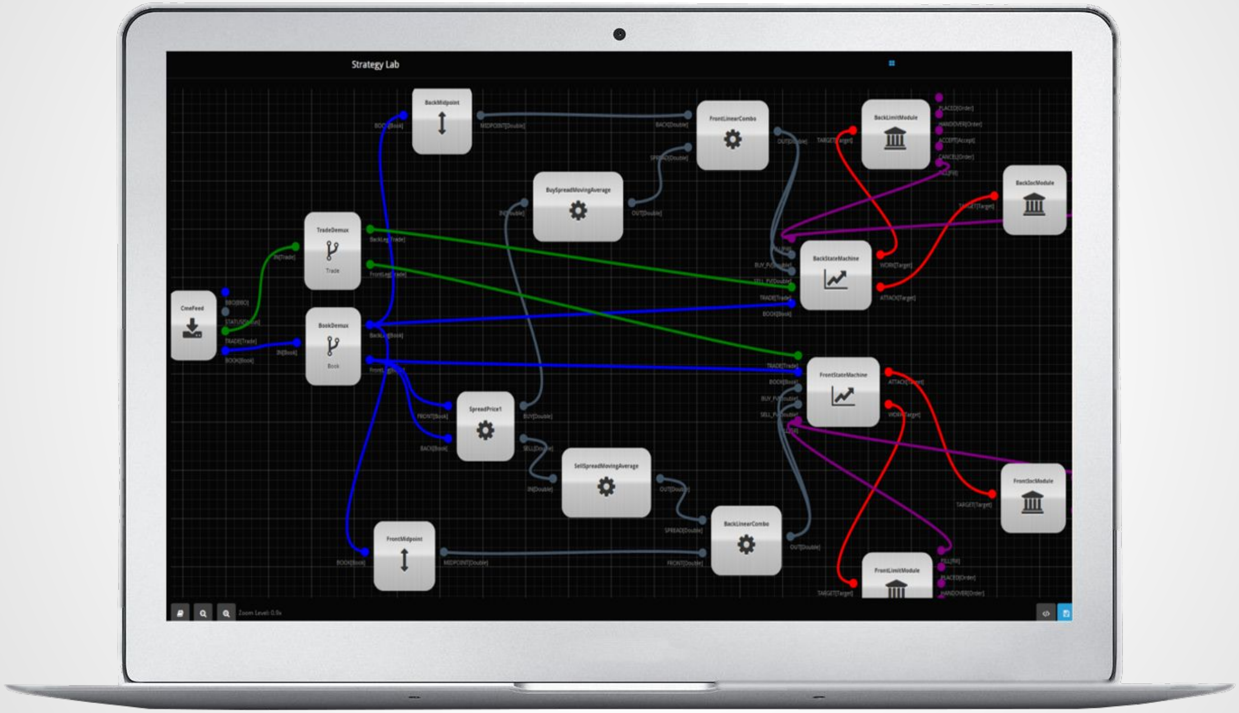
StrategyLab



Flow Based Strategy Development

Motivation

Along with the Attila platform, we inherited upwards of 75 different strategies. Each strategy a unique snowflake which at one point required a large team of dev, ops and trading talent to create, run and maintain. The strategies all had unique database schemas and architectural requirements. Most of these strategies were surely started by a “copy and paste” process where an existing strategy would be copied and modified. This practice resulted in a huge amount of code duplication, and “reinventing the wheel” at each strategy iteration. Furthermore, the primary command and control interface employed was SSH (read black-box); which although ubiquitous and dependable, is difficult to present to the average trader as a realistic user interface.

Fortunately, most of the strategy scalability and maintainability issues were propagated by the individual trading teams, and the core architecture team did a good job at maintaining a balance between performance and abstraction. Observing the multitude of strategy iterations available, it is difficult to think that there isn’t a better way!

The StrategyLab Approach

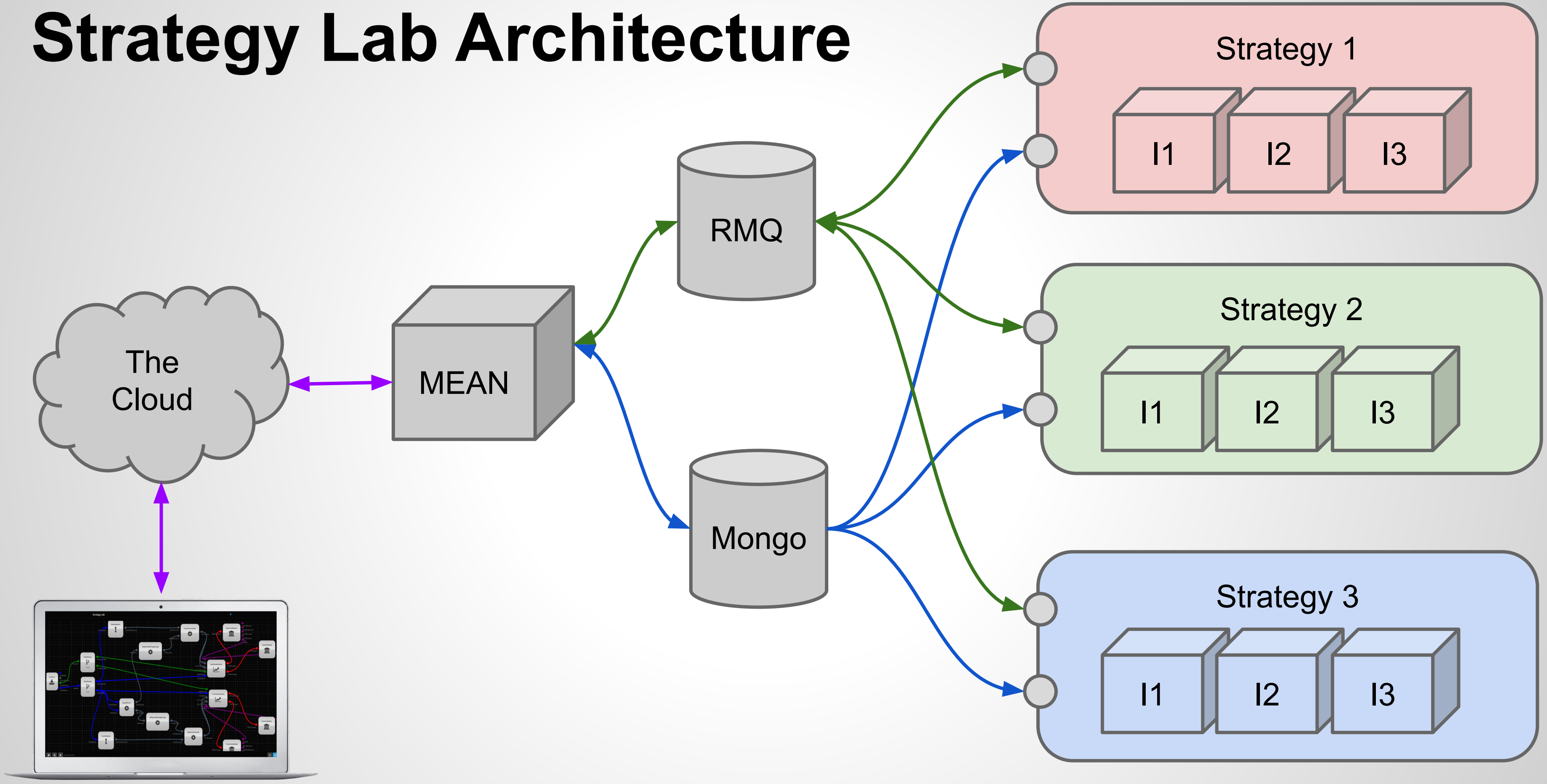
With the development of strategy lab, we address the aforementioned difficulties by adhering to the following principles:

- Don’t Repeat Yourself.
- Create a graphical flow based “Domain Specific Language” which encapsulates the fundamental components found throughout Attila strategy development.
- Maintain the highest possible performance in communication between components.
- Fully leverage the existing core Attila platform and Boost Python scripting interface.

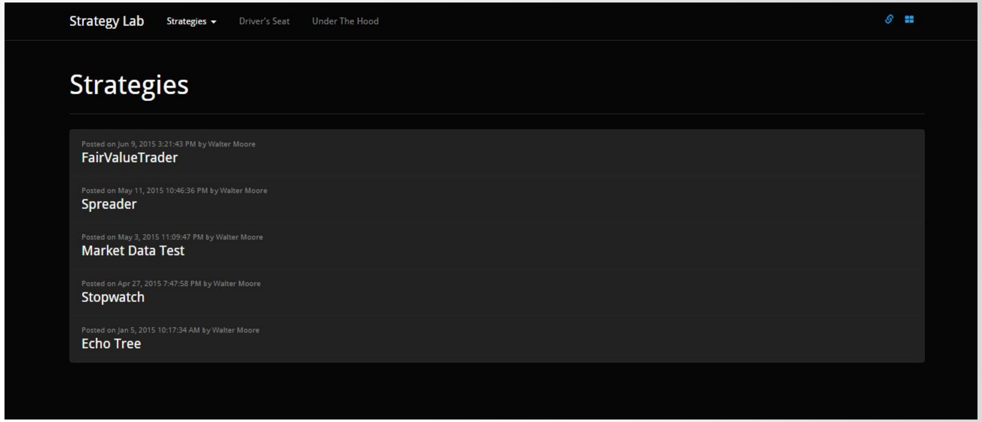
The StrategyLab Advantage

- Visually design strategies before having to write a single line of code.
- Safely change message routing between components without compilation while maintaining type safety.
- Code generation takes care of a substantial portion of strategy boilerplate.
- Super easy strategy replication.
- Not limited to trading system design. We can create DSLs for any environment.
- Easily control many instances of a strategy colocated around the world from a single screen.
- Cloud based, yet top of class performance.

Strategy Lab Architecture



What is a strategy?



Strategies

Posted on Jun 9, 2015 3:21:43 PM by Walter Moore
FairValueTrader

Posted on May 11, 2015 10:46:36 PM by Walter Moore
Spreader

Posted on May 3, 2015 11:09:47 PM by Walter Moore
Market Data Test

Posted on Apr 27, 2015 7:47:58 PM by Walter Moore
Stopwatch

Posted on Jan 5, 2015 10:17:34 AM by Walter Moore
Echo Tree

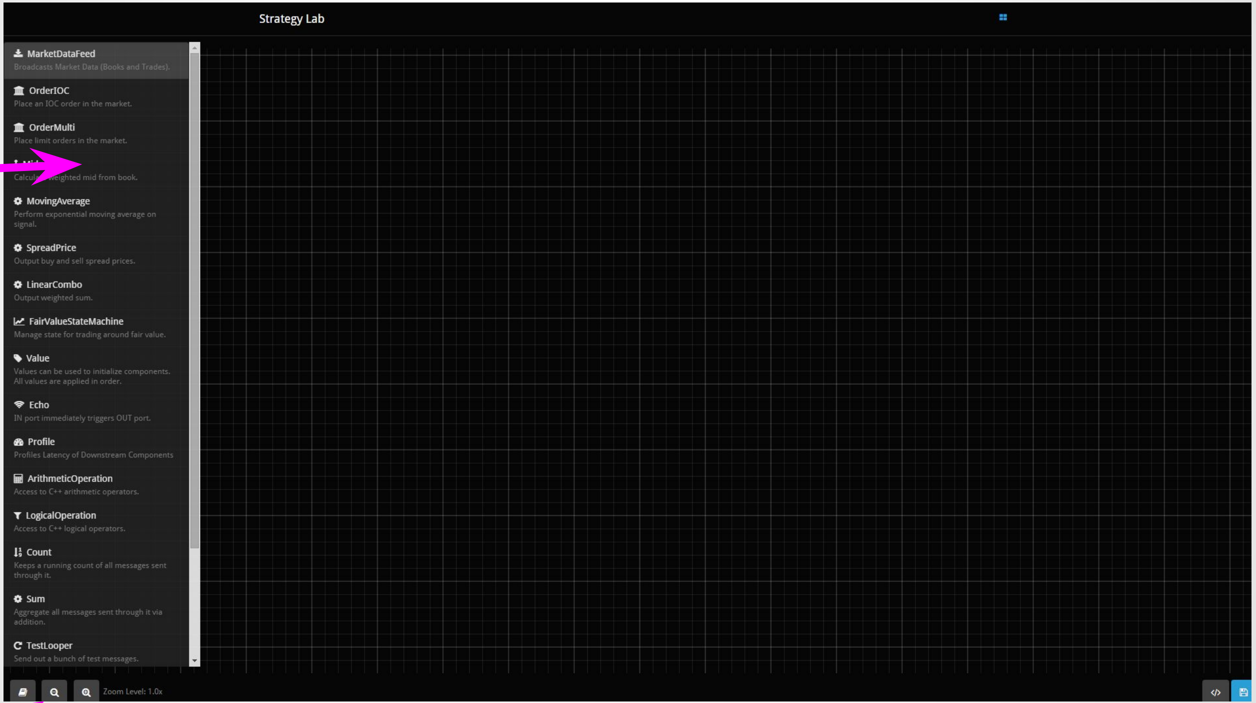
- A set of connected components (i.e. a graph).
- Definitely not limited to trading.

Under The Hood

- Surface for creating and manipulating components and connections.
- Generic port types.
- Port groups allow for an adjustable number of in/out ports (e.g. 1 per ticker symbol).

Clean Slate

Drag and drop components from the library onto the graph surface.


Strategy Lab

MarketDataFeed
Broadcasts Market Data (Books and Trades).

OrderIOC
Place an IOC order in the market.

OrderMulti
Place limit orders in the market.

Mi
Calcul weighted mid from book.

MovingAverage
Perform exponential moving average on signal.

SpreadPrice
Output buy and sell spread prices.

LinearCombo
Output weighted sum.

FairValueStateMachine
Manage state for trading around fair value.

Value
Values can be used to initialize components. All values are applied in order.

Echo
IN port immediately triggers OUT port.

Profile
Profiles Latency of Downstream Components

ArithmeticOperation
Access to C++ arithmetic operators.

LogicalOperation
Access to C++ logical operators.

Count
Keeps a running count of all messages sent through it.

Sum
Aggregate all messages sent through it via addition.

TestLooper
Send out a bunch of test messages.

Zoom Level: 1.0x

Zoom In/Out, Toggle Library Panel

Import / Export, Save

Where does the library come from?

```
{
  "PORT_TYPES": [...],
  "COMPONENT_TYPES": [...],
  "GENERIC_COMPONENT_TYPES": [...]
}
```

Central to the StrategyLab system is the library.json file. This file designates port (i.e. message) and component types. From this file, StrategyLab knows how to render components dropped onto the graph surface. The StrategyLab back end also uses library.json to generate all the base classes and python exports for components.

Example Port Type Definition:

```json
{
  "module": "Finance",
  "header": "messages.hpp",
  "name": "Book",
  "operators": false,
  "export": true,
  "testValue": "Book()",
  "type": "Book",
  "color": "Blue"
}
```

Port types tell the StrategyLab front end what color these connections should be. The back end uses port types to generate code for all generic realizations.

Example Component Definition

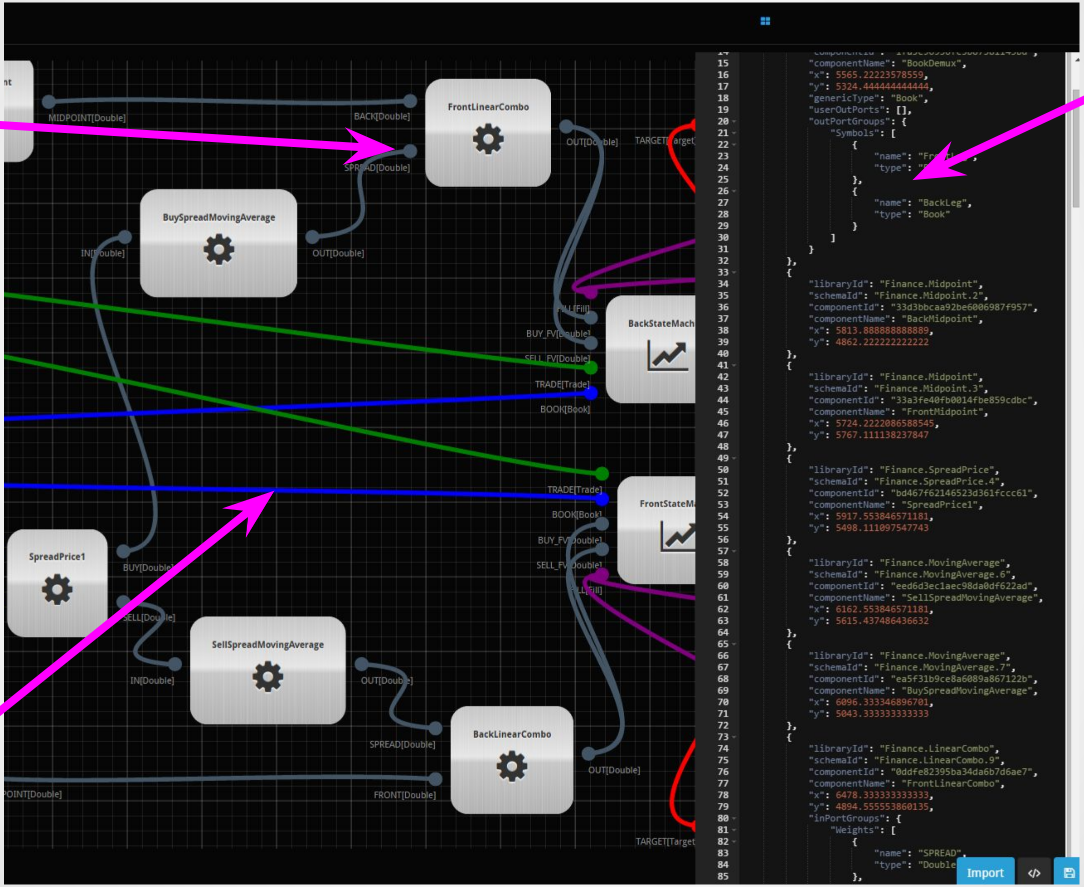
Component definitions tell the StrategyLab front end what ports it needs to render, as well as some metadata for user experience. Parameters accessible on the “Driver’s Seat” page are defined via exportFields, inPortGroups or outPortGroups.

```
{
    "module": "Finance",
    "name": "MarketDataFeed",
    "type": "MarketDataFeed",
    "header": "marketDataFeed.hpp",
    "description": "Broadcasts Market Data (Books and Trades).",
    "image": "download",
    "inPorts": [],
    "outPorts": [
        {
            "name": "STATUS",
            "type": "Status"
        },{
            "name": "BBO",
            "type": "BBO"
        },{
            "name": "BOOK",
            "type": "Book"
        },{
            "name": "TRADE",
            "type": "Trade"
        }
    ],
    "exportDefs": [
        {
            "cppName": "registerInstrument",
            "pythonName": "RegisterInstrument"
        }
    ]
}
```

Drag and drop connections between component ports.

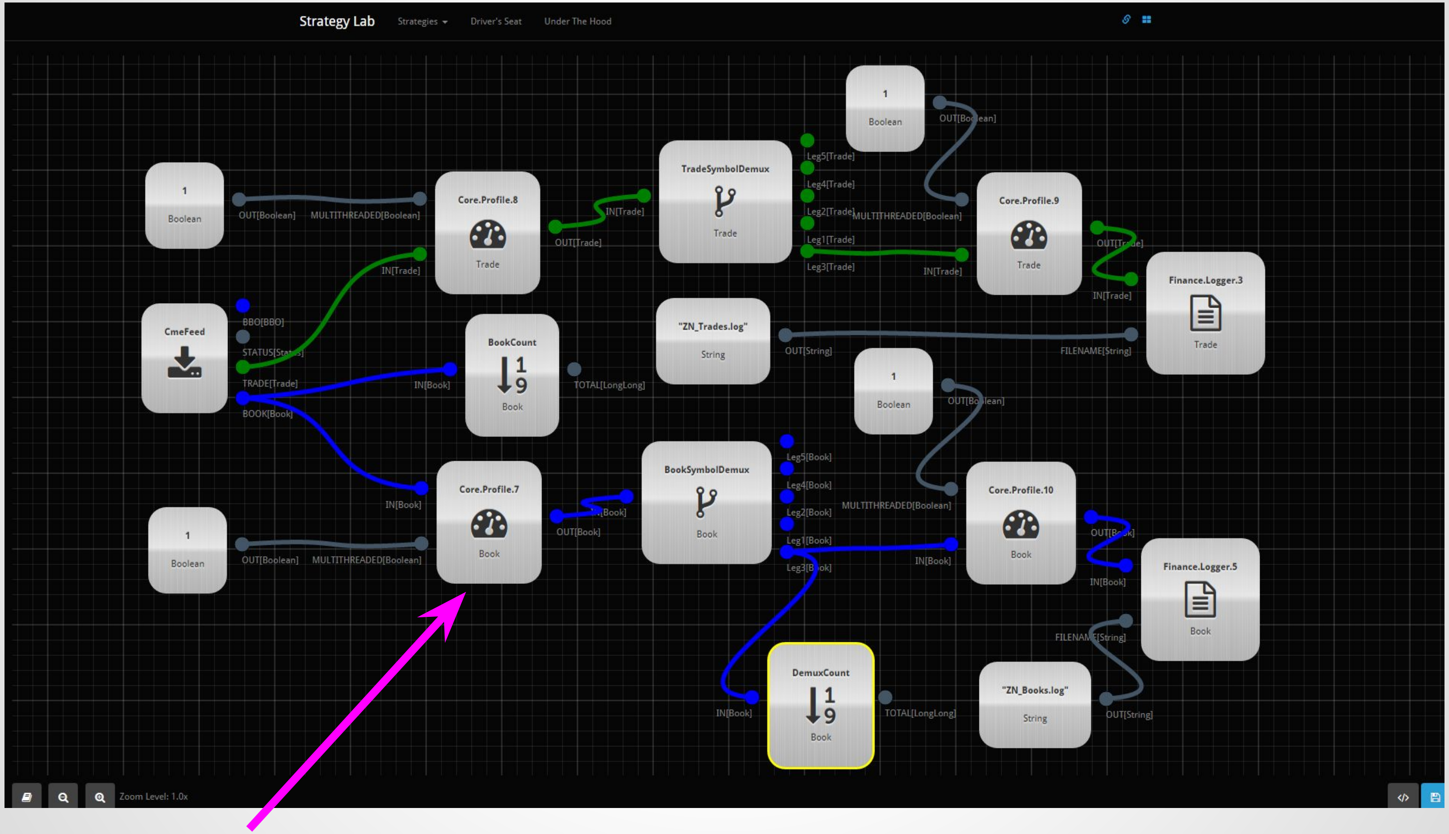
Port and connection color indicate message type.

Import or export graph as a plain text JSON object.

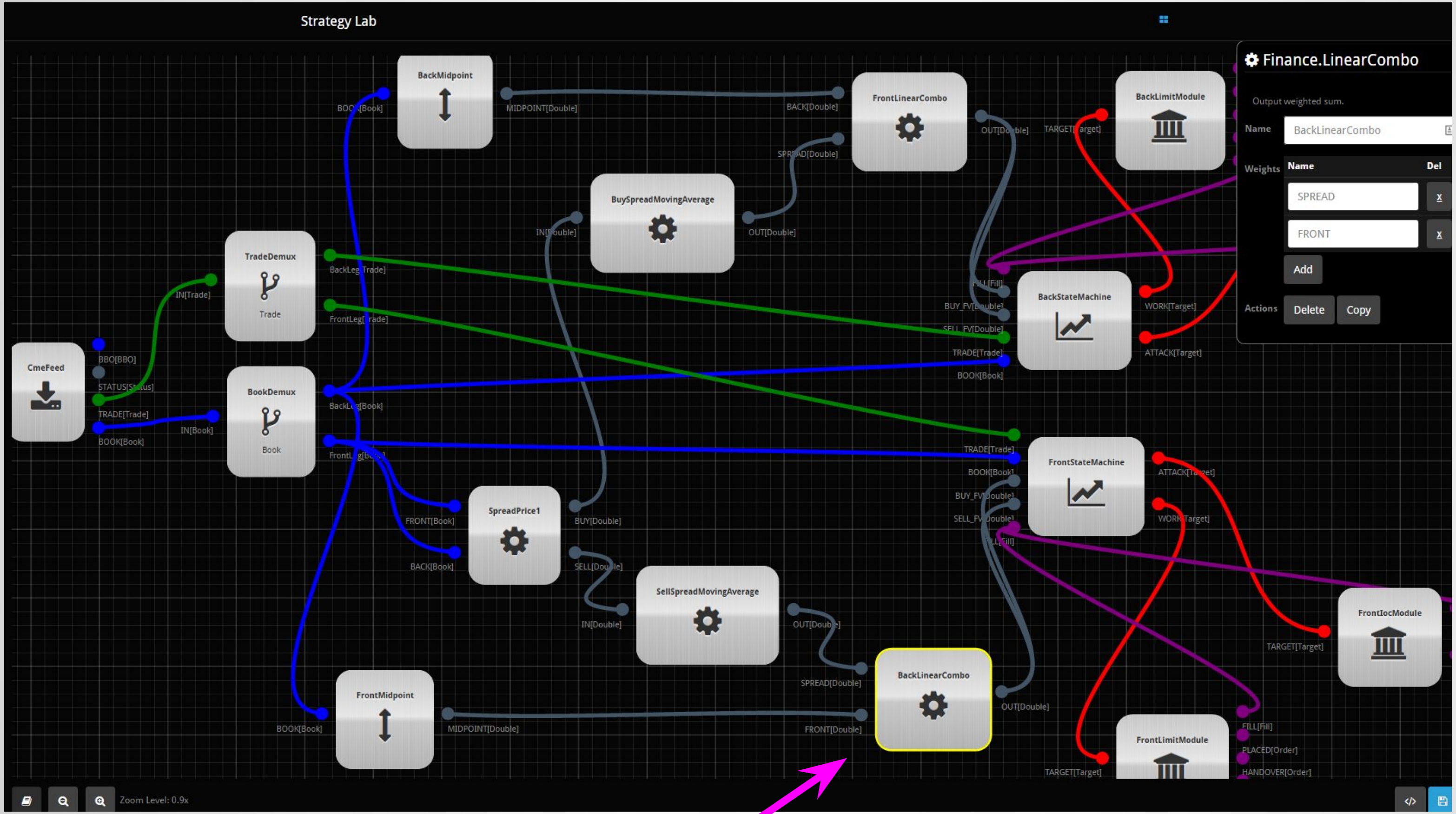


```
            "componentName": "BookDemux",
            "x": 5565.22223578559,
            "y": 5324.444444444444,
            "genericType": "Book",
            "userOutPorts": [],
            "outPortGroups": {
                "Symbols": [
                    {
                        "name": "Fr
                        "type": "
                    },
                    {
                        "name": "BackLeg",
                        "type": "Book"
                    }
                ]
            }
        },
        {
            "libraryId": "Finance.Midpoint",
            "schemaId": "Finance.Midpoint.2",
            "componentId": "33d3bbcaa92be6006987f957",
            "componentName": "BackMidpoint",
            "x": 5813.888888888889,
            "y": 4862.222222222222
        },
        {
            "libraryId": "Finance.Midpoint",
            "schemaId": "Finance.Midpoint.3",
            "componentId": "33a3fe40fb0014fbe859cdbc",
            "componentName": "FrontMidpoint",
            "x": 5724.222086588545,
            "y": 5767.111138237847
        },
        {
            "libraryId": "Finance.SpreadPrice",
            "schemaId": "Finance.SpreadPrice.4",
            "componentId": "bd467f62146523d361fccc61",
            "componentName": "SpreadPrice1",
            "x": 5917.553846571181,
            "y": 5498.111097547743
        },
        {
            "libraryId": "Finance.MovingAverage",
            "schemaId": "Finance.MovingAverage.6",
            "componentId": "eed6d3ec1aec98da0df622ad",
            "componentName": "SellSpreadMovingAverage",
            "x": 6162.553846571181,
            "y": 5615.437486436632
        },
        {
            "libraryId": "Finance.MovingAverage",
            "schemaId": "Finance.MovingAverage.7",
            "componentId": "ea5f31b9ce8a6089a867122b",
            "componentName": "BuySpreadMovingAverage",
            "x": 6096.333346896701,
            "y": 5043.333333333333
        },
        {
            "libraryId": "Finance.LinearCombo",
            "schemaId": "Finance.LinearCombo.9",
            "componentId": "0ddfe82395ba34da6b7d6ae7",
            "componentName": "FrontLinearCombo",
            "x": 6478.333333333333,
            "y": 4894.555553860135,
            "inPortGroups": {
                "Weights": [
                    {
                        "name": "SPREAD",
                        "type": "Double
                    },
```

Market Data Logging Example



Profiling Component For Inline Sampling



Finance.LinearCombo

Output weighted sum.

Name: BackLinearCombo

Weights

Name	Del
SPREAD	x
FRONT	x

Add

Actions: Delete Copy

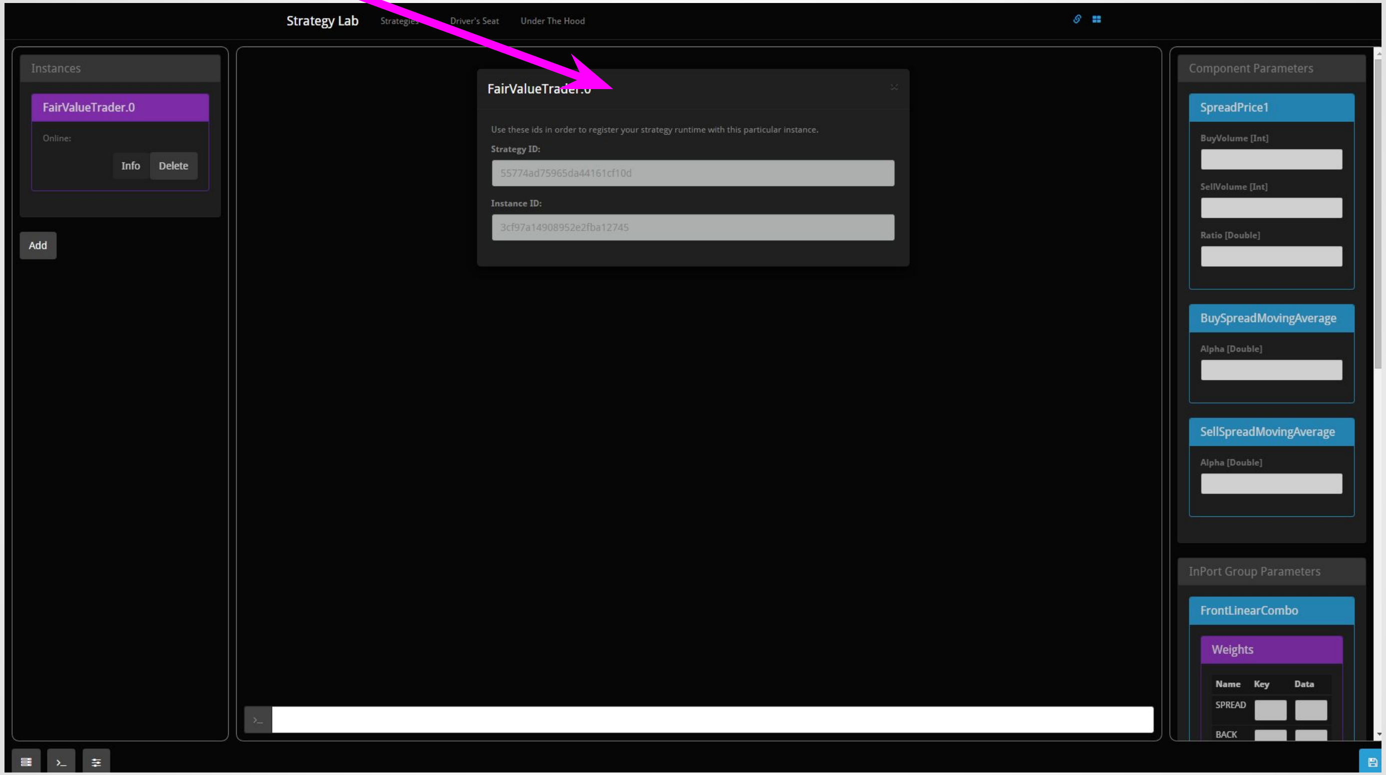
Port groups allow user defined filtering by symbol, etc.

Select Component

Driver’s Seat

- Instance configuration.
- Python console.
- Command and control.
- Dashboard.

Register StrategyLab runtime instance with this info:



FairValueTrader.0

Use these ids in order to register your strategy runtime with this particular instance.

Strategy ID: 55774ad75965da44161cf10d

Instance ID: 3cf97a14908952e2fba12745

Instance Parameters:

Toggle Panels:

Save:

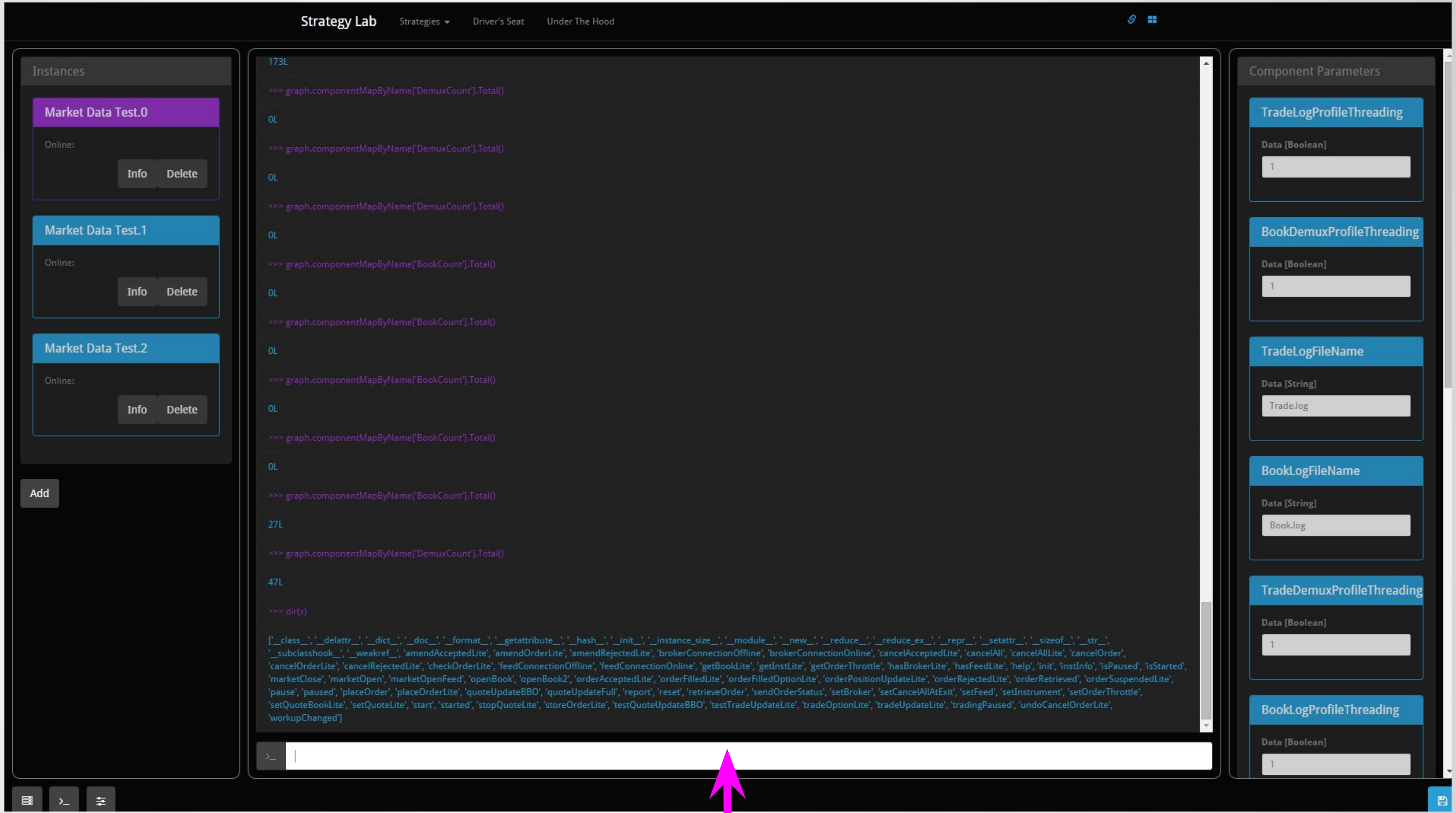
A simple StrategyLab runtime:

```
import graphtools as strategy_lab_graph
import consoletools as strategy_lab_console

strategyId = "54aab91ecaef24bc05d27f1a"
instanceId = "fd6673b0a36567316afb4bd8"

echoTreeGraph = strategy_lab_graph.getMongoGraph(strategyId,'library.json', instanceId)
strategy_lab_console.start(strategyId, instanceId,locals=dict(globals(), **locals()));
```

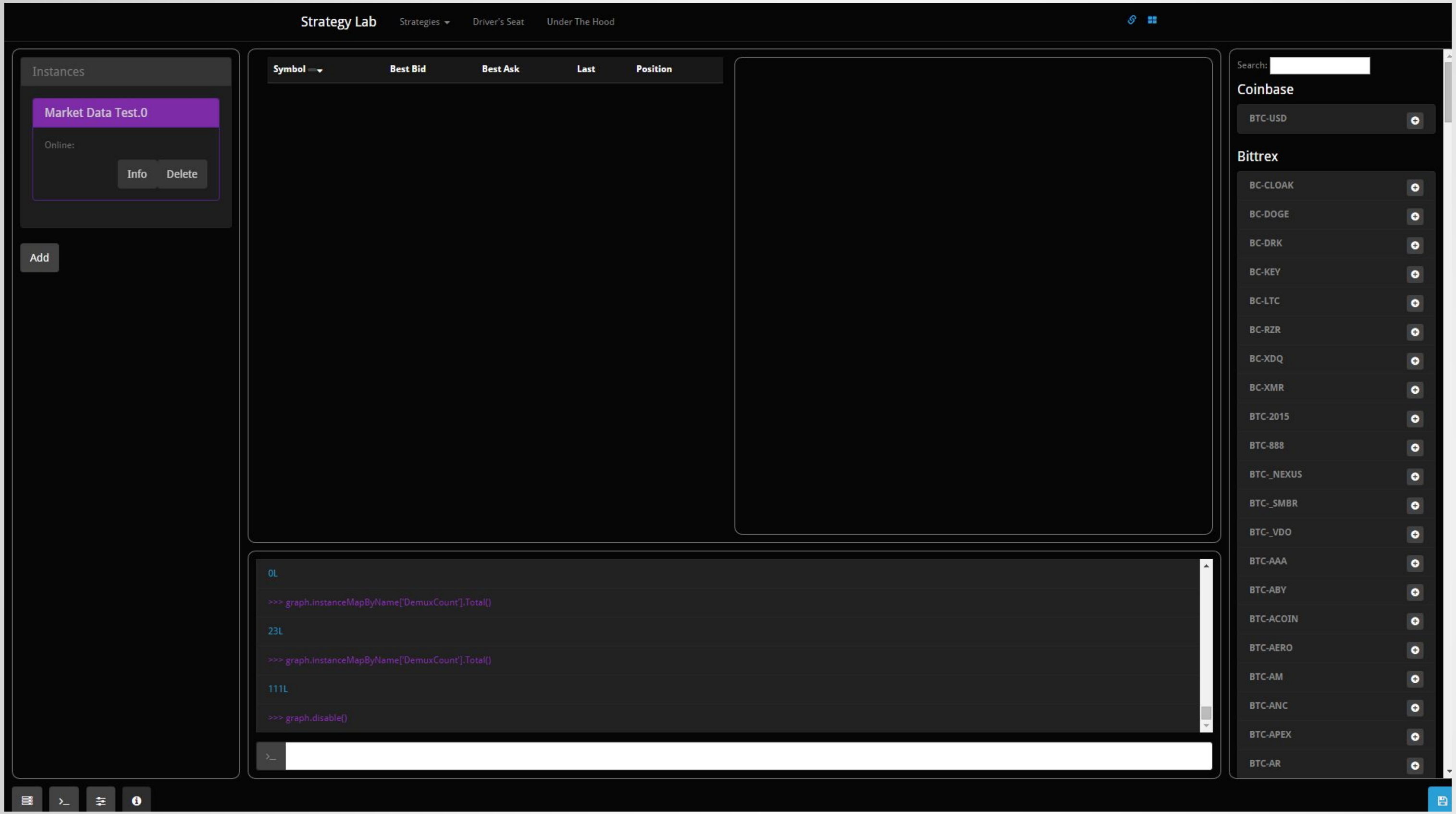
Each instance has its own python console, set of parameters and (todo) dashboard.



```
173L
>>> graph.componentMapByName['DemuxCount'].Total()
0L
>>> graph.componentMapByName['DemuxCount'].Total()
0L
>>> graph.componentMapByName['DemuxCount'].Total()
0L
>>> graph.componentMapByName['BookCount'].Total()
0L
>>> graph.componentMapByName['BookCount'].Total()
0L
>>> graph.componentMapByName['BookCount'].Total()
0L
>>> graph.componentMapByName['BookCount'].Total()
0L
>>> graph.componentMapByName['BookCount'].Total()
27L
>>> graph.componentMapByName['DemuxCount'].Total()
47L
>>> dir(s)
['__class__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattribute__', '__hash__', '__init__', '__instance_size__', '__module__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '__weakref__', 'amendAcceptedLite', 'amendOrderLite', 'amendRejectedLite', 'brokerConnectionOffline', 'brokerConnectionOnline', 'cancelAcceptedLite', 'cancelAll', 'cancelAllLite', 'cancelOrder',
'cancelOrderLite', 'cancelRejectedLite', 'checkOrderLite', 'feedConnectionOffline', 'feedConnectionOnline', 'getBookLite', 'getInstLite', 'getOrderThrottle', 'hasBrokerLite', 'hasFeedLite', 'help', 'init', 'instInfo', 'isPaused', 'isStarted',
'marketClose', 'marketOpen', 'marketOpenFeed', 'openBook', 'openBook2', 'orderAcceptedLite', 'orderFilledLite', 'orderFilledOptionLite', 'orderPositionUpdateLite', 'orderRejectedLite', 'orderRetrieved', 'orderSuspendedLite',
'pause', 'paused', 'placeOrder', 'placeOrderLite', 'quoteUpdateBBO', 'quoteUpdateFull', 'report', 'reset', 'retrieveOrder', 'sendOrderStatus', 'setBroker', 'setCancelAllAtExit', 'setFeed', 'setInstrument', 'setOrderThrottle',
'setQuoteBookLite', 'setQuoteLite', 'start', 'started', 'stopQuoteLite', 'storeOrderLite', 'testQuoteUpdateBBO', 'testTradeUpdateLite', 'tradeOptionLite', 'tradeUpdateLite', 'tradingPaused', 'undoCancelOrderLite',
'workupChanged']
```

TradeLogProfileThreading	Data [Boolean]: 1
BookDemuxProfileThreading	Data [Boolean]: 1
TradeLogFileName	Data [String]: Trade.log
BookLogFileName	Data [String]: Book.log
TradeDemuxProfileThreading	Data [Boolean]: 1
BookLogProfileThreading	Data [Boolean]: 1

Fully interactive python console allows command line control of the sandboxed StrategyLab runtime environment.



```
0L
>>> graph.instanceMapByName['DemuxCount'].Total()
23L
>>> graph.instanceMapByName['DemuxCount'].Total()
111L
>>> graph.disable()
```

Coinbase

- BTC-USD

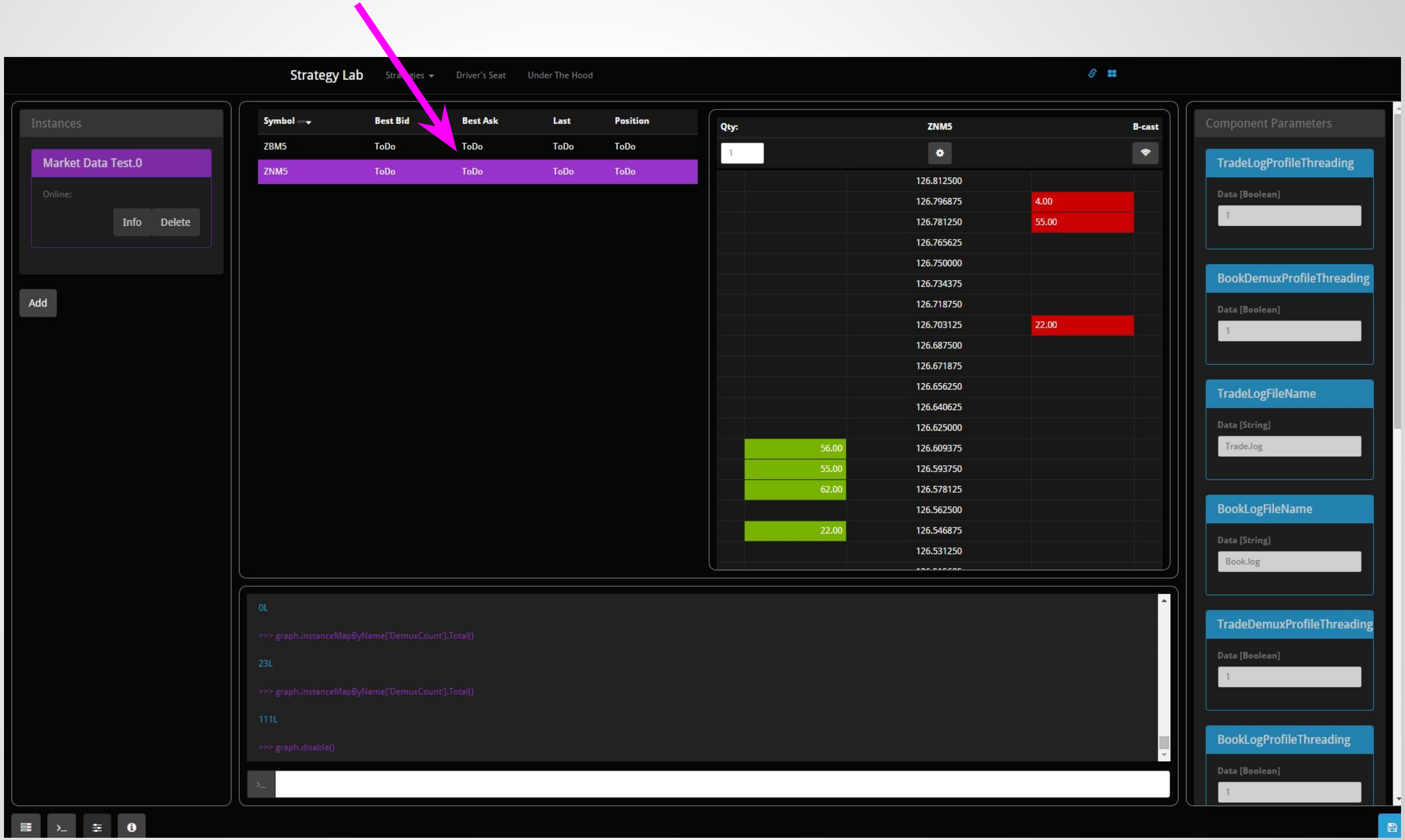
Bittrex

- BC-CLOAK
- BC-DOGE
- BC-DRK
- BC-KEY
- BC-LTC
- BC-RZR
- BC-XDQ
- BC-XMR
- BTC-2015
- BTC-888
- BTC-_NEXUS
- BTC-_SMBR
- BTC-_VDO
- BTC-AAA
- BTC-ABY
- BTC-ACOIN
- BTC-AERO
- BTC-AM
- BTC-ANC
- BTC-APEX
- BTC-AR

Search for instruments here and add them to the currently selected instance.

Toggle Instrument Selector

Select an instance instrument in order to populate book depth



Symbol	Best Bid	Best Ask	Last	Position
ZBM5	ToDo	ToDo	ToDo	ToDo
ZNM5	ToDo	ToDo	ToDo	ToDo

Qty: 1 — ZNM5 — B-cast

	126.812500		
	126.796875	4.00	
	126.781250	55.00	
	126.765625		
	126.750000		
	126.734375		
	126.718750		
	126.703125	22.00	
	126.687500		
	126.671875		
	126.656250		
	126.640625		
	126.625000		
56.00	126.609375		
55.00	126.593750		
62.00	126.578125		
	126.562500		
22.00	126.546875		
	126.531250		

```
0L
>>> graph.instanceMapByName['DemuxCount'].Total()
23L
>>> graph.instanceMapByName['DemuxCount'].Total()
111L
>>> graph.disable()
```

TradeLogProfileThreading
Data [Boolean]: 1

BookDemuxProfileThreading
Data [Boolean]: 1

TradeLogFileName
Data [String]: Trade.log

BookLogFileName
Data [String]: Book.log

TradeDemuxProfileThreading
Data [Boolean]: 1

BookLogProfileThreading
Data [Boolean]: 1

Development Queue

Short Term:

- Fair Value Trader
- JSON component configs.
- Unify instrument models.
- Python based components.
- RMQ component.
- Market View integration.
- Queued/threaded components.
- General purpose chart.

Long Term:

- Split out library modules.
- Component library editor.
- Dynamic message types.
- Store script in DB and edit from SL.
- Dashboard Generation.
- State machine editor.