Exchange Stream API Message format

Overview

Exchange Streaming API provides the ability to subscribe to market changes (both price and definitions) and to your orders.

The protocol is based on ssl sockets (normal) with a CRLF json protocol.

We publish a definition of the schema of the json messages in the Swagger format.

We maintain sample code in Java & C# here: https://github.com/betfair/stream-api-sample-code

> **Json Deserializers**
> Expect fields & enum values to be added to json responses as and when the need arises; your deserialization code should be tolerant of such additive changes.

Sample application - C# & Java

A console based C# & Java sample application is available for the Market & Order Streaming API and is available via https://github.com/betfair/stream-api-sample-code

Users wishing to interact with the Streaming API using one of these languages are strongly advised to make use of this sample code.

Swagger Definition

For users wishing to use other languages or develop their own implementation, we provide a swagger schema to allow browsing & code generation.

We recommend using Swagger Code Gen (http://swagger.io/swagger-codegen/) for generation,

- As a pre-requisite Java version 7 or higher must be installed
- Download both,
    - The Swagger Code Gen jar from: https://oss.sonatype.org/content/repositories/releases/io/swagger/swagger-codegen-cli/2.2.1/swagger-codegen-cli-2.2.1.jar
    - The Swagger Definition from our GitHub repository: https://github.com/betfair/stream-api-sample-code/blob/master/ESASwa

ggerSchema.json

- Run the following command to view a list of available languages to generate code for: *java -jar swagger-codegen-cli-2.2.1.jar*
- Run the following command to generate the code: *java -jar swagger-codegen-cli-2.2.1.jar generate -i ESASwaggerSchema.json -l <LANGUAGE> -o <OUTPUT_DIRECTORY>*

The Swagger editor can also be used to view the domain model

- http://editor.swagger.io/#!/
- Use File -> Import File and choose the Swagger Definition downloaded from our GitHub repository

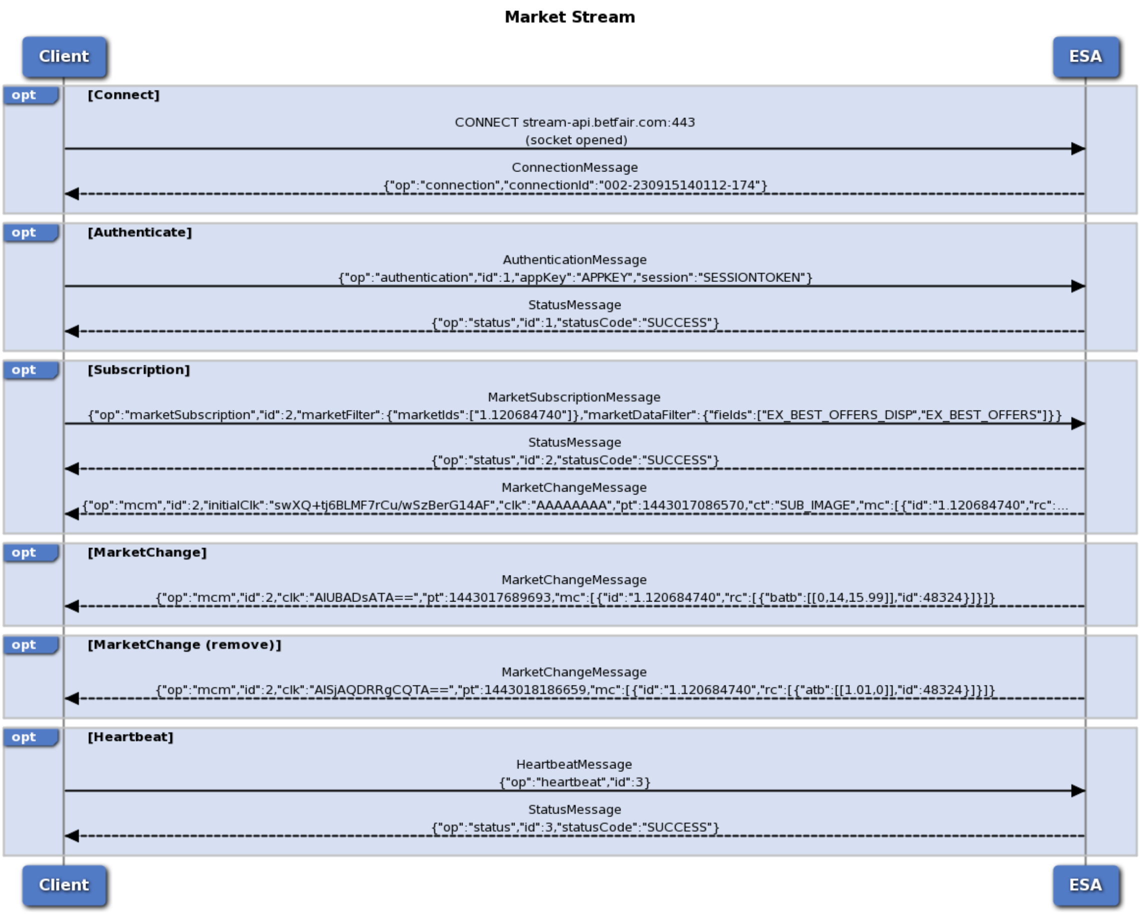
A few points to note with swagger:

- It's cross platform and we can't control how it works / behaves - but it does save a lot of error prone typing.
- Enums and Inheritance are a little flaky:
  - Enums for error codes / filters etc. are defined but are treated as strings in c# (so you will need to copy definitions from the swagger spec until this is fixed by swagger).
  - Inheritance is defined but not generated correctly - you will have to manually manipulate the op=<type> field
    - In c# JsonCreationConverter is the typical way to model inheritance
    - In java look at JsonSubTypes
- We are not a REST service - so only the swagger generated model package is relevant

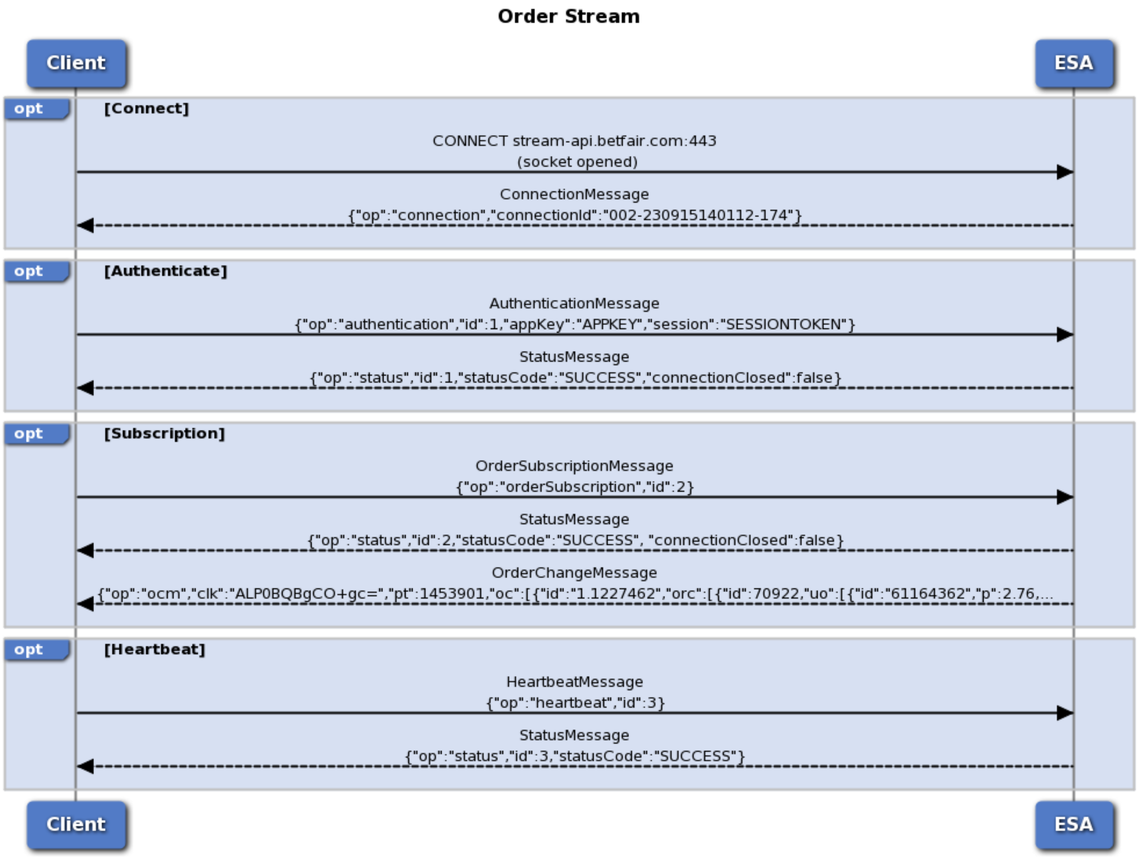
Typical Interactions with API:

The typical API interactions are documented below (detail is below this).

Market Stream:



Order Stream:



Connection

Protocol

Every message is in json & terminated with a line feed (CRLF):

```
{json message}\r\n
```

Json Serializer Setup
As the protocol is CRLF delimited don't forget to turn-off Json pretty printing (C# has this on by default)

TCP / SSL Connection

Connection is established with an SSL socket to the following address:

External (SSL):
`stream-api.betfair.com:443`

Early access to future releases and general developer testing should be conducted against this address (it has no HA and will have service gaps during release & as such should not be used for trading):

External Pre-Production / Integration (SSL):
`stream-api-integration.betfair.com:443`

Correct address to use
Do not use the integration address (stream-api-integration.betfair.com) for trading) - it has no backup!

Basic Message Protocol

Two base message classes exist:

- RequestMessage - These are messages sent to the server.
- ResponseMessage - These are messages received from the server.

Every child message type has:

- id - A unique counter you should supply on a RequestMessage and which will be supplied back on a ResponseMessage.
- op - This identifies the request type and may be used to switch / deserialize correctly

Note: Any fields representing time and having a long type will represent the UNIX Timestamps (See https://currentmillis.com/ for conversions)

RequestMessage

RequestMessage is the base class for requests from the client; the discriminator is op=<message type>

Key fields:

- op=authentication - The AuthenticationMessage - authenticates your connection.
- op=marketSubscription - The MarketSubscriptionMessage - subscribes to market changes.
- op=orderSubscription - The OrderSubscriptionMessage - subscribes to order changes.
- op=heartbeat - The HeartbeatMessage - use if you need to keep a firewall open or want to test connectivity.

RequestMessages
- Remember to set op=<message type> - otherwise we can't decode the request
- Remember to set id=<unique sequence> - this will let you link requests with responses (these should be logged and provided on support calls)
- Every RequestMessage will receive a StatusMessage with the status of the call (linked by the id that you send).
    - **All errors apart from SUBSCRIPTION_LIMIT_EXCEEDED close the connection**

ResponseMessage

ResponseMessage is the base class for responses back to the client; the discriminator is op=<message type>

Key fields:

- op=connection - The ConnectionMessage sent on your connection.
- op=status - The StatusMessage (returned in response to every RequestMessage)
- op=mcm - The MarketChangeMessage that carries the initial image and updates to markets that you have subscribed to.
- op=ocm - The OrderChangeMessage that carries the initial image and updates to orders that you have subscribed to.

ResponseMessages
As mentioned earlier the id=<request id> and links your request with your response.

ChangeMessages carry the id of the original request that established the subscription

Status / StatusMessage

Every request receives a status response with a matching id.

Key fields:

- statusCode - The status of the request i.e success / fail
    - SUCCESS - Call processed correctly
    - FAILURE - Call failed (inspect errorCode and errorMessage for reason)
- connectionClosed - Boolean set to true if the connection was closed as a result of a failure
- errorCode - The type of error in case of a failure - see the swagger spec / enum.
- errorMessage - Additional message in case of a failure

ErrorCode

This categorizes the various error codes that could be expected (these are subject to change and extension)

Category	ErrorCode	Description
Protocol		General errors not sent with id linking to specific request (as no request context)
	INVALID_INPUT	Failure code returned when an invalid input is provided (could not deserialize the message)
	TIMEOUT	Failure code when a client times out (i.e. too slow sending data)
Authentication		Specific to authentication
	NO_APP_KEY	Failure code returned when an application key is not found in the message
	INVALID_APP_KEY	Failure code returned when an invalid application key is received
	NO_SESSION	Failure code returned when a session token is not found in the message
	INVALID_SESSION_INFORMATION	Failure code returned when an invalid session token is received
	NOT_AUTHORIZED	Failure code returned when client is not authorized to perform the operation
	MAX_CONNECTION_LIMIT_EXCEEDED	Failure code returned when a client tries to create more connections than allowed to
Subscription		Specific to subscription requests
	SUBSCRIPTION_LIMIT_EXCEEDED	Customer tried to subscribe to more markets than allowed to
	INVALID_CLOCK	Failure code returned when an invalid clock is provided on re-subscription (check initialClk / clk supplied)
General		General errors which may or may not be linked to specific request id
	UNEXPECTED_ERROR	Failure code returned when an internal error occurred on the server
	CONNECTION_FAILED	Failure code used when the client / server connection is terminated

Connection / ConnectionMessage

This is received by the client when it successfully opens a connection to the server

Key fields:

- connectionId - This is a unique identifier that **you must supply for support.**

Initial ConnectionMessage
On establishing a connection a client receives a ConnectionMessage - the **connectionId must be logged & supplied on any support queries**:

```
{"op":"connection","connectionId":"002-230915140112-174"}
```

Authentication / AuthenticationMessage

This message is the first message that the client must send on connecting to the server - you must be authenticated before any other request is processed

Key fields:

- op=authentication - This is the operation type
- appKey - This is your application key to identify your application
- session - The session token generated from API login.

> **Common Authentication Errors**
> Some common authentication errors that you should handle - these are defined on ErrorCodes enum (these will all close your connection):
>
> - NO_APP_KEY / INVALID_APP_KEY - Check you are using the correct app key
> - NO_SESSION / INVALID_SESSION_INFORMATION - Check the session is current
> - NOT_AUTHORIZED - Check that you are using the correct appkey / session and that it has been setup by BDP
> - MAX_CONNECTION_LIMIT_EXCEEDED - Check that you are not creating too many connections / are closing connections properly.

Subscription / SubscriptionMessage

This message changes the client's subscription - there are currently two subscription message types:

- op=marketSubscription- MarketSubscriptionMessage which streams:
  - op=mcm - MarketChangeMessage - the price changes for a market
- op=orderSubscription- OrderSubscriptionMessage which streams:
  - op=ocm - OrderChangeMessage - the order changes for a market

On creating a subscription you will receive:

- StatusMessage confirming the status of your request
- A stream of ChangeMessages linked with the id of the request which is composed of:
  - Initial image
  - Deltas to the initial image

It is possible to subscribe multiple times - each replaces the previous (each will send a new initial image and deltas) - **they are not additive**.

Key fields on a SubscriptionMessage:

- segmentationEnabled=true - Segmentation will shortly be switched on by default (so please code with this set)
  - segmentation breaks up large messages and improves: end to end performance, latency, time to first and last byte
  - see the topic on change message segmentation for a full explanation of how this works.
- conflateMs - Specifies a forced conflation rate (in milliseconds)
- heartbeatMs - Specifies a minimum interval that a client would expect to receive a message (in milliseconds)
  - If no change is delivered in this interval then an empty change message will be sent with a ChangeType.HEARTBEAT
- initialClk & clk - these two sequence tokens allow for faster recovery in the event of a disconnection:
  - If supplied (with identical subscription criteria) you will receive a delta to your previous state rather than a full initial image
  - see the topic on re-subscription for a full explanation of how this works.

ChangeMessage

This message is the payload that delivers changes (both initial image & updates) to a client - there are currently two change message types:

- op=mcm - MarketChangeMessage
- op=ocm - OrderChangeMessage

Key fields on a ChangeMessage:

- ct= ChangeType - this enumeration is used to identify the type of change
  - SUB_IMAGE - The initial image returned from a subscribe
  - RESUB_DELTA - A patch returned from a resubscribe
  - HEARTBEAT - An empty message published if no data has been sent within heartbeatMs

        - We send these to maintain the connection to you and detect closed connections
        - You can use the heartbeatMs to verify that you are still connected
    - <null / not set> - An update message
- segmentationType - SegmentationType - this enumeration identifies multi-part segmented messages:
    - SEG_START - Start of a segmented message
    - SEG - Middle part of a segmented message
    - SEG_END - Last part of a segmented message
    - <null / not set> - A non-segmented message
- conflateMs - the actual conflation being used
    - This might be different to what you specified - if you account is for instance delayed or your request was out of bounds
- heartbeatMs - the actual heartbeat being used
    - This might be different to what you specified as we bounds check
    - You can use this to verify your connection is live (as you should receive 1 message within this time period).
- pt - publishTime - the time we sent the message
- initialClk & clk - these two sequence tokens allow for faster recovery in the event of a disconnection:
    - If we send these then they should be stored
    - see the topic on re-subscription for a full explanation of how this works.

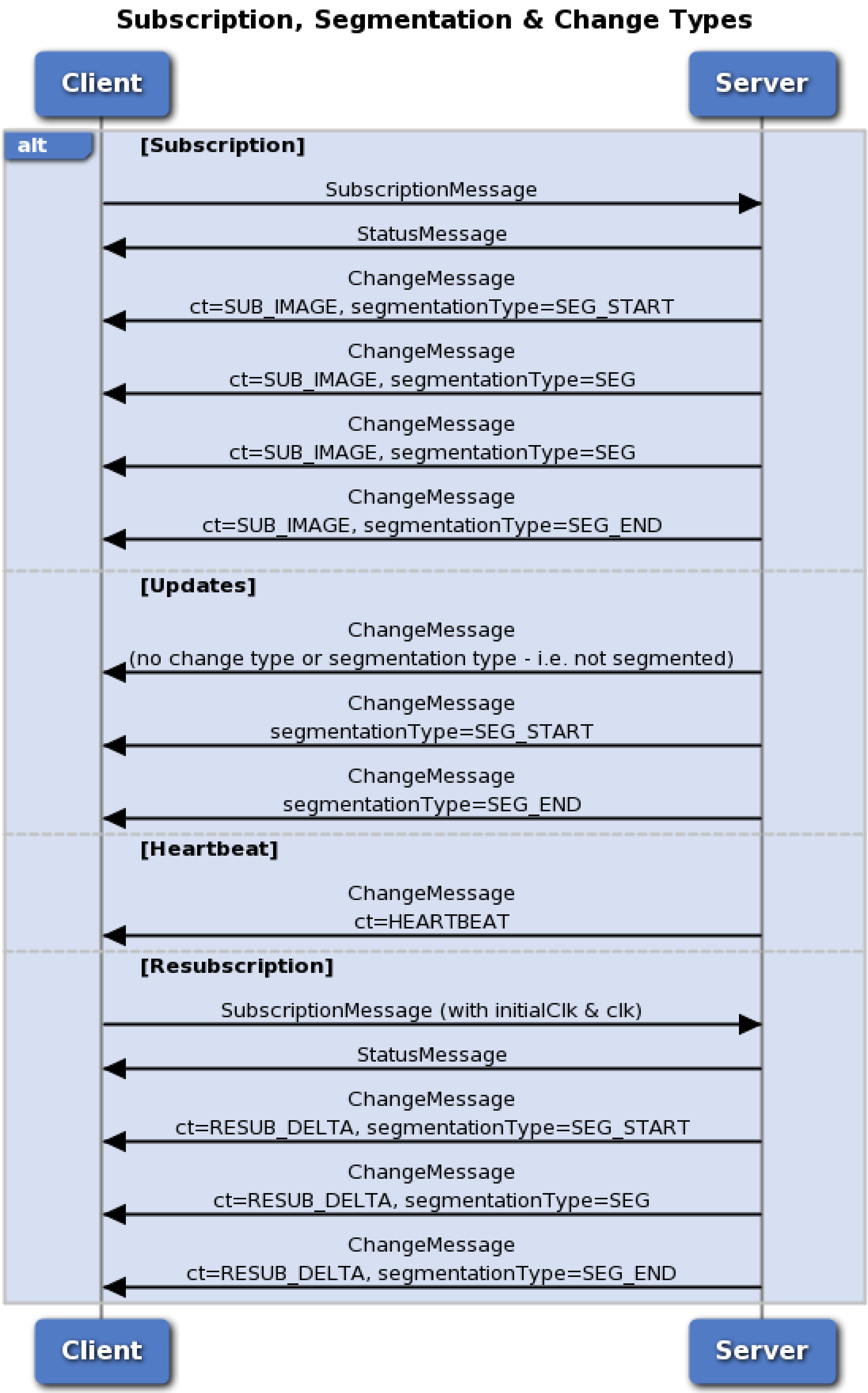
Heartbeat ChangeMessages
heartbeatMs is a guarantee of how often (even with no changes) you will receive a ChangeMessage; i.e.:

If heartbeatMs= 500 and your subscription has not changed in 500ms then we will send an empty ChangeMessage with ct=HEARTBEAT

(this verifies your connection is live and processing data)

Change Message Segmentation

The below shows the key interactions for subscription & changes with segmentation applied:



Typically on changing your subscription you will want to clear any local cache you maintain.

Initial Image Handling

- How can I detect the start of an initial image & clear my cache?
  - ct=ChangeType.SUB_IMAGE and segmentType=null or SegmentType.SEG_START indicates the start of a new

image

- How can I detect the end of an initial image?
  - ct=ChangeType.SUB_IMAGE and segmentType=null or SegmentType.SEG_END indicates the end of a new image
- When I change Subscription how do I safely ignore messages for a previous subscription?
  - All ChangeMessages carry have id=<request id> this allows safe disposal during subscription change

MarketSubscriptionMessage

This subscription type is used to receive price changes for one or more markets; your subscription criteria determine what you see.

Coarse vs Fine Grain Subscriptions
It is preferable to use coarse grain subscriptions (subscribe to a super-set) rather than fine grain (specific market ids).

- If you find yourself frequently changing subscriptions you probably want to find a wider super-set to subscribe to

A MarketSubscription has two types of filter:

- marketFilter - MarketFilter - this is a horizontal filter of markets that you require (i.e. rows)
- marketDataFilter - MarketDataFilter - this is a vertical filter of fields that you require (i.e. columns)

Limiting the amount of data that you consume will make your initial image much smaller (and faster) & suppress changes that are uninteresting to you.

Market Filtering / MarketFilter

As with the APING API users have the ability to filter the market data they get from the new Exchange Stream API (ESA).

All subscriptions are evaluated with a few default criteria:

- Standard jurisdictional filtering that restricts visibility (mirroring site behavior)
- Permissions that control:
  - Specific sports that you are entitled for
  - A maximum consumption limit (exceeding this will result in an error with details of the limit: ErrorCode.SUBSCRIPTION_LIMIT_EXCEEDED)

Users can then specify the following filters when they subscribe to ESA:

Filter name	Type	Mandatory	Description
marketIds	Set<String>	No	If no marketIds passed user will be subscribed to all markets
bspMarket	Boolean	No	Restrict to bsp markets only, if True or non-bsp markets if False. If not specified then returns both BSP and non-BSP markets
bettingTypes	Set<BettingType>	No	Restrict to markets that match the betting type of the market (i.e. Odds, Asian Handicap Singles, or Asian Handicap Doubles)
eventTypeIds	Set<String>	No	Restrict markets by event type associated with the market. (i.e., "1" for Football, "7" for Horse Racing, etc)
eventIds	Set<String>	No	Restrict markets by the event id associated with the market.
turnInPlayEnabled	Boolean	No	Restrict to markets that will turn in play if True or will not turn in play if false. If not specified, returns both.
marketTypes	Set<String>	No	Restrict to markets that match the type of the market (i.e., MATCH_ODDS, HALF_TIME_SCORE). You should use this instead of relying on the market name as the market type codes are the same in all locales
venues	Set<String>	No	Restrict markets by the venue associated with the market. Currently only Horse Racing markets have venues.
countryCodes	Set<String>	No	Restrict to markets that are in the specified country or countries

Currencies
Market subscriptions are always in underlying exchange currency - GBP

For example a subscription message with almost all filters enabled will look something like this:


```
{"op":"marketSubscription","id":2,"marketFilter":{"marketIds":["1.120684740"],"bspMarket":true,"bettingTypes":["ODDS"],"eventTypeIds":["1"],"eventIds":["27540841"],"turnInPlayEnabled":true,"marketTypes":["MATCH_ODDS"],"countryCodes":["ES"]},"marketDataFilter":{}}
```


Subscriptions with no matching markets
We don't verify your subscription criteria as you could potentially subscribe to either a wild card (which would include future markets) or a future marketid which we do not have yet but would send on arrival

Market data field filtering / MarketDataFilter

A market data filter restricts the fields that you get back (and only if the fields have changed).

Key fields:

- fields - A set of field filter flags (see below)
- ladderLevels - For depth based ladders the number of levels to send (1 to 10)

The field filter flags are defined as:

Filter name	Fields:	Type	Description
EX_BEST_OFFERS_DISP	bdatb, bdatl	level, price, size	Best prices including virtual prices - depth is controlled by ladderLevels (1 to 10)
EX_BEST_OFFERS	batb, batl	level, price, size	Best prices not including virtual prices - depth is controlled by ladderLevels (1 to 10)
EX_ALL_OFFERS	atb, atl	price, size	Full available to BACK/LAY ladder
EX_TRADED	trd	price, size	Full traded ladder
EX_TRADED_VOL	tv	size	Market and runner level traded volume
EX_LTP	ltp	price	Last traded price
EX_MARKET_DEF	marketDefinition	MarketDefinition	Send market definitions.
SP_TRADED	spb, spl	price, size	Starting price ladder
SP_PROJECTED	spn, spf	price	Starting price projection prices

Multiple field filters may be combined; a subscription message that contains data fields should look like the following:


```
{"op":"marketSubscription","id":2,"marketFilter":{"marketIds":["1.120684740"]},"marketDataFilter":{"fields":["EX_BEST_OFFERS_DISP","EX_BEST_OFFERS","EX_ALL_OFFERS","EX_TRADED","EX_TRADED_VOL","EX_LTP","EX_MARKET_DEF","SP_TRADED","SP_PROJECTED"]}}
```


Correctly configuring field filters
Correctly configuring field filters can help by:

- Reducing the size (and time) of initial images
- Reducing the rate of change (as only changes matching your field filter are sent)

MC / MarketChangeMessage

This is the ChangeMessage stream of data we send back to you once you subscribe to the market stream.

Key fields:

- <as for ChangeMessage>
- mc / MarketChange - this list of market changes contains the changes the markets that you have subscribed to.
    - img / Image - replace existing prices / data with the data supplied: it is not a delta (or null if delta)
    - marketDefinition / MarketDefinition - this is sent in full (but only if it has changed)
    - rc / RunnerChange - this is sent to supply the details of a runner (namely prices)
        - con / Conflated = true - if this is sent then more than one change is combined in this message (purely informational).

- Values - only sent if value has changed:
    - tv - Traded Volume
    - ltp - Last Traded Price
    - spn - Starting Price Near
    - spf - Starting Price Far
- Level / Depth Based Ladders (level, price, size - triples - keyed by level):
    - size=0 - indicates a remove
    - batb / batl - Best Available To Back / Best Available To Lay (non-virtual)
    - bdatb / bdatl - Best Display Available To Back / Best Display Available To Lay (virtual)
- Price point / full depth Ladders (price, size - tuples - keyed by price):
    - size=0 - indicates a remove
    - atb / atl - Available To Back / Available To Lay (these are the raw / full depth non-virtual prices)
    - spb / spl - Starting Price Back / Starting Price Lay
    - trd - Traded

Building a price cache

Most of the change based data (RunnerChange) is delta based - this means a few rules:

- img / Image - if this is set to true then you should replace this item in your cache
- Values - the values sent are nullable & are not sent if they are not changed (i.e. if tv has not changed then there will be no field in the message)
- Level / Depth Based ladders
    - [0, 1.2, 20] -> Insert / Update level 0 (top of book) with price 1.2 and size 20
    - [0, 1.2, 0] -> Remove level 0 (top of book) i.e. ladder is now empty
- Price point / full depth ladders
    - [1.2, 20] -> Insert / Update price 1.2 with size 20
    - [1.2, 0] -> Remove price 1.2 i.e. there is no size at this price

OrderSubscriptionMessage

This subscription type is used to receive order changes; the subscription message has one type of filter

- orderFilter (optional)

OrderFilter

This optional filter already filters by your account; but additional data shaping is supported

Filter name	Type	Mandatory	Default	Description
accountIds	Set<Integer>	No	null	This is for internal use only & should not be set on your filter (your subscription is already locked to your account).
includeOverallPosition	Boolean	No	true	Returns overall / net position (OrderRunnerChange.mb / OrderRunnerChange.ml)
customerStrategyRefs	Set<String>	No	null	Restricts to specified customerStrategyRefs; this will filter orders and StrategyMatchChanges accordingly (Note: overall postition is not filtered)
partitionMatchedByStrategyRef	Boolean	No	false	Returns strategy positions (OrderRunnerChange.smc=Map<customerStrategyRef, StrategyMatchChange>) - these are sent in delta format as per overall position.

OCM / OrderChangeMessage

This is the ChangeMessage stream of data we send back to you once you subscribe to the order stream.

Key fields:

- <as for ChangeMessage>
- oc / OrderAccountChange - the modifications to account's orders (will be null on a heartbeat)
    - id / Market Id - the id of the market the order is on
    - orc / Order Changes - a list of changes to orders on a runner
        - id / Selection Id - the id of the runner (selection)
        - hc / Handicap - the handicap of the runner (selection) (null if not applicable i.e. on non Asian Handicap markets)
        - uo / Unmatched Orders - orders on this runner that are unmatched
            - Every order change is sent in full; the transient on a change to EXECUTION_COMPLETE is sent (but it would not be sent on initial image)
            - id / Bet Id - the id of the order
            - p / Price - the original placed price of the order
            - s / Size - the original placed size of the order
            - bsp / BSP Liability - the BSP liability of the order (null if the order is not a BSP order)
            - side / Side - the side of the order

        - status / Status - the status of the order (E = EXECUTABLE, EC = EXECUTION_COMPLETE)
        - pt / Persistence Type - whether the order will persist at in play or not (L = LAPSE, P = PERSIST, MOC = Market On Close)
        - ot / Order Type - the type of the order (L = LIMIT, MOC = MARKET_ON_CLOSE, LOC = LIMIT_ON_CLOSE)
        - pd / Placed Date - the date the order was placed
        - md / Matched Date - the date the order was matched (null if the order is not matched)
        - ld / Lapsed Date - the date the order was lapsed (null if the order is not lapsed)
        - avp / Average Price Matched - the average price the order was matched at (null if the order is not matched
        - sm / Size Matched - the amount of the order that has been matched
        - sr / Size Remaining - the amount of the order that is remaining unmatched
        - sl / Size Lapsed - the amount of the order that has been lapsed
        - sc / Size Cancelled - the amount of the order that has been cancelled
        - sv / Size Voided - the amount of the order that has been voided
        - rac / Regulator Auth Code - the auth code returned by the regulator
        - rc / Regulator Code - the regulator of the order
        - rfo / Reference Order - the customer supplied order reference
        - rfs / Reference Strategy - the customer supplied strategy reference used to group orders together - default is ""
    - Price point / full depth Ladders (price, size - tuples - keyed by price) of matches:
        - mb / Matched Backs - matched amounts by distinct matched price on the Back side for this runner
        - ml / Matched Lays - matched amounts by distinct matched price on the Lay side for this runner

> **Currencies**
> Order subscriptions are always in the user's currency

Building an order cache

An order cache is somewhat simpler as orders are sent in full (on change) and only matches need delta merging

- img / Image - if this is set to true then you should replace this item in your cache
- Orders - replace each order according to order id.
- Price point / full depth ladders
    - [1.2, 20] -> Insert / Update price 1.2 with size 20
    - [1.2, 0] -> Remove price 1.2 i.e. there is no size at this price
    - An empty list of points also means the ladder is now empty

> **Unmatched Orders**
> New subscriptions: Will receive an initial image with only E - Executable orders (unmatched).
>
> Live subscriptions: Will receive a transient of the order to EC - Execution Complete as the order transits into that state (allowing you to remove the order from your cache).

Heartbeat / HearbeatMessage

This is an explicit heartbeat request (in addition to server heartbeat interval which is automatic).

This functionality should not normally be necessary unless you need to keep a firewall open.

> **Do I need to use HeartbeatMessage?**
> No - under normal circumstances the subscription level ChangeType.HEARTBEAT is an acceptable guarantee of connection health.
>
> Use the HeartbeatMessage only if you need to keep a firewall open - as it will incur some performance penalty (as a response will block your connection)

Re-connection / Re-subscription

If a client is disconnected a client may connect, authenticate and re-subscribe.

Prerequisite steps:

- Store your subscription criteria (re-subscribe will only work correctly with identical subscription critieria
- Store initialClk (normally only initial image) & Clk (normally on every non-segmented message or a SEG_END) on any change

message they are sent on.

Connection is broken.

- Connect & Authenticate as normal
- Subscribe setting initialClk and Clk to the last values sent on the subscription
- Change message with ChangeType.RESUB_DELTA is sent - this will patch your cache
- Some markets might have img=true set indicating they are either new or can't be patched.

Easiest way to implement re-subscribe

- Store any new subscription message you send as a "pending subscription"
- Store this as a "active subscription" once you get your initial image
- Update the initialClk & clk on the subscription message with any non-null values
- Resend this message after re-connecting

Performance Considerations

Here are a few tips on performance which are worth bearing in mind:

Performance tips

- A single market subscription & a subscription to all markets have an identical latency:
  - Cost is identical as the two subscriptions above would evaluate in sequence and thus with the same average latency.
  - Initial image is more costly to send than extra updates.
  - Limiting data with appropriate filters reduces initial image time
- Segmented data will always out perform non-segmented data:
  - You will be processing a buffer while another is in-flight and another is being prepared to send
- Writes to your connection are directly effected by how quickly you consume data & clear your socket's buffer
  - Consuming data slowly is effectively identical to setting conflation.
  - If you receive conf=true flag on a market - then you are consuming data slower than the rate of deliver

Currency Support

The Exchange Stream API supports GBP currency only.

Those looking to convert data from GBP to a different currency should use listCurrencyRates to do so.

Australian Exchange

Australian markets are now available on the main International Exchange and as such are available via the Exchange Stream API. The separate Australian Exchange (markets starting 2.) is no longer in operation.

Runner Removals on the Order Stream

When a Rule 4 Runner Removal occurs in a Horse Race the price of matched bets on remaining runners are reduced by a Reduction Factor.

For these matched bets, you will receive on the Order Stream both a uo for the affected bet and the relevant updates to mb or ml (reducing the matched volume at the original matched price and adding volume at the new reduced price).

Initial bet placement at price 12


```
{"op":"ocm","id":2,"clk":"AK0CAPsBALEC","pt":1467219304831,"oc":[{"id":"
1.102151675","orc":[{"fullImage":true,"id":6113662,"uo":[{"id":"10822867
886","p":12,"s":2,"side":"B","status":"E","pt":"L","ot":"L","pd":1467219
304000,"sm":0,"sr":2,"sl":0,"sc":0,"sv":0,"rac":"","rc":"REG_GGC"}]}]}]}
```

Bet fully matched at price 12


```
{"op":"ocm","id":2,"clk":"AK0CAPsBALMC","pt":1467219316709,"oc":[{"id":"1.102151675","orc":[{"id":6113662,"uo":[{"id":"10822867886","p":12,"s":2,"side":"B","status":"EC","pt":"L","ot":"L","pd":1467219304000,"md":1467219316000,"avp":12,"sm":2,"sr":0,"sl":0,"sc":0,"sv":0}],"mb":[[12,2]]}]}]}
```


Runner removed (and so bet reduced in price to 9.47)


```
{"op":"ocm","id":2,"clk":"AK0CAJACALsC","pt":1467219376611,"oc":[{"id":"1.102151675","orc":[{"id":6113662,"uo":[{"id":"10822867886","p":12,"s":2,"side":"B","status":"EC","pt":"L","ot":"L","pd":1467219304000,"md":1467219316000,"avp":9.47,"sm":2,"sr":0,"sl":0,"sc":0,"sv":0}],"mb":[[9.47,2],[12,0]]}]}]}
```


See the avp in the uo record showing the new price of 9.47 and see the two entries in mb, one to remove the previously added size of 2 at price point 12 and one to add the size of 2 into the new price point of size 9.47.

Bets placed on the actual removed runner will be voided/lapsed (for matched/unmatched bets respectively) and these updates will also be sent on the Order Stream.

Line Markets

Line markets being sent on the Market Stream can be identified by the bettingType field of MarketDefinition (with value of "LINE").

The MarketDefinition of Line markets provide some additional fields that will be null for all other types,

- lineMaxUnit - maximum value for the outcome, in market units for this market (eg 100 runs).
- lineMinUnit - minimum value for the outcome, in market units for this market (eg 0 runs).
- lineInterval - the odds ladder on this market will be between the range of lineMinUnit and lineMaxUnit, in increments of the interval value.e.g. If lineMinUnit=10 runs, lineMaxUnit=20 runs, lineInterval=0.5 runs, then valid odds include 10, 10.5, 11, 11.5 up to 20 runs.

For updates for Orders on Line markets received on the Order Stream be aware of how the following properties behave,

- price - line markets operate at even-money odds of 2.0. However, price for these markets refers to the line positions available as defined by the markets min-max range and interval steps.
- side - for Line markets a 'B' bet refers to a SELL line and an 'L' bet refers to a BUY line.
- averagePriceMatched - this value is not meaningful for activity on Line markets and is not guaranteed to be returned or maintained for these markets.