Reference Guide

AlgoTrader

Version 5.0
# Table of Contents

Preface .................................................................................................................. xiv

1. Document Conventions ....................................................................................... xiv
   1.1. Typographic Conventions .............................................................................. xiv
   1.2. Pull-quote Conventions ................................................................................ xv
   1.3. Notes and Warnings ..................................................................................... xvi

1. Introduction .......................................................................................................... 1

2. Installation and Deployment .................................................................................. 2
   2.1. Development Environment Installation ..................................................... 2
       2.1.1. Prerequisites .......................................................................................... 2
       2.1.2. AlgoTrader Eclipse IDE Installation .................................................... 7
       2.1.3. AlgoTrader Server Code Installation .................................................. 8
       2.1.4. Next Steps ............................................................................................ 11
   2.2. Server Environment Installation .................................................................... 11
       2.2.1. Docker based Installation ..................................................................... 11
       2.2.2. Docker Containers ............................................................................... 12
       2.2.3. Docker Compose .................................................................................. 18
       2.2.4. Docker Management ............................................................................ 23
   2.3. VM Arguments ................................................................................................ 24

3. Starting AlgoTrader .............................................................................................. 26
   3.1. Simulation Mode ............................................................................................. 28
   3.2. Live Trading Mode ......................................................................................... 29
       3.2.1. Embedded Mode .................................................................................. 29
       3.2.2. Distributed Mode ................................................................................ 30
   3.3. Server Environment ....................................................................................... 31
       3.3.1. Embedded Mode .................................................................................. 31
       3.3.2. Distributed Mode ................................................................................ 33

4. Strategy Development ......................................................................................... 35
   4.1. Creating a Trading Strategy .......................................................................... 35
       4.1.1. AlgoTrader Strategy Wizard ................................................................. 35
       4.1.2. AlgoTrader Maven Archetype ............................................................... 37
       4.1.3. Generated Artifacts Java Archetype .................................................... 38
       4.1.4. Generated Artifacts Simple Java Archetype ....................................... 40
       4.1.5. Generated Artifacts Esper Archetype ................................................. 43
   4.2. Building a Trading Strategy ........................................................................... 48
   4.3. Hints for Strategy Development .................................................................... 49
       4.3.1. Java based Strategies ........................................................................... 49
       4.3.2. Esper based Strategies ......................................................................... 57
   4.4. Strategy life-cycle events ............................................................................... 64
   4.5. Strategy Groups ............................................................................................. 66

5. Strategy Backtesting ............................................................................................ 69
   5.1. Exchange Simulator ....................................................................................... 69
   5.2. Simulation Process ......................................................................................... 71
# Table of Contents

5.3. Single Run Simulation ................................................................. 72  
5.4. Automated Parameter Optimization ............................................. 72  
5.5. Performance Statistics ............................................................... 74  
5.6. In-Process Exchange Simulator (deprecated) ................................. 77  
5.7. Multi Security Simulations .......................................................... 77  
6. Architecture .................................................................................... 79  
7. Domain Model ................................................................................ 80  
7.1. Entities ......................................................................................... 80  
  7.1.1. Strategy .................................................................................. 82  
  7.1.2. Security .................................................................................. 84  
  7.1.3. Market Data Events ................................................................. 85  
  7.1.4. Order ....................................................................................... 86  
  7.1.5. Account .................................................................................... 88  
  7.1.6. Transaction .............................................................................. 89  
  7.1.7. Position .................................................................................... 90  
  7.1.8. Cash Balance .......................................................................... 91  
  7.1.9. Subscription ............................................................................ 91  
  7.1.10. Exchange .............................................................................. 92  
  7.1.11. Order Preference ................................................................... 93  
7.2. Services ......................................................................................... 93  
  7.2.1. Main Services .......................................................................... 94  
  7.2.2. Client Services ......................................................................... 95  
  7.2.3. Account Service ....................................................................... 95  
  7.2.4. Calendar Service ..................................................................... 95  
  7.2.5. Combination Service ............................................................... 96  
  7.2.6. Future Service ......................................................................... 96  
  7.2.7. Historical Data Service .............................................................. 96  
  7.2.8. Market Data Service ................................................................. 96  
  7.2.9. Measurement Service ............................................................... 96  
  7.2.10. Option Service ....................................................................... 97  
  7.2.11. Order Service ......................................................................... 97  
  7.2.12. Portfolio Service ..................................................................... 97  
  7.2.13. Position Service ..................................................................... 98  
  7.2.14. Property Service .................................................................... 99  
  7.2.15. Reference Data Service ........................................................... 100  
  7.2.16. Market Data Cache Service .................................................... 100  
  7.2.17. Lookup Service ..................................................................... 100  
  7.2.18. Strategy Service & Config Aware Strategy Service .................... 101  
  7.2.19. Subscription Service ............................................................... 101  
  7.2.20. Reset Service ......................................................................... 102  
7.3. Value Object ................................................................................ 103  
7.4. Enumerations ............................................................................... 103  
8. Esper Engine .................................................................................... 104  
  8.1. Esper Introduction ....................................................................... 104
### Table of Contents

1. **Client** .......................................................... 103
   - 8.1.1. Introduction to event streams and complex events using Esper ........................................ 104
   - 8.1.2. Event representations .................................................. 105
   - 8.1.3. Event Stream Analysis .................................................. 105
   - 8.1.4. Combining Pattern Matching with Event Stream Analysis ................................................. 106
   - 8.1.5. Named windows ......................................................... 107
   - 8.1.6. Variables .................................................................. 107
   - 8.2. Esper Quick Start Guide .................................................. 107
     - 8.2.1. Event Types ............................................................... 107
     - 8.2.2. Creating a Statement .................................................. 108
     - 8.2.3. Adding a Subscriber ................................................... 108
     - 8.2.4. Adding a Listener ....................................................... 109
     - 8.2.5. Sending events .......................................................... 110
     - 8.2.6. Configuration ........................................................... 110
   - 8.3. Esper Documentation ................................................... 110
   - 8.4. AlgoTrader specific Esper Artifacts ...................................... 111
     - 8.4.1. Engine & EngineManager ............................................ 111
     - 8.4.2. Modules ................................................................. 112
     - 8.4.3. Tags .................................................................. 113
     - 8.4.4. Subscribers .............................................................. 113
     - 8.4.5. Listeners ............................................................... 114
     - 8.4.6. Service method invocation in Esper scripts ......................................................... 114
     - 8.4.7. Aggregation Functions ................................................ 115
     - 8.4.8. Callbacks ............................................................... 117
   - 8.5. Esper Threading ........................................................ 120

2. **Database** .................................................................. 121
   - 9.1. Instances .................................................................. 121
   - 9.2. Flyway ................................................................. 121
   - 9.3. Files ................................................................. 121
   - 9.4. Data Source .......................................................... 122

3. **Client** ................................................................... 123
   - 10.1. HTML5 UI .............................................................. 123
     - 10.1.1. Header ........................................................... 124
     - 10.1.2. Order Table .......................................................... 127
     - 10.1.3. Advanced Order Form ............................................ 128
     - 10.1.4. Algo Order details UI .............................................. 130
     - 10.1.5. Transaction Table ................................................. 131
     - 10.1.6. Positions Table ..................................................... 132
     - 10.1.7. Market Data Table ................................................ 133
     - 10.1.8. Column Selection and Grouping .................................. 134
     - 10.1.9. CSV Export ......................................................... 136
     - 10.1.10. Chart Widget ...................................................... 136
     - 10.1.11. About pop-up ..................................................... 139
     - 10.1.12. Technologies ...................................................... 139
     - 10.1.13. HTML5 Custom Widgets ...................................... 139
10.2. AlgoTrader Eclipse IDE ................................................................. 144
  10.2.1. AlgoTrader Perspective ..................................................... 144
  10.2.2. Strategy Wizard ............................................................... 145
  10.2.3. AlgoTrader Configuration Editor ....................................... 145
  10.2.4. Esper Colorer ................................................................. 153
10.3. Reference Data Manager ......................................................... 154
10.4. Historical Data Manager ......................................................... 156
11. Performance Measurement ............................................................. 163
  11.1. Portfolio Value Logging ....................................................... 163
  11.2. Portfolio Value Restoration Feature ...................................... 163
12. Risk Management ........................................................................... 165
  12.1. Pre-Trade Checks ................................................................. 165
13. Forex Handling .............................................................................. 167
  13.1. Currency Handling ............................................................... 167
    13.1.1. Futures ........................................................................... 168
    13.1.2. Forex ............................................................................... 168
    13.1.3. Currency Attribution ..................................................... 168
  13.2. Forex-Hedging ......................................................................... 169
    13.2.1. Exchange vs. Margin Trading ......................................... 169
    13.2.2. FX Future ........................................................................ 170
14. Options & Futures ........................................................................... 171
  14.1. Expiration .............................................................................. 171
  14.2. Leverage & Exposure ............................................................... 171
  14.3. Symbol, ISIN & RIC ............................................................... 172
  14.4. Delta Hedging ......................................................................... 172
  14.5. Option & Future Chain Download ......................................... 173
  14.6. Option Greeks ........................................................................ 173
  14.7. Option Pricing Engine ............................................................. 173
    14.7.1. SABR Calibration ............................................................. 173
    14.7.2. Option Pricing ............................................................... 174
    14.7.3. References ...................................................................... 174
  14.8. OTC Options ......................................................................... 174
15. Broker/Exchange Interfaces .............................................................. 175
16. Order Management ........................................................................... 177
  16.1. Order Validation ................................................................. 177
  16.2. Place Order ........................................................................... 177
    16.2.1. Order Preferences .......................................................... 178
    16.2.2. Trade Suggestions ......................................................... 179
    16.2.3. Order Properties ............................................................. 179
  16.3. Order Status ........................................................................... 180
  16.4. Receive Fills .......................................................................... 181
  16.5. Handling of Fees and Commissions ....................................... 182
  16.6. Examples of Orders and Executions ....................................... 182
    16.6.1. Margin Order with Fee in Transaction Currency ............... 182
16.6.2. Exchange Order with Fee in Base Currency ........................................ 183
16.6.3. Exchange Order with Fee in Alternate Currency .............................. 183
16.7. Internal Order Id Format ........................................................................... 184
16.8. Symbology ................................................................................................. 185
17. Market Data .................................................................................................... 186
  17.1. Creation of Bars based on Ticks ................................................................. 188
  17.2. Creation of Bars based on Bars .................................................................. 189
     17.2.1. Esper Bar Aggregation ...................................................................... 189
     17.2.2. Java Bar Aggregation ...................................................................... 189
  17.3. Numeric Precision .................................................................................... 190
  17.4. Price normalization .................................................................................. 190
  17.5. Market Data Gap Checking ..................................................................... 190
  17.6. Generic Events ......................................................................................... 191
  17.7. Generic Tick Events ................................................................................ 192
18. Historical Data ................................................................................................ 193
  18.1. InfluxDB .................................................................................................... 194
  18.2. Live Data Recording ................................................................................ 198
  18.3. Historical Data Download ....................................................................... 198
  18.4. Interactive Brokers Historical Data Download ............................................. 200
  18.5. Quandl Historical Data Download .............................................................. 201
  18.6. Google Finance Historical Data Download ............................................... 201
  18.7. CoinAPI Historical Data Download ......................................................... 201
  18.8. CoinMarketCap Historical Data Download .............................................. 201
  18.9. Market Data File Format ......................................................................... 201
     18.9.1. Tick Data Files ............................................................................... 202
     18.9.2. Bar Data Files ................................................................................ 203
19. Reference Data ................................................................................................ 204
20. Account Data .................................................................................................. 206
  20.1. Account balances ..................................................................................... 207
  20.2. Withdrawal ............................................................................................... 207
  20.3. Deposit address ......................................................................................... 208
  20.4. Account Events ......................................................................................... 208
21. AlgoTrader API ............................................................................................... 210
  21.1. JSON data binding ................................................................................... 210
  21.2. REST API ................................................................................................ 210
  21.3. WebSocket/STOMP API ......................................................................... 211
  21.4. Inbound FIX API ..................................................................................... 214
     21.4.1. Logon message ............................................................................... 216
     21.4.2. Logout message ............................................................................... 216
     21.4.3. Test Request message ..................................................................... 216
     21.4.4. Heartbeat message ......................................................................... 217
     21.4.5. Resend Request message ................................................................. 217
     21.4.6. New Order Single message ............................................................... 217
     21.4.7. Order Cancel Request message ....................................................... 218
22. Adapters ................................................................. 221
  22.1. Fix Interface ..................................................... 221
    22.1.1. FIX configuration ....................................... 223
    22.1.2. FIX logging ............................................. 224
    22.1.3. FIX message persistence ............................... 226
    22.1.4. FIX Drop-copy support ................................ 227
  22.2. Crypto Exchange interfaces ................................ 227
    22.2.1. Custom currency mapping ............................... 227
    22.2.2. Crypto-Order Constraints .............................. 227
    22.2.3. Supported Crypto-Order Types ......................... 228
  22.3. Session life-cycle events .................................. 228
  22.4. Automatic order reconciliation after re-connect ......... 229
  22.5. Bloomberg .................................................. 229
  22.6. Currenex ................................................... 230
  22.7. DukasCopy .................................................. 230
  22.8. Exante (XNT) ................................................ 230
  22.9. EzeSoft / Real Tick .......................................... 231
  22.10. Fortex .................................................... 231
  22.11. FXCM ..................................................... 231
  22.12. IB Native Interface ....................................... 231
    22.12.1. IB Market Data Subscriptions ....................... 234
    22.12.2. Delayed IB Market Data ............................. 238
  22.13. IB Fix Interface ........................................... 239
  22.14. JP Morgan ................................................ 240
  22.15. LMAX ..................................................... 240
  22.16. Nexus Prime ............................................... 240
  22.17. PrimeXM ................................................... 241
  22.18. Quandl .................................................... 241
  22.19. QuantHouse ............................................... 242
  22.20. SocGen ..................................................... 242
  22.21. Trading Technologies (TT) ............................... 243
  22.22. UBS ....................................................... 243
  22.23. Binance ................................................... 243
    22.23.1. Binance Order Constraints .......................... 244
    22.23.2. Binance Account Management ........................ 245
  22.24. Bitfinex .................................................. 245
    22.24.1. Bitfinex Order Constraints .......................... 246
    22.24.2. Bitfinex Account Management ........................ 246
  22.25. Bitflyer .................................................. 247
    22.25.1. Bitflyer Order Constraints .......................... 247
    22.25.2. Bitflyer Account Management ........................ 248
  22.26. BitMEX ................................................... 249
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.26.1. BitMex Order Constraints</td>
<td>250</td>
</tr>
<tr>
<td>22.26.2. BitMex Account Management</td>
<td>250</td>
</tr>
<tr>
<td>22.27. Bitstamp</td>
<td>250</td>
</tr>
<tr>
<td>22.27.1. Bitstamp Order Constraints</td>
<td>251</td>
</tr>
<tr>
<td>22.27.2. Bitstamp Account Management</td>
<td>252</td>
</tr>
<tr>
<td>22.28. CoinAPI</td>
<td>252</td>
</tr>
<tr>
<td>22.29. Coinbase Pro</td>
<td>253</td>
</tr>
<tr>
<td>22.29.1. Coinbase Pro Order Constraints</td>
<td>254</td>
</tr>
<tr>
<td>22.29.2. Coinbase Pro Account Management</td>
<td>255</td>
</tr>
<tr>
<td>22.30. Coinigy</td>
<td>255</td>
</tr>
<tr>
<td>22.30.1. Setup Instructions</td>
<td>255</td>
</tr>
<tr>
<td>22.30.2. Coinigy Order Constraints</td>
<td>256</td>
</tr>
<tr>
<td>22.30.3. Coinigy Account Management</td>
<td>257</td>
</tr>
<tr>
<td>22.31. CoinMarketCap</td>
<td>257</td>
</tr>
<tr>
<td>23. Execution Algos</td>
<td>258</td>
</tr>
<tr>
<td>23.1. Existing Execution Algos</td>
<td>258</td>
</tr>
<tr>
<td>23.2. Execution Algos Retry and Back-off policies</td>
<td>264</td>
</tr>
<tr>
<td>24. Synthetic Securities and Derivative Spreads</td>
<td>267</td>
</tr>
<tr>
<td>24.1. Combination Example</td>
<td>268</td>
</tr>
<tr>
<td>24.2. Combination Service</td>
<td>268</td>
</tr>
<tr>
<td>24.2.1. Create Combination</td>
<td>268</td>
</tr>
<tr>
<td>24.2.2. Update Component Quantity</td>
<td>268</td>
</tr>
<tr>
<td>24.2.3. Remove a Component</td>
<td>269</td>
</tr>
<tr>
<td>25. Spring Services</td>
<td>270</td>
</tr>
<tr>
<td>25.1. Starter Classes</td>
<td>270</td>
</tr>
<tr>
<td>25.2. Spring Profiles</td>
<td>271</td>
</tr>
<tr>
<td>26. Configuration and Preferences API</td>
<td>274</td>
</tr>
<tr>
<td>26.1. Configuration Files</td>
<td>274</td>
</tr>
<tr>
<td>26.1.1. Encrypting sensitive configuration values</td>
<td>275</td>
</tr>
<tr>
<td>26.2. Esper Variables</td>
<td>275</td>
</tr>
<tr>
<td>27. Processes and Networking</td>
<td>277</td>
</tr>
<tr>
<td>27.1. SSL security</td>
<td>277</td>
</tr>
<tr>
<td>27.1.1. Importing Certificate into Chrome Browser</td>
<td>278</td>
</tr>
<tr>
<td>28. Metrics</td>
<td>279</td>
</tr>
<tr>
<td>28.1. Configuration</td>
<td>279</td>
</tr>
<tr>
<td>28.2. Metrics Reporting</td>
<td>279</td>
</tr>
<tr>
<td>29. Logging</td>
<td>280</td>
</tr>
<tr>
<td>29.1. log4j2.xml</td>
<td>280</td>
</tr>
<tr>
<td>29.2. Production log4j2.xml</td>
<td>280</td>
</tr>
<tr>
<td>30. Reporting</td>
<td>281</td>
</tr>
<tr>
<td>A. Example Strategy &quot;BreakOut&quot;</td>
<td>282</td>
</tr>
<tr>
<td>A.1. Trading Idea</td>
<td>282</td>
</tr>
<tr>
<td>A.2. Example</td>
<td>282</td>
</tr>
<tr>
<td>A.3. Implementation</td>
<td>283</td>
</tr>
</tbody>
</table>
List of Figures

2.1. Eclipse default JRE ................................................................. 10
3.1. Eclipse Run Configurations .................................................. 26
3.2. Eclipse Run Configurations .................................................. 27
3.3. Eclipse Run Configurations .................................................. 28
4.1. Strategy Development Process ............................................. 35
4.2. Wizard Selection ............................................................... 36
4.3. Wizard Location and Working Set .......................................... 36
4.4. Wizard Maven Properties .................................................... 36
4.5. Wizard Maven Properties .................................................... 37
5.1. Back Test Report ............................................................... 75
6.1. Architecture ...................................................................... 79
7.1. Entities Overview .............................................................. 80
7.2. Strategy ........................................................................... 82
7.3. Securities ......................................................................... 84
7.4. Orders ............................................................................... 86
7.5. Account ........................................................................... 88
7.6. Transaction .......................................................... 89
7.7. Position ........................................................................... 90
7.8. Subscription ..................................................................... 91
7.9. Exchange .......................................................................... 92
7.10. Order Preference .............................................................. 93
10.1. AlgoTrader UI Header ........................................................ 124
10.2. AlgoTrader UI Header Settings ............................................ 125
10.3. AlgoTrader UI Management ................................................ 125
10.4. AlgoTrader UI Management Form ...................................... 126
10.5. AlgoTrader UI Notification ................................................ 126
10.6. AlgoTrader UI Alert List .................................................... 127
10.7. AlgoTrader UI Order Table ................................................ 127
10.8. AlgoTrader UI Manual Order Entry .................................... 127
10.9. AlgoTrader UI Manual Order Modification ....................... 128
10.10. AlgoTrader UI Advanced Order Form ................................ 128
10.11. AlgoTrader UI Advanced Order Form - Crypto mode of Routing section .................................................. 129
10.12. AlgoTrader UI Advanced Order Form - Equity mode of Routing section .................................................. 130
10.13. Execution Algo details icon visible in action column in the Orders table .................................................. 131
10.14. Execution Algo details modal window with a grid listing all children of a given Execution Algo ........ 131
10.15. AlgoTrader UI Transaction Table ....................................... 131
10.16. AlgoTrader UI Transaction Entry ....................................... 132
10.17. AlgoTrader UI Transaction Entry and Fees Entry ................ 132
10.18. AlgoTrader UI Position Table ............................................ 133
10.19. AlgoTrader UI Market Data Table ...................................... 133
10.20. AlgoTrader UI Market Data Subscribe ................................. 133
10.21. AlgoTrader UI Market Data Unsubscribe .............................. 134
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.22. AlgoTrader UI Transaction Column Selection</td>
<td>135</td>
</tr>
<tr>
<td>10.23. AlgoTrader UI Column Filter</td>
<td>135</td>
</tr>
<tr>
<td>10.24. AlgoTrader UI Chart Widget</td>
<td>136</td>
</tr>
<tr>
<td>10.25. AlgoTrader UI Chart Widget settings</td>
<td>137</td>
</tr>
<tr>
<td>10.27. &quot;About&quot; pop-up</td>
<td>139</td>
</tr>
<tr>
<td>10.28. HTML5 Custom Widget Example</td>
<td>140</td>
</tr>
<tr>
<td>10.29. AlgoTrader Configuration Editor Error</td>
<td>146</td>
</tr>
<tr>
<td>10.30. AlgoTrader Configuration Editor</td>
<td>146</td>
</tr>
<tr>
<td>10.31. Strategy Data</td>
<td>147</td>
</tr>
<tr>
<td>10.32. Configuration file without strategy groups</td>
<td>148</td>
</tr>
<tr>
<td>10.33. Titles and Separators in a property file</td>
<td>150</td>
</tr>
<tr>
<td>10.34. Syntax Highlighter</td>
<td>153</td>
</tr>
<tr>
<td>14.1. Leverage &amp; Exposure</td>
<td>172</td>
</tr>
<tr>
<td>16.1. Order Status Transitions</td>
<td>181</td>
</tr>
<tr>
<td>17.1. Market Data Event Types</td>
<td>187</td>
</tr>
<tr>
<td>22.1. Market Data Subscriptions 1</td>
<td>235</td>
</tr>
<tr>
<td>22.2. Market Data Subscriptions 2</td>
<td>236</td>
</tr>
<tr>
<td>22.3. Market Data Subscriptions 3</td>
<td>236</td>
</tr>
<tr>
<td>22.4. Market Data Subscriptions 4</td>
<td>237</td>
</tr>
<tr>
<td>22.5. Paper Trading Account 1</td>
<td>238</td>
</tr>
<tr>
<td>22.6. Paper Trading Account 2</td>
<td>238</td>
</tr>
<tr>
<td>22.7. Delayed IB Market Data</td>
<td>239</td>
</tr>
<tr>
<td>23.1. Adaptive Send Child Order Retry Policy</td>
<td>265</td>
</tr>
<tr>
<td>23.2. Adaptive Cancel Child Order Retry Policy</td>
<td>266</td>
</tr>
<tr>
<td>24.1. Combinations and Components</td>
<td>267</td>
</tr>
<tr>
<td>24.2. Combination Example</td>
<td>268</td>
</tr>
<tr>
<td>A.1. BreakOut Strategy Example</td>
<td>283</td>
</tr>
<tr>
<td>B.1. Box Trading Ranges</td>
<td>286</td>
</tr>
<tr>
<td>B.2. Box Strategy</td>
<td>287</td>
</tr>
<tr>
<td>B.3. Box States</td>
<td>288</td>
</tr>
<tr>
<td>B.4. Box Strategy Performance</td>
<td>288</td>
</tr>
<tr>
<td>B.5. Box HTML5 Custom Widget Example</td>
<td>290</td>
</tr>
<tr>
<td>C.1. Pair Trading Portfolio ID</td>
<td>296</td>
</tr>
<tr>
<td>C.2. Pairs Trading HTML5 Custom Widget Example</td>
<td>298</td>
</tr>
<tr>
<td>D.1. IPO HTML5 Custom Widget Example</td>
<td>301</td>
</tr>
<tr>
<td>E.1. EMA Strategy Example</td>
<td>305</td>
</tr>
</tbody>
</table>
List of Tables

4.1. Strategy life-cycle phase ................................................................. 65
7.1. Entities .................................................................................. 80
7.2. Strategy Classes ................................................................. 83
7.3. Portfolio Value Details .......................................................... 83
7.4. Security Types .................................................................. 84
7.5. Market Data Types ............................................................ 86
7.6. Order Classes .................................................................. 87
7.7. Position Valuation Details .................................................. 91
7.8. Exchange ........................................................................ 92
7.9. Order Preference ............................................................... 93
7.10. Main Services .................................................................. 94
7.11. Client Services ................................................................. 95
8.1. AlgoTrader Server modules ................................................ 112
8.2. Esper tags ...................................................................... 113
10.1. out-of-the-box types .......................................................... 150
13.1. Position Currency Attribution ............................................. 168
13.2. Transaction Currency Attribution ....................................... 169
14.1. Bar Data Format ................................................................. 171
18.1. Tick Data Format ................................................................. 203
18.2. Bar Data Format ................................................................. 203
21.1. Event topics .................................................................. 211
21.2. Logon ........................................................................... 216
21.3. Logout ............................................................................ 216
21.4. Test Request ................................................................ 216
21.5. Heartbeat .................................................................... 217
21.6. Resend Request ................................................................. 217
21.7. New Order Single ............................................................. 217
21.8. Cancel Request ................................................................. 218
21.9. Cancel Replace Request .................................................... 219
21.10. Execution Report .............................................................. 219
22.1. Order type constraints ....................................................... 228
22.2. Binance constraints ............................................................. 244
22.3. Supported Functionality ..................................................... 245
22.4. Bitfinex constraints ............................................................. 246
22.5. Supported Functionality ..................................................... 246
22.6. BitFlyer constraints ............................................................. 248
22.7. Supported Functionality ..................................................... 248
22.8. Supported Instruments ......................................................... 249
22.9. BitMEX constraints ............................................................. 250
22.10. Supported Functionality ..................................................... 250
22.11. BitStamp constraints .......................................................... 251
22.12. Supported Functionality ..................................................... 252
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.13. Coinbase Pro constraints</td>
<td>254</td>
</tr>
<tr>
<td>22.14. Supported Functionality</td>
<td>255</td>
</tr>
<tr>
<td>22.15. Coingy constraints</td>
<td>256</td>
</tr>
<tr>
<td>22.16. Supported Functionality</td>
<td>257</td>
</tr>
<tr>
<td>23.1. SlicingOrder</td>
<td>258</td>
</tr>
<tr>
<td>23.2. VWAPOrder</td>
<td>260</td>
</tr>
<tr>
<td>23.3. AdaptiveOrder</td>
<td>262</td>
</tr>
<tr>
<td>25.1. Adapter Spring Profiles</td>
<td>272</td>
</tr>
<tr>
<td>29.1. Default Log4j Appenders</td>
<td>280</td>
</tr>
<tr>
<td>29.2. Production Log4j Appenders</td>
<td>280</td>
</tr>
</tbody>
</table>
Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the Liberation Fonts\(^1\) set. The Liberation Fonts set is also used in HTML editions. If not, alternative but equivalent typefaces are displayed.

1.1. Typographic Conventions

The following typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

System input, including shell commands, file names and paths, and key caps and key-combinations are presented as follows.

To see the contents of the file my_next_bestselling_novel in the current working directory, enter the `cat my_next_bestselling_novel` command at the shell prompt and press **Enter** to execute the command.

The above includes a file name, a shell command and a key cap, all distinguishable thanks to context.

Key-combinations can be distinguished from key caps by the symbol connecting each part of a key-combination. For example:

Press **Enter** to execute the command.

Press **Ctrl-Alt-F1** to switch to the first virtual terminal. Press **Ctrl-Alt-F7** to return to the X-Windows session.

The first sentence highlights the particular key cap to press. The second highlights two sets of three key caps, each set pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph are presented as follows.

File-related classes include `filesystem` for file systems, `file` for files, and `dir` for directories.

Each class has its own associated set of permissions.

Words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles are presented as follows.

Choose **System → Preferences → Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).

---

\(^1\) [https://pagure.io/liberation-fonts](https://pagure.io/liberation-fonts)
To insert a special character into a *gedit* file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to the document and choose **Edit** → **Paste** from the *gedit* menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all distinguishable by context.

Note the shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow ‘Select **Mouse** from the **Preferences** sub-menu in the **System** menu of the main menu bar’ approach.

Italics denotes text that does not need to be imputed literally or displayed text that changes depending on circumstance. Replaceable or variable text is presented as follows.

To connect to a remote machine using ssh, type `ssh username@domain.name` at a shell prompt. If the remote machine is `example.com` and the username on that machine is `john`, type `ssh john@example.com`.

The `mount -o remount file-system` command remounts the named file system. For example, to remount the `home` file system, the command is `mount -o remount /home`.

To see the version of a currently installed package, use the `rpm -q package` command. It will return a result as follows: `package-version-release`.

Note the words in italics above — `username`, `domain.name`, `file-system`, `package`, `version` and `release`. Each word is a placeholder, either for text entered when issuing a command or for text displayed by the system.

### 1.2. Pull-quote Conventions

Two commonly multi-line data types are set off visually from the surrounding text.

Output sent to a terminal is presented as follows:

```
books        Desktop   documentation  drafts  mss    photos   stuff  git
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are presented and highlighted as follows:

```java
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient {

    public static void main(String args[]) throws Exception {
```
InitialContext iniCtx = new InitialContext();
Object ref = iniCtx.lookup("EchoBean");
EchoHome home = (EchoHome) ref;
Echo echo = home.create();

System.out.println("Created Echo");
System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
Introduction

AlgoTrader is a comprehensive algorithmic trading platform that enables both buy side and sell side trading firms to rapidly develop, simulate, backtest and deploy automated quantitative trading strategies on a single platform. Designed by industry experts, it gives users maximum control over their trading experience. Initially designed for global equities, futures, forex and options, AlgoTrader now fully supports automated trading of Cryptocurrencies. AlgoTrader is an extremely reliable and robust system built on a multi-threaded, memory efficient, highly concurrent architecture. It is optimized for high availability and performance to support uninterrupted trading.

The following links provide general information about the system

- System overview
- Demo
- Trial Version
- Videos
- Architecture
- Screenshots
- Product Features
- Product Factsheet
- List of 3rd party libraries

1 https://www.algotrader.com/product/overview/
2 https://www.algotrader.com/product/demo-system/
3 https://www.algotrader.com/product/demo-system/
4 https://www.algotrader.com/product/video/
5 https://www.algotrader.com/product/architecture/
6 https://www.algotrader.com/product/screenshots/
7 https://www.algotrader.com/features/
8 http://doc.algotrader.ch/AlgoTraderFactsheet.pdf
9 https://www.algotrader.com/product/3rd-party-libraries/
Chapter 2.

Installation and Deployment

2.1. Development Environment Installation

2.1.1. Prerequisites

Note

It is generally recommend not to use paths with spaces for any of the components used by AlgoTrader (e.g. C:\Program Files).

Java JDK

Install the latest Java JDK 1.8 from Oracle.

Important

AlgoTrader requires Java 1.8.x. Do not use Java 1.9 or greater.

It is necessary to have a Java JDK (Java Development Environment), a Java JRE (Java Runtime Environment) will not be sufficient.

Please set the JAVA_HOME environment variable to point at the directory where the Java JDK is installed. You also need to add JAVA_HOME\bin to your PATH variable. Setup java environment variables.

Maven

AlgoTrader uses Apache Maven for handling of dependencies. The AlgoTrader Eclipse IDE already has an embedded Maven installation integrated. In case one wants to use Maven from the command line, it is necessary to download and install the latest version of Maven and setup it’s environment variables according to the link Maven setup.

In particular, please set the MAVEN_HOME environment variable to point at the directory where Maven is installed. You also need to add MAVEN_HOME\bin to your PATH variable.

All AlgoTrader artifacts are located on our Nexus server which is password protected:

https://repo.algotrader.ch/nexus/

It is necessary the create the following file <user-home>\m2\settings.xml below content there, to make sure Maven can access our Nexus server. Folder .m2 should have been automatically created while running maven for the first time.

---

1 https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

2 http://maven.apache.org/install.html

3 http://maven.apache.org/install.html
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <server>
      <id>algotraderrepo</id>
      <username>myusername</username>
      <password>mypassword</password>
    </server>
    <server>
      <id>archetype</id>
      <username>myusername</username>
      <password>mypassword</password>
    </server>
  </servers>

  <profiles>
    <profile>
      <id>algotrader</id>
      <repositories>
        <repository>
          <id>algotraderrepo</id>
          <url>https://repo.algotrader.ch/nexus/repository/general/</url>
        </repository>
        <repository>
          <id>archetype</id>
          <url>https://repo.algotrader.ch/nexus/repository/general/</url>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>algotraderrepo</id>
          <url>https://repo.algotrader.ch/nexus/repository/general/</url>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>

  <activeProfiles>
    <activeProfile>algotrader</activeProfile>
  </activeProfiles>
</settings>
Note
Please replace myusername and mypassword (both appear twice!) with the username and password provided when licensing AlgoTrader.

Git
AlgoTrader uses Git as its source code management system. The AlgoTrader Eclipse IDE already has a Git installation integrated (EGit).

In case one wants to use Git from the command line it is necessary to download and install the latest version of Git.

On Windows use TortoiseGit in combination with Git for Windows.

MySQL

Note
MySQL is not needed for simulations based on the embedded / in-memory database H2.

Download and install MySQL Community Server.

Per default the system uses the user name root and password password. To change username and/or password the following properties need to be updated inside conf-core.properties. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```java
# database user name
#{"type":"String","label":"Data Source User name"}
dataSource.user = root

# database password
#{"type":"String","label":"Data Source Password"}
dataSource.password = password
```

You can create the root user/set the DB password using the following command:

```
mysqladmin -u myUsername password myPassword
```

To work with MySQL it is recommended to install a MySQL client. There are many different MySQL clients available to choose from:

- devart dbForge Studio Express (free)

---

4 https://tortoisegit.org
5 https://gitforwindows.org/
6 https://dev.mysql.com/downloads/
7 https://www.devart.com/dbforge/mysql/studio/download.html
Prerequisites

- MySQL Workbench\(^8\) (free)
- SQLyog MySQL\(^9\) (commercial)

**Note**

The Java MySQL JDBC driver sometimes has issues connecting to the MySQL database depending on the MySQL timezone setting. Java Exceptions like the following are an indication for this issue:

```
java.sql.SQLException: The server time zone value 'Coordinated Universal Time' is unrecognized or represents more than one time zone. You must configure either the server or JDBC driver (via the serverTimezone configuration property) to use a more specific time zone value if you want to utilize time zone support.
```

To fix the issue it is recommended to change the MySQL timezone setting by executing the following MySQL statement:

```
SET GLOBAL time_zone = 'UTC';
```

In case you get an error like this

```
mysql> SET GLOBAL time_zone = 'UTC';
ERROR 1298 (HY000): Unknown or incorrect time zone: 'UTC'
```

This means that your mysql database does not know the UTC timezone yet. Follow the instructions under [MySQL Time Zone Issues]\(^{10}\) to install them. For Windows this means to download the 5.7.sql file with the posix timezone definitions and manually loading them into your local DB instance. For Linux this can be easily fixed by running following command:

```
mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

To be sure you do not lose this information, e.g. on reboot, you can set a time zone in the MYSQL my.cnf or my.ini file

```
[mysql]
default-time-zone=+00:00
default_time_zone=servers
```

---

\(^8\) [https://www.mysql.com/products/workbench/](https://www.mysql.com/products/workbench/)
\(^9\) [https://www.webyog.com/](https://www.webyog.com/)
\(^{10}\) [https://dev.mysql.com/downloads/timezones.html](https://dev.mysql.com/downloads/timezones.html)
InfluxDB

**Note**

InfluxDB is an optional component that is used to store historical data. In back testing it is possible to use CSV files to provide historical data as an alternative to using InfluxDB.

**Linux/MacOS:** Download the latest version of InfluxDB\(^{11}\) and install it according to the InfluxDB installation instructions\(^{12}\) (Install it in a directory/tree without spaces in the names).

**Windows:** Download version 1.5.2 of InfluxDB\(^{13}\) and unpack the file in a directory/tree without spaces in the names.

**Important**

On Windows AlgoTrader requires InfluxDB 1.5.2. Do not use InfluxDB 1.6.0 or greater.

To install InfluxDB as a windows service please follow these steps:

- download nssm\(^{14}\)
- unpack nssm
- update the following sections inside influxdb.conf by replacing `<username>` with the username that will run InfluxDB

```plaintext
[meta]
# Where the metadata/raft database is stored
dir = "C:\Users\<username>\.influxdb\meta"

[data]
# The directory where the TSM storage engine stores TSM files.
dir = "C:\Users\<username>\.influxdb\data"

# The directory where the TSM storage engine stores WAL files.
wal-dir = "C:\Users\<username>\.influxdb\wal"
```

- Go to nssm installed folder, choose win64 or win32 folder, start a command prompt and type:

```
.nssm.exe install InfluxDB <full-path-to-influxd.exe> -config <full-path-to-influxdb.conf>
```

\(^{11}\) [https://portal.influxdata.com/downloads/](https://portal.influxdata.com/downloads/)
\(^{12}\) [https://docs.influxdata.com/influxdb/v1.5/introduction/installation/](https://docs.influxdata.com/influxdb/v1.5/introduction/installation/)
\(^{13}\) [https://dl.influxdata.com/influxdb/releases/influxdb-1.5.2_windows_amd64.zip](https://dl.influxdata.com/influxdb/releases/influxdb-1.5.2_windows_amd64.zip)
\(^{14}\) [https://nssm.cc/download](https://nssm.cc/download)
AlgoTrader Eclipse IDE Installation

(put path in quotation marks if it contains spaces), Example:

```
nssm.exe install InfluxDB c:\AlgoTrader\influxdb-1.5.2\influxd.exe -config c:\AlgoTrader\influxdb-1.5.2\influxdb.conf
```

Make sure you run the influxd.exe command, not influx.exe.

- start InfluxDB service via Windows Service Manager
- Add an environment variable named `HOME` pointing to the directory where InfluxDB is installed

All InfluxDB related settings are available within the file `conf-influxdb.properties`

Per default username/password authentication is disabled. To set username and password based authentication please visit the InfluxDB Authentication and Authorization guide\(^{15}\).

There are several client options available to access InfluxDB:

- `InfluxDB Chronograf`\(^{16}\)
- `Grafana`\(^{17}\)

2.1.2. AlgoTrader Eclipse IDE Installation

AlgoTrader provides a custom IDE (Integrated Development Environment) based on Eclipse\(^{18}\). Other Java IDEs (e.g. IntelliJ) or a standard Eclipse IDE\(^{19}\) can be used also, but the AlgoTrader Eclipse Plugins will not be available.

To install the AlgoTrader Eclipse IDE:

- Installation based on AlgoTrader Eclipse Product (Recommended), see Section 2.1.2.1, "Installation based on AlgoTrader Eclipse Product"

### Important

Prior to installing the AlgoTrader Eclipse IDE please make sure that the necessary prerequisites are installed.

2.1.2.1. Installation based on AlgoTrader Eclipse Product

To assist in setting up the AlgoTrader Eclipse IDE, there is a fully configured Eclipse based product which contains all the AlgoTrader bundles and the required Eclipse plugins. Specifically, the AlgoTrader Eclipse product contains:

\(^{15}\) [https://docs.influxdata.com/influxdb/v1.5/query_language/authentication_and_authorization/](https://docs.influxdata.com/influxdb/v1.5/query_language/authentication_and_authorization/)

\(^{16}\) [https://docs.influxdata.com/chronograf/v1.5/](https://docs.influxdata.com/chronograf/v1.5/)

\(^{17}\) [https://docs.grafana.org/datasources/influxdb/](https://docs.grafana.org/datasources/influxdb/)

\(^{18}\) [https://www.eclipse.org/](https://www.eclipse.org/)

\(^{19}\) [https://www.eclipse.org/](https://www.eclipse.org/)
• Eclipse Mars
• AlgoTrader Plugins
• Eclipse Git Team Provider (EGit)
• XML Tools
• Maven Integration For Eclipse (m2e)
• M2E Extensions
• All necessary Eclipse Plugins

The AlgoTrader Eclipse Product can be downloaded from:

• Windows: https://repo.algotrader.ch/eclipse/product/algotrader-win32.win32.x86_64.zip
• Mac: https://repo.algotrader.ch/eclipse/product/algotrader-macosx.cocoa.x86_64.zip
• Linux: https://repo.algotrader.ch/eclipse/product/algotrader-linux.gtk.x86_64.zip

Username and Password are provided when licensing AlgoTrader.

Instructions:

• Download the product according to the Operating System
• Unzip the product zip file
• Start the product clicking on the AlgoTrader executable file
• After starting an Eclipse You need to setup a workspace. Select the default workspace or define a folder to it. If folder in the path doesn't exist, it will be created by default.
• Then open the Java perspective in the top right corner of the screen by clicking on the icon left of the AlgoTrader icon.

2.1.3. AlgoTrader Server Code Installation

The AlgoTrader server code can be installed either via command line or via Eclipse

2.1.3.1. Command Line

To install the AlgoTrader server code via command line please perform the following steps.

2.1.3.1.1. Git Checkout

If one hasn't installed git, please refer to git installation in chapter Section 2.1.1, “Prerequisites”

Perform a Git clone from the command line:
git clone https://gitlab.algotrader.ch/main/Bootstrap.git

Note
user name and password will be provided when signing up for an AlgoTrader license

Note
Please make sure the Maven settings file is updated according to Section 2.1.1, “Prerequisites”.

2.1.3.1.2. Maven Build

Execute the following maven command to build all maven projects

mvn clean install

Note
When running the build process for the first time, this will take a few minutes since all maven dependencies have to be downloaded.

2.1.3.2. Eclipse

To build AlgoTrader from within Eclipse please follow this process.

2.1.3.2.1. Git Checkout

- Inside Eclipse switch to the Java Perspective (Windows --> Open Perspective --> Java)
- Click File / Import / Git / Projects from Git / Clone URI
- Select the following URI https://gitlab.algotrader.ch/main/Bootstrap.git
- Enter User and Password (provided when licensing AlgoTrader)
- Click Next
- Select master
- Click Next
- Select Import existing projects and click Next
- Select the project bootstrap and click finish
• Select the new project bootstrap in the package explorer, right click and select Import / Maven / Existing Maven Projects and select:
  • conf
  • launch

• Click Finish

This will result in the following Eclipse projects:

• algotrader-conf
• algotrader-launch

Note
The compilation will show errors, which should go away after the next section has been completed.

2.1.3.2.2. Maven Build

Before running the maven build from within Eclipse please make sure that the default Eclipse Java runtime environment is pointing to a Java JDK. To verify please go to Window / Preferences / Java / Installed JREs. If the default JRE is pointing to a Java JRE, then please add a reference to the Java JDK which was installed according to Section 2.1.1, “Prerequisites”

Figure 2.1. Eclipse default JRE

To generate the code right click on the project bootstrap inside Eclipse and select / Run As / Maven install. This will generate all maven modules.
Now refresh all projects. Eclipse will compile all java code automatically. In case there is an error message 
Project configuration is not up-to-date with pom.xml... on any of the projects please select: Maven->Update 
Project from the project context menu.

2.1.4. Next Steps

After the AlgoTrader Eclipse IDE has been installed please continue with one of the following steps:

• Start AlgoTrader according to Chapter 3, Starting AlgoTrader
• Start an example trading strategy according to the Appendix of this document
• Create a trading strategy according to Chapter 4, Strategy Development

2.2. Server Environment Installation

AlgoTrader uses Docker\(^\text{20}\) for server side installations.

Docker allows packaging of applications with all of its dependencies into a standardized unit for software 
development.

At the core of the Docker platform is Docker Engine, a lightweight runtime and robust tooling that builds and 
runs Docker containers.

For an in-depth description of Docker please visit the What is Docker\(^\text{21}\) page.

To get started with Docker please visit the Docker Engine Documentation\(^\text{22}\) page.

2.2.1. Docker based Installation

Docker is supported on Linux, Windows, OS X as well as different cloud services (e.g. Amazon AWS or Digital 
Ocean).

Please follow these setups to setup a Docker based AlgoTrader installation:

Install Docker Engine
  according to the Docker Engine installation instructions\(^\text{23}\)

Note
On Mac and Windows please install Docker Toolbox that contains the Docker Engine

Install Docker Compose
  according to the Docker Compose installation instructions\(^\text{24}\)

\(^{20}\) https://www.docker.com/
\(^{21}\) https://www.docker.com/why-docker
\(^{22}\) https://docs.docker.com/engine/
\(^{23}\) https://docs.docker.com/engine/installation/
\(^{24}\) https://docs.docker.com/compose/install/
Copy docker compose file

Copy the following file to the server and make changes as necessary according to Section 2.2.3, “Docker Compose”:


Login to Nexus

Login to the Docker Repository with the username and password provided when licensing AlgoTrader.

docker login docker.algotrader.ch

Run docker compose

Invoke the following command inside the directory where the docker-compose.yml file is located:

docker-compose up -d

Open the AlgoTrader UI

Open the browser and point it to the following URL using the IP retrieved in the previous step.

http://localhost:9090

The above process will setup an AlgoTrader based system made up of the following Docker containers:

• AlgoTrader server
• Interactive Brokers Gateway
• MySQL (with the AlgoTrader MySQL sample data populated)

To startup AlgoTrader with different startup options please visit this chapter on starting AlgoTrader in a Section 3.3, “Server Environment”

To startup one of the AlgoTrader example strategies please visit the appendix of this documentation.

2.2.2. Docker Containers

A typical Docker based AlgoTrader installation is made up of the following Docker containers that can be configured via a docker-compose.yml file.

2.2.2.1. AlgoTrader Container

The AlgoTrader code is located in the directory /usr/local/algotrader inside the Docker container.
The following Docker environment variables are relevant for the AlgoTrader container:

- **ALGOTRADER_HOST** (default: algotrader). This variable is used by strategies to reference the AlgoTrader Docker container.
- **DATABASE_HOST** (default: mysql)
- **DATABASE_PORT** (default: 3306)
- **DATABASE_NAME** (default: algotrader)
- **DATABASE_USER** (default: root)
- **DATABASE_PASSWORD** (default: password)
- **IB_GATEWAY_HOST** (default: ibgateway)
- **IB_GATEWAY_ACCOUNT** optional, only needs to be specified if the IB login has multiple accounts associated
- **INFLUXDB_HOST** (default: influxdb)
- **VM_ARGUMENTS** Additional VM arguments to be added to the java process (e.g. `-DlogLevel=INFO`)
- **SPRING_PROFILES** (default: live,pooledDataSource,iBMarketData,iBNative,iBHistoricalData,embeddedBroker,html5) Spring profiles to be used (comma separated list)
- **STARTER_CLASS** (default: ch.algotrader.starter.ServerStarter)
- **SECRET_FILE** Secret file name (e.g. /run/secrets/api_keys)

In addition the command line switch `-i` can be used to load the MySQL sample data file (samples/db/mysql/mysql-data.sql) on first start up. The sample data will only be loaded if the security table contains no data. To use the `-i` switch please use the following directive inside docker-compose.yml:

```yaml
command: -i
```

The AlgoTrader Docker container will run through the following process on startup:

1. Wait for MySQL to be available. When starting up MySQL and AlgoTrader at the same time (using docker compose) it will take the database a few seconds to become available.

2. Run all Flyway migrate scripts.(see Section 9.2, "Flyway").

3. Load MySQL sample data if the `-i` command line switch is used (see above).

4. Start the AlgoTrader server

### 2.2.2.2. AlgoTrader Strategy Containers

AlgoTrader based trading strategies run in separate Docker containers when running in distributed mode. When running a single strategy in embedded-mode the strategy will run inside the same Docker container as the AlgoTrader server.
The strategy code is located in the directory `/usr/local/strategy` inside the Docker container.

All strategy Docker containers are based on the AlgoTrader Docker container, so environment variables from the AlgoTrader docker container can be reused inside strategy containers.

To build a strategy Docker container the following Dockerfile has to be added to the root of the project:

```
FROM docker.algotrader.ch/algotrader/algotrader:latest

ENV STRATEGY_NAME=XYZ

WORKDIR /usr/local/strategy
ADD target/*.jar lib/

ENTRYPOINT ["*/usr/local/algotrader/bin/docker-strategy-run.sh"]
CMD ["-e"]
```

Please replace `XYZ` with the name of the strategy.

Strategy Docker containers use the `/usr/local/algotrader/bin/docker-strategy-run.sh` startup script that is provided by the AlgoTrader Docker container.

The startup script supports both embedded and distributed mode, see: Section 3.2, “Live Trading Mode”

To start the strategy Docker container in embedded mode please use the `-e` command line switch inside the `docker-compose.yml` file of your strategy:

```
command: -e
```

This will cause the system to run through the following process:

1. Wait for MySQL to be available. When starting up MySQL and AlgoTrader at the same time (using docker compose) it will take the database a few seconds to become available.

2. Run all Flyway migrate scripts.(see Section 9.2, “Flyway”).

3. Load MySQL data from `db/mysql/mysql-data.sql`. MySQL is only loaded if the entry in the strategy table for `STRATEGY_NAME` is missing

4. Start the strategy in embedded mode

To start the strategy Docker container in distributed mode please use the `-d` command line switch inside the `docker-compose.yml` file of your strategy:

```
command: -d
```

This will cause the system to run through the following process:

1. Wait for the AlgoTrader RMI port (1199) to be available
2. Wait for the ActiveMQ JMS port (61616) to be available

3. Start the strategy in distributed mode

When running the system in distributed mode the AlgoTrader server needs to be run in a separate Docker container. Since trading strategies do not have access to the database directly MySQL data needs to be loaded manually by connecting to the database with a MySQL client. It is therefore suggest to follow this process when starting up the system in distributed mode:

1. Startup MySQL, IB Gateway and AlgoTrader

   docker-compose up -d mysql ibgateway algotrader

   **Note**

   Please see the following chapter about changing IB API settings (Read-Only API)

2. Load MySQL data by connecting a MySQL client to port 3306 of the Docker Engine

3. Start strategies

   docker-compose up -d XYZ

   Please replace *XYZ* with the name of the strategy.

### 2.2.2.3. Interactive Brokers Gateway

This container is made up of the following two components:

- Interactive Brokers *IB Gateway*\(^{26}\)

- *IB Controller*\(^{27}\) which allows running IB Gateway in an automated fashion

The following environment variables are relevant for the IB Controller container:

- **TWS_USERNAME** (default: pmdemo)
- **TWS_PASSWORD** (default: demouser)
- **TRADING_MODE** (default: paper)

To run IB Gateway on a headless server (i.e. the Docker container) an *xvfb*\(^{28}\) virtual frame buffer is used.

Unfortunately only few settings of the IB Gateway can be managed via the IB Controller. All other settings have to be managed via the IB Gateway UI itself which is not visible on the Docker container.

---

\(^{27}\) https://github.com/ib-controller/ib-controller  
\(^{28}\) https://en.wikipedia.org/wiki/Xvfb
This is especially cumbersome for the Read-Only API trading mode that is set by default. If this mode is active, placement of orders is not allowed.

To change any of the IB Gateway settings (e.g. Read-Only API trading mode) please execute the following steps:

1. The IB Gateway container stores IB settings inside a Docker Volume\(^{29}\). This volume can be mapped to a local directory as follows.

   On Linux and Mac

   ```
   volumes:
   - /var/lib/tws:/var/lib/tws
   ```

   This will make IB Gateway settings available in the local directory /var/lib/tws

   On Windows

   ```
   volumes:
   - c:/Users/Administrator/Documents/tws:/var/lib/tws
   ```

   This will make IB Gateway settings available in the local directory c:\Users\Administrator\Documents\tws

2. Install and start IB Gateway on a regular workstation (Windows, Mac or Linux)

3. Go to Configure / Settings / API / Settings

4. Make necessary changes (e.g. deselect the Read-Only API check box) and click OK

5. Close the IB Gateway

6. Inside the IB Gateway installation folder there will be one or multiple sub-directories starting which have a name made up of 8-9 characters starting with a the letter d. Please select the directory with the latest time-stamp and makes sure it contains a file named ibg.xml

7. Copy this directory (e.g. darykqwzr) into the IB Gateway settings directory linked above:

8. Copy the jts.ini file into the IB Gateway settings directory linked above:

9. Start the IB Gateway Docker Container:

   ```
   docker-compose create start ibgateway
   ```

\(^{29}\) https://docs.docker.com/storage/volumes/
Note
The above steps will not work for the public pmdemo account which gets reset upon each startup.

2.2.2.4. MySQL
MySQL provides a fully configured Docker container. For further details please visit MySQL on Docker Hub

The following environment variables are relevant for the MySQL container:

- `MYSQL_ROOT_PASSWORD` *(default: password)*
- `MYSQL_DATABASE` *(default: algotrader)*

MySQL data is stored inside a Docker Volume. This volume can be mapped to a local directory as follows.

**On Linux and Mac**

```
  volumes:  
  - /var/lib/mysql:/var/lib/mysql
```

This will make MySQL data available in the local directory `/var/lib/mysql`

**On Windows**

```
  volumes:  
  - c:/Users/Administrator/Documents/mysql:/var/lib/mysql
```

This will make MySQL data available in the local directory `c:\Users\Administrator\Documents\mysql`

2.2.2.5. InfluxDB
InfluxDB provides a fully configured Docker container. For further details please visit InfluxDB on Docker Hub

InfluxDB data is stored inside a Docker Volume. This volume can be mapped to a local directory as follows.

**On Linux and Mac**

```
  volumes:  
  - /var/lib/influxdb:/var/lib/influxdb
```

**On Windows**

```
  volumes:  
  - c:/Users/Administrator/Documents/influxdb:
```

This will make InfluxDB data available in the local directory `c:\Users\Administrator\Documents\influxdb`

---

30 [https://hub.docker.com/_/mysql/](https://hub.docker.com/_/mysql/)
31 [https://docs.docker.com/storage/volumes/](https://docs.docker.com/storage/volumes/)
32 [https://hub.docker.com/_/influxdb/](https://hub.docker.com/_/influxdb/)
33 [https://docs.docker.com/storage/volumes/](https://docs.docker.com/storage/volumes/)
This will make MySQL data available in the local directory `/var/lib/mysql`.

On Windows:

```bash
volumes:
- c:/Users/Administrator/Documents/influxdb:/var/lib/influxdb
```

This will make MySQL data available in the local directory `c:\Users\Administrator\Documents\influxdb`.

### 2.2.3. Docker Compose

Docker based applications typically consist of many small applications that work together. Docker transforms these applications into individual containers that are linked together. Instead of having to build, run and manage each individual container, Docker Compose allows definition of multi-container application with all of its dependencies in a single file, then startup the application in a single command. The application's structure and configuration are held in a single place, which makes starting up applications simple and repeatable everywhere.

For further details regarding Docker Compose please visit the [Docker Compose documentation](https://docs.docker.com/compose/).

Docker Compose uses `docker-compose.yml` files to configure multi-container applications. AlgoTrader ships with a default `docker-compose.yml` file located inside the top-level AlgoTrader project directory:

```yaml
algotrader:
  image: docker.algotrader.ch/algotrader/algotrader
  command: -i
  environment:
    - VM_ARGUMENTS=-Dkeygen.id=...
  links:
    - mysql
    - ibgateway
    - influxdb
  ports:
    - 9090:9090
    - 61614:61614

ibgateway:
  image: docker.algotrader.ch/interactivebrokers/ibgateway
  environment:
    TWS_USERNAME: pndemo
    TWS_PASSWORD: demouser
  volumes:
    - /var/lib/tws

mysql:
```

[34](https://docs.docker.com/compose/)
Using this `docker-compose.yml` file will create Docker containers for AlgoTrader, IB Gateway, MySQL and InfluxDB. The following items are present in the file:

- **service name**: e.g. `algotrader`, `ibgateway`
- **image**: the name of the image to use in the format `namespace/repository:version` (e.g. `docker.algotrader.ch/algotrader/algotrader:latest`)
- **command**: command line argument to pass to the Docker container (e.g. `-i`). See Section 2.2.2, “Docker Containers” for supported command line arguments.
- **links**: services to link to this container (e.g. `mysql & ibgateway`). Adding a link will make the target container accessible with the correct IP by its link name.
- **ports**: ports to map on the host machine. E.g. `3306:3306` will map port 3306 of the MySQL container to 3306 of the host machine.
- **environment**: environment variables to use. See Section 2.2.2, “Docker Containers” for supported environment variables.
- **volumes**: Docker Data volumes to create. E.g. `c:/Users/mysql:/var/lib/mysql` will map the directory `/var/lib/mysql` inside the container to `c:\mysql` on the host machine.

### 2.2.3.1. Passwords

The example `docker-compose.yml` file specifies multiple passwords directly. For security purposes this often not advisable. As an alternative passwords can be stored using Docker secrets.

### 2.2.3.2. Config Files

All properties within the AlgoTrader `*.properties` files can be overwritten using the docker environment variable `VM_ARGUMENTS` as mentioned in Section 2.2.2.1, “AlgoTrader Container”.

---

35 https://docs.docker.com/engine/swarm/secrets/
If a large number of properties need to be changed or if other config files need to be changed (e.g. `fix.cfg`) it is recommended to follow this process:

1. start a local instance of the AlgoTrader container using the command

   ```
   docker run -i -t --entrypoint /bin/bash docker.algotrader.ch/algotrader/algotrader
   root@9f9adac97f3c:/usr/local/algotrader#
   ```

   Take note of the container ID that has been created, 9f9adac97f3c

2. make changes to any of the config files inside `/usr/local/algotrader/conf`

3. Exit the modified AlgoTrader container using `exit`

4. Commit the change to the modified AlgoTrader container using the following command

   ```
   docker commit CONTAINER_ID docker.algotrader.ch/algotrader/algotrader:VERSION
   ```

   using the `CONTAINER_ID` retrieved above and setting a new `VERSION`

It is also possible to override the entire `/usr/local/algotrader/conf` file using Docker volumes.

It can be done by putting additional volume to algotrader container in `docker-compose.yml` file:

```yaml
volumes:
  - ./conf:/usr/local/algotrader/conf/
```

And copying all necessary config files (`fix.cfg` included) to the created `.conf` folder on local machine.

### 2.2.3.3. Encrypting sensitive configuration with Docker Secrets

For security reasons, it is recommended to store sensitive configuration (e.g. adapter API key and API secret) in encrypted form.

Docker provides a solution to called docker secrets. [Docker Secrets documentation](https://docs.docker.com/engine/swarm/secrets/)

To store config values as docker secrets one needs to follow these steps:

1. prepare the temporary file `myKeys.txt` with the sensitive data

   For example by copying the following properties from `conf-bmx.properties`:

   ```
   bmx.apiKey=revdPMrxxxxxxxxxfdsPo
   ```

[36](https://docs.docker.com/storage/volumes/)
[37](https://docs.docker.com/engine/swarm/secrets/)
2. create a new secret api_keys using docker secret command

   docker secret create api_keys myKeys.txt

3. remove the temporary file myKeys.txt

4. use the following docker-compose.yml with references to the api_keys secret

   Note the additional section secrets: at the bottom of the file:

```yaml
version: '3.1'

services:

  algotrader:
    image: docker.algotrader.ch/algotrader/algotrader
    command: -i
    environment:
      SPRING_PROFILES: live,pooledDataSource,bMX,bMXMarketData,bMXAccount,embeddedBroker,html5
      SECRET_FILE: /run/secrets/api_keys
    secrets:
      - api_keys
    deploy:
      restart_policy:
        condition: none
    ports:
      - 9090:9090
      - 61614:61614

  ibgateway:
    image: docker.algotrader.ch/interactivebrokers/ibgateway
    environment:
      TWS_USERNAME: pmdemo
      TWS_PASSWORD: demouser
    volumes:
      - /var/lib/tws
    depends_on:
      - algotrader

  mysql:
    image: mysql:5.7.22
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: algotrader
    ports:
- 3306:3306
  volumes:
  - /var/lib/mysql
  depends_on:
  - algotrader

influxdb:
  image: influxdb:1.5.2
  ports:
  - 8086:8086
  volumes:
  - /var/lib/influxdb
  depends_on:
  - algotrader

secrets:
  api_keys:
    external: true

---

**Note**

Docker secrets as shown above are only available to swarm services, not to standalone containers. To use this feature, algotrader must run as a service with a scale of 1, on a single node cluster.

Prior to creating and using secrets initialize machine as swarm manager using following command:

```
  docker swarm init
```

Operating in Swarm mode means using different docker commands, see examples below.

To deploy and run a services use **docker stack** instead of **docker-compose**

```
  docker stack deploy --compose-file=docker-compose.yml mystack
```

To look at logs:

```
  docker service logs -f mystack_algotrader
```

Also, the algotrader network has to be created with swarm scope:

```
  docker network create algotrader --scope swarm --driver overlay
```

Otherwise the docker swarm will not be able to use it and the algotrader image will not start. To create Docker secret from e.g. *myKeys.txt* file use:

---
docker secret create api_keys myKeys.txt

Please also refer to Section 26.1.1, “Encrypting sensitive configuration values”

2.2.3.4. Log Files

It is a Docker best-practice to have only one running process per Docker container. Typically application output is logged directly to the console where it can be viewed using the command `docker logs` according to the Docker documentation

Sometimes this is not enough and one wishes to write additional information (e.g. fix messages) to a separate log file. To get access to this log file from outside the container it is advised to create an additional volume:

```
volumes:
    - ~/fix.log:/usr/local/algotrader/logs/fix.log
```

2.2.4. Docker Management

In addition to using the Docker command line, several options exist for management of docker based installations

2.2.4.1. Portainer

Portainer is another alternative Docker web interface.

To use Docker UI please add the following to the `docker-compose.yml` file:

38 https://docs.docker.com/engine/reference/commandline/logs/
docker-ui:
  image: portainer/portainer
  command: -H unix:///var/run/docker.sock
  ports:
    - 9000:9000
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock

For further details please visit Docker UI on Docker Hub\(^{39}\) and Docker UI on GitHub\(^{40}\).

### 2.2.4.2. Kitematic

When running Docker on Windows or Mac Docker Kitematic\(^{41}\) provides a UI for management of the Docker engine.

---

### 2.3. VM Arguments

Many characteristics of the system can be customized with VM arguments, the following list provides an overview of commonly used VM arguments.

- `-DlogLevel`  
  log4j log level (ERROR, WARN, INFO or DEBUG)

---

\(^{39}\) [https://hub.docker.com/r/dockerui/dockerui/](https://hub.docker.com/r/dockerui/dockerui/)

\(^{40}\) [https://github.com/kevana/ui-for-docker](https://github.com/kevana/ui-for-docker)

\(^{41}\) [https://kitematic.com/](https://kitematic.com/)
-Dspring.profiles.active
  list of Spring Profiles to activate (see Section 25.1, “Starter Classes”)

-Xmx
  increase the Java Heap Size to specified amount (e.g. 2048M)

AlgoTrader specific configuration parameters can be changed inside the .properties files. As an alternative configuration parameters can also be provided as VM arguments in which case they will overwrite existing parameters inside *.properties files.

-Dstatement.closePosition=false
Starting AlgoTrader

As a first step one needs to make sure that the appropriate algotrader license key is properly configured. The license key was provided in the Email after signing up for the AlgoTrader free 30-day trial or when purchasing an algotrader license. The license key needs to be configured inside the file `algotrader-conf/src/main/resources/conf.properties` as follows:

Now AlgoTrader can be started through various Eclipse Run Configurations which are available to launch the various operation modes of AlgoTrader. To access the available Eclipse Run Configurations follow screen shots below. For more information on Eclipse Run Configurations, please see https://wiki.eclipse.org/FAQ_What_is_a_launch_configuration.

Open Run Configurations

Select the downward facing black arrow next to the green run symbol

![Image of Eclipse Run Configurations](image)

**Figure 3.1. Eclipse Run Configurations**

Select Run Configuration

Select the Run Configuration in the list on the left. The right hand side will show the project the Run Configuration will start in as well as the Main Java class

---

1 https://wiki.eclipse.org/FAQ_What_is_a_launch_configuration%3F
Figure 3.2. Eclipse Run Configurations
Update program and VM arguments if necessary

![Eclipse Run Configurations](image)

**Figure 3.3. Eclipse Run Configurations**

### 3.1. Simulation Mode

To run AlgoTrader in Simulation Mode and perform a back test of a strategy the class `ch.algotrader.starter.SimulationStarter` has to be invoked. This will run the strategy using the embedded in-memory database h2.

In Eclipse Run Configurations named `SimulationStarter-simulate-xxx` are provided which contain the following items:

**Main-Tab**

- **Project**: strategy project
- **Main Class**: `ch.algotrader.starter.SimulationStarter`

**Arguments-Tab / Program Arguments**

```
simulateWithCurrentParams
```

**Arguments-Tab / VM Arguments**

```
-Dsimulation=true
-DstrategyName=XXX
```
3.2. Live Trading Mode

To run AlgoTrader in Live Trading Mode the corresponding Adapter (e.g. local broker client, Fix session, VPN Connection, etc.) needs to be up and running.

If using InteractiveBrokers the Trader Workstation or the IB Gateway have to be running with the following configurations under API/Settings:

- Enable ActiveX and Socket Clients
- Read-Only Mode: false
- Socket Port: 4001
- Trusted IP Addresses: 127.0.0.1

When running AlgoTrader in Live Trading Mode the AlgoTrader Server and the Strategies can either be run in separate JVM’s (distributed mode) or the entire system can be run within one single JVM (embedded mode).

Before starting AlgoTrader check the database table strategy. The column AUTO_ACTIVATE should be set to true for records corresponding to the AlgoTrader Server and the trading strategy one wants to trade.

3.2.1. Embedded Mode

In Embedded Mode both the AlgoTrader Server and the Strategy run within the same JVM.

Note

Only one Strategy can be run at once in Embedded Mode

In Eclipse Run Configurations named EmbeddedStrategyStarter-xxx are provided which contain the following items:

Main-Tab

- Project: strategy project
3.2.2. Distributed Mode

In Distributed Mode the AlgoTrader Server and the Strategy / Strategies run in separate JVM's and have to be started separately.

To Start the AlgoTrader Server in distributed mode the Eclipse launch configurations ServerStarterXX is provided which contain the following items:

Main-Tab

- Project: algotrader-core
- Main Class: ch.algotrader.starter.ServerStarter

Arguments-Tab / VM Arguments

- Dspring.profiles.active=live,<dataSource>,<marketDataProfile>,<brokerProfile>,embeddedBroker,html5

- dataSource: the hibernate datasource to use, either: pooledDataSource, singleDataSource

- marketDataProfile: the SpringProfile corresponding to the market data interface in use (e.g. iBMarketData or bBMarketData)

- tradingProfile: the SpringProfile corresponding to the adapter in use (e.g. iBNative or iBFix)
To start a Strategy Eclipse Run Configurations StrategyStarter-xxx are provided which contain the following items:

**Main-Tab**
- **Project:** strategy project
- **Main Class:** ch.algotrader.starter.StrategyStarter

**Arguments-Tab / VM Arguments**

```
-DstrategyName=TEST
-Dspring.profiles.active=live
```

### 3.3. Server Environment

AlgoTrader uses Docker for server environment installations. When using Docker this various components of the system as well as their configurations are managed through `docker-compose.yml` files. As a first step when using The AlgoTrader Docker server environment is to configure the AlgoTrader license key within the `docker-compose.yml` file by replacing the `...` with the license key that was provided in the Email after signing up for the AlgoTrader free 30-day trial or when purchasing an algotrader license.

```
xyz:
  image: xyz
  command: -d
  environment:
    - VM_ARGUMENTS=-Dkeygen.id=...

algotrader:
  image: docker.algotrader.ch/algotrader/algotrader
  container_name: algotrader
  environment:
    - VM_ARGUMENTS=-Dkeygen.id=...
```

### 3.3.1. Embedded Mode

In Embedded Mode both the AlgoTrader Server and the Strategy run within the same JVM.

**Note**

Only one Strategy can be run at once in Embedded Mode

To run the system in embedded mode create a `docker-compose.yml` file similar to the following:

```
xyz:
  image: xyz
  command: -e
  environment:
    - VM_ARGUMENTS=-Dkeygen.id=...
  links:
    - mysql
    - ibgateway
    - influxdb
  ports:
    - 9090:9090
    - 61614:61614
  environment:
    STRATEGY_NAME: XYZ

ibgateway:
  image: docker.algotrader.ch/interactivebrokers/ibgateway
  environment:
    TWS_USERNAME: pmdemo
    TWS_PASSWORD: demouser
  volumes:
    - /var/lib/tws

mysql:
  image: mysql:5.7.22
  environment:
    MYSQL_ROOT_PASSWORD: password
    MYSQL_DATABASE: algotrader
  ports:
    - 3306:3306
  volumes:
    - /var/lib/mysql

influxdb:
  image: influxdb:1.5.2
  ports:
    - 8086:8086
  volumes:
    - /var/lib/influxdb

Please replace xyz/XYZ with the name of the trading strategy. Please refer to Chapter 4, Strategy Development on how to create a new trading strategy.

To start the system in embedded mode please run the following command from the directory where the docker-compose.yml file is located:

docker-compose up -d
This will create the following docker containers: strategy (xyz), ibgateway & mysql

For further details please see Section 2.2.3, “Docker Compose”

3.3.2. Distributed Mode

In Distributed Mode the AlgoTrader Server and the Strategy / Strategies run in separate JVM's and have to be started separately.

To run the system in distributed mode create a docker-compose.yml file similar to the following:

```yaml
xyz:
  image: xyz
  command: -d
  environment:
    - VM_ARGUMENTS=-Dkeygen.id=...
  links:
    - algotrader
  environment:
    STRATEGY_NAME: XYZ

algotrader:
  image: docker.algotrader.ch/algotrader/algotrader
  container_name: algotrader
  environment:
    - VM_ARGUMENTS=-Dkeygen.id=...
  links:
    - mysql
    - ibgateway
    - influxdb
  ports:
    - 9090:9090
    - 61614:61614

ibgateway:
  image: docker.algotrader.ch/interactivebrokers/ibgateway
  container_name: ibgateway
  environment:
    TWS_USERNAME: pmdemo
    TWS_PASSWORD: demouser
  volumes:
    - /var/lib/tws

mysql:
  image: mysql:5.7.22
  container_name: mysql
  environment:
    MYSQL_ROOT_PASSWORD: password
    MYSQL_DATABASE: algotrader
```
ports:
  - 3306:3306
volumes:
  - /var/lib/mysql

influxdb:
  image: influxdb:1.5.2
  ports:
    - 8086:8086
  volumes:
    - /var/lib/influxdb

Please replace \textit{xyz} / \textit{XYZ} with the name of the trading strategy.

To start the system in distributed mode please run the following command from the directory where the \texttt{docker-compose.yml} file is located:

\begin{verbatim}
docker-compose up -d
\end{verbatim}

This will create the following docker containers: strategy (xyz), algotrader, ibgateway & mysql

For further details please see \textit{Section 2.2.3, “Docker Compose”}
Strategy Development

Warning

It is recommended to perform thorough Simulation / Back Testing of newly developed strategies. After that the strategy should be tested with a Paper Trading Account. At the end of a thorough test procedure, the new strategy can be put into production. At the beginning of live trading it is recommended to use a small trading account only.

The following diagram shows the general procedure for developing new strategies:

Figure 4.1. Strategy Development Process

4.1. Creating a Trading Strategy

The following paragraph will give a short example based on a simple moving average strategy (with the Short Name \textit{EMA}).

The following 2 options can assist in creating a new trading strategy, the AlgoTrader Strategy Wizard and the AlgoTrader Maven Archetype.

In addition to this setup, you will need to create a database entry for your strategy. Please refer to \textit{the strategy table definition}.

4.1.1. AlgoTrader Strategy Wizard

The Strategy Wizard provides an easy way to automatically create all artifacts necessary for an AlgoTrader based trading strategy. Internally the Strategy Wizards makes use of the AlgoTrader Archetype, see
Section 4.1, “Creating a Trading Strategy”. The Strategy Wizard provides options for three different types of Trading Strategies:

- Esper based Strategies
- Java based Strategies (without Esper)
- Simple Strategies, consisting of only one file (without Esper)

Inside Eclipse switch to the Java Perspective. The Strategy Wizard can be started via the File / New / Other which will bring up the following screen where the Maven Project wizard can be selected:

Figure 4.2. Wizard Selection

On the next step the location for the newly created project (trading strategy) as well as the Eclipse working set can be selected.

Figure 4.3. Wizard Location and Working Set

On the next screen please select the AlgoTrader Catalog and algotrader-archetype-esper for Esper based strategies or algotrader-archetype-java for standard strategies (without Esper) or algotrader-archetype-simple for extremely basic strategies

Figure 4.4. Wizard Maven Properties

Group Id
The maven group id (e.g. algotrader), all lower-case, can contain periods

Artifact Id
The maven artifact id (e.g. algotrader-ema), all lower-case, can contain dashes

Version
The maven version (e.g. 1.0.0-SNAPSHOT), x.y.z, plus optionally -SNAPSHOT

Package
The java package name (ch.algotrader.strategy), all lower-case, can contain periods.

name
The name of the strategy (e.g. ema), all lower-case, no periods, no dashes

serviceName
The name of the strategy service (e.g. EMA), first letter upper-case or all upper-case, do not include Service at the end (e.g. do not specify EMAService)
Important

For Spring Auto-Wiring to work the package name needs to be ch.algotrader.strategy. If a different package is assigned services (e.g. OrderService and LookupService) will not be available.

For all of these items previously entered values can be reused by clicking the combo-box to the right of the field.

Figure 4.5. Wizard Maven Properties

When clicking finish the Strategy Wizard will create a new Eclipse project using the AlgoTrader Artifact.

4.1.2. AlgoTrader Maven Archetype

The AlgoTrader Maven Archetype is a project template that can be used to create a new AlgoTrader trading Strategy. To use the Maven Archetype execute the following command from the command line in a new empty directory.

To create Esper bases strategies execute (replace <version> with the corresponding AlgoTrader version):

```
mvn archetype:generate -DarchetypeGroupId=algotrader -DarchetypeArtifactId=algotrader-archetype-esper -DarchetypeVersion=<version>
```

To create Java strategies execute (replace <version> with the corresponding AlgoTrader version):

```
mvn archetype:generate -DarchetypeGroupId=algotrader -DarchetypeArtifactId=algotrader-archetype-java -DarchetypeVersion=<version>
```

To create simple strategies consisting of only one single java file execute (replace <version> with the corresponding AlgoTrader version):

```
mvn archetype:generate -DarchetypeGroupId=algotrader -DarchetypeArtifactId=algotrader-archetype-simple -DarchetypeVersion=<version>
```

The Maven Archetype will ask for the following input parameters:

- **groupId**
  - The maven group id (e.g. algotrader), all lower-case, can contain periods

- **artifactId**
  - The maven artifact id (e.g. algotrader-ema), all lower-case, can contain dashes

- **version**
  - The maven version (e.g. 1.0.0-SNAPSHOT), x.y.z, plus optionally -SNAPSHOT
packageName
   The java package name (ch.algotrader.strategy), all lower-case, can contain periods.

name
   The name of the strategy (e.g. ema), all lower-case, no periods, no dashes

serviceName
   The name of the strategy service (e.g. EMA), first letter upper-case or all upper-case, do not include Service at the end (e.g. do not specify EMAService)

Important
   For Spring Auto-Wiring to work the package name needs to be ch.algotrader.strategy. If a different package is assigned services (e.g. OrderService and LookupService) will not be available.

4.1.3. Generated Artifacts Java Archetype

The Java Archetype will generate the following artifacts:

/src/main/java/ch/algotrader/strategy/EMAService.java
   The strategy service class

/src/main/java/ch/algotrader/strategy/EMAServiceConfig.java
   The strategy configuration class

/src/main/java/ch/algotrader/strategy/Metrics.java
   A sample class that can be used within the strategy

/src/main/java/ch/algotrader/strategy/State.java
   A sample enum that can be used within the strategy

/launch/*.launch
   Eclipse Run Configurations to start the Strategy in embedded mode and simulation mode

/pom.xml
   The Maven project object model file containing general information about the Trading Strategy

/Dockerfile
   The Docker file

4.1.3.1. EMAService.java

This is the main Java-class containing the Business Logic.

The references to the Services provided by the AlgoTrader Server (e.g. OrderService, PositionService, etc.) will be injected on startup by the Spring Framework
public class EMAService extends ConfigAwareStrategyService<EMAConfig> {

    @Override
    public void onStart(final LifecycleEventVO event) {
        getSubscriptionService().subscribeMarketDataEvent(getStrategyName(), getConfig().getSecurityId());
    }

    @Override
    public void onBar(BarVO bar) {
        MarketOrderVO order = MarketOrderVOBuilder.create()
            .setStrategyId(getStrategy().getId())
            .setAccountId(getConfig().getAccountId())
            .setSecurityId(getConfig().getSecurityId())
            .setQuantity(new BigDecimal(getConfig().getOrderQuantity()))
            .setSide(bar.getClose().compareTo(bar.getOpen()) > 0 ? Side.BUY : Side.SELL)
            .build();

        getOrderService().sendOrder(order);
    }
}

The class EMAService method contains the following items:

1. Once the strategy has reached the START live cycle phase subscribe to the security needed for this strategy.
2. Construct an Order Value Object using the MarketOrderVOBuilder. The OrderVO contains a reference to the strategy, the security, the account as well as the quantity and the order side.
3. Send the Order to the market via the OrderService.

4.1.3.2. pom.xml

The Maven pom.xml file contains the Maven project definition as well as Maven dependencies:
4.1.3.3. Dockerfile

The Dockerfile contains all relevant information to build a Docker container:

FROM docker.algotrader.ch/algotrader/algotrader:latest

ENV STRATEGY_NAME=EMA

WORKDIR /usr/local/strategy
ADD target/*.jar lib/

ENTRYPOINT ["/usr/local/algotrader/bin/docker-strategy-run.sh"]
CMD ["-e"]

4.1.4. Generated Artifacts Simple Java Archetype

The Simple Java Archetype will generate the following artifacts:
The strategy service class

Eclipse Run Configurations to start the Strategy in embedded mode and simulation mode

The Maven project object model file containing general information about the Trading Strategy

The Docker file

4.1.4.1. EMAService.java

This is the main Java-class containing the Business Logic.

The references to the Services provided by the AlgoTrader Server (e.g. OrderService, PositionService, etc.) will be injected on startup by the Spring Framework

```java
public class EMAService extends ConfigAwareStrategyService<EMAConfig> {

    @Override
    public void onStart(final LifecycleEventVO event) {
        getSubscriptionService().subscribeMarketDataEvent(getStrategyName(), get_Config().getSecurityId(), AdapterType.IB.name());
    }

    @Override
    public void onBar(BarVO bar) {

        MarketOrderVO order = MarketOrderVOBuilder.create()
            .setStrategyId(getStrategy().getId())
            .setAccountId(get_Config().getAccountId())
            .setSecurityId(get_Config().getSecurityId())
            .setQuantity(new BigDecimal(get_Config().getOrderQuantity()))
            .setSide(bar.getClose().compareTo(bar.getOpen()) > 0 ? Side.BUY : Side.SELL)
            .build();

        getOrderService().sendOrder(order);
    }
}
```

The class EMAService method contains the following items:

1. Once the strategy has reached the START live cycle phase subscribe to the security needed for this strategy
Construct an Order Value Object using the `MarketOrderVOBuilder`. The `OrderVO` contains a reference to the strategy, the security, the account as well as the quantity and the order side.

Send the Order to the market via the `OrderService`.

### 4.1.4.2. `pom.xml`

The Maven `pom.xml` file contains the Maven project definition as well as Maven dependencies:

```xml
  <modelVersion>4.0.0</modelVersion>
  <groupId>algotrader</groupId>
  <artifactId>algotrader-ema</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>ema strategy</name>

  <build>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>${java.version}</source>
          <target>${java.version}</target>
        </configuration>
      </plugin>
    </plugins>
  </build>

  <dependencies>
    <dependency>
      <groupId>algotrader</groupId>
      <artifactId>algotrader-core</artifactId>
      <version>...</version>
    </dependency>
  </dependencies>

  <properties>
    <java.version>1.8</java.version>
  </properties>
</project>
```

### 4.1.4.3. Dockerfile

The Dockerfile contains all relevant information to build a Docker container:
FROM docker.algotrader.ch/algotrader/algotrader:latest

ENV STRATEGY_NAME=EMA

WORKDIR /usr/local/strategy
ADD target/*.jar lib/

ENTRYPOINT ["/usr/local/algotrader/bin/docker-strategy-run.sh"]
CMD ["-e"]

### 4.1.5. Generated Artifacts Esper Archetype

The Esper Archetype will generate the following artifacts:

- `/src/main/java/ch/algotrader/strategy/EMAService.java`
  The strategy service class

- `/src/main/resources/module-ema.epl`
  Esper Module containing statements related to signal generation

- `/src/main/resources/conf-ema.properties`
  Contains parameters used by the strategy (e.g. Moving average durations etc.)

- `/src/main/resources/META-INF/esper-ema.cfg.xml`
  Contains event-types, imports, variables and general Esper settings

- `/src/main/resources/META-INF/applicationContext-client-ema.xml`
  Spring Application Context File for the strategy

- `/src/main/resources/db/mysql/mysql-ema.sql`
  MySQL data file containing db data needed for this trading strategy in live trading mode

- `/launch/*.launch`
  Eclipse Run Configurations to start the Strategy in embedded mode and simulation mode

- `/pom.xml`
  The Maven project object model file containing general information about the Trading Strategy

- `/Dockerfile`
  The Docker file

#### 4.1.5.1. EMAService.java

This is the main Java-class containing the Business Logic.

The references to the Services provided by the AlgoTrader Server (e.g. OrderService, PositionService, etc.) will be auto injected on startup by the Spring Framework
public class EMAService extends StrategyService {

    private @Value("#{emaConfigParams.accountId}") long accountId;
    private @Value("#{emaConfigParams.securityId}") long securityId;
    private @Value("#{emaConfigParams.orderQuantity}") long orderQuantity;

    @Override
    public void onStart(final LifecycleEventVO event) {
        getSubscriptionService().subscribeMarketDataEvent(getStrategyName(), this.securityId, AdapterType.IB.name());
    }

    public void sendOrder(Side side) {

        MarketOrderVO order = MarketOrderVOBuilder.create()
            .setStrategyId(getStrategy().getId())
            .setAccountId(this.accountId)
            .setSecurityId(this.securityId)
            .setQuantity(this.orderQuantity)
            .setSide(side)
            .build();

        getOrderService().sendOrder(order);
    }
}

The class EMAService method contains the following items:

1. Gets references to settings defined in conf-ema.properties
2. Once the strategy has reached the START live cycle phase subscribe to the security needed for this strategy
3. Construct an Order Value Object using the MarketOrderVOBuilder. The OrderVO contains a reference to the strategy, the security, the account as well as the quantity and the order side.
4. Send the Order to the market via the OrderService

4.1.5.2. module-ema.epl

This Esper module contains the Esper statements for signal generation

The statement MOVING_AVERAGE generate signals by using the moving average function.

the statement SEND_ORDER calls the EMAService.sendOrder() whenever there is a moving average crossover on the indicators. The Tag @Subscriber is used to instruct the EsperEngine to attach the Subscriber to this statement. This way the sendOrder method is called whenever there is a signal.
CONFIDENTIAL

Generated Artifacts Esper Archetype

insert into
Indicator
select
ema(currentValue, movingAveragePeriodShort) ema(currentValue, movingAveragePeriodLong) as value
from
TickVO;
@Name('SEND_ORDER')
@Subscriber(className='emaService#sendOrder')
select
case when indicator.value > 0 then Side.BUY else Side.SELL end as side
from
pattern [every indicator=Indicator]
where
(indicator.value > 0 and prior(1, indicator.value) <= 0)
or
(indicator.value < 0 and prior(1, indicator.value) >= 0)

4.1.5.3. conf-ema.properties
This configuration file contains the parameters of the strategy:
#{"type":"Integer","label":"Account ID"}
accountId = 1
#{"type":"Integer","label":"Security ID"}
securityId = 1
#{"type":"Integer","label":"Default Order Quantity"}
orderQuantity = 100000
#{"type":"Integer","label":"Moving Average Period Short"}
movingAveragePeriodShort = 10
#{"type":"Integer","label":"Moving Average Period Long"}
movingAveragePeriodLong = 20

This file uses special JSON comments that are used by the AlgoTrader Config Editor, see Section 10.2.3,
“AlgoTrader Configuration Editor”

4.1.5.4. esper-ema.cfg.xml
The esper configuration file looks like this:

<?xml version="1.0" encoding="UTF-8"?>
<esper-configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

45


The above file configures the required variables along with their type. The actual values for the variables are taken from conf-ema.properties.

4.1.5.5. applicationContext-client-ema.xml

A typical Spring Configuration File looks like this:

```xml
<beans xmlns="http://www.springframework.org/schema/beans" 
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" 
xmlns:p="http://www.springframework.org/schema/p" 
xsi:schemaLocation="http://www.springframework.org/schema/beans 
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
  <bean id="emaConfigParams" 
    class="ch.algotrader.config.spring.CustomConfigParamsFactoryBean">
    <property name="global" ref="configParams"/>
  </bean>
  <bean id="emaEngine" 
    class="ch.algotrader.esper.EngineFactoryBean">
    <property name="strategyName" value="EMA"/>
    <property name="configResource" value="esper-ema.cfg.xml"/>
    <property name="configParams" ref="emaConfigParams"/>
    <property name="initModules" value="test"/>
  </bean>
  <bean id="emaService" 
    class="ch.algotrader.EMAService" autowire="byName">
    <property name="strategyName" value="EMA"/>
    <property name="engine" ref="emaEngine"/>
  </bean>
</beans>
```
This file contains the following Spring Bean Definitions:

1. contains a Map of all properties based on settings defined in `conf-ema.properties`
2. Creates the Esper Engine based on `strategyName`, `configResource`, `configParams` and `initModules` and optional `runModules` definitions
3. Creates the Strategy Service based on `strategyName` definition and `engine` reference. All dependencies of the `ch.algotrader.service.StrategyService` will be injected automatically through auto wiring

### 4.1.5.6. mysql-ema.sql

The MySQL database script contains the following items:

```sql
INSERT INTO 'strategy' (`ID`, `NAME`, `AUTO_ACTIVATE`, `VERSION`) VALUES
(2, 'EMA', True, 0);
```

The file contains an entry in the table strategy. The column `AUTO_ACTIVATE` means that the strategy will be automatically run in simulation mode.

### 4.1.5.7. pom.xml

The Maven `pom.xml` file contains the Maven project definition as well as Maven dependencies:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>algotrader</groupId>
  <artifactId>algotrader-ema</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>ema strategy</name>

  <build>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>${java.version}</source>
          <target>${java.version}</target>
        </configuration>
      </plugin>
    </plugins>
  </build>

  <dependencies>
    <dependency>
      <groupId>algotrader</groupId>
```
4.1.5.8. Dockerfile

The Dockerfile contains all relevant information to build a Docker container:

```
FROM docker.algotrader.ch/algotrader/algotrader:latest

ENV STRATEGY_NAME=EMA

WORKDIR /usr/local/strategy
ADD target/*.jar lib/

ENTRYPOINT ["/usr/local/algotrader/bin/docker-strategy-run.sh"]
CMD ["-e"]
```

4.2. Building a Trading Strategy

Execute the following Maven command to start a Maven build of the trading strategy:

```
mvn install
```

The Maven modules can now be deployed to a Maven repository (e.g. Sonatype Nexus) using:

```
mvn deploy
```

For further details regarding a maven deploy please visit the [Maven deploy plug-in](https://maven.apache.org/plugins/maven-deploy-plugin/) page

Execute the following Docker command to create a Docker image of the trading strategy that can be used for productive deployments:

```
docker build -t xyz .
```

Please replace `xyz` with the name of the trading strategy.
The Docker image can now be pushed to a Docker repository (e.g. Docker Hub, Sonatype Nexus or Amazon ECR). For further details on pushing to a Docker repository please visit:

- **Docker Hub** [²]
- **Sonatype Nexus 3.0** [³]
- **Amazon ECR** [⁴]

### 4.3. Hints for Strategy Development

It is possible to develop trading strategies purely in Java for simplicity. Very often it is though helpful to use Esper for Market Data Analysis and Signal Generation in addition to Java code. The following sections will provide hints on developing strategies both in Java and Esper.

#### 4.3.1. Java based Strategies

Java bases Strategies typically consist of a single Java class where all logic is implemented inside event handler methods (e.g. `onInit`, `onBar`, `onTick`, `onOrderStatus`, etc.)

##### 4.3.1.1. Strategy starters

The following two Starters are available to start a trading strategy in embedded and in distributed mode.

**StrategyStarter**

`StrategyStarter` starts a strategy in stand-alone mode running in a separate JVM process. The strategy will use ActiveMQ message broker to receive market data and other events from the server JVM process. The server JVM process is expected to be running before the strategy JVM is started.

**EmbeddedStrategyStarter**

`EmbeddedStrategyStarter` starts a strategy in single JVM (embedded) mode, when server and strategy run in the same process. Market data and other events are delivered directly to the strategy instances by a single event dispatcher.

##### 4.3.1.2. Event Handler Methods

AlgoTrader is an event based system. All strategy related events are propagated to strategies as event objects (e.g. `Order`, `OrderStatus`, `Tick`, `Bar`, etc.). Inside strategies these events are made available through event handler methods, e.g.:

```java
@override
public void onBar(BarVO bar) {
    // do something
}
```

---

² https://docs.docker.com/docker-hub/repos/#pushing-a-repository-image-to-docker-hub
³ https://support.sonatype.com/hc/en-us/articles/360000761828
⁴ https://docs.aws.amazon.com/AmazonECR/latest/userguide/docker-push-ecr-image.html
4.3.1.3. Life-Cycle Events

Strategy classes can provide listeners for life-cycle events in order to receive notifications about strategy life-cycle phase transitions and execute custom life-cycle dependent logic.

```java
@Override
public void onInit(final LifecycleEventVO event) {
    // do something
}
```

```java
@Override
public void onPrefeed(final LifecycleEventVO event) {
    // do something
}
```

For further information on life-cycle events please visit Section 22.3, “Session life-cycle events”.

4.3.1.4. State based Strategy

Often a strategy has several states that it runs through during execution (e.g. FLAT, PENDING_LONG, PENDING_SHORT, LONG, SHORT, etc.). For these situations it is advisable to use a Java Enum, e.g.:

```java
package ch.algotrader.strategy;

public enum State {
    FLAT, PENDING_LONG, PENDING_SHORT, LONG, SHORT;
}
```

In case a strategy trades multiple instruments and each instrument has its own state it is suggested to create a Metrics object for each instrument. The Metrics object is a simple Java POJO that holds the state per instrument and potential other information regarding the instrument (e.g. values of technical indicators):

```java
public class Metrics implements Serializable {

    private static final long serialVersionUID = 5972079135237671512L;

    private long securityId;
    private State state;
    private double ema; // exponential moving average

    public Metrics(long securityId, State state, double ema) {
        this.securityId = securityId;
        this.state = state;
    }
}
```
4.3.1.5. Prevent an action from happening multiple times

Often a Java Action is triggered multiple times by a certain situation, because the underlying cause takes a finite amount of time to be resolved.

Example: The current market level exceeds a defined stop, which triggers a closing order. However during the time the order is being executed at the market, additional market data events are received. Because the position is not yet closed by that time, another undesired closing order might get placed.

To prevent actions from happening multiple times a state object mentioned in the previous section is again very helpful.

Whenever the predefined signal condition is met the state will be set to PENDING.

Below is an example implementation of a state based strategy.

```java
@Component
public class ABCService extends StrategyService {

    private final long[] securityIds = {1, 2, 3, 4, 5};  

    private Map<Long, Metrics> metricsMap = new HashMap<>();

    @Override
    public void onInit(final LifecycleEventVO event) {
        for (long securityId : securityIds) {
            metricsMap.put(securityId, new Metrics(securityId, State.FLAT));  
        }
    }

    @Override
    public void onTick(TickVO tick) {
        long securityId = tick.getSecurityId();
        Metrics metrics = metricsMap.get(securityId);

        if (metrics.getEma() > ... && State.FLAT == metrics.getState()) {
            metrics.setState(State.PENDING_LONG);  
            sendOrder(securityId);  
        }
    }
```
private void sendOrder(long securityId) {
    // create and send the order
}

@Override
public void onOrderStatus(OrderStatusVO orderStatus) {
    if (Status.EXECUTED == orderStatus.getStatus()) {
        Order order = getOrderService().getOrderByIntId(orderStatus.getIntId());
        Metrics metrics = metricsMap.get(order.getSecurity().getId());
        metrics.setState(State.LONG);
    }
}

1. List the securityIds the strategy wants to use
2. Store a Map of Metrics keyed by securityId
3. Create one Metrics object per securityId
4. Once a trigger was met (based on the trading logic) and the current State is FLAT set the State to PENDING
5. Send the order. The send order will not trigger again once the state has been set to PENDING_LONG.
6. Once the order got fully executed set the State to EXECUTED. As an alternative to using the onOrderStatus method a Section 8.4.8.2, “Trade callback” could be used.

4.3.1.6 Tagging of orders

Strategies that trade multiple securities at the same time often have the requirement to associate a particular order with a certain strategy. It is therefore often necessary to "tag" an order with additional meta data. This can be accomplished using order properties, see Section 16.2.3, “Order Properties”. In tagging orders using order properties it is also possible to distinguish automatically placed orders from manually placed orders (through AlgoTrader client or external broker GUI).

4.3.1.7 Using Base Strategy Names

Typically a strategy running corresponds to one entry in the database table strategy. In certain situations it may be necessary for a running strategy to place orders into separate entries in the database table strategy. Reasons for this might be:

- A strategy wants to have of multiple positions on the same instrument (i.e. long and short positions at the same time)
- A strategy needs to track positions for different accounts

For this purpose it is possible to use so called Base Strategy Names.
Example:

A strategy named EXAMPLE would like to interact with the two separate entries in the database named LONG and SHORT. For this purpose the strategy needs to be started with the strategyName set to EXAMPLE inside the file conf-cng.properties. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```
strategyName = EXAMPLE
```

In addition one needs to create the two entries EXAMPLE|LONG and EXAMPLE|SHORT in the database table strategy. The | separator causes strategy events like OrderStatusVO, FillVO and TransactionVO are propagated to the running strategy EXAMPLE.

Note

Using other characters than | will not work and will cause strategy events not to arrive in the EXAMPLE strategy.

4.3.1.8. Rolling of Futures and Options

Due to the expiring nature of Futures and Options, corresponding Positions have to be rolled prior to the Expiration Date. Typically, this would involve the following steps:

1. Close the Front-Month Position
2. Unsubscribe the Front-Month Future/Option
3. Subscribe the Back-Month Future/Option
4. On First Tick (see Section 8.4.8.1, “First tick callback”) open a new Position

When dealing with futures one has to decide on when to roll from the Front-Month future into the Back-Month future. For this different philosophies exist:

- Roll on a fixed day prior to the expiration date or the first notice date
- Roll when the Back-Month future starts having a higher traded volume than the Front-Month future
- Use the constant maturity method as described in the following section

Since Futures have an expiration date and are therefore not continuous, it is often not possible to base indicators on them. There are several method for dealing with this situation:

- use the raw data and ignore the fact that price time series will have jumps
- On the rollover day, add the difference between yesterdays closing price of the Back-Month future and yesterday's closing price of the Front-Month future to the combined time series. Alternatively one can use the difference between today's opening price of the Back-Month future and yesterday's closing price of the Front-Month future if only the time series for the generic 1st future is available. This method can be used for P&L calculation it might however lead to negative prices on long time series.
• Instead of using addition as mentioned in the previous item use multiplication. This method however cannot be used for P&L calculations. Depending on the indicator in use either addition or multiplication will be adequate.

• Use the constant maturity method as described in the following section

Please see Section 5.7, “Multi Security Simulations” for options on how to back test futures and options based strategies that require multiple securities to be subscribed and unsubscribed during the back test.

4.3.1.9. Synchronizing system clock

Many strategies require that the local system clock is in sync with the remote server clock. Unfortunately it is not possible to directly sync the local time with the remote server clock. However most servers are using NTP or some other time sync mechanism to make sure there local clock is in sync with the official time defined by NTP servers. As a result local system clock should also be synchronized with NTP servers. In most cases this can be done directly through the operating system (e.g. Windows Time Service). For Windows Servers there is also the time sync tool available which tends to be more precise than the Windows Time Service.

4.3.1.10. Prevent LazyInitializationException

Strategy code running runs outside of Hibernate Sessions. Traversal along the Object Tree beyond what is already loaded into the Hibernate session will throw a LazyInitializationException. All n-to-1 associations (e.g. Position.getStrategy) will be fetched eagerly so no LazyInitializationException will be throws.

The LazyInitializationException can still occur on rare occasions. To handle those situations there is a corresponding method to fully initialize the association inside each Entity association (e.g. Transaction.initializeSecurity(Initializer initializer) or Combination.initializeComponents(Initializer initializer)). The method takes a reference to an Initializer. In strategies the CacheManager can be passed as an Initializer and in server-side services the HibernateInitializer can be used. Subsequent calls to the same association will then get access to the already initialized Entity or Collection.

Traverse of an uninitialized relation from Position to Security in strategy code would look like this:

```java
Position position = ...;
position.initializeSecurity(getCacheManager()); // initialize security
position.getSecurity(); // security is now initialized
```

4.3.1.11. Comparing CSV files

This feature is deprecated and will be removed in the next release.

Sometimes one needs to compare output data files to verify a trading strategy or to find a problem during strategy development. CSV files can be imported into Microsoft Excel for the purpose of comparing them.

---

http://www.timesynctool.com/
Comparing two files however remains tricky and repeated exporting/importing and comparing in Excel can be cumbersome.

AlgoTrader provides a Java utility to compare ("diff") CSV files and assert the contents of the two files. For example the following statement compares a backtesting output file `actual.csv` with an expected result file `expected.csv`:

```
CsvDiff.diffAndLogAssertionErrors(fileDiffer,
       new File("expected.csv"), new File("actual.csv"));
```

The `fileDiffer` argument instructs the tool how to perform the diff operation. It can be constructed as follows:

```
FileDiffer fileDiffer = new FileDiffer(expectedDef, actualDef, differ);
```

In the above statement, `expectedDef` and `actualDef` define the columns of the two files; the `differ` gives exact instructions on how to perform the comparison of the files.

The columns of a CSV file are provided as implementations of `CsvColumn` which can be done with an enum:

```
public enum OrderReport implements CsvColumn {

    Date(new DateConverter("dd.MM.yyyy HH:mm:ss")),
    Instrument(SymbolConverter.INSTANCE),
    Side(StringConverter.INSTANCE),
    OrderType(StringConverter.INSTANCE),
    Size(LongConverter.INSTANCE),
    Limit(DoubleConverter.INSTANCE);

    private OrderReport(ValueConverter<?>> converter) {
        this.converter = converter;
    }

    private final ValueConverter<?>> converter;

    @Override
    public int index() {
        return ordinal();
    }

    @Override
    public ValueConverter<?>> converter() {
        return converter;
    }
}
```
An example CSV file definition is then created as follows:

```java
boolean hasHeaderLine = true;
CsvDefinition orderReportDef = new CsvDefinition(hasHeaderLine, OrderReport.values());
```

After defining the columns of both CSV files via `CsvDefinition` as indicated above we need to give instructions on how to perform the actual diff operation itself. In the simplest case we just compare the CSV files line by line and assert all or selected columns of the two files:

```java
SimpleDiffer differ = new SimpleDiffer.Builder()
   .assertEqual(OrderReport.Date, OrderReport.Date)
   .build();
```

The diff can also contain more complex instructions. For instance assume that we have exactly one BUY/SELL order per instrument and day but the ordering of instrument and side within a given date may be random. To align the correct rows for comparison, we must provide grouping and sorting hints for this case:

```java
SimpleDiffer simpleDiffer = new SimpleDiffer.Builder()
   .assertEqual(OrderReport.Date, OrderReport.Date)
   .build();

SortingDiffer sortingDiffer = new SortingDiffer.Builder()
   .build(simpleDiffer);

GroupDiffer differ = new GroupDiffer.Builder()
   .add(OrderReport.Date, OrderReport.Date)
   .build(sortingDiffer);
```

Now we are ready to run the `CsvDiff.diffAndLogAssertionErrors(..)` statement from above. The result may look similar to the following:

```
[..FAILED] 2 diffs at lines[exp/act]=45/51 in columns[exp/act]=[Size/Size] {exp-file=...}
[..FAILED] lines: exp=45, act=51
```
4.3.2. Esper based Strategies

When developing strategies using Esper in addition to Java code it is generally recommended to do Time-based Market Data Analysis and Signal Generation inside Esper Statements. Procedural actions like placing an order or subscribing to a Security are predominantly done inside Java Code.

4.3.2.1. Print Statement Selects

By means of the TestSubscriber, selected values of a Statement can be printed to the Console.

```java
@Name('TEST')
@Subscriber(className='ch.algotrader.esper.subscriber.TestSubscriber')
@SimulationOnly
select
    valueA,
    valueB,
    valueC
from
    TestEvent;
```

4.3.2.2. Logging values of an Indicator to a log file

Often strategies are based on a technical indicator. During simulation it is often desirable to log the values of such an indicator to a log file. This can be done with the following statement:

```java
@Name('INSERT_INTO_INDICATOR')
@Listeners(classNames={'ch.algotrader.esper.listener.IndicatorListener'})
select
    dateTime.toDate() as dateTime,
    valueA,
    valueB
```
valueC
from
Indicator;

Above statement will log dateTime, valueA, valueB and valueC to the file files/report/IndicatorReport.csv

4.3.2.3. Access to Esper Variables

Esper has a sophisticated variable management functionality. It is possible to access those variables from Java through the following methods:

```java
// set variable value
engine.setVariableValue("target", target);

// retrieve variable value
Double target = (Double) engine.getVariableValue("target");
```

For further details on Esper Variables please visit the Esper Documentation⁶

4.3.2.4. Esper Utility classes

Complex computations should be handled outside Esper. For this purpose, it is often easier to create a small Utility class and use its methods inside the statement. Example:

```java
package ch.algotrader.strategy;

public class MyUtil {

    public static double calculate(BigDecimal last) {

        return ...; // add calculation here
    }
}
```

This Utility class also needs to be declared in the file esper-...xml:

```xml
<auto-import import-name="ch.algotrader.strategy.MyUtil"/>
```

Now, the Utility class can be used in an Esper Statement like this to adjust a trailing stop loss:

⁶ http://esper.espertech.com/release-5.5.0/esper-reference/html/epl_clauses.html#variables_overview
select
tick.last,
value
from
TickVO as tick,
method:MyUtil.calculate(tick.last) as value

For further details on using static methods please visit the Esper documentation\(^7\)

4.3.2.5. Prioritizing Statements

If two statements are based on the same Event, it is necessary to set a priority for each statement to make sure, the system behaves deterministically:

```epl
@Name('STATEMENT_1')
@Priority(2)
select * from A;

@Name('STATEMENT_2')
@Priority(1)
select * from A;
```

For further details on statement priorities please visit the Esper documentation\(^8\)

4.3.2.6. Market Data Event Pre-feeding

When starting up a strategy in Live Trading Mode, it is often necessary to initialize technical indicators that have a look-back period. This initialization is done by feeding historical market data into the Esper Engine.

A typical pre-feed method will look like this:

```java
@Override
protected void onPrefeed(final LifecycleEventVO event) {
    switch (event.getOperationMode()) {
        case REAL_TIME:
            feedMarketData();
            break;
    }
}

public void feedMarketData() {
    ZonedDateTime startDateTime = ZonedDateTime.now().minusHours(1);

    // Feed historical market data here
}
```

\(^7\) [http://esper.espertech.com/release-5.5.0/esper-reference/html/epl_clauses.html#joining_method_syntax](http://esper.espertech.com/release-5.5.0/esper-reference/html/epl_clauses.html#joining_method_syntax)

Collection<TickVO> ticks = getHistoricalDataService().getTicksByMinDate(securityId, startDateTime, 1);
getEngine().initCoordination();
getEngine().coordinate(new CollectionInputAdapter(ticks, "dateTime"));
getEngine().startCoordination();

This method will load all ticks for the security (defined by securityId) for the last hour. It will then feed all of them sequentially to the local EsperEngine.

For further details on Esper Coordination please visit the Esper documentation\(^9\)

---

### 4.3.2.7. State based Strategy

Often a strategy has several states that it runs through during execution (e.g. FLAT, PENDING_LONG, PENDING_SHORT, LONG, SHORT, etc.). For these situations it is advisable to use a Java Enum, e.g.:

```java
package ch.algotrader.strategy;

public enum State {
    FLAT, PENDING_LONG, PENDING_SHORT, LONG, SHORT;
}
```

In addition a corresponding Esper Variable has to be configured:

```xml
<variable name="state" type="ch.algotrader.strategy.State"/>
```

If the variable needs to have an initial value, the variable has to be declared with a statement.

```java
@Name('CREATE_VAR_STATE')
create variable ch.algotrader.enumeration.State state = State.FLAT;
```

It is now possible to query current state inside Esper statements like this:

---

select * from ... where state = State.FLAT;

In case a strategy trades multiple instruments and each instrument has its own state an *Esper Named Window*\(^\text{10}\) can be used instead of an Esper Variable.

```java
@Name('METRICS_WINDOW')
create window
    MetricsWindow.std:lastevent()
    as
    Metrics;

@Name('INSERT_INTO_METRICS_WINDOW')
insert into
    MetricsWindow
select *
from
    Metrics;
```

The first statement creates the Named Window and the second statements inserts all Metrics events into the Named Window.

The Metrics object is a simple Java POJO that holds the state per instrument (and potential other information regarding the instrument):

```java
public class Metrics implements Serializable {  
    private static final long serialVersionUID = 5972079135237671512L;

    private long securityId;
    private State state;

    public Metrics(long securityId, State state) {
        this.securityId = securityId;
        this.state = state;
    }

    // getters and setters
}
```

To initialize the Named Window one Metrics event per instrument has to be sent into the Esper Engine upon startup of the strategy.

\(^{10}\)http://esper.espertech.com/release-5.5.0/esper-reference/html/nwtable.html
public void onInit(final LifecycleEventVO event) {
    getEngine().sendEvent(new Metrics(securityId, State.FLAT));
}

It is now possible to query current state inside Esper statements like this:

select state from MetricsWindow where securityId = ...;

### 4.3.2.8. Prevent an action from happening multiple times

Often a Java Action is triggered multiple times by a certain situation, because the underlying cause takes a finite amount of time to be resolved.

Example: The current market level exceeds a defined stop, which triggers a closing order. However during the time the order is being executed at the market, additional market data events are received. Because the position is not yet closed by that time, another undesired closing order might get placed.

To prevent actions from happening multiple times a state object mentioned in the previous section is again very helpful.

Whenever the predefined signal condition is met the state will be set to PENDING

```java
@Name('LONG_TRIGGER')
on
//trigger event (can also be the MetricsWindow itself)
update
MetricsWindow as metricsWindow
set
state = State.PENDING_LONG
where
// condition
and
    metricsWindow.state = State.SHORT or metricsWindow.state = State.FLAT

@Name('SEND_ORDER')
@Subscriber(className='xyzService#sendOrder')
select
    state
from
    MetricsWindow
where
    state = State.PENDING_LONG
or
```
Once the state has been changed to PENDING_LONG the statement `SEND_ORDER` will trigger an order to be sent. Since the `LONG_TRIGGER` statement only triggers if the state is either SHORT or FLAT it will not trigger again once the state has been set to PENDING_LONG.

Once the order has been fully executed (potentially using a Section 8.4.8.2, “Trade callback”) the state needs to be changed to LONG.

```java
getEngine().executeQuery("update MetricsWindow set state = State.LONG where securityId = " + securityId);
```

4.3.2.9. Creation of Bars based on Ticks

Often strategies rely on indicators that are based on OHLC Bars. Since Live Market Data (i.e. Ticks) are not delivered in the format of Bars, it is often necessary to create Bars from arriving Ticks. Section 17.1, “Creation of Bars based on Ticks” explains how to do this.

4.3.2.10. Reacting upon a newly subscribed security

Especially for Option and Future based strategies it is often not possible to subscribe to the entire Option or Futures Chain in advance. Therefore the actual Security the strategy is interested in, is evaluated and subscribed to at runtime. There are often steps that should take place immediately after the first market data event has arrived for such a security.

Section 8.4.8.1, “First tick callback” explains how to use a `FirstTickCallback` for this purpose

4.3.2.11. Reacting upon an order executions

A common use case is to wait for the full execution or cancellation of an order and then take some additional action.

Section 8.4.8.2, “Trade callback” explains how to use a `TradeCallback` for this purpose.

In this context it is also important to remember that when trying to close a position there might still be open orders associated with the corresponding security and strategy. It is suggested to cancel all corresponding orders, attach a `TradeCallback` to the cancellation and only close the position once all cancels have been confirmed.

Also keep in mind that an order might receive multiple fills in live trading. For example if one wants to send a Stop Order for each executed Order it is important to use the filled quantity and not on the original order quantity.

In handling partial fills the `TargetPositionOrder` can be useful, please see: Chapter 23, Execution Algos
4.3.2.12. Waiting on market data session upon strategy startup

Upon system startup strategies run through the life cycle phases as defined in Section 3.2, “Live Trading Mode”. At the same time market data connections are established. Due to the asynchronous nature of these two processes it is not predetermined which one will complete first. Typically within the START life cycle phase market data is subscribed. This however will fail if the market data adapter has not reached its SUBSCRIBED state by that time. To circumvent this issue the following Esper pattern can be used, which waits for both the first LifecyclePhaseVO and SessionEventVO to arrive before it fires:

```sql
select *
from pattern[LifecycleEventVO(phase=LifecyclePhase.'START')
    and SessionEventVO(state=ConnectionState.SUBSCRIBED)];
```

4.3.2.13. Execute an action once a day at a certain time

```sql
@Subscriber(className='OrderService#cancelAllOrders')
select null
from pattern [every {timer:at(0, 18, *, *, 1:5)}];
```

The above statement will cancel all open orders at 18:00:00 Mo-Fri.

4.3.2.14. Prevent Memory leaks

All trading strategies allocate a certain amount of memory to objects. If however those object allocations are never released the corresponding memory will not get freed which will lead to a memory leak. This is especially a concern for strategies that are kept running for an extended period of time.

In addition one has to be careful with Esper statements not to introduce memory leaks. For example the following statement is potentially dangerous since it just keeps all Tick Events it receives:

```sql
select * from TickVO.win:keepall();
```

4.4. Strategy life-cycle events

The system provides life-cycle events to strategies when switching to another phase in the strategy life-cycle.

In addition the AlgoTrader life cycle manager supports two modes of operation: REAL_TIME and SIMULATION. In both modes all strategies transition through the same life-cycle phases. Depending on the operation mode not all phases may be relevant.
Table 4.1. Strategy life-cycle phase

<table>
<thead>
<tr>
<th>Phase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>Called after deploying all modules of the Server Engine but before deploying the init modules of the Strategy Engines.</td>
</tr>
<tr>
<td>PREFEED</td>
<td>Called after deploying the init modules with Engine#deployInitModules() of Strategy Engines but before deploying their run modules and before feeding any market data events.</td>
</tr>
<tr>
<td>START</td>
<td>Called after deploying the run modules of all Engines. At this time Market data events start feeding into strategy engines.</td>
</tr>
<tr>
<td>EXIT</td>
<td>In SIMULATION mode this event occurs after finishing the simulation and before sending an EndOfSimulationVO event and before publishing simulation results. In REAL_TIME operation mode an EXIT life cycle event occurs when the virtual machine begins its shutdown.</td>
</tr>
</tbody>
</table>

Strategies can subscribe to these life-cycle events by overwriting the corresponding live-cycle method of the StrategyService:

```java
@Override
public void onInit(final LifecycleEventVO event) {
    ...
}
```

```java
@Override
public void onPrefeed(final LifecycleEventVO event) {
    ...
}
```

```java
@Override
public void onStart(final LifecycleEventVO event) {
    ...
}
```

```java
@Override
public void onExit(final LifecycleEventVO event) {
    ...
}
```

Additionally, the user has the availability to subscribe to all incoming Life cycle events at once by overriding the onLifecycleEvent method instead. This is especially useful in Distributed mode, when there are two types of life-cycle: STRATEGY and SERVER.

```java
@Override
public void onLifecycleEvent(final LifecycleEventVO event) {
    ...
}
```
Important

Overriding the `onLifecycleEvents` method directly is a non-standard approach and considered appropriate only in Distributed mode when the strategy should be aware of the Server JVM life-cycle as well. It requires the user to route different phase type events (INIT, PREFEED, START, EXIT) to appropriate methods and also differentiate those events by their type (SERVER vs STRATEGY). If this method is not overridden by the user, the SERVER events inside the Strategy JVM are ignored by default.

4.5. Strategy Groups

AlgoTrader provides extensive support for strategy groups. In simulation mode multiple strategies can be run simultaneously as part of a strategy group. It is even possible to run multiple instances of the same strategy (with different parameters).

For this purpose AlgoTrader provides support for strategy and engine templates as well as strategy groups based on Spring XML configuration and abstract Spring beans. This enables strategy developers to define abstract templates for strategy engines and strategy services and then define concrete instances of those strategies with custom configuration.

Definition of strategy templates and engine templates typically look like this..

```xml
<bean id="boxServiceTemplate" class="ch.algotrader.strategy.box.BoxService" abstract="true">
  <property name="lookupService" ref="lookupService"/>
  <property name="marketDataCacheService" ref="marketDataCacheService"/>
  <property name="portfolioService" ref="portfolioService"/>
  <property name="positionService" ref="positionService"/>
  <property name="measurementService" ref="measurementService"/>
  <property name="orderService" ref="orderService"/>
  <property name="subscriptionService" ref="subscriptionService"/>
  <property name="commonConfig" ref="commonConfig"/>
</bean>

<bean id="boxEngineTemplate" class="ch.algotrader.esper.EngineFactoryBean" abstract="true">
  <property name="configResource" value="esper-box.cfg.xml"/>
  <property name="initModules" value="box-init"/>
  <property name="runModules" value="current-values, box-run"/>
</bean>
```

Based on these templates concrete strategy instances can be configured.
<at:strategy name="box-narrow"
configClass="ch.algotrader.strategy.box.BoxConfig"
engineTemplate="boxEngineTemplate"
serviceTemplate="boxServiceTemplate"
resourceName="box-narrow.properties" />

<at:strategy name="box-wide"
configClass="ch.algotrader.strategy.box.BoxConfig"
engineTemplate="boxEngineTemplate"
serviceTemplate="boxServiceTemplate"
resourceName="box-wide.properties" />

Only parameters configClass, engineTemplate, and serviceTemplate attributes are mandatory. Configuration can be further simplified if configuration resources follow the convention <strategy-name>.properties.

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
https://www.algotrader.com/schema/spring/config
https://www.algotrader.com/schema/spring/config/algotrader.xsd">

<at:strategy name="box-narrow"
configClass="ch.algotrader.strategy.box.BoxConfig"
engineTemplate="boxEngineTemplate"
serviceTemplate="boxServiceTemplate" />

<at:strategy name="box-wide"
configClass="ch.algotrader.strategy.box.BoxConfig"
engineTemplate="boxEngineTemplate"
serviceTemplate="boxServiceTemplate" />

</beans>

Note

AlgoTrader strategy groups use custom XML namespace at. To enable this name space above xmlns has to be added to the document definition.
Multiple strategy instances can be grouped together and assigned individual weights in the group. Strategy groups can executed as one unit using provided Spring profile.

```xml
<beans profile="simpleGroup">
    <at:strategyGroup id="simpleGroup">
        <at:strategyItem name="box-narrow" weight="0.2"/>
        <at:strategyItem name="box-wide" weight="0.8"/>
    </at:strategyGroup>
</beans>
```

To start the `simpleGroup` activate the Spring profile through the following `Section 2.3, "VM Arguments"

-Dspring.profiles.active=...,simple

The AlgoTrader Eclipse IDE provides a visual editor for strategy groups, for further details please see `Section 10.2.3, "AlgoTrader Configuration Editor"`
Strategy Backtesting

For back testing historical data can be provided to strategies either via .csv files or via Section 18.1, “InfluxDB”: Securities specified within the table subscription or securities subscribed to via the SubscriptionService are fed to the Strategy.

To feed data from CSV files during a back test the following property needs to be set inside conf.properties. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```java
# should market data events be feed from CSV files
dataSource.feedCSV = true
```

For further details on file format and storage location of CSV files please see Section 18.9, “Market Data File Format”.

**Note**
When feeding historical data with CSV files it is not possible to set a particular time range for the simulation. If this is a requirement please feed data through InfluxDB.

To feed data from InfluxDB during a back test the following properties need to be set inside conf.properties. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```java
# should market data events be feed from the database
dataSource.feedDB = true

# the back test start date when feeding from InfluxDB
dataSource.feedMinDate = 2016-01-01

# the back test end date when feeding from InfluxDB
dataSource.feedMaxDate = 2016-12-31
```

The tables Subscription, Position, Combination & Component have a field persistent which has the following meaning:

- **persistent = true:** the corresponding entry will NOT be deleted before the start of a backtest
- **persistent = false:** the corresponding entry will be deleted before the start of a backtest

### 5.1. Exchange Simulator

The system provides an Exchange Simulator that is mainly used in back testing mode, but can also be used in live trading. The Exchange Simulator executes Orders by using an ExecutionModel. An Execution Model
contains the logic which decides whether an order gets executed under the current market situation and what portion of the order gets executed. In addition the ExecutionModel also contains the logic to calculate commissions and fees that should be added to an order.

AlgoTrader contains a DefaultExecutionModel which provides a reasonable default logic for executing orders. The DefaultExecutionModel provides the following properties inside the file conf.properties where they can be changed. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```java
# percent slippage that will be added to an order
#{"type":"Double","label":"Percent Slippage"}
execution.slippagePct = 0.0

# execution commission per order
#{"type":"Double","label":"Commission Per Order"}
execution.commissionPerOrder = 0.0

# execution commission per contract
#{"type":"Double","label":"Commission Per Contract"}
execution.commissionPerContract = 0.0

# execution commission in percent of the order amount (i.e. quantity x price)
#{"type":"Double","label":"Commission In Percent"}
execution.commissionInPercent = 0.0
```

The DefaultExecutionModel always charges commissions in the transaction currency of the order. E.g. when buying BTC/USD the fees will be charged in USD.

For further details on the DefaultExecutionModel please consult the JavaDoc.

It is possible to replace the DefaultExecutionModel with a custom implementation that implements the interface ExecutionModel. The custom Execution Model needs to be registered as a Spring Bean in the following locations:

- For simulation: /META-INF/applicationContext-client-xxx.xml in the strategy project under src/main/resources.
- For live trading (to be used globally): /META-INF/applicationContext-env.xml. This file needs to be in the class path, e.g. in the conf project under src/main/resources.

During the simulation process transaction as well as position and cash_balance updates are executed in the database. It is therefore possible to use a standard database reporting tool to perform additional analysis on it, provided you are not running vs. the H2 in-memory database (Spring profile embeddedDataSource).

To use the Exchange Simulator the Spring profile simulation has to be used, e.g.

```
-Dspring.profiles.active=simulation...
```
5.2. Simulation Process

During a simulation process the following steps are executed sequentially by the SimulationExecutorImpl:

1. Create strategy entries in the database

2. The database is reset to its original state via the ResetService

3. An initial amount (USD 1,000,000 per default) is allocated to each strategy (the initial amount can be changed through the simulation.initialBalance setting inside conf.properties)

4. All server Esper modules are deployed

5. The life cycle phase INIT is broadcasted to all strategies. During this phase potential initiation steps can be invoked.

6. All strategy initModules Modules are deployed (if using Esper based strategies)

7. The life cycle phase PREFEED is broadcasted to all strategies. During this phase technical indicators can be initialized using historical data

8. All strategy runModules Modules are deployed (if using Esper based strategies)

9. Market data subscriptions are initialized based on entries in the table subscription

10. The life cycle phase START is broadcasted to all strategies. During this phase eventual actions like security subscriptions can be taken care of

11. At that time the actual simulation starts and market data events are starting to be sent into the Esper Engines

12. The life cycle phase EXIT is broadcasted to all strategies. During this phase eventual cleanup actions can be taken care of

13. At the end of each simulation run, metrics are printed to the console (if enabled), see Chapter 28, Metrics
14. All open orders are cancelled

15. All open positions are closed

16. An EndOfSimulationVO event is sent to all strategies

17. SimulationResults are retrieved from the strategies

18. Esper Engines are re-initialized

19. The In-Process Exchange Simulator is reset

20. The Market Data Cache is flushed

21. The second-level cache is cleared

22. All reports are closed

23. The Excel based back test report is created and statistics are displayed to the console, see Section 5.5, “Performance Statistics”

5.3. Single Run Simulation

To run a strategy in Simulation Mode with the currently defined parameters use the procedure defined in Section 3.1, “Simulation Mode”.

5.4. Automated Parameter Optimization

The system allows running multiple simulations in parallel. Using cloud based servers thousands of simulation runs can be carried out in a matter of a few hours. For additional information please visit the full blog post on cloud based trading strategy optimization using algotrader and Amazon Elastic MapReduce\(^1\).

Using Numerical Optimization functions (i.e. Brent & Newton) optimal parameter ranges can be determined in an automated fashion.

The following options exist (set in program arguments):

**simulateBySingleParam**

One Simulation run with a parameter set to the defined value. The example below will do one run with parameter a set to 0.8

```
simulateBySingleParam a:0.8
```

**simulateByMultiParam**

One Simulation run with multiple parameters set to defined values. The example below will do one run with parameter a set to 0.8 and b set to 12.0

---

simulateByMultiParam a:0.8,b:12.0

**optimizeSingleParamLinear**

Multiple Simulation runs by incrementing the value of one parameter within a defined interval. The example below will increment the value of parameter a starting at 0.1 to 0.9, incrementing by 0.1 for each run.

optimizeSingleParamLinear a:0.1:0.9:0.1

**optimizeSingleParamByValues**

Multiple Simulation runs by iterating the value of one parameter according to defined list. The example below will iterate the value of parameter a through the following list: 0.2, 0.8, 0.9 and 1.2

optimizeSingleParamByValues a:0.2:0.8:0.9:1.2

**optimizeSingleParam**

Multiple Simulation runs by setting the value of one parameter within the defined range and trying to find the maximum Sharpe Ratio. The optimizer being used is `UnivariateRealOptimizer`. The example below will set the value of parameter a between 0.1 and 1.0 (accuracy 0.01).

optimizeSingleParam a:0.1:1.0:0.01

**optimizeMultiParamLinear**

Multiple Simulation runs by doing a matrix Optimization of 2 or 3 parameters by incrementing their values within a defined intervals. The example below will iterate through all possible combinations by incrementing the value of parameter a starting at 0.1 to 0.9 (increment: 0.1), and incrementing the value of parameter b starting at 10.0 to 100.0 (increment: 5.0)

optimizeMultiParamLinear a:0.1:0.9:0.1 b:10.0:100.0:5.0

**optimizeMultiParam**

Multiple Simulation runs by adjusting the value of multiple parameters around their start values and trying to find the maximum Sharpe Ratio. The example below will start the optimization by setting the value of parameter a to 85.0 and parameter b to 150.0

optimizeMultiParam a:85.0 b:150.0

In order to process parameters with the correct decimal scale the following property needs to be updated inside `conf.properties`. Alternatively the property can be changed via *Section 2.3, “VM Arguments”*:

```
# the number of digits all portfolio balances will be displayed with
```
Note

In order for the parameter optimization to work the following properties need to be updated inside `conf.properties`. Alternatively the properties can be changed via `Section 2.3, “VM Arguments”`:

```
# if set to true, writing to CSV reports will be disabled
report.disabled = true

# if set to true, the Excel back test report will open at the end of a simulation
report.openBackTestReport = true
```

Note

Before each back test run the Esper Engines will be reset. However Strategy services are not reset. Due to this any state that is maintained within the Strategy service needs to be reset within the `onInit` method.

Note

The values of Esper variables as well as Java properties get initialized on startup using Spring. The actual optimization only happens once the Spring context is fully initialized. Due to this it is necessary to overwrite the default values in the `onInit` from `system.properties`. This can be done as follows for Esper variables:

```
getEngine().setVariableValue("propertyA", System.getProperty("propertyA"));
```

And like this for Java properties

```
this.propertyB = System.getProperty("propertyB");
```

5.5. Performance Statistics

At the end of each single simulation run, a CSV and Excel based back test report with performance statistics is created.
Figure 5.1. Back Test Report
The following 4 files are created in the sub-folder /files/report:

- **BackTestReport.xlsx**: the Excel based back test report (see image above)
- **MetricReport.csv**: contains key performance metrics
- **PortfolioReport.csv**: contains daily portfolio values (i.e. netLiqValue, marketValue, realizedPL, unrealizedPL, cashBalance, openPositions & leverage)
- **TradeReport.csv**: contains all trades including their profit

The Excel based back test report can be modified in terms of formatting and layout if needed.

In addition when running a single simulation run, statistics will be displayed to the console in the following format:

```
execution time (min): 2.43
dataSet: eurusd-1min-20111218-20130121
netLiqValue=1'229'714.00

month-year:          Dec-11  Jan-12  Feb-12  Mar-12  Apr-12  May-12  Jun-12  Jul-12  Aug-12  Sep-12  Oct-12  Nov-12  ...
monthlyPerformance:   0.58%   4.03%   2.66%  -0.19%  2.80%  -1.96%  2.44%  3.23%  0.66%  -0.58%  3.67%  2.31%  ...
year:                  2011    2012    2012
yearlyPerformance:    0.58%  22.33%  -0.06%
posMonths=10 negMonths=4 posYears=2 negYears=1
avgM=1.50% stdM=1.79% avgY=19.39% stdY=6.21% sharpeRatio=3.12
maxMonthlyDrawDown=1.96% bestMonthlyPerformance=4.03% maxDrawDown=4.49%
maxDrawDownPeriod=46.00days colmarRatio=4.32
WinningTrades: count=428(53.97%) totalProfit=1'277'201.37 avgProfit=2'984.12
avgProfitPct=0.23%
LoosingTrades: count=365(46.03%) totalProfit=-1'047'487.34 avgProfit=-2'869.83
avgProfitPct=-0.26%
AllTrades: count=793(100.00%) totalProfit=229'714.04 avgProfit=289.68
avgProfitPct=0.00%
```

When running parameter optimizations, statistics will be displayed in the following summary format showing the current parameter values as well as corresponding performance statistics of one run on one single line:

```
a=90 avgY=39.86% stdY=20.16% sharpe=1.97 maxDDM=11.29% bestMP=8.35% ...
a=105 avgY=34.60% stdY=20.33% sharpe=1.69 maxDDM=11.56% bestMP=8.39% ...
```

In addition to above General Performance statistics, strategy specific performance statistics are printed to the console. These are retrieved by calling the method `StrategyService.getSimulationResults` of the strategy.

The amount of output during the simulation can be adjusted by setting the Log Level according to *Chapter 29, Logging*. 
5.6. In-Process Exchange Simulator (deprecated)

Important

The In-Process Exchange Simulator has been deprecated and usage is discouraged - it is a subject for removal and replacement in future release. You can still use Simulation mode as before, but your Strategy code should place orders through OrderService, not the Simulator - meaning, the Strategy code should be the same for both production and back-testing.

Executing all transactions in the database during simulation is useful for reporting purposes but also incurs additional processing time. For trivial strategies that do not need to perform any sort of sophisticated querying based on transaction data, an additional in-process / in-memory exchange simulator is available that uses Hash Maps as the underlying storage mechanism. This will allow for significantly faster processing of orders during simulation. The in-process exchange simulator can be used as follows within strategies:

```java
Order order = new MarketOrder.Factory.newInstance();
order.setSecurity(security);
order.setStrategy(strategy);
order.setQuantity(qty);
order.setSide(Side.BUY);

getSimulator().sendOrder(order);

Position position = getSimulator().findPositionByStrategyAndSecurity(strategy, security);
```

Important

As the simulator is not integrated with the database any lookup regarding Order, OrderStatus, Transaction, Position & CashBalance through the LookupService and PortfolioService will return nothing. Also Execution Algos are currently not supported with the In-Process Exchange Simulator.

5.7. Multi Security Simulations

By default, only those securities will be considered for simulations which have been subscribed to in the INIT or PREFEED phase.

Some strategies that are based on multiple securities need to subscribe and unsubscribe securities during the simulation. A typical example for this would be a Futures bases strategy that needs to unsubscribe an expiring Future and at the same time subscribe to the next Future in the chain. To be able to subscribe and unsubscribe securities during a simulation the following two options exist:
• Changing the following property inside the file `conf.properties` will cause all CSV files present in the dataset directory to be used for the simulation. (However, strategies still only receive market data for securities they have subscribed to). Alternatively the properties can be changed via *Section 2.3, “VM Arguments”*: 

```
# should all files in the dataSetLocation be used or just the ones corresponding to defined subscriptions
dataSource.feedAllMarketDataFiles = true
```

• subscribe to all securities required in the `INIT` or `PREEFEED` phase, and then unsubscribes securities in the `START` phase, that are only need later in the simulation.
Architecture

The architecture of AlgoTrader is composed of the following components.

Figure 6.1. Architecture

The AlgoTrader Server provides the infrastructure for all strategies running on top of it. The AlgoTrader Server holds the main Esper Complex Event Processing (CEP) engine. It is responsible for all domain model objects and their persistence in the database. Different market data adapters are available to process live and historical market data. On the other end adapters for different execution brokers and exchanges are available, which are responsible for placing orders and receiving executions.

The AlgoTrader Server also provides business components for back testing, parameter optimization, analysis, execution management, risk management, reporting and hedging.

On top of the AlgoTrader Server any number of strategies can be deployed. Strategies can either be coded purely in Java or in a combination of Java and Esper code. Esper based strategies make use of a dedicated Esper CEP engine. A strategy can deploy any number of SQL-like Esper statements for time-based market data analysis and signals generation. Esper statements can invoke any number of procedural actions, such as placing an order or closing a position, which are coded in Java. The combination of Esper statements and Java Code provides a best-of-both-worlds approach.

For management and monitoring of the system different GUI clients exist. The AlgoTrader UI provides trading related functionality like charting, orders, positions & market data. Eclipse or IntelliJ IDE’s are used for strategy development.

For productive installations and deployment AlgoTrader uses Docker.
Chapter 7.

Domain Model

The following sections describe the Domain Model of the system using UML (unified modeling language).

7.1. Entities

Figure 7.1. Entities Overview

The Main Entities of the system are specified within the following table:

Table 7.1. Entities

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy</td>
<td>Each object of this class represents a running strategy within the system</td>
</tr>
<tr>
<td>Security</td>
<td>This is the base class of all securities in the system</td>
</tr>
<tr>
<td>SecurityFamily</td>
<td>A group of Securities (e.g. all S&amp;P 500 Futures)</td>
</tr>
<tr>
<td>Subscription</td>
<td>Market Data Subscriptions of a Strategy for a particular Security are represented by this class. For every Subscription the Strategy will receive Live Market Data for the corresponding Security</td>
</tr>
<tr>
<td>MarketDataEvent</td>
<td>Represents any type of market data related to a particular Security</td>
</tr>
<tr>
<td>Order</td>
<td>An Order for a particular Security</td>
</tr>
<tr>
<td>Account</td>
<td>An account held with an external Broker or Exchange</td>
</tr>
</tbody>
</table>
A full list of all Entities of the system will be discussed throughout the next sections. Entities of the system can be categorized into the following three areas:

**Reference Data**
- Represent static referential data like:

**Market Data**
- Represent external events (Tick and Bar) coming from market data providers or internal events (Generic Events) coming from another trading strategy. Market Data is typically immutable and of only momentary interest to the trading strategies. Market Data Events are available as Value Objects only (but not as Entities):
  - MarketDataEventVO and its subclasses TickVO, BarVO, QuoteVO, BidVO, AskVO, TradeVO and GenericTickVO as well as any type of GenericEventVO

**Transaction Data**
- Represent the financial state of trading strategies. Some of them (e.g. Transactions and Measurements) are immutable whereas others (e.g. Positions and Balances) are mutable and change their values while Orders are getting executed:
  - Order, Transaction, Position, CashBalance, Measurement, PortfolioValue and related Entities

Besides providing Getters and Setters all Entities provide the following common features:

**VO Converter**
- The static inner Converter class can be used to automatically convert the Entity to its corresponding Value Object, see Section 7.3, “Value Object”

**Factory**
- The static inner Factory class can be used to create new instances of an Entity
7.1.1. Strategy

The strategy entity represents an individual strategy running inside AlgoTrader.

Regarding the question "what is a productive strategy?". It essentially up to the user, what he would like to consider as one strategy. A strategy can have one or multiple instruments. And also regarding trading logic there is no limitation.

However please note that the entire performance and reporting functionality of AlgoTrader happens on the strategy level. So if one would like to see performance metrics on an instrument level one would have to instantiate multiple strategies. Also, if it is a requirement to start and stop individual functions separately, it is best to put them into two separate strategies.

On the technical side each separate strategy allocates a certain amount of overhead (memory and CPU). For that reason it is best to combine functionality into as few strategies as possible if there are no good reasons not to separate them.

The field autoActivate means that if a strategy is set to active corresponding market data subscriptions are initiated automatically upon startup of the system. This is useful in distributed mode when strategies and the server run in different processes. If you restart the server in this scenario, subscriptions for the strategies are automatically loaded again (without having to restart the strategies).
There are several classes that are directly related to the strategy

**Table 7.2. Strategy Classes**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PortfolioValue</td>
<td>On regular time intervals certain portfolio values (e.g. NetLiqValue, CashBalance, etc.) are saved to the database for every strategy.</td>
</tr>
<tr>
<td>Measurement</td>
<td>Custom Measurements (e.g. current value of a custom indicator) related to a strategy can be saved using this class</td>
</tr>
<tr>
<td>CashBalance</td>
<td>A CashBalance represents the current cash amount of a particular strategy in a particular currency</td>
</tr>
</tbody>
</table>

**Table 7.3. Portfolio Value Details**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cashBalance</td>
<td>Market value of all open forex positions + cash amount available to the strategy</td>
</tr>
<tr>
<td>marketValue</td>
<td>Market value of all open (non-forex) positions</td>
</tr>
<tr>
<td>netLiqValue</td>
<td>Cash balance + market value</td>
</tr>
<tr>
<td>realizedPL</td>
<td>Realized P&amp;L of all positions</td>
</tr>
<tr>
<td>unrealizedPL</td>
<td>Unrealized P&amp;L of all positions</td>
</tr>
</tbody>
</table>

All valuations (strategy and position level) can be queried via the PortfolioService. Fees are considered in the calculations if properly configured.
7.1.2. Security

The above UML Class diagram shows all available Security classes

Table 7.4. Security Types

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
<td>A tradeable Option</td>
</tr>
<tr>
<td>Future</td>
<td>A tradeable Future</td>
</tr>
<tr>
<td>Forex</td>
<td>A Foreign Exchange Currency (FX) or Crypto Currency</td>
</tr>
<tr>
<td>Stock</td>
<td>A Single Stock</td>
</tr>
<tr>
<td>Fund</td>
<td>An ETF, Mutual Fund, etc.</td>
</tr>
<tr>
<td>Index</td>
<td>An Index (e.g. Equity, Volatility, Commodity)</td>
</tr>
<tr>
<td>GenericFuture</td>
<td>A virtual Future with a fixed duration</td>
</tr>
<tr>
<td>Entity</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>InterestRate</td>
<td>Any type of Interest Rate</td>
</tr>
<tr>
<td>Bond</td>
<td>A corporate or government Bond</td>
</tr>
<tr>
<td>Commodity</td>
<td>A physical Commodity (e.g. Energy, Metals, Agriculture or Livestock). For Commodity Futures use Future.</td>
</tr>
<tr>
<td>PerpetualSwap</td>
<td>A perpetual swap contract supported by various crypto derivatives exchanges (e.g. BitMEX)</td>
</tr>
<tr>
<td>Combination</td>
<td>A synthetic security composed of one or many Components (see Chapter 24, Synthetic Securities and Derivative Spreads)</td>
</tr>
<tr>
<td>SecurityReference</td>
<td>A generic link between one security and another</td>
</tr>
</tbody>
</table>

The Security class provides the following two important methods:

- `getValue` which calculates the current (market) value of the instrument based on a quantity and a price parameter

- `getPrice` which calculates the current price of the instrument based on a (market) value and a quantity parameter

For most instruments the formula is:

- \( \text{value} = \text{quantity} \times \text{contractSize} \times \text{price} \)

- \( \text{price} = \text{value} / \text{quantity} / \text{contractSize} \)

However for PerpetualSwaps the formulas are different:

- \( \text{value} = \text{quantity} \times \text{contractSize} / \text{price} \)

- \( \text{price} = \text{quantity} \times \text{contractSize} / \text{value} \)

A Security Family contains common information about an entire family of securities (i.e. all general information about options on S&P500 are stored using this class). The class provides fields like MIN_QTY, MAX_QTY and MIN_PRICE. This information is used by ReferenceDataService when downloading new future and option changes, values from Security Family will then be copied onto the newly created Futures and/or Options. In regular operation mode (i.e. simulation of live trading) the information from Security Families are not used but only the information contained within Securities.

The class `EasyToBorrow` contains information about how many contracts of a particular Stock can be shorted through a specific Broker.

`SecurityReference` is a generic link between one security the owner and another the target. Using this class it is possible for a Security to have links to multiple other Securities.

### 7.1.3. Market Data Events

Market Data Events are available as Value Objects only but not as Entities. There are three different kinds of Market Data Events:
Table 7.5. Market Data Types

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarVO</td>
<td>Open-High-Low-Close Price Bars, also containing volumes and volume weighted average prices</td>
</tr>
<tr>
<td>TickVO</td>
<td>Snapshot of the market at a particular point in time, containing information like last price, last time, bid, ask, volume, etc.</td>
</tr>
<tr>
<td>QuoteVO</td>
<td>Its subclasses represent the current best bid and offer BidVO and AskVO</td>
</tr>
<tr>
<td>TradeVO</td>
<td>An actual order that was executed on the market, containing information like last price, last size and volume</td>
</tr>
<tr>
<td>GenericTickVO</td>
<td>Represents additional price information made available by market data providers (e.g. open price, close price, vwap price)</td>
</tr>
</tbody>
</table>

For simulation purposes Bars and Ticks can be supplied through CSV files (see Section 18.9, “Market Data File Format”) or through InfluxDB (see Chapter 18, Historical Data). In live trading Trades, Bids and Asks are received by the broker / exchange specific MarketDataService.

For conversion between Ticks and Bars please see Section 17.1, “Creation of Bars based on Ticks”.

7.1.4. Order

![Order UML Class Diagram](image)

Figure 7.4. Orders

The following UML Class diagram shows the Order and its related subclasses.
Table 7.6. Order Classes

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
<td>Base Class for all Order Types</td>
</tr>
<tr>
<td>OrderStatus</td>
<td>Order Status changes received back from the Broker / Exchange (e.g. PARTIALLY_EXECUTED or CANCELLED) are represented by this class</td>
</tr>
<tr>
<td>OrderCompletion</td>
<td>Similar to Order Status but only gets created once an order is fully executed or cancelled and all corresponding database activity has been completed.</td>
</tr>
<tr>
<td>OrderProperty</td>
<td>An arbitrary property that can be attached to an Order. Through the type attribute the OrderProperty can be marked as internal only or as fix property or as IB property.</td>
</tr>
<tr>
<td>Fill</td>
<td>Filled orders are represented by this Class</td>
</tr>
<tr>
<td>Transaction</td>
<td>Each Fill is recorded as a transaction in the database using this entity. In addition the table transaction also carries transactions like INTREST, DEBIT, CREDIT &amp; FEES</td>
</tr>
<tr>
<td>SimpleOrder</td>
<td>An Order that can be sent directly to the market</td>
</tr>
<tr>
<td>MarketOrder</td>
<td>Predefined SimpleOrder types</td>
</tr>
<tr>
<td>LimitOrder</td>
<td></td>
</tr>
<tr>
<td>StopOrder</td>
<td></td>
</tr>
<tr>
<td>StopLimitOrder</td>
<td></td>
</tr>
<tr>
<td>AlgoOrder</td>
<td>A composite order that will generate multiple SimpleOrders. An AlgoOrder cannot be sent directly to the market. AlgoOrders are also called &quot;Execution Algos&quot;, see Chapter 23, Execution Algos</td>
</tr>
<tr>
<td>TWAPOrder</td>
<td>This algorithm aims to match the Time-Weighted Average Price</td>
</tr>
<tr>
<td>VWAPOrder</td>
<td>This algorithm aims to match the Volume-Weighted Average Price</td>
</tr>
<tr>
<td>TargetPositionOrder</td>
<td>This algorithm automatically manages orders to reach the specified target quantity.</td>
</tr>
<tr>
<td>TrailingLimitOrder</td>
<td>This algorithm submits an order directly to the broker/exchange, with a limit price set a fixed distance from the current market price.</td>
</tr>
<tr>
<td>SlicingOrder</td>
<td>An AlgoOrder, that will split a large order into multiple child orders. The size of the child order, time in the market and delay between orders are randomized within the specified range.</td>
</tr>
</tbody>
</table>

**Note**

AlgoOrders and Order parent/child associations are persisted to the database. After a system restart, pending AlgoOrder will be visible but it will not continue execution automatically - it will not create new child orders. Execution reports for existing child orders will be processed.
7.1.5. Account

An Account represents either an actual account, an account group (IB specific) or an allocation profile (IB specific). An account is assigned to a particular adapterType (e.g. IB_NATIVE or FXCM_FIX) which identifies the OrderService to use for this account. In addition the field sessionQualifier which is needed to define the actual session in place (for FIX Connections). With this setup, it is possible to have multiple Sessions (session qualifiers) per AdapterType and to have multiple Accounts per Session. If the field active is set to true a potential corresponding Fix session will be activated.

Optionally an accountServiceType (e.g. IB_NATIVE or BFX) can be added which identifies the AccountService to use for this account.

Accounts have an optional dependency to Exchange for cases when an account can only be used to trade on one single Exchange (typical for Crypto Currency Exchanges).

Note

Orders sent to the market will always contain Account related information in an adequate way (e.g. as a FIX Tag 1). Also Transactions which are based on an actual order will have an association with a particular Account. However Positions do not hold any information regarding Accounts. It is thus possible that a Position holds aggregated Quantities from several external Accounts. Also it is possible to open a position through an account but then close it through another (i.e. when using separate execution and clearing brokers). With this setup Strategies do not have to worry about the actual Accounts the funds are located in. This way, a strategy will always only see one Position per Security.
7.1.6. Transaction

Each Fill is recorded as a transaction in the database using this entity. In addition the table transaction also stores transactions like INTREST, DEBIT, CREDIT & FEES. A transaction is immutable and contains all relevant information like dateTime, quantity, price, executionCommissions, clearingCommissions and fees as well as references to Account, Strategy, Security and Position.

Depending on the type of transaction the field quantity has the following values:

- **BUY**: > 0
- **SELL**: < 0
- **EXPIRATION**: any value
- **TRANSFER**: any value
- **CREDIT**: 1
- **EXCHANGE_CREDIT**: 1
- **INTREST_RECEIVED**: 1
• REFUND: 1
• DIVIDEND: 1
• DEBIT: -1
• EXCHANGE_DEBIT: -1
• INTREST_PAID: -1
• FEES: -1 (+1 for maker FEES paid)

As the sign of the transaction is defined by the quantity the fields executionCommissions, clearingCommissions and fees will always be positive for fees/commissions charged (they will be negative for fees/commissions paid, e.g. maker rebates).

Some crypto exchanges provide fee information. If the fees are in the currency of the transaction, they are stored in the fee attribute of the transaction. In case the fees are charged in another currency (for example Binance charges in its own currency - BNB), a new transaction is created with transactionType = FEES, quantity = -1 for fees paid (and quantity = -1 for fees received, e.g. maker rebates), price = fee value and currency = fee currency.

7.1.7. Position

For any Strategy holding a particular Security a Position is created in the database. Even if this position is later on closed (i.e. quantity = 0) the position will still remain in the database, because the associated Transactions still have references to it.

In general, position values (e.g. marketPrice, marketValue, averagePrice, cost, unrealizedPL & realizedPL) are calculated per actual strategy related position and show the price that would need to payed if the position was closed at this moment.
Table 7.7. Position Valuation Details

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>realizedPL</td>
<td>Total profit of closed parts of a position (parts of a position might still be open)</td>
</tr>
<tr>
<td>unrealizedPL</td>
<td>Profit of the currently open part of a position</td>
</tr>
<tr>
<td>cost</td>
<td>Total cost incurred to open the current position (potentially through multiple orders). These values are based on the fee configurations</td>
</tr>
</tbody>
</table>

All valuations (strategy and position level) are available through the Section 7.2.12, “Portfolio Service”.

7.1.8. Cash Balance

A CashBalance represents the current cash amount of a particular strategy in a particular currency.

Warning

Cash Balances are derived by taking all Transactions of the given Security and Strategy into account. It is therefore important not to modify Cash Balance entries directly in the database. In case transactions are added or modified manually to the database, please the management action reset position and cash balances in the Figure 10.3, “AlgoTrader UI Management”.

7.1.9. Subscription

Market Data Subscriptions of a Strategy for particular Securities are represented by this class. For every Subscription the Strategy will receive Live Market Data for the corresponding Security.

Figure 7.8. Subscription
Figure 7.9. Exchange

Exchanges around the world have different trading hours. Quite often there are different trading hours even for different securities trading on the same exchange. In addition each exchange typically has different holidays or days where trading starts late or trading stops early. Especially for futures trading there are often small gaps between different trading periods of the same trading date. FX trading is often available 24 hours a day without any gaps.

All of these scenarios are captured and maintained through the Entities Exchange, TradingHours and Holiday:

Table 7.8. Exchange

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exchange</td>
<td>Represents an individual Security, a group of Securities or an entire Exchange (if all Securities have the same trading hours). An Exchange has a name, a code (typically MIC) as well as a timezone.</td>
</tr>
<tr>
<td>TradingHours</td>
<td>Defines an individual trading period (e.g. 09:00am to 16:30pm). In addition TradingHours identify the weekdays they are valid for.</td>
</tr>
<tr>
<td>Holiday</td>
<td>Identifies a holiday of a specific exchange. In addition a Holiday can identify a late opening or early closing of trading on a particular trading day.</td>
</tr>
</tbody>
</table>
7.1.11. Order Preference

```plaintext
<<Entity>>
Account
+name : String(algoTrader_property_use_in_equals)
+active : boolean(nonunique)
+adaptType : String(nonunique)
+accountServiceType : String[0..1](nonunique)
+sessionQualifier : String[0..1](nonunique)
+extAccount : String[0..1]
+extAccountGroup : String[0..1]
+extAllocationProfile : String[0..1]
+extClearingAccount : String[0..1]
+cnrgid : Long[0..1]
+defaultAccount : 0..1
+"*
```

```plaintext
<<Entity>>
OrderPreference
+name : String(algoTrader_property_use_in_equals)
+orderType : OrderType(nonunique)
```

Figure 7.10. Order Preference

Table 7.9. Order Preference

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OrderPreference</td>
<td>This class allows definition of order default values (e.g. account, order type, delays, etc.). Except for the order type, all values have to be defined through Properties.</td>
</tr>
</tbody>
</table>

for further details see Section 16.2.1, "Order Preferences"

7.2. Services

The system is based on a Service Oriented Architecture (SOA). All operations of the system are provided as Spring Services / Beans. The following groups of services exist:

1. Main Services, are available to both the AlgoTrader Server and Strategies
2. Client Services, which will be instantiated by each Strategy (and the AlgoTrader Server itself)
3. Private Services, which are only used by the AlgoTrader Server

For a full list of all Services please visit our JavaDoc¹

¹ http://doc.algotrader.ch/javadoc/ch/algotrader/service/package-frame.html
Inside strategies all service are injected by the Spring Framework and can be accessed as follows within the strategy service:

```java
// subscribe for live market data
getSubscriptionService().subscribeMarketDataEvent(strategyName, securityId, adapterType);

// lookup a instrument by symbol
getLookupService().getSecurityBySymbol(symbol);

// send an order to the broker or exchange
getOrderService().sendOrder(order);
```

### 7.2.1. Main Services

**Table 7.10. Main Services**

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccountService</td>
<td>Responsible for retrieval of account balances and initiation of withdrawals</td>
</tr>
<tr>
<td>CalendarService</td>
<td>Responsible for information about Exchange trading hours and holidays</td>
</tr>
<tr>
<td>CombinationService</td>
<td>Responsible for handling all Combination / Component related DB-Operations.</td>
</tr>
<tr>
<td>FutureService</td>
<td>Responsible for all future specific operations</td>
</tr>
<tr>
<td>HistoricalDataService</td>
<td>Responsible for the retrieval of historical data from Historical Data Providers</td>
</tr>
<tr>
<td>MarketDataService</td>
<td>Responsible for the retrieval of market data as well as Subscription Management.</td>
</tr>
<tr>
<td>MeasurementService</td>
<td>Responsible for persistence and retrieval of Measurements related to Strategy</td>
</tr>
<tr>
<td>OptionService</td>
<td>Responsible for all option specific operations</td>
</tr>
<tr>
<td>OrderService</td>
<td>Responsible for sending orders to the Broker / Exchange</td>
</tr>
<tr>
<td>PortfolioService</td>
<td>Responsible for providing portfolio values</td>
</tr>
<tr>
<td>PositionService</td>
<td>Responsible for management of positions, e.g. close position and reduce position</td>
</tr>
<tr>
<td>PropertyService</td>
<td>Responsible for persistence of Properties related to a PropertyHolder</td>
</tr>
<tr>
<td>ReferenceDataService</td>
<td>Responsible for the download of option and future chains</td>
</tr>
</tbody>
</table>
7.2.2. Client Services

**Table 7.11. Client Services**

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MarketDataCacheService</td>
<td>Provides a strategy local cache of market data and FX conversion rates</td>
</tr>
<tr>
<td>LookupService</td>
<td>Provides general data lookup operations to other services</td>
</tr>
<tr>
<td>ConfigAwareStrategyService</td>
<td>Base class for all strategy services which has references to all necessary</td>
</tr>
<tr>
<td></td>
<td>services and implements all event listener interfaces. In addition the</td>
</tr>
<tr>
<td></td>
<td>service receives a reference to the strategy config</td>
</tr>
<tr>
<td>StrategyService</td>
<td>Base class for all strategy services which has references to all necessary</td>
</tr>
<tr>
<td></td>
<td>services and implements all event listener interfaces</td>
</tr>
<tr>
<td>SubscriptionService</td>
<td>This service is used by the strategy for subscription management. The actual</td>
</tr>
<tr>
<td></td>
<td>DB related operations are carried out by the MarketDataService. The</td>
</tr>
<tr>
<td></td>
<td>MarketDataService should not be called directly by strategies.</td>
</tr>
</tbody>
</table>

7.2.3. Account Service

The AccountService interface defines a method for retrieving account balances as well as the initiation of crypto withdrawals for crypto exchanges. For further details see Chapter 20, Account Data.

7.2.4. Calendar Service

The CalendarService is responsible for providing information about Exchange trading hours and holidays. Especially when trading multiple exchanges around the globe the CalendarService becomes very useful. It provides convenient methods like:

- isOpen (is the specified exchange open at the current time or at the specified date time). Will return true if no TradingHours are defined
- isTradingDay (is the current day or the specified day a trading day at the specified exchange)
- getOpenTime (gets the open time of the specified exchange on the current day or the specified day)
- getCloseTime (gets the close time of the specified exchange on the current day or the specified day)
- getNextOpenTime (gets the next open time of the specified exchange after the current date time or the specified date time)
- getNextCloseTime (gets the close open time of the specified exchange after the current date time or the specified date time)

In addition the Calendar service provides methods to identify a particular trading day, which will be important to associate a particular order for clearing. If a trading session overlaps from one day to another (e.g. starts on Sunday 23:00pm), the trading day will be considered the day when the session ends (e.g. Monday). However in this example Monday would need to be set to true in the corresponding TradingHours object.
7.2.5. Combination Service

AlgoTrader supports Synthetic Securities & Derivative Spreads. A Combination consists of one or many Components. For further details see Chapter 24, Synthetic Securities and Derivative Spreads.

7.2.6. Future Service

AlgoTrader has full support for Future based trading strategies. For further details see Chapter 14, Options & Futures

7.2.7. Historical Data Service

AlgoTrader provides several Historical Data Interfaces out-of-the-box. The system can store historical data in the integrated Section 18.1, “InfluxDB” and feed stored or recorded historical data to strategies during back tests. The system also integrates a feature for live data recording as well as live tick-to-bar aggregation. For further details please see Chapter 18, Historical Data

7.2.8. Market Data Service

AlgoTrader provides several Market Data Interfaces out-of-the-box. Live market data is available to trading strategies running within the system. For further details please see Chapter 17, Market Data

7.2.9. Measurement Service

The MeasurementService allows storage of arbitrary measurements in the database. Measurements contain a name, a time stamp and a value of type Integer, Double, Money (BigDecimal), Text (String) or Boolean. Only one value is allowed per measurement (i.e. one record cannot store both Integer and String values for example. In addition a Measurement also needs to have a reference to a strategy.

A Measurement can be created as follows whereas the time stamp will be set according to the current system time:
getMeasurementService().createMeasurement(strategyName, "myMeasurement", 12.12345);

In addition a Measurement can also be created by providing an explicit time stamp:

ZoneDateTime zonedDateTime = ZonedDateTime.of(2019, 4, 1, 0, 0, 0, 0, ZoneId.of("UTC"));
getMeasurementService().createMeasurement(strategyName, "myMeasurement", zonedDateTime, 12.12345);

A Measurement can be deleted by using the following method:

getMeasurementService().deleteMeasurement(measurementId);

To read Measurements from the database the Section 7.2.17, “Lookup Service” has to be used which provides various Measurement lookup methods, e.g. getMeasurementByMaxDate or getAllMeasurementsByMaxDate.

Note

Only one measurement entry per strategy and name is allowed for a given time stamp. If there is an existing measurement with matching strategy, name and time stamp then creating a new measurement with new value will overwrite the previous value.

7.2.10. Option Service

AlgoTrader has full support for Option based trading strategies including an Option pricing engine. For further details see Chapter 14, Options & Futures

7.2.11. Order Service

The OrderService is responsible for sending orders to brokers and exchanges in live trading as well as sending orders to the internal Section 5.1, “Exchange Simulator” during back tests. For further details please see Chapter 16, Order Management

7.2.12. Portfolio Service

Financial valuations (strategy and position level) are available through the PortfolioService.
Since some values (e.g. market value) depend on whether the position is long or short, aggregated position values of different strategies for the same security cannot be retrieved just by adding position values from the corresponding strategies. Example:

- Security: VIX Dec 2012
- Current Bid: 16.50
- Current Ask: 16.60
- Strategy A: quantity +10 -> market value: 10 * 1000 * 16.50 = 165'000
- Strategy B: quantity -10 -> market value: 10 * 1000 * 16.60 = -166'000

The sum of above market values would be -1'000 which is obviously wrong.

As a consequence the PortfolioDAO provides lookup-methods that aggregate positions from the same security (of different strategies) in the correct manner (e.g. findOpenPositionsAggregated).

### Warning

Positions are derived by taking all Transactions of the given Security and Strategy into account. It is therefore important not to modify Position entries directly in the database. In case transactions are added or modified manually to the database, please the management action reset position and cash balances in the Figure 10.3, “AlgoTrader UI Management”

### 7.2.13. Position Service

The PositionService provides the following position related methods:

- closePosition closes a single position
- closeAllPositions closes all positions in the system
- reducePosition reduces a position by the specified quantity
- transferPosition transfers a position from one strategy to another
- resetPositions calculates all Position based on Transactions in the database and makes adjustments if necessary.

### Note

Closing and Reducing a position through the PositionService requires the definition of an order_preference with the name DEFAULT. For further details see Section 16.2.1, “Order Preferences”
The default order preference also includes an account, which means this feature is typically only usable with one account/adapter. If more than one account is in use, positions should be closed through the Section 7.2.11, “Order Service” by sending an order with a quantity that will offset the current position.

7.2.14. Property Service

The PropertyService can be used to assign arbitrary properties to the following classes:

- Account
- Exchange
- Order
- OrderPreference
- Position
- Security
- SecurityFamily
- Strategy
- Subscription
- Transaction

These classes are derived from the abstract class PropertyHolder. One or more Properties can be assigned to them. A Property can be any Java type (including Integer, Double, BigDecimal, String, Date, Boolean, etc.) but also arbitrary Java objects as long as the type implements Serializable.

Using the PropertyService a Property can be added as follows:

```java
getPropertyService().addProperty(StrategyImpl.class, 12, "myPropertyName", 12.12345);
```

The above example will add a Property named myPropertyName and value = 12.12345 to Strategy with id = 12.

Adding a custom object (implementing Serializable) is possible also:

```java
MyObject myObject = new MyObject("abc", 123, 22.44);
getPropertyService().addProperty(StrategyImpl.class, 12, "myPropertyName", myObject);
```

A Property can be deleted by using the following method:
getPropertyService( ).removeProperty(StrategyImpl.class, 12, "myPropertyName");

Properties are available on the corresponding PropertyHolder objects as follows:

double value = strategy.getProperty("myDoubleProperty", Double.class);
// or
double value = (Double) strategy.getProperty("myTextProperty");

All properties assigned to a particular PropertyHolder can be retrieved as follows:

Map<String, Object> properties = strategy.getProperties();

7.2.15. Reference Data Service

Amongst others reference Data consists of static data like Security, SecurityFamily, SecurityReference, Account Entities. Reference Data can either be configured in the database directly through the corresponding tables, one can use the ReferenceDataService and corresponding ReferenceDataStarter, or conveniently the Reference Data Manager UI. For further details see Chapter 19, Reference Data

7.2.16. Market Data Cache Service

The MarketDataCacheService is intended to provide current market data and exchange rates to the strategy. the MarketDataCacheService keeps a local copy of each subscribed Security.

To access the last traded price of an instrument one can use the following code inside strategies:

TickVO tick = (TickVO) getMarketDataCacheService( ).getCurrentMarketDataEvent( securityId );
BigDecimal lastPrice = tick.getLast();

To access the access the current exchange rate between USD and EUR one can use the following code inside strategies:

double rate = getMarketDataCacheService( ).getForexRate( Currency.USD, Currency.EUR );

7.2.17. Lookup Service

the LookupService provides a large number of lookup methods for all objects available in the database. Examples:

• getSecurityBySymbol gets a security by its symbol
- **getExchangeByCode** gets an Exchange by its exchange code
- **getPositionBySecurityAndStrategy** gets a Position by Security and Strategy
- **getOpenPositionsByStrategy** gets open Positions for the specified Strategy
- **getAccountByName** gets an Account by its name

In addition to standard lookup methods above the `LookupService` also provides the following the generic lookup methods `find` and `findUnique` that can be used in situations where a standard lookup method is not available. These methods can be used as follows:

```java
String query = "from StrategyImpl where name = :strategyName";
NamedParam param = new NamedParam("strategyName", "ABC");

Strategy strategy = getLookupService().find(Strategy.class, query, QueryType.HQL, false, param);
```

Please consult the [JavaDoc](http://doc.algotrader.ch/javadoc/ch/algotrader/service/LookupService.html) for a full list of available methods.

In order to minimize the number of hits to the database the `LookupService` uses various levels of caching when reading from the database.

### 7.2.18. Strategy Service & Config Aware Strategy Service

All strategy main classes need to either extend `StrategyService` or `ConfigAwareStrategyService`. The `ConfigAwareStrategyService` provides the same functionality as the `StrategyService` but in addition also provides a reference to the strategy config object. For further details see *Chapter 4, Strategy Development*.

### 7.2.19. Subscription Service

The Subscription service allows a strategy to subscribe for market data. For that purpose the service provides several methods:

To subscribe for market data use the following method:

```java
getSubscriptionService().subscribeMarketDataEvent(strategyName, securityId, adapterType);
```

**Note**

The `adapterType` specifies the adapter to use when subscribing for market data (e.g. `IB_NATIVE` specifies the InteractiveBrokers native API adapter)

---

2 [http://doc.algotrader.ch/javadoc/ch/algotrader/service/LookupService.html](http://doc.algotrader.ch/javadoc/ch/algotrader/service/LookupService.html)
Upon subscription market data will be feed to the trading strategy that initiated the market data subscription. Market data will be feed to the corresponding Section 4.3.1.2, “Event Handler Methods” (e.g. onBar and onTick) and also into the Esper Engine (if using Esper) where they are available as Bar and Tick events.

To unsubscribe market data use the following method:

```
getSubscriptionService().unsubscribeMarketDataEvent(strategyName, securityId, adapterType);
```

The SubscriptionService also supports the subscription for GenericEvents, see Section 17.6, “Generic Events”

### 7.2.20. Reset Service

The ResetService can be used to reset the state of the database to a pre-defined state either before a simulation or if a reset of live trading is required.

To reset a live trading system multiple types of resets can be specified to the reset method using the Enumeration ResetType

- **TRADES**
  - deletes all transactions (except the initial CREDIT)
  - resets all cash balances (except the one associated with the initial CREDIT)
  - deletes all non-persistent positions and resets all persistent ones

- **ORDERS**
  - delete all orders, order stats as well as order properties

- **SUBSCRIPTION**
  - deletes non-persistent subscriptions

- **COMBINATIONS_AND_COMPONENTS**
  - deletes non-persistent combinations and components

- **PROPERTIES**
  - deletes non-persistent properties

- **MEASUREMENTS**
  - deletes measurements

- **PORTFOLIO_VALUES**
  - deletes portfolio values

- **OPTIONS**
  - deletes all options
FUTURES
   deletes all futures

MARKET_DATA
   deletes all bar and tick data

The method resetSimulation will reset the following items before each Simulation: Trades, Subscriptions, Combinations/Components, Properties, Options (if option prices are simulated) and Futures (if future prices are simulated).

7.3. Value Object

In contrast to Entities which are used to persist information, Value Objects are typically used for transmitting objects (e.g. via JMS or RMI). For each Entity a corresponding Value Object is generated. Value Objects are immutable (i.e. all fields are final and need to be set through the constructor).

Each Entity contains an inner Converter class that can be used to convert the Entity to its corresponding Value Object.

In addition to Value Objects ValueObjectBuilders exist which help creating Value Objects. Example:

```java
MarketOrderVO order = MarketOrderVOBuilder.create()
   .setStrategyId(strategyId)
   .setAccountId(accountId)
   .setSecurityId(securityId)
   .setQuantity(quantity)
   .setSide(side)
   .build();
```

For a full list of all Value Objects please visit our JavaDoc³

7.4. Enumerations

For selectable items with a fixed number of choices AlgoTrader contains Java 5 Enumerations. For a full list of all Enumerations please visit our JavaDoc⁴

³ http://doc.algotrader.ch/javadoc/ch/algotrader/vo/package-frame.html
⁴ http://doc.algotrader.ch/javadoc/ch/algotrader/enumeration/package-frame.html
Esper Engine

AlgoTrader uses the CEP (Complex Event Processing) engine Esper\(^1\). AlgoTrader based strategies can optionally make use of a dedicated Esper engine in addition to the Esper engine used by the AlgoTrader server.

### 8.1. Esper Introduction

Esper is an Event Stream Processing (ESP) and event correlation engine (CEP, Complex Event Processing). Targeted to real-time Event Driven Architectures (EDA), Esper is capable of triggering custom actions written as Plain Old Java Objects (POJO) when event conditions occur among event streams. It is designed for high-volume event correlation where millions of events coming in would make it impossible to store them all to later query them using classical database architecture.

A tailored Event Processing Language (EPL) allows expressing rich event conditions, correlation, possibly spanning time windows, thus minimizing the development effort required to set up a system that can react to complex situations.

Esper is a lightweight kernel written in Java which is fully embeddable into any Java process. It enables rapid development of applications that process large volumes of incoming messages or events.

#### 8.1.1. Introduction to event streams and complex events using Esper

Information is critical to make wise decisions. This is true in real life but also in computing, and especially in the finance and trading area. Information flows in from different sources in the form of messages or events (e.g. market data events), giving a hint on the state at a given time such as stock price. That said, looking at those discrete events is most of the time meaningless. A trader needs to look at the stock trend over a period, possibly combined with other information to make the best deal at the right time.

While discrete events when looked one by one might be meaningless, event streams (i.e. an infinite set of events) considered over a sliding window and further correlated, are highly meaningful, and reacting to them with the minimal latency is critical for effective action and competitive advantage.

Relational databases or message-based systems such as JMS make it really hard to deal with temporal data and real-time queries. Indeed, databases require explicit querying to return meaningful data and are not suited to push data as it changes. JMS systems are stateless and require the developer to implement the temporal and aggregation logic himself. By contrast, the Esper engine provides a higher abstraction and intelligence and can be thought of as a database turned upside-down: instead of storing the data and running queries against stored data, Esper allows applications to store queries and run the data through. Response from the Esper engine is real-time when conditions occur that match user defined queries. The execution model is thus continuous rather than only when a query is submitted.

In Esper, a tailored EPL allows registering queries in the engine. A listener class, which is basically a POJO, will then be called by the engine when the EPL condition is matched as events flow in. The EPL enables to

\(^1\) [http://www.espertech.com/esper/](http://www.espertech.com/esper/)

\(^2\) Most of this section has been reproduced from the Esper website
express complex matching conditions that include temporal windows, joining of different event streams, as well as filtering, aggregation, and sorting. Esper statements can also be combined together with “followed by” conditions thus deriving complex events from more simple events. Events can be represented as JavaBean classes, legacy Java classes, XML document or `java.util.Map`, which promotes reuse of existing systems acting as messages publishers.

A trivial yet meaningful example is as follow: assume a trader wants to buy Google stock as soon as the price goes below some floor value, not when looking at each tick but when the computation is done over a sliding time window, say of 30 seconds. Given a `TickVO` event bean with a last price field and a reference to a Security ID and the following EPL, a listener POJO would get notified as ticks come in to trigger the buy order:

```python
select
  avg(last)
from
  TickVO.win:time(30 sec)
where
  securityId=12
```

### 8.1.2. Event representations

Java classes are a simple, rich and versatile way to represent events in Esper. Java classes offer inheritance and polymorphism via interfaces and super-classes, and can represent a complex business domain via an object graph. In AlgoTrader event class like `TickVO`, `BarVO`, `OrderVO`, `OrderStatusVO` etc. are made available to Esper engines by default. In addition any arbitrary java class can be used inside Esper engines after declaring them.

In addition to Java classes, Maps and XML are an alternative way of representing events.

### 8.1.3. Event Stream Analysis

EPL statements derive and aggregate information from one or more streams of events, to join or merge event streams, and to feed results from one event stream to subsequent statements.

EPL is similar to SQL in it's use of the `select` clause and the `where` clause. However EPL statements instead of tables use event streams and a concept called `views`. Similar to tables in an SQL statement, views define the data available for querying and filtering. Views can represent windows over a stream of events. Views can also sort events, derive statistics from event properties, group events or handle unique event property values.

This is a sample EPL statement that computes the average of the last price for the last 30 seconds of Tick events:

```python
select
  avg(last)
from
  TickVO.win:time(30 sec)
```
A sample EPL that returns the average of the last price per symbol for the last 100 Ticks.

```plaintext
select
    securityId
    avg(last) as averagePrice
from
    TickVO.win:length(100)
group by
    securityId
```

This example joins 2 event streams. The first event stream consists of Bar events for which we keep the last 30 minutes (1800 seconds). The second stream is Tick events for which we consider the last 30 seconds. The streams are joined on `securityId`.

```plaintext
select
    bar.securityId as securityId,
    max(bar.high) as maxHigh,
    min(bar.low) as minLow,
    last(tick.last) as lastPrice
from
    BarVO.win:time(30 min) as bar,
    TickVO.win:time(30 sec) as tick
where
    bar.securityId = tick.securityId
```

### 8.1.4. Combining Pattern Matching with Event Stream Analysis

Patterns match when a sequence (or absence) of events is detected. Pattern match results are available for further analysis and processing.

The pattern below detects a situation where an `OrderStatus` event is not followed by another `OrderStatus` event corresponding to the same internal order id within 10 seconds. The statement further counts all such occurrences grouped per internal order id.

```plaintext
select
    a.intId,
    count(*)
from
    pattern [every a=OrderStatus
                -> (timer:interval(10 sec) and not OrderStatus(intId=a.intId))]
group by
    id
```
8.1.5. Named windows

A named window is a global data window that can take part in many statement queries, and that can be selected-from, inserted-into and deleted-from by multiple statements. Named windows are similar to a table in a relational database system.

One can create a named window for example as follows:

```java
create window
    SecurityWindow
as
    (symbol String, triggerPrice double)
```

One can trigger a select, update or delete when an event arrives. Here we show a select that simply counts the number of rows:

```java
on
    TriggerEvent
select
    count(*)
from
    SecurityWindow
```

Named windows can also be queried with fire-and-forget queries through `ch.algotrader.esper.Engine.executeQuery` and `ch.algotrader.esper.Engine.executeSingelObjectQuery`.

8.1.6. Variables

A variable is a scalar, object or event value that is available for use in all statements including patterns. Variables can be used in an expression anywhere in EPL.

8.2. Esper Quick Start Guide

This quick start guide provides step-by-step instructions for using Esper inside AlgoTrader.

8.2.1. Event Types

Java classes are a good choice for representing events, however Map-based or XML event representations can also be good choices depending on the architectural requirements.

AlgoTrader provides a number of Value Objects that can be used as Esper Events (e.g. `TickVO`, `BarVO`, `OrderStatusVO`, etc.)

3Most of this section has been reproduced from the Esper website.
8.2.2. Creating a Statement

A statement is a continuous query registered with an Esper engine instance that provides results to listeners as new data arrives, in real-time, or on demand via the iterator API (see `ch.algotrader.esper.Engine.executeQuery`).

The next code snippet shows an Esper module containing a continuous query. The query returns the average price over all `TickVO` events that arrived in the last 30 seconds:

```sql
select
  avg(price) as avgPrice
from
  TickVO.win:time(30 sec)
```

Each of the Esper engines inside AlgoTrader can contain several modules. Modules specified through the `initModules` and `runModules` attribute of the Esper Engine Spring Bean Definition are loaded automatically on start-up.

```xml
<bean id="testEngine" class="ch.algotrader.esper.EngineFactoryBean">
  <property name="strategyName" value="TEST"/>
  <property name="configResource" value="esper-test.cfg.xml"/>
  <property name="configParams" ref="testConfigParams"/>
  <property name="initModules" value="market-data"/>
  <property name="runModules" value="order-handling"/>
</bean>
```

**Note**

A module definition of `market-data` will look for a module file called `module-market-data.epl`.

Additional modules can be deployed at runtime using the method `Engine.deployModule`.

8.2.3. Adding a Subscriber

A subscriber object is a direct binding of query results to a Java object. The object, a POJO, receives statement results via method invocation. The subscriber class does not need to implement an interface or extend a superclass. Only one subscriber object may be set for a statement.

Subscriber objects have several advantages over listeners. First, they offer a substantial performance benefit: Query results are delivered directly to the Java method(s) through Java virtual machine method calls, and there is no intermediate representation (`EventBean`). Second, as subscribers receive strongly-typed parameters, the subscriber code tends to be simpler.
The subscriber class must provide a public method to receive events. The number and types of parameters declared by the update method must match the number and types of columns as specified in the select clause, in the same order as in the select clause.

For the following statement:

```java
@Subscriber(className='ch.algotrader.listener.MySubscriber#process)
select
    orderId, price, count(*)
from
    OrderEvent;
```

the Subscriber class looks as follows:

```java
public class MySubscriber {
    public void process(String orderId, double price, long count) {
        System.out.println("orderId=" + orderId + ",price=" + price + ",count=" + count);
    }
}
```

### 8.2.4. Adding a Listener

Listeners are invoked by the engine in response to one or more events that change a statement's result set. Listeners implement the `UpdateListener` interface and act on `EventBean` instances as the next code snippet outlines:

```java
public class MyListener implements UpdateListener {
    public void update(EventBean[] newEvents, EventBean[] oldEvents) {
        System.out.println("avg=" + newEvents[0].get("avgPrice"));
    }
}
```

By attaching the listener to the statement via the following annotation the engine provides the statement's results to the listener:

```java
@Listeners(classNames={'ch.algotrader.listener.MyListener'})
select
    avg(price) as avgPrice
from
    TickVO.win:time(30 sec)
```
8.2.5. Sending events

The runtime API accepts events for processing. As a statement’s results change, the engine indicates the new results to listeners right when the events are processed by the engine.

Incoming market data events (e.g. Ticks), submitted Orders, OrderStatus events, received Fills, etc are automatically sent into the corresponding Esper engines.

Additionally custom events can be sent into an Esper engine. The following code snipped creates an arbitrary event and sends it into the Esper engine instead an AlgoTrader strategy named EXAMPLE.

```java
MyEvent event = new MyEvent("TEST_EVENT");
engine.sendEvent(event);
```

8.2.6. Configuration

Esper Configuration helps make statements more readable and provides the opportunity to plug-in extensions.

Each Esper Engine loads the default esper-common.cfg.xml file. In addition strategies load all Esper configuration files named esper-xxx.cfg.xml in the class path. This configuration file defines settings like:

- Event Types
- Auto Import Classes & Packages
- Custom Aggregation Functions
- Variables
- General Engine Settings

8.3. Esper Documentation

Esper provides in depth documentation[^4].

The following chapters of the Esper Documentation are relevant for developing trading strategies with AlgoTrader based on Esper:

- 2. Event Representations[^5]
- 5. EPL Reference: Clauses[^7]

8.4. AlgoTrader specific Esper Artifacts

8.4.1. Engine & EngineManager

Inside AlgoTrader Esper engine instances are wrapped by the EngineImpl which implements the Engine interface. Individual Engine instances can be located through the EngineManager singleton.

8.4.1.1. Engine

The Engine interface has methods available for the following tasks:

• deployment / un-deployment of Esper statements and modules
• sending events
• execute fire-and-forget queries
• retrieve current statement state
• management of the Esper clock
• synchronized processing (coordination) of events from different sources into the Esper engine
• management of Esper variables
• adding Section 8.4.8, “Callbacks”

For further information please visit the relevant JavaDoc.

For testing purposes there is an abstract do-nothing implementation of the Engine interface available named AbstractEngine.

8.4.1.2. EngineManager

The EngineManager interface represents the main entry point to different Engines running inside the JVM. The EngineManager has methods available for the following tasks:

---
8 http://esper.espertech.com/release-5.5.0/esper-reference/html/event_patterns.html
9 http://esper.espertech.com/release-5.5.0/esper-reference/html/epl-operator.html
10 http://esper.espertech.com/release-5.5.0/esper-reference/html/functionreference.html
12 http://esper.espertech.com/release-5.5.0/esper-reference/html/examples.html
• Lookup of available Engines
• Query of the current Engine time
• Management of statement metrics

For further information please visit the relevant JavaDoc.

Note

Engine instances are managed and configured through Spring configuration. Engines are standard Spring managed beans that get automatically registered with EngineManager upon startup.

8.4.2. Modules

8.4.2.1. AlgoTrader Server Modules

The AlgoTrader Server contains the following Esper modules:

Table 8.1. AlgoTrader Server modules

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module-algo-xxx.epl</td>
<td>Each Execution Algo has its own module</td>
</tr>
<tr>
<td>module-combination.epl</td>
<td>Combination / Component related functionality (see Chapter 24, Synthetic Securities and Derivative Spreads)</td>
</tr>
<tr>
<td>module-current-values.epl</td>
<td>Store current market data values</td>
</tr>
<tr>
<td>module-cng.epl</td>
<td>Coinigy specific statements</td>
</tr>
<tr>
<td>module-ib.epl</td>
<td>IB specific statements</td>
</tr>
<tr>
<td>module-market-data.epl</td>
<td>Statements related to market data</td>
</tr>
<tr>
<td>module-metrics.epl</td>
<td>Statements needed for Engine Metrics</td>
</tr>
<tr>
<td>module-performance.epl</td>
<td>Evaluation of performance metrics</td>
</tr>
<tr>
<td>module-portfolio.epl</td>
<td>Portfolio management functions</td>
</tr>
<tr>
<td>module-prepared.epl</td>
<td>Prepared Statements available to strategies</td>
</tr>
<tr>
<td>module-server-prepared.epl</td>
<td>Server Side prepared Statements</td>
</tr>
<tr>
<td>module-trades.epl</td>
<td>Statements related to orders and executions</td>
</tr>
</tbody>
</table>

Note

init and run modules of the AlgoTrader Server can be defined through config properties in conf-core.properties.
8.4.2.2. Strategy Modules

Strategies are completely free in the definition of their Esper Statements. Examples of Statements used by strategies are:

- Creation of technical indicators (e.g. Moving Average, Bollinger Bands, ATR, etc.)
- Creation of trade signals
- Trend evaluation
- Open / Close / Increase / Reduce Positions
- Roll Positions (for Options and Futures)
- Pattern recognition

**Note**

It is generally recommended to use Esper statements for anything up to signal generation but then use Java for execution of actions (e.g. send and order, set a stop or close a position)

8.4.3. Tags

In addition to the *Esper standard tags*\(^\text{13}\) the following tags are available:

**Table 8.2. Esper tags**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Condition(key='xxx')</td>
<td>Statement is only deployed if defined configuration parameter is set to &quot;true&quot;</td>
</tr>
<tr>
<td>@SimulationOnly</td>
<td>Statement is only deployed in simulation</td>
</tr>
<tr>
<td>@RunTimeOnly()</td>
<td>Statement is only deployed in Live-Trading mode</td>
</tr>
<tr>
<td>@Listeners(classNames={&quot;...&quot;})</td>
<td>attaches one or multiple listeners to the statement</td>
</tr>
<tr>
<td>@Subscriber(className='...')</td>
<td>attaches a subscriber to the statement</td>
</tr>
</tbody>
</table>

8.4.4. Subscribers

The system provides the following Subscribers out-of-the-box:

IndicatorSubscriber

- Prints all values as a comma-separated-list (CSV) to the file `files/report/IndicatorReport.csv` (Headers are not available).

TestSubscriber
Prints all values to the Log by using the `toString` method of the event object.

VoidSubscriber
Do-nothing subscriber, useful when select clauses call static methods

ExceptionSubscriber
Prints a value as an Error to the Log.

Any public method of a component defined in the Spring application context can potentially be used as a subscriber provided the parameter signature of the method can be supported by Esper: One can define a subscriber by specifying a Spring bean name followed by hash (#) followed by a method name exposed by this bean.

```java
@Name('TAKE_PROFIT')
@Subscriber(className='boxService#takeProfit')
```

The syntax also support property placeholder expansion. This can especially useful when using multiple instances of the same strategy in the same JVM process.

```java
@Name('TAKE_PROFIT')
@ Subscriber(className='${strategyName}Service#takeProfit')
```

### 8.4.5. Listeners

The system provides the following Listeners out-of-the-box:

**IndicatorListener**
Prints all values as a comma-separated-list (CSV) to the file `files/report/IndicatorReport.csv`. Headers will be extracted from the supplied Statement.

**RendererListener:**
Prints all values to the Log in XML format.

**TestListener**
Prints all values to the Log by using the `toString` method of the event object.

**StatementAwareTestListener**
Prints all values including the statement name to the Log by using the `toString` method of the event object.

### 8.4.6. Service method invocation in Esper scripts

AlgoTrader Esper statements can access standard platform services directly

By default AlgoTrader exposes the following services to Esper statements, which strategies can make use of:
• LookupService
• PortfolioService
• CalendarService
• OrderService
• PositionService
• MarketDataService
• OptionService

Here are some examples. To retrieve a security based on its securityId one can use the following statement:

```java
select
  bar,
  security
from
  BarVO as bar unidirectional,
  method:lookupService.getSecurity(bar.securityId) as security;
```

To retrieve an open order by its intId on can use the following statement:

```java
select
  fill,
  openOrder
from
  FillVO as fill unidirectional,
  method:orderService.getOrderByIntId(fill.orderIntId) as openOrder;
```

### 8.4.7. Aggregation Functions

The system provides the following custom aggregation functions:

#### 8.4.7.1. ExponentialMovingAverage

The `ExponentialMovingAverageFunction` can be used to construct an exponential moving average of a time series, e.g.:

```java
select ema(last, 10) from TickVO;
```
This will create a 10-period exponential moving average of the Tick last price.

### 8.4.7.2. GenericTALibFunction

The `GenericTALibFunction` is a port of [ta-lib](http://ta-lib.org/) to AlgoTrader. It supports all TA-Lib operations.

Please consult [TA-Lib](http://doc.algotrader.ch/ta-lib.html) for a list of all TA-Lib methods and their parameters.

If the TA-Lib Function returns just one value, the value is directly exposed by the `AggregationFunction`.

Example: The TA-Lib function `movingAverage` has just one double typed return value which can be accessed directly.

```
insert into MovingAverage
select talib("movingAverage", close.doubleValue(), 30, "Sma") as result
from BarVO;

select result
from MovingAverage;
```

If the TA-Lib Function returns multiple-values, a dynamic class will be generated on the fly, which gives access to properly typed return-values. All return value names are lower-case!

Example: The TA-Lib function `stochF` has return values: `outFastK` and `outFastD`. The returned dynamic class will have double typed properties by the name of: `fastk` and `fastd` (all lowercase).

```
insert into Stochastic
select talib("stochF", high.doubleValue(), low.doubleValue(), close.doubleValue(), 3, 2, "Sma") as result
from BarVO;

select result.fastk, result.fastd
from Stochastic;
```

Some functions are influenced by the entire range of the past data. These functions are sometimes called functions with memory. An example is the EMA (Exponential Moving Average). For these functions an optional unstable period parameter can be specified. The following statement will create a 30 period moving average with an unstable period of 10.

```
insert into MovingAverage
select talib("movingAverage", close.doubleValue(), 30, "Ema", 10) as result
from BarVO;
```

14 http://ta-lib.org/
15 http://doc.algotrader.ch/ta-lib.html
For further details about the unstable period please see: SetUnstablePeriod\(^\text{16}\)

For additional information please visit the corresponding JavaDoc\(^\text{17}\).

\[\text{Note}\]

As an alternative to the ta-lib based exponential moving average function the Esper aggregation function Section 8.4.7.1, “ExponentialMovingAverage” can be used which keeps the entire history an not just the unstable period.

### 8.4.8. Callbacks

#### 8.4.8.1. First tick callback

A \(\text{BiConsumer<String, List<TickVO>>}\) function can be registered with the Esper engine using the \(\text{Engine#addFirstTickCallback()}\) method. Whenever at least one Tick of each specified security has arrived the consumer will be executed receiving the name of the strategy and a list of ticks as input.

A typical use case of a first tick callback looks like this:

```java
engine.addFirstTickCallback(securityIds, (strategyName, ticks) -> {
    ...
});

for (long securityId : securityIds) {
    getSubscriptionService().subscribeMarketDataEvent(strategyName, securityId);
}
```

#### 8.4.8.2. Trade callback

A \(\text{BiConsumer<String, List<OrderStatusVO>>}\) function can be registered with the Esper engine using the \(\text{Engine#addTradeCallback()}\) method. Whenever all corresponding orders have been fully executed or canceled the consumer will be executed receiving the name of the strategy and a list of order status messages as input.

In order to correctly associate the trade callback with a specific order an \(\text{orderId}\) has to be retrieved from the order service and set onto the order before attaching the trade callback.

A typical use case of a trade callback looks like this:

```java
String orderId = getOrderService().getNextOrderId(order.getClass(), accountId);
```

\(^{16}\) http://ta-lib.org/d_api/ta_setunstableperiod.html
\(^{17}\) http://doc.algotrader.ch/javadoc/ch/algotrader/esper/aggregation/GenericTALibFunction.html
The `Engine#addFullExecutionCallback()` method can be used to register a callback logging an error if the order did not execute fully.

```java
String orderId = getOrderService().getNextOrderId(order.getClass(), accountId);
order.setIntId(orderId);
engine.addFullExecutionCallback(Collections.singleton(orderId));
getOrderService().sendOrders(orders);
```

**Note**

With Fix based Broker / Exchange connections the `TradeCallback` only works with the initial order but not with any subsequent order modifications.

### 8.4.8.3. Trade persisted callback

A `Consumer<List<OrderCompletionVO>>` function can be registered with the Esper engine using the `Engine#addTradePersistedCallback()` method. Whenever all corresponding orders have been fully executed and all corresponding database transactions (e.g. `OrderStatus`, `Transaction` and `Position`) have been fully executed the consumer will be executed receiving a list of `OrderCompletionVO` objects as input.

This callback is particularly useful for situations where one needs to have a guarantee that all order related database transactions have been fully executed before continuing with next steps, e.g. to retrieve the current position quantity after an order has been executed. If using a regular trade callback for this, the Position Entity might not have been fully persisted by the time the consumer is executed. However when using the trade persisted callback it is guaranteed that the Position Entity has been fully updated in the database.

In order to correctly associate the trade callback with a specific order an `orderId` has to be retrieved from the order service and set onto the order before attaching the trade callback.

A typical use case of a trade persisted callback looks like this:

```java
String orderId = getOrderService().getNextOrderId(order.getClass(), accountId);
```
order.setIntId(orderId);

engine.addTradePersistedCallback(Collections.singleton(orderId), orderCompletions -> {
    ...
});

getOrderService().sendOrders(orders);

Note

The Trade persisted callback is only supported in runtime mode but it is not supported in simulation mode. It is recommended to use the Section 8.4.8.2, “Trade callback” instead in simulation mode. The Trade persisted callback will only get executed if there has been at least one (partial) execution but not if an order has been cancelled or rejected before there has been any execution.

8.4.8.4. Open / close position callback

A Consumer<PositionMutationVO> function can be registered with the Esper engine using the Engine#addOpenPositionCallback() or Engine#addClosePositionCallback() methods. Whenever a corresponding transaction causes a new position to open / close the consumer will be executed receiving PositionMutationVO as input.

engine.addOpenPositionCallback(securityId, positionMutation -> {
    ...
});

engine.addClosePositionCallback(securityId, positionMutation -> {
    ...
});

Note

It is guaranteed that the position is fully persistent to the database by the time the consumer is called.

8.4.8.5. Timer callback

A Consumer<Date> function can be registered with the Esper engine using the Engine#addTimerCallback() method. Whenever the system time has reached the specified time the consumer will be executed receiving the actual time as input. To distinguish multiple timers from each other an additional name parameter is available to name each timer instance.
```java
LocalDate time = DateTimeLegacy.toLocalDate(DateTime.now().plusHours(1));

engine.addTimerCallback(time, "in one hour", date -> {
    ...
});
```

8.5. Esper Threading

Esper has several options for enabling a multi-threaded environment, see *Esper API Threading*\(^\text{18}\)

In Live-Trading Mode AlgoTrader uses outbound threading with 3 threads by default. This means that all Subscriber / Listener Tasks are handled by a thread-pool of 3 threads. The number of outbound threads can be changed inside `conf.properties` or via `Section 2.3, "VM Arguments"`:

```
# number of Esper outbound threads to be used in Live Trading Mode for the Server Engine
misc.outboundServerEngineThreads = 3

# number of Esper outbound threads to be used in Live Trading Mode for the Strategy Engines
misc.outboundStrategyEngineThreads = 3
```

For debugging reasons AlgoTrader logs the name of the thread using log4j, see *Chapter 29, Logging*

---

Database

9.1. Instances

AlgoTrader uses MySQL\(^1\) to store transaction data. In addition an embedded in-memory H2\(^2\) database can be used for simulations.

9.2. Flyway

AlgoTrader uses the database migration library flyway\(^3\) to keep databases in-line with the current version of AlgoTrader. Flyway executes database migration scripts to ensure that the database is in the state corresponding to the version of AlgoTrader installed.

Flyway is embedded into the AlgoTrader platform, and can perform a migration of the Database on each startup automatically. In case the configured schema does not exist on MySQL, it will create it. This schemas default-name is algotrader. The feature is activated on all startup-scripts including docker but deactivated in the development environment. misc.flywayMigrateOnStartup is the property to control this behaviour, it can be passed as a VM Argument.

```
-Dmisc.flywayMigrateOnStartup=true
```

In case you need to reset the Database, the easiest way is to delete the corresponding schema on MySQL and start AlgoTrader. For further migration related issues, a class within AlgoTrader ch.algotrader.starter.FlywayRunner can be used to execute other flyway commands. It is not recommended to use Flyways command-line-tool instead, because of potential problems when using several flyway versions on the same schema.

For additional information on flyway please visit the flyway documentation\(^4\).

The database migration scripts are located in /bootstrap/conf/flyway.

9.3. Files

The directory /bootstrap/conf contains all relevant database files:

- flyway/sql contains the flyway migration scripts that will be executed to initialize and update the MySQL database
- src/main/resources/db-samples/h2 and src/main/resources/db-samples/mysql database sample data for accounts, exchanges, securities (FX majors, S&P 500 & EURO STOXX 50 stocks), security families and order preferences.

---

\(^1\) https://www.mysql.com/
\(^2\) http://www.h2database.com/
\(^3\) https://flywaydb.org/
\(^4\) https://flywaydb.org/documentation/maven/
With the embedded database H2 all H2 scripts that are available in the class path under the following wild card pathname will be loaded by the system: `h2/h2-*.sql` and `/db/h2/h2-*.sql`.

### 9.4. Data Source

AlgoTrader provides different Data Source for different scenarios. One of the available Data Sources needs to be configured via the following VM argument:

```
-Dspring.profiles.active=<dataSource>
```

`dataSource` can be one of the following

- **pooledDataSource**: A Data Source that uses connection pooling based on [C3P0](https://www.mchange.com/projects/c3p0/)
- **singleDataSource**: A Data Source that uses one single database connection and no connection pooling.
- **embeddedDataSource**: An [H2](http://www.h2database.com/) based in-memory / in-process Data Source that is ideal for simulation of strategies. Using the `embeddedDataSource` reduces the duration of back-test runs by 30%-50% and allows multiple parallel simulations on the same machine without needing to installing a physical database.
Chapter 10.

Client

AlgoTrader provides different types of clients all of which are targeting a different audience and use case:

- The html5 UI: used to monitor/manage trading activities while AlgoTrader is running
- Eclipse IDE: Contains the Strategy Creation Wizard, the Config Editor as well as the strategy simulation environment
- Reference Data Manager (RDM)
- Historical Data Manager (HDM)

10.1. HTML5 UI

Note

Officially AlgoTrader only supports Google Chrome. However, the AlgoTrader UI will most likely work with the most recent version of any modern browser.

The AlgoTrader UI provides the following features.

- Live Market Data updated in real-time.
- Tables showing current Orders, Transactions, Positions and Market Data.
- All tables provide (multicolumn) sorting, filtering, column selection and reordering, and scrolling.
- Display of Alarms and Notifications in case something unexpected happens.
- Supports multiple currencies and automatic currency conversion.
- Since the UI is based on HTML5, it can easily be integrated into corporate IT infrastructures using firewalls, VPNs, and remote locations.
- Auto-completion feature for security selection.

When AlgoTrader server starts, it automatically opens the client.

To manually open the client, point the browser to one of the following URL.

http://localhost:9090
10.1.1. Header

Figure 10.1. AlgoTrader UI Header

Using the strategy selection menu located at the top of the screen (to the right), one can select either an aggregated view over the entire system or a strategy specific view.

When a single strategy is selected the client will show orders, transactions, positions and market data subscriptions related to the selected strategy only. When ALL is selected the client will show orders, transactions, positions and market data subscriptions for all strategies.

The header tiles show general figures (like Net Liquidity, Unrealized P&L, Cash, etc.). The default valuation currency is USD. You can change it by updating the misc.portfolioBaseCurrency value in the conf.properties file.

If one opens a menu on top right corner (hamburger menu) one can see Settings link which opens the settings form. The following settings are available there:

1. Tiles: one can configure the visibility of general figures in header

2. Order defaults: default order related values like default quantity and default time-in-force

3. Tables update throttling in ms - sets the update interval of all tables, e.g. if the interval is 333ms, the tables will buffer all data updates and only make display changes every 333ms. Increasing that parameter may help if the UI is displaying a lot of data (>100 rows) and becomes unresponsive, e.g. reacts slowly to clicking on buttons, sorting columns etc.

4. Use Trading View historical data - when checked means that the historical data for chart will be coming from TradingView's own data source, if unchecked the chart will take historical data from data source the AlgoTrader back end is configured with
To open the general management form, click on the management menu on the top of the screen (next to Strategy selector).

The management menu provides the following operations:
Figure 10.4. AlgoTrader UI Management Form

Notifications are displayed in the lower right hand side of the screen

![AlgoTrader UI Management Form](image)

Figure 10.5. AlgoTrader UI Notification

There are three types of notifications: informations (green), warnings (orange) and alerts (red). In addition to warnings and alerts appearing in bottom right corner, a bell icon will appear at the top right of the screen.

To open the list of all warnings and alerts click on the bell icon. Alerts can be removed from the list by clicking on the close icon
10.1.2. Order Table

The orders will appear in the Order Table in real time. Executed orders are removed from the table after a preset number of seconds which can be configured through the Settings Dialog.

Manual orders can be entered using the order entry form. After entering the order (by specifying security, trade side, order type, quantity, strategy, account and trade type), click the Submit button.

To cancel all open orders, click the Cancel All button at the top right of the Order Table.

To cancel a specific open order, click the Cancel icon besides the order.

To modify an open order, click the Modify icon besides the order.
To specify additional order parameters (e.g. time in force and exchange) or place advanced order types (like Algo Orders) click the **Advanced Form** button.

### 10.1.3. Advanced Order Form

This form lets one enter all types of supported orders, including Algo Orders and allows to use Smart Order Routing feature.

The first field at top left corner, **Order Type**, lists all available order types. When Order Type is a simple order type (i.e. Market, Limit, Stop, Stop Limit), then the form allows to specify Time in Force (TIF) and broker/exchange specific parameters, in addition to the Simple Order Form.
For Algo Orders Smart Order Routing is available as well as additional details specific to that type of an Algo Order.

There’s also a possibility to use one of pre-defined Order Preferences to fill out the form, they’re listed in the header of Algo Orders properties section. For further details see Section 16.2.1, “Order Preferences”

Below the order type, the Routing section is displayed which currently has two modes: Crypto and Equity

![Image of Advanced Order Form]

**Figure 10.11. AlgoTrader UI Advanced Order Form - Crypto mode of Routing section**

The Crypto section allows to choose a crypto currency pair from the in DB defined crypto currency pairs and to select a subset of accounts to which the order should be sent. The form will only allow to select those accounts for which corresponding crypto currency pairs are available in the DB. The system will route each child order to the exchange/account that currently has the most favourable price.
The **Equity** section allows to choose a single security, an account and multiple exchanges. The system will route each child order to the exchange that currently has the most favourable price.

### 10.1.4. Algo Order details UI

It is possible to browse and monitor the state of an Algo Order from the web UI. In order to do that, go to the Order tab, and click on the Algo Order icon. A new pop-up window will appear with details of the Algo Order and an additional table with the list of all placed Children orders of the Algo Order (along with details like quantity and state).
10.1.5. Transaction Table

The executed trades are listed in the Transactions Table. Since strategies can produce a lot of transactions, only the most recent 50 are shown on startup. All transactions can of course be seen/exported from the database.

To manually add a transaction, click the Add button at the top right of the Transaction Table.
If a transaction’s fee currency is different from the transaction currency (e.g. for certain crypto-exchanges), the transaction window will show additional rows to reflect the fees.

Where transaction and fee currencies are the same, the transaction fee will be visible in the fee column of the transaction.

Open positions are listed within the Positions Table.
Figure 10.18. AlgoTrader UI Position Table

To close all positions, click the Close All button at the top right of the Position Table.

To close a specific position, click the Close x icon besides the position.

To increase a position by a certain quantity, click the Add + icon besides the position.

To reduce a position by a certain quantity, click the Reduce - icon besides the position.

All of these three actions will pre-fill the order form with appropriate data (security, side, amount in case of close action) and move focus to the form. To send the order please click Submit.

To open a position's security chart, click the Chart icon besides the position.

10.1.7. Market Data Table

Shows market data in real time.

Figure 10.19. AlgoTrader UI Market Data Table

To subscribe for a security's market data, click the Subscribe button at the top right of the Market Data Table.

Figure 10.20. AlgoTrader UI Market Data Subscribe
To unsubscribe to a security’s market data, click the Unsubscribe icon besides the security.

For any given strategy, it is only possible to unsubscribe from securities the strategy has no position. The unsubscribe icon will only be enabled if the corresponding strategy has no position for the given security.

![Unsubscribe for: AAPL APPLE INC](image)

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Feed</th>
<th>Unsubscribe</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVER</td>
<td>IB</td>
<td></td>
</tr>
<tr>
<td>STOCKS</td>
<td>IB</td>
<td></td>
</tr>
<tr>
<td>FOREX</td>
<td>IB</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 10.21. AlgoTrader UI Market Data Unsubscribe**

The security’s chart can be opened by clicking the Chart icon besides the security.

### 10.1.8. Column Selection and Grouping

All Tables have a configuration button at the top right of the table. Click the settings button to select which columns to show. See the following example for the Transaction Table.

To see all columns of a certain type (e.g. Transaction or Strategy) please click on the expand button.

One can also automatically adjust columns' width by clicking on the Auto-size columns button in the Table actions section.
In every table, any column can be sorted (ascending or descending) by clicking on the column header.

A filter can be applied to any column by clicking at the far right part of the column header. The following image shows the transaction table filtered by a specific symbol.

Figure 10.22. AlgoTrader UI Transaction Column Selection

Figure 10.23. AlgoTrader UI Column Filter
10.1.9. CSV Export

All Tables have a Export CSV button at the top right of the table. Clicking the button will initiate the export of all the data visible in given table into a .csv file. Exported files will get unique names with table name and the export date.

10.1.10. Chart Widget

The AlgoTrader UI also comes with the interactive TradingView chart library.

TradingView has regular and advanced chart types and it comes with a massive library of over 100 pre-built technical indicators covering the most popular trading concepts.

The chart widget is useful during strategy development (for initial idea generation, validation, etc.) and for monitoring.

Note

TradingView widget can work in two modes. It can either use data provided by TradingView itself, or it can use custom market data adapters which are configured in AlgoTrader.

In case of TradingView market data source there might be slight differences between market data shown in the market data table and the chart widget.

Please, see the TradingView documentation.

Figure 10.24. AlgoTrader UI Chart Widget

1 https://www.tradingview.com/wiki/Main_Page
To switch market data source between TradingView and custom ones, open settings panel.

**Figure 10.25. AlgoTrader UI Chart Widget settings**

To select particular security to be displayed in the chart, click on the chart icon in the operations column in Market Data or Positions grids.
Figure 10.26. AlgoTrader UI Chart Widget - selecting security
10.1.11. About pop-up

The About pop-up can be opened through the "hamburger" menu in top right corner. The pop-up shows the UI and back-end code version which may be needed in case of support requests.

![About pop-up screenshot](image)

Figure 10.27. "About" pop-up

10.1.12. Technologies

The Algotrader UI is based on the following technologies/architectures:

- HTML5
- React ²
- STOMP³ over WebSockets⁴
- RESTful Web services
- Bootstrap⁵
- TradingView⁶ chart component
- AGGrid⁷ component

10.1.13. HTML5 Custom Widgets

It is possible to extend the AlgoTrader UI with custom widgets in order to visualize strategy specific data or let the user interact with strategy specific functionality (e.g. modify parameters or change the state of a strategy). The following screen shot shows an example of the custom widget in use by the Appendix B, Example Strategy “Box”:

² https://reactjs.org/
³ https://stomp.github.io/
⁴ http://www.websocket.org/
⁵ https://getbootstrap.com/
⁶ https://www.tradingview.com/
⁷ https://www.ag-grid.com/
Figure 10.28. HTML5 Custom Widget Example

HTML5 custom widgets use WebSockets over STOMP to communicate with the strategy.

To integrate the HTML5 custom widget into the main AlgoTrader UI the following items need to be created inside the strategy module (in case you are using the EmbeddedStrategyStarter) or inside the algotrader-core module (in case you are using the ServerStarter). The examples are based on the Appendix B, Example Strategy "Box".

/src/main/resources/html5/index.html

This file contains the layout of the main HTML5 screen including the HTML5 custom widget. The custom widget needs to be included similar to the following code snippet:

```html
<!DOCTYPE html>
```
Please note that the total of all columns and the total of all rows needs to equal 12 (e.g. col-xs-5, col-xs-5 and col-xs-2).
The `index.html` file in strategy's resources replace the file that would be normally served by the AlgoTrader server. The base file can be obtained for example by opening the source of the AlgoTrader Dashboard (CTRL+U in Chrome). Then the file can be extended to contain a place for custom widget (div id="box" in above example) and a code for that widget (box.js)

/src/main/resources/html5/box.html
This file contains the HTML code for the custom widget. Individual tags will be referenced by JavaScript code through tag ids.

```
<h3>Box Strategy</h3>
<table class="table table-striped">
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td id="box_state"></td>
</tr>
</tbody>
</table>
<button id="box_terminate" class="btn btn-danger btn-xs" role="button">
   TERMINATE TRADE
</button>
```

/src/main/resources/html5/box.js
This file contains the JavaScript code for the custom widget:
```
$.get("box.html", function(result){
    $("#box").html(result);
    
    $.get(document.documentURI + "rest/broker/url/ws", function(wsURI){
        var ws = new WebSocket(wsURI, "stomp");
        var stompClient = Stomp.over(ws);
        stompClient.connect({}, function(frame) {
            stompClient.subscribe('/topic/strategy.box.metrics', function(message){
                var metrics = JSON.parse(message.body));
                $("#box_state").text(metrics.state);
            });
        });
    });
});
```
}, { "activemq.retroactive" : true});

$('#box_terminate').on('click', function(event) {
    stompClient.send("strategy.box.terminate", {});
});

Loads the html content
Populate the <div id="box"> tag inside index.html with the html content
Requests the WebSocket URI via REST call
Connects to STOMP of WebSocket
Subscribes for metrics updates of the strategy which are sent to the topic strategy.box.metrics. Setting {"activemq.retroactive" : true} will allow the custom widget to get the last metrics event from the strategy.box.metrics topic upon subscription
Populates the content tags with the contents of the metrics events
Sets button actions. Clicking the terminate button will send an empty message to the strategy.box.terminate topic.

/src/main/java/ch/algotrader/strategy/box/BoxService.java
The strategy service class is responsible for sending events to the custom widget and for processing incoming events from the custom widget.

Strategy service classes can send events to the custom widget by using the JsonTemplate available inside the strategy service. The following code snippet will send a box event to the topic strategy.box.metrics:

getJsonTemplate().convertAndSend("strategy.box.metrics", box);

Strategy service class methods can be annotated with the JmsListener annotation in order to receive incoming events from the custom widget. The following code snippet will attach the terminateSeries method to the topic strategy.box.terminate:

@JmsListener(destination = "strategy.box.terminate")
public void terminateSeries() {
    ...
}

To see the full source code of above examples please see the corresponding source code of the Appendix B, Example Strategy "Box". Additional HTML5 custom widgets are available inside the example strategies Appendix D, Example Strategy "IPO" and Appendix C, Example Strategy "Pairs Trading".
10.2. AlgoTrader Eclipse IDE

The AlgoTrader Eclipse IDE provides a perspective (AlgoTrader perspective) which is ideal for quantitative / none technical users of AlgoTrader as it hides all code related artifacts. In addition the AlgoTrader Eclipse IDE contains the Strategy Wizard and the AlgoTrader Configuration Editor.

See section Section 2.1, “Development Environment Installation” for instructions on how to install the AlgoTrader Eclipse IDE.

10.2.1. AlgoTrader Perspective

Quantitative users can use this perspective to modify configurations of the system and trading strategies and start the system with different configurations. Code artifacts (java classes, config files etc.) are not visible in this perspective. However all log-files and reports are shown.

The perspective shows AlgoTrader projects only, i.e. projects that have the AlgoTrader nature (ch.algotrader.quant.ui.algotradernature) defined inside the .project file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
  <name>...</name>
  <comment></comment>
  <projects>
  </projects>
  <buildSpec>
    <buildCommand>
      <name>org.eclipse.jdt.core.javabuilder</name>
      <arguments>
      </arguments>
    </buildCommand>
    <buildCommand>
      <name>org.eclipse.m2e.core.maven2Builder</name>
      <arguments>
      </arguments>
    </buildCommand>
  </buildSpec>
  <natures>
    <nature>org.eclipse.jdt.core.javานature</nature>
    <nature>org.eclipse.m2e.core.maven2Nature</nature>
    <nature>ch.algotrader.quant.ui.algotradernature</nature>
  </natures>
</projectDescription>
```

It offers the following main capabilities:

- By right clicking on an AlgoTrader project the AlgoTrader Configuration Editor can be opened. This action is available under the AlgoTrader/Configuration menu item.
By right clicking on an AlgoTrader project an AlgoTrader project can be duplicated. This action is available under the AlgoTrader/Duplicate menu item.

By right clicking on an AlgoTrader a back test of the strategy can be started. This action can be performed by clicking the AlgoTrader/Run Strategy menu item. This context-menu item is only enabled for strategies that do not contain strategy groups.

**Note**
There is no new operation. Quantitative users of the system can duplicate projects but they cannot create new ones. Creation of new projects needs to be done by developers through the regular Java perspectives.

### 10.2.2. Strategy Wizard

The Strategy Wizard provides an easy way to automatically create all artifacts necessary for an AlgoTrader based trading strategy. Internally the Strategy Wizard makes use of the AlgoTrader Archetype, see Section 4.1.1, “AlgoTrader Strategy Wizard”.

### 10.2.3. AlgoTrader Configuration Editor

The AlgoTrader Configuration Editor provides an editor for AlgoTrader configuration files. For more information on the AlgoTrader configuration functionality please see: Chapter 26, Configuration and Preferences API and Section 4.5, “Strategy Groups”.

To open the AlgoTrader Configuration Editor, right click on an AlgoTrader project, then click the menu item AlgoTrader/Configuration. The AlgoTrader/Configuration menu will only be available if a file with the name applicationContext-client-xxx.xml exists in the project class path. If that file does not exist, the menu item will be disabled and the editor cannot be opened.

Furthermore, the AlgoTrader Configuration Editor behaves differently depending on the content of the applicationContext-client-xxx.xml file. If the configuration file contains strategy group definitions according to Section 4.5, “Strategy Groups” the editor will show three tabs.

To validate configuration file element types, the editor requires knowledge of the enumerations in the AlgoTrader common package.

If you see an error similar to the below, add the AlgoTrader common jar file to the class path of the project where you see the error.
10.2.3.1. Strategy Group Tab

The tab **Strategy Group** allows modification of strategy group definitions.

![Strategy Group Tab](image)

**Figure 10.30. AlgoTrader Configuration Editor**

The image above shows a strategy group configuration file opened in the AlgoTrader Configuration Editor.

On the left hand side of the Strategy Group Tab all available configurations are list and grouped by their Config Class (e.g., `Box` for `ch.algotrader.strategy.box.BoxConfig`).
The buttons "New" and "Edit" (1) can be used to add and modify individual configuration items.

![Strategy Groups](image)

**Figure 10.31. Strategy Data**

The "New" and "Edit" dialog contains the following elements (for details visit: Section 4.5, “Strategy Groups”):

- **Name**: the name of the .properties file
- **Config Class**: the name of the ConfigBean linked to the .properties file
- **Service Template**: The Spring Service Template to use
- **Engine Template**: The Spring Esper Template to use
- **Engine Name**: The name of the Esper Engine (not editable)

In addition it is possible to rename, duplicate and delete individual items (2). When selecting a configuration the corresponding .properties file will open in the lower part of the Strategy Group Tab (4).

On the right hand side of the Strategy Group Tab individual configurations can be added (5) to different strategy groups where they can be back tested in parallel. Items from the left hand side can be added to multiple groups on the right hand side. When adding an item to a strategy group an allocation weight has to be assigned. When adding multiple items to a strategy group the total weight of the strategy group must be 100%.
The right hand side of the Strategy Group Tab contains buttons to "Create", "Edit", "Rename" and "Delete" strategy groups (3). The button "Run" will start a backtest of the selected strategy group (see Section 10.2.3.4, "Starting Back Tests")

Both lists in the Strategy Group Tab have moving (6), collapsing/expanding (7), sorting (8) and full text filtering (9) of items. Also there is a function to filter by selected strategy item (10). In addition it is possible to change the orientation between horizontal and vertical (11).

10.2.3.2. Properties Tab

The tab Properties shows a list of all .properties file in the class path which can be edited here

![Properties](image)

**Figure 10.32. Configuration file without strategy groups**

When a property file is selected, the content of that file is shown in a table viewer. The table viewer has two columns: key and value.

The AlgoTrader Configuration Editor supports in-place editing of cells under value column.

In addition to the standard key=value pairs, the AlgoTrader Configuration Editor interprets special comments. These comments provide the AlgoTrader Configuration Editor with the information needed to display the key=value data (i.e. the type of data, the widget class it should use for an in-place editor etc.).

Example:

```json
#{"type":"String","label":"Last name:"
```
lastName = Mustermann

#{"type":"String","label":"First name:"}
surName = Joe

#{"type":"Date","required":"false","label":"Date of birth:"}
dateOfBirth = 1980-01-01

The AlgoTrader Configuration Editor remembers association of each key=value pair with it's special comment. When the editor saves properties back to the .properties, all key=value pairs are written with their special comments.

Each special comment is essentially a JSON object with three attributes: **type**, **required** and **label**. All three attributes are optional.

- **type** attribute is string and describes the data type of the key=value pair. It defaults to "String". Each data type is implicitly (via separate configuration) mapped to the widget class, which is used for in-place editing.
- **label** attribute is string and describes text label for representing the key=value pair in the table viewer. It defaults to key
- **required** attribute is boolean (true|false) and describes whether the key=value pair requires a non-empty value. It defaults to true.

The AlgoTrader Configuration Editor also supports separators/subtitles. To add a separator, a special comment needs to be defined in the Source tab as follows:

```json
#{"type":"String","label":"Strategy Name"}
strategyName=BOX

#{"type":"Separator"}

#{"type":"Integer","label":"Maximum Units"}
maxUnits=16

#{"type":"Integer","label":"Box Length (in Minutes)"}
boxLength=90

#{"type":"Separator", "label":"Time Settings"}

#{"type":"Integer","label":"Last Day of Trading (6=Friday)"}
endDay=6

#{"type":"Integer","label":"Latest Hour on last Day to enter a new position"}
endHour=16

#{"type":"Integer","label":"Hour on last Day to terminate the series"}
```
The content of this properties file will be rendered in the Properties tab as follows:

![Properties](image)

**Figure 10.33. Titles and Separators in a property file**

### 10.2.3.2.1. Out-of-the-box types

The AlgoTrader Configuration Editor supports the following data types and widget mappings out-of-the-box:

**Table 10.1. out-of-the-box types**

<table>
<thead>
<tr>
<th>Type</th>
<th>Widget</th>
<th>Style</th>
<th>RegEx Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>swt.Text</td>
<td>SWT.SINGLE</td>
<td>^d*$</td>
</tr>
<tr>
<td>Integer</td>
<td>swt.Text</td>
<td>SWT.SINGLE</td>
<td>^d*($\d*$)?$</td>
</tr>
<tr>
<td>Double</td>
<td>swt.Text</td>
<td>SWT.SINGLE</td>
<td>^d*($\d*$)?$</td>
</tr>
<tr>
<td>Time</td>
<td>nebula.CDateTime</td>
<td>CDT.DROP_DOWN</td>
<td>HH:mm:ss</td>
</tr>
<tr>
<td>Date</td>
<td>nebula.CDateTime</td>
<td>CDT.DROP_DOWN</td>
<td>yyyy-MM-dd</td>
</tr>
<tr>
<td>DateTime</td>
<td>nebula.CDateTime</td>
<td>CDT.DROP_DOWN</td>
<td>yyyy-MM-dd HH:mm:ss</td>
</tr>
<tr>
<td>Boolean</td>
<td>swt.Button</td>
<td>SWT.CHECK</td>
<td></td>
</tr>
<tr>
<td>Enumeration</td>
<td>swt.Combo</td>
<td>SWT.READ_ONLY</td>
<td></td>
</tr>
<tr>
<td>Email</td>
<td>swt.Text</td>
<td>SWT.SINGLE</td>
<td>^.+@.+.[a-z][2,4]$</td>
</tr>
<tr>
<td>URL</td>
<td>swt.Text</td>
<td>SWT.SINGLE</td>
<td>^https?&amp;#124;ftp&amp;#124;file://[^-a-zA-Z0-9+&amp;@#/%=<del>-_|:;]*[^-a-zA-Z0-9+&amp;@#/%=</del>-_|:;]*$</td>
</tr>
<tr>
<td>URI</td>
<td>swt.Text</td>
<td>SWT.SINGLE</td>
<td>^(([^?:#]+):)?(/([^?:#]<em>))</em>([^?:#]+)?([^?:#]+)?(^[#]+)?(^[#]+)?$</td>
</tr>
</tbody>
</table>
10.2.3.2.2. Enum types support

The AlgoTrader Configuration Editor supports editing of properties of enum types.

Let's suppose, the program contains Enum type definition:

```java
package ch.algotrader;

public enum Color {
    RED, GREEN, BLUE
}
```

A new property type for this enum type can be defined as follows:

```xml
<extension point="ch.algotrader.config-editor.PropertyDef">
    <PropertyDef id="Color"
        dataType="/somepackage.Color"
        cellEditorFactory="ch.algotrader.configeditor.editingsupport.EnumCellEditorFactory">
    </PropertyDef>
</extension>
```

When such a property type is defined in "plugin.xml" of some plugin and when this plugin is packaged/installed together with the program, the program can read, edit and save new property type in `.properties` file:

```properties
#{"type":"Color"}
backgroundColor=BLUE
```

Note the plugin containing PropertyDef extension must have the following line in "MANIFEST.MF":

```plaintext
Require-Bundle: ch.algotrader.config-editor
```

otherwise the AlgoTrader Config Editor will not be able to access the class designated by `dataType` attribute and will throw `ClassNotFoundException`.

10.2.3.2.3. Input validation

The AlgoTrader Configuration Editor implements RegEx based input validation (one expression per data type). When user input does not conform to the specified regular expression, the AlgoTrader Configuration Editor shows explanatory error message and does not allow to save the file(s).
10.2.3.4. Property definitions

The AlgoTrader Configuration Editor provides an extension point, which allows the definition of new property types:

```xml
<extension point="ch.algotrader.config-editor.PropertyDef">
  <PropertyDef id="Email"
    dataType="java.lang.String"
    regex="^[^\s].*\.(\w+)$"
    regexErrorMessage="The input {0} is not a valid e-mail"
    cellEditorFactory="ch.algotrader.configeditor.editingsupport.TextCellEditorFactory">
  </PropertyDef>
</extension>
```

Detailed explanation of PropertyDef extension point attributes:

- **id**: string, required. Property type identifier, must match the attribute type in property comment.
- **dataType**: fully qualified java class name, required. Internal type of de serialized property or implementation of ch.algotrader.configeditor.IPropertySerializer interface.
- **regex**: string, optional. Regular expression for validating user input.
- **regexErrorMessage**: string, optional. Error message shown to the user when input does not validate against the specified Regex. When omitted, the default message is shown: "User input {0} does not satisfy pattern {1}" where {0} is a placeholder for user input and {1} is a placeholder for the regular expression.
- **cellEditorFactory**: fully qualified java class name, required. Factory creating cell editor, must implement ch.algotrader.configeditor.CellEditorFactory interface.

10.2.3.3. Source Tab

Furthermore, the AlgoTrader Configuration Editor provides a Source tab, which shows the content of the file applicationContext-client-xxx.xml.

10.2.3.4. Starting Back Tests

As described in the Section 10.2.1, “AlgoTrader Perspective” section, it is possible to start a back test of a strategy by right clicking on the strategy and selecting AlgoTrader/Run Strategy (this option is only available for strategies that are not using strategy groups).

Back Tests can also be started from within the AlgoTrader Configuration Editor:

- For strategies without strategy group definitions there is only one possible configuration that can be back tested. To start a back test please click the "Run" button at the top of the Properties tab.
• For strategies with strategy group definitions multiple configurations exist which can be back tested separately. To start a back test please select the desired strategy group on the right hand side of the Strategy Group Tab and click the "Run" button.

10.2.4. Esper Colorer

AlgoTrader provides a custom Esper EPL Syntax Highlighter based on the Colorer\(^8\) Library.

Together with the Eclipse Colorer Plugin\(^9\) it provides the following features:

• Automatic Code Outlining
• Pairs/Brace Matching
• Automatic Code Folding
• Present different colors for:
  • Reserved Keywords
  • Symbols
  • Comments
  • Literals
  • Numbers

![Figure 10.34. Syntax Highlighter](image)

**Note**
The Syntax Highlighter does not provide Code Completion or Syntax Checking!

---

\(^8\) [http://colorer.sourceforge.net/](http://colorer.sourceforge.net/)

10.3. Reference Data Manager

The AlgoTrader Reference Data Manager (RDM) is a web based tool accessible via the AlgoTrader UI. It provides a way to manually edit reference data related database tables with user friendly interface and without the need to restart AlgoTrader after making changes. The list of editable tables is as follows:

- Account
- Component
- Exchange
- Holiday
- Measurement
- OrderPreference
- Strategy
- SecurityFamily
- SecurityReference
- Security

The Reference Data Manager also allows running the reference data download from the user interface without having to run ReferenceDataStarter separately.

In order to use RDM you need to start AlgoTrader with html5 Spring profile enabled. You can then access RDM through the top right menu of the AlgoTrader client, menu item Reference data.

If you start AlgoTrader with a reference data Spring profile enabled (see Chapter 19, Reference Data), RDM will allow you to run the reference data download from its interface. After running it, the new securities, security families, etc. will be immediately available to use with AlgoTrader without a restart. The following image shows an example where the reference data download was run with AlgoTrader set up with Bitfinex exchange reference data Spring profile.

The new securities are visible in the Security tab in RDM.
The new securities are available in the AlgoTrader UI.

It is possible to enable a FIX adapter at run time, by activating the relevant Account table entry via RDM.
10.4. Historical Data Manager

The Historical Data Manager (HDM) is a web based tool accessible via the AlgoTrader UI. It provides a way to download, view and edit historical data (see Chapter 18, Historical Data) stored in InfluxDB database AlgoTrader is configured with. It also has a functionality for exporting and importing the historical data from/to InfluxDB in different CSV file formats. In order to use Historical Data Manager, AlgoTrader needs to be started with html5, influxDB Spring profiles activated and a historical data adapter profile (e.g. iBHistoricalData for Interactive Brokers adapter).

You can access the HDM through the top right menu of the AlgoTrader client.

In the following screen shot, the view of 1 minute AMZN stock historical data is selected. The data table can be ordered by dateTime in ascending or descending order. After selecting an instrument in the tree menu on the left side of the screen, all instrument's data with the relevant parameters (i.e. adapter type and bar size) is displayed.
The date and time range of displayed data is visible in the Min. date and Max. date fields at the top of the screen. The date and time range of displayed data may be changed by editing the two fields and clicking the Set range button. Note also the delete all button at the top, it removes all data that is currently visible.

After clicking on a row in the historical data table, a window is opened from where an individual measurement may be edited or deleted.
Clicking the Add new bar button opens the window for entering a new measurement.

**Note**

The HDM only displays dates of measurements down to seconds, even if they have smaller than a second fractions defined in the InfluxDB store. InfluxDB allows storing measurements with
precision down to nanosecond. Adding a new measurement with a date that is exactly same as an existing measurement (same in nanoseconds), will effectively overwrite the existing one.

The Export into CSV button lets the user export the data currently displayed in the historical data table in CSV file format. The HDM supports several different CSV file formats.
This is an example of a CSV file exported in Excel CSV Type format.

```
DateTime,open,high,low,close,vol
1542633300000,5222.12,5222.1,5200,5200,2.345
1542634200000,5222,5222.1,5200,5200,2.345
1542634200000,5200,5200,5200,5200,1
1542635100000,5110.9,5171.2,5110.9,5129.4,10.441372778
1542635100000,5110.9,5171.2,5110.9,5129.4,10.441372778
1543396500000,3990.8,3983.7,3983.7,0.47
1543396500000,3990.8,3983.7,3983.7,0.47
1543401900000,4018.87,4020.8,4010.87,4020.8,0.03
1543401900000,4018.87,4020.8,4010.87,4020.8,0.03
1543404600000,4007.9,4014.9,4007.9,4014.9,0.38
1543405500000,4013.4,4018.9,4010.67,4018.9,0.5
1543406400000,4008.2,4008.2,4000.2,4000.2,0.41
1543406400000,4008.2,4008.2,4000.2,4000.2,0.41
```

When selecting Historical Data Import in the tree menu on the left, the following screen is displayed. It provides a facility to import CSV files in several formats. Imported data is stored in InfluxDB.

Note that importing consists of 2 phases, the file upload phase where the uploaded file is stored in a temporary file in the operating system where AlgoTrader server runs, and the file processing phase which converts and stores the data in InfluxDB.
When selecting **Historical Data Download** in the tree menu on the left, the following screen is displayed. It provides a facility to download data via the historical data adapter AlgoTrader is currently configured with. Downloaded data is directly stored in InfluxDB. This functionality is only available in HDM when a historical data adapter is configured. The functionality is identical to running a Historical Data Starter (see *Section 18.3, “Historical Data Download”*). After finishing the historical data download, the last few lines of the AlgoTrader server log is displayed.
**Note**

An individual historical data download might take a considerable amount of time depending on the date range and granularity (e.g. 1 year worth of 1 second bars).
Performance Measurement

AlgoTrader provides a sophisticated Performance Measurement functionality that consists of the following components:

- Portfolio Value logging to the database in Live-Trading Mode
- Portfolio Value logging to a .csv file in Simulation Mode
- Back test report and performance statistics display at the end of a simulation run (see Section 5.5, "Performance Statistics")
- Portfolio Value Restoration Feature

All Performance Measurement features depend mainly on the Entity Portfolio Value which has the following attributes

- dateTime
- netLiqValue
- marketValue
- realizedPL
- unrealizedPL
- cashBalance
- openPositions
- leverage
- cashFlow (optional)

11.1. Portfolio Value Logging

In Live-Trading Mode Portfolio Values are recorded to the database for all running strategies on an hourly basis. In addition Portfolio Values are recorded every time a transaction occurs that influences performance. For the AlgoTrader Server these are Credits and Debits and for strategies, these are Rebalances. The corresponding value of the transaction is recorded in the optional attribute cashFlow of the Portfolio Value.

In Simulation Mode Portfolio Values are recorded to the file PortfolioReport.csv through the PortfolioReport class on a daily-basis using the AlgoTrader Reporting Functionality, see: Chapter 30, Reporting.

11.2. Portfolio Value Restoration Feature

In case a transaction that influences performance needs to be recorded for a prior period in time, all Portfolio Values since that time period are invalid and need to be restored.
Through the class `RestorePortfolioValueStarter` which uses `PortfolioPersistenceService.restorePortfolioValues()` Portfolio Values can be restored for the specified strategy and time period. For the restoration of each Portfolio Value all corresponding transactions up to that time have to be evaluated.

**Note**

The Portfolio Value restoration can take a considerable amount of time to complete.
Risk Management

Using the provided functionality of the system, the following risk metrics can be enforced by AlgoTrader based strategies:

- Fixed and/or Trailing Stop-Loss Limit for each Position
- Minimum Cash-Balance
- Maximum Loss per Position
- Maximum Leverage of the Portfolio
- Maximum Monthly Draw-Down
- Maximum Market Exposure
- etc.

Note

Enforcing these rules requires coding and is not available through configuration only.

12.1. Pre-Trade Checks

AlgoTrader offers a mechanism to inject arbitrary pre-trade checks. These pre-trade checks can for example reject orders in case of violations of:

- Maximum order quantity
- Maximum orders per time period (e.g. per day)
- etc.

To implement pre-trade checks the following steps are needed:

1. Create a client-specific base project which has a maven dependency to the algotrader-core module. This Project will be used to start the application.

2. Inside this base project provide as many implementations of the OrderValidator class as needed.

3. Provide a custom wiring-class inside the base-project that is initializing these OrderValidators.

```java
public interface OrderValidator {
    void validateOrder(Order order) throws OrderValidationException;
}
When AlgoTrader is started, all OrderValidator implementations are injected into the OrderService. Every time an Order is sent or modified (not on cancel) and one of the OrderValidators throws an exception, the Order will be rejected and will not go out to the broker or exchange.
Forex Handling

The System provides full Forex and Currency Exchange Management. FX Rates can be retrieved in real-time. All portfolio figures are calculated based on up-to-date FX-Rates.

FX conversion is provided through the MarketDataService. When subscribing for a non-base currency instrument the system will automatically subscribe the Forex instrument necessary to convert the non-base currency balance (e.g. realized PnL and market value) into the base currency. In some cases multiple Forex / Currency Pairs (based on the same base and transaction currency) are available in the system which are traded through different brokers / exchanges.

For those cases a special setting misc.defaultMarketAdapters defines the priority in which market adapters and exchanges are used. The default value is the following:

```
misc.defaultMarketAdapters =
  BITMEX:CNP,OKEX:CNP,BINANCE:CNP,HUOBI:CNP,BITFINEX:CNP,BITHUMB:CNP,HITBTC:CNP,KRAKEN:CNP,
```

This is a comma separated list of preferred <exchange>:<adapterType>. Exchange code is taken from the database security.exchange_code value and adapter type is ch.algotrader.enumeration.AdapterType. Optional specific security pairs are also supported, using '@' symbol, e.g. 'BTCUSD@BITF:CNG'. When the platform needs to make a conversion for a given forex, it will iterate over this list and determine the first match. Match means all of the below:

- if currency pair is specified, it should match the symbol of the Forex
- This currency pair is traded on given exchange
- Current adapter type is active, e.g. it's Spring profile is enabled (e.g. if cNGMarketData is active then 'CNG' will be matched for Coinigy securities)

To change the default settings the following property needs to be updated inside conf.properties. Alternatively the properties can be changed via Section 2.3, “VM Arguments”

```
# default exchanges/adapter types to use for market data.
SYMBOL@EXCHANGE_CODE:ADAPTER_TYPE
-Dmisc.defaultMarketAdapters=IDEALPRO:IB
```

For example, -Dmisc.defaultMarketAdapters=BTCUSD@BITF:CNG,BITF:BFX means that for BTCUSD conversions Bitmex via Coinigy will be used, for all others direct Bitfinex connection will be used (assuming both Coinigy and Bitfinex adapters are running).

13.1. Currency Handling

In most cases securities are attributed in their currency (as defined by Security). Their market value is attributed towards Market Value.
There are however the following exceptions.

13.1.1. Futures

- Futures are fully margined, that's why buying a Future does not actually influence cash but only the margin requirement.
- AlgoTrader however treats Futures as regular Securities (i.e. add Future Market Value to System Market Value and deduct Transaction Price from cash)

**Note**
InteractiveBrokers handles Forex slightly different than AlgoTrader:

- IB displays Future Unrealized P/L under Cash.
- To compare AlgoTrader Balances to IB Balances (if there are Future Positions), one has to compare the Net Liquidation Value and not Cash / Market Value individually

13.1.2. Forex

- Forex (e.g. EUR.USD) consists of the Base Currency (e.g. EUR) and Transaction Currency (e.g. USD)
- In Balances Forex are attributed towards Cash (and not system Market Value) in the Base Currency
- The gross value of a transaction is booked in the Transaction Currency, whereas the commission is booked in the Base Currency.

**Note**
The TWS Trades Window displays commissions in Trade Currency, but IB Flex Reports displays commissions correctly in the Base Currency.

13.1.3. Currency Attribution

The following table describes Currency Attribution of Positions. The logic is implemented by `Position.getAttribution()`

**Table 13.1. Position Currency Attribution**

<table>
<thead>
<tr>
<th></th>
<th>General Security</th>
<th>Forex</th>
<th>Future on Forex</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributed to</td>
<td>Market Value</td>
<td>Cash</td>
<td>Market Value</td>
</tr>
</tbody>
</table>

168
The following table describes Currency Attribution of Transactions. The logic is implemented by Transaction.getAttributions().

**Table 13.2. Transaction Currency Attribution**

<table>
<thead>
<tr>
<th>General Security</th>
<th>Forex</th>
<th>Future on Forex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currency</td>
<td>Currency of Security</td>
<td>Based Currency of Forex</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Based Forex</td>
</tr>
<tr>
<td>Amount</td>
<td>quantity * contract size * price</td>
<td>quantity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>quantity * contract size</td>
</tr>
</tbody>
</table>

13.2. Forex-Hedging

The system provides automatic Forex-Hedging by the Service ch.algotrader.service.ForexService. This service will maintain multiple FX Positions to hedge all non base currency balances. For actual Forex-Hedging the following two options exist.

13.2.1. Exchange vs. Margin Trading

FX brokers and crypto exchanges support exchange trading and/or margin trading.

Exchange trading allows you to exchange/convert one crypto/fiat amount into another (e.g. converting USD to BTC). Using exchange trading short trades are not possible.

Margin trading allows you to have long and short positions on any pair (e.g. long BTC/USD or short ETH/BTC).

The SimpleOrder property exchangeOrder (which is false per default) drives which account you are trading against:

- exchangeOrder = true means use the exchange account
- exchangeOrder = false means use the margin account

Note

Note that every broker / exchange differs on how to submit exchange account vs. margin account orders.

Some require you to use different securities (e.g. Bitflyer), others different order types or price types.
Margin positions can be useful also for Hedging purposes. Depending on the current exposure of a non-base currency equity portfolio, an offsetting margin position can be maintained to result in a zero non-base currency exposure.

13.2.2. FX Future

The second option for FX Hedging is by means of FX Futures which is a subclass of Future.

For some FX security families there are multiple entries available with different contract sizes (e.g. 12'500, 62'500 and 125'000 for EUR.USD). Because of this the hedgingFamily has to be defined as a Property on the EUR.USD subscription.

Since FX Futures expire, the Hedging Position has to be rolled before Expiration, which is also done automatically based on configured parameters.
Chapter 14.

Options & Futures

14.1. Expiration

Both Options and Futures implement the interface `ExpirableI` which has a property `expiration` that represents the expiration date.

In addition to the `expiration` date (sometimes also called `lastTradingDay`) Futures also have the following two fields:

- `firstNoticeDay`: this is the day on which the buyer of a futures contract can be called upon by the exchange to take delivery
- `maturityMonthYear`: The contract month & year of a future

**Note**

`expiration` and `maturityMonthYear` do not necessarily need to be within the same month. e.g. `maturityMonthYear: 2015-12, expiration: 2015-11-30`

The following table contains references on how these three fields are used by different market data providers

<table>
<thead>
<tr>
<th>Field</th>
<th>InteractiveBrokers</th>
<th>Bloomberg</th>
</tr>
</thead>
<tbody>
<tr>
<td>maturityMonthYear</td>
<td>m_maturityMonthYear</td>
<td>FUT_MONTH_YR</td>
</tr>
<tr>
<td>firstNoticeDay</td>
<td>not available</td>
<td>FUT_NOTICE_FIRST</td>
</tr>
<tr>
<td>expiration</td>
<td>m_expiry</td>
<td>LAST_TRADEABLE_DT</td>
</tr>
</tbody>
</table>

For an detailed explanation between `lastTradingDay` and `firstNoticeDay` please visit this page on futures expiration

**Note**

IB also shows an expiration date in TWS which is either the same day or the next day after the last trading day

14.2. Leverage & Exposure

AlgoTrader uses the following definitions for Leverage & Exposure:

---

14.3. Symbol, ISIN & RIC

The two classes `FutureSymbol` and `OptionSymbol` provide static methods for generating the Symbol, ISIN and RIC for Futures and Options. Both Symbol and ISIN use the property `symbolRoot` of the `SecurityFamily`. The RIC on the other side uses the property `ricRoot`.

14.4. Delta Hedging

Automatic Delta Hedging is provided by the Service `ch.algotrader.service.OptionService`. The method `hedgeDelta` will calculate the delta adjusted market value for all securities based on one particular underlying. The method will subsequently build up a hedging position based on the nearest Future of the same underlying.
Since some underlyings have more than one Futures Chain based on them, the property `hedgingFamilyNeeds` needs to be defined on the Subscription of the underlying. In addition the method `hedgeDelta` can also be invoked with the `ServerManagementService`.

### 14.5. Option & Future Chain Download

The `ReferenceDataService` can be used to download all currently traded and past Options and/or Futures into the database, see *Chapter 19, Reference Data*.

### 14.6. Option Greeks

The following Option Greeks are available through the Class `OptionUtil`:

- option price (through Black-and-Scholes)
- implied volatility (through Black-and-Scholes, Newton Rapson Method and SABR Surface)
- intrinsic value
- delta
- vega
- theta
- forward price
- moneyness
- strike by delta

### 14.7. Option Pricing Engine

The system provides a sophisticated option pricing engine which is developed around the SABR volatility Model.

Based on historical Volatility at different Moneyness levels (e.g. ATM, ATM +10%, ATM +20%, ATM -10% & ATM-20%) or Delta levels (e.g. 50%, 35%, 75%) volatility parameters are calculated (=calibration) and used for option pricing.

#### 14.7.1. SABR Calibration

The `OptionService` is responsible for SABR calibration. The calibration process happens for one specific expiration and takes an array of strikes with their corresponding array of volatilities. the calibration process returns a `SABRSmile ValueObject`, which basically contains the three parameters `rho`, `volVol` and `alpha` (in addition to the time-to-expiration and at-the-money volatility). The actual calibration happens through the class `SABR`.

SABR Calibration can be done either by actual Option Prices or directly by the Implied Volatility. Also there are methods to do a SABR Calibration just for one expiration (returning one SABR Smile Value Object) or
for an entire Volatility Surface (returning a SABRSurface ValueObject which consist of multiple SABRSmile ValueObjects)

The SABR Calibration depends on different Implied Volatilities (a subclass of Security) being defined in the database. A Implied Volatility needs to define either a moneyness or a delta (in addition to Duration and Option Type).

### 14.7.2. Option Pricing

Based on the SABR Calibration the actual option pricing takes place. This is handled through the class OptionUtil with the method getImpliedVolatilitySABR. This method takes SABRSurface parameter. The actual option pricing happens in two steps:

1. For all available expirations a volatility is calculated for the requested strike
2. Using spline interpolation the volatility for the requested expiration is calculated

### 14.7.3. References

- *Hedging under SABR Model, Refined risk management under the SABR model*[^2]
- *Extensions of the SABR model for equity options*[^3]

### 14.8. OTC Options

Since OTC Options do not have a predefined chain definition, the OptionService contains a method createOTCOption to create an OTC option based on the specified expirationDate, strike and type.

Broker/Exchange Interfaces

The System provides generic interface functions to connect AlgoTrader to different brokers / exchanges.

The following broker / exchange specific interfaces are currently available:

• Fix Interfaces
  • Currenex
  • DukasCopy
  • Exante
  • EzeSoft/RealTick
  • Fortex
  • FXCM
  • InteractiveBrokers
  • JP Morgan
  • LMAX
  • Nexus Prime
  • PrimeXM
  • SocGen
  • Trading Technologies (TT)
  • UBS
  • Bitstamp
  • Coinbase Pro

• Native Interfaces
  • Bloomberg
  • Interactive Brokers

• Crypto REST/WebSocket Adapters
  • Binance
  • Bitfinex
• Bitflyer
• BitMEX
• Coinigy

AlgoTrader uses **QuickFix/J**\(^1\) for all fix interfaces.

For further details on Broker/Exchange adapters please see *Chapter 22, Adapters*

---

\(^1\) https://www.quickfixj.org/
Order Management

16.1. Order Validation

Before sending an Order, it is advised to call the `validate` method on the order. This will validate the order regarding limits, amount, quantity, etc. In case validation fails an Exception will be thrown and the order can be modified.

**Note**

The `validate` method will be called (again) inside the `sendOrder` method, in case the validation fails an Exception will be thrown.

In addition to the validate method all configured Section 12.1, “Pre-Trade Checks” will be invoked. In case any pre-trade check is breached the order will be rejected.

16.2. Place Order

The method `sendOrder` of the `OrderService` is responsible for placing Orders. This method takes an Order Entity or Order Value Object as parameter.

Sending an order using an Order Entity looks like this:

```java
MarketOrder order = MarketOrder.Factory.newInstance();
order.setStrategy(strategy);
order.setAccount(account);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
getOrderService().sendOrder(order);
```

The associated Entities (i.e. strategy, account and security) can be retrieved via the `LookupService`.

Sending an order using Order Value Objects looks like this:

```java
MarketOrderVO order = MarketOrderVOBuilder.create()
    .setStrategyId(strategyId)
    .setAccountId(accountId)
    .setSecurityId(securityId)
    .setQuantity(orderQuantity)
    .setSide(Side.BUY)
```
Sending of AlgoTrader orders is currently only supported via Order Entities. Creating and sending an AlgoOrder looks like this:

```java
SlicingOrder order = new SlicingOrder();
order.setStrategy(strategy);
order.setAccount(account);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
order.setMinQuantity(BigDecimal.valueOf(10));
order.setMaxQuantity(BigDecimal.valueOf(100));
order.setMinVolPct(0.01);
order.setMaxVolPct(0.1);
order.setMinDuration(1);
order.setMaxDuration(5);
order.setMinDelay(1);
order.setMaxDelay(5);
getOrderService().sendOrder(order);
```

The broker / exchange specific SimpleOrderExecService will create the broker / exchange specific order, assign an intId if none has been assigned yet and send the order to the broker / exchange.

After sending the Order to the broker / exchange, the order object is propagated to the AlgoTrader Server Esper service instance (running inside the AlgoTrader Server) as well as to the Esper service instance of the corresponding strategy (where potential actions like cancel order or modify order can be executed).

Open orders are kept in an internal order book until their full execution or cancellation. Completed orders remain in the book and are accessible through OrderService until evicted. AlgoTrader evicts completed orders at 24:00 local time daily by default. One can also manually evict orders by calling `OrderService#evictExecutedOrders()` method.

The actual exchange an Order is sent to will be retrieved from the associated Security/SecurityFamily. Alternatively it is possible to assign an Exchange to an Order Entity or Order Value Object directly.

### 16.2.1. Order Preferences

As AlgoOrders typically have a lot of parameters (e.g. minQuantity, maxQuantity for the SlicingOrder) it is possible to save a set of settings using the OrderPreference Entity. The following OrderPreference SLICING is contained within the db-samples (`/algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql`):
The `OrderPreference` SLICING defines default settings for the Slicing Order. Through the column `DEFAULT_ACCOUNT_FK` it is also possible to define a default account for the `AlgoOrder`. With this information available in the database an order can now be created as follows:

```java
Order order = getOrderService().createOrderByOrderPreference("SLICING");
order.setStrategy(strategy);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
getOrderService().sendOrder(order);
```

### 16.2.2. Trade Suggestions

In addition to the `OrderService#sendOrder` method there is a `OrderService#suggestOrder` method, which will not send out an order to the broker / exchange but instead create an Email with a Trade Suggestion sent to the registered Email addresses. This allows for manual confirmation of automatically generated Signals.

### 16.2.3. Order Properties

Even though the Order Entity already contains commonly used properties like side, quantity, time-in-force, etc. it is possible to attach additional arbitrary properties to an Order. Order properties have a name, a value and a type.

Order properties of type `INTERNAL` are kept inside the system and are not propagated to external brokers.

If the type of an Order property is `FIX` it is assigned to an outgoing Fix order as an additional fix tag. It is therefore mandatory that the name of the order property is a number (representing the fix tag).

If the type of an Order property is `IB` the system will try to find an IB order field corresponding to the `IB order property name`

An internal order property can be attached to an order as follows:

```java
order.addOrderProperty("test", "XYZ", OrderPropertyType.INTERNAL);
```

1. http://interactivebrokers.github.io/tws-api/classIBApi_1_1Order.html
2. http://interactivebrokers.github.io/tws-api/classIBApi_1_1Order.html
An Fix order property can be attached to an order as follows:

```java
order.addOrderProperty("4000", "XYZ", OrderPropertyType.FIX);
```

An IB order property can be attached to an order as follows:

```java
order.addOrderProperty("faGroup", "group1", OrderPropertyType.IB);
```

Any object can be attached to an order as a property value using `addProperty(String name, Object value)` method, it can be done as follows:

```java
order.addProperty("myProperty", "propertyValue");
```

Order Property can be removed from order as follows:

```java
order.removeProperty("name");
```

The method `getOrderProperties` cannot be used to manage order properties it's returning unmodifiable map of `OrderProperty` objects excluding all other properties provided to order, to remove or update properties it's needed to use methods above.

**Note**

`OrderProperties` are only supported on Order Entities but not on Order value objects (VOs).

### 16.3. Order Status

The current status of an order is represented by one or many `OrderStatus` events. Whenever order status events are received back from the broker/exchange, `OrderStatus` events are created and propagated to the AlgoTrader Server Esper service Instance and subsequently to the strategy that initiated the order.

Permitted Order Status transitions are depicted by the following state machine diagram.
If an order does not receive either an Acknowledgment or Fill within a configurable time period (default: 1 sec) after sending the Order, an Exception is thrown, as there might be a problem with the broker / exchange connection. This is enforced by the notifyMissingOrderReply statement which can be turned off by changing the following property inside conf-core.properties. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```
# notify in case there is no reply on an order
statement.notifyMissingOrderReply = false
```

Once all fills corresponding to an Order are fully persisted an OrderCompletionVO event is generated and propagated to the Strategy. By the time an OrderStatus event is received by the strategy, corresponding database activity might not have been completed yet. However by the time an OrderCompletionVO event is received, it is guaranteed that all database activity has been completed. It is therefore safe to invoke any sort of the database lookup at this time. For further details see Section 8.4.8, “Callbacks”.

### 16.4. Receive Fills

Whenever fills are received back from the broker / exchange, Fill events are created and propagated to the AlgoTrader Server Esper service Instance and subsequently to the strategy that initiated the order.
The Fill events trigger the creation of a Transaction object (a persistent Record in the database). In addition the Fill and corresponding Transactions are also propagated to the strategy, where actions can be taken upon.

**Note**

Fills and Transactions are separated from each other for the following reason. A Fill contains all the information received from the broker / exchange (and a reference to the Order). Whereas a Transaction contains all the information related to accounting (i.e. references to position and strategy). In addition to Transactions related to Fills, there are Transactions that are independent of Fills (i.e. Deposits, Withdrawals, Interest, etc.).

### 16.5. Handling of Fees and Commissions

AlgoTrader supports handling of the following three types of fees and commissions:

- Exchange Fees
- Execution Commissions (typically charged by executing brokers)
- Clearing Commissions (typically charged by clearing brokers)

All three types are available in the database table `transaction`.

Most adapter for traditional asset classes (e.g. equities, forex and derivatives) do not provide fee information on execution messages. Some crypto exchanges provide fee information. If the fees are in the currency of the transaction, they are stored in the `fee` attribute of the transaction. In case the fees are charged in another currency (for example Binance charges in its own currency - BNB), a new transaction is created with `transactionType = FEES`, `quantity = -1` for fees paid (and `quantity = -1` for fees received, e.g. maker rebates), `price = fee value` and `currency = fee currency`.

For the other adapters, AlgoTrader uses the internal Execution Model (see Section 5.1, "Exchange Simulator") to assign fees and commissions based on configuration (e.g. commission-per-contract).

### 16.6. Examples of Orders and Executions

The following sections show a few example orders and their corresponding executions/fills. All examples start from a "clean" database with empty `transaction`, `cash_balance` and `position` tables.

#### 16.6.1. Margin Order with Fee in Transaction Currency

Margin Order: BUY 0.002 BTC/USD @ Bitfinex

Execution: 0.002 @ 4363.20, gross value: 8.7264, Fees: 0.01745280 USD

This will result in the following values:

Transactions:
Exchange Order with Fee in Base Currency

Exchange Order: BUY 0.003 BTC/USD @ Bitfinex

Execution: 0.003 @ 4374.23439227, gross value: 13.12270318, Fees: 0.000006 BTC

This will result in the following values:

Transactions:

<table>
<thead>
<tr>
<th>PRICE</th>
<th>QUANTITY</th>
<th>FEE</th>
<th>CURRENCY</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.12270318</td>
<td>-1</td>
<td></td>
<td>USD</td>
<td>EXCHANGE_DEBIT</td>
</tr>
<tr>
<td>0.003</td>
<td>1</td>
<td>0.000006</td>
<td>BTC</td>
<td>EXCHANGE_CREDIT</td>
</tr>
</tbody>
</table>

Cash Balances:

<table>
<thead>
<tr>
<th>CURRENCY</th>
<th>AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>USD</td>
<td>-13.12270318</td>
</tr>
<tr>
<td>BTC</td>
<td>0.002994</td>
</tr>
</tbody>
</table>

No positions as this is an exchange order

Exchange Order with Fee in Alternate Currency

Exchange Order: BUY 0.003 BTC/USDT @ Binance

Execution: 0.003 @ 4395.51, gross value: 13.18653, Fees: 0.001796 BNB

This will result in the following values:

Transactions:

<table>
<thead>
<tr>
<th>QUANTITY</th>
<th>PRICE</th>
<th>CURRENCY</th>
<th>TYPE</th>
</tr>
</thead>
</table>

183
Internal Order Id Format

intId is the internally assigned order identifier whereas extId is the id assigned by the external broker / exchange.

In general the internal order identifier has the following format:

`<session_qualifier><id>.<version>`

Example: `lmax1.1`

- `session_qualifier`: each Fix session has a unique session qualifier
- `id`: an integer which is auto-incremented per session. For Fix, the last id is retrieved from the order table during start up
- `version`: The number of modifications that took place on the Order, starting with 0 when the order is first submitted.

By default AlgoTrader automatically assigns an IntId value to all outgoing orders. Open and executed orders can be identified and looked up by their IntId.

Especially when using a Section 8.4.8.2, “Trade callback” it is necessary to generate and assign an IntId value to the order prior to submitting it to the order service. The OrderService#getNextOrderId() method can be used to generate a unique IntId value per session associated with an Account record.

```java
String orderId = getOrderService().getNextOrderId(order.getClass(), accountId);
order.setIntId(orderId);
engine.addTradeCallback(Collections.singleton(orderId), (strategyName, orderStati) -> {
    ...
```
getOrderService().sendOrders(orders);

**Note**

Please note that care must be taken when using OrderService#getNextOrderId() with the IB order service. The IB native interface expects orders to be sent with their order ids in ascending order. The Class IBOrderIdSynchronizer is responsible to make sure order ids are actually in ascending order. In case an order id is skipped the IBOrderIdSynchronizer will wait for up to maxOrderSyncTime milliseconds for the order with the correct order id to arrive.

16.8. Symbology

In the electronic trading domain there are different ways to identify a security, some of which are:

- Symbol
- Bloomberg ID
- For options: underlying, expiration, strike & type
- etc.

Different Brokers employ different types of Symbology to identify a security. For this purpose AlgoTrader provides the notion of SymbologyResolver which is responsible for assigning appropriate information to outgoing broker / exchange communication. These SymbologyResolvers can be extended on a per adapter basis.
Market Data

AlgoTrader provides Market Data Interfaces with the following market data providers:

- Fix Interfaces
  - Currenex
  - DukasCopy
  - Exante
  - Fortex
  - FXCM
  - Interactive Brokers
  - LMAX
  - Nexus Prime
  - PrimeXM
  - Trading Technologies (TT)
  - Bitstamp
- Native Interfaces
  - Bloomberg
  - Interactive Brokers
  - QuantHouse
- Crypto REST/WebSocket Interfaces
  - Binance
  - Bitfinex
  - Bitflyer
  - BitMEX
  - CoinAPI
  - Coinbase Pro
  - Coinigy
AlgoTrader allows having multiple market data interfaces active at the same time so market data received from different market data providers can be compared in real-time.

To enable either of those Market Data Interfaces the following two steps have to be executed:

1. The correct Spring Profile has to be activated according to Section 25.1, “Starter Classes”

2. For Bloomberg market-data-bb and for InteractiveBrokers market-data-ib has to be added to the VM argument server-engine.init when running the AlgoTrader server.

Market Data Events itself are available in different types:

- **BarVO** Open-High-Low-Close Price Bars, also containing volumes and volume weighted average prices
- **TickVO**: Snapshot of the market at a particular point in time, containing information like last price, last time, bid, ask, volume, etc.
- **QuoteVO**: Its subclasses represent the current best bid and offer **BidVO** and **AskVO**
- **TradeVO**: An actual order that was executed on the market, containing information like last price, last size and volume
- **GenericTickVO**: Represents additional price information made available by market data providers (e.g. open price, close price, vwap price)

As the following diagram shows, market data providers deliver Price Events (**TradeVO**, **BidVO** & **AskVO**) or individual **TickVO** Fields.

In back testing AlgoTrader supports both Ticks or Bars. In both live trading and back testing Price events and Tick events can be aggregated into Bar events

![Figure 17.1. Market Data Event Types](image-url)
Inside each strategy the `MarketDataCacheService` keeps a copy of the last `MarketDataEvent` for each Security. For further details see Section 7.2.16, "Market Data Cache Service".

### 17.1. Creation of Bars based on Ticks

Through the live data recording feature (see Section 18.2, "Live Data Recording") tick data can automatically be aggregated into bar-data.

In addition, it is possible to aggregate ticks into bars through Esper code in both Simulation and Live Trading by using an Esper `time_batch` window:

```sql
select
    first(last) as open,
    max(last) as high,
    min(last) as low,
    last(last) as close
from
    TickVO.win:time_batch(1 min)
group by
    securityId;
```

and `expr_batch` for constant volume bars:

```sql
select
    max(last) as high,
    min(last) as low,
    first(last) as open,
    last(last) as close
from
    TickVO.win:expr_batch(sum(volume) > 1000)
group by
    securityId;
```

---

**Warning**

Some market data providers (e.g. Interactive Brokers) will only provide market data snapshots at regular time intervals. This can cause deviations in the bar high and low price.

Due to clock differences between the local machine and the market data provider's servers there might be slight deviations in the bar open and close price. It is therefore important to enable system clock synchronization on the server where AlgoTrader is installed.
17.2. Creation of Bars based on Bars

You might need to have more than one aggregation level of bars in your strategy. For this you can initially create the smallest bar size you require according to *Section 18.2, “Live Data Recording”* and then aggregate these bars to get larger bars.

There are several ways to do this:

- Using Esper
- Using Java

17.2.1. Esper Bar Aggregation

Aggregating 1 minute bars into 10 minute bars:

```java
@Name('AGGREGATE_BAR_TO_BAR')
insert into BarVO
select first(bar.dateTime),
      last(bar.adapterType),
      last(bar.securityId),
      Duration.valueOf('MIN_10'),
      first(bar.open),
      max(bar.high),
      min(bar.low),
      last(bar.close),
      sum(bar.vol)
from BarVO(barSize = Duration.valueOf('MIN_1')).win:time_batch(10 min, 0L, "FORCE_UPDATE")
as bar
group by bar.securityId,
       bar.adapterType;
```

17.2.2. Java Bar Aggregation

Java Bar to Bar Aggregation can be performed using the AlgoTrader `BarToBarAggregation`. Inside strategies an aggregation to 10 Minute Bars looks like this:

```java
private BarToBarAggregation aggregation = new BarToBarAggregation(Duration.MIN_10, c -> {
    // action will be performed on each newly produced bar
    logger.info(c);
});
```
@Override
public void onBar(BarVO bar) {
    // add bar to the aggregation
    aggregation.add(bar);
}

17.3. Numeric Precision

In general, different securities are traded with different numeric precision (e.g., S&P Futures prices are 2 digits, whereas FX prices are usually 5-6 digits, and crypto currencies are up to 8 digits). To accommodate different numeric precisions, AlgoTrader provides the following fields inside the class `Security`:

- `minQty`: The minimum tradeable quantity
- `maxQty`: The maximum tradeable quantity
- `qtyIncr`: Minimum quantity increase
- `minPrice`: The minimum price
- `maxPrice`: The maximum price
- `priceIncr`: Minimum price increase
- `minNotional`: Minimum order value (quantity x price)

There are also other parameters that describe order minimums and maximums for quantity and price. They are described in section: Section 22.2.2, “Crypto-Order Constraints”

17.4. Price normalization

Price normalization comes into play when multiple market data and/or trading adapters are in use that use different price multipliers / contract sizes for the same instrument. For example, one adapter might quote prices in dollars with a contract size of 10 where as another one might quote them in cents with a contract size of 1000. The default contract size will be stored in the `security_family` table.

17.5. Market Data Gap Checking

Since a continuous data feed of market data is essential for most trading strategies, AlgoTrader contains a feature that automatically warns if no market data has been received for a prolonged period of time. For this purpose, the class `Security` has a property `maxGap`, that defines the maximum number of minutes allowed without any market data updates. This is enforced by the `CHECK_SECURITY_TICK_GAPS` statement which can be turned on by changing the following property inside `conf-core.properties`. Alternatively, the properties can be changed via Section 2.3, “VM Arguments”

# enables security tick gap check
Crypto currency exchanges are typically using web sockets to deliver market data. Web socket connections are typically not very stable and it can happen that a web socket connection disconnects or suddenly stops receiving data. For this purpose AlgoTrader has a feature that automatically reconnects the corresponding adapter if no market data has been received for a prolonged period of time. This is enforced by the CHECK_ADAPTER_TICK_GAP statement which can be turned on by changing the following property inside conf-core.properties. Alternatively the properties can be changed via Section 2.3, “VM Arguments”.

17.6. Generic Events

In addition to MarketDataEvents (i.e. TickVOs and BarVOs) there are general purpose Generic Events that can contain any type of information (e.g. virtual market data, signals, exposure values, etc.). In contrary to MarketDataEvents, which are sent from the AlgoTrader Server to subscribed strategies, these Generic Events can be sent from one strategy to one or more receiving strategies. A Generic Event class has to subclass GenericEventVO. Subscription to Generic Events is based on the class of the Generic Event.

Example: To subscribe to a Generic Event of type Signal (which is a subclass of GenericEventVO), the following needs to be done:

getSubscriptionService().subscribeGenericEvents(Collections.singleton(Signal.class));

To send a GenericEvent, use the following method:

getEventDispatcher().broadcast(signal, EventRecipient.REMOTE_ONLY);

Alternatively instead of broadcasting, it is possible to send the events directly to the desired strategy as follows:

eventDispatcher.sendEvent(strategy.getName(), signalObject);

If the event is sent directly to the strategy, the target strategy doesn’t need to be explicitly subscribed to this event.

GenericEvents are automatically propagated to the Strategy Esper Engine where they can be accessed as follows:

select * from Signal;
In order to use it from Esper as shown above the event type Signal needs to be added to the Esper config file (which in turn is defined in the strategy's context file)

Example generic event type registration:

```xml
<event-type
    name="StrategyPropertyChangeEventVO"
    class="ch.algotrader.vo.StrategyPropertyChangeEventVO"/>
```

Alternatively, if the intended usage is only from Java code and there is no need to use it within Esper, the event is accessible directly through the `onGenericEvents` method:

```java
@override
public void onGenericEvent(GenericEventVO event) {
    ...
}
```

It is also possible to feed Generic Events via CSV Files in Simulation Mode. To enable this, the following property inside `conf.properties` has to be updated. Alternatively the properties can be changed via Section 2.3, "VM Arguments"

```properties
# should generic events be feed
dataSource.feedGenericEvents = true
```

The file name of the CSV File has to be according to this schema:

```text
<className>.<rank>.csv
```

- `className` is the fully qualified class name (e.g. `ch.algotrader.event.Signal`)
- `rank` is the sort order for situations where there are multiple `GenericEvent` types for the same time stamp

### 17.7. Generic Tick Events

In contrast to Section 17.6, “Generic Events” the system also provides Generic Tick Events (class `GenericTickVO`). A Generic Tick Event represents additional price information on a particular instrument made available by market data providers (e.g. open price, close price, vwap price).

As `GenericTickVO` is a subclass of `MarketDataEventVO` a strategy will automatically gen Generic Tick Events delivered when it has subscribed to the corresponding instrument.

A Generic Tick has a `TickType` which can be one of `OPEN`, `HIGH`, `LOW`, `CLOSE`, `SETTLEMENT`, `OPEN_INTEREST`, `IMBALANCE` or `VWAP`. A Generic Tick Event can hold either a `BigDecimal`, `Double` or `Integer` value.
Historical Data

AlgoTrader provides Historical Data Interfaces with the following market data providers:

- Native Interfaces
  - Bloomberg
  - Interactive Brokers
  - Quandl
- Crypto REST/WebSocket Interfaces
  - CoinAPI
  - CoinMarketCap

AlgoTrader uses the time series database InfluxDB for storage of historical data. InfluxDB is an open source database written in Go specifically designed to handle time series data with high availability and high performance requirements.

In addition the platform also provides a feature for downloading historical data from external market data providers. This historical data can be used for strategy simulations or for any type of analysis.

The Historical Data Manager (Section 10.4, “Historical Data Manager”) client provides a convenient UI for managing historical data.

The HistoricalDataService provides all relevant functions for historical data:

- Retrieval: e.g. getTicksByMaxDate, getLastNBarsBySecurityAndBarSize, etc.
- Storage: storeHistoricalBars
- Download: e.g. downloadHistoricalBars, downloadHistoricalTicks, etc.

To use the Historical Data Service the corresponding Spring profiles have to be added via VM argument:

Bloomberg Historical Data Service:

```
-Dspring.profiles.active=influxDB,bBHistoricalData
```

InteractiveBrokers Historical Data Service:

Quandl Historical Data Service:

-\texttt{spring.profiles.active=influxDB,qDLHistoricalData}

CoinAPI Historical Data Service:

-\texttt{spring.profiles.active=influxDB,cNPHistoricalData}

CoinMarketCap Historical Data Service:

-\texttt{spring.profiles.active=influxDB,cMCHistoricalData}

Noop Historical Data Service:

-\texttt{spring.profiles.active=influxDB,noopHistoricalData}

\textbf{Note}

The Noop Historical Data Service does not have a connection to an external data source. It can be used during Simulation to access existing historical data from InfluxDB.

\textbf{Note}

When running a strategy in distributed mode in case that requires historical data service, a special \texttt{historicalData} profile must be specified on the strategy side. In addition an actual historical data profile (e.g. \texttt{iBHistoricalData}) must be specified on the server side.

-\texttt{spring.profiles.active=live,historicalData}

\subsection{18.1. InfluxDB}

For detailed information on InfluxDB please have a look at the \textit{InfluxDB Documentation}\footnote{https://docs.influxdata.com/influxdb/}.

InfluxDB can be installed locally or using Docker, please see \textit{Chapter 2, Installation and Deployment}.

Data in an InfluxDB instance can be managed via the AlgoTrader Historical Data Manager (\textit{Section 10.4, "Historical Data Manager"}).
InfluxDB provides both a command line client (CLI) as well as a REST based client which is used by various client side libraries. AlgoTrader uses the `influxdb-java`³ library to communicate with InfluxDB. For all operations that involve the time series database InfluxDB, the following spring profile has to be specified via VM argument:

```
-Dspring.profiles.active=influxDB
```

If InfluxDB is installed locally, the `influx` command should be available via the command line. Executing `influx` will start the CLI and automatically connect to the local InfluxDB instance. The output should look like this:

```
$ influx -precision rfc3339
Connected to http://localhost:8086 version 1.1.x
InfluxDB shell 1.1.x
>
```

**Note**

- The InfluxDB HTTP API runs on port 8086 by default. Therefore, `influx` will connect to port 8086 and `localhost` by default. If these defaults need to be altered please run `influx --help`.

- The `-precision` argument specifies the format/precision of any returned timestamps. In the example above, `rfc3339` tells InfluxDB to return timestamps in **RFC3339 format**⁴ (YYYY-MM-DDTHH:MM:SS.nnnnnnnnnnZ).

The command line is now ready to take input in the form of the Influx Query Language (a.k.a InfluxQL) statements. To exit the InfluxQL shell, type `exit` and hit return.

Most InfluxQL statements must operate against a specific database. The CLI provides a convenience statement, `USE <db-name>`, which will automatically set the database for all future requests. To use the `algotrader` database please type:

```
> USE algotrader
Using database algotrader
>
```

Now future commands will only be run against the `algotrader` database.

At this point SQL-like queries can be executed against the database. In InfluxDB tables are called measurements. AlgoTrader uses the two measurements `tick` and `bar`. Columns that hold actual data (e.g. `open` or `high`) are called fields, and columns holding static data (e.g. `barSize`) are called tags.

³ https://github.com/influxdata/influxdb-java
⁴ https://www.ietf.org/rfc/rfc3339.txt
As an example the following query shows all current bars in the database:

```
> select * from bar
name: bar
time securityId vol
----- ------ -------
--- 0 2017-01-02T16:48:05Z MIN_1 116.41
  91 2017-01-02T16:49:05Z MIN_1 116.44
  93 2017-01-02T16:50:04Z MIN_1 116.44
0 2017-01-02T16:59:00Z MIN_1 116.49
```

For an in depth description of the query syntax please visit the InfluxDB query language documentation.

To import existing data into InfluxDB please use the following command:

```
> influx -import -path <path-to-file>
```

To import bar data the import file has to be formatted as follows:

```
# DML
# CONTEXT-DATABASE: algotrader

bar,securityId=25,adapterType=IB,barSize=MIN_1
open=1.30319,high=1.30402,low=1.30319,close=1.30367,vol=0 1324245720000000000
bar,securityId=25,adapterType=IB,barSize=MIN_1
open=1.30369,high=1.30369,low=1.30351,close=1.30352,vol=0 1324245780000000000
bar,securityId=25,adapterType=IB,barSize=MIN_1
open=1.30353,high=1.30383,low=1.30353,close=1.30382,vol=0 1324245840000000000
bar,securityId=25,adapterType=IB,barSize=MIN_1
open=1.30381,high=1.30411,low=1.30373,close=1.30373,vol=0 1324245900000000000
bar,securityId=25,adapterType=IB,barSize=MIN_1
open=1.30378,high=1.30428,low=1.30376,close=1.30425,vol=0 1324245960000000000
bar,securityId=25,adapterType=IB,barSize=MIN_1
open=1.30426,high=1.30426,low=1.30396,close=1.30399,vol=0 1324246020000000000
bar,securityId=25,adapterType=IB,barSize=MIN_1
open=1.30401,high=1.30411,low=1.30371,close=1.30378,vol=0 1324246080000000000
```

To import tick data the import file has to be formatted as follows:

```
# DML
```

---

5 https://docs.influxdata.com/influxdb/v1.1/query_language/
# CONTEXT-DATABASE: algotrader

tick,securityId=25,adapterType=IB
 last=1.303670,lastDateTime=1324245720000,bid=1.303670,ask=1.303670,volBid=0,volAsk=0,vol=0
 1324245600000000000
 tick,securityId=25,adapterType=IB
 last=1.303670,lastDateTime=1324245720000,bid=1.303670,ask=1.303670,volBid=0,volAsk=0,vol=0
 1324245660000000000
 tick,securityId=25,adapterType=IB
 last=1.303670,lastDateTime=1324245720000,bid=1.303670,ask=1.303670,volBid=0,volAsk=0,vol=0
 1324245720000000000
 tick,securityId=25,adapterType=IB
 last=1.303670,lastDateTime=1324245720000,bid=1.303670,ask=1.303670,volBid=0,volAsk=0,vol=0
 1324245780000000000
 tick,securityId=25,adapterType=IB
 last=1.303670,lastDateTime=1324245720000,bid=1.303670,ask=1.303670,volBid=0,volAsk=0,vol=0
 1324245840000000000
 tick,securityId=25,adapterType=IB
 last=1.303670,lastDateTime=1324245720000,bid=1.303670,ask=1.303670,volBid=0,volAsk=0,vol=0
 1324245900000000000
 tick,securityId=25,adapterType=IB
 last=1.303670,lastDateTime=1324245720000,bid=1.303670,ask=1.303670,volBid=0,volAsk=0,vol=0
 1324245960000000000

For further information on InfluxDB import please visit the InfluxDB documentation\(^6\)

---

**Note**

- The last column in the import file represents the time stamp, which needs to be defined in nanoseconds since the 1970-01-01
- It is also possible to compress import files. In this case the command line switch `-compressed` has to be used when importing files.

To export bar data from InfluxDB into the AlgoTrader CSV file format (see [Section 18.9, “Market Data File Format”](#)) please use the following command:

```
> influx -execute "SELECT time as dateTime,open,high,low,close,vol FROM bar" -database "algotrader" -format csv -precision ms > bar.csv
```

To export tick data from InfluxDB please use the following command:

```
> influx -execute "SELECT time as dateTime,last,lastDateTime,volBid,volAsk,bid,ask,vol FROM tick" -database "algotrader" -format csv -precision ms > tick.csv
```

---

\(^6\) [https://docs.influxdata.com/influxdb/v1.2/tools/shell/](https://docs.influxdata.com/influxdb/v1.2/tools/shell/)
18.2. Live Data Recording

Using InfluxDB it is possible to store tick-level live data for all subscribed instruments while the system is running. To enable this feature the following properties inside `conf-core.properties` has to be enabled. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```
# enables market data persistence
statement.persistMarketData = true
```

In addition recorded tick-level data can be aggregated into bar-data on the fly by using the following properties inside `conf-core.properties`. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```
# enables market data persistence
statement.aggregateBars = true

# the bar size used for tick-to-bar aggregation and end-of-day historical bar download
historicalData.barSize = MIN_1
```

In case a certain instrument provides trade information (e.g. Equities and Crypto Currencies) the last traded price is used to calculate the Bar values. In case no trading information is available (e.g. Forex and Indices) the midpoint price (average of bid and ask) is used. The bar aggregation feature will also create bars for time periods when no market data arrives, in this case open, high, low and close will be equal to the previous bars close price.

18.3. Historical Data Download

The `storeHistoricalBars` method of the Historical Data Service saves historical bars directly into InfluxDB. If the parameter `replace` is set to false the method `storeHistoricalBars` will save newly retrieved Bars after the last Bar currently in the database. Bars before the current last Bar will not be touched. If the parameter...
Replace is set to true the method storeHistoricalBars however will replace all current Bars in the database within the specified time period.

Download and storage of historical data can be invoked via the HistoricalDataStarter.

```
HistoricalDataStarter replaceBars startDate endDate marketDataEventType barSize securityId(s)
```

For example:

```
HistoricalDataStarter true 2016-01-01 2016-12-31 TRADES DAY_1 10 11 12
```

AlgoTrader also provides features to download missing historical data for all subscribed instruments either on startup or at a specific time of the day. For these functions the following properties are available inside conf-core.properties where they can be changed. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```
# enables end-of-day historical bar download
statement.downloadHistoricalDataEOD = true

# the bar size used for tick-to-bar aggregation and end-of-day historical bar download
historicalData.barSize = MIN_1

# the market data event type used by the end-of-day historical bar download
historicalData.marketDataEventType = MIDPOINT

# Hour-of-Day when the end-of-day historical bar download takes place
historicalData.downloadHour = 2

# enables historical bar download on startup
historicalData.downloadHistoricalDataOnStartup = true
```

Note

Available market data event types are:

- TRADES
- MIDPOINT
- BID
- ASK
- BID_ASK
- BEST_BID
Depending on whether Interactive Brokers, Bloomberg or Quandl is used for the historical data download the corresponding `marketData` profile has to be specified via VM argument.

**InteractiveBrokers:**

```bash
-Dspring.profiles.active=influxDB,iBHistoricalData
```

**Bloomberg:**

```bash
-Dspring.profiles.active=influxDB,bBHistoricalData
```

**Quandl:**

```bash
-Dspring.profiles.active=influxDB,qdlHistoricalData
```

### 18.4. Interactive Brokers Historical Data Download

The Historical Data Download incorporates historical data limitations\(^7\) in place by Interactive Brokers.

With IB API the following conditions can lead to pacing violations:

- Making six or more historical data requests for the same Contract, Exchange and Tick Type within two seconds.

- Making more than 60 historical data requests in any ten-minute period.

The AlgoTrader Historical Data Download can optionally avoid potential pacing violation by separating subsequent download requests by 10 seconds. This feature can be enabled via the following property inside `conf-ib.properties` has to be updated. Alternatively the properties can be changed via **Section 2.3, “VM Arguments”**

```bash
# ensures 10 seconds delay between historic data requests
ib.pacingViolationGuard = true
```

The Historical Data Download also takes the Valid Duration and Bar Size Settings\(^8\) for Historical Data Requests into account and splits large requests into subsequent smaller requests.

---

\(^7\) [http://interactivebrokers.github.io/tws-api/historical_limitations.html](http://interactivebrokers.github.io/tws-api/historical_limitations.html)

\(^8\) [http://interactivebrokers.github.io/tws-api/historical_limitations.html](http://interactivebrokers.github.io/tws-api/historical_limitations.html)
18.5. Quandl Historical Data Download

Quandl\(^9\) is a public service that provides a wide range of financial, economic and alternative data. AlgoTrader allows downloading historical data from Quandl. For more information please visit section Section 22.18, “Quandl”.

18.6. Google Finance Historical Data Download

To download historical data from Google Finance\(^10\) the following two classes are available: The class ch.algotrader.starter.GoogleDailyDownloader is available to download daily closing prices and the class ch.algotrader.starter.GoogleIntradayDownloader can be used to download intraday prices.

18.7. CoinAPI Historical Data Download

CoinAPI\(^11\) is a platform which provides fast, reliable unified data APIs to cryptocurrency markets. AlgoTrader allows downloading historical data from CoinAPI. For more information please visit section Section 22.28, “CoinAPI”.

18.8. CoinMarketCap Historical Data Download

CoinMarketCap\(^12\) is a provider for cryptocurrency data. AlgoTrader allows downloading historical data from CoinMarketCap. For more information please visit section Section 22.31, “CoinMarketCap”.

18.9. Market Data File Format

When using CSV files for the back test all data files are placed inside the following directory structure:

\[
/\text{baseDir}/<\text{eventType}>/<\text{dataSet}>/<\text{filename}>.csv
\]

- **baseDir** is the parent directory where all market data files are stored. This is either the files/ directory under the project algo trader-core or an arbitrary directory defined via the following property inside conf.properties has to be updated. Alternatively the properties can be changed via Section 2.3, “VM Arguments”

```java
# alternate dataSetLocation (default is <working-dir>/files/ i.e. usually
<algotrader>/core/files/ )
dataSource.dataSetLocation = files
```

- **eventType** is either **tickData**, **barData** or **genericData** (see Section 17.6, “Generic Events”).

---

\(^9\) https://www.quandl.com/
\(^10\) https://www.google.com/finance/historical?q=aapl
\(^11\) https://www.coinapi.io/
\(^12\) https://coinmarketcap.com/
• `dataSet` is the name of the dataset used for the simulation run. This can be defined via the following property inside `conf.properties` has to be updated. Alternatively the properties can be changed via `Section 2.3, “VM Arguments”`

```
# name of dataSet to be used for simulations and market data persistence
dataSource.dataSet = current
```

• `filename` can be either of the following values followed by `.csv`
  
  • isin
  
  • symbol
  
  • bbgid
  
  • ric
  
  • conid
  
  • securityId

An alternative approach is to feed market data for multiple securities using one file. E.g. it is possible to feed market data for futures using market data from the corresponding generic future. In this approach an additional column `security` has to be added to the market data file which will be used to identify the actual Security.

The first line within the file is the header row.

The file name for `Section 17.6, “Generic Events”` follows a different logic.

### 18.9.1. Tick Data Files

The Format of the Tick Data Files is based on a standard CSV Structure:

• `dateTime`
• `last`
• `lastDateTime`
• `volBid`
• `volAsk`
• `bid`
• `ask`
• `vol`
• `security (optional)`
dateTime and lastDateTime values are expected to be in the \texttt{yyyy-MM-dd HH:mm:ss} format and to represent local time. Alternatively one can also use long values that represent Java milliseconds since 1970.

Example:

### Table 18.1. Tick Data Format

<table>
<thead>
<tr>
<th>dateTime</th>
<th>last</th>
<th>lastDateTime</th>
<th>volBid</th>
<th>volAsk</th>
<th>bid</th>
<th>ask</th>
<th>vol</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-01-01 14:00:01</td>
<td>188</td>
<td>2016-01-01 14:00:01</td>
<td>47</td>
<td>52</td>
<td>178.1</td>
<td>183.2</td>
<td>20</td>
</tr>
<tr>
<td>2016-01-01 14:00:02</td>
<td>188</td>
<td>2016-01-01 14:00:02</td>
<td>47</td>
<td>52</td>
<td>177.2</td>
<td>182.9</td>
<td>20</td>
</tr>
</tbody>
</table>

#### 18.9.2. Bar Data Files

The Format of the Bar Data Files is based on a standard CSV Structure:

- dateTime
- open
- high
- low
- close
- vol
- vwap (optional)
- security (optional)

dateTime values are expected to be in the \texttt{yyyy-MM-dd HH:mm:ss} format and to represent local time. Alternatively one can also use long values that represent Java milliseconds since 1970.

Example:

### Table 18.2. Bar Data Format

<table>
<thead>
<tr>
<th>dateTime</th>
<th>open</th>
<th>high</th>
<th>low</th>
<th>close</th>
<th>vol</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-01-01 14:00:00</td>
<td>1.29366</td>
<td>1.29369</td>
<td>1.29360</td>
<td>1.29369</td>
<td>2000</td>
</tr>
<tr>
<td>2016-01-01 14:01:00</td>
<td>1.29367</td>
<td>1.29389</td>
<td>1.29367</td>
<td>1.29378</td>
<td>2500</td>
</tr>
</tbody>
</table>
Reference Data


Reference Data can either be configured in the database directly through the corresponding tables, one can use the ReferenceDataService and corresponding ReferenceDataStarter or you can use the Reference Data Manager UI, see Section 10.3, “Reference Data Manager”.

Depending on the Reference Data Adapter in use the following download options are available for download:

- All Future of all Security Families (of family type Future)
- All Futures of a particular Security Family
- All Options of all Security Families (of family type Option)
- All Options of a particular Security Family
- All Stocks of a particular Security Family
- All items available through the particular Reference Data Adapter

For further details please see the JavaDoc of the ReferenceDataStarter class.

Example: To download missing Futures of a specified Security Families use the following command

```
ReferenceDataStarter futures securityFamilyId1 securityFamilyId2 ...
```

It is recommended to run this Service in the interval of Option / Future Expirations to make sure that the entire chain is available to strategies.

Depending on the Reference Data Adapter in use the corresponding referenceData profile has to be specified via VM argument.

Bloomberg:

```
-Dspring.profiles.active=<dataSource>,bBReferenceData
```

InteractiveBrokers:

```
-Dspring.profiles.active=<dataSource>,iBReferenceData
```

Trading Technologies:

```
-Dspring.profiles.active=<dataSource>,tTReferenceData
```

Binance:
Bitfinex:
-Dspring.profiles.active=<dataSource>,bFXReferenceData

Bitflyer:
-Dspring.profiles.active=<dataSource>,bFLReferenceData

BitMEX:
-Dspring.profiles.active=<dataSource>,bMXReferenceData

Bitstamp:
-Dspring.profiles.active=<dataSource>,bTSReferenceData

CoinAPI:
-Dspring.profiles.active=<dataSource>,cNPReferenceData

Coinbase Pro:
-Dspring.profiles.active=<dataSource>,cNBReferenceData

Coinigy:
-Dspring.profiles.active=<dataSource>,cNGReferenceData

CoinMarketCap:
-Dspring.profiles.active=<dataSource>,cMCReferenceData

Note
When running a strategy in distributed mode that requires reference data service, a special referenceData profile must be specified on the strategy side. In addition an actual historical data profile (e.g. iBReferenceData) must be specified on the server side.

-Dspring.profiles.active=live,referenceData
Account Data

AlgoTrader provides Account Data Service for the following adapters:

- Interactive Brokers via Native API
- Binance via Binance API\(^1\)
- Bitfinex via Bitfinex API\(^2\)
- Bitflyer via Bitflyer API\(^3\)
- BitMEX via BitMEX API\(^4\)
- Bitstamp via Bitstamp API\(^5\)
- Coinbase Pro via Coinbase API\(^6\)
- Coinigy via Coinigy API\(^7\)

Depending on the Account Data Adapter in use the corresponding account profile has to be specified via VM argument.

InteractiveBrokers:

```
-Dspring.profiles.active=iBAccount
```

Binance:

```
-Dspring.profiles.active=bNCAccount
```

Bitfinex:

```
-Dspring.profiles.active=bFXAccount
```

Bitflyer:

```
-Dspring.profiles.active=bFLAccount
```

BitMEX:

```
-Dspring.profiles.active=bMEXAccount
```

\(^1\) https://github.com/binance-exchange/binance-java-api
\(^2\) https://docs.bitfinex.com/docs
\(^3\) https://bitflyer.com/api
\(^4\) https://www.bitmex.com/app/apiOverview
\(^5\) https://www.bitstamp.net/api/
\(^6\) https://docs.pro.coinbase.com/
\(^7\) https://coinigy.docs.apiary.io/
Bitstamp:

-Dspring.profiles.active=bTSAccount

Coinbase Pro:

-Dspring.profiles.active=cNBAccount

Coinigy:

-Dspring.profiles.active=cNGAccount

Note

When running a strategy in distributed mode in case that requires account data service, a special account profile must be specified on the strategy side. In addition an actual historical data profile (e.g. iBAccount) must be specified on the server side.

-Dspring.profiles.active=live,account

20.1. Account balances

The AccountService interface defines a method for retrieving account balances for the specified account ID. A list of NamedCurrencyAmountVO items is returned. The retrieveAccountBalances method can be called from the strategy like this:

\[
\text{List<NamedCurrencyAmountVO> balances = getAccountService().retrieveAccountBalances(accountId);}\]

20.2. Withdrawal

The AccountService interface defines a method to initiate a crypto withdrawal for crypto exchanges. The withdraw method can be called from the strategy like this:

\[
\text{WithdrawStatusVO status = getAccountService().withdraw(accountId, currency, amount, withdrawContext)}\]

The withdrawContext parameter contains additional information that might be required by certain exchanges (e.g. address and/or paymentId). The method returns a WithdrawStatusVO which contains information like message and externalId.
For withdrawal operations exchanges typically require additional security constraints, which might include:

- API key with explicit permission to withdraw
- Two-factor authentication (2FA)
- IP address whitelisting
- Deposit address whitelisting
- Email confirmation

Please consult the corresponding exchange’s documentation about the details on how to enable withdrawals.

**Note**

Typically each adapter has a configurable list of supported currencies, for example for Bitfinex this list is configured in `conf-bfx.properties` file. This is adapter specific, but often a withdrawal operation is only allowed if the currency is explicitly listed (please check the corresponding adapter section).

### 20.3. Deposit address

The `AccountService` provides a method for getting the deposit address. The method can be called from a strategy like this:

```java
String depositAddress = getAccountService().getDepositAddress(DepositAddressVo depositParams);
```

This returns the deposit address for an exchange, currency and wallet type (wallet type is required for Bitfinex only). It returns the value as a String. Please check the exchange documentation on how to create deposit addresses. The `getDepositAddress` method sends the request to an enabled exchange (see Section 25.1, “Starter Classes”). If the requested exchange profile is not present, an exception is thrown. Some exchanges support different currencies (please check the corresponding adapter section).

### 20.4. Account Events

Using the `AccountService` it is also possible to subscribe for account events. After successful subscription the strategy will be notified each time the account balances are changed.

In order to subscribe the strategy needs to call

```java
getAccountService().subscribeAccountEvent(accountId);
```
To process events the strategy must implement a listener method:

```java
void onAccountEvent(AccountEventVO event)
```

The `AccountEventVO` has a method `getAccountBalances` which returns the received balances.

```java
List<NamedCurrencyAmountVO> getAccountBalances();
```

Unlike the `retrievalAccountBalances` which always return all balances, the account update event may contain only the balance which has changed.

**Note**

Subscription for account events is supported only by selected exchanges (please check the corresponding adapter section).
AlgoTrader API

The following sections describe how the AlgoTrader system can be accessed by external systems.

21.1. JSON data binding

AlgoTrader uses a consistent message format based on JSON for both the WebSocket/STOMP API and the REST API.

Using the JSON messages in combination with the AlgoTrader REST API and WebSocket/STOMP API any popular development language can be used to build trading strategies making use of the AlgoTrader platform, for example:

- C#
- C++
- Python
- R
- MatLab
- JavaScript / NodeJS

21.2. REST API

Access to most of the functionality and data provided by AlgoTrader is available via a REST based API.

The full documentation of the AlgoTrader REST API is available here.¹

HTTP GET endpoints can easily be queried via the Browser. To retrieve a JSON formatted list of all accounts open to the following URL in the Browser:

```
http://localhost:9090/rest/account
```

In addition one can use `Curl²`, a popular utility for execution of HTTP requests. The following example shows how to retrieve a JSON formatted list of all accounts by executing HTTP GET request:

```
$ curl -X GET http://localhost:9090/rest/account -i
HTTP/1.1 200 OK
Content-Type: application/json
```

¹ http://doc.algotrader.ch/apidocs/index.html
² https://curl.haxx.se/
Similarly one can request AlgoTrader to subscribe to market data for security with id 11 by executing HTTP PUT request:

```bash
$ curl -X PUT -H "content-type: application/json" 
http://localhost:9090/rest/subscription/marketdata/subscribe 
-d "{"strategyName":"SERVER","securityId":11,"subscribe":true}" -i
HTTP/1.1 200 OK
Content-Length: 0
Server: Jetty(9.3.6.v20151106)
```

### 21.3. WebSocket/STOMP API

AlgoTrader employs STOMP messaging protocol over WebSockets transport to implement multi-topic, multi-client message delivery based on the Publish-Subscribe pattern. AlgoTrader acts as a message producer that generates messages representing various system or trading related events and publishes them to predefined topics. Browsers running the AlgoTrader UI (and potentially any external application supporting STOMP over WebSockets) act as message consumers that subscribe to message topics of interest such as market data, orders, order status updates, position changes, executed transactions and so on. Consumers express their interest in a particular type of event by subscribing to message topics. Consumers should no longer need to filter out unwanted messages. They are expected to subscribe only to a subset of messages they are interested in.

For further details on the STOMP protocol please visit the [STOMP website]({#stomp.github.io/}).

AlgoTrader publishes events to multiple event topics.

**Table 21.1. Event topics**

<table>
<thead>
<tr>
<th>Topic format</th>
<th>Type of events</th>
</tr>
</thead>
<tbody>
<tr>
<td>tick.&lt;security_id&gt;</td>
<td>TickVO</td>
</tr>
<tr>
<td>order.&lt;strategy_name&gt;.&lt;order_int_id&gt;</td>
<td>OrderVO</td>
</tr>
<tr>
<td>order-status.&lt;strategy_name&gt;.&lt;order_int_id&gt;</td>
<td>OrderStatusVO</td>
</tr>
<tr>
<td>transaction.&lt;strategy_name&gt;.&lt;uuid&gt;</td>
<td>TransactionVO</td>
</tr>
</tbody>
</table>

---

3 [https://stomp.github.io/](https://stomp.github.io/)
Topics are organized by name spaces. A consumer wishing to receive market data for security with id 12 only can subscribe to the following topic:

\texttt{tick.12}

A consumer wishing to receive market data all securities can subscribe to the following wild card topic.

\texttt{tick.*}

Strategy specific events are organized by strategy name. A consumer wishing to receive order status updates for the order with internal id 10 issued by strategy MY\_STRATEGY can subscribe to the following topic

\texttt{order-status.MY\_STRATEGY.10}

A consumer wishing to receive order status updates for all orders issued by strategy MY\_STRATEGY can subscribe to the following wild card topic

\texttt{order-status.MY\_STRATEGY.*}

The * wild card selects all elements within the same namespace

A consumer wishing to receive order status updates for all orders of all strategies can subscribe to the following wild card topic

\texttt{order-status.>}

The > wild card selects all topics within the same namespace and their sub-namespaces.

In order to ensure optimal performance of HTML5 clients AlgoTrader can throttle market data event delivered by the WebSockets transport. The embedded message broker by default attempts to ensure that the total rate of events per connection does not exceed 50 per second. At the same time instruments with infrequent market data updates are not throttled if their total event rate is below 0.1 per second (less that one event every 10 seconds).

Throttling rates can be adjusted by changing the following configuration parameters:
activeMQ.maxRatePerConnection = 50
activeMQ.minRatePerConsumer = 0.1

In JavaScript STOMP messages can be consumed like this:

```html
<html>
<head>
    <script src="https://unpkg.com/@stomp/stompjs@4.0.6/lib/stomp.min.js"></script>
    <script type="text/javascript">
        var ws = new WebSocket("ws://localhost:61614", "stomp");
        var stompClient = Stomp.over(ws);
        stompClient.connect({}, function(frame) {
            stompClient.subscribe('/topic/tick.*', function(message){
                console.log(JSON.parse(message.body));
            });
        });
    </script>
</head>
</html>
```

For further details please visit the [STOMP JavaScript documentation](http://jmesnil.net/stomp-websocket/doc/).

AlgoTrader supports STOMP over normal TCP connections. It designed initially for Python clients.

Sample subscription consumer code in Python:

```python
import time
import sys
import stomp

class MyListener(stomp.ConnectionListener):
    def on_error(self, headers, message):
        print('received an error "%s"' % message)
    def on_message(self, headers, message):
        print('received a message "%s"' % message)

hosts = [('localhost', 61618)]

conn = stomp.Connection(host_and_ports=hosts)
conn.set_ssl(for_hosts=hosts)
conn.set_listener('', MyListener())
conn.start()
print('stared')
conn.connect()
```

4 http://jmesnil.net/stomp-websocket/doc/
21.4. Inbound FIX API

AlgoTrader provides Inbound FIX protocol based API which allows submission, modification and cancellation of simple orders (Market, Limit, Stop, Stop Limit). Those are then forwarded via the configured trading adapter to the relevant broker / exchange. You can also configure AlgoTrader to connect to multiple exchanges and use Inbound FIX API to submit orders to any of them.

Order status changes from the broker/exchange are then forwarded back to the external FIX client. This mechanism allows using the industry standard FIX API for order management via all trading adapters AlgoTrader provides. Thus also for exchanges or brokers that do not natively support FIX.

The following is an example of the QuickFixJ config (fix.cfg) that you can use to make the Inbound FIX API available.

```
[session]
ConnectionType=acceptor
StartTime=00:00:00
EndTime=00:00:00
HeartBtInt=30
ValidOrderTypes=1,2,3,4
SenderCompID=AT
TargetCompID=BANZAI
  # FIX client should have those 2 in reverse order:
  # SenderCompID=BANZAI
  # TargetCompID=AT
UseDataDictionary=Y
FileStorePath=files/fix
LogFilePath=log
BeginString=FIX.4.4
SocketAcceptPort=9880
ValidateIncomingMessage=N
ResetOnLogon=Y
RefreshOnLogon=Y
```

You can test AlgoTrader Inbound FIX API using e.g. Banzai FIX GUI client⁵. Note that AlgoTrader specific custom tags 7001 (StrategyID), 7002 (StrategyName), 7003 (ExchangeOrder) on New Order Single messages are not supported by Banzai client. They can be added in the source code easily if necessary.

---

⁵ https://github.com/quickfix-j/quickfixj/tree/master/quickfixj-examples/banzai
The following is an example banzai.cfg config for connecting to locally running AlgoTrader:

```
[default]
FileStorePath=target/data/banzai
ConnectionType=initiator
SenderCompID=BANZAI
TargetCompID=AT
SocketConnectHost=localhost
StartTime=00:00:00
EndTime=00:00:00
HeartBtInt=300
ReconnectInterval=5
```

This is an example runner configuration for starting AlgoTrader with Inbound FIX API enabled and connected to Interactive Brokers:

```
Main Class: ch.algotrader.starter.ServerStarter
VM Options: -Dspring.profiles.active=live,pooledDataSource,embeddedBroker,inboundFix,iBNative,iBMarketData,html5
```

Note that you can open the HTML5 UI (http://localhost:9090) and the orders you place via Inbound FIX API will be visible there. QuickFixJ logs with FIX messages between Banzai and AlgoTrader can be found in \log \FIX.4.4-AT-BANZAI.messages.log file. The raw FIX messages can be translated by e.g. FIX decoder\(^6\).

The protocol version provided by the Algotrader Inbound FIX API is FIX 4.4. We use standard messages with 3 custom tags in New Order Single message. Following messages are supported:

- Logon (A)
- Logout (5)
- Test Request (1)
- Heartbeat (0)
- Resend Request (2)
- Execution Report (8)
- New Order Single (D)
- Business Message Reject (j)

\(^6\)https://drewnoakes.com/fix-decoder/
• Order Cancel/Replace Request (G)
• Order Cancel Request (F)
• Order Cancel Reject (9)

21.4.1. Logon message

Used to connect to AlgoTrader. Sent from external client to AlgoTrader.

Table 21.2. Logon

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Standard Msg Type</td>
<td>Y</td>
<td>Char</td>
<td>A</td>
</tr>
<tr>
<td>98</td>
<td>EncryptMethod</td>
<td>Y</td>
<td>Char</td>
<td>Should be '0' (NONE_OTHER) unless Inbound FIX API configured differently.</td>
</tr>
<tr>
<td>141</td>
<td>ResetSeqNumFlag</td>
<td></td>
<td>Char</td>
<td>'N' - Default. Sequence numbers should not be reset. 'Y' - Sequence numbers should be reset.</td>
</tr>
<tr>
<td>108</td>
<td>HeartBtInt</td>
<td></td>
<td>Char</td>
<td>Interval in seconds. Default: 30.</td>
</tr>
</tbody>
</table>

21.4.2. Logout message

Used to disconnect from AlgoTrader. Sent from external client to AlgoTrader.

Table 21.3. Logout

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Standard Msg Type</td>
<td>Y</td>
<td>Char</td>
<td>5</td>
</tr>
<tr>
<td>58</td>
<td>Text</td>
<td>N</td>
<td>String</td>
<td>Some arbitrary text value to indicate the intention to Logout.</td>
</tr>
</tbody>
</table>

21.4.3. Test Request message

The TestRequest message forces a Heartbeat from the opposing application. Sent from external client to AlgoTrader or vice versa.

Table 21.4. Test Request

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Standard Msg Type</td>
<td>Y</td>
<td>Char</td>
<td>1</td>
</tr>
</tbody>
</table>
21.4.4. Heartbeat message

Used to verify the connection with AlgoTrader. Sent from external client to AlgoTrader or vice versa.

**Table 21.5. Heartbeat**

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Standard Msg Type</td>
<td>Y</td>
<td>Char</td>
<td>0</td>
</tr>
<tr>
<td>112</td>
<td>TestReqID</td>
<td>N</td>
<td>String</td>
<td>The value in this field to be returned with the Heartbeat message (MsgType = 0).</td>
</tr>
</tbody>
</table>

21.4.5. Resend Request message

The Resend Request message is sent by the client to Algotrader to request the retransmission of messages.

**Table 21.6. Resend Request**

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Standard Msg Type</td>
<td>Y</td>
<td>Char</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>BeginSeqNo</td>
<td>Y</td>
<td>String</td>
<td>Message number of the first message in range to be resent.</td>
</tr>
<tr>
<td>16</td>
<td>EndSeqNo</td>
<td>Y</td>
<td>String</td>
<td>Message sequence number of last message in range to be resent.</td>
</tr>
</tbody>
</table>

21.4.6. New Order Single message

Used to submit a new order to AlgoTrader. Sent from external client to AlgoTrader.

**Table 21.7. New Order Single**

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Standard Msg Type</td>
<td>Y</td>
<td>Char</td>
<td>D</td>
</tr>
<tr>
<td>11</td>
<td>ClOrdID</td>
<td>Y</td>
<td>String</td>
<td>AlgoTrader order ID</td>
</tr>
<tr>
<td>54</td>
<td>Side</td>
<td>Y</td>
<td>Char</td>
<td>1 or 2 for Buy/Sell</td>
</tr>
<tr>
<td>60</td>
<td>TransactTime</td>
<td>Y</td>
<td>UTCTimestamp</td>
<td>UTC time when the message was sent</td>
</tr>
<tr>
<td>40</td>
<td>OrdType</td>
<td>Y</td>
<td>Char</td>
<td>1,2,3,4 for Market, Limit, Stop, StopLimit</td>
</tr>
<tr>
<td>38</td>
<td>OrderQty</td>
<td>Y</td>
<td>Qty</td>
<td>Total order quantity</td>
</tr>
<tr>
<td>Tag #</td>
<td>Field Name</td>
<td>Required</td>
<td>Data type</td>
<td>Comments</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------</td>
<td>----------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>21</td>
<td>HandlInst</td>
<td>N</td>
<td>Char</td>
<td>only value '1' is supported and assumed if not set</td>
</tr>
<tr>
<td>1</td>
<td>Account</td>
<td>Y</td>
<td>String</td>
<td>AlgoTrader account ID</td>
</tr>
<tr>
<td>22</td>
<td>SecurityIDSource</td>
<td>N</td>
<td>String</td>
<td>only value &quot;8&quot; is supported and assumed if not set</td>
</tr>
<tr>
<td>48</td>
<td>SecurityID</td>
<td>Y</td>
<td>String</td>
<td>AlgoTrader security ID</td>
</tr>
<tr>
<td>100</td>
<td>ExDestination</td>
<td>N</td>
<td>String</td>
<td>AlgoTrader exchange ID</td>
</tr>
<tr>
<td>207</td>
<td>SecurityExchange</td>
<td></td>
<td>String</td>
<td>AlgoTrader exchange name. Required if 100 is not set</td>
</tr>
<tr>
<td>59</td>
<td>TimeInForce</td>
<td>Y</td>
<td>Char</td>
<td>How long an order remains active. Supported: 0, 1, 2, 3, 4, 6, 7 (depending support by the exchange)</td>
</tr>
<tr>
<td>432</td>
<td>ExpireDate</td>
<td>C</td>
<td>LocalDate</td>
<td>Required for TIF Good Till Date</td>
</tr>
<tr>
<td>126</td>
<td>ExpireTime</td>
<td>C</td>
<td>LocalTime</td>
<td>Required for TIF Good Till Date</td>
</tr>
<tr>
<td>44</td>
<td>Price</td>
<td>C</td>
<td>Price</td>
<td>Required for Limit, StopLimit orders</td>
</tr>
<tr>
<td>99</td>
<td>StopPx</td>
<td>C</td>
<td>Price</td>
<td>Required for Stop, StopLimit orders</td>
</tr>
<tr>
<td>7001</td>
<td>StrategyID</td>
<td>N</td>
<td>String</td>
<td>AlgoTrader strategy ID</td>
</tr>
<tr>
<td>7002</td>
<td>StrategyName</td>
<td>N</td>
<td>String</td>
<td>AlgoTrader strategy name. If both 7001 and 7002 are not set, strategy=SERVER is assumed</td>
</tr>
<tr>
<td>7003</td>
<td>ExchangeOrderN</td>
<td></td>
<td>Boolean/</td>
<td>'Y' means exchange order and 'N' margin order on certain exchanges. Margin is default value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Char</td>
<td></td>
</tr>
</tbody>
</table>

### 21.4.7. Order Cancel Request message

Used to cancel an order. Sent from external client to AlgoTrader.

### Table 21.8. Cancel Request

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Standard Msg Type</td>
<td>Y</td>
<td>Char</td>
<td>F</td>
</tr>
<tr>
<td>11</td>
<td>AlgoTrader Order ID</td>
<td>Y</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

### 21.4.8. Order Cancel Replace Request message

Used to update an existing order provided that the corresponding exchange supports order modification. Sent from external client to AlgoTrader.
### Table 21.9. Cancel Replace Request

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Standard Msg Type</td>
<td>Y</td>
<td>Char</td>
<td>G</td>
</tr>
<tr>
<td>11</td>
<td>AlgoTrader Order ID</td>
<td>Y</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

#### 21.4.9. Execution Report message

Used to receive order update information from AlgoTrader, such as confirmation for fills, cancellations, modifications or rejections. Sent from AlgoTrader to the external client.

### Table 21.10. Execution Report

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Standard Msg Type</td>
<td>Y</td>
<td>Char</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Account</td>
<td>Y</td>
<td>String</td>
<td>AlgoTrader account ID</td>
</tr>
<tr>
<td>17</td>
<td>ExeclID</td>
<td>Y</td>
<td>String</td>
<td>Unique identifier for this execution report, UUID format</td>
</tr>
<tr>
<td>11</td>
<td>ClOrdID</td>
<td>Y</td>
<td>String</td>
<td>AlgoTrader order ID</td>
</tr>
<tr>
<td>37</td>
<td>OrderID</td>
<td>Y</td>
<td>String</td>
<td>AlgoTrader order ID</td>
</tr>
<tr>
<td>41</td>
<td>OrigClOrdID</td>
<td>C</td>
<td>String</td>
<td>Previous order identifier, equal to previous ClOrdID and sent after modification requests</td>
</tr>
<tr>
<td>38</td>
<td>OrderQty</td>
<td>Y</td>
<td>String</td>
<td>Total order quantity</td>
</tr>
<tr>
<td>6</td>
<td>AvgPx</td>
<td>Y</td>
<td>Price</td>
<td>Average price of all fills, 0 if no executions yet</td>
</tr>
<tr>
<td>151</td>
<td>LeavesQty</td>
<td>Y</td>
<td>Qty</td>
<td>Remaining quantity in the market. If order is still in the market LeavesQty = Qty - CumQty, 0 otherwise</td>
</tr>
<tr>
<td>14</td>
<td>CumQty</td>
<td>Y</td>
<td>Qty</td>
<td>Filled quantity so far</td>
</tr>
<tr>
<td>167</td>
<td>SecurityType</td>
<td>N</td>
<td>String</td>
<td>Type of security</td>
</tr>
<tr>
<td>48</td>
<td>SecurityID</td>
<td>Y</td>
<td>String</td>
<td>AlgoTrader Security ID</td>
</tr>
<tr>
<td>55</td>
<td>Symbol</td>
<td>Y</td>
<td>String</td>
<td>AlgoTrader Security Symbol</td>
</tr>
<tr>
<td>59</td>
<td>TimeInForce</td>
<td>Y</td>
<td>Char</td>
<td>How long an order remains active. Supported: 0, 1, 2, 3, 4, 6, 7</td>
</tr>
<tr>
<td>54</td>
<td>Side</td>
<td>Y</td>
<td>Char</td>
<td>1 or 2 for Buy/Sell</td>
</tr>
<tr>
<td>150</td>
<td>ExecType</td>
<td>Y</td>
<td>Char</td>
<td>Indicates the reason for sending this Execution Report</td>
</tr>
<tr>
<td>Tag #</td>
<td>Field Name</td>
<td>Required</td>
<td>Data type</td>
<td>Comments</td>
</tr>
<tr>
<td>-------</td>
<td>------------</td>
<td>----------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- 0: NEW, sent as a confirmation that the order has been accepted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- 6: PENDING CANCEL, received, but not processed yet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- 8: REJECTED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- E: PENDING REPLACE, received, but not processed yet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- I: ORDER STATUS, update</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- F: TRADE, fill</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- A: PENDING_NEW, received, but not processed yet</td>
</tr>
<tr>
<td>39</td>
<td>OrdStatus</td>
<td>Y</td>
<td>Char</td>
<td>Current order status</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- 0: NEW, order submitted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- 1: PARTIALLY FILLED, order partially executed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- 2: FILLED, order executed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- 4: CANCELED, order canceled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- 6: PENDING CANCEL, order will be cancelled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- 8: REJECTED, order rejected</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- A: PENDING_NEW, order received, but not submitted yet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- E: PENDING REPLACE, order will be replaced</td>
</tr>
<tr>
<td>58</td>
<td>Text</td>
<td>C</td>
<td>String</td>
<td>Additional information. Typically contains the details in case of rejection</td>
</tr>
</tbody>
</table>
Adapters

The following sections give a detailed overview of the different adapters available for AlgoTrader.

22.1. Fix Interface

AlgoTrader uses QuickFix/J\(^1\) for its Fix connections and currently supports FIX 4.2 and 4.4. Because FIX messages are not compatible between different versions, the two distinct services `Fix42OrderService` and `Fix44OrderService` exist. Incoming messages are handled by their corresponding `Fix42MessageHandler` and `Fix44MessageHandler`.

To configure a Fix trading connection the following steps have to be taken care of:

- Add the corresponding fix trading profile to the VM argument `spring.profiles.active` (e.g. `cNXFix`):

  ```
  -Dspring.profiles.active=live,pooledDataSource,cNXFix,embeddedBroker,html5,InfluxDB
  ```

- Add the fix session to `/algotrader/conf/src/main/resources/fix.cfg` (Use the `fix-template.cfg` file as basis - do not delete the default section):

  ```
  [session]
  SessionQualifier=CNXT
  BeginString=FIX.4.4
  SenderCompID=xxx
  TargetCompID=CNX
  SocketConnectHost=dret-fix-ssl.currenex.com
  SocketConnectPort=443
  SocketUseSSL=Y
  Username=xxx
  Password=xxx
  ValidateIncomingMessage=N
  ResetOnLogon=Y
  Inactive=Y
  ```

- Make sure there is an entry in the MySQL account table where the column `ORDER_SERVICE_TYPE` matches the type of the fix interface (e.g. `CNX_FIX`), the column `SESSION_QUALIFIER` matches the `SessionQualifier` specified in the file `fix.cfg` and the `ACTIVE` column is set to 1.

If market data is also received through a Fix interface the following items need to be added as well:

- Add the corresponding fix market data profile to the VM argument `spring.profiles.active` (e.g. `cNXFix`):

\(^1\) https://www.quickfixj.org/
Add the fix session to /algotrader/core/fix.cfg (an example file fix-template.cfg is provided in the same directory):

```plaintext
[session]
SessionQualifier=CNXMD
BeginString=FIX.4.4
SenderCompID=xxx
TargetCompID=CNX
SocketConnectHost=dret-fix-ssl.currenex.com
SocketConnectPort=443
SocketUseSSL=Y
Username=xxx
Password=xxx
ValidateIncomingMessage=N
ResetOnLogon=Y
Inactive=Y
```

When making subscriptions add the AdapterType corresponding to the Fix interface (e.g. CNX)

Important

Please make sure to have the setting `Inactive=Y` in both trading and market-data sections. Without this setting the fix session will be initialized before the remaining system has been fully initialized and might cause either trading or market data to malfunction.

For further information regarding QuickFix/J configuration please visit the QuickFix/J documentation

Per default Fix interfaces uses the following items to identify a particular instrument:

Options

- Exchange IB_CODE
- Option TRANSACTION_CURRENCY
- SecurityFamily SYMBOL_ROOT
- Option STRIKE
- Option TYPE
- Option EXPIRATION

---

2 https://www.quickfixj.org/usermanual/1.6.4//usage/configuration.html
Fix configuration

Option CONTRACT_SIZE

Future
  Future TRANSACTION_CURRENCY
  Exchange IB_CODE
  SecurityFamily SYMBOL_ROOT
  Future EXPIRATION
  Future CONTRACT_SIZE

Forex
  Forex TRANSACTION_CURRENCY
  Exchange IB_CODE
  Forex BASE_CURRENCY

Stock
  Stock TRANSACTION_CURRENCY
  Exchange IB_CODE
  Stock SYMBOL

Fund
  Fund TRANSACTION_CURRENCY
  Exchange IB_CODE
  Index SYMBOL

22.1.1. Fix configuration

All Fix configurations are stored in fix.cfg file by default.

The file fix-template.cfg contains default parameters suggested by AlgoTrader for all Fix sessions. The individual [session] blocks should be added after the [default] block.

```
[default]
ConnectionType=initiator
HeartBtInt=30
ReconnectInterval=5
FileStorePath=files/fix
FileLogPath=log
FileLogHeartbeats=N
FileIncludeMilliseconds=Y
```
Details of individual FIX sessions are expected to be provided by the brokerages.

For further information regarding QuickFix/J configuration please visit the QuickFix/J documentation\(^3\)

### 22.1.2. FIX logging

In addition to stock QuickFix/J configuration capabilities AlgoTrader provides a custom option to select a logging back-end out of those supported by QuickFix/J per individual session through custom `Logimpl` parameter.

Supported parameter values are:

- **file** (default)
  - Log to QuickFix/J standard file logger.
  
  The file option (which is the default) will create fix log files in the log sub-directory of the directory where AlgoTrader was started. One log file will be created for messages and one for QuickFix/J internal events. The name of both files contains the full FIX session name. To use the file logger the following settings are required within the `fix.cfg` file:

  ```
  FileLogPath=log
  FileLogHeartbeats=N
  FileIncludeMilliseconds=Y
  FileIncludeTimeStampForMessages=Y
  ```

- **none**
  - Disable logging. No FIX session events or messages will be logged.
  
  The none option might be especially useful for volume intensive market data sessions where persistent message log could be unnecessary or even excessive. Custom Fix logging options can be configured as follows:

  ```
  [session]
  SessionQualifier=FIXMD
  BeginString=FIX.4.4
  ...  
  LogImpl=none
  ```

- **screen**
  - Logs all QuickFix/J events and messages to the standard console logger.

\(^3\) [https://www.quickfixj.org/usermanual/1.6.4//usage/configuration.html](https://www.quickfixj.org/usermanual/1.6.4//usage/configuration.html)
slf4j

Log to the Simple Logging Facade for Java (SLF4J). Log entries will be committed to the logging back-end configured by SLF4J.

The slf4j allows log messages to be re-directed to the Log4J Chapter 29, Logging system which is highly configurable. Here is an example:

```
[session]
SessionQualifier=FIXMD
BeginString=FIX.4.4
...
LogImpl=slf4j
```

Log messages are logged to the following 4 loggers per default:

- `quickfixj.event` for regular events (e.g. sessions logging on)
- `quickfixj.errorEvent` for error events (e.g. connection issues)
- `quickfixj.msg.incoming` for incoming messages
- `quickfixj.msg.outgoing` for outgoing messages

A typical log statement would then look like this

```
```

It is also possible to customize the log categories per session for added flexibility. It is for example possible to log FIX messages from different FIX sessions into separate Files. The following example has two sessions, one for market data (FIXMD) and one for trading (FIXORD).

```
[session]
SessionQualifier=FIXMD
BeginString=FIX.4.4
...
LogImpl=slf4j
SLF4JLogIncomingMessageCategory=quickfixj.msg.md.incoming
SLF4JLogOutgoingMessageCategory=quickfixj.msg.md.outgoing
```

```
[session]
SessionQualifier=FIXORD
BeginString=FIX.4.4
...
LogImpl=slf4j
```
By adding the following sections to the log4j2.xml separate log files for market data (fix-md.log) and trading (fix-ord.log) will be created.

```xml
...<RollingFile
name="FixMD"
    fileName="log/fix-md.log"
    filePattern="log/fix-md-%d{MM-dd-yyyy}%i.log.gz">
    <PatternLayout pattern="%d %m %n"/>
    <Policies>
        <TimeBasedTriggeringPolicy />
        <SizeBasedTriggeringPolicy size="250 MB"/>
    </Policies>
</RollingFile>

<RollingFile
name="FixORD"
    fileName="log/fix-ord.log"
    filePattern="log/fix-ord-%d{MM-dd-yyyy}%i.log.gz">
    <PatternLayout pattern="%d %m %n"/>
    <Policies>
        <TimeBasedTriggeringPolicy />
        <SizeBasedTriggeringPolicy size="250 MB"/>
    </Policies>
</RollingFile>
...<logger name="quickfixj.msg.md" level="info" additivity="false">
    <AppenderRef ref="FixMD"/>
</logger>

<logger name="quickfixj.msg.ord" level="info" additivity="false">
    <AppenderRef ref="FixORD"/>
</logger>
```

For both files a daily rolling file scheme is used where at the end of a day all contents of each file will be zipped (e.g. fix-md-12-08-2019.log.gz). In addition if the file size reaches 250MB a new zip file will be created as well.

### 22.1.3. FIX message persistence

One can use standard QuickFix/J facilities to store FIX messages either in a local file or in a relational database through JDBC DataSource interface. For details please refer to the QuickFix/J documentation. For further details please visit the QuickFix/J documentation⁴

---

⁴ [https://www.quickfixj.org/usermanual/1.6.4/#usage/configuration.html#Storage](https://www.quickfixj.org/usermanual/1.6.4/#usage/configuration.html#Storage)
22.1.4. FIX Drop-copy support

LMAX and Trading Technologies interfaces provide support for so called drop-copy mode wherein the adapter can receive order status and fill messages from orders initiated externally (usually by external applications such as native trading front-ends). By default external fills get recorded as transactions of the SERVER strategy and allocated to the external account specified in the original execution report message. One, however, can provide a custom implementation of DropCopyAllocator interface in order to apply custom transaction allocation logic.

22.2. Crypto Exchange interfaces

AlgoTrader provides several crypto currency exchange adapters which are based on REST, Web Socket or REST.

When trading crypto currencies it is recommended to update the following properties inside conf.properties. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```
# the currency all portfolio balances will be calculated in
misc.portfolioBaseCurrency=BTC

# the number of digits all portfolio balances will be displayed with
misc.portfolioDigits=8
```

This will cause all balances to be displayed in BTC with a precision of 8 digits

22.2.1. Custom currency mapping

Not all exchanges use the same names for the same coins (e.g. BTC or XBT for Bitcoin). So that the system recognises it’s the same instrument and trades it properly across exchanges, there is a mapping algotrader/bootstrap/conf/src/main/resources/currency-code-mappings.csv. If the matching entry exists in that file (adapter code, exchange code and adapter currency code), then the defined AlgoTrader currency code will be used throughout the system and the adapter specific name will be used while communicating with the exchange.

22.2.2. Crypto-Order Constraints

All securities contain fields which describe valid order quantities and prices.

Crypto reference data services set these values provided this information is made available by the exchange.

The crypto-adapters validate orders versus these constraints and reject them before sending them to the exchanges/brokers.

The constraints are defined by:

- MinQty - minimum order quantity
- MaxQty - maximum order quantity
• QtyIncr - quantity increment
• MinPrice - minimum price
• MaxPrice - maximum price
• PriceIncr - price increment
• MinNotional - minimum notional value of the order. For example for limit order: Quantity * Limit price

Securities are updated automatically if there is a change on the exchange side. To change this behaviour (not to override those constraints), the user has to set the corresponding property inside the adapter specific property file (e.g. inside the conf-bmx.xml file for the BMX adapter). Alternatively the properties can be changed via Section 2.3, "VM Arguments":

```java
#{"type":"Boolean","label":"if true override existing security families fields..."}
bmx.overrideSecurityFamilies = true
```

Check the corresponding crypto-adapter sections below on what constraints are validated for each broker/exchange.

### 22.2.3. Supported Crypto-Order Types

The Following table contains valid order types per adapter

**Table 22.1. Order type constraints**

<table>
<thead>
<tr>
<th>Broker/Exchange</th>
<th>Exchange Trading</th>
<th>Margin Trading</th>
<th>Market</th>
<th>Limit</th>
<th>Stop</th>
<th>Stop Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binance</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Bitfinex</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>BitFlyer</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>BitMex</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>BitStamp</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Coinbase Pro</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Coingy</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

### 22.3. Session life-cycle events

All trading interface adapters generate session events, which enable the server engine as well as individual strategies to listen for and react to session events such as session being fully established or temporary loss of connectivity.
Specially when connecting to a slow Market Data adapters it might be necessary to listing to those session events and only subscribe for securities once the session is in the `LOGGED_ON` state.

### 22.4. Automatic order reconciliation after re-connect

Most crypto exchanges provide REST and/or Web Socket API's only. Unfortunately these protocols are stateless. So in case the AlgoTrader server gets disconnected from an exchange for some time or the AlgoTrader server is restarted, an active order may get cancelled or executed in the meantime.

For this purpose AlgoTrader provides an automatic order reconciliation feature. An automated order reconciliation process is invoked once the AlgoTrader server reconnects to synchronize the orders statuses between the exchange and the AlgoTrader server.

In case of trading adapters that use FIX interface, order reconciliation is provided automatically by the exchange FIX server. This is done using the `Resend` FIX protocol message. Unfortunately the `Resend` message is not supported by the Coinbase Pro exchange FIX server.

In case you want to turn off the AlgoTrader provided order reconciliation, add `noOrderReconciliation` into active Spring profiles list when starting the AlgoTrader server.

Please check the corresponding adapter section to see if the automatic order reconciliation is implemented for your exchange.

### 22.5. Bloomberg

The Bloomberg interface supports Market Data, Historical Data as well as Reference Data.

The Bloomberg interface provides both synchronous connections and asynchronous connections. Asynchronous connections are generally used for live market data whereas synchronous connections are used for retrieval of historical data as well as retrieval of reference data.

If market data is received through the Bloomberg interface the following items need to be added:

- Add the profile `bBMarketData` to the VM argument `spring.profiles.active`.

```java
class MyService {
    public void onSessionEvent(final SessionEventVO event) {
        switch (event.getState()) {
            case CONNECTED:
                // session connected but not yet authenticated
                break;
            case LOGGED_ON:
                // session connected and authenticated
                break;
            case DISCONNECTED:
                // session disconnected
                break;
            default:
                // default case
                break;
        }
    }
}
```
Dspring.profiles.active-live,pooledDataSource,bBMarketData,embeddedBroker,html5,InfluxDB

- When making subscriptions add the AdapterType BB

Bloomberg uses the BCGID field of the Security table to identify instruments.

For further details on the Bloomberg interface please visit the Bloomberg Open API

22.6. Currenex

The main features of the Currenex platform are

- Live tradeable rates
- Liquidity in all the major currency pairs
- Straight through processing of order executions

The Currenex implementation of the FIX/4.4 protocol has some peculiarities

- Requires additional acknowledgement TradingSessionStatus message indicating the trading session is fully initialized
- Supports only subset of standard Order Expiry (Time in Force) types
- Uses FOREX_MARKET (type C) and FOREX_LIMIT (type F) for Market and Limit orders

Currenex uses the columns Forex BASE_CURRENCY and SecurityFamily CURRENCY to identify an instrument.

22.7. DukasCopy

Algotrader supports both trading and market data connectivity via broker DukasCopy.

Since the DukasCopy FIX protocol implementation does not follow the FIX standard very closely, we at the moment don’t support order modifications via Dukascopy trading adapter. Modification of an open order can be achieved by cancelling and submitting a new order.

Note that Dukascopy does not support Stop Limit orders.

22.8. Exante (XNT)

Exante brokerage adapter, FIX 4.4 protocol based.

The adapter supports market data and trading.

---

5 https://www.bloomberg.com/professional/support/api-library/
Exante provides its customers with a demo platform with the same characteristics as the live platform. Market data in the Demo platform is delayed by at least 30 minutes. Live platform requires setting up an SSH tunnel, details for which are available at their website.

Reference data download is currently not supported by AlgoTrader. Individual securities may be enabled on AlgoTrader side for use with Exante by setting their XNTID database field with the correct Exante instrument code.

22.9. EzeSoft / Real Tick

EzeSoft / RealTick provides connectivity to about 30 institutional and 10 retail brokers.

The EzeSoft / RealTick Fix interface currently supports only Order Processing.

The Fix Implementation of EzeSoft / RealTick is well conforming with the Fix Standard no customizations had to be made.

The Fix interface uses standard Fix instrument definitions mentioned at the end of section Section 22.1, “Fix Interface”.

22.10. Fortex

Fortex uses almost vanilla Fix/4.4 protocol with a very customizations. It supports FX only.

- Supports GTC, IOC and FOK time-in-force parameters only
- Requires filled quantity to be included in order cancellation messages

Fortex uses the columns Forex BASE_CURRENCY and SecurityFamily CURRENCY to identify an instrument.

22.11. FXCM

FXCM interface FIX/4.4 protocol does not deviate much from the standard but has some peculiarities about the way FIX sessions are established

- Unlike many other FIX connectivity providers who provide separate FIX sessions for market data and trading interfaces FXCM by default offers one session for both market data feed and trading operations
- Uses extra UserRequest / UserResponse message exchange to authenticate the user and to fully initialize the session

FXCM uses the columns Forex BASE_CURRENCY and SecurityFamily CURRENCY to identify an instrument.

22.12. IB Native Interface

The native IB Interface connects to the local Trader Workstation (TWS) or IB Gateway instance and uses methods supplied by the IB client. The interface is fully capable of handling IB’s Financial Advisor functionality like Sub Accounts, Account Groups and Allocation Profiles.

The IB interface supports Market Data, Historical Data, Order Processing, Retrieval of account information as well as Reference Data.
Note
You get market data for a minimum of 100 instruments with subscriptions (depends on your commissions and assets with IB). You can buy up to 10 quote boosters for USD 30 each, which provide 100 additional instruments each (max 1000). For details, consult the IB market data fees.6

Similar restrictions/extensions exist for historical data. Those details can be viewed here.7

To configure an IB connection the following steps have to be taken care of:

- Add the profile iBNative to the VM argument spring.profiles.active:'

```
-Dspring.profiles.active=live,pooledDataSource,iBNative,embeddedBroker,html5,InfluxDB
```

- Make sure there is an entry in the account database where the column ORDER_SERVICE_TYPE is set to IB_NATIVE).

If market data is also received through the IB interface the following items need to be added as well:

- Add the profile iBMarketData to the VM argument spring.profiles.active:

```
-Dspring.profiles.active=live,pooledDataSource,iBMarketData,embeddedBroker,html5,InfluxDB
```

- When making subscriptions add the AdapterType IB

The IB interface has the following options to identify a particular instrument:

- CONID specified in the security table

- Use instrument symbols and additional data depending on the instrument type:

   Options
   
   Exchange IB_CODE
   SecurityFamily CURRENCY
   SecurityFamily SYMBOL_ROOT
   Option STRIKE
   Option TYPE
   Option EXPIRATION

7 http://interactivebrokers.github.io/tws-api/historical_limitations.html#hd_availability
SecurityFamily CONTRACT_SIZE

Future
  SecurityFamily CURRENCY
  Exchange IB_CODE
  SecurityFamily SYMBOL_ROOT
  Future EXPIRATION
  SecurityFamily CONTRACT_SIZE

Stock
  SecurityFamily CURRENCY
  Exchange IB_CODE
  Stock SYMBOL

Index
  SecurityFamily CURRENCY
  Exchange IB_CODE
  Index SYMBOL

Combination
  SecurityFamily CURRENCY
  Exchange IB_CODE
  SecurityFamily BASE_SYMBOL
  Security CONID of each Component
  Component QUANTITY

In addition the following items apply to the IB Native interface

- The IB Native interface uses the \textit{RT\_VOLUME} events to process incoming trade events
- The IB Native interface propagates daily \texttt{OPEN} and \texttt{CLOSE} prices to strategies in case the following property inside \texttt{conf-ib.properties} is enabled. Alternatively the properties can be changed via \texttt{Section 2.3, “VM Arguments”}

```plaintext
# enables emission of generic open and close ticks
ib.emitOpenClose = true
```

\footnote{https://interactivebrokers.github.io/tws-api/tick_types.html\#rt\_volume}
• The IB Native interface propagates VWAP prices to strategies in case the following property inside `conf-ib.properties` is enabled. Alternatively the properties can be changed via Section 2.3, “VM Arguments”

```java
# enables emission of generic VWAP ticks
ib.emitVWAP = true
```

• The IB Native interface expects orders to be sent with their order ids in ascending order. The Class `IBOrderIdSynchronizer` is responsible to make sure order ids are actually in ascending order. In case an order id is skipped the `IBOrderIdSynchronizer` will wait for up to `maxOrderSyncTime` milliseconds for the order with the correct order id to arrive.

• The IB Native interface supports trading of tradeable / non-synthetic combinations by placing BAG orders through the IB interface.

• The IB Native interface reports `volBid`, `volAsk` and `vol` in lots of 100 contracts for US equities. Please see the following page for further details on handling of Odd Lot Orders

For further details on the IB native interface please visit the IB API Reference Guide

### 22.12.1. IB Market Data Subscriptions

In the traditional financial sector (excluding cryptocurrencies) market data is not free and requires market data subscriptions.

IB provides free 15min delayed data. It is possible to obtain this free 15min delayed data when logged in via a trial account. This however is available only when logged in via Trader Workstation (TWS). Please see Section 22.12.2, “Delayed IB Market Data”

Market data can be accessed both through the IB paper trading account as well as the live trading account.

---

**Note**

- The paper trading account has one single username assigned to it. The live trading account can have multiple user names.

- For each username (live account & paper trading account) only one session can exist at the same time. If you login with the same username on a different machine the other session will get logged out.

- If the live account username (that is sharing its market data subscription with the paper trading account) is currently logged in, the paper trading account doesn't get market data until the live account is again logged out.

---

*9 https://ibkr.info/node/1062
10 https://www.interactivebrokers.com/en/software/api/api.htm*
To get a market data subscriptions one has to login to the live trading account. Then follow these steps:

1. Select Settings / User Settings in the menu on the left. Then select Market Data Subscriptions on the right
2. Then click on the icon next to Current Subscriptions
3. Then select the region (e.g. North America)
4. On the next screen individual market data subscriptions can be selected

Figure 22.1. Market Data Subscriptions 1

---

11 https://gdcdyn.interactivebrokers.com/sso/Login
Typical market data subscriptions are:

- **IDEAL FX**: free Forex market data
- **NASDAQ (Network C/UTP)**: live market data for NASDAQ listed equities
- **NYSE (Network A/CTA)**: live market data for NYSE listed equities
- **US Securities Snapshot and Futures Value Bundle**: live market data for US futures and snapshot data for US equities (AT cannot process snapshot data, so in addition NASDAQ and NYSE has to be subscribed as well)
Figure 22.4. Market Data Subscriptions 4

To use these market data subscriptions through the paper trading account follow these steps:

1. Select Settings / User Settings in the menu on the left. Then select Paper Trading Account on the right

2. Then select Yes next to Share real-time market data subscriptions with paper trading account

3. Then Select the username whose market data you want to share. This will share the market data subscriptions of the live account with the paper trading account.
Figure 22.5. Paper Trading Account 1

Figure 22.6. Paper Trading Account 2

Note

In case no market data arrives through the IB interface it is usually best to login to InteractiveBrokers Trader Workstation (TWS) as there are usually warning messages that indicate what might be the issue.

22.12.2. Delayed IB Market Data

There are several prerequisites in order to use the free delayed Market Data from IB:

- TWS needs to be used. IB Gateway is not supported for free delayed market data
• Delayed market data subscriptions are available with trial accounts only. After starting the TWS, simply go to Return to the demo and enter your email address (see screen shot below)

• Similarly to IB Gateway, TWS should be configured to work with AlgoTrader: After logged in, go to Edit -> Global Configuration... -> API -> Settings. Make sure following settings are used: Enable ActiveX and Socket Clients should be enabled, Read-Only API should be disabled, Socket port should be set to 4001.

• The ib.pricefeed.allowDelayedMarketData property should be set to true (it is false by default)

![Login](image)

Figure 22.7. Delayed IB Market Data

22.13. IB Fix Interface

The IB Fix Interface provides the same Order Management features as the IB Native Interface. However Market Data is not available through this interface.

The interface is fully capable of handling IB's Financial Advisor functionality like Sub Accounts, Account Groups and Allocation Profiles.

For further details on the IB Fix interface please visit the [IB FIX/CTCI Users' Guide](https://www.interactivebrokers.com/en/index.php?f=4988)

The IB Fix interface uses standard Fix instrument definitions mentioned at the end of section Section 22.1, “Fix Interface”.

---

22.14. JP Morgan

The JP Morgan Fix interface supports Order Processing only.

As the JP Morgan Fix Implementation is well conforming with the Fix Standard no customizations had to be made.

The JP Morgan Fix interface uses standard Fix instrument definitions mentioned at the end of section Section 22.1, “Fix Interface”.

22.15. LMAX

Supports only a limited number of securities, mainly Forex.

LMAX implementation of the FIX/4.4 protocol has some peculiarities:

- Uses predefined contract modifiers for market data events and order quantities. The contract modifiers are not included in FIX messages and have to be applied by the interface adaptor.
- Uses custom message dictionary.
- Supports only IOC and FOK time-in-force parameters for market orders.
- Supports DAY, GTC, IOC and FOK time-in-force parameters for limit orders.
- Supports only DAY and GTC time-in-force parameters for stop orders.
- Supports trading status signaling temporary suspension and resumption in trading of individual securities.

LMAX uses the column LMAXID of the security table to identify an instrument.

22.16. Nexus Prime

Nexus Prime is a MetaTrader MT4 FIX interface provided by IS Risk Analytics. The Nexus Prime interface uses Fix 4.4 and it supports FX only. Due to the underlying MetaTrader MT4 a few limitations apply.

- Market Data subscriptions cannot be cancelled.
- Orders cannot be modified, instead one needs to cancel the current order first and then resend a new one.
- Buy limit orders need to be placed below the market price. Sell limit orders need to be placed above the market price.
- Buy stop orders need to be placed above the market price. Sell stop orders need to be placed below the market price.

13 https://www.isriskanalytics.com/trading-technology/
• Minimum trade size allowed on most currency pairs is .01 lots which is 1000 notional
Nexus Prime uses the columns Forex BASE_CURRENCY and SecurityFamily CURRENCY to identify an instrument.

22.17. PrimeXM

The PrimeXM FIX/4.4 interface implementation follows the Fix Standard closely, but uses MassQuote messages for conveying the market data. Each MassQuote message has to be acknowledged by the FIX client.

Only Forex instruments are supported by the PrimeXM Fix Interface.

Market, limit and stop orders are supported but only with time in force IOC (Instant or Cancel) or FOK (Fill or Kill).

Order modifications are not supported.

22.18. Quandl

Quandl is a public service that provides a wide range of financial, economic and alternative data. It is mostly end of day data but also some intra-day (e.g. hourly) data. To find out if they have what you are looking for, check their data products page. AlgoTrader allows downloading historical data from Quandl. For more information about Quandl please have a look at the Quandl Docs/Help.

Data on Quandl is divided into databases. Each database contains multiple datasets. For instance EOD database contains end-of-day data for all publicly-traded US stocks. Each database/dataset pair is uniquely identified by database_code/dataset_code pair. For instance EOD/AAPL is the globally unique code for the AAPL stock dataset within the EOD database. The Quandl database browser can be used to find suitable databases for desired instrument type, region and data type.

The QdlHistoricalDataService is integrated with the AlgoTrader Historical Data Download and needs to be enabled by specifying the qdlHistoricalData Spring profile (see section Section 18.3, “Historical Data Download”). The QdlHistoricalDataService transforms retrieved Quandl data into AlgoTrader bars. Transformation rules between the Quandl data format and AlgoTrader Bar format are defined in the file quandl.yml. By default the file quandl.yml already contains the transformation rules for most commonly used Quandl databases. Additional transformation rules can be added to the file as needed:

```yaml
EOD:
  barSize: DAY_1
  columnMapping:
    dateTime: Date
    open: Open
    high: High
```

14 https://www.quandl.com/
15 https://www.quandl.com/search?query=
16 https://www.quandl.com/docs-and-help
17 https://www.quandl.com/search
The Quandl database code

barSize supported by the Quandl database (e.g. DAY_1 or MIN_1)

Column mappings between Quandl data fields and AlgoTrader BarVO fields

The relevant properties for the Quandl adapter are defined inside the file conf-qdl.properties where they can be changed. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```java
#{"type":"String","label":"API Key"}
qdl.apiKey = ATVxxxxxxxxxxxxx
```

To use the QdlHistoricalDataService please replace the property qdl.apiKey with the API Key that can be retrieved through the Quandl Account Settings.

In terms of historical data download a mapping between the Quandl database and the Security entity is defined by the quandl_database field in the security table. Similarly a mapping between the Quandl dataset and the Security entity is defined by the quandl_dataset field in the security table. AlgoTrader sample data files (samples/db/mysql/mysql-data.sql and samples/db/mysql/h2-data.sql) already contain quandl_database/quandl_dataset values for all sample security families and most sample securities.

### 22.19. QuantHouse

The QuantHouse adapter is based on the QuantHouse ultra low latency market data feed QuantFEED. The QuantHouse adapter supports live Market Data.

If market data is received through the QuantHouse interface the following items need to be added:

- Add the profile qHMarketData to the VM argument spring.profiles.active:

```java
Dspring.profiles.active=live,pooledDataSource,qHMarketData,embeddedBroker,html5,InfluxDB
```

- When making subscriptions add the AdapterType QH

QuantHouse uses the Exchange MIC and Security SYMBOL fields to identify instruments.

For further details on the QuantHouse interface please contact [QuantHouse](https://www.quanthouse.com/)

### 22.20. SocGen

The SocGen FIX/4.2 interface supports Order Processing only.

---

18 [https://www.quanthouse.com/](https://www.quanthouse.com/)
The SocGen Fix Implementation follows the Fix Standard closely, but some minor customizations according to the 'SocGen FIX Rules of Engagement' had to be made. Additionally exchange specific restrictions rules defining the allowed order type / TIF combinations were added.

Only Future instrument orders are supported by the SocGen Fix Interface.

### 22.21. Trading Technologies (TT)

Supports a wide range of future and option contracts tradeable at multiple venues / exchanges.

- TT uses the column TTID of the security table to identify instruments
- Provides a reference data service that can be used to download contract definitions
- Supports drop-copy sessions

The relevant properties for TT adapter are defined inside the file `conf-tt.properties` where they can be changed. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```json
"type":"String","label":"market request user name"
tt.username = xxx
```

The `tt.username` property defines the username on the Trading Technologies side

### 22.22. UBS

The UBS Fix interface supports Order Processing for futures and options only.

As the UBS Fix Implementation is well conforming with the Fix Standard no customizations had to be made

The UBS Fix interface uses standard Fix instrument definitions mentioned at the end of section Section 22.1, “Fix Interface”.

### 22.23. Binance

Binance\(^{19}\) is a cryptocurrency exchange. Please see the API reference\(^{20}\) page for the technical details.

Binance provides Java library for interacting with Binance API. It supports REST requests to endpoint providing orders functionality, account data and reference data. Support for market data is done using WebSocket API.

The relevant properties for the Binance adapter are defined inside the file `conf-bnc.properties` where they can be changed. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```json
"type":"String","label":"API Key"
bnc.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

---

\(^{19}\) https://www.binance.com/
\(^{20}\) https://github.com/binance-exchange/binance-java-api
A Binance account is necessary in order to use Binance adapter. Unique apiKey and apiSecret settings must be set to the actual values (either in the properties file or by setting a VM argument).

Note

Binance is very time sensitive, i.e. if your computer is ahead of the Binance system clock, the API might reject your orders with an exception similar to

```java
com.binance.api.client.exception.BinanceApiException: Timestamp for this request was 1000ms ahead of the server's time
```

To prevent these issues, we suggest synchronizing your system clock with an internet reference time using e.g. this time sync tool\(^2\)

22.23.1. Binance Order Constraints

Binance does not support margin trading.

AlgoTrader currently support market, limit and stop limit orders on Binance.

Note that Binance has restrictions to the amount of (algo) orders that can be placed on an instrument or exchange. See the Binance API filter page\(^2\) for details.

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>minimum order quantity</td>
</tr>
<tr>
<td>MaxQty</td>
<td>maximum order quantity</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>quantity increment</td>
</tr>
<tr>
<td>MinPrice</td>
<td>minimum price</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>maximum price</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>price increment</td>
</tr>
</tbody>
</table>

Automatic order reconciliation (on WebSocket reconnect) is implemented for Binance.

### 22.23.2. Binance Account Management

**Table 22.3. Supported Functionality**

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>It is possible to withdraw BTC, ETH, LTC, NEO and BNB. The withdrawal address has to be provided in withdrawContext. A secondary address identifier for coins like XRP, XMR etc has to be provided as paymentId in the withdrawContext. The address description is optional.</td>
</tr>
<tr>
<td>Deposit Addresses</td>
<td>It is possible to deposit BTC, ETH, LTC, NEO and BNB. Please check <a href="https://binance.zendesk.com/signin?return_to=https%3A%2F%2Fbinance.zendesk.com%2Fhc%2Fen-us%2Farticles%2F115000622212-How-to-Register-and-Deposit-on-Binance">How to Register and Deposit on Binance</a> on how to query the corresponding addresses</td>
</tr>
<tr>
<td>Account Events</td>
<td>Supported</td>
</tr>
</tbody>
</table>

### 22.24. Bitfinex

*Bitfinex* is a cryptocurrency exchange. The Bitfinex adapter provides order execution, market data, reference data and account data functionality. Please see the [Bitfinex API reference](https://docs.bitfinex.com/docs) page for technical details about the supported features.

The relevant properties for the Bitfinex adapter are defined inside the file `conf-bfx.properties` where they can be changed. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```java
#{"type":"String","label":"API Key"}
bfx.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"API Secret"}
bfx.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"Integer","label":"REST API Rate Limit Milliseconds"}
bfx.rateLimit = 1500

#{"type":"Integer","label":"API-level constant'}
bfx.scale = 5
```

---


25 [https://www.bitfinex.com/](https://www.bitfinex.com/)

26 [https://docs.bitfinex.com/docs](https://docs.bitfinex.com/docs)
A Bitfinex account is necessary in order to use Bitfinex adapter. Unique apiKey and apiSecret settings must be set to the actual values (either in the properties file or by setting a VM argument).

### 22.24.1. Bitfinex Order Constraints

AlgoTrader supports Bitfinex exchange and margin trading.

Market, limit and stop orders are supported. For exchange account trading, different order types are used (exchange market, exchange limit and exchange stop).

#### Table 22.4. Bitfinex constraints

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>minimum order quantity</td>
</tr>
<tr>
<td>MaxQty</td>
<td>maximum order quantity</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>0.00000001</td>
</tr>
<tr>
<td>MinPrice</td>
<td>-</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>-</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>price precision</td>
</tr>
<tr>
<td>MinNotional</td>
<td>-</td>
</tr>
</tbody>
</table>

Details

For details see: [Bitfinex](https://api.bitfinex.com/v1/symbols_details)

Automatic order reconciliation (on WebSocket reconnect) is implemented for Bitfinex.

### 22.24.2. Bitfinex Account Management

#### Table 22.5. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>AlgoTrader uses the v1 API. Supported currencies can be found <a href="https://docs.bitfinex.com/v1/reference#rest-auth-withdrawal">here</a>. The withdrawal address and payment id have to be provided in the withdrawContext. Bitfinex supports 3 types of wallets: trading exchange and deposit. The default is exchange. It can be changed in conf-bfx.properties file.</td>
</tr>
<tr>
<td>Deposit</td>
<td>The available currencies can be found <a href="https://api.bitfinex.com/v2/conf/pub:map:currency:label">here</a>. Bitfinex supports three types of wallets: trading, exchange and deposit. Note that the USDT deposit method only works for verified accounts, otherwise this method throws an exception.</td>
</tr>
</tbody>
</table>

---

27 [https://api.bitfinex.com/v1/symbols_details](https://api.bitfinex.com/v1/symbols_details)
28 [https://docs.bitfinex.com/v1/reference#rest-auth-withdrawal](https://docs.bitfinex.com/v1/reference#rest-auth-withdrawal)
22.25. Bitflyer

Bitflyer is a cryptocurrency exchange. The Bitflyer adapter supports order execution, market data, reference data and account data functionality. Please see the Bitflyer API reference page for technical details.

**Note**

At this point (April 2018), Bitflyer does not yet support cross-border trading, so trading vs. USD is only possible with a US account.

The relevant properties for the Bitflyer adapter are defined inside the file `conf-bfl.properties` where they can be changed. Alternatively the properties can be changed via Section 2.3, "VM Arguments":

```java
#{"type":"String","label":"API Key"}
  bfl.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"API Secret"}
  bfl.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"PubNub Subscribe Key"}
  bfl.pubNubSubscribeKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"Integer","label":"REST API Rate Limit Milliseconds"}
  bfl.rateLimit = 1500

#{"type":"Boolean","label":"if true import all currencies, otherwise only those defined in ch.algotrader.enumeration.Currency"}
  bfl.importAllPairs = true
```

A Bitflyer account is necessary in order to use Bitflyer adapter. Unique apiKey, apiSecret as well as market data subscription key settings must be set to the actual values (either in the properties file or by setting a VM argument).

22.25.1. Bitflyer Order Constraints

We support Bitflyer margin and exchange trading.

Bitflyer uses different instruments for exchange and margin trading (see Bitflyer margin trading).
• **Exchange:** BTCJPY with security.DESCRIPTION BTC/JPY@FLYR

• **Margin:** BTCJPY with security.DESCRIPTION BTC/JPY-FX@FLYR

Bitflyer supports market and limit orders.

**Note**

At this point (July 2018), Bitflyer does not support order modifications.

### Table 22.6. BitFlyer constraints

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>BTC/JPY: 0.001</td>
</tr>
<tr>
<td></td>
<td>ETH/BTC: 0.01</td>
</tr>
<tr>
<td></td>
<td>FX: BTC/JPY: 0.01</td>
</tr>
<tr>
<td>MaxQty</td>
<td>-</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>0.00000001</td>
</tr>
<tr>
<td>MinPrice</td>
<td>-</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>-</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>1 JPY - if transaction Currency is JPY</td>
</tr>
<tr>
<td></td>
<td>0.000001 BTC - if transaction Currency is BTC</td>
</tr>
<tr>
<td>MinNotional</td>
<td>-</td>
</tr>
<tr>
<td>Details</td>
<td>The BitFlyer limits are described in FAQ: BitFlyer amounts[^33]</td>
</tr>
</tbody>
</table>


Automatic order reconciliation (on WebSocket reconnect) is implemented for Bitflyer.

### 22.25.2. Bitflyer Account Management

### Table 22.7. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>It is possible to withdraw JPY for Japanese accounts, USD for U.S. accounts, and EUR for European accounts. The bank account id has to be provided in withdrawContext as address parameter.</td>
</tr>
<tr>
<td>Deposit</td>
<td>Bitflyer supports BTC, ETH, LTC, BCH, MONA, LSK. Before an address is returned, it needs to be enabled/created under the Account Funding section after logging in to the BitFlyer Web UI.</td>
</tr>
</tbody>
</table>

22.26. BitMEX

*BitMEX*[^34] is a cryptocurrency futures exchange. The BitMEX adapter provides order execution, market data, reference data and account data functionality through REST and WebSocket API. Please see the API reference[^35] page for technical details about the supported features.

The relevant properties for the BitMEX adapter are defined inside the file `conf-bmx.properties` where they can be changed. Alternatively the properties can be changed via `Section 2.3, "VM Arguments"`:

```java
#{"type":"String","label":"API Key"}
bmx.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"API Secret"}
bmx.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"Integer","label":"REST API Rate Limit Milliseconds"}
bmx.rateLimit = 1000

#{"type":"Integer","label":"API-level constant’}
bmx.balanceScale = 8
```

A BitMEX account is necessary in order to use the BitMEX adapter. Unique `apiKey` and `apiSecret` settings must be set to the actual values (either in the properties file or by setting a VM argument).

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Future</td>
<td>Regular future contracts expiring every 3 months. Please note that the XBT (BTC) contracts have a variable contract size. For more information please see [<em>Futures Guide</em>][36].</td>
</tr>
<tr>
<td>PerpetualSwap</td>
<td>Bitcoin perpetual contract XBTUSD (BTCUSD) are represented by this class. For more information please see [<em>Perpetual Contract Specification</em>][37].</td>
</tr>
<tr>
<td>Index</td>
<td>For a complete list of supported indices please see [<em>Indices</em>][38].</td>
</tr>
</tbody>
</table>

BitMEX supports trading of the perpetual contract and of the futures.

The minimum quantity for all contracts is 1 contract (lot size = 1). Only integer number of contracts are allowed. The `QUANTITY_SCALE` for all securities is set to 0 and must not be changed.

[^34]: https://www.bitmex.com/
[^35]: https://www.bitmex.com/app/apiOverview
[^36]: https://www.bitmex.com/app/futuresGuide
[^37]: https://www.bitmex.com/app/contract/XBTUSD
[^38]: https://www.bitmex.com/app/index/BXBT
Placing an order to buy one XBTUSD means buying the amount of Bitcoin worth 1 USD. For more information please consult the BitMEX perpetual contract details page.\(^{39}\).

### 22.26.1. BitMex Order Constraints

BitMex is a Futures exchange, so only margin trading is supported.

AlgoTrader support market, limit, stop and stop limit orders.

**Table 22.9. BitMex constraints**

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>1</td>
</tr>
<tr>
<td>MaxQty</td>
<td>-</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>1</td>
</tr>
<tr>
<td>MinPrice</td>
<td>1 Satoshi</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>-</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>1 Satoshi</td>
</tr>
<tr>
<td>MinNotional</td>
<td>-</td>
</tr>
</tbody>
</table>

Automatic order reconciliation (on WebSocket reconnect) is implemented for BitMex.

### 22.26.2. BitMex Account Management

**Table 22.10. Supported Functionality**

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>Only BTC is supported</td>
</tr>
<tr>
<td>Deposit</td>
<td>Only BTC is supported</td>
</tr>
<tr>
<td>Addresses</td>
<td>Supported</td>
</tr>
<tr>
<td>Account Events</td>
<td>Supported</td>
</tr>
</tbody>
</table>

### 22.27. Bitstamp

*Bitstamp*\(^{40}\) is a cryptocurrency exchange. Please see the API reference\(^{41}\) page for the technical details.

Order and market data related functionality is provided via FIX/4.4 protocol. Account data and reference data is provided via REST API.

---

\(^{39}\) [https://www.bitmex.com/app/seriesGuide/XBT](https://www.bitmex.com/app/seriesGuide/XBT)

\(^{40}\) [https://www.bitstamp.net/](https://www.bitstamp.net/)

\(^{41}\) [https://www.bitstamp.net/api/](https://www.bitstamp.net/api/)
Bitstamp FIX/4.4 interface follows the standard closely, but offers only one session for both market data feed and trading operations. Bitstamp market data supports only limited number of cryptocurrency (Forex) securities. Order modifications are not supported. For more information about the Bitstamp FIX specification please have a look at the Bitstamp public FIX interface.

The relevant properties for the Bitstamp adapter are defined inside the file conf-bts.properties where they can be changed. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```java
#{"type":"String","label":"API Key"}
bts.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"API Secret"}
bts.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"Customer ID"}
bts.customerId = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"Integer","label":"REST API Rate Limit Milliseconds"}
bts.rateLimit = 1500

#{"type":"Boolean","label":"if true import all currencies, otherwise only those defined in ch.algotrader.enumeration.Currency"}
bts.importAllPairs = true
```

A Bitstamp account is necessary in order to use Bitstamp adapter. Unique apiKey and apiSecret settings must be set to the actual values (either in the properties file or by setting a VM argument)

### 22.27.1. Bitstamp Order Constraints

Bitstamp does not support margin trading.

AlgoTrader supports market and limit orders.

At the time of writing this, Bitstamp does not support order modifications.

#### Table 22.11. BitStamp constraints

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>0.00000001</td>
</tr>
<tr>
<td>MaxQty</td>
<td>-</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>0.00000001</td>
</tr>
<tr>
<td>MinPrice</td>
<td>0.00001</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>-</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>0.00001</td>
</tr>
</tbody>
</table>

---

42 [https://www.bitstamp.net/fix/](https://www.bitstamp.net/fix/)
<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinNotional</td>
<td>5 USD</td>
</tr>
<tr>
<td>Details</td>
<td>There is one additional rule for BitStamp: the order value should be at least USD 5. The USD 5 value is calculated by BitStamp with the latest market prices. This is however not validated on the AlgoTrader side. For further details see BitStamp limits.</td>
</tr>
</tbody>
</table>

Automatic order reconciliation is provided automatically by Bitstamp's FIX server.

### 22.27.2. Bitstamp Account Management

#### Table 22.12. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>Support for LTC, ETH, BTC, BCH, XRP. The withdrawal address has to be provided in the withdrawContext.</td>
</tr>
<tr>
<td>Deposit Addresses</td>
<td>Support for LTC, ETH, BTC, BCH, XRP</td>
</tr>
<tr>
<td>Account Events</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

### 22.28. CoinAPI

CoinAPI[^44] is a market data gateway to multiple crypto exchanges. CoinAPI provides historical and live market data. It also provides reference data for the supported instruments, however it doesn't provide trading related functionality. Please see the API reference[^45] page for the technical details.

Historical data is available down to 1 second bars. Historical data availability varies by currency. Up to 100 daily requests can be placed for free. Consult their pricing[^46] if you require more.

Instruments and exchanges must have CNPID value setup in security and exchange database tables.

The relevant properties for the CoinAPI adapter are defined inside the file `conf-cnp.properties` where they can be changed. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```java
#{"type":"String","label":"API Key"
  cnp.apiKey = XXXXXXXXXXXXXXXX

#{"type":"Integer","label":"REST API Rate Limit Milliseconds"
  cnp.ratelimut = 1500
```

[^43]: https://www.bitstamp.net/article/bitstamp-minimum-trade-changing-to-5/
[^44]: https://www.coinapi.io/
[^45]: https://docs.coinapi.io/
[^46]: https://www.coinapi.io/pricing
Unique apiKey and apiSecret settings must be set to the actual values (either in the properties file or by setting a VM argument)

22.29. Coinbase Pro

Coinbase is a cryptocurrency exchange. The Coinbase Pro adapter supports order execution, market data, reference data and account data functionality. Please see the Coinbase API reference page for technical details.

Note
Currently Coinbase Pro does not yet support cross-border trading, so trading vs. USD is only possible with a US account.

The relevant properties for the Coinbase Pro adapter are defined inside the file conf-cnb.properties where they can be changed. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```java
#cnp_SCALE = 5
#cnp.importAllPairs = true
#cnp.websocketUpdates = trade,quote

#cnp.webSocketUrl = wss://ws-feed.pro.coinbase.com
#cnp.webSocketTimeoutSeconds = 10
#cnp.apiKey = XXX
#cnp.apiSecret = XXX
#cnp.passphrase = XXX
```

47 https://pro.coinbase.com
48 https://docs.pro.coinbase.com
Coinbase Pro Order Constraints

---

A Coinbase Pro account is necessary in order to use Coinbase Pro adapter, but not needed for market data. Coinbase Pro provides a sandbox/testing environment, both for the public website (Sandbox website url) and the exchange. You'll need to generate (via the Coinbase website) and configure (see properties above) an API key and API secret in order to use the Coinbase Pro Adapter. The sandbox/testing environment has different URLs (see commented out values `cnb.webSocketUrl, cnb.restUrl` above).

### 22.29.1. Coinbase Pro Order Constraints

Coinbase Pro does not support margin trading.

AlgoTrader supports market, limit and stop limit orders.

---

**Note**

Coinbase Pro does not currently support order modifications.

---

**Table 22.13. Coinbase Pro constraints**

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>0.001 BTC</td>
</tr>
<tr>
<td></td>
<td>0.01 BCH</td>
</tr>
<tr>
<td></td>
<td>0.01 ETH</td>
</tr>
<tr>
<td></td>
<td>0.1 LTC</td>
</tr>
<tr>
<td>MaxQty</td>
<td>-</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>BaseMinSize scale</td>
</tr>
<tr>
<td>MinPrice</td>
<td>-</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>-</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>Quote Increment</td>
</tr>
<tr>
<td>MinNotional</td>
<td>-</td>
</tr>
</tbody>
</table>

---

49 https://public.sandbox.pro.coinbase.com
Automatic order reconciliation (on FIX reconnect) is implemented for Binance (required, as the Coinbase Pro FIX server does not support the Resend message).

22.29.2. Coinbase Pro Account Management

Table 22.14. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>Support for all cryptocurrencies</td>
</tr>
<tr>
<td>Deposit</td>
<td>Support for all cryptocurrencies. Note that the deposit address returned is for an account in Coinbase (non-PRO). The transfer to Coinbase Pro (system for trading) needs to be done manually via the Coinbase Pro website.</td>
</tr>
<tr>
<td>Account Events</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

22.30. Coinigy

Coinigy provides connectivity to 45+ of most popular cryptocurrency exchanges allowing to trade hundreds of different crypto currencies. The Coinigy Interface connects to the Coinigy API endpoints via REST and Socket Cluster protocols.

The Coinigy interface supports Market Data, Order Processing, Retrieval of account information as well as Reference Data.

Coinigy uses the columns Security CNGID and Exchange CNGID to identify an instrument.

For further details on the Coinigy interface please visit the Coinigy API Documentation.

22.30.1. Setup Instructions

To setup a connection to Coinigy the following steps have to be taken:

- Sign-up for a Coinigy account on Coinigy Sign up
- Enable two factor authentication (2FA) on the account following the 2FA Instructions
- In the API accounts settings add the API keys from all of the exchanges where an account is setup according to these Instructions

---

50 https://support.pro.coinbase.com/customer/en/portal/articles/2945311-limits
51 https://coinigy.docs.apiary.io
52 https://www.coinigy.com/auth/signup
In the account preferences generate a new Coinigy API key and Secret Key set it inside `conf-cng.properties`.

In the account preferences click the button 'Click to reveal my Private Channel ID (WebSocket API)' and set the Private Channel ID inside `conf-cng.properties`.

The relevant properties for the Coinigy adapter are defined inside the file `conf-cng.properties` where they can be changed. Alternatively the properties can be changed via Section 2.3, "VM Arguments":

```java
#cng.wssPrivateChannel = XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
#cng.apiKey = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
#cng.apiSecret = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
#cng.reverseExchanges = BITS,BTCC,PLNX
#cng.defaultExchangeCodes = PLNX,BITF,KRKN,GDAX,BTCE,OK,BTRX,BT38,BITS,HUOB
# default exchanges/adapter types to use for market data.
# misc.defaultMarketFeeds=BMEX:CNG,OKEX:CNG,BINA:CNG
```

In order to populate the database with Coinigy Accounts, Exchanges, Security Families and Securities run the `ReferenceDataStarter` with `cNGReferenceData` spring profile enabled and program argument: `all`. For further details please visit Chapter 19, Reference Data.

### 22.30.2. Coinigy Order Constraints

Currently the following limitations and known issues exist:

- Only Limit and Stop Limit orders are supported (margin and exchange trading)
- Order modifications are not supported
- Partial fills are not reported due to current limitations of certain exchanges with regards to partial fills
- For some exchanges order status updates are not immediately available

### Table 22.15. Coinigy constraints

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>-</td>
</tr>
</tbody>
</table>
Automatic order reconciliation (on WebSocket reconnect) is implemented for Coinigy.

### 22.30.3. Coinigy Account Management

#### Table 22.16. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>Not supported</td>
</tr>
<tr>
<td>Deposit</td>
<td>Not supported</td>
</tr>
<tr>
<td>Addresses</td>
<td>Not supported</td>
</tr>
<tr>
<td>Account Events</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

### 22.31. CoinMarketCap

*CoinMarketCap*[^55] - Cryptocurrency Market Capitalizations is a website providing information about all existing crypto currencies and exchanges. The CoinMarketCap interface connects to the website via HTML and [REST API][^56].

The CoinMarketCap interface provides the publicly available daily historical data and reference data for all listed crypto currencies. No account is necessary in order to use the CoinMarketCap adapter.

[^55]: https://coinmarketcap.com/
[^56]: https://coinmarketcap.com/api/
Chapter 23.  

EXECUTION ALGOS

23.1. Existing Execution Algos

AlgoTrader provides several built-in Execution Algos.

SlicingOrder

The Slicing Algo is only recommended for traditional assets like Equities and derivatives. It is disabled for cryptocurrencies, consider the TWAP or VWAP algos for similar functionality which will be compatible with assets traded on crypto exchanges.

Splits an order into several child orders. child order quantities and time in the market are randomized. The SlicingOrder has the following order properties:

Table 23.1. SlicingOrder

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Unit</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minQuantity</td>
<td>Minimum quantity for each child order</td>
<td>BigDecimal</td>
<td></td>
</tr>
<tr>
<td>maxQuantity</td>
<td>Maximum quantity for each child order</td>
<td>BigDecimal</td>
<td></td>
</tr>
<tr>
<td>minVolPct</td>
<td>Minimum % of volBid / volAsk to take</td>
<td>double</td>
<td>0%</td>
</tr>
<tr>
<td>maxVolPct</td>
<td>Maximum % of volBid / volAsk to take</td>
<td>double</td>
<td>100%</td>
</tr>
<tr>
<td>minDuration</td>
<td>Minimum duration of each child order</td>
<td>seconds</td>
<td>1.0</td>
</tr>
<tr>
<td>maxDuration</td>
<td>Maximum duration of each child order</td>
<td>seconds</td>
<td>1.0</td>
</tr>
<tr>
<td>minDelay</td>
<td>Minimum delay between two child orders</td>
<td>seconds</td>
<td>1.0</td>
</tr>
<tr>
<td>maxDelay</td>
<td>Maximum delay between two child orders</td>
<td>seconds</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The quantity of each child order is randomized between minVolPct and maxVolPct of the current volume offered at the exchange. In addition minQuantity and maxQuantity restriction can be imposed. If maxVolPct is zero, then the current market volume will not be considered when sizing the order. If maxQuantity is zero, then no maximum quantity will be enforced on top of the market volume restriction.

The SlicingOrder will make sure that the remainingQty for the next child order is greater than minQuantity. Maximum quantity rules have precedence over minimum quantity rules.

Example:

minVolPct: 25%, minQuantity: 20, maxVolPct: 100%, maxQuantity: 100, BUY order, quantity: 40, vol ask: 10

minimum quantity: Max(25% x 10, 20) = 20

maximum quantity: Min(100% x 10, 100) = 10
This will result in an order of quantity 10

Each order will stay in the market for minDuration to maxDuration seconds (if it is not filled before that). Between each child order there will be a random delay of minDelay to maxDelay seconds. In addition, the SlicingOrder has a sophisticated pricing logic. For a BUY order the first child order will be place 1 tick below the Ask. For a SELL order the first tick will be placed one tick above the Bid. Depending on whether a child order gets filled, the price of the next child order is adjusted. If a child order gets filled, the price of the next child order will be reduced by one tick (for BUY orders) but it will always be at least one tick above the Bid. If the child order does not get filled, the price of the next child order is increased by one tick (for BUY orders) but it will never be higher than the ask. A SlicingOrder can be created and sent as follows:

```java
SlicingOrder order = new SlicingOrder();
order.setStrategy(strategy);
order.setAccount(account);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
order.setMinQuantity(BigDecimal.valueOf(10));
order.setMaxQuantity(BigDecimal.valueOf(100));
order.setMinVolPct(0.01);
order.setMaxVolPct(0.1);
order.setMinDuration(1);
order.setMaxDuration(5);
order.setMinDelay(1);
order.setMaxDelay(5);
getOrderService().sendOrder(order);
```

Alternatively Section 16.2.1, “Order Preferences” can be used to create a SlicingOrder. The AlgoTrader sample data contains an OrderPreference named SLICING (with the default values shown in the table above) which allows placing a SlicingOrder as follows:

```java
Order order = getOrderService().createOrderByOrderPreference("SLICING");
order.setStrategy(strategy);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
getOrderService().sendOrder(order);
```

VWAPOrder

The VWAPOrder seeks to achieve the Volume-Weighted Average price (VWAP)\(^1\). VWAP is a trading benchmark used by many institutional investors. VWAP is calculated by adding up the market value

---

\(^1\) https://en.wikipedia.org/wiki/Volume-weighted_average_price
traded for every transaction (price multiplied by number of contracts traded) and then dividing by the total contract traded. The VWAPOrder is based on the AdaptiveOrder (see) below and uses its pricing logic. The VWAPOrder has the following order properties in addition to the ones defined by the AdaptiveOrder.

Table 23.2. VWAPOrder

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Unit</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lookbackPeriod</td>
<td>look back period</td>
<td>days</td>
<td>5</td>
</tr>
<tr>
<td>bucketSize</td>
<td>size of each historical volume bucket</td>
<td>Duration</td>
<td>MIN_15</td>
</tr>
</tbody>
</table>

The VWAPOrder retrieves historical prices for the number of days specified in the lookbackPeriod parameter and splits the trading day into buckets with a length in minutes according to the bucketSize parameter.

When a VWAPOrder is either fully-executed or cancelled a message containing the average price, the benchmark price as well as the execution duration and number of executions is logged to the console.

For the VWAPOrder to work a historical data adapter will need to be enabled, see Section 7.2.7, “Historical Data Service”.

A VWAPOrder can be created and sent as follows:

```java
VWAPOrder order = new VWAPOrder();
order.setStrategy(strategy);
order.setAccount(account);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
order.setBucketSize(Duration.MIN_10);
order.setLookbackPeriod(10);
order.setDuration(600);
order.setSliceLength(10);
order.setCancelTime(0.5);
order.setTimeRand(0.25);
order.setQtyRand(0.25);
order.setIncrement(0.05);
order.setInitialOffset(0.8);
order.setMinOffset(0.05);
order.setMaxOffset(1.0);
getOrderService().sendOrder(order);
```

Alternatively Section 16.2.1, “Order Preferences” can be used to create a VWAPOrder. The AlgoTrader sample data contains an OrderPreference named VWAP (with the default values shown in the table above) which allows placing a VWAPOrder as follows:
Order order = getOrderService().createOrderByOrderPreference("VWAP");
order.setStrategy(strategy);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
getOrderService().sendOrder(order);

TWAPOrder

The **TWAPOrder** seeks to achieve the *Time-Weighted Average price (TWAP)*\(^2\). TWAP is a trading benchmark used by many institutional investors. TWAP is derived by calculating the average execution price over a certain time period irrespective of the executed quantity.

The **TWAPOrder** is based on the **AdaptiveOrder** (see) below and uses its pricing logic. The **TWAPOrder** has no additional order properties in addition to the ones defined by the **AdaptiveOrder**.

When a **TWAPOrder** is either fully-executed or cancelled a message containing the average price, the benchmark price as well as the execution duration and number of executions is logged to the console. As the reporting functionality needs historical data a historical data adapter will need to be enabled, see *Section 7.2.7, “Historical Data Service”*. The **TWAPOrder** can still be used without historical data but not report will be logged to the console.

A **TWAPOrder** can be created and sent as follows:

TWAPOrder order = new TWAPOrder();
order.setStrategy(strategy);
order.setAccount(account);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
order.setDuration(600)
order.setSliceLength(10)
order.setCancelTime(0.5)
order.setTimeRand(0.25)
order.setQtyRand(0.25)
order.setIncrement(0.05)
order.setInitialOffset(0.8)
order.setMinOffset(0.05)
order.setMaxOffset(1.0)
getOrderService().sendOrder(order);

Alternatively, Section 16.2.1, “Order Preferences” can be used to create a TWAPOrder. The AlgoTrader sample data contains an OrderPreference named TWAP (with the default values shown in the table above) which allows placing a TWAPOrder as follows:

```java
Order order = getOrderService().createOrderByOrderPreference("TWAP");
order.setStrategy(strategy);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
getOrderService().sendOrder(order);
```

AdaptiveOrder

The AdaptiveOrder is the parent class of the VWAPOrder and TWAPOrder and defines the pricing logic for those. However, it is not possible to send an AdaptiveOrder directly. The AdaptiveOrder has the following order properties:

### Table 23.3. AdaptiveOrder

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Unit</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>startTime</td>
<td>start time of the algo</td>
<td>ZonedDateTime</td>
<td></td>
</tr>
<tr>
<td>endTime</td>
<td>end time of the algo</td>
<td>ZonedDateTime</td>
<td></td>
</tr>
<tr>
<td>duration</td>
<td>duration of the algo</td>
<td>seconds</td>
<td>600</td>
</tr>
<tr>
<td>minSliceQty</td>
<td>minimum child order quantity</td>
<td>BigDecimal</td>
<td></td>
</tr>
<tr>
<td>maxVolPct</td>
<td>maximum % of volBid / volAsk to take</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>sliceLength</td>
<td>average child order length</td>
<td>seconds</td>
<td>10</td>
</tr>
<tr>
<td>cancelTime</td>
<td>% of sliceLength when a child order gets cancelled</td>
<td>%</td>
<td>50%</td>
</tr>
<tr>
<td>timeRand</td>
<td>sliceLength and cancelTime randomization</td>
<td>%</td>
<td>25%</td>
</tr>
<tr>
<td>qtyRand</td>
<td>child order quantity randomization</td>
<td>%</td>
<td>25%</td>
</tr>
<tr>
<td>increment</td>
<td>price increment/decrement</td>
<td>%</td>
<td>5%</td>
</tr>
<tr>
<td>initialOffset</td>
<td>initial offset in % of the first child order</td>
<td>%</td>
<td>80%</td>
</tr>
<tr>
<td>minOffset</td>
<td>minimum offset</td>
<td>%</td>
<td>5%</td>
</tr>
<tr>
<td>maxOffset</td>
<td>maximum offset</td>
<td>%</td>
<td>100%</td>
</tr>
</tbody>
</table>

The AdaptiveOrder uses a pricing logic similar to the Slicing Execution Algo.

The first child order will be placed at the initialOffset between the Bid and the Ask (e.g. at 80%). Depending on whether a child order gets filled, the price of the next child order is adjusted. If the previous child order got fully or partially filled, the price of the next child order will be reduced by increment % of the spread. If the previous child order did not get filled, the price of the next child order is increased by increment % of the spread. The limit price will always be adjusted within the following price range:
The AdaptiveOrder will execute over a predefined time period which can be set by two of the following arguments: startTime, endTime or duration.

New child orders will be sent in randomized time intervals:

\[
\text{Between } \text{sliceLength} \times (1-\text{timeRand}) \text{ and } \text{sliceLength} \times (1+\text{timeRand})
\]

In case a child order is not fully executed it will get cancelled after the following period of time:

\[
\text{Between } \text{sliceLength} \times \text{cancelTime} \times (1-\text{timeRand}) \text{ and } \text{sliceLength} \times \text{cancelTime} \times (1+\text{timeRand})
\]

The quantity of each child order is randomized in the following interval

\[
\text{between } \text{sliceQty} \times (1-\text{qtyRand}) \text{ and } \text{sliceQty} \times (1+\text{qtyRand})
\]

Calculated child order quantities respect the optional minSliceQty. In addition, the AdaptiveOrder also respects the optional property maxVolPct which will cause to algo not to place child orders larger than the current VolAsk (for Buy orders) or VolBid (for Sell orders).

As the Algo needs to be execute within a predefined time period the child order quantities are adjusted throughout the order execution. Quantity adjustments take into consideration previously executed quantity in order to fully executed the algo within its time constraints. No further quantity adjustments take place once 90% of the order execution time has passed.

**TargetPositionOrder**

The TargetPositionOrder seeks to bring the actual position to an intended target quantity. The TargetPositionOrder starts off by looking up the actual position quantity, calculating the delta between the actual and target quantity and issuing a market order to fill the difference. In many cases the TargetPositionOrder differs little from sending a simple market order. Orders can take some time to fully execute. In the meantime the target position may change. The target quantity of a TargetPositionOrder can be altered at any point of time which will cause the order to re-evaluate its actual state and cancel or modify currently pending order and issue a new order if necessary to match the expected target position. The order also reacts intelligently to stray fills that can occur.

By default TargetPositionOrder is considered to be fully executed once its target position has been reached. The order is then removed from the order book. Often however strategies might want to maintain a particular position over a longer period of time. TargetPositionOrder can be issued with keepAlive attribute set to true to make the order active until explicitly canceled. The order will transition into Status#TARGET_REACHED state once fully executed and will stay there until the target is adjusted or the order is canceled. A TargetPositionOrder can be created and sent as follows:

```java
TargetPositionOrder order = new TargetPositionOrder();
order.setStrategy(strategy);
```
order.setAccount(account);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
order.setKeepAlive(true);
order.setTarget(BigDecimal.valueOf(111.1));

getOrderService().sendOrder(order);

Alternatively OrderPreferences can be used to create a TargetPositionOrder.

### TrailingLimitOrder

A TrailingLimitOrder submits an order directly to the exchange with a limit price set a fixed distance away from the current market price. The limit price is adjusted relative to the market price when the market moves in favor of the order. The TrailingLimitOrder is typically used when entering a position on an instrument with a Bullish view.

For a BUY order the limit price will be set a specific amount (defined by the `trailingAmount` parameter) below the current market price. In case the market price rises, the limit price is increased once the specified minimum amount (defined by the `increment` parameter) is exceeded. If the market price falls, the limit price stays untouched. If the market price falls below the limit price the order will get filled by the exchange (depending on adequate liquidity).

For a SELL order the limit price will be set a specific amount (defined by the `trailingAmount` parameter) above the current market price. In case the market price falls, the limit price is decreased once the specified minimum amount (defined by the `increment` parameter) is exceeded. If the market price rises, the limit price stays untouched. If the market price rises above the limit price the order will get filled by the exchange (depending on adequate liquidity).

A TrailingLimitOrder can be created and sent as follows:

```java
TrailingLimitOrder order = new TrailingLimitOrder();
order.setStrategy(strategy);
order.setAccount(account);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
order.setTrailingAmount(BigDecimal.valueOf(0.5));
order.setIncrement(BigDecimal.valueOf(0.1));
```

Alternatively OrderPreferences can be used to create a TrailingLimitOrder.

### 23.2. Execution Algos Retry and Back-off policies

AlgoTrader provides an automated retry handling for Orders sent by Execution Algos (this feature is currently only available for AdaptiveOrders (TWAPOrder and VWAPOrder).
All child orders of TWAP/VWAP Order will be retried in case of:

- Child order send - exception
- Child order send - status REJECTED
- Child order cancel - exception
- Child order cancel - status REJECTED

When the number of retries has exceeded the defined retry limit (default: 3 retries for each order), the entire parent Order (all children Orders) will be cancelled.

The automated retry handling is depicted in the following two diagrams
Figure 23.2. Adaptive Cancel Child Order Retry Policy
Chapter 24.

Synthetic Securities and Derivative Spreads

Figure 24.1. Combinations and Components

AlgoTrader supports Synthetic Securities & Derivative Spreads based on the two Entities Combination and Component.

Combinations are handled like every other Security. A Combination consists of one or many Components. Each component has a quantity.

When trading combinations there are two options:

- tradeable / non-synthetic combinations
- synthetic / non-tradeable combinations

For synthetic / non-tradeable combinations the AlgoTrader Server generates Ticks based on the size of the components of the combination and the current market values of the associated securities. This calculation is handled by the module module-combination.epl which provides the Component Window.

**Note**

It is possible to trade tradeable / non-synthetic combinations through the IB interface. For combination orders AlgoTrader will place BAG orders through the IB interface. For this to work it is necessary to have conids defined for all components of the combination.

On executions AlgoTrader will create fills for each component and for the combination itself. As a consequence there will be positions on all components as well as the combination itself.
A Combination is available to all strategies and can be subscribed/unsubscribed in the usual manner.

### 24.1. Combination Example

<table>
<thead>
<tr>
<th>Market Data (3)</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>last</strong></td>
<td><strong>security id</strong></td>
<td><strong>description</strong></td>
<td><strong>symbol</strong></td>
<td><strong>bid</strong></td>
<td><strong>ask</strong></td>
</tr>
<tr>
<td>2772.75</td>
<td>1</td>
<td>ES SEP18</td>
<td>ESUB</td>
<td>2772.75</td>
<td>2773</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>ES DEC18</td>
<td>ESZ8</td>
<td>2772</td>
<td>2777</td>
</tr>
<tr>
<td>32</td>
<td>52</td>
<td>ESCOMB</td>
<td>ES023.02</td>
<td>38828.95</td>
<td>38834</td>
</tr>
</tbody>
</table>

**Figure 24.2. Combination Example**

The example above shows a Combination based on 11 ES Mini September 2018 Futures and 3 ES Mini December 2018 Futures. The example shows that the market price of the combination is based on the total prices of both components, e.g. for the ask price:

\[11 \times 2773 + 3 \times 2777 = 38834\]

### 24.2. Combination Service

The `CombinationService` is responsible for handling all Combination / Component related DB-Operations.

#### 24.2.1. Create Combination

The following code example shows how to create a combination, add components to it, create a non-tradeable position based on it and subscribe to it:

```java
Combination combination = getCombinationService().createCombination( CombinationType.RATIO_SPREAD, securityFamilyId);

for (Security security : securities) {
    getCombinationService().addComponentQuantity( combination.getId(), security.getId(), quantity);
}

getSubscriptionService().subscribeMarketDataEvent(strategyName, combination.getId());
```

#### 24.2.2. Update Component Quantity

The quantity of a Component can be set like this:

```java
getCombinationService().setComponentQuantity( combinationSecurityId, componentSecurityId, quantity);
```
To add an amount to the current quantity of a Component:

```java
getCombinationService().addComponentQuantity(
   combinationSecurityId, componentSecurityId, quantity);
```

### Important

If Components are modified directly in the database, it is necessary to clear the cache as well as to call the method `ServerManagementService.resetComponentWindow` immediately afterwards. If this is not done within a short period of time this might lead to miss-pricing of the corresponding `Combination`. It is therefore preferable to modify Components via the `AlgoTrader Client` or the `CombinationService`.

#### 24.2.3. Remove a Component

```java
getCombinationService().removeComponent(combinationSecurityId, componentSecurityId);
```
Spring Services

25.1. Starter Classes

AlgoTrader provides the following starter classes to start up the system for the various operational modes:

- **Reference Data Starters**

  When downloading reference data (see *Chapter 19, Reference Data*) you need to have the following profiles active: `singleDataSource` or `pooledDataSource` and the profile of the adapter you want to get reference data from (see table below).

  You can only have one adapter reference data profile enabled at a time.

- **Historical Data Starters**

  When downloading historical data (see *Section 18.3, “Historical Data Download”*) you need to have the following profiles active: `singleDataSource` or `pooledDataSource`, `influxDB` and the profile of the adapter you want to get historical data from (see table below).

  You can only have one adapter historical data profile enabled at a time.

- **Simulation Starter**

  To run a back-test (see *Chapter 5, Strategy Backtesting*), you need to have the following profiles active: any `dataSource` profile (although `embeddedDataSource` is recommended), `simulation`. If you are using InfluxDB for back testing, you also need to add `influxDB`.

- **Embedded Strategy Starters**

  When running strategy in embedded mode (see *Section 3.2.1, “Embedded Mode”*), you need to have the following profiles activate: `singleDataSource` or `pooledDataSource`, `live`, `embeddedBroker` and `html5` (if you want to see/use the UI) and the market data, trading profiles. If account data (see *Chapter 20, Account Data*) is required you also need the `account` profile (see table below).

  The system can be run with several market data, trading and account profiles in the same process.

  If historical data (see *Chapter 18, Historical Data*) is required you also need one `historicalData` profile (see table below) and `influxDB`. If you do not have a historical data provider but still want to store and retrieve historical data using InfluxDB, you need to set `noopHistoricalData` in addition to `influxDB`.

- **Server Starters**

  When running the AlgoTrader server in distributed mode (see *Section 3.2.2, “Distributed Mode”*), you need to have the following profiles activate: `singleDataSource` or `pooledDataSource`, `live`, `embeddedBroker` and `html5` (if you want to see/use the UI) and the market data, trading profiles.
If account data (see *Chapter 20, Account Data*) is required you also need the account profile (see table below).

The system can be run with several market data, trading and account profiles in the same process.

If historical data (see *Chapter 18, Historical Data*) is required you also need one historicalData profile (see table below) and influxDB. If you do not have a historical data provider but still want to store and retrieve historical data using InfluxDB, you need to set noopHistoricalData in addition to influxDB.

**Strategy Starters**

For strategies running in distributed mode (see *Section 3.2.2, “Distributed Mode”*), it is sufficient to activate the live Spring profile. In order for a strategy to use historical data, reference data or account related information the profiles historicalData, referenceData and account have to be enabled in on the strategy.

**Reset Starter**

The ResetStarter can be used to reset the state of the database to a pre-defined state either before a simulation or if a reset of live trading is required. For parameters and details, see *Section 7.2.20, “Reset Service”*

**Restore Portfolio Value Starter**

The RestorePortfolioValueStarter can restore Portfolio Values for a specified strategy and time period (see *Section 11.2, “Portfolio Value Restoration Feature”)

### 25.2. Spring Profiles

AlgoTrader is heavily relying on Spring Profiles to activate/deactivate various parts of the system based on user requirements.

**General Profiles**

- **simulation**: Contains Spring Beans that are used for Back Tests or when using the Exchange Simulator, e.g. SimulationExecutor, SimulationOrderService and ResetService
- **live (client side)**: Contains Spring Beans needed by Strategies in Live Trading: Esper Engine, LifecycleManager, CacheManager & LookupService
- **live (server side)**: Contains Spring Beans needed by the Server in Live Trading mode: e.g. Esper Engine
- **noopHistoricalData**: a no-operation HistoricalDataService. This profile is need for cases where you want to store/retrieve historical data in InfluxDB but no historical data adapter is active.
- **embeddedBroker**: embedded ActiveMQ broker, which is required for sending messages to the UI and to strategies running in distributed mode
- **html5**: the AlgoTrader UI
• influxDB: influxDB interface

Additional services profiles (used only by strategy running in distributed mode)

• historicalData: means the strategy will require historical data service

• referenceData: means the strategy will require reference data service

• account: means the strategy will require account service

Data Sources: only one data source can be configured

• pooledDataSource: c3p0 Pooled Data Source (typically used in live trading both in embedded and distributed mode)

• singleDataSource: Spring Driver Manager Data Source (typically used by the Reference Data Starter and Historical Data Starter)

• embeddedDataSource: H2 embedded in-memory Data Source (typically used when performing back tests)

Adapters

Table 25.1. Adapter Spring Profiles

<table>
<thead>
<tr>
<th>Adapter</th>
<th>Trading</th>
<th>Market Data</th>
<th>Historical Data</th>
<th>Reference Data</th>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bloomberg</td>
<td></td>
<td>bBMarketData</td>
<td>bBHistoricalData</td>
<td>bBReferenceData</td>
<td></td>
</tr>
<tr>
<td>Currenex</td>
<td>cNXFix</td>
<td>cNXMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DukasCopy</td>
<td>dCFix</td>
<td>dCMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exante</td>
<td>xNTFix</td>
<td>xNTMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EzeSoft</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>/ rTFix</td>
</tr>
<tr>
<td>RealTick</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fortex</td>
<td>fTXFix</td>
<td>fTXMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FXCM</td>
<td>fXCMFix</td>
<td>fXCMMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InteractiveBrokers</td>
<td>IBNative</td>
<td>iBMarketData iBHistoricalData&amp;iBReferenceData</td>
<td>iBAccount</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JPM Morgan</td>
<td>jPMFix</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMAX</td>
<td>lMAXFix</td>
<td>lMAXMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nexus Prime</td>
<td>nXSPFix</td>
<td>nXSMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PrimeXM</td>
<td>pXMFix</td>
<td>pXMMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quandl</td>
<td></td>
<td></td>
<td>qDLHistoricalData</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QuantHouse</td>
<td></td>
<td>qHMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SocGen</td>
<td>sGFix</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adapter</td>
<td>Trading</td>
<td>Market Data</td>
<td>Historical Data</td>
<td>Reference Data</td>
<td>Account</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------</td>
<td>-------------------</td>
<td>-----------------</td>
<td>----------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Trading Technologies</td>
<td>tTFix</td>
<td>tTMarketData</td>
<td>tTReferenceData</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UBS</td>
<td>uBSFix</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binance</td>
<td>bNC</td>
<td>bNCMarketData</td>
<td></td>
<td>bNCReferenceData</td>
<td>bNCAccount</td>
</tr>
<tr>
<td>Bitfinex</td>
<td>bFX</td>
<td>bFXMarketData</td>
<td></td>
<td>bFXReferenceData</td>
<td>bFXAccount</td>
</tr>
<tr>
<td>Bitflyer</td>
<td>bFL</td>
<td>bFLMarketData</td>
<td></td>
<td>bFLReferenceData</td>
<td>bFLAccount</td>
</tr>
<tr>
<td>BitMEX</td>
<td>bMX</td>
<td>bMXMarketData</td>
<td></td>
<td>bMXReferenceData</td>
<td>bMXAccount</td>
</tr>
<tr>
<td>Bitstamp</td>
<td>bTSFix</td>
<td>bTSMarketData</td>
<td></td>
<td>bTSReferenceData</td>
<td>bTSAccount</td>
</tr>
<tr>
<td>CoinAPI</td>
<td>cNP</td>
<td>cNPMarketData</td>
<td>cNPHistoricalData</td>
<td>cNPReferenceData</td>
<td></td>
</tr>
<tr>
<td>Coinbase</td>
<td>cNBFix</td>
<td>cNBMarketData</td>
<td></td>
<td>cNBReferenceData</td>
<td>cNBAccount</td>
</tr>
<tr>
<td>Coinigy</td>
<td>cNG</td>
<td>cNGMarketData</td>
<td></td>
<td>cNGReferenceData</td>
<td>cNGAccount</td>
</tr>
<tr>
<td>CoinMarketCap</td>
<td></td>
<td></td>
<td></td>
<td>cMCHistoricalData</td>
<td>cMCReferenceData</td>
</tr>
</tbody>
</table>

**Note**

Please append MarketData, HistoricalData, ReferenceData or Account to the Spring Profiles listed above. Example: The Bloomberg market data profile is bBMarketData instead of just bB.

All other services not mentioned above are active in all profiles.

To enable a Profile on start-up, the following VM argument has to be used:

```
-Dspring.profiles.active=iBMarketData,iBNative
```
Chapter 26.

Configuration and Preferences API

26.1. Configuration Files

The AlgoTrader Server contains the following two main configuration files.

conf.properties the main public configuration file:

- Dataset Configuration
- Simulation Settings
- Reporting Settings
- Order / Execution Settings
- RMI Settings
- ActiveMQ Settings
- Jetty Settings

conf-core.properties contains settings that are only used by the core project:

- Data Source Configuration
- Server Engine module definition
- Esper Statements
- Hedging Settings
- ActiveMQ Settings
- Jetty Settings
- SSL Settings
- Http Client Settings
- Mail Settings
- AlgoTrader UI Settings

In addition adapters may have their own settings file. e.g. conf-ib.properties for IB and conf-bb.properties for BB.
Configuration parameters can be changed inside the `.properties` files. As an alternative configuration parameters can be provided as VM arguments in which case they will overwrite existing parameters inside `*.properties` files.

```
-Dstatement.closePosition=false
```

Most configuration parameters are prefixed with a namespace (e.g. `dataSource`, `simulation`, `statement`, `misc`, etc.)

**26.1.1. Encrypting sensitive configuration values**

For security reasons, it is recommended to store sensitive configuration like adapter API key and API secret, in encrypted form.

In Docker based installations the recommended method for that is using [Docker Secrets](http://docs.docker.com/config/containers/secrets/).

For non-Docker installations it is possible to encrypt property value using [Jasypt](http://www.jasypt.org/cli.html) command line utility:

```
...\jasypt-1.9.2\bin>encrypt password="mypassword" input="revdPMxxxxxxxxxwCPo"
```

The encrypted value should be copied to relevant property file, e.g. to `conf-bmx.properties`. Alternatively the properties can be changed via *Section 2.3, “VM Arguments”*:

```
#{"type":"String","label":"API Key"}
bmx.apiKey=ENC(Gv5bH18YmbavDnC3DEaCMKTOh7wRq5VuKpeNo5tYmaALkpJw0ApEMA==)
```

The steps above should be done for each sensitive value (e.g. `apiKey`, `apiSecret`) of each adapter.

In addition the following VM argument must be set to enable encryption

```
-Dconfig.encryption=true
```

Each time the system starts, the user will be prompted to enter the password.

**26.2. Esper Variables**

The configuration files are also used to define values for Esper variables. Because the Esper Variable system is strong typed, variables with their type have to be configured within the corresponding Esper configuration files. e.g.

1 http://www.jasypt.org/cli.html
<variable name="simulation_eventsPerDay" type="long"/>

**Note**

Name spaces have to be specified using an underscore instead of a period. e.g. `simulation_eventsPerDay` corresponds to `simulation.eventsPerDay` in the property file.
Processes and Networking

27.1. SSL security

By default AlgoTrader is assumed to be running within a secure network segment wherein user authentication and authorization as well SSL security are enforced by the runtime environment / operating system. The AlgoTrader process, individual strategy process and browsers running the HTML5 UI exchange data unencrypted primarily to avoid overhead of encryption for maximal performance.

SSL security can be activated through the following property in `conf.properties`. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```
# TLS/SSL transport security
ssl.enabled = true
```

By default AlgoTrader ships with a self-signed certificate which can be import into the browser. Please note that modern browsers will show a warning when using self-signed certificates due to your domain name being different from the AlgoTrader's own domain.

It is therefore strongly recommended to procure a certificate from a major CA (certification authority) trusted by common browsers. Alternatively you can create your own self-signed certificate for testing purposes, the following command will created a certificate for the domain `xxx.algotrader.com`

```
```

To use SSL security please update the following properties in `conf.properties`. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```
# Keystore with SSL key
ssl.keystore = classpath://identity.jks

# Keystore type (JKS will be assumed by default)
ssl.keystoreType =

# Keystore password
ssl.keystorePassword = password

# Private key password
ssl.keyPassword = password
```

When running with TLS transport security turned on AlgoTrader also enforces BASIC user authentication with a user name and a password when logging into the HTML5 UI. User credentials can be provided in `conf.properties`. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:
# Web UI user name
jetty.user = myusername

# Web UI password
jetty.password = secret

## 27.1.1. Importing Certificate into Chrome Browser

1. On the page with the untrusted certificate, click **Ctrl-Shift-I** to open Developer Tools and go to Security

2. Click View Certificate / Details tab > Copy to File. Choose DER encoded binary (.CER)
   
   • On MacOS drag&drop certificate icon to Finder window

3. Open up Chrome Settings > Show advanced settings > HTTPS/SSL > Manage Certificates.

4. Import the exported .CER file, save into "Trusted Root Certificate Authorities"

5. Check all boxes and click OK. Restart Chrome.
Metrics

In Simulation Mode the performance objective of the system is high-throughput, whereas in Live Trading Mode the objective is low latency. To pinpoint potential performance bottlenecks, AlgoTrader has a built-in metrics functionality.

28.1. Configuration

To enable this feature:

- in conf.properties set misc.metricsEnabled to true
- add module metrics to all relevant strategy db records
- inside esper-common.cfg.xml, uncomment the following sections:

```xml
<event-type name="StatementMetric" class="ch.algotrader.vo.StatementMetricVO"/>

// in simulation
<metrics-reporting enabled="true" engine-interval="-1" statement-interval="86400000" />

// in live trading
<metrics-reporting enabled="true" engine-interval="-1" statement-interval="10000" />
```

28.2. Metrics Reporting

AlgoTrader Metrics Reporting logs a detailed summary of all metrics as well as statement time consumption to the console. Displayed values are Execution time (in nano seconds) and Execution Count.

In Simulation Mode (if metrics are enabled) there will be a metrics report at the end of each simulation run.

**Note**

Subscriber time consumption is not included in statement metrics, whereas static method invocation is included.
Chapter 29.  

Logging

AlgoTrader logging is provided by *Apache Log4j* \(^1\) framework. The Logging system is configured by means of `log4j2.xml` file.

The log level can be changed through the following VM argument:

```
-DlogLevel=ERROR
```

### 29.1. log4j2.xml

#### Table 29.1. Default Log4j Appenders

<table>
<thead>
<tr>
<th>Appender</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StdOut</td>
<td>Logs to Standard Out</td>
</tr>
<tr>
<td>StdErr</td>
<td>Logs to Standard Error</td>
</tr>
<tr>
<td>LogEvent</td>
<td>Custom UI appender. Sends log messages to UI</td>
</tr>
</tbody>
</table>

### 29.2. Production log4j2.xml

For production usage it is recommended to adapt the `log4j2.xml` to client specific needs. Additional samples for production use are available inside `log4j2.xml`.

#### Table 29.2. Production Log4j Appenders

<table>
<thead>
<tr>
<th>Appender</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Logs to an appending file</td>
</tr>
<tr>
<td>Mail</td>
<td>Sends Email Messages on Errors</td>
</tr>
</tbody>
</table>

**Note**

- Problems with the Email Appender go to System.err (on server see `nohup.log`)
- To prevent saturation of the logs several loggers have been defined with a logging level higher than the root log level

Detailed description of Log4j2 appenders and advanced configuration can be found at the *Apache Logging* \(^2\) site.

\(^1\) [http://logging.apache.org/log4j/2.x/](http://logging.apache.org/log4j/2.x/)

\(^2\) [http://logging.apache.org/log4j/2.x/manual/configuration.html](http://logging.apache.org/log4j/2.x/manual/configuration.html)
Reporting

AlgoTrader provides a convenient way to create custom CSV reports for strategy specific reporting. All relevant classes are available inside the package ch.algotrader.report.

To use the reporting functionality create a class similar to this:

```java
public class MyCustomReport {

    private ListReporter reporter;

    public OrderReport() {
        String[] header = new String[] { "Date", "Symbol", "Quantity", "Signal" };
        this.reporter = new ListReporter(Report.generateFile("OrderReport"), header);
    }

    public void write(Date date, String symbol, int quantity, String signal) {
        this.reporter.write(date, symbol, quantity, signal);
    }
}
```

This will create a .csv report named OrderReport.csv inside the directory /files/report/ which contains the columns Date, Symbol, Quantity, Signal.

Some Reports are available out-of-the-box, for further details please see Section 5.5, “Performance Statistics”
Appendix A. Example Strategy
"BreakOut"

A.1. Trading Idea

Warning

The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it Live.

The Strategy trades the EUR.USD FX Market and is based on a simple Breakout Indicator. The Strategy opens a long (short) position when the current price exceeds (falls below) the maximum (minimum) of the last n bars. After a new position is opened, a profit target price is set as well as a stop loss. If either profit target or stop is reached, the position is closed. If neither stop nor profit target is reached until the end of n-bars, the position is closed.

Positions are sized based on a defined leverage and the current Net Liquidation Value. All Orders are placed as Market Orders. The initial account size is EUR 1'000'000.

A.2. Example

The following 5-min bar chart gives an example of the BreakOut strategy. At 10:20 an aggregation of the last 5 bars between 09:55 and 10:20 is created, based on which the upper limit at 1113.85 and the lower limit at 1110.53 are calculated. At 10:22:37 the upper limit is crossed for the first time and a long position is entered and both a profit target at 1116.40 and a stop loss at 1111.49 are set automatically. At 10:31:52 the profit target is reached and the position is automatically closed.

Note

This example strategy is a good example of combining a bar based strategy with tick-by-tick based actions. The creation of the upper and lower limits are based on the five 5-min bars but the opening and closing of the position takes place as soon as the limits are reached without waiting for the current bar to finish. This is one of the unique features of AlgoTrader that distinguishes it form other trading platforms that operate exclusively based on bars.
Figure A.1. BreakOut Strategy Example

A.3. Implementation

The main artifacts needed for the Implementation of a new Strategy are described in Chapter 4, Strategy Development.

The following list will give an overview of the specific artifacts implemented by the BreakOut Strategy (Note: Most of the functionality is documented via Javadoc or Esper comments):

```
/srcrepo/main/java/ch/algotrader/strategy/breakOut/BreakOutService.java

  The strategy service class providing the main entry method invoked by the Esper ENTRY_LONG and ENTRY_SHORT statements:
```

```
/srcrepo/main/java/ch/algotrader/strategy/breakOut/BreakOutConfig.java

  Contains all strategy configuration items

/srcrepo/main/resources/module-breakOut.epl

  Esper Module containing all statements for this strategy:
  ```
  • INSERT_INTO_BAR: Creates High/Low Bars
  • INSERT_INTO_BOUND: Calculates minimum and maximum of last n bars
  • ON_BOUND_SET_TRIGGERS: sets the upperTrigger and lowerTrigger based on the minimum and maximum of the last n bars
  • ENTRY_LONG / ENTRY_SHORT: open position if last tick is higher (lower) than previous n bars.
  ```
• **CLOSE_LONG_POSITION / CLOSE_SHORT_POSITION**: Close position if last tick is higher (lower) than target or lower (lower) than stop

• **CLOSE_OPEN_POSITION**: Close position if neither target nor stop are reached before the end of n-bars

/src/main/resources/breakOut-default.properties
Contains default parameters used by the strategy (e.g. `lengthOfBar` and `numberOfBars`)

/src/main/resources/META-INF/esper-breakOut.cfg.xml
Contains event-types definitions (i.e. `CurrentValue`), variables (e.g. `lengthOfBar` and `numberOfBars`).

/src/main/resources/META-INF/applicationContext-client-breakOut.xml
Contains the Spring Bean definitions for `breakOutConfigParams`, `breakOutConfig`, `breakOutEngine`, `breakOutService`.

/src/main/resources/db/mysql/mysql-breakout.sql
Contains the MySQL database records. Needs to be imported into the database before running the strategy with the MySQL database.

To start the Strategy please see the explanations in *Chapter 3, Starting AlgoTrader*.

### A.4. Installation & Startup

To setup the strategy for back testing and live trading on a development workstation please execute the following steps:

**Git Clone**

Perform a Git clone from the command line:

```
git clone https://gitlab.algotrader.ch/general/examples.git
```

**Import the projects** `breakOut` into Eclipse via File / Import / Maven / Existing Maven Projects:

**Deploy Tick Data File**

download file:

`eurusd-1min-20111218-20130121.zip`

to:

`breakOut/files/tickdata/eurusd-1min-20111218-20130121/EURUSD.csv`

**Start the Simulation**

launch the Eclipse Run Configuration: `SimulationStarter-simulate-breakOut`

To start the strategy in live trading mode on a development workstation please execute the following steps:

---

1 [https://repo.algotrader.ch/tickdata/eurusd-1min-20111218-20130121.zip](https://repo.algotrader.ch/tickdata/eurusd-1min-20111218-20130121.zip)
Initialize the database

- load the db-samples script into the MySQL database: `/algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql`
- load the strategy specific script into the MySQL database: `/breakOut/src/main/resources/db/mysql/mysql-data.sql`

Start the Strategy

- invoke the Eclipse Run Configuration: `EmbeddedStarter-breakOut`

To start the strategy in live trading mode on a productive server please execute the following steps:

Copy docker compose file

- Copy the following file to the server and make changes as needed:
  
  `https://gitlab.algotrader.ch/general/examples/blob/master/breakOut/docker-compose.yml`

Run docker compose

- Invoke the following command inside the directory where the `docker-compose.yml` file is located:

  ```
  docker-compose up -d
  ```

**Note**

Prior to starting the strategy for the very first time please start the AlgoTrader server by itself by executing the following command inside `/algotrader-launch`. This will load the MySQL sample data.

  ```
  docker-compose up -d mysql ibgateway algotrader
  ```
Appendix B. Example Strategy "Box"

B.1. Trading Idea

Warning
The purpose of this Strategy is to demonstrate the capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! Due to frequent Draw Downs, it might lead to large losses. Even when modifying or extending the Strategy use caution before trading it Live.

The Strategy trades the EUR.USD FX Market and is based on the Stairstep Breakouts (SSBO) Indicator that is presented on www.forexfactory.com¹ by forexhard².

The Trading Idea behind the Strategy is the following: Markets will often stay within a trading range for a considerable amount of time before they break-out in either direction. The following Chart shows some examples of trading ranges.

![Figure B.1. Box Trading Ranges](image)

After a break-out markets might return back into the trading range but will eventually make a major move in one direction.

According to the defined settings, the Strategy looks for a trading range with a minimum length in Minutes (e.g. 90 Minutes) and a maximum width in Pips (e.g. 30 Pips). The chart below displays a typical trading range in dark blue color.

---

As soon as trading range has been built according to these parameters, the Strategy waits for the first breakout to happen. The strategy enters the market in the direction of the breakout as soon as a small margin called buffer (dashed red line, e.g. 5 Pips) has been crossed. In the example above, this happened at 10:48.

The Strategy will set a stop at the opposite side of the box (e.g. 1.3618 = 39 Pips) and a target with the same distance (e.g. 1.3544 = 39 Pips).

If the target is reached, the Strategy resets itself and waits for a new Box to build itself.

If the Position gets stopped out (at the opposite side of the Box), The Strategy waits for the next break-out to happen (on the same Box) and enters the market again after the buffer-line has been crossed. This time the size of the position is doubled in order to cover the losses of the first entry, in case the target is reached this time around. Position size is doubled up to a defined maximum. Because of this doubling the system can be categorized as a Martingale Strategy (see *Martingale Betting System*).

The following State Chart Diagram depicts the different states the Strategy will pass through:

---

3 https://en.wikipedia.org/wiki/Martingale_%28betting_system%29
Figure B.3. Box States

The default setting of the Strategy will go up to level 5 which will result in a position size of 16 times the original size. So the individual sizes on the different levels will be: 1, 2, 4, 8 & 16. Each successful series will therefore present a profit of 1 unit. Very often series will be successful on a level that is below the maximum level (e.g. below level 5). However if the Strategy has a loosing set, which will be terminated at the maximum level (e.g. at level 5), there will be a loss of 16 times the original position size.

The Strategy will often have multiple successful series in a row before having one major draw down. A typical performance chart will therefore look like this:

Figure B.4. Box Strategy Performance

To prevent having open positions over the weekend the Strategy does not create any new boxes after a defined time on Friday (e.g. 4PM). Also, it will terminate a potential ongoing series at a defined time on Friday (e.g. 10PM)
B.2. Implementation

The main artifacts needed for the Implementation of a new Strategy are described in Chapter 4, Strategy Development.

The following list will give an overview of the specific artifacts implemented by the Box Strategy. Most of the functionality is documented via Javadoc or Esper comments:

/src/main/java/ch/algotrader/strategy/box/BoxService.java
   The strategy service class providing the main methods invoked by different Esper statements.

/src/main/java/ch/algotrader/strategy/box/BoxConfig.java
   Contains all strategy configuration items

/src/main/java/ch/algotrader/strategy/box/Box.java
   A POJO class representing all properties of a Box (e.g. top, bottom, startDateTime and endDateTime)

/src/main/java/ch/algotrader/strategy/box/State.java
   A Java Enum representing the different States the Strategy can pass through (INIT, CREATED, LONG, SHORT, FLAT)

/src/main/resources/module-box-init.epl
   Esper Module containing statements for capturing market data, creating variables and creating Boxes. In Live Trading, these statements will be deployed before the pre feeding.

/src/main/resources/module-box-run.epl
   Esper Module containing statements that invoke the business actions on the BoxService (entry, takeProfit, closePosition, reverse and terminateSeries). In Live Trading these statements will be deployed after pre-feeding is finished.

/src/main/resources/box-default.properties
   Contains parameters used by the strategy (e.g. boxLength and boxRange)

/src/main/resources/META-INF/esper-box.cfg.xml
   Contains event-types definitions (i.e. CurrentValue), imports (i.e. Box and State), variables (e.g. boxLength and boxRange)

/src/main/resources/META-INF/applicationContext-client-box.xml
   Contains the Spring Bean definitions for boxConfigParams, boxConfig, boxEngine, boxService within the Spring profile standalone.

/src/main/resources/db/mysql/mysql-box.sql
   Contains the MySQL database records. Needs to be imported into the database before running the strategy with the MySQL database.

/src/main/resources/html5
   HTML5 and JavaScript files needed for the strategy custom web UI
To start the Strategy please see the explanations in *Chapter 3, Starting AlgoTrader*.

### B.3. Strategy Monitoring

The Box strategy is equipped with an HTML5 custom widget that displays current metrics like State, Units, Upper Target, etc. The custom widget also contains a button to terminate the current series.

![Box HTML5 Custom Widget Example](image)

**Figure B.5. Box HTML5 Custom Widget Example**

**Note**

It might be necessary to fully reload the browser on first startup to show the custom widget using Ctrl + Shift + R.
B.4. Installation & Startup

To setup the strategy for back testing and live trading on a development workstation please execute the following steps:

Git clone
   Perform a Git clone from the command line:

   `git clone https://gitlab.algotrader.ch/general/examples.git`

Import the projects box into Eclipse via File / Import / Maven / Existing Maven Projects:

Deploy Bar Data File
   download file:

   `eurusd-1min-20111218-20130121.zip`

   and unpack to:

   box/files/bardata/eurusd-1min-20111218-20130121/EURUSD.csv

Start the Simulation
   launch the Eclipse Run Configuration: SimulationStarter-simulate-box

To start the strategy in live trading mode on a development workstation please execute the following steps:

Initialize the database
   load the db-samples script into the MySQL database: /algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql

   load the strategy specific script into the MySQL database: /box/src/main/resources/db/mysql/mysql-data.sql

Start the Strategy
   invoke the Eclipse Run Configuration: EmbeddedStarter-box

To start the strategy in live trading mode on a productive server please execute the following steps:

Copy docker compose file
   Copy the following file to the server and make changes as needed:

   `https://gitlab.algotrader.ch/general/examples/blob/master/box/docker-compose.yml`

Run docker compose
   Invoke the following command inside the directory where the docker-compose.yml file is located:

4 https://repo.algotrader.ch/bardata/eurusd-1min-20111218-20130121.zip
docker-compose up -d

Note
Prior to starting the strategy for the very first time please start the AlgoTrader server by itself by executing the following command inside `/algotrader-launch`. This will load the MySQL sample data.

docker-compose up -d mysql ibgateway algotrader
Appendix C. Example Strategy "Pairs Trading"

C.1. Trading Idea

Warning

The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it Live.

The Pairs Trading strategy uses the web service www.pairtradinglab.com¹ to trade pairs of US equities

C.1.1. What Is Pairs Trading?

Pairs trading is a well-known market neutral trading strategy, that gives traders the ability to profit from practically any market conditions. Whether conditions reflect an uptrend, downtrend, or sideways movement, traders can take advantage of the current market using pairs trading. This type of strategy is typically categorized as a statistical arbitrage trading strategy.

The strategy works by monitoring the performance of two historically correlated securities. When the correlation between those two securities demonstrate a temporary weakness, a pairs trade can be conducted by shorting the outperforming stock and going long on the under performing stock. Basically, one is betting that the spread between the two will converge eventually.

C.1.2. Pair Trading Lab

Pair Trading Lab offers tools to assist in setting up and backtesting a pairs trading portfolio. Along with a database of more than 10 million pre-analyzed pairs, Pair Trading Lab offers the following:

- Advanced online back tester
- Online co-integration analyzer
- Private repository of backtests, studies, and pairs
- Portfolio organizer and portfolio backtester

C.1.3. AlgoTrader - Pair Trading Lab Integration

With the integration between AlgoTrader and Pair Trading Lab, it is possible take advantage of the capabilities of both systems in combination:

¹ https://www.pairtradinglab.com/
Pair Trading Lab will be used to:

- Create backtests of pairs
- Verify a pair trading idea and inspect the behavior and robustness of pairs
- Test pairs for co-integration
- Search the PTL database of more than 10 million pre-analyzed U.S. market pairs using complex filters
- Create and maintain lists of interesting pairs, rate them, and tag them
- Create, maintain, and backtest portfolios of pair strategies

Then the AlgoTrader - Pair Trading Lab integration can be used to download selected pairs and/or portfolio of pairs from Pair Trading Lab into AlgoTrader where they can then be traded automatically.

The AlgoTrader based pairs trading strategy implementation is based on the Ratio Model\(^2\)

### C.2. Implementation

The main artifacts needed for the Implementation of a new Strategy are described in Chapter 4, Strategy Development.

The following list will give an overview of the specific artifacts implemented by the Pairs Trading Strategy (Note: Most of the functionality is documented via Javadoc or Esper comments):

- `/src/main/java/ch/algotrader/strategy/pairstrading/service/PairsTradingService.java`
  The strategy service class providing the main trading logic
- `/src/main/java/ch/algotrader/strategy/pairstrading/csv/CsvImporter.java`
  Import utility to download pairs from Pair Trading Lab and configure them in AlgoTrader
- `/src/main/java/ch/algotrader/strategy/pairstrading/util/PairsTradingConfig.java`
  Contains all strategy configuration items
- `/src/main/java/ch/algotrader/strategy/pairstrading/util/PairsTradingCalc.java`
  Contains the logic of the ratio model
- `/src/main/resources/module-pairstrading.epl`
  Esper Module containing all statements for this strategy:
  - `PAIR_WINDOW`: Contains all current pair definitions
  - `SIGNAL_WINDOW`: Contains current signals (will be updated on each tick)

\(^2\) [https://wiki.pairtradinglab.com/wiki/Pair_Trading_Models#Ratio_Model](https://wiki.pairtradinglab.com/wiki/Pair_Trading_Models#Ratio_Model)
Installation & Startup

C.3. Installation & Startup

Before using the strategy please execute the following steps:

Pair Trading Lab account sign-up
   Sign up for a free account³ at Pair Trading Lab

Create a pair portfolio
   create a pair portfolio⁴ and add some pairs

⁴ https://www.pairtradinglab.com/portfolio-manager

- LAST_TICK, INSERT_LATEST_TICK, UPDATE_LATEST_TICK_1 & UPDATE_LATEST_TICK_2: keep track of current prices for all pairs
- UPDATE_HISTORICAL_BARS & DAILY_RECALC: daily triggers for downloading historical data and updating entry thresholds
- INSERT_ZSCORE: calculates the z-score for each new price update
- INSERT INTO SIGNAL_EVENT & ON SIGNAL: create and propagate SignalEvents in case an entry or exit trigger is reached

/src/main/resources/conf-pairstrading.properties
  Contains default parameters used by the strategy

/src/main/resources/META-INF/esper-pairstrading.cfg.xml
  Contains event-types definitions (i.e. PairEvent and SignalEvent)

/src/main/resources/META-INF/applicationContext-client-pairstrading.xml
  Contains pairsTradingParams, pairsTradingConfig, pairsTradingEngine, pairsTradingService as well as the strategy specific beans csvImporter, orderSubmissionService and pairsTradingLabNavigator.

/src/main/resources/db/h2/h2-pairstrading.sql
  Contains the H2 database records needed to simulate the strategy with the embedded in-memory database H2.

/src/main/resources/db/mysql/mysql-pairstrading.sql
  Contains the MySQL database records. Needs to be imported into the database before running the strategy with the MySQL database.

/src/main/resources/html5
  HTML5 and JavaScript files needed for the strategy custom web UI

To start the Strategy please see the explanations in Chapter 3, Starting AlgoTrader.
Extract Portfolio ID

csvImportPortfolio needs to be extracted from the URL when clicking on pair in the PTL Trader / Portfolio Manager

Figure C.1. Pair Trading Portfolio ID

To start the strategy in live trading mode on a development workstation please execute the following steps:

Git Clone

Perform a Git clone from the command line:

```shell
git clone https://gitlab.algotrader.ch/general/examples.git
```

Import the projects pairstrading into Eclipse via File / Import / Maven / Existing Maven Projects:

Deploy MySQL data

Load the file /src/main/resources/db/mysql/mysql-data.sql into MySQL

Configure Pair Trading Lab Credentials

Inside the file conf-pairstrading.properties the following items need to be configured. Alternatively the properties can be changed via Section 2.3, “VM Arguments”:

```properties
#{"type":"String","required":"false","label":"Pair Trading Lab Portfolio ID"}
csvImportPortfolio = xyz

#{"type":"String","required":"false","label":"Pair Trading Lab Username"}
csvImportUser = user
```
strategy monitoring

C.4. Strategy Monitoring

The Pairs Trading strategy is equipped with a separate HTML5 management page. The page is available through the path /pairstrading.html, e.g.:

http://localhost:9090/pairstrading.html
Figure C.2. Pairs Trading HTML5 Custom Widget Example

Note

It might be necessary to fully reload the browser on first startup to show the custom widget using Ctrl + Shift + R.

The HTML5 management page provides the following controls:

- **PairInfo & Pairs**: current pair definitions as downloaded from Pair Trading Lab. movingAvg and standardDev are calculated on a daily basis (by the Esper statement `UPDATE_HISTORICAL_BARS`) using historical closing prices.

- **Signals**: intraday pair values based on live data. ratio shows the current price ratio between individual instruments of a pair. zScore shows the current ratio relative to the Bollinger band around the ratio time series. When the zScore hits the zScoreEntry threshold a position is entered, and when the zScore hits the zScoreExit threshold the position is closed. If the zScore happens to be above zScoreMax (e.g. after a large overnight gap) no new position will be opened. The signal field shows the current state of a pair (i.e. LONG, SHORT, EXIT & HOLD).
• The action **Import Historical Bars** is used to import historical closing prices of all instruments for the relevant look back period. This action is automatically executed once a day. In addition it can be invoked manually at any time.

• The action **Re-Calc Entry Thresholds** is used to update `movingAvg` and `standardDev` based on historical data in the database. This action is automatically executed once a day. In addition it can be invoked manually at any time.

• The action **Import Pairs** imports and/or update pairs from Pair Trading Lab.
Appendix D. Example Strategy "IPO"

D.1. Trading Idea

Warning

The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it Live.

The strategy trades US equity IPOs (initial public offering). When a new stock is launched on the exchange for the first time the strategy tries to realize trading profits of the first trading day. New IPOs are announced on web pages like IPOScoop1 together with an indicative open price.

D.2. Strategy Monitoring

The IPO strategy is equipped with the following HTML5 custom widget which displays currently active IPOs and allows adding and removing IPOs while the strategy is running

---

1 https://www.iposcoop.com/
Figure D.1. IPO HTML5 Custom Widget Example

**Note**

It might be necessary to fully reload the browser on first startup to show the custom widget using Ctrl + Shift + R.

Inside the custom widget the user can enter a new IPO to be traded by populating the following fields:

- **Symbol** to be traded
- **SecurityFamily** the symbol belongs to
- **Limit** The maximum limit price of the initial order
- **Multiplier** to be applied to the opening price for secondary orders (see below)
For each symbol entered, the strategy will place orders at a configurable time in the morning (e.g. 4:30am).

Note
Trading of IPOs usually starts within 2-3 hours after the official market open

Definitions:

• Cash commitment = starting capital / number of symbols to be traded
• Quantity per symbol = cash commitment / user-defined limit price

The strategy will place a limit-at-the-open order at user-defined limit price for the entry quantity specified above. Using a limit-at-the-open order will cause the order to participate in the opening auction of the IPO.

Immediately following the open of regular trading, the strategy will check to see if the entire cash commitment for the given symbol has been exhausted.

1. If it has been exhausted, no further action will be taken
2. If the cash commitment has not been exhausted and the stock opened above the \textit{LIMIT PRICE}, the unfilled quantity will be cancelled, and no further action will be taken by the strategy.
3. Otherwise, if the cash commitment for the name has NOT been exhausted a secondary limit order is placed as per below:

• Remaining cash = Cash commitment - cash used on current position
• Quantity per symbol = Remaining cash / opening price
• Limit price = opening price $\times$ \textit{MULTIPLIER} (e.g. 1.02)

If the secondary order is not filled by a configurable end time (e.g. 3:30pm), it will be cancelled by the system.

Immediately after executing any buy orders, the strategy will place a market-on-close order for the entire position.

D.3. Implementation

The main artifacts needed for the Implementation of a new Strategy are described in Chapter 4, Strategy Development.

The following list will give an overview of the specific artifacts implemented by the IPO Strategy (Note: Most of the functionality is documented via Javadoc or Esper comments):

\texttt{/src/main/java/ch/algotrader/strategy/ipo/IPOService.java}

The strategy service class providing the main trading logic
/src/main/java/ch/algotrader/strategy/ipo/IPO.java
Java POJO class representing a single IPO

/src/main/resources/module-ipo.epl
Esper Module containing all statements for this strategy:

- **SEND_ATO_ORDERS**: sends out at-the-open orders at the configured
- **SEND_LIMIT_ORDERS**: triggers the secondary order service once the at-the-open order has been fully executed and the official open price (via GenericTickVO) has been disseminated. An Esper Join is used for this since either one of those events can arrive first
- **CLOSE_OPEN_ORDERS**: cancels all orders at the configured time
- **DAILY_CLEAN_UP**: unsubscribes all market data and resets the list of IPOs an initial capital

/src/main/resources/conf-ipo.properties
Contains default parameters used by the strategy

/src/main/resources/META-INF/esper-ipo.cfg.xml
Contains Esper variables for the strategy

/src/main/resources/META-INF/applicationContext-client-ipo.xml
Contains ipoConfigParams, ipoEngine & ipoService.

/src/main/resources/db/mysql/mysql-ipo.sql
Contains the MySQL database records. Needs to be imported into the database before running the strategy with the MySQL database.

/src/main/resources/html5
HTML5 and JavaScript files needed for the strategy custom web UI

To start the Strategy please see the explanations in Chapter 3, Starting AlgoTrader.

### D.4. Installation & Startup

To start the strategy in live trading mode on a development workstation please execute the following steps:

**Git Clone**
Perform a Git clone from the command line:

```bash
git clone https://gitlab.algotrader.ch/general/examples.git
```

Import the projects ípo into Eclipse via File / Import / Maven / Existing Maven Projects:

**Deploy MySQL data**
Load the file /src/main/resources/db/mysql/mysql-data.sql into MySQL

**Start the Strategy in Live Trading Mode**
launch the Eclipse Run Configuration:EmbeddedStarter-ipo
To start the strategy in live trading mode on a productive server please execute the following steps:

Copy docker compose file
Copy the following file to the server and make changes as needed:

https://gitlab.algotrader.ch/general/examples/blob/master/ipo/docker-compose.yml

Run docker compose
Invoke the following command inside the directory where the docker-compose.yml file is located:

```
docker-compose up -d
```

**Note**
Prior to starting the strategy for the very first time please start the AlgoTrader server by itself by executing the following command inside /algotrader-launch. This will load the MySQL sample data.

```
docker-compose up -d mysql ibgateway algotrader
```
Appendix E. Example Strategy "EMA"

E.1. Trading Idea

The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it Live.

The Strategy is a simple example without Esper that trades the EUR.USD FX Market and is based on two exponential moving averages.

The Strategy sends a BUY order when shorter moving average (e.g. 10-days) crosses above the longer moving average (e.g. 20-days) and it sends a SELL order when shorter moving average crosses below the longer moving average.

Figure E.1. EMA Strategy Example

E.2. Implementation

The main artifacts needed for the Implementation of a new Strategy are described in Chapter 4, Strategy Development.
The strategy uses the TA4J\(^1\) library which provides over 100 technical indicator that are computed on a continuous basis.

The EMA Strategy consist of one single Java class only:

```
/src/main/java/ch/algotrader/strategy/ema/EMAService.java
```

The strategy service class providing `onStart` and `onBar` method containing the trading logic

To start the Strategy please see the explanations in *Chapter 3, Starting AlgoTrader*.

### E.3. Installation & Startup

To setup the strategy for back testing and live trading on a development workstation please execute the following steps:

**Git Clone**

Perform a Git clone from the command line:

```
git clone https://gitlab.algotrader.ch/general/examples.git
```

Import the projects `ema` into Eclipse via File / Import / Maven / Existing Maven Projects:

**Deploy Bar Data File**

Download file:

`eurusd-1min-20111218-20130121.zip`\(^2\)

and unpack to:

`ema/files/bardata/eurusd-1min-20111218-20130121/EURUSD.csv`

**Start the Simulation**

Launch the Eclipse Run Configuration: `SimulationStarter-simulate-ema`

To start the strategy in live trading mode on a development workstation please execute the following steps:

**Initialize the database**

Load the db-samples script into the MySQL database:

`/algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql`

Load the strategy specific script into the MySQL database:

`/ema/src/main/resources/db/mysql/mysql-data.sql`

**Start the Strategy**

Invoke the Eclipse Run Configuration: `EmbeddedStarter-ema`

---

\(^1\) https://github.com/mdeverdelhan/ta4j-origins

\(^2\) https://repo.algotrader.ch/bardata/eurusd-1min-20111218-20130121.zip
To start the strategy in live trading mode on a productive server please execute the following steps:

**Copy docker compose file**

Copy the following file to the server and make changes as needed:

https://gitlab.algotrader.ch/general/examples/blob/master/ema/docker-compose.yml

**Run docker compose**

Invoke the following command inside the directory where the docker-compose.yml file is located:

```bash
docker-compose up -d
```

**Note**

Prior to starting the strategy for the very first time please start the AlgoTrader server by itself by executing the following command inside /algotrader-launch. this will load the MySQL sample data

```bash
docker-compose up -d mysql ibgateway algotrader
```

---

3 https://gitlab.algotrader.ch/general/examples/blob/master/breakOut/docker-compose.yml
Appendix F. Example Strategy "Random"

F.1. Trading Idea

Warning

The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it Live.

The Strategy is a simple example that places random orders at regular intervals. The Random strategy is used for the AlgoTrader Demo\(^1\).

The Strategy sends a BUY order when shorter moving average (e.g. 10-days) crosses above the longer moving average (e.g. 20-days) and it sends a SELL order when shorter moving average crosses below the longer moving average.

F.2. Implementation

The main artifacts needed for the Implementation of a new Strategy are described in Chapter 4, Strategy Development.

The Random Strategy consist of the following artifacts:

\[\text{/src/main/java/ch/algotrader/strategy/random/RandomService.java}\]

The strategy service class providing the main methods invoked by different Esper statements.

\[\text{/src/main/java/ch/algotrader/strategy/random/RandomConfig.java}\]

Contains all strategy configuration items.

\[\text{/src/main/resources/module-random.epl}\]

Esper Module containing statements to place and cancel orders as well as update subscriptions once a day.

\[\text{/src/main/resources/conf-random.properties}\]

Contains parameters used by the strategy (e.g. \texttt{positionMax} and \texttt{orderMax}).

\[\text{/src/main/resources/META-INF/esper-random.cfg.xml}\]

Contains variables (i.e. \texttt{placeOrderInterval} and \texttt{cancelOrderInterval}).

\[\text{/src/main/resources/META-INF/applicationContext-client-random.xml}\]

Contains the Spring Bean definitions for \texttt{randomConfigParams}, \texttt{randomConfig}, \texttt{randomEngine}, \texttt{randomService}.

\(^1\) \url{http://html5.algotrader.ch/}
/src/main/resources/db/mysql/mysql-data.sql

Contains the MySQL database records. Needs to be imported into the database before running the strategy with the MySQL database.

To start the Strategy please see the explanations in Chapter 3, Starting AlgoTrader.

F.3. Installation & Startup

To setup the strategy for back testing and live trading on a development workstation please execute the following steps:

Git Clone

Perform a Git clone from the command line:

```
$ git clone https://gitlab.algotrader.ch/general/examples.git
```

Import the projects random into Eclipse via File / Import / Maven / Existing Maven Projects:

To start the strategy in live trading mode on a development workstation please execute the following steps:

Initialize the database

load the db-samples script into the MySQL database: /algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql

load the strategy specific script into the MySQL database: /random/src/main/resources/db/mysql/mysql-data.sql

Start the Strategy

invoke the Eclipse Run Configuration: EmbeddedStarter-random

To start the strategy in live trading mode on a productive server please execute the following steps:

Copy docker compose file

Copy the following file to the server and make changes as needed:

```
https://gitlab.algotrader.ch/general/examples/blob/master/random/docker-compose.yml
```

Run docker compose

Invoke the following command inside the directory where the docker-compose.yml file is located:

```
docker-compose up -d
```

---

2 https://gitlab.algotrader.ch/general/examples/blob/master/breakOut/docker-compose.yml
Note

Prior to starting the strategy for the very first time please start the AlgoTrader server by itself by executing the following command inside /algotrader-launch. This will load the MySQL sample data.

docker-compose up -d mysql ibgateway algotrader
Appendix G. Example strategy "EMA" in Python

G.1. Description

This strategy effectively replicates the EMA strategy mentioned above, only it's written in Python and integrates with the rest of the platform by using the available exposed API (REST and ActiveMQ). This is merely just a demonstration of the concept: by utilizing AlgoTrader API it is possible to write strategies in languages other than Java. In order to run this strategy, AlgoTrader server must be started. Running ema-api.py starts executing the strategy, which subscribes to the market data (ActiveMQ topic) and submits the orders (via REST).

All the configurations settings are located inside ema-api.py file. Only installed Python interpreter is required (tested on 3.6.3)