and trading applications should not be implemented with REST since executions are asynchronous and very time-sensitive.

WebSocket and HTTP/2

I lump these two protocols together since they both seek to overcome the HTTP limitation of one-way communication in request/response style. Both support unsolicited messages pushed back from server to client. Both WebSocket and HTTP/2 are specified by industry standards and are supported on all popular platforms.

A WebSocket session is initiated as an HTTP request for a protocol upgrade. Because it starts out as a HTTP handshake, WebSocket traffic can be managed in familiar ways to network operators. Furthermore, it can negotiate all the security features afforded by Transport Layer Security (TLS, formerly known as SSL).

Once the protocol upgrade is accepted, WebSocket breaks free of the request/response paradigm and becomes an asynchronous messaging protocol. That is, either side can send unsolicited messages at any

time, like the original FIX session protocol. The one additional service provided by WebSocket is framing of messages over the underlying stream transport. Messages are therefore discrete events.

The original HTTP protocol opened and closed connections for each request/response transaction. That is generally acceptable for REST APIs but is very chatty and inefficient for long-running exchanges of information. Like WebSocket, HTTP/2 can keep a session open so subsequent requests can be sent over an established session. Moreover, a server can anticipate a request and send an unsolicited response. In fact, requests and responses can be interleaved giving the effect of multiple simultaneous channels over a shared transport (multiplexing like FIXP).

Unlike WebSocket, HTTP/2 retains HTTP syntax even after initial contact. Each request or response frame has standard HTTP headers, although they are now binary encoded and compressed. HTTP/2 is akin to HTTPS since implementations *only* work over TLS and not unsecured transports. HTTP/2 is likely to gradually replace the original HTTP/1.

WebSocket and HTTP/2 are suitable for applications that are time-sensitive and event-driven, such as trading and market data.

The Future of FIX on the Web

Requirements

The vision for a new protocol stack is that any competent software developer should be able to put together a FIX system using well-known web technologies and a bit of glue. Transactional applications can be architected as REST APIs, but there is an unmet need for a protocol with asynchronous push for trading and market data applications.

The ideal FIX protocol stack should have these characteristics:

- Convey familiar FIX application-layer semantics (message meaning).
- Use a high-performance message encoding for latency sensitive trading applications or browser-friendly JSON to be consumed by web UIs.
- Use web-friendly session and transport layers. System designers should be able to control message delivery guarantees.

A Proposed Stack

This proposed protocol stack combines proven standards, some from the set of Internet protocols, to meet the requirements stated above.

To meet these needs, we leave behind traditional FIX engines along with their FIX 4 and FIX 5 protocol stacks. (They can forever continue to meet the needs for which they are fit-for-purpose.) This is not meant to disparage what came before, but rather to recognize that it is time for the next step in a natural evolution.

Layer	**Protocol**
Application	FIX Latest or bespoke message specification
Presentation	Message encodings: SBE for low latency JSON for UIs and peripheral client integration Framing: WebSocket
Session	Message delivery and recovery: FIXP Negotiation: HTTP Security: TLS
Transport	TCP

Application Layer

The plan is to no longer introduce major abrupt version changes in FIX, but rather to issue many small, continual updates. When the Global Technical Committee accepts a gap analysis document with proposed message changes, they are incrementally added to the standard as an Extension Pack. The repository version with the most recent Extension Pack is now called *FIX Latest.* Soon, FIX message and field definitions will be stored in Orchestra format (a superset and successor to FIX Unified Repository).

Users may either use the published *FIX Latest* or edit an Orchestra file that represents their own rules of engagement and share it with their counterparties. It can represent the precise message layout for each use case of a message type. For example, an Execution Report for a trade looks slightly different than one for a booked order or rejection. Each use case of a message type is called a scenario. In addition to message structures, Orchestra can express workflow and rules about accepted field values.

Presentation Layer

WebSocket supports both text and binary message encodings. WebSocket frames messages on the wire, so an external framing protocol such as SOFH is not needed.

To reduce latency, a binary message encoding should be used. It obviates the need for translation back and forth between a computer's native data format and a character-based representation. Simple Binary Encoding yields deterministic performance due to template-enforced message layouts. If desired, other binary encodings could be used, e.g. Google Protocol Buffers.

Alternatively, if clients are browser-based, JSON encoding could be used to facilitate UI development.

One purpose of the machine-readable rules of engagement is to generate message templates in the selected template format. Generation of JSON schema files from Orchestra has already been demonstrated, and the same could be done for SBE message schemas.

Session Layer

A WebSocket session is initiated when a client sends an HTTP request with a header field requesting protocol upgrade. The HTTP handshake proceeds normally, including TLS handshake. Thus, standard security mechanisms are supported, including authentication and cipher suites. A cipher suite is responsible for key exchanges, encryption of messages, and non-repudiation by means of a message authentication code (MAC).

Once the server accepts the protocol upgrade request, the remainder of the session consists of asynchronous messages. HTTP no longer plays a role, so messages need no HTTP headers or any other interference from the session layer. However, negotiated TLS algorithms still operate to protect security.

Since WebSocket works over a TCP transport, a degree of delivery reliability is inherent. However, TCP makes no guarantees if the connection breaks or is deliberately suspended and is later restarted. FIXP provides such delivery guarantees for what is called a durable session. After re-establishing a session, message exchange resumes from the last checkpoint as evidenced by message sequence numbers.

Another shortcoming of bare TCP is that the interval to detect a connection break during an idle period is unacceptably long for high performance trading. To minimize risk, a trader must always be assured of a live connection. WebSocket uses Ping/Pong frames as keep-alives. Unfortunately, the standard does

not make specific recommendations for their intervals, and behavior of implementations is usually not transparent or configurable. Therefore, we use the heartbeats of the FIXP since their intervals are configurable in session negotiation, even though they partially overlap the function of Ping/Pong. FIXP heartbeats have added value because they convey the next application message sequence number, so message gaps and communication breaks can be detected quickly.

Security

TLS installs cipher suites that continue to operate for the duration of a session. One concern for high performance traders is the latency introduced by message encryption and decryption. However, the cost is born equally by all participants, leaving no one at an advantage or disadvantage. We cannot afford to compromise privacy for the sake of latency.

Project Conga

Project Conga is an open source project sponsored by FIX Trading Community as a proof of concept for the proposed protocol stack. It demonstrates high performance FIX semantics over WebSocket with SBE. The prøject is public in GitHub, so anyone can use it as the starting point for developing a real application. User contributions in the form of issues and code are welcome.

The test application in Conga is order entry and a simple match engine. For demo purposes, the messages only contain the fields necessary for matching, not for back office processing. The only messages inbound are NewOrderSingle and OrderCancelRequest while outbound messages are ExecutionReport and OrderCancelReject. However, a user who wishes to expand this into a real application could substitute their own message definitions in the form of an SBE message schema. Naturally, a real match engine would need to handle more complex business logic. But this a technology demonstration, not a full-fledged application.

Point-to-point connection

Bidirectional flow

Idempotent or Recoverable delivery

Negotiated Security

On the server side, WebSocket sessions are managed by an embedded web server. The communications protocol handlers are in the same process as the business logic. This avoids an extra hop from a separate web server to an application server. Since the protocol is initiated with an HTTP request, TLS negotiation is available for free. In Conga, the server side is authenticated by a certificate. Configuration

enhancements could provide mutual authentication with certificates on both sides or through pre-shared keys.

Once a WebSocket session is negotiated, messaging is totally asynchronous.

The session layer is enriched with FIXP session messages. Each direction of a session may be configured as either an idempotent or a recoverable flow. Both depend on sequence numbers of messages in each direction. Idempotency simply prevents duplicate messages from being processed while a recoverable flow fills gaps through retransmissions. For message recovery, an in-memory cache of recently sent messages is provided. In the initial implementation, the cache is not persisted, so it does not survive application failure.

At this point, Conga only demonstrates SBE messages, but the communication code could be reused with a substitution of JSON or other encoding since care was taken to abstract out message encoding.

Possible Future Enhancements

- Use alternative encodings, e.g. Google Protocol Buffers, or JSON (not high performance). A JSON plug-in is under development.
- Enhanced client authentication. The FIX-over-TLS recommendation offers several alternatives. One is mutual authentication with a client-side certificate or pre-shared keys. Another alternative is to only authenticate the server with a certificate, but after a secure pipe is established with TLS, to send message with client credentials (Negotiate message in FIXP). This avoids the management headache of distributing keys to clients. The Cybersecurity Working Group and its subgroups plans further recommendations on authentication. Project Conga should offer the recommended alternatives.
- Message persistence is generally needed by the sender of a recoverable flow in case of application failure and recovery. (Persistence by a receiver is not required by the protocol, but is likely needed for auditing.) Persistence has a significant performance cost as compared to in-memory caching of messages. Nevertheless, message logging and recovery technologies must be considered for a production system. Requirements include redundancy, and hardware-based solutions should be considered such as network taps. This is beyond the scope of this demonstration project.
- Session multiplexing. This enhancement is possible using Websockets/HTTP/2 but users would need to weigh its possible benefits versus the complexity in implementation when the alternative to simply using multiple transports is common and accepted practice. What is the cost of an additional WebSocket transport?

References

Project Conga is in GitHub at https://github.com/FIXTradingCommunity/conga. The project is public and open-sourced with Apache 2.0 license.

FIX Standards

Simple Binary Encoding (SBE) technical standards are published on the FIX web site at https://www.fixtrading.org/standards/sbe/.

FIX Performance Session Layer (FIXP) at https://www.fixtrading.org/standards/fixp/

FIX-over-TLS (FIXS) at https://www.fixtrading.org/standards/fixs/

FIX Repository and Orchestra at https://www.fixtrading.org/standards/fix-repository/

Other Protocol Standards

The WebSocket Protocol (RFC 6455) is a standard of Internet Engineering Task Force (IETF) at https://tools.ietf.org/html/rfc6455. See its References section for related Internet standards, including HTTP and TLS.

JSON is a subset of the JavaScript programming language, also known as ECMAScript. Standard ECMA-404 The JSON Data Interchange Syntax 2nd edition (December 2017) is published at https://www.ecma-international.org/publications/standards/Ecma-404.htm.