# Table of Contents

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

2. Installation ............................................................................................................................ 3  
   2.1. Setting up your AWS Trial Instance ............................................................................... 3  
   2.2. Setting up a Windows Machine with the AlgoTrader Installer .................................. 8  
   2.3. Instance Maintenance ................................................................................................... 10

3. Starting a Trading Strategy .................................................................................................... 11  
   3.1. Starting a Back Test ...................................................................................................... 12  
   3.2. Starting Live Trading ................................................................................................. 14

4. Creating a Trading Strategy .................................................................................................. 18  
   4.1. AlgoTrader Strategy Wizard ....................................................................................... 19  
   4.2. Adding Strategy Logic ............................................................................................... 21  
   4.3. Adding Strategy Logic in Python ................................................................................. 26

5. Managing data ....................................................................................................................... 33  
   5.1. Reference Data ............................................................................................................ 35  
   5.2. Historical Data ............................................................................................................ 35

6. Cryptocurrency Trading ........................................................................................................ 37

7. Cryptocurrency Trading with Python ..................................................................................... 40

8. Shutting down the AWS System ............................................................................................ 43
List of Figures

3.1. IB Demo Account Login ........................................................................................................ 14
Introduction

The AlgoTrader Quick Start Guide is based on the AlgoTrader 30-day trial provided either via Amazon AWS\(^1\) or via an installer, both of which can be requested from our sales team.

The AlgoTrader 30-day trial version includes a fully functional AlgoTrader installation as well as the following example strategies:

- **Box Strategy**\(^2\)
- **Break Out Strategy**\(^3\)
- **Exponential Moving Average Strategy**\(^4\)
- **IPO Strategy**\(^5\)
- **Pairs Trading Strategy**\(^6\)
- **Random Strategy**\(^7\)
- **Spreader Strategy**\(^8\)
- **Dividend Capture Strategy**\(^9\)
- **NLP Strategy**\(^10\)
- **Delta Hedge Strategy**\(^11\)
- **Short Strangle Strategy**\(^12\)
- **Market Making Strategy**\(^13\)
- **RSI Strategy**\(^14\)

There is also a simple Python strategy in the examples sub-directory *python-strategies*, which shows how to use AlgoTrader as a market data supplier and execution engine for a Python strategy through WebSocket and REST interfaces.

\(^1\) https://aws.amazon.com/
\(^2\) http://doc.algotrader.com/html/Box_Strategy.html
\(^3\) http://doc.algotrader.com/html/BreakOut_Strategy.html
\(^6\) http://doc.algotrader.com/html/PairsTrading_Strategy.html
\(^7\) http://doc.algotrader.com/html/Random_Strategy.html
\(^8\) http://doc.algotrader.com/html/Spreader_Strategy.html
\(^12\) http://doc.algotrader.com/html/Short_Strangle.html
\(^13\) http://doc.algotrader.com/html/Marketmaker_Strategy.html
Since AlgoTrader 5.1 it is also possible to fully integrate a Python Strategy with AlgoTrader and to make use of all the AlgoTrader strategy services directly (including backtesting) through Py4J, as highlighted in Section 4.3, “Adding Strategy Logic in Python”.

The AlgoTrader 30-day trial version contains the following pre-installed components:

- An OpenJDK version compatible with AlgoTrader
- The AlgoTrader Server and Example Strategies
- IntelliJ IDEA
- PyCharm CE IDE
- Python 2.7 and 3.7
- Pip package installer for Python
- MySql Database
- InfluxDB Database
- dbForge Studio Express
- Interactive Brokers Trader Workstation
- Notepad++

In addition you will also need a modern browser and if you want to visualise the backtest results, MS Excel. To run strategies in a more productive manner, we also recommend using the Interactive Brokers Gateway instead of the pre-installed Trader Workstation. The workstation however also lets you see your orders and positions at IB.

**Warning**

It is prohibited to reverse engineer, decompile, disassemble, or copy any parts of the AlgoTrader 30-day trial
Chapter 2. CONFIDENTIAL

Installation

2.1. Setting up your AWS Trial Instance

Warning

Amazon AWS usage cost will apply based on the instance type select. For further details, please visit https://aws.amazon.com/ec2/pricing/

The following steps will guide through the installation of a Windows AWS Instance containing the AlgoTrader 30-day trial version

1. Open the Amazon AWS console:

https://console.aws.amazon.com/console/home

and Login using the Amazon username and password.

2. In the upper right-hand corner of the screen make sure the N. Virginia Region is selected:

3. Select EC2

4. Click Launch Instance in the middle of the screen

Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Launch Instance

Note: Your instances will launch in the US East (N. Virginia) region
5. In the menu on the left-hand side select My AMIs and check Shared with me

6. Select the AlgoTrader-x.x.x-Trial-WIN-xxxx.xx.xx

    AlgoTrader-
    Root device type: ebs Virtualization type: hvm Owner: 766273269739

   Trial-WIN-

---

**Important**

The AlgoTrader-x.x.x-Trial-WIN-xxxx.xx.xx image is only available in the N. Virginia Region but not in any other Regions. It is thus necessary that the N. Virginia Region is selected in the top-right corner of the screen.

7. On the next screen select the Instance Type. We recommend at least instance type t2.medium, ideally instance type m5.large

<table>
<thead>
<tr>
<th>Type</th>
<th>vCPUs</th>
<th>Memory (GiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2.medium</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>m5.large</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>
Warning

Amazon AWS usage cost depends on the instance type selected. For further details, please visit: https://aws.amazon.com/ec2/pricing/

8. Click Review and Launch on the bottom right of the screen

9. Click Launch on the bottom right of the screen

10. On the Dialog that shows select Proceed without a key pair, select I acknowledge... and click Launch Instance

    Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about removing existing key pairs from a public AMI.

11. Click View Instances on the bottom right of the screen

12. On the next screen, you can see the Instance starting up. Wait until it is running and note the Public IP address.
Note

It will take at least 4 minutes for the Instance to startup and become available.

13. Use Microsoft Remote Desktop to connect to the instance by typing in the Public IP address that was noted in the previous step.

![Remote Desktop Connection](image)

**Note**

- You need to use Remote Desktop Connection (RDP), a Web Browser will NOT work for this.
- For more information on Remote Desktop Connections please visit: Windows¹, Mac² and Linux³.

14. Specify **User name** and **Password** that was provided in the Email after signing up for the AlgoTrader free 30-day trial.

15. On the next dialog select Don't ask... and click Yes.

16. The Amazon Instance including AlgoTrader is now ready for usage!

Note

- Amazon Windows Instances tend to run a bit slow when they are first created. Responsiveness will however increase after some time.
2.2. Setting up a Windows Machine with the AlgoTrader Installer

On the machine, where you wish to install the trial, download the installer using the link provided to you by sales and run it. If you get a message similar to the below, press More Info, then Run Anyway and follow the setup procedure.

You can opt out of some of the applications at installation time. Note that this manual assumes you did not. The installed software should be sandboxed and will not interfere with other installations of the same software you might have on your machine.

Now start the IntelliJ IDEA using the Idea for Algotrader desktop icon and wait for the examples project to be read.

---

1. The performance of the Amazon Instance also depends on the instance type selected in step 7.

AWS Security

The trial image comes with minimal security configurations. We recommend that you setup additional security precautions if you have the technical knowledge for the tasks below:

- Change the Administrator password when you first log in.
- Restrict the IPs that are allowed to access your machine: see [AWS Access Restrictions](https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/authorizing-access-to-an-instance.html) (adding a Rule for Inbound RDP Traffic to a Windows Instance) - be sure that the IP addresses you use to access the instance are static (if in doubt, skip this part).
- Consider creating regular backups of your AWS instance to make restoring your instance faster if ever required: [creating AWS images](https://docs.aws.amazon.com/toolkit-for-visual-studio/latest/user-guide/tkv-create-ami-from-instance.html) or [setting up AWS backups](https://aws.amazon.com/de/blogs/aws/aws-backup-ec2-instances-efs-single-file-restore-and-cross-region-backup/).
To be able to run AlgoTrader, you first need to run a Maven install: at the bottom right of the screen, press Enable auto-import, then install the Maven project Bootstrap in the Maven menu on the right, then repeat with the AlgoTrader Example Strategies.

Once that has completed (you should see the log BUILD SUCCESS), you are ready to use AlgoTrader on your machine.
2.3. Instance Maintenance

**Note**

- Note that both the trial image and the AWS installer have disabled updates for the installed IDEs and **DBForge Studio**. If you intend to use this installation as a development machine, you will need to update them manually.

- Automatic Windows updates are disabled in the image. You should do the same in your local installation because you do not want the box restarting while your strategy is running. You will however have to occasionally perform manual Windows updates.
Starting a Trading Strategy

This section gives a quick introduction on how to start a trading strategy by discussing the Box example strategy\(^1\).

To start any of the other strategies please consult the relevant parts in the documentation:

- Break Out Strategy\(^2\)
- Exponential Moving Average Strategy\(^3\)
- IPO Strategy\(^4\)
- Pairs Trading Strategy\(^5\)
- Random Strategy\(^6\)
- Spreader Strategy\(^7\)
- Dividend Capture Strategy\(^8\)
- NLP Strategy\(^9\)
- Delta Hedge Strategy\(^10\)
- Short Strangle Strategy\(^11\)
- Market Making Strategy\(^12\)
- RSI Strategy\(^13\)

First you to start the AlgoTrader IntelliJ IDE (Integrated Development Environment) using the Idea for AlgoTrader icon on the desktop.

As a first step, please copy the license key that was provided in the Email after signing up for the AlgoTrader free 30-day trial into the file `algotrader-conf/src/main/resources/conf.properties`.

---

\(^1\) [http://doc.algotrader.com/html/Box_Strategy.html](http://doc.algotrader.com/html/Box_Strategy.html)
\(^12\) [http://doc.algotrader.com/html/Marketmaker_Strategy.html](http://doc.algotrader.com/html/Marketmaker_Strategy.html)
3.1. Starting a Back Test

The back-test in this example strategy runs with CSV files. If you have a historical data provider that AlgoTrader supports, you could also run vs. InfluxDB and would have to load the historical data first by running a HistoricalDataStarter (see Historical Data).⁴¹

Launch the SimulationStarter-simulate-box by first clicking the downward facing arrow next to the green start icon (top right), selecting SimulationStarter-simulate-box and pressing the green start icon.

⁴¹ http://doc.algotrader.com/html_single/index.html#Historical_Data
The system will now perform a back test based on historical data.

Once the back test has finished, the Excel based back test report will prompt to be opened. For this, it would be beneficial if you installed Excel on the machine first. Alternatively, you can move the files under Desktop\Algotrader\Examples\examples\box\files\report\ to machine that has Excel installed and open BackTestReport.xlsm.

For further information regarding back testing, visit the following chapters in the documentation:

- **Strategy Backtesting**[^15]
- **Starting a Strategy in Simulation Mode**[^16]

[^16]: http://doc.algotrader.com/html/Starting_AlgoTrader.html#Simulation_Mode
3.2. Starting Live Trading

The following sections describe how to use AlgoTrader in Live Trading mode by using Interactive Brokers for market data and trading.

**Warning**

Do not use a live account until absolutely sure that the trading strategy works as expected. Until then use this demo account or an IB paper trading account.

If you do not have an Interactive Brokers account, you can create a demo account on the Interactive Brokers Trader Workstation (TWS) using the shortcut on the desktop.

The downside of the TWS demo account is that it uses delayed market data, your orders/trades/positions are reset at the end of each day. It also automatically shuts down on a daily basis.

After starting the TWS, go to Try the demo or Return to the demo and enter your email address.

![IB Demo Account Login](image)

**Figure 3.1. IB Demo Account Login**

The first time you start TWS, go to File / Global Configuration, then select API / Settings, make sure that Enable ActiveX and Socket Clients is enabled, Read-Only API is disabled and Socket port is set to 4001.
You can now launch the AlgoTrader example box strategy by first clicking the downward facing arrow next to the green start icon, selecting `EmbeddedStrategyStarter-box` and pressing the green start icon. This will start the AlgoTrader server as well as the Box strategy and will connect to the Interactive Brokers Gateway.

Once the system is started up it will automatically open the AlgoTrader UI (within the Chrome browser).
You can add and resize the strategy widget by pressing on the + on the Dashboard menu on the left.
For further information on live trading, please visit [Starting a Strategy in Live Trading Mode](http://doc.algotrader.com/html/Starting_AlgoTrader.html#Live_Trading_Mode)

For further information on the AlgoTrader Web UI please visit the AlgoTrader documentation regarding the [HTML5 UI](http://doc.algotrader.com/html/Client.html#HTML5_Client)
Creating a Trading Strategy

This section will give a quick introduction on how to create a trading strategy by discussing the EMA (Exponential Moving Average) Strategy.

**Note**

The AlgoTrader 30-day free trial already contains the final EMA strategy with all artifacts. In case you want to follow below steps please delete the existing EMA strategy first, by:

- Removing the existing Maven project by opening the examples pom.xml file, searching (CTRL +F) for the string `module>ema<` and removing up to 2 lines from that file where it is found. You will be asked if you want to remove them from the project too. Please confirm that by pressing yes.

- Deleting the code from disk by right-clicking on the ema strategy and selecting Delete.
4.1. AlgoTrader Strategy Wizard

The AlgoTrader Strategy Wizard provides an easy way to automatically create all artifacts necessary for an AlgoTrader based trading strategy. The Wizard can be started by right-clicking on the examples project and selecting New / Module.
which will bring up the following screen where you should select **Maven** on the left, check the **Create from archetype** box and in this case select the **algotrader-archetype-simple**, then press **Next**.

Configure the next screen as displayed on the picture and click **Next**. The **Version** number will be different for later AlgoTrader versions, so please leave that value as it was set automatically.

**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.

On the following page, add 2 additional name/value pairs using the + button:

- name: `ema`
- serviceNamne: `EMA`
When clicking Finish the Strategy Wizard will create a new project called ema.

4.2. Adding Strategy Logic

The Strategy Wizard also generated boiler plate code that needs to be replaced with the actual logic of the EMA strategy.

AlgoTrader strategies are regular Java programs. Due to this any type of java library or add-ons can be used. The EMA strategy is based on the TA4J\(^1\) library which contains a collection of over 100 technical indicators.

Now, double click the EMAService.java file which contains the main logic of the EMA strategy.

The header of the EMAService.java is already generated and no further changes are necessary. It contains the java class name (EMAService) as well as the name of the interface it is derived from (StrategyService).

\(^1\) https://github.com/mdeverdelhan/ta4j-origins
Note

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.

The next part of the EMAService.java contains settings the strategy will use. Three of them are already generated by the Wizard but a few more need to be added.

```java
private final long accountId = 100;
private final long securityId = 25;
private final BigDecimal orderQuantity = new BigDecimal("10000");
private final int emaPeriodShort = 10;
private final int emaPeriodLong = 20;
private final AdapterType defaultAdapterType = AdapterType.IB;

private TimeSeries series;
private DifferenceIndicator emaDifference;
```

- The `accountId` defines the id of the account the strategy will use for trading.
- The `securityId` will define the id of the instrument the strategy will trade.
- The `orderQuantity` is the number of contracts the strategy will trade.
- The `emaPeriodShort` is the look back period of the shorter EMA indicator.
- The `emaPeriodLong` is the look back period of the longer EMA indicator.
- The `defaultAdapterType` indicates we want to get market data from Binance by default.

In addition, the following two fields need to be defined:

- The `TimeSeries` object used by the exponential moving average indicators
- The `DifferenceIndicator` which will contain the difference between the short and the long EMA

Next, the Java Constructor for the `EMAService` class needs to be updated:

```java
public EMAService() {
    setStrategyName("EMA");

    init();
}
```

And the `init()` method called therein:
private void init() {
    this.series = new BaseTimeSeries();
    this.series.setMaximumBarCount(this.emaPeriodLong);

    ClosePriceIndicator closePriceIndicator = new ClosePriceIndicator(this.series);
    EMAIndicator emaShort = new EMAIndicator(closePriceIndicator, this.emaPeriodShort);
    EMAIndicator emaLong = new EMAIndicator(closePriceIndicator, this.emaPeriodLong);
    this.emaDifference = new DifferenceIndicator(emaShort, emaLong);
}

• First the EMAService constructor sets the name of the Strategy used during the back test.

• Next the TimeSeries object is initialized to a length of one Bar. In addition, the number of bars the Time Series is set (in this case 20 Bars).

• Next a ClosePriceIndicator is created which causes the system to look at closing prices of Bar events.

• Then both the short and the long EMA indicator need to be created by associating them with the ClosePriceIndicator and setting the lookbackPeriod (in this case 10 and 20).

• Last the DifferenceIndicator needs to be created which contains the difference between the sort EMA and the long EMA indicator.

Next, update the onInit (an AlgoTrader Live Cycle Method) method, which simply calls the init() method we defined earlier.

@Override
protected void onInit(LifecycleEventVO event) {
    init();
}

Next, update the onStart (an AlgoTrader Live Cycle Method) method, which will be called when the strategy starts up.

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

For further details please visit the AlgoTrader documentation regarding Life Cycle Events.²

² http://doc.algotrader.com/html_single/index.html#Strategy_Life_Cycle_Events
The `onStart` method calls `subscribeMarketDataEvent` of the `SubscriptionService` by passing the `strategyName` and the `securityId` of the instrument the strategy wants to receive market data for. The `SubscriptionService` is automatically made available to the strategy through Spring Auto Wiring.

Next, update the `onBar` method, which will be invoked on every incoming Bar:

```java
@override
public void onBar(BarVO bar) {
    this.series.addBar(toBar(bar));

    int i = this.series.getEndIndex();
    Num currentValue = this.emaDifference.getValue(i);
    Num previousValue = this.emaDifference.getValue(i - 1);

    if (currentValue.isPositive() && previousValue.isNegativeOrZero()) {
        sendOrder(Side.BUY);
    } else if (currentValue.isNegative() && previousValue.isPositiveOrZero()) {
        sendOrder(Side.SELL);
    }
}
```

- The method first calls the `addBar` method which will add the incoming Bar to the Time Series defined above.
- Next, the index `i` of the last element of the Time Series is retrieved.
- Then the value of the last and the second-last element of the `DifferenceIndicator` is retrieved.

Then the actual trading rules need to be defined:

- If the current value of the `DifferenceIndicator` is positive and the previous value was negative or zero a `BUY` order is sent. In other words, if the short EMA crossed above the long EMA a `BUY` order is sent.
- If the current value of the `DifferenceIndicator` is negative and the previous value was positive or zero a `SELL` order is sent. In other words, if the short EMA crossed below the long EMA a `SELL` order is sent.

The trading logic is depicted in the following chart also.
As the last item, create the `sendOrder` method, which will take care of constructing an order object and handing it over to the `OrderService`:

```java
private 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 `sendOrder` method creates a `MarketOrder` by using the `MarketOrderVOBuilder` and assigns the `strategyId`, `accountId`, `securityId`, `orderQuantity`, the order side (BUY or SELL) and finally calls `build` to create the `MarketOrder` object. The order object is then handed over to the `OrderService` which will execute the order. The `OrderService` is automatically made available to the strategy through Spring Auto Wiring.

For further details on how orders are please visit the AlgoTrader documentation regarding `Order Management`.

In addition the following Java import statements need to be added to the top:

---

import ch.algotrader.entity.marketData.BarVO;
import ch.algotrader.entity.trade.MarketOrderVO;
import ch.algotrader.entity.trade.MarketOrderVOBuilder;
import ch.algotrader.enumeration.AdapterType;
import ch.algotrader.enumeration.Side;
import ch.algotrader.service.StrategyService;
import ch.algotrader.vo.LifecycleEventVO;
import org.ta4j.core.BaseTimeSeries;
import org.ta4j.core.TimeSeries;
import org.ta4j.core.indicators.EMAIndicator;
import org.ta4j.core.indicators.helpers.ClosePriceIndicator;
import org.ta4j.core.indicators.helpers.DifferenceIndicator;
import org.ta4j.core.num.Num;
import java.math.BigDecimal;
import static ch.algotrader.util.TA4JUtil.toBar;

The implementation of the trading strategy is now finished a first back test can be started according to these instructions.\(^4\)

The EMA strategy is an example strategy based on Java code only. For details on how to build a trading strategy using Esper please visit the AlgoTrader documentation regarding Strategy Development\(^5\).

### 4.3. Adding Strategy Logic in Python

The AlgoTrader Python Interface allows you to implement strategies in Python 2 and 3.

Please refer to the Reference Guide for more details on how to setup AlgoTrader with Python, most notably the installation\(^6\) and Python strategy development\(^7\) sections.

The AlgoTrader trial version has an example Python strategy installed. You can find it by searching for ema-python-strategy.py and opening the file in your Python IDE.

The header of the file contains all necessary imports, in particular the algotrader_com interface.

```python
import logging
from decimal import Decimal
import numpy as np
```

\(^4\) https://doc.algotrader.com/html_single/index.html#EMA_Strategy
\(^6\) http://doc.algotrader.com/html/Installation.html
\(^7\) http://doc.algotrader.com/html/Strategy_Development.html#Python_Strategy_Development
from algotrader_com.domain.order import MarketOrder
from algotrader_com.interfaces.connection import connect_to_algotrader,
    wait_for_algotrader_to_disconnect
from algotrader_com.services.strategy import StrategyService
...

The next part of the file contains settings that the strategy will use.

...  
ACCOUNT_ID = 1
SECURITY_ID = 25
ORDER_QUANTITY = Decimal("10000")
DEFAULT_ADAPTER_TYPE = "IB"
EMA_PERIOD_SHORT = 10
EMA_PERIOD_LONG = 20
...

• The ACCOUNT_ID defines the id of the account, which the strategy will use for trading.

• The SECURITY_ID defines the id of the instrument, which the strategy will trade.

• The ORDER_QUANTITY is the number of contracts the strategy will trade.

• The DEFAULT_ADAPTER_TYPE indicates from where we get market data and where we want to trade (Interactive Brokers by default).

• The EMA_PERIOD_SHORT is the lookback period of the shorter EMA indicator.

• The EMA_PERIOD_LONG is the lookback period of the longer EMA indicator.

In addition, the following fields need to be defined:

• The close_price_window1 and close_price_window2, used by the exponential moving average indicators.

• The strategy_id, a strategy variable that holds the Strategy Id, fetched from the AlgoTrader server.

Next, the Python Class for the EMAStrategyService needs to be created:

...  
class EMAStrategyService(StrategyService):
    
    def __init__(self):
        StrategyService.__init__(self)

        STRATEGY_NAME = "EMA"
        previous_difference = 0
**First, the Superclass** `StrategyService` constructor is called.

**Next, the** `STRATEGY_NAME` global variable is set. That will be later passed to the Entry Point

**The** `previous_difference` variable is initialized. This will be used to hold a difference between two EMA series.

**Then, the** `first_order_sent` boolean variable is initialized with `False` value. This will be used to determine the closing quantity of orders (First order will have size `ORDER_QUANTITY`, all following orders must have size `ORDER_QUANTITY * 2` since they have to close existing position and open new in the opposite direction).

**Next, the** `position` variable is initialized. This will be used to store the current position size.

Implement the `on_init` method, which will be called when the strategy gets initialized. We have to pass `STRATEGY_NAME` to the Entry Point in this method.

```python
... def on_init(self, lifecycle_event):
    self.python_to_at_entry_point.set_strategy_name(self.STRATEGY_NAME)
...
```

Next, update the `on_start` (an AlgoTrader Live Cycle Method) method, which will be called when the strategy starts up.

```python
... def on_start(self, lifecycle_event):
    # noinspection PyBroadException
    try:
        self.python_to_at_entry_point.subscription_service.subscribe_market_data_event(
            self.STRATEGY_NAME, SECURITY_ID, DEFAULT_ADAPTER_TYPE)
    except:
        pass
...
```

For further details please visit the AlgoTrader documentation regarding *Life Cycle Events*.

The `on_start` method calls `subscribe_market_data_event` of the `subscription_service` exposed by the AlgoTrader’s Python interface by passing the `STRATEGY_NAME` and the `SECURITY_ID` of the instrument the strategy wants to receive market data for.

---

Next, update the on_bar method, which will be invoked on every incoming Bar:

```python
...  
    def on_bar(self, bar):
        close_price_window1.append(float(bar.close))
        close_price_window2.append(float(bar.close))

    ...
    if len(close_price_window1) > EMA_PERIOD_SHORT + 1:  # remove the oldest
        close_price_window1.pop(0)  # remove the oldest element from the list
    if len(close_price_window2) > EMA_PERIOD_LONG + 1:  # remove the oldest
        close_price_window2.pop(0)  # remove the oldest element from the list

    # if we have enough data already, calculate EMA averages difference, buy/sell
    # on cross
    if len(close_price_window2) >= EMA_PERIOD_LONG:
        ...
```

- The method first updates `close_price_window1` and `close_price_window2` with the latest bar received in the method's parameter.

- Then the lengths of the price windows (`close_price_window1` and `close_price_window2`) must be cut to the desired size. In our case, `close_price_window1` will have a maximum length of 10 (10 period EMA) and `close_price_window2` will have a maximum length of 20 (20 periods EMA).

- Next, we check if we already have enough data to compute the EMA, which we need to proceed with the strategy logic.

Then the actual trading rules are defined:

```python
...  
    close_price_window1.pop(0)

    ema1 = _numpy_ewma_vectorized_v2(np.array(close_price_window1),
    EMA_PERIOD_SHORT)
    ema2 = _numpy_ewma_vectorized_v2(np.array(close_price_window2),
    EMA_PERIOD_LONG)

    difference = ema1[-1] - ema2[-1]

    global strategy_id
    if strategy_id is None:
        strategy_id = self.python_to_at_entry_point.get_strategy_id()
```
account_id = ACCOUNT_ID
security_id = SECURITY_ID
if not self.first_order_sent:
    quantity = ORDER_QUANTITY
else:
    quantity = ORDER_QUANTITY * 2  # closing opposite position and opening new one

if difference > 0.0 and (self.previous_difference == 0 or self.previous_difference < 0.0):
    side = "BUY"
    market_order = MarketOrder(quantity=quantity, side=side,
                                strategy_id=strategy_id, account_id=account_id,
                                security_id=security_id)
    self.python_to_at_entry_point.order_service.send_order(market_order)
    self.position += float(market_order.quantity)
    self.first_order_sent = True
if difference < 0.0 and (self.previous_difference == 0 or self.previous_difference > 0.0):
    side = "SELL"
    market_order = MarketOrder(quantity=quantity, side=side,
                                strategy_id=strategy_id, account_id=account_id,
                                security_id=security_id)
    self.python_to_at_entry_point.order_service.send_order(market_order)
    self.position -= float(market_order.quantity)
    self.first_order_sent = True
    self.previous_difference = difference

• Calculate both EMA indicators (in the example we use a custom function for fast EMA calculation, see \_numpy\_ewma\_vectorized\_v2 for details).

• Calculate the difference indicator (ema1 - ema2).

• If the current value of the difference indicator is positive and the previous value was negative or zero a BUY order is sent. In other words, if the short EMA crossed above the long EMA a BUY order is sent.

• If the current value of the difference indicator is negative and the previous value was positive or zero a SELL order is sent. In other words, if the short EMA crossed below the long EMA a SELL order is sent.

The trading logic is depicted in the following chart:
Adding Strategy Logic in Python

The orders are sent through AlgoTrader's Python interface using the send_order method in the order_service provided by Entry Point.

```
...  
market_order = MarketOrder(quantity=quantity, side=side, strategy_id=strategy_id, 
account_id=account_id, security_id=security_id)  
self.python_to_at_entry_point.order_service.send_order(market_order)  
...  
```

The above creates a MarketOrder entity and assigns the `strategy_id`, the `account_id`, the `security_id`, the `quantity`, the order `side` (BUY or SELL). The order object is then handed over to the `order_service` which will execute the order.

For further details on how order are created and behave, please consult the AlgoTrader documentation regarding *Order Management*[^9].

In addition, the following statements were added to the bottom of the strategy.

The first is optional. If not specified, all callback methods are subscribed:

```
...  
only_subscribe_methods_list = ["onInit", "onStart", "onExit", "onBar", "onTick"]  
...  
```

The line below connects the Python interface to the AltoTrader server:

```
...  
The below tries to subscribe to market data if the AlgoTrader server is already running on strategy start-up. Otherwise, the subscriptions will be initialized during the onStart lifecycle event:

```python
# noinspection PyBroadException
try:  # try subscribing to market data, if AT is already up. otherwise data will be
    # subscribed on START lifecycle event
    _python_to_at_entry_point.subscription_service
d Subscription(EMAStrategyService.STRATEGY_NAME, SECURITY_ID,
    DEFAULT_ADAPTER_TYPE)
except:  
    pass
```

This example strategy will stop when the server process is stopped.

The EMA strategy is an example strategy based on Python code only. For details on how to build a trading strategy using Java or Esper please visit the AlgoTrader documentation regarding [Strategy Development](http://doc.algotrader.com/html/Strategy_Development.html)

---

Managing data

During live trading, all relevant information like orders, positions and transactions are stored in the MySql database.

To view database data please open DBForge for Algotrader via the desktop shortcut.

On the left-hand side of the application double click on Algotrader, then algotrader and Tables.

You now see all AlgoTrader tables listed below
To view the contents of a table (e.g. the exchange table), double-click its name and go to the Data tab.
The table `security` contains a list of all available instruments that can be traded with the system. This table can for example be used to find the `securityId` for the `BTC/USDT` pair traded on Binance.

5.1. Reference Data

Reference Data like instrument definitions, strategies etc. are also stored in MySQL (in tables like `security`, `security_family`, `strategy`, etc.)

Before an AlgoTrader based trading strategy can trade a particular instrument in needs to be defined in the database.

Note

- The AlgoTrader fee 30-day trial version is pre-configured with sample reference data for commonly used FX pairs, US & European Equities, Futures & Cryptocurrencies.
- The MySQL data can also be managed through the AlgoTrader UI, using the Reference Data Manager\(^1\).

For further details visit the AlgoTrader documentation regarding Reference Data\(^2\).

5.2. Historical Data

For Back Testing AlgoTrader can use historical data provided by .csv files. For the EMA strategy the AlgoTrader Strategy Wizard created a sample historical bar data file `/algotrader-ema/files/bardata/fx/`

\(^1\) [http://doc.algotrader.com/html/Client.html#RDM](http://doc.algotrader.com/html/Client.html#RDM)

EURUSD.csv. The file name (EURUSD) needs to match the symbol column in the database table security of the instrument the strategy is going to trade.

For further details on naming conventions and the location of historical data .csv files see the AlgoTrader documentation regarding Market Data File Format\(^3\).

As a more sophisticated alternative to providing historical data through .csv files, the Time Series database InfluxDB\(^4\) can be used for storage and retrieval of historical data. For further details on downloading, storing and using InfluxDB data for back testing please visit the AlgoTrader documentation on Historical Data\(^5\).

You can also import historical data csv files into InfluxDB through our historical data manager\(^6\) (Historical Data Import).

The InfluxDB data can also be browsed and managed in the Historical Data Manager.

\(^4\) [https://www.influxdata.com/products/influxdb-overview/](https://www.influxdata.com/products/influxdb-overview/)
Chapter 6.

Cryptocurrency Trading

The AlgoTrader 30-day trial version can also be used to trade Bitcoin and other Cryptocurrencies via the following exchange adapters:

- Binance
- Bitfinex
- Bitstamp
- Bitflyer
- BitMEX
- Coinbase Pro

This chapter describes how to setup trading with the Binance exchange.

To setup a connection to Binance the following steps have to be taken:

- Sign-up for a Binance account on Register with Binance
- Enable two factor authentication (2FA) on the account following the 2FA instructions (either SMS or Google Authenticator) on the Account Page
- On the account page generate a new Binance API key and Secret Key and use it in the settings below.
- On the trial instance, we have already run the ReferenceDataStarter for Binance. If you intend to use another exchange, you will have to run the corresponding reference data starter for it first.

First the Binance API key, then the Secret Key noted above need to be added by clicking the downward facing arrow next to the green start icon and then selecting Edit Configurations.

1 http://www.binance.com
2 http://www.bitfinex.com
3 https://www.bitstamp.net
4 https://bitflyer.com
5 https://www.bitmex.com
6 https://pro.coinbase.com/
7 https://www.binance.com/register.html
8 https://www.binance.com/de/my/security
In this example we are going to use the existing EMA strategy to trade via Binance. For this purpose, the Binance API key and Secret Key need to be added to the IntelliJ run configuration `EmbeddedStrategyStarter-ema-binance` by selecting it on the left-hand side and expanding the VM Options.

Here the Binance API key and Secret Key noted above need to be added. Then click OK.

Per default the EMA strategy trades the **EUR/USD** currency pair through Interactive Brokers. To now switch the strategy to trade through Binance we need to update the settings at the top of the `EMAService`:

- Update the `accountId` to match the Binance account in the database (should be 2029).
- Update the `securityId` (Whichever Id represents the **BTC/USDT** cryptocurrency pair on Binance in table `security` - should be 2041).
- Update the `orderQuantity` to a small enough number
- Update the `defaultAdapterType` `BNC` in order for the strategy to subscribe for market data through the Binance adapter.
- Add the exchange order flag in the `SendOrder` method to make a currency conversion (vs. opening a margin FX position)
public class EMAService extends StrategyService {

    private final long accountId = 851;
    private final long securityId = 864;
    private final BigDecimal orderQuantity = new BigDecimal("0.02");
    private final int emaPeriodShort = 10;
    private final int emaPeriodLong = 20;
    private final AdapterType defaultAdapterType = AdapterType.BNC;

    private void sendOrder(Side side) {
        MarketOrderVO order = MarketOrderVOBuilder.create()
            .setStrategyId(getStrategy().getId())
            .setAccountId(this.accountId)
            .setSecurityId(this.securityId)
            .setQuantity(this.orderQuantity)
            .setSide(side)
            .setExchangeOrder(true)
            .build();
        getOrderService().sendOrder(order);
    }

    Now the strategy can be started by selecting the EmbeddedStrategyStarter-ema-binance and click Run.
Cryptocurrency Trading with Python

The AlgoTrader 30-day trial version can also be used to trade Bitcoin and other Cryptocurrencies via the following exchange adapters:

- **Binance**
- **Bitfinex**
- **Bitstamp**
- **Bitflyer**
- **BitMEX**
- **Coinbase Pro**

This chapter describes how to setup trading with the Binance exchange.

To setup a connection to Binance the following steps have to be taken:

- Sign-up for a Binance account on [Register with Binance](https://www.binance.com/register.html)
- Enable two factor authentication (2FA) on the account following the 2FA instructions (either SMS or Google Authenticator) on the [Account Page](https://www.binance.com/de/my/security)
- On the account page generate a new Binance API key and Secret Key and use it in the settings below.
- On the trial instance, we have already run the ReferenceDataStarter for Binance. If you intend to use another exchange, you will have to run the corresponding reference data starter for it first.

First the Binance API key, then the Secret Key noted above need to be added by clicking the downward facing arrow next to the green start icon and then selecting **Edit Configurations**.

---

1. [http://www.binance.com](http://www.binance.com)
2. [http://www.bitfinex.com](http://www.bitfinex.com)
3. [https://www.bitstamp.net](https://www.bitstamp.net)
4. [https://bitflyer.com](https://bitflyer.com)
5. [https://www.bitmex.com](https://www.bitmex.com)
6. [https://pro.coinbase.com/](https://pro.coinbase.com/)
7. [https://www.binance.com/register.html](https://www.binance.com/register.html)
8. [https://www.binance.com/de/my/security](https://www.binance.com/de/my/security)
In this example we are going to use the existing EMA strategy to trade via Binance. For this purpose, the Binance API key and Secret Key need to be added to the launch configuration `EmbeddedStrategyStarter-ema-python-binance` by selecting it on the left-hand side and expanding the `VM Options Tab`.

Here the Binance API key and Secret Key noted above need to be added. Then click `Apply`.

Per default the EMA strategy trades the `EUR/USD` currency pair through Interactive Brokers. To now switch the strategy to trade through Binance we need to update the settings at the top of the `ema-python-strategy.py`. Open your Python IDE and edit following:

- **Update the** `ACCOUNT_ID` **to match the Binance account in the database (should be 2029).**
- **Update the** `SECURITY_ID` **(SECURITY_ID 2041 represents the BTC/USDT cryptocurrency pair for Binance in table security).**
• Update the ORDER_QUANTITY to a small enough number

• Update the DEFAULT_ADAPTER_TYPE BNC in order for the strategy to subscribe for market data through the Binance adapter

• Set the exchange order flag in the order you create to make a currency conversion (vs. opening a margin FX position)

```python
ACCOUNT_ID = 204
SECURITY_ID = 898
ORDER_QUANTITY = Decimal("10000")
DEFAULT_ADAPTER_TYPE = "BNC"
EMA_PERIOD_SHORT = 10
EMA_PERIOD_LONG = 20
```

Now the strategy can be started by selecting the EmbeddedStrategyStarter-ema-python-binance and click Run.

You can now launch `ema-python-strategy.py` in your Python IDE or from command line.
Shutting down the AWS System

If you are running the trial on AWS, you can shut down the system by clicking on the Start Menu in the lower left-hand corner of the Windows Desktop and then select Power Options in the upper right-hand corner:

Alternatively, the system can be shutdown via the Amazon AWS Console [https://console.aws.amazon.com/console/home] by first selecting the Amazon Instance and the under Actions select Instance State and then either Stop or Terminate.

Note

- If Stop is clicked the instance can be restarted at a later point in time. In the stopped state the Amazon Instance will still incur disc space using costs as mentioned in: [https://aws.amazon.com/ec2/pricing/]

- If Terminate is clicked the instance cannot be restarted. In the terminated state, no further Amazon instance costs apply