dxFeed Market Data:
Order Book Reconstruction

May 2019

Information in this document is subject to change without notice and does not represent a commitment on the part of dxFeed.

Please visit www.dxfeed.com for latest information updates.

Contents

1. Introduction

dxFeed market data feeds (real-time, delayed or historical) allow to reconstruct order books, price level aggregations, and aggregations by Market Maker or a bank. In this specification, we describe order book reconstruction.

For reconstruction, you can:

1. Use **OrderBookModel** class in dxFeed API.
2. Get acknowledged with **Order** event attributes and implement the reconstruction algorithm.

We recommend to use dxFeed API, because it correctly implements reconstruction protocol.

2. dxFeed API

To get the best and smoothest solution for order book reconstruction task, we recommend to use our dxFeed Java API: https://downloads.dxfeed.com/api/. It provides **OrderBookModel** class which builds a high-quality order book from a set of market events (com.dxfeed.event.market.Order). This class keeps track of transactions and snapshots, filters data, etc. Learn more about events and classes in dxFeed API: https://docs.dxfeed.com/dxfeed/api/allclasses-noframe.html

3. Reconstructing order book without OrderBookModel class of dxFeed API

You can reconstruct an order book without using **OrderBookModel** class in dxFeed API. For this, do the following:

1. Learn about Indexes and Eventflags fields to know how to track snapshots, updates and transactions.
2. Follow the algorithm.

3.1. Indexes

Order book consists of undefined number of orders. Each exchange specify id for each order. Since each exchange use different id, we assign our own indexes to each order (we put orders in special slots that has indexes assigned).

Index of an order distinguishes different entries in the order book. Please note that an order may move to another index, so this identifier is not permanent. Every event can be a new order, an order modification, or an order cancellation.

3.2. Regional events

Since every regional event has an indication of its exchange (non-zero source id or exchange code in QD-model), you need to reconstruct an order book for each source id separately. For example, you need to create two order books, if you have both order#bzx and order#byx events in your market data feed.

3.3. Event flags applicable to Order event

Event flags define borders of snapshots and transactions retransmissions.

1. TX_PENDING indicates a pending transactional update. When TX_PENDING is true, it means that an ongoing transaction update, that spans multiple events, is in process.
2. REMOVE_EVENT indicates that the event with the corresponding index has to be removed.
3. SNAPSHOT_BEGIN indicates when the loading of a snapshot starts. Snapshot load starts on new subscription and the first indexed event that arrives for each non-zero source id on new subscription may have SNAPSHOT_BEGIN set to true. It means, that an ongoing snapshot consisting of multiple events is incoming.
4. SNAPSHOT_END indicates the end of a snapshot. It indicates that the data source had sent all the data pertaining to the subscription for the corresponding indexed event.

3.4. Snapshots

Order event stream consists of snapshots and updates for these snapshots. Snapshots start with SNAPSHOT_BEGIN flag and end with SNAPSHOT_END flag.

NB! Due to possible reconnections and retransmissions, the snapshots can overlap each other, so in the event stream the flags can intersect: it is possible to find SNAPSHOT_END before SNAPSHOT_BEGIN, or it is possible to find SNAPSHOT_BEGIN after SNAPSHOT_BEGIN, and so on. Therefore, the reconstruction algorithm should extract a complete snapshot ignoring the wrong residues of other snapshots.

If you catch the SNAPSHOT_BEGIN flag, you have to dismiss the previous order book.

3.5. Transaction model

Updates of order books happen as a sequence of transaction. If a transaction consists of several events, then all these events except the last one have TX_PENDING flag. If an event does not have TX_PENDING flag, it means that this events finishes the current transaction or, if there is no current transaction, then this event is operated as a single-event transaction.

Snapshots can overlap with order book modifications and transactions. While the snapshot is being transmitted, updates of the order book can occur. There are two consequences:

1. The snapshot can have modifications of the order book.
2. A snapshot update during the transfer can create a transaction until the end of the snapshot (we will add TX_PENDING flag until the end of the snapshot). It means that it will create an artificial transaction until the end of this snapshot.

3.5.1. Examples of transactions

One transaction:

Order#NTV,AAPL,20180926-100107.858-0400,0,1219,20180926-100107-0400,854:1,223.31,17,1307,NSDQ,**EventFlags=TX_PENDING**

Order#NTV,AAPL,20180926-100107.858-0400,0,1623,20180926-100107-0400,854:2,223.29,59,1303,NSDQ

Single-event transaction:

Order#NTV,AAPL,20180926-100107.826-0400,0,723,20180926-100107-0400,822:0,223.07,100,1303,NSDQ

Two single-event transactions:

Order#NTV,AAPL,20180926-100107.441-0400,0,2449,20180926-100107-0400,437:0,223.3,100,1307,NSDQ

Order#NTV,AAPL,20180926-100107.441-0400,0,525,20180926-100107-0400,437:1,223.26,100,1303,NSDQ

3.6. Algorithm

The proper Order Book Reconstruction Algorithm implies the use of a pending queue, which accumulates events and is applied transactionally to the reconstructed integral model of an order book. Order Book Reconstruction Algorithm tracks incoming event flags and the current accumulation status of the pending queue to determine the next action.

NB! In case of non-zero source ids, you need to reconstruct an order book for each source id separately. For example, if you have both order#bzx and order#byx events in your market data feed, you need to distinguish these events and create separate pending queues and consistent order books.

Order Book Reconstruction Algorithm has:

- a pending queue
- a consistent order book that we reconstruct
- three flags: isSnapshot, txPending, hasSnapshot

The algorithm is as follows:

1. If event has SNAPSHOT_BEGIN flag, then clear the pending queue, set isSnapshot = true, hasSnapshot = false.
2. If event has SNAPSHOT_END flag and isSnapshot is true, then set isSnapshot = false and hasSnapshot = true.
3. If event has TX_PENDING, then set txPending = true.
4. If event does not have TX_PENDING, then set txPending = false.
5. Add event to the pending queue.
6. If isSnapshot is false and txPending is false, then apply (see below) pending queue to the consistent order book:
   a. If hasSnapshot is true, then clear (remove all orders) consistent order book and set hasSnapshot = false.

   b. Go through pending queue and apply each order to consistent order book:
      - If this order has REMOVE_EVENT flag, then remove event with this index from consistent order book.
      - Else add or update event with this index to order book.
   c. Clear pending queue.
7. Wait for the next order event.

This algorithm also works in case you need to reconstruct price levels or aggregations by Market Maker.