OS.ENGINE

MANUAL

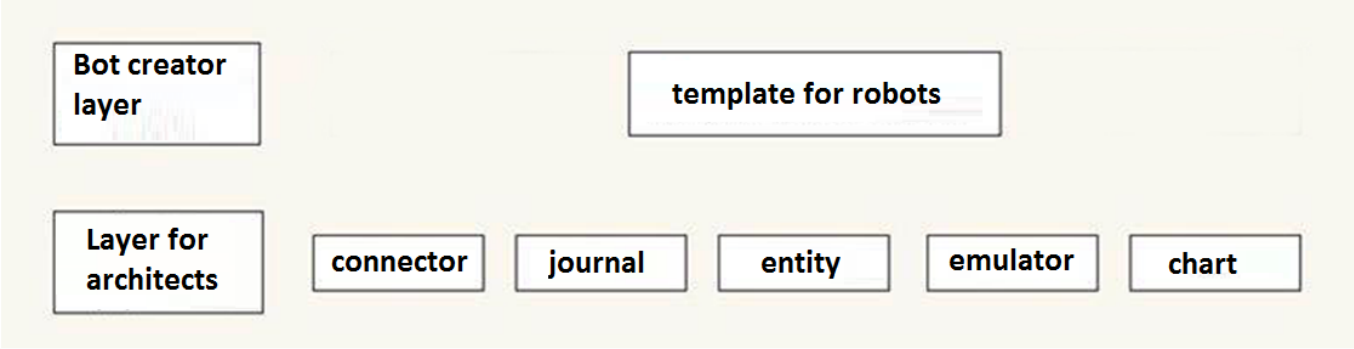
(BOT CREATE)

O-S-A.NET

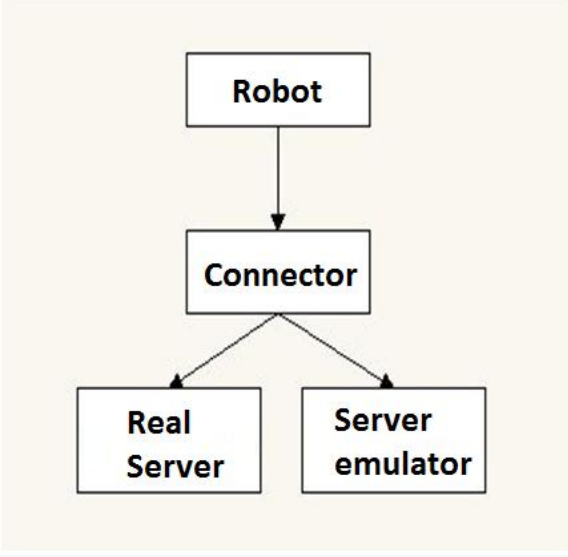
Table of content

1. Ideology

- Os.Engine is a library for creating trading robots.
- The library is built in such a way that it would be possible to divide the entry-level for algorithmic traders and hardcore programmers. The low entry threshold is achieved by separating the layers of the internal architecture. Instead of two years (stockSharp, architect level) we have two months (like WealthLab, a lower-than-junior level).
- In this manual, we consider the zero level. I.e. it will be unnecessary to dig into the entrails of the library to make a simple robot.



- A few words about the architecture:
    1. Tester

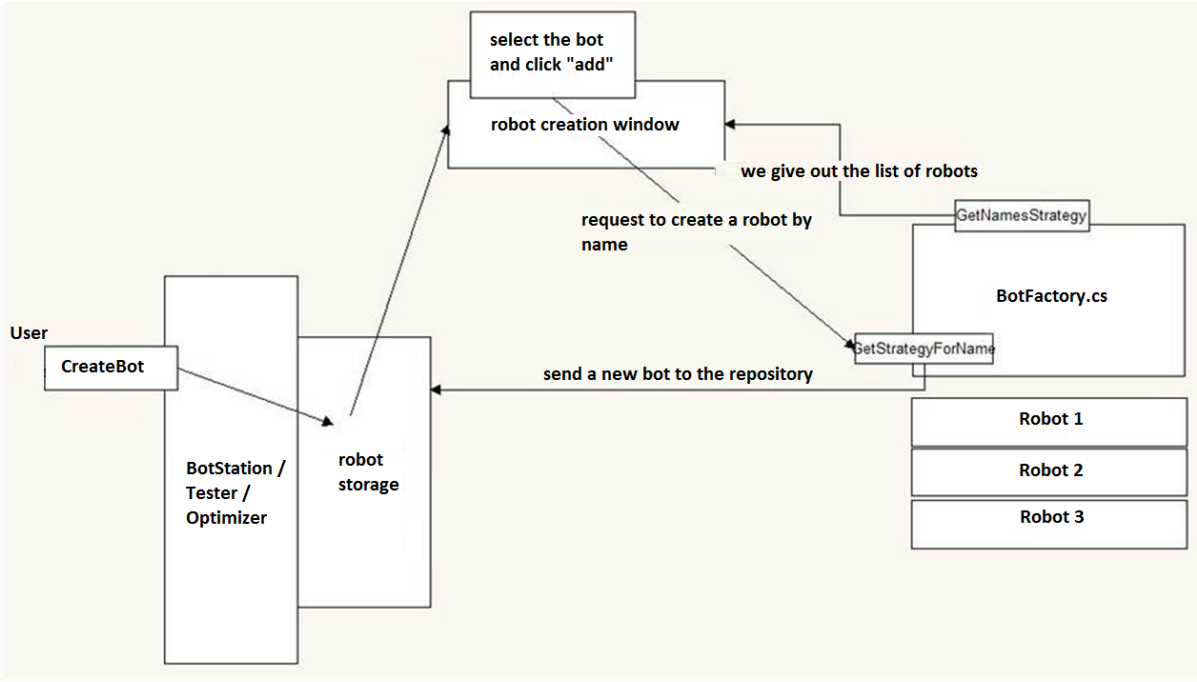


The robot never knows whether it trades on a real server or on a training server. Architecture does not allow writing a robot without a tester.

2. Bot creation architecture

The implementation of creating a bot is not quite obvious and is connected to inheritance, so this is almost the only thing you need to understand before writing code.

The illustration of the formula of creating a bot:



Flow chart (the logic of work when the user works with the terminal):

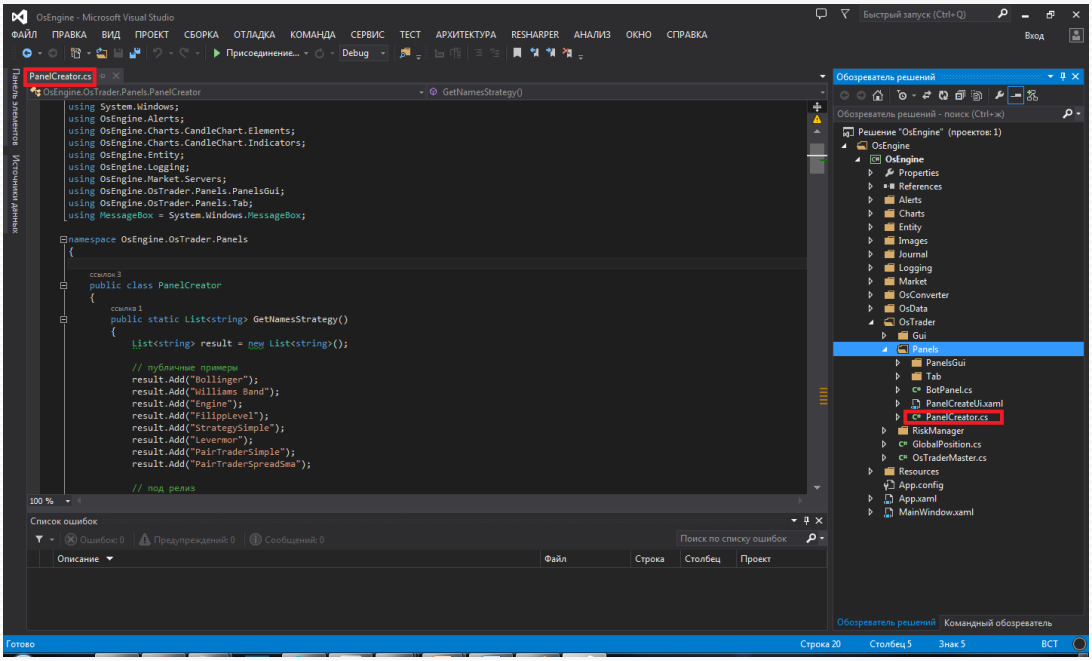
1. From the user interface, there is a signal that the user wants to create a bot (panel).
2. Bots repository creates a menu for bots creation.
3. In the process of creation, the menu requests from the PanelCreator class the names of all strategies created in the system.
4. The user selects the type of a bot and clicks "Create", at this time we refer to the PanelCreator class and ask it to create a bot with the same name.
5. The robot goes to the bot repository and appears in the user interface.

If this is not quite clear, there is no need to worry. In practice, it is much easier, since the whole logic of creating a bot (panel) is in one place. There are also simple examples to study.

3. Creating a trading robot

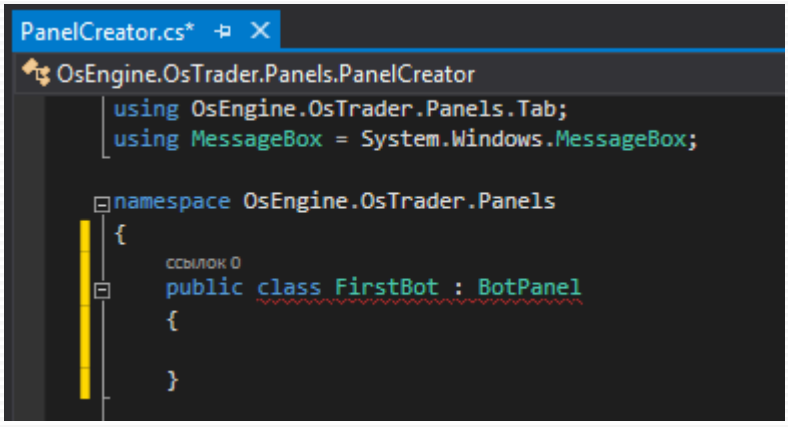
Open the library and go to Os.Trader/Panels/PanelCreator

Ignore everything else, we don't need it. This is for hardcore programmers and platform architects. **PanelCreator** is for creation of bots. Open the file.



In order to create your own bot you need:

1. To create a public class and designate it as the heir of the **BotPanel** class.



3. Creating a trading robot

2. In the generated class, it is required to create two methods **GetNameStrategyType** and **ShowIndividualSettingsDialog** :

```
ссылок 0
public class FirstBot : BotPanel
{
    ссылок 43
    public override string GetNameStrategyType()
    {
        return "FirstBot";
    }

    ссылок 43
    public override void ShowIndividualSettingsDialog()
    {
        MessageBox.Show("У данной стратегии пока нет настроек");
    }
}
```

- 3. Then we need to find the **PanelCreator** class, and the **GetNamesStrategy** method in it and add information about your class. After adding it here, the library will be able to refer to the name of your robot**:**

PanelCreator.cs*

OsEngine.OsTrader.Panels.PanelCreator

```
ссылок 3
public class PanelCreator
{

    ссылка 1
    public static List<string> GetNamesStrategy()
    {
        List<string> result = new List<string>();

        result.Add("FirstBot");

        // публичные примеры
        result.Add("Bollinger");
        result.Add("Williams Band");
```

add line

3. Creating a trading robot

4. In the same class of **PanelCreator** find **GetStrategyForName** method and add a mechanism for creating a bot:

```
ссылок 2
public static BotPanel GetStrategyForName(string nameClass, string name)
{

    BotPanel bot = null;
// примеры и бесплатные боты

    if (nameClass == "FirstBot")
    {
        bot = new FirstBot(name);
    }

    if (nameClass == "Williams Band")
    {
        bot = new StrategyBillWilliams(name);
    }
```

add construction

5. Then we are back to the created class of **FirstBot** and add the constructor to it, in the future we will be able to subscribe to a completion event of candles or ticks, and begin to gather the logic of the bot

```
namespace OsEngine.OsTrader.Panels
{
    ссылок 2
    public class FirstBot : BotPanel
    {
        ссылок 43
        public override string GetNameStrategyType()
        {
            return "FirstBot";
        }

        ссылок 43
        public override void ShowIndividualSettingsDialog()
        {
            MessageBox.Show("У данной стратегии пока нет настроек");
        }

        ссылка 1
        public FirstBot(string name) : base(name)
        {

        }
    }
```

create constructor

3. Creating a trading robot

5. Now we need to create a tab for this bot **BotTabSimple**

```
namespace OsEngine.OsTrader.Panels
{
    ссылок 2
    public class FirstBot : BotPanel
    {
        ссылок 43
        public override string GetNameStrategyType()
        {
            return "FirstBot";
        }

        ссылок 43
        public override void ShowIndividualSettingsDialog()
        {
            MessageBox.Show("У данной стратегии пока нет настроек");
        }

        ссылка 1
        public FirstBot(string name) : base(name)
        {
            TabCreate(BotTabType.Simple);
            _tab = TabsSimple[0];

        }

        private BotTabSimple _tab;
    }
```

Create realization

Create tab to trade

IMPORTANT

- **BotTabSimple** is a basic implementation of tabs, or, in other words, the standard for writing many of the basic strategies, most often this is exactly what you need. BotTabSimple can have only one source (instrument), with which we can perform trade operations.
- There are also other implementations such as **BotTabIndex**
- **BotTabIndex** is another form of implementation of a tab. This implementation allows loading multiple sources (tools) and building an index (synthetics)
- One panel (bot) can have several tabs. For example: 3 BotTabSimple and 1 BotTabIndex

After all these operations, we can compile the program, and as a result, we got an empty robot template. But this is not a complete bot.

3. Creating a trading robot

6. Now we have to subscribe to a completion event of candles **CandleFinishedEvent** and create **TradeLogic** method for processing logic:

```
ссылок 2
public class FirstBot : BotPanel
{
    ссылок 43
    public override string GetNameStrategyType()
    {
        return "FirstBot";
    }

    ссылок 43
    public override void ShowIndividualSettingsDialog()
    {
        MessageBox.Show("У данной стратегии пока нет настроек");
    }

    ссылка 1
    public FirstBot(string name) : base(name)
    {
        TabCreate(BotTabType.Simple);
        _tab = TabsSimple[0];

        _tab.CandleFinishedEvent += TradeLogic;

    }

    private BotTabSimple _tab;

    ссылка 1
    private void TradeLogic(List<Candle> candles)
    {

    }
}
```

subscribe to the candle completion event

Now we can write trade logic in our **TradeLogic** method**.** In this method, we pass candles of the tool with which we will process our logic.

Here is a simple scope of work for our example:

1. if the candles are less than 5 – we exit the method.
2. if there are open positions – close and exit.
3. if there are no positions – we select the direction to go. If the close-out of the last candle is higher than the close-out of the previous one – buy.
4. if the close-out of the last candle is lower than the close-out of the previous one – sell.

3. Creating a trading robot

We receive:

```
ссылка 1
private void TradeLogic(List<Candle> candles)
{
    // 1. если свечей меньше чем 5 - выходим из метода.
    if (candles.Count < 5)
    {
        return;
    }

    // 2. если уже есть открытые позиции – закрываем и выходим.
    if (_tab.PositionsOpenAll != null && _tab.PositionsOpenAll.Count != 0)
    {
        _tab.CloseAllAtMarket();
        return;
    }

    // 3. если закрытие последней свечи выше закрытия предыдущей – покупаем.
    if (candles[candles.Count - 1].Close > candles[candles.Count - 2].Close)
    {
        _tab.BuyAtMarket(1);
    }

    // 4. если закрытие последней свечи ниже закрытия предыдущей, продаем.
    if (candles[candles.Count - 1].Close < candles[candles.Count - 2].Close)
    {
        _tab.SellAtMarket(1);
    }
}
```

This is it, our simple bot is done. If we run it it will work

The current example reveals only the basic designs that are required to write any robot.

More complex examples can be considered on their own in the examples available with the library.

4. Opportunities for trading

Consider what standard range of opportunities we have to work with the market:

Subscription to events:

- BestBidAskChangeEvent - changed the best bid/ask.
- CandleFinishedEvent - the last candle ended.
- CandleUpdateEvent - the last candle updated.
- FirstTickToDayEvent - the morning session started. The first ticks appear
- LogMessageEvent - outgoing messages for log.
- MarketDepthUpdateEvent – come
- NewTickEvent – new ticks appear.
- OrderUpdateEvent – an order was updated in the system.
- PositionClosingFailEvent – the position was not closed.
- PositionClosingSuccesEvent – a position successfully closed.
- PositionOpeningFailEvent – the position was not opened.
- PositionOpeningSuccesEvent – a position successfully opened.
- ServerTimeChangeEvent – server time was changed.

Methods of trade:

• BuyAtAceberg	SellAtAceberg	- buying, selling at iceberg
• BuyAtAcebergToPosition	SellAtAcebergToPosition	- add a new iceberg order to the position
• BuyAtLimit	SellAtLimit	- buying, selling at a certain price
• BuyAtLimitToPosition	SellAtLimitToPosition	- add a new limit order to the position
• BuyAtMarket	SellAtMarket	- buy, sell at any price
• BuyAtMarketToPosition	SellAtMarketToPosition	- add a new market order to the position
• BuyAtStop	SellAtStop	- buying, selling at a price breaching
• BuyAtStopCancel	SellAtStopCancel	- withdraw all orders at a price breaching

- CloseAllAtMarket - close all positions by market
- CloseAllOrderInSystem - withdraw all orders opened by the robot from the system
- CloseAllOrderToPosition - withdraw all orders from the system related to this transaction
- CloseAtAceberg - close position at iceberg at a certain price
- CloseAtLimit - close a position at a certain price
- CloseAtMarket - close a position at a current price
- CloseAtProfit - place a profit order for a position
- CloseAtStop - place a stop order for a position
- CloseAtTrailingStop - set trailing stop order for a position
- CloseOrder - withdraw the order

4. Opportunities for trading

Available data:

- PositionAll - all positions belonging to the bot
- PositionOpenLong - take all long positions
- PositionOpenShort - take all short positions
- PositionsLast - last open position
- PositionsCloseAll - all closed positions of the bot
- PositionsOpenAll - all open positions of the bot

- PriceBestAsk - best purchase price of the tool
- PriceBestBid - best selling price of the tool

- PriceCenterMarketDepth - the price of the center of the glass
- Securiti - trading tool
- ServerStatus - status of the server
- Trades - all ticks on the instrument
- MarketDepth - glass data