Продвинутое использование торговой платформы MetaTrader 5 Создание индикаторов и торговых роботов на MQL5 и Python

Издание 3-е исправленное и дополненное

Разработка индикаторов и советников с использованием языков MQL5 и Python для платформы MetaTrader 5

ТИМУР МАШНИН

Тимур Машнин

Продвинутое использование торговой платформы MetaTrader 5. Создание индикаторов и торговых роботов на MQL5 и Python. Издание 3-е, исправленное и дополненное

«Автор»

2022

Машнин Т.

Продвинутое использование торговой платформы MetaTrader 5. Создание индикаторов и торговых роботов на MQL5 и Python. Издание 3-е, исправленное и дополненное / Т. Машнин — «Автор», 2022

Эта книга познакомит с практическим использованием языка MetaQuotes Language 5 (MQL5) программирования технических индикаторов, торговых роботов и вспомогательных приложений для автоматизации торговли на финансовых рынках с помощью торговой платформы MetaTrader 5.Вы научитесь создавать MQL5 приложения, используя как процедурное программирование, так и объектно-ориентированное программирование.Познакомитесь с общей структурой и свойствами технических индикаторов и советников, научитесь использовать функции обратного вызова MQL5 для создания пользовательских индикаторов и советников, реализующих автоматическую торговую систему.Познакомитесь с генетическими алгоритмами для создания самооптимизирующегося советника.Узнаете как создать нейронную сеть для предсказания цен на рынке и разработать советник с использование машинного обучения на языке Python для алгоритмической торговли.

Содержание

Тимур Машнин
Продвинутое использование торговой платформы MetaTrader 5. Создание индикаторов и торговых роботов на MQL5 и Python. Издание 3-е, исправленное и дополненное

Исходный код

Исходный код к этой книге можно посмотреть и скачать по адресу https://github.com/novts/MetaTrader-5-Creating-Trading-Robots-and-Indicators-with-MQL5

Введение

Надеюсь, вы все уже прочитали справочник MQL5 на сайте https://www.mql5.com/ru/docs.

Здесь мы не будем пересказывать этот документ, а сосредоточимся на его практическом использовании. Мы будем лишь позволять себе изредка только его цитирование.

Как сказано в предисловии к справочнику:

Для выполнения конкретных задач по автоматизации торговых операций MQL5-программы разделены на четыре специализированных типа.

И далее идет перечисление: Советник, Пользовательский индикатор, Скрипт, Библиотека и Включаемый файл.

Скрипты используются для выполнения одноразовых действий, обрабатывая только событие своего запуска, и поэтому не будут нам здесь интересны.

Также нам не будут интересны библиотеки, так как использование включаемых файлов более предпочтительно для уменьшения накладных расходов.

Поэтому мы сосредоточимся на создании советников и индикаторов с использованием включаемых файлов. Такова наша цель применения языка программирования MQL5, синтаксис которого, конечно, интересен, но будет нам только в помощь.

На самом деле программирование на языке MQL5 представляет собой яркий пример событийно-ориентированного программирования, так как весь код MQL5-приложения построен на переопределении функций обратного вызова – обработчиков событий клиентского терминала и пользователя.

А уже в коде функций обратного вызова можно использовать либо процедурное программирование, либо объектно-ориентированное программирование. Здесь мы рассмотрим оба этих подхода.

Начало работы

Для начала работы выберем какого-нибудь посредника, чтобы подключиться к его серверу и получать реальные котировки рынка для разработки и тестирования наших MQL5 приложений.

Под посредником мы имеем в виду торгового представителя, юридическое лицо, профессионального участника рынка, имеющего право совершать операции на рынке по поручению клиента и за его счёт или от своего имени и за счёт клиента на основании возмездных договоров с клиентом.

Теперь, что такое рынок?

Существуют разные типы рынков.

Это валютный рынок, это фондовый рынок или рынок ценных бумаг, это товарный рынок, и это рынок фьючерсов и опционов.

Мы с вами сосредоточимся на валютном рынке или рынке форекс.

Что такое рынок форекс?

FOREX – это сокращение от двух слов Foreign Exchange, что означает Валютный Обмен.

В отличие от других рынков, где торговля происходит на биржах, рынок форекс – это внебиржевой рынок межбанковского обмена валюты без какой-либо централизованной площадки.

Участники рынка форекс – это центральные банки, коммерческие банки, инвестиционные банки, брокеры и дилеры, пенсионные фонды, страховые компании, транснациональные корпорации и т. д.

Реально, большая часть сделок по обмену одних валют на другие происходит на ВНЕБИРЖЕВОМ рынке между крупными международными банками с использованием межбанковского информационно-торгового терминала.

И торговля идет на очень большие суммы. Минимальным лотом является сумма в 1 миллион долларов или евро, стандартным – 5 или 10 миллионов долларов.

Такая торговля валютами обеспечивает в первую очередь экспортно-импортные операции клиентов банков, и во вторую, интересы собственных торгово-инвестиционных отделов международных банков.

И совершают банки сделки как на межбанковском внебиржевом рынке, так и на валютных биржах.

Откуда берутся котировки на рынке Форекс?

Если взять, например, фондовый рынок, то там есть специальное учреждение – биржа, где торгуются определённые ценные бумаги (только там и нигде больше), и эта самая биржа и выступает единым центром распространения котировок остальным участникам, в том числе дилинговым центрам.

В случае с Форексом такого центра не существует, рынок не имеет единого места торговли и объединяет всех участников посредством современных средств передачи данных.

Поскольку основной объем торговых операций осуществляется через банковские учреждения, рынок Форекс называют международным межбанковским рынком.

Все крупнейшие участники данного рынка, международные банки, осуществляют котирование и выступают своего рода «двигателями рынка», совершая сделки либо с другими банками, либо с клиентами – инвестиционными фондами, компаниями, физическими лицами.

Все остальные участники рынка Forex запрашивают у них котировки и проводят по ним свои операции.

Выставление котировок по валютным парам международные банки производят, как правило, в электронном режиме.

И котировки формируются как на основе запросов других участников, так и в потоковом режиме (индикативном), когда банк выставляет «справочный» курс, по которому он готов совершить сделку, однако не обязан будет это делать.

Окончательная цена сделки зависит от суммы сделки, статуса участника, текущего положения на рынке и других факторов.

Индикативные и реальные котировки поступают в глобальные информационные системы (Reuters, Bloomberg, Dow Jones и др.), откуда их получают другие пользователи, в том числе и дилинговые центры.

Именно котировки, полученные от обслуживающего дилингового центра, видит трейдер в своем торговом терминале, который он использует в процессе торговли.

Таким образом, если сравнивать Форекс с биржевым рынком, то здесь отсутствует цена, единая для всех без исключения участников.

Зачастую операции совершаются по разным котировкам, причем цена будет более выгодной для второстепенных участников, имеющих налаженные контакты с основными участниками – банками, а также участников, торгующих большими объемами валюты.

В то же время, благодаря высокой ликвидности рынка котировки в большинстве случаев различаются только на 1-2 пункта, что делает практически невозможным пространственный арбитраж, когда участник покупает валюту у одного продавца по какой-либо цене, зная, что он сможет в тот же момент продать её другому покупателю на более выгодных условиях.

Теперь, что такое дилинговый центр форекс?

Дилинговый центр – это небанковская организация, обеспечивающая возможность клиентам с небольшими суммами торгового капитала на условиях маржинальной торговли заключать спекулятивные сделки.

И естественно, перед передачей котировок своим клиентам, дилинговый центр накладывает на них собственный фильтр, включающий, помимо прочего, спред, который будет составлять его заработок.

Теперь, таким образом, дилинговый центр обеспечивает возможность клиентам с небольшими суммами торгового капитала на условиях маржинальной торговли заключать спекулятивные сделки.

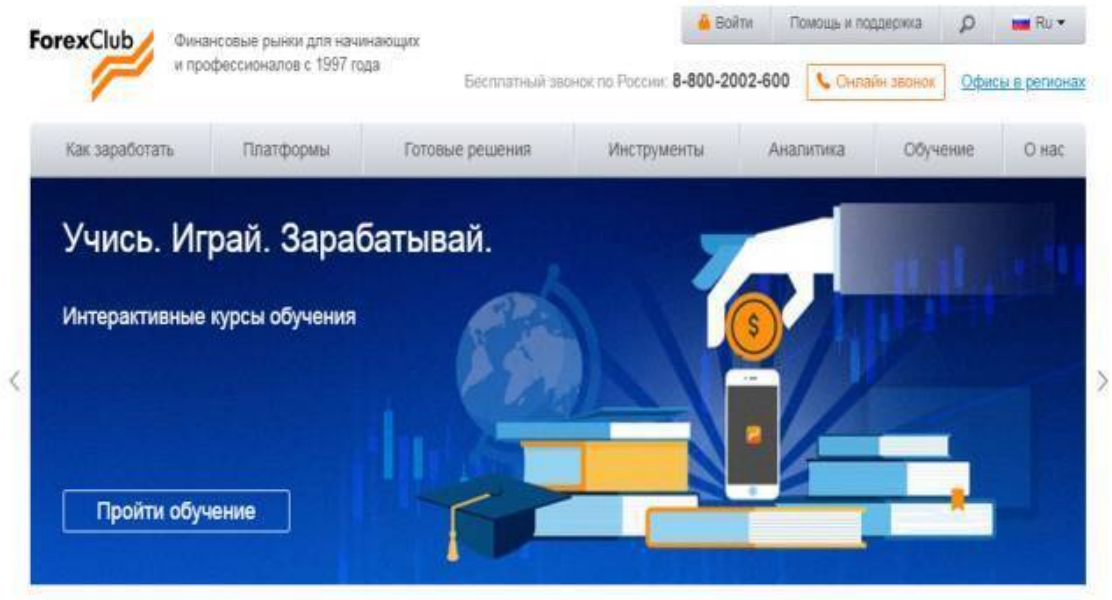
Как объясняют сами дилинговые центры, они отправляют на реальный внебиржевой рынок не все клиентские ордера, а только их агрегированную составляющую, превышающую определенный размер. А остальные ордера дилинговый центр сводит с противоположными ордерами, полученными от других клиентов.

На самом деле, как правило, ни один дилинговый центр практически никогда не выводит «сделки» своих клиентов на открытый рынок, потому как знает, что условия игры таковы, что клиент рано или поздно проиграет. Следовательно, выводить сделки на рынок нет никакой надобности.

Таким образом клиент или трейдер торгует не против рынка, а против дилингового центра.

В начале мы сказали, что нам нужен какой-нибудь посредник, чтобы подключиться к его серверу и получать реальные котировки рынка для разработки и тестирования наших MQL5 приложений.

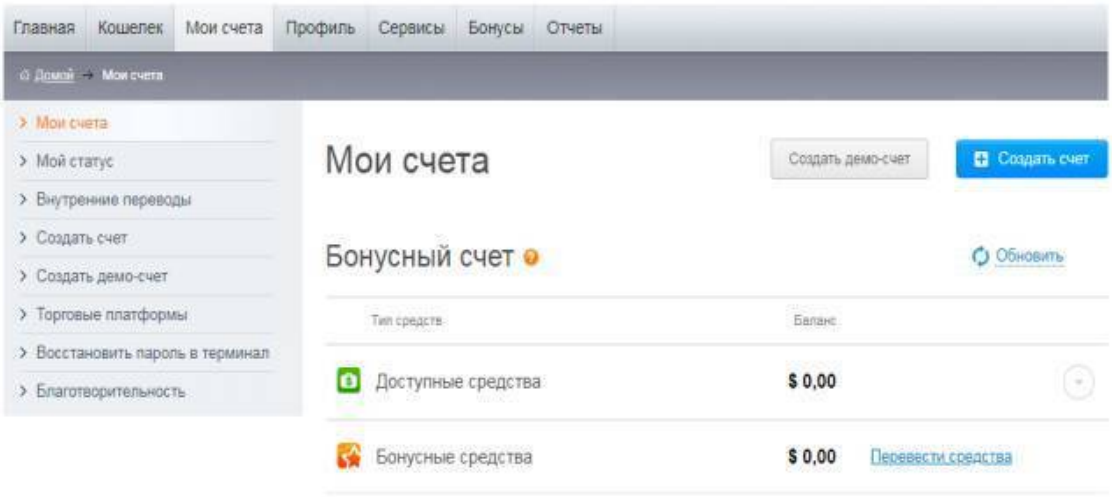
Так как у нас нет миллионов долларов, чтобы непосредственно торговать на Форексе, и мы не можем себе позволить установить свой межбанковский информационно-торговый терминал, в качестве посредника выберем дилинговый центр.



Давайте выберем, например, дилинговый центр Forex Club.

Я не являюсь фанатом данной компании, это просто для нашего кодирования.

Для реальной торговли лучше выбрать, наверное, какой-нибудь банк.



Зарегистрируемся и создадим демо-счет для платформы MetaTrader 5.

Forex Club предлагает два типа счетов:

Немедленное исполнение (Instant Execution)

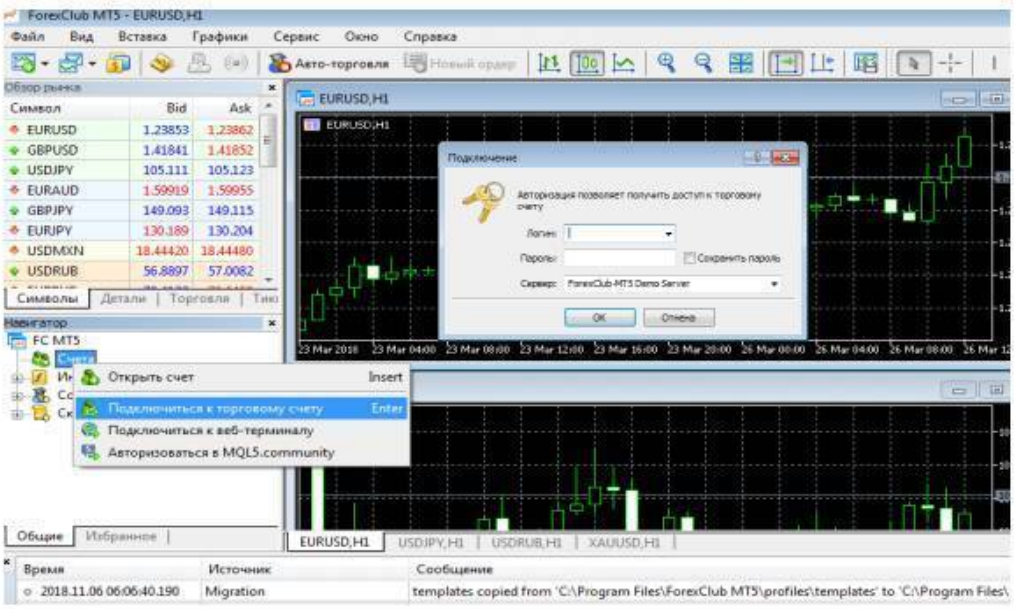
В этом режиме исполнение рыночного ордера осуществляется по предложенной цене. При отправке запроса на исполнение, платформа автоматически подставляет в ордер текущие цены.

И исполнение по рынку (Market Execution)

В этом режиме исполнения рыночного ордера решение о цене исполнения принимает дилинговый центр без дополнительного согласования с трейдером.

Мы откроем счет – немедленное исполнение (Instant Execution).

Далее скачаем и установим платформу MetaTrader 5.

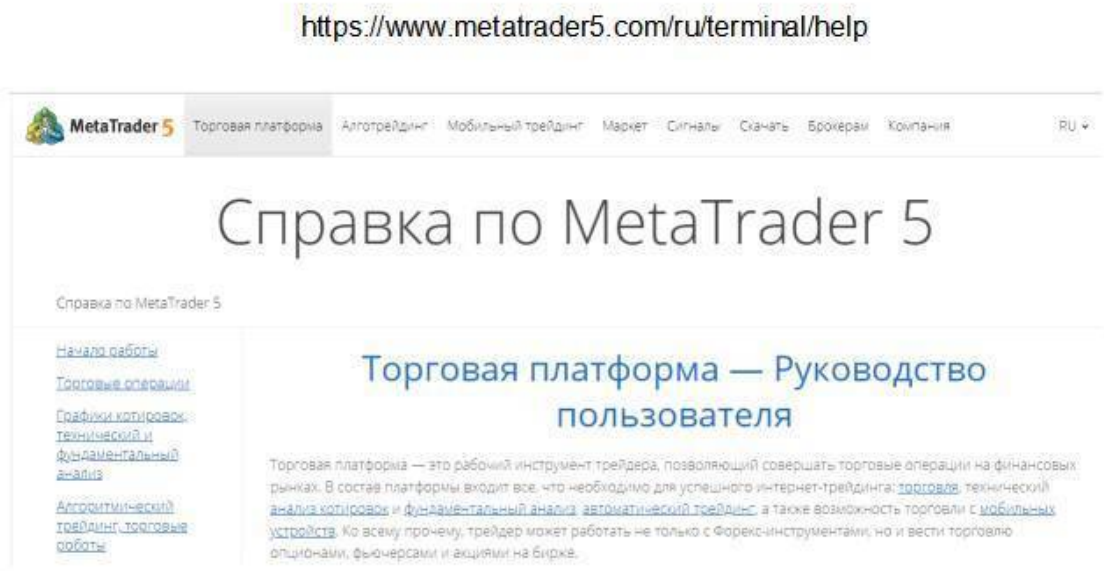


И подключимся к серверу Forex Club, используя логин и пароль демо счета.

Далее, нажав правой кнопкой мышки на графике и зайдя в свойства, настроим внешний вид графика, как вам нравится.

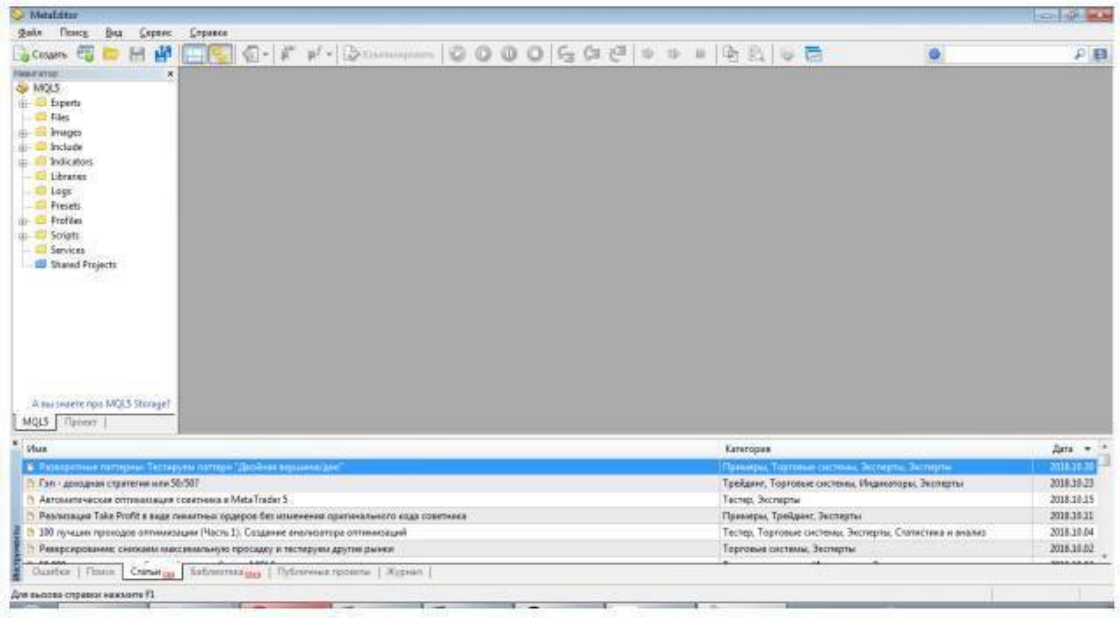
Мультирыночная платформа MetaTrader 5 позволяет совершать торговые операции на Forex, фондовых биржах и фьючерсами.

С помощью MetaTrader 5 можно также проводить технический анализ котировок, работать с торговыми роботами и копировать сделки других трейдеров.



Более подробно про платформу MetaTrader 5 и про ее интерфейс можно почитать в соответствующей справке.

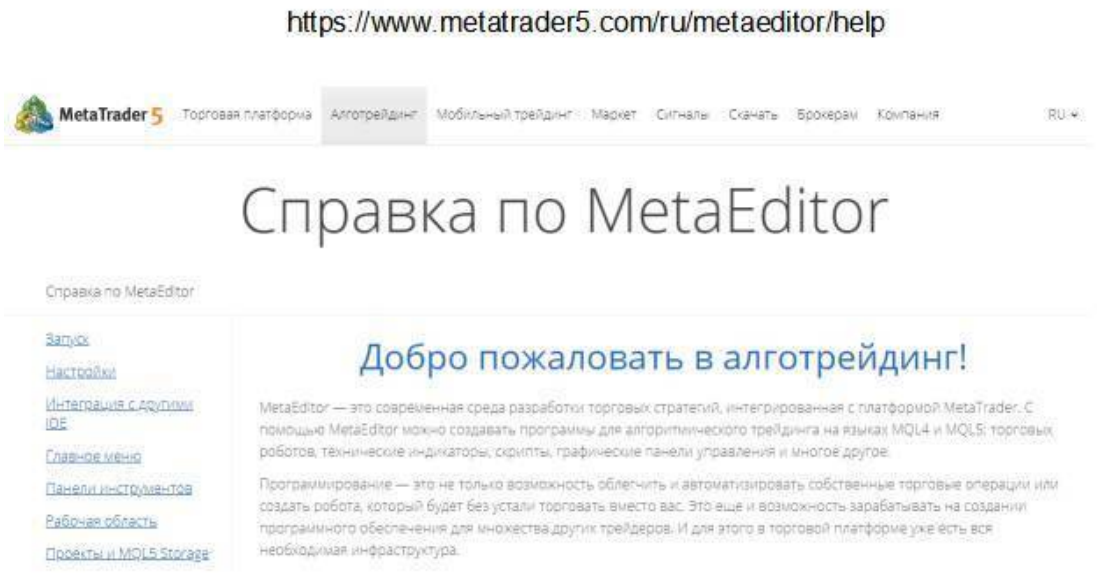
Мы не будем пересказывать эту справку, так как это было бы слишком нагло брать деньги за книгу, в которой пересказывается общедоступная справка.



Также помимо терминала MetaTrader 5, нас интересует редактор MQL5, который можно открыть либо с помощью ярлыка, либо в меню Сервис терминала MetaTrader 5.

MetaEditor – это современная среда разработки торговых стратегий, интегрированная с платформой MetaTrader.

С помощью MetaEditor можно создавать торговых роботов, технические индикаторы, скрипты, графические панели управления и многое другое.

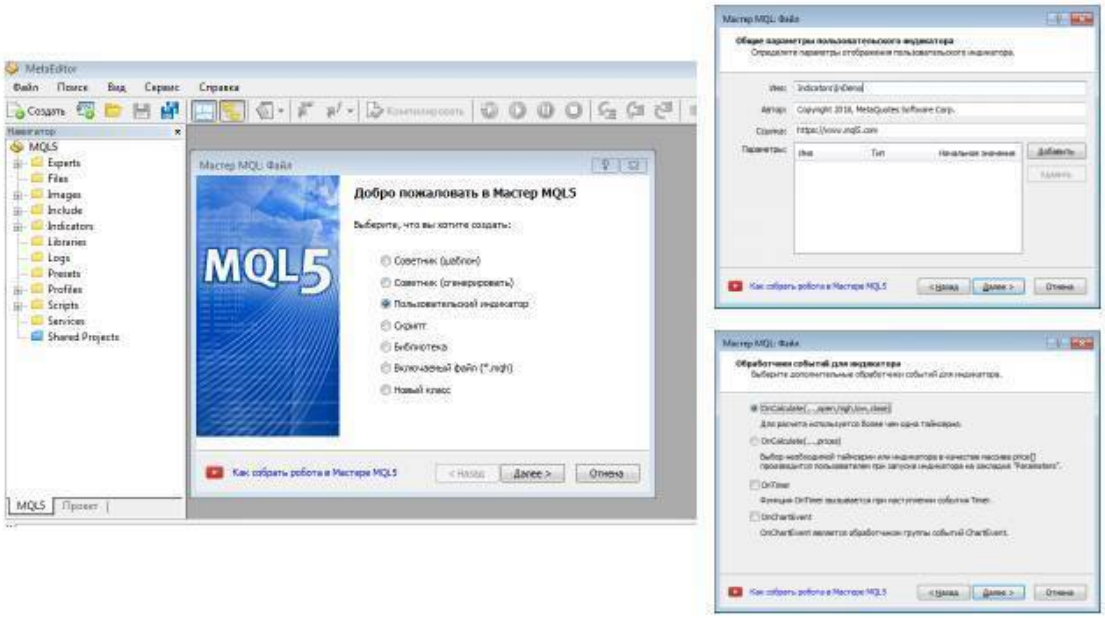


Для редактора MetaEditor также есть подробная справка, которую мы также не будем пересказывать.

Мы лучше сразу займемся практическим кодированием.

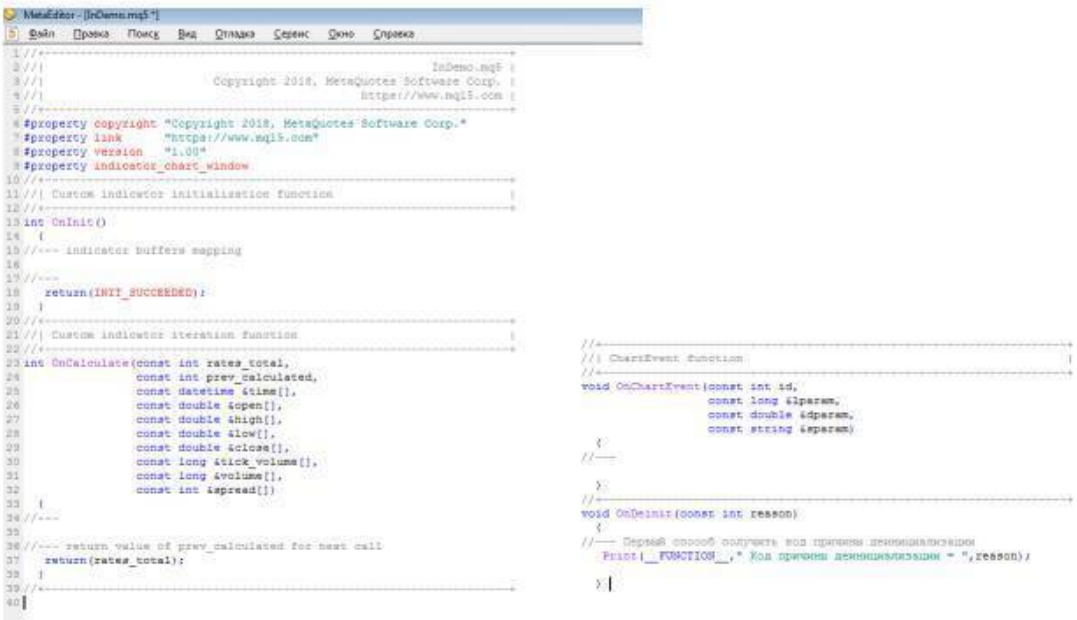
Общая структура индикатора

Для создания основы пользовательского индикатора используем редактор MetaEditor.



Нажмем кнопку меню Создать и в окне мастера выберем Пользовательский индикатор.

Нажмем Далее, введем имя создаваемого индикатора, нажмем Далее и отметим функции, которые мастер должен сгенерировать и в следующем окне нажмем Готово.



В результате будет создан код основы индикатора.

Код индикатора начинается с блока объявления свойств индикатора и различных объектов, используемых индикатором, таких как массивы буферов индикатора, параметры ввода, глобальные переменные, хэндлы используемых технических индикаторов, константы.

Данный блок кода выполняется приложением Торговая Платформа MetaTrader 5 сразу при присоединении индикатора к графику символа.

После блока объявления свойств индикатора, его параметров и переменных, идет описание функций обратного вызова, которые терминал вызывает при наступлении таких событий, как инициализация индикатора после его загрузки, перед деинициализацией индикатора, при изменении ценовых данных, при изменении графика символа пользователем.

Для обработки вышеуказанных событий необходимо описать такие функции как OnInit(), OnDeinit(), OnCalculate() и OnChartEvent().

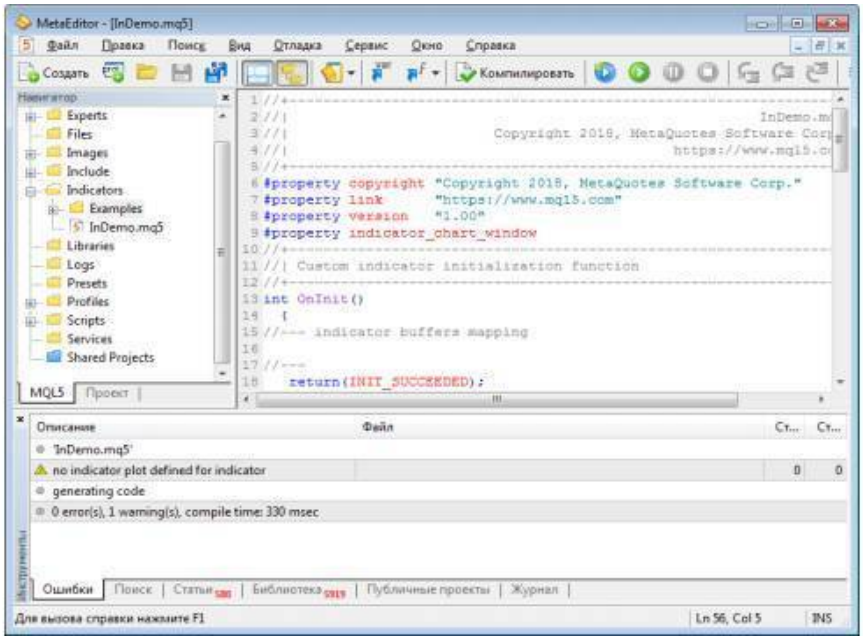
В функции OnInit() индикатора, как правило, объявленные в начальном блоке массивы связываются с буферами индикатора, определяя его выводимые значения, задаются цвета индикатора, точность отображения значений индикатора, его подписи и другие параметры отображения индикатора. Кроме того, в функции OnInit() индикатора могут получаться хэндлы используемых технических индикаторов и рассчитываться другие используемые переменные.

В функции OnDeinit() индикатора, как правило, с графика символа удаляются графические объекты индикатора, а также удаляются хэндлы используемых технических индикаторов.

В функции OnCalculate() собственно и производится расчет значений индикатора, заполняя ими объявленные в начальном блоке массивы, которые в функции OnInit() индикатора были связаны с буферами индикатора, данные из которых берутся терминалом для отрисовки индикатора. Кроме того, в функции OnCalculate() могут изменяться цвета индикатора и другие параметры его отображения.

В функции OnChartEvent() могут обрабатываться события, генерируемые другими индикаторами на графике, а также удаление пользователем графического объекта индикатора и другие события, возникающие при работе пользователя с графиком.

На этом код индикатора заканчивается, хотя там могут быть также определены пользовательские функции, которые вызываются из функций обратного вызова OnInit(), OnDeinit(), OnCalculate() и OnChartEvent().

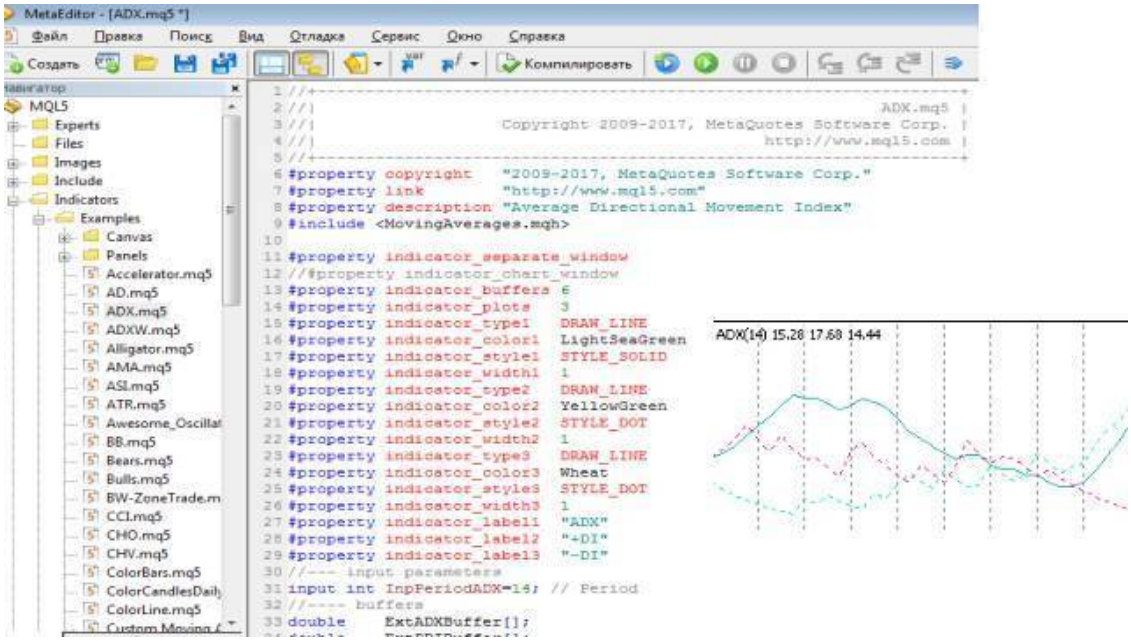


Для компиляции нашего индикатора нажмем кнопку Компилировать редактора, при этом в нижнем окне отобразится результат компиляции.

После компиляции наш индикатор автоматически появится в торговом терминале, и мы сможем присоединить его к графику финансового инструмента.

Свойства индикатора

Давайте более подробно рассмотрим свойства индикатора.



Цитата из справочника:

Свойства программ (#property). У каждой mql5-программы можно указать дополнительные специфические параметры #property, которые помогают клиентскому терминалу правильно обслуживать программы без необходимости их явного запуска. В первую очередь это касается внешних настроек индикаторов. Свойства, описанные во включаемых файлах, полностью игнорируются. Свойства необходимо задавать в главном mq5-файле: #property идентификатор значение.

Включаемый файл указывается с помощью ключевого слова #include, после которого следует путь к включаемому файлу.

Включаемый файл – это часто используемый блок кода. Такие файлы могут включаться в исходные тексты экспертов, скриптов, пользовательских индикаторов и библиотек на этапе компиляции. Использование включаемых файлов более предпочтительно, чем использование библиотек, из-за дополнительных накладных расходов при вызове библиотечных функций.

Включаемые файлы могут находиться в той же директории, что и исходный файл, в этом случае используется директива #include с двойными кавычками. Другое место хранения включаемых файлов – в директории <каталог_терминала>\MQL5\Include, в этом случае используется директива #include с угловыми скобками.

В качестве первого свойства индикатора, как правило, указывается имя разработчика, например:

#property copyright

Далее указывается ссылка на сайт разработчика:

#property link

После этого идет описание индикатора, каждая строка которого обозначается с помощью идентификатора description, например:

#property description "Average Directional Movement Index"

Далее указывается версия индикатора:

#property version "1.00"

На этом, как правило, объявление общих свойств индикатора заканчивается.

Индикатор может появляться в окне терминала двумя способами – на графике символа или в отдельном окне под графиком символа.

Свойство:

#property indicator_chart_window

Определяет отрисовку индикатора на графике символа.

А свойство:

#property indicator_separate_window

Определяет вывод индикатора в отдельное окно.

Одно из самых важных свойств индикатора – это количество буферов для расчета индикатора, например:

#property indicator_buffers 6

Данное свойство тесно связано с двумя другими свойствами индикатора – количеством графических построений и видом графических построений.

Количество графических построений – это количество цветных диаграмм, составляющих индикатор.

Например, для индикатора ADX:

#property indicator_plots 3

Индикатор состоит из трех диаграмм (линий) – индикатора направленности +DI, индикатора направленности –DI и самого индикатора ADX.

Вид графических построений – это та графическая форма, из которой составляется график индикатора.

Например, для индикатора ADX:

#property indicator_type1 DRAW_LINE
#property indicator_type2 DRAW_LINE
#property indicator_type3 DRAW_LINE

Таким образом, каждая диаграмма индикатора ADX – это линия.

Графическая форма сопоставляется с графическим построением с помощью номера графического построения, следующего после indicator_type.

Цвет каждого графического построения индикатора задается свойством indicator_colorN.

Например, для индикатора ADX:

#property indicator_color1 LightSeaGreen
#property indicator_color2 YellowGreen
#property indicator_color3 Wheat

Цвет сопоставляется с графическим построением с помощью номера графического построения, следующего после indicator_color.

https://www.mql5.com/ru/docs/constants/objectconstants/webcolors

В справочнике MQL5 есть таблица Web-цветов для определения цвета графического построения.

```
//---- buffers
double    ExtADXBuffer[];
double    ExtPDIBuffer[];
double    ExtNDIBuffer[];
double    ExtPDBuffer[];
double    ExtNDBuffer[];
double    ExtTmpBuffer[];
//--- global variables
int       ExtADXPeriod;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- check for input parameters
   if(InpPeriodADX>=100 || InpPeriodADX<=0)
     {
      ExtADXPeriod=14;
      printf("Incorrect value for input variable Period_ADX=%d. Indicator wil
     }
   else ExtADXPeriod=InpPeriodADX;
//---- indicator buffers
   SetIndexBuffer(0,ExtADXBuffer);
   SetIndexBuffer(1,ExtPDIBuffer);
   SetIndexBuffer(2,ExtNDIBuffer);
   SetIndexBuffer(3,ExtPDBuffer,INDICATOR_CALCULATIONS);
   SetIndexBuffer(4,ExtNDBuffer,INDICATOR_CALCULATIONS);
   SetIndexBuffer(5,ExtTmpBuffer,INDICATOR_CALCULATIONS);
//--- indicator digits
   IndicatorSetInteger(INDICATOR_DIGITS,2);
```

Вернемся теперь к количеству буферов для расчета индикатора.

Так как данные для построения каждой диаграммы индикатора берутся из своего буфера индикатора, количество заявленных буферов индикатора не может быть меньше, чем заявленное число графических построений индикатора.

Сразу же возникает вопрос, каким образом конкретный массив, представляющий буфер индикатора, сопоставляется с конкретным графическим построением индикатора.

Делается это в функции обратного вызова OnInit() с помощью вызова функции SetIndexBuffer.

Например, для индикатора ADX:

SetIndexBuffer(0,ExtADXBuffer);

SetIndexBuffer(1,ExtPDIBuffer);

SetIndexBuffer(2,ExtNDIBuffer);

Где первый аргумент, это номер графического построения.

Таким образом, массив связывается с диаграммой индикатора, а диаграмма связывается с ее формой и цветом.

Однако с буферами индикатора все немного сложнее.

Их количество может быть заявлено больше, чем количество графических построений индикатора.

Что это означает?

Это означает, что некоторые массивы, представляющие буфера индикатора, используются не для построения диаграмм индикатора, а для промежуточных вычислений.

Например, для индикатора ADX:

SetIndexBuffer(3,ExtPDBuffer,INDICATOR_CALCULATIONS);

SetIndexBuffer(4,ExtNDBuffer,INDICATOR_CALCULATIONS);

SetIndexBuffer(5,ExtTmpBuffer,INDICATOR_CALCULATIONS);

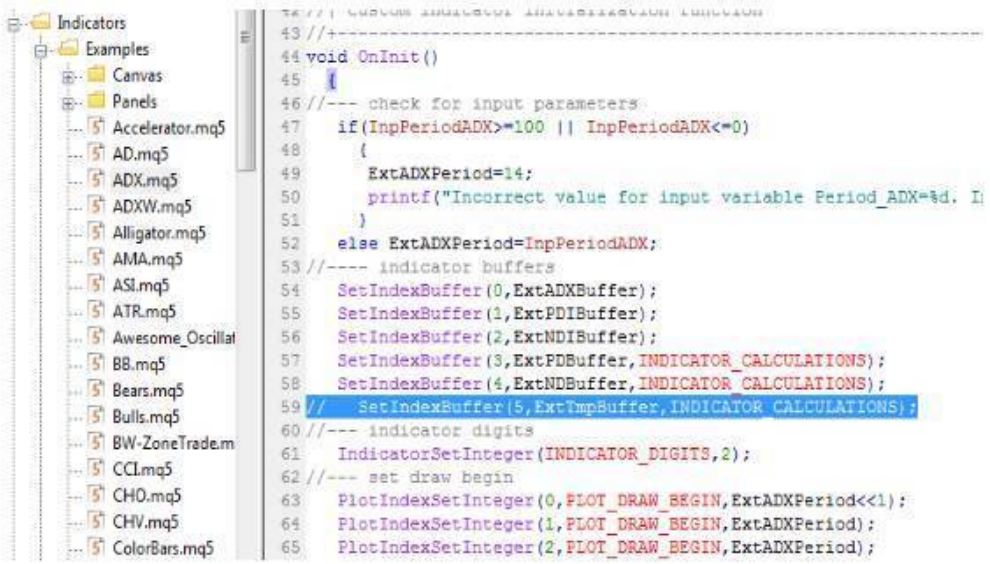
Такой массив определяется с помощью третьего параметра INDICATOR_CALCULATIONS.

Это дает следующее:

Все дело в частичном заполнении массива.

Если массив, указанный в функции SetIndexBuffer, является динамическим, т.е. объявлен без указания размера, но он привязан к буферу индикатора с помощью функции SetIndexBuffer, клиентский терминал сам заботится о том, чтобы размер такого массива соответствовал ценовой истории.

Рассмотрим это на примере индикатора ADX.



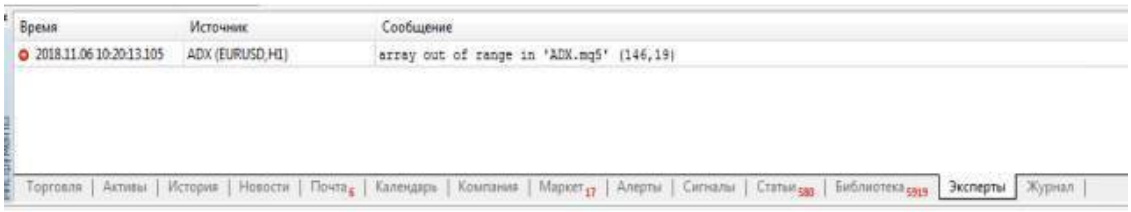
В редакторе MQL5, в окне Navigator (Навигатор), в разделе Indicators->Examples выберем и откроем исходный код индикатора ADX.

В функции OnInit() закомментируем строку:

// SetIndexBuffer(5,ExtTmpBuffer,INDICATOR_CALCULATIONS);

Теперь массив ExtTmpBuffer является просто динамическим массивом.

Откомпилируем код индикатора и присоединим индикатор к графику в терминале MetaTrader 5.



В результате Терминал выдаст ошибку.

array out of range

Это произошло потому, что мы перед заполнением данного массива значениями не указали его размера и не зарезервировали под него память.

Так что его размер был равен нулю, когда мы попытались в него что-то записать.

Статическим мы этот массив сделать тоже не можем, т.е. объявить его сразу с указанием размера, так как значения такого промежуточного массива рассчитываются в функции обратного вызова OnCalculate на основе загруженной в функцию OnCalculate истории цен, а именно массивов open[], high[], low[], и close[].

Но точный размер массивов open[], high[], low[], и close[] неизвестен, он обозначается лишь переменной rates_total.

```
73 //+------------------------------------------------------------------+
74 //| Custom indicator iteration function                              |
75 //+------------------------------------------------------------------+
76 int OnCalculate(const int rates_total,
77                  const int prev_calculated,
78                  const datetime &time[],
79                  const double &open[],
80                  const double &high[],
81                  const double &low[],
82                  const double &close[],
83                  const long &tick_volume[],
84                  const long &volume[],
85                  const int &spread[])
86    {
87    ArrayResize(ExtTmpBuffer,rates_total);
88 //--- checking for bars count
89    if(rates_total<ExtADXPeriod)
90       return(0);
91 //--- detect start position
92    int start;
93    if(prev_calculated>1) start=prev_calculated-1;
94    else
95      {
96       start=1;
97       ExtPDIBuffer[0]=0.0;
98       ExtNDIBuffer[0]=0.0;
99       ExtADXBuffer[0]=0.0;
100     }
```

Хорошо, но мы можем в функции OnCalculate применить функцию ArrayResize, чтобы установить размер массива:

ArrayResize(ExtTmpBuffer,rates_total);

Передав в функцию в качестве аргумента переменную rates_total – количество баров на графике, на котором запущен индикатор.

Теперь после компиляции индикатор заработает как надо.

Но дело в том, что в функции OnCalculate мы сначала рассчитываем индикатор для всей ценовой истории, т.е. для rates_total значений, а затем при поступлении нового тика по символу индикатора, и соответственно вызове функции OnCalculate, мы рассчитываем значение индикатора для этого нового тика по символу и записываем новое значение индикатора в его массив буфера.

Чтобы это реализовать с промежуточным массивом, нужно внимательно следить за его размером и записывать новое значение в конец массива.

Вместо всего этого, проще всего привязать промежуточный массив к буферу индикатора с помощью функции SetIndexBuffer и таким образом решить все эти проблемы.

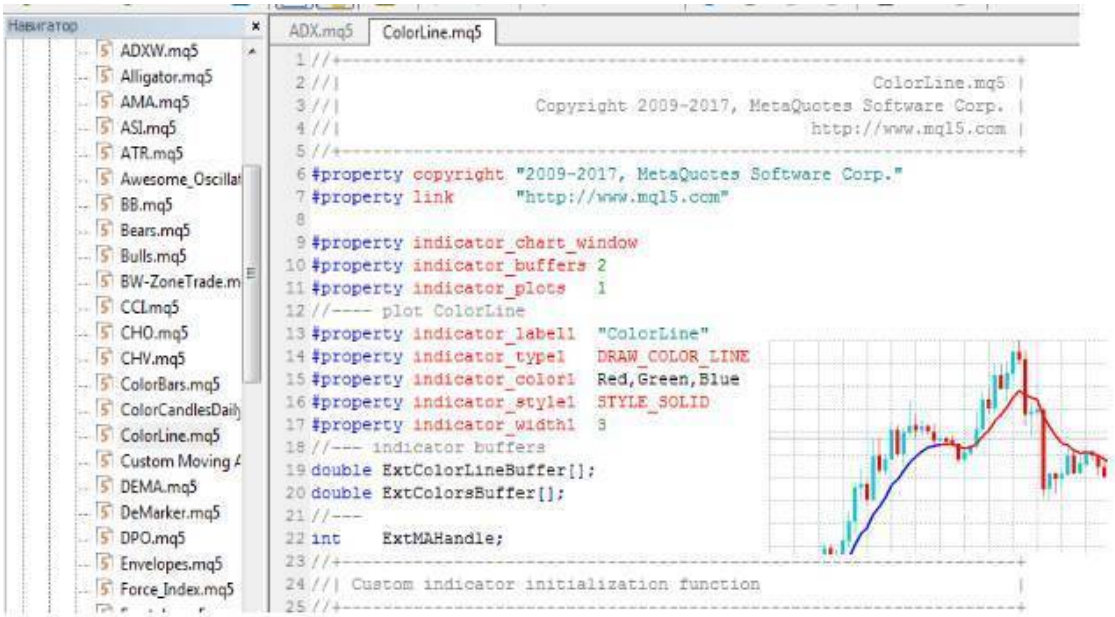

```
int  CopyBuffer(
   int       indicator_handle,     // handle индикатора
   int       buffer_num,           // номер буфера индикатора
   int       start_pos,            // откуда начнем
   int       count,                // сколько копируем
   double    buffer[]              // массив, куда будут скопированы данные
   );
```

Аналогичная ситуация возникает, когда значения таких промежуточных массивов заполняются с помощью функции CopyBuffer, когда мы строим пользовательский индикатор на основе других индикаторов.

Функция CopyBuffer распределяет размер принимающего массива под размер копируемых данных.

Если копируется вся ценовая история, то проблем нет и в этом случае использовать INDICATOR_CALCULATIONS необязательно.

Если же мы хотим скопировать только одно новое поступившее значение, функция CopyBuffer определит размер принимающего массива как 1, и нужно будет использовать этот принимающий массив как еще один массив-посредник, из которого уже записывать значение в промежуточный массив индикатора. И в этом случае просто функцией ArrayResize для принимающего массива проблему не решить.



Теперь что нам делать, если мы хотим раскрашивать наши диаграммы индикатора в разные цвета в зависимости от цены?

Во-первых, мы должны указать, что наша графическая форма нашего графического построения является цветной, например:

#property indicator_type1 DRAW_COLOR_LINE

В идентификатор геометрической формы добавляется слово COLOR.

Далее значение свойства #property indicator_buffers увеличивается на единицу и объявляется еще один массив для хранения цвета.

```
#property indicator_buffers 2
#property indicator_plots   1
//---- plot ColorLine
#property indicator_label1  "ColorLine"
#property indicator_type1   DRAW_COLOR_LINE
#property indicator_color1  Red,Green,Blue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  3
//--- indicator buffers
double ExtColorLineBuffer[];
double ExtColorsBuffer[];
//---
int    ExtMAHandle;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ExtColorLineBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ExtColorsBuffer,INDICATOR_COLOR_INDEX);
//--- get MA handle
   ExtMAHandle=iMA(Symbol(),0,10,0,MODE_EMA,PRICE_CLOSE);
  }
```

Функцией SetIndexBuffer объявленный дополнительный массив сопоставляется с буфером цвета индикатора, например:

SetIndexBuffer(1,ExtColorsBuffer,INDICATOR_COLOR_INDEX);

В свойстве #property indicator_color, раскрашиваемого графического построения, указывается несколько цветов, например:

#property indicator_color1 Red,Green,Blue

```
//+------------------------------------------------------------------+
int getIndexOfColor(int i)
  {
   int j=i%300;
   if(j<100) return(0);// first index
   if(j<200) return(1);// second index
   return(2); // third index
  }
```

```
      //--- now set line color for every bar
      for(int i=0;i<rates_total && !IsStopped();i++)
         ExtColorsBuffer[i]=getIndexOfColor(i);
   }
```

И, наконец, каждому элементу массива, представляющего буфер цвета индикатора, присваивается номер цвета, определенный в свойстве #property indicator_color.

В данном случае, это 0, 1 и 2.

Теперь при отрисовке диаграммы индикатора, из буфера берется значение диаграммы, по позиции значения оно сопоставляется со значением буфера цвета, и элемент диаграммы становится цветным.

```
//---- plot ColorLine
#property indicator_label1  "ColorLine"
#property indicator_type1   DRAW_COLOR_LINE
//#property indicator_color1  Red,Green,Blue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  3
//--- indicator buffers
double ExtColorLineBuffer[];
double ExtColorsBuffer[];
//---
int    ExtMAHandle;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {

  PlotIndexSetInteger(0,PLOT_COLOR_INDEXES,3);
  PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,Red);
  PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,Green);
  PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,Blue);
```

Вместо свойства #property indicator_color, цвета графического построения можно задать программным способом:

Задаем количество индексов цветов для графического построения с помощью функции:

PlotIndexSetInteger(0,PLOT_COLOR_INDEXES,3);

И задаем цвет для каждого индекса с помощью функции:

PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,Red);

Где первый параметр – индекс графического построения, соответственно первое графическое построение имеет индекс 0.

Это идентично объявлению:

#property indicator_color1 Red,Green,Blue

```
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Average Directional Movement Index"
#include <MovingAverages.mqh>

#property indicator_separate_window
//#property indicator_chart_window
#property indicator_buffers 6
#property indicator_plots   3
#property indicator_type1   DRAW_LINE
#property indicator_color1  LightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
#property indicator_type2   DRAW_LINE
#property indicator_color2  YellowGreen
#property indicator_style2  STYLE_DOT
#property indicator_width2  1
#property indicator_type3   DRAW_LINE
#property indicator_color3  Wheat
#property indicator_style3  STYLE_DOT
#property indicator_width3  1
#property indicator_label1  "ADX"
#property indicator_label2  "+DI"
#property indicator_label3  "-DI"
```

Давайте продолжим рассмотрение свойств индикатора.

Толщина линии диаграммы индикатора задается свойством indicator_widthN, где N – номер графического построения, например:

#property indicator_width1 1

Также можно задать стиль линии диаграммы индикатора – сплошная линия, прерывистая, пунктирная, штрих-пунктирная, штрих – с помощью свойства indicator_styleN, где N – номер графического построения, например:

#property indicator_style1 STYLE_SOLID

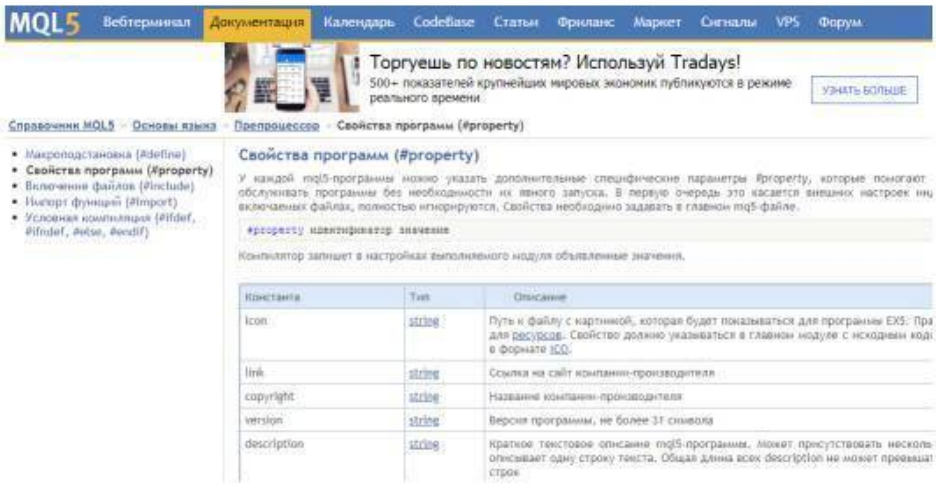
И, наконец, свойство indicator_labelN указывает метки диаграмм индикатора в DataWindow или Окно данных, например:

#property indicator_label1 "ADX"

#property indicator_label2 "+DI"

#property indicator_label3 "-DI"

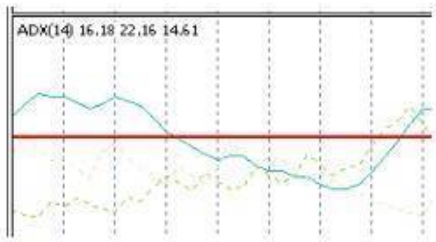
https://www.mql5.com/ru/docs/basis/preprosessor/compilation



Другие свойства индикатора можно посмотреть в справочнике.

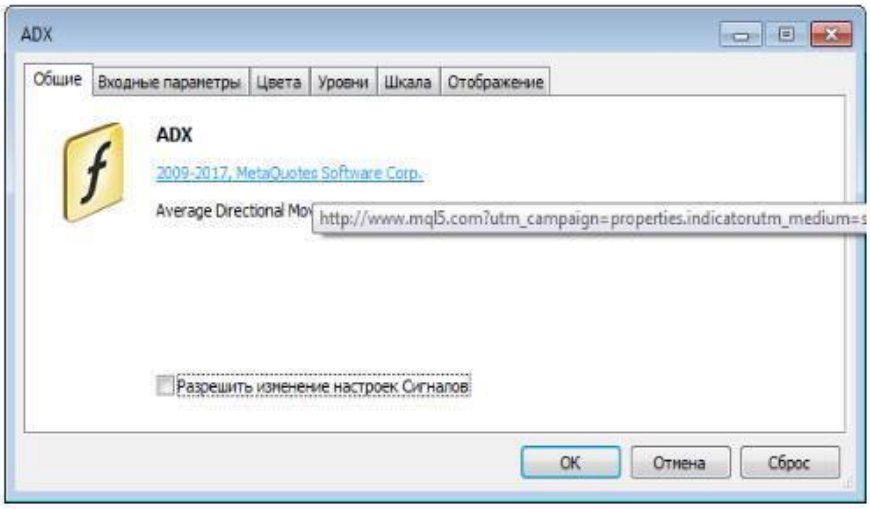
Правда можно отметить еще одну группу свойств, которая позволяет нарисовать горизонтальный уровень индикатора в отдельном окне, например:

```
#property indicator_separate_window
//#property indicator_chart_window
#property indicator_buffers 6
#property indicator_plots   3
#property indicator_type1   DRAW_LINE
#property indicator_color1  LightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
#property indicator_type2   DRAW_LINE
#property indicator_color2  YellowGreen
#property indicator_style2  STYLE_DOT
#property indicator_width2  1
#property indicator_type3   DRAW_LINE
#property indicator_color3  Wheat
#property indicator_style3  STYLE_DOT
#property indicator_width3  1
#property indicator_label1  "ADX"
#property indicator_label2  "+DI"
#property indicator_label3  "-DI"

#property indicator_level1 30.0
#property indicator_levelcolor Red
#property indicator_levelstyle STYLE_SOLID
#property indicator_levelwidth 2
```

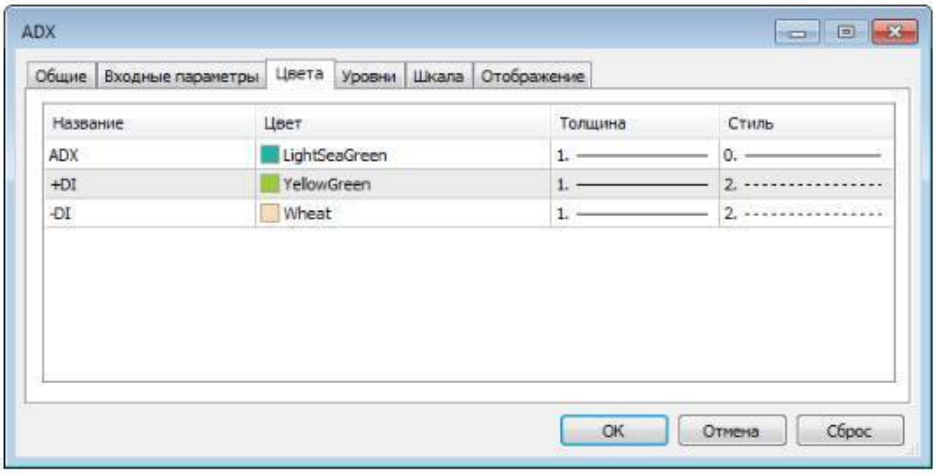


#property indicator_level1 30.0
#property indicator_levelcolor Red
#property indicator_levelstyle STYLE_SOLID
#property indicator_levelwidth 2

В результате добавления этих строк кода в индикатор ADX, у него появится горизонтальный уровень.



Теперь, на примере индикатора ADX, при присоединении индикатора к графику в MetaTrader 5, во-первых, откроется диалоговое окно индикатора, которое во вкладке Общие отобразит значения свойств copyright, link и description.



А во вкладке Цвета отобразятся значения свойств indicator_label, indicator_color, indicator_width, indicator_style.

Само же название индикатора определяется именем файла индикатора.

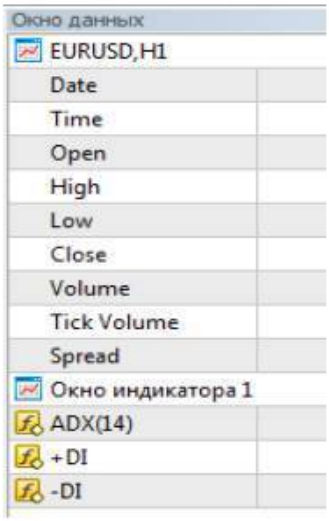
К слову сказать, диалоговое окно индикатора можно открыть и после присоединения индикатора к графику, с помощью контекстного меню, щелкнув правой кнопкой мышки на индикаторе и выбрав свойства индикатора.

При наведении курсора на название индикатора в окне Navigator терминала всплывает подсказка, отображающая свойство copyright.


```
#property indicator_label1  "ADX"
```

```
67 //--- indicator short name
68    string short_name="ADX("+string(ExtADXPeriod)+")";
69    IndicatorSetString(INDICATOR_SHORTNAME,short_name);
```

После присоединения индикатора свойство:
#property indicator_label1 "ADX"
работать не будет, так как в функции OnInit() с помощью вызова функции:
string short_name="ADX("+string(ExtADXPeriod)+")";
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
изменена подпись индикатора на ADX(14) – период индикатора.




```
70 //--- change 1-st index label
71    PlotIndexSetString(0,PLOT_LABEL,short_name);
```

А вызовом функции:

PlotIndexSetString(0,PLOT_LABEL,short_name);

изменена метка индикатора в окне Окно Данных, которое открывается в меню Вид терминала.

Значения же свойств:

#property indicator_label2 "+DI"

#property indicator_label3 "-DI"

отображаются, как и было определено, во всплывающих подсказках к диаграммам индикатора и отображаются в окне Окно Данных.

```
5  //+------------------------------------------------------------------+
6  #property copyright   "2009-2017, MetaQuotes Software Corp."
7  #property link        "http://www.mql5.com"
8  #property description "Average Directional Movement Index"
9  #include <MovingAverages.mqh>
10
11 #property indicator_separate_window
12 //#property indicator_chart_window
13 #property indicator_buffers 6
14 #property indicator_plots   3
15 #property indicator_type1   DRAW_LINE
16 #property indicator_color1  LightSeaGreen
17 #property indicator_style1  STYLE_SOLID
18 #property indicator_width1  1
19 #property indicator_type2   DRAW_LINE
20 #property indicator_color2  YellowGreen
21 #property indicator_style2  STYLE_DOT
22 #property indicator_width2  1
23 #property indicator_type3   DRAW_LINE
24 #property indicator_color3  Wheat
25 #property indicator_style3  STYLE_DOT
26 #property indicator_width3  1
27 #property indicator_label1  "ADX"
28 #property indicator_label2  "+DI"
29 #property indicator_label3  "-DI"
30
```

```
54 //---- indicator buffers
55    SetIndexBuffer(0,ExtADXBuffer);
56    SetIndexBuffer(1,ExtPDIBuffer);
57    SetIndexBuffer(2,ExtNDIBuffer);
58    SetIndexBuffer(3,ExtPDBuffer,INDICATOR_CALCULATIONS);
59    SetIndexBuffer(4,ExtNDBuffer,INDICATOR_CALCULATIONS);
60    SetIndexBuffer(5,ExtTmpBuffer,INDICATOR_CALCULATIONS);
```

В коде индикатора ADX объявленное количество буферов индикатора больше, чем количество графических построений.

Свойство indicator_buffers равно 6

А свойство indicator_plots равно 3

Сделано это для того, чтобы использовать три буфера индикатора для промежуточных расчетов.

Это массивы ExtPDBuffer, ExtNDBuffer и ExtTmpBuffer.

В функции OnCalculate индикатора, значения массивов ExtPDBuffer, ExtNDBuffer, ExtTmpBuffer рассчитываются на основе загруженной ценовой истории, а затем уже на их основе рассчитываются значения массивов ExtADXBuffer, ExtPDIBuffer, ExtNDIBuffer, которые используются для отрисовки диаграмм индикатора.

Как уже было сказано, буферы индикатора для промежуточных вычислений здесь объявляются с константой INDICATOR_CALCULATIONS, так как заранее неизвестен размер загружаемой ценовой истории.

Теперь, в описании индикатора ADX сказано, что:

Сигнал на покупку формируется тогда, когда +DI поднимается выше – DI и при этом сам ADX растет.

В момент, когда +DI расположен выше – DI, но сам ADX начинает снижаться, индикатор подает сигнал о том, что рынок «перегрет» и пришло время фиксировать прибыль.

Сигнал на продажу формируется тогда, когда +DI опускается ниже – DI и при этом ADX растет.

В момент, когда +DI расположен ниже – DI, но сам ADX начинает снижаться, индикатор подает сигнал о том, что рынок «перегрет» и пришло время фиксировать прибыль.

Давайте, модифицируем код индикатора ADX таким образом, чтобы раскрасить диаграмму ADX в четыре цвета, которые соответствуют описанным выше четырем торговым сигналам.

```
12 //#property indicator_chart_window
13 #property indicator_buffers 7
14 #property indicator_plots   3
15 #property indicator_type1   DRAW_COLOR_LINE
16 #property indicator_color1  LightSeaGreen
17 #property indicator_style1  STYLE_SOLID
18 #property indicator_width1  1
19 #property indicator_type2   DRAW_LINE
20 #property indicator_color2  YellowGreen
21 #property indicator_style2  STYLE_DOT
22 #property indicator_width2  1
23 #property indicator_type3   DRAW_LINE
24 #property indicator_color3  Wheat
25 #property indicator_style3  STYLE_DOT
26 #property indicator_width3  1
27 #property indicator_label1  "ADX"
28 #property indicator_label2  "+DI"
29 #property indicator_label3  "-DI"
30
31 //--- input parameters
32 input int InpPeriodADX=14; // Period
33 //---- buffers
34 double    ExtADXBuffer[];
35 double    ExtPDIBuffer[];
36 double    ExtNDIBuffer[];
37 double    ExtPDBuffer[];
38 double    ExtNDBuffer[];
39 double    ExtTmpBuffer[];
40 //Буфер цвета
41 double    ExtColorsBuffer[];
```

В качестве первого шага изменим свойство indicator_type1 на DRAW_COLOR_LINE. Далее увеличим на единицу значение свойства indicator_buffers на значение 7. Объявим массив для буфера цвета ExtColorsBuffer.

```
47 void OnInit()
48   {
49 //--- check for input parameters
50    if(InpPeriodADX>=100 || InpPeriodADX<=0)
51      {
52       ExtADXPeriod=14;
53       printf("Incorrect value for input variable Period_ADX=%d.
54      }
55    else ExtADXPeriod=InpPeriodADX;
56 //---- indicator buffers
57 SetIndexBuffer(0,ExtADXBuffer);
58 SetIndexBuffer(1,ExtColorsBuffer,INDICATOR_COLOR_INDEX);
59 SetIndexBuffer(2,ExtPDIBuffer);
60 SetIndexBuffer(3,ExtNDIBuffer);
61 SetIndexBuffer(4,ExtPDBuffer,INDICATOR_CALCULATIONS);
62 SetIndexBuffer(5,ExtNDBuffer,INDICATOR_CALCULATIONS);
63 SetIndexBuffer(6,ExtTmpBuffer,INDICATOR_CALCULATIONS);
```

И в функции OnInit() свяжем объявленный массив с буфером цвета с помощью функции SetIndexBuffer.

Тут есть хитрость – индекс буфера цвета должен следовать за индексом буфера значений индикатора.

Если, например, связать массив ExtColorsBuffer с буфером с индексом 6, тогда индикатор не будет корректно отрисовываться.

```
#property indicator_separate_window
//#property indicator_chart_window
#property indicator_buffers 7
#property indicator_plots   3
#property indicator_type1   DRAW_COLOR_LINE
#property indicator_color1  LightSeaGreen, clrBlue, clrLightBlue, clrRed, clrLightPink
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
#property indicator_type2   DRAW_LINE
#property indicator_color2  YellowGreen
#property indicator_style2  STYLE_DOT
#property indicator_width2  1
#property indicator_type3   DRAW_LINE
#property indicator_color3  Wheat
#property indicator_style3  STYLE_DOT
#property indicator_width3  1
#property indicator_label1  "ADX"
#property indicator_label2  "+DI"
#property indicator_label3  "-DI"
```

В свойство indicator_color1 добавим цветов.
И увеличим толщину линии с помощью свойства indicator_width1.

```
ExtColorsBuffer[i]=0;
if(ExtPDIBuffer[i]>ExtNDIBuffer[i]&&ExtADXBuffer[i]>ExtADXBuffer[i-1]){
ExtColorsBuffer[i]=1;
}
if(ExtPDIBuffer[i]>ExtNDIBuffer[i]&&ExtADXBuffer[i]<ExtADXBuffer[i-1]){
ExtColorsBuffer[i]=2;
}
if(ExtPDIBuffer[i]<ExtNDIBuffer[i]&&ExtADXBuffer[i]>ExtADXBuffer[i-1]){
ExtColorsBuffer[i]=3;
}
if(ExtPDIBuffer[i]<ExtNDIBuffer[i]&&ExtADXBuffer[i]<ExtADXBuffer[i-1]){
ExtColorsBuffer[i]=4;
}
```

В функции OnCalculate в конце перед закрывающей скобкой цикла for добавим код заполнения буфера цвета значениями согласно описанной нами стратегии.

Откомпилируем код и получим индикатор с визуальным отображением сигналов на покупку и продажу:

```
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Relative Strength Index"
//--- indicator settings
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
#property indicator_level1 30
#property indicator_level2 70
#property indicator_buffers 3
#property indicator_plots   1
#property indicator_type1   DRAW_LINE
#property indicator_color1  DodgerBlue
//--- input parameters
input int InpPeriodRSI=14; // Period
//--- indicator buffers
double    ExtRSIBuffer[];
double    ExtPosBuffer[];
double    ExtNegBuffer[];
//--- global variable
int       ExtPeriodRSI;
//+------------------------------------------------------------------+
```

В редакторе MQL5 откроем другой индикатор из папки Examples – RSI.

Данный индикатор имеет два ключевых уровня, которые определяют области перекупленности и перепроданности.

В коде индикатора эти уровни определены как свойства:

#property indicator_level1 30

#property indicator_level2 70

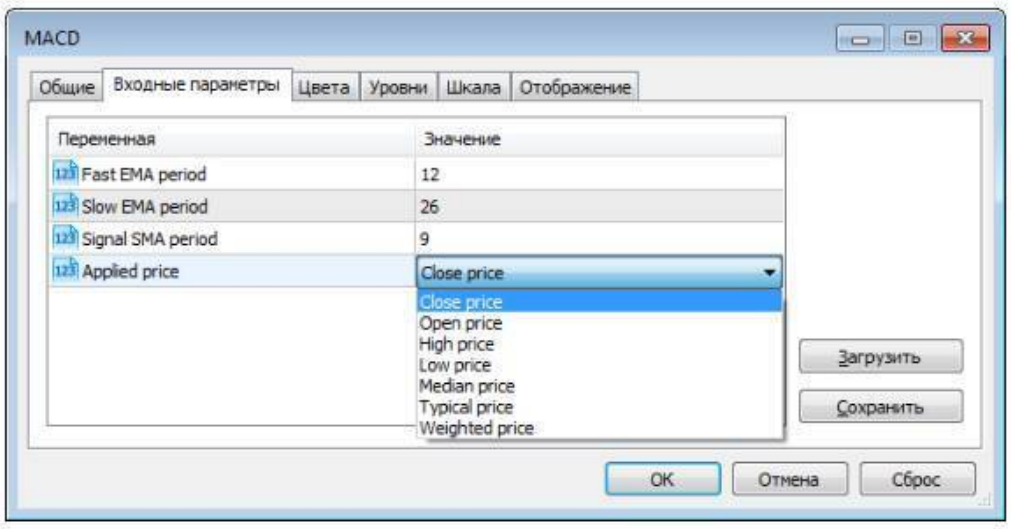
Давайте улучшим отображение этих уровней, добавив им цвета и стиля.

```
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Relative Strength Index"
//--- indicator settings
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
#property indicator_level1 30
#property indicator_level2 70
#property indicator_buffers 3
#property indicator_plots   1
#property indicator_type1   DRAW_LINE
#property indicator_color1  DodgerBlue

#property indicator_levelcolor Red
#property indicator_levelstyle STYLE_SOLID
#property indicator_levelwidth 1
```

Для этого добавим свойства:
#property indicator_levelcolor Red
#property indicator_levelstyle STYLE_SOLID
#property indicator_levelwidth 1
Теперь индикатор будет выглядеть следующим образом.

Параметры ввода и переменные индикатора

Параметры ввода – это те параметры индикатора, которые отображаются пользователю перед присоединением индикатора к графику во вкладке Входные параметры диалогового окна.



Например, для индикатора MACD – это периоды скользящих средних и тип применяемой цены.

Здесь пользователь может поменять параметры индикатора по умолчанию, и индикатор присоединится к графику с уже измененными параметрами.

Также пользователь может поменять параметры индикатора после присоединения индикатора к графику, щелкнув правой кнопкой мышки на индикаторе и выбрав свойства индикатора.

```
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Moving Average Convergence/Divergence"
#include <MovingAverages.mqh>
//--- indicator settings
#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   2
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_type2   DRAW_LINE
#property indicator_color1  Silver
#property indicator_color2  Red
#property indicator_width1  2
#property indicator_width2  1
#property indicator_label1  "MACD"
#property indicator_label2  "Signal"
//--- input parameters
input int                InpFastEMA=12;              // Fast EMA period
input int                InpSlowEMA=26;              // Slow EMA period
input int                InpSignalSMA=9;             // Signal SMA period
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE; // Applied price
//--- indicator buffers
double                   ExtMacdBuffer[];
double                   ExtSignalBuffer[];
double                   ExtFastMaBuffer[];
double                   ExtSlowMaBuffer[];
//--- MA handles
int                      ExtFastMaHandle;
int                      ExtSlowMaHandle;
//+------------------------------------------------------------------+
```

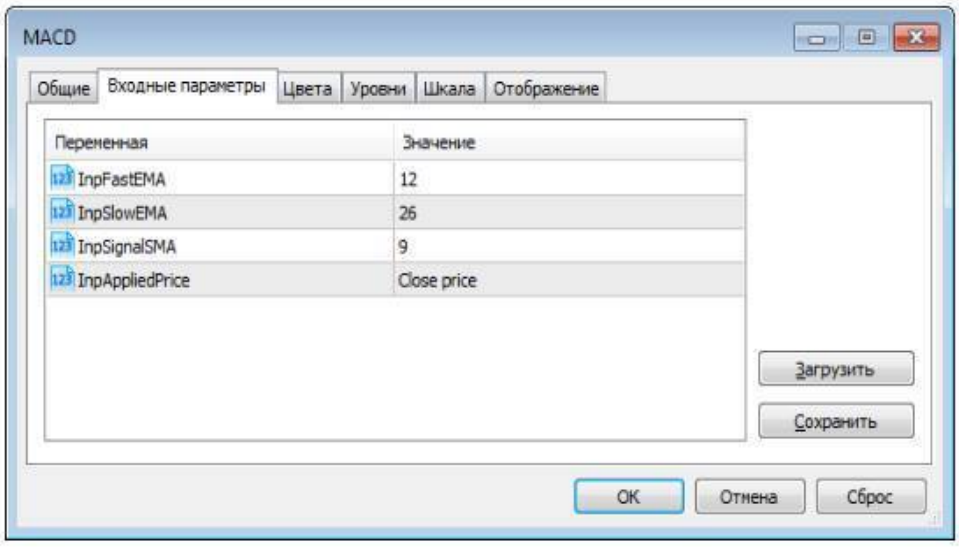
В коде индикатора такие параметры задаются input переменными с модификатором input, который указывается перед типом данных. Как правило, input переменные объявляются сразу после свойств индикатора.

Например, для индикатора MACD – это периоды для

экспоненциальной скользящей средней с коротким периодом от цены, экспоненциальной скользящей средней с длинным периодом от цены, сглаживающей скользящей средней с коротким периодом от разницы двух остальных скользящих, и тип применяемой цены.

Здесь надо отметить то, что в диалоговом окне присоединения индикатора к графику отображаются не имена переменных, а комментарии к ним.

Если убрать комментарии, входные параметры отобразятся следующим образом.



Здесь уже отображаются имена переменных.

Как вы сами, наверное, уже догадались, комментарии используются для отображения, чтобы облегчить пользователю понимание их предназначения.

Здесь также видно, что входными параметрами могут быть не только отдельные переменные, но и перечисления, которые отображаются в виде выпадающих списков.

```
22 //--- input parameters
23 input int                  InpFastEMA=12;              // Fast EMA period
24 input int                  InpSlowEMA=26;              // Slow EMA period
25 input int                  InpSignalSMA=9;             // Signal SMA period
26 input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE; // Applied price
27 //--- indicator buffers
28 double                   ExtMacdBuffer[];
29 double                   ExtSignalBuffer[];
30 double                   ExtFastMaBuffer[];
31 double                   ExtSlowMaBuffer[];
```

Для индикатора MACD используется встроенное перечисление ENUM_APPLIED_PRICE, но можно также определить и свое перечисление.

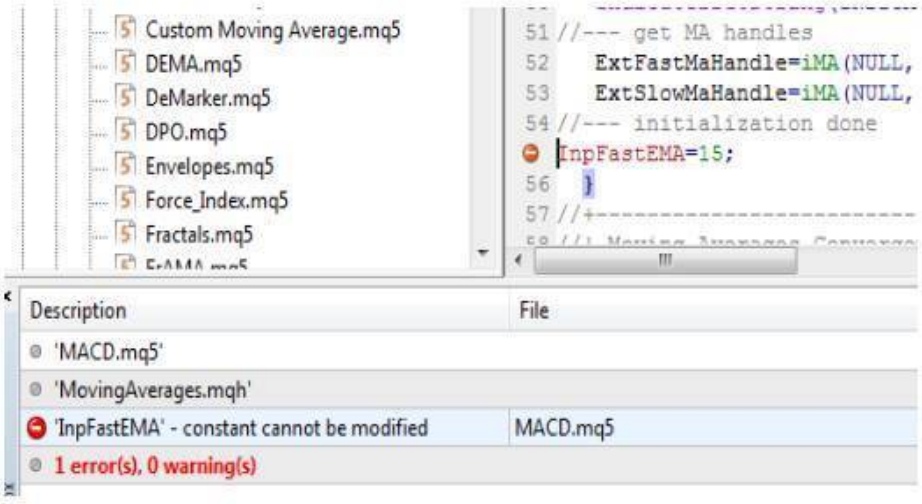
В справочнике приводится соответствующий пример.

```
#property script_show_inputs
//--- day of week
enum dayOfWeek
{
  S=0,    // Sunday
  M=1,    // Monday
  T=2,    // Tuesday
  W=3,    // Wednesday
  Th=4,   // Thursday
  Fr=5,   // Friday,
  St=6,   // Saturday
 };
//--- input parameters
input dayOfWeek swapday=W;
```

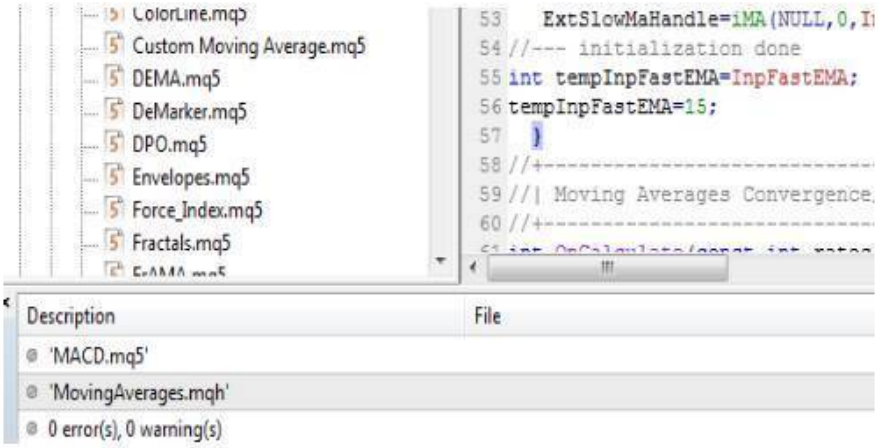
В этом примере команда #property script_show_inputs используется для скриптов, для индикаторов ее можно опустить.

Основное отличие input переменных от других типов переменных состоит в том, что изменить их значение может только пользователь в диалоговом окне индикатора.

Если в коде индикатора попытаться изменить значение входного параметра, при компиляции возникнет ошибка.



Поэтому, если вы хотите при расчетах использовать измененное значение входного параметра, нужно использовать промежуточную переменную.



Помимо input переменных MQL5-код использует локальные переменные, статические переменные, глобальные переменные и extern переменные.

Классы памяти

Существуют три класса памяти: static, input и extern. Эти модификаторы класса памяти явно указывают компилятору, что соответствующие переменные распределяются в предопределенной области памяти, называемой глобальным пулом. При этом данные модификаторы указывают на особую обработку данных переменных.

Если переменная, объявленная на локальном уровне, не является статической, то распределение памяти под такую переменную производится автоматически на программном стеке. Освобождение памяти, выделенной под не статический массив, производится также автоматически при выходе за пределы области видимости блока, в котором массив объявлен.

С локальными переменными в принципе все понятно, они объявляются в блоке кода, например, в цикле или функции, там же инициализируются, и, после выполнения блока кода, память, выделенная под локальные переменные в программном стеке, освобождается.

Тут особо надо отметить, что для локальных объектов, созданных с помощью оператора new, в конце блока кода нужно применить оператор delete для освобождения памяти.

Глобальные переменные, как правило, объявляются после свойств индикатора, входных параметров и массивов буферов индикатора, перед функциями.

Глобальные переменные видны в пределах всей программы, их значение может быть изменено в любом месте программы и память, выделяемая под глобальные переменные вне программного стека, освобождается при выгрузке программы.

Здесь видно, что input переменные – это те же глобальные переменные, за исключением опции – их значение не может быть изменено в любом месте программы.

Если глобальную или локальную переменную объявить со спецификатором const – это так же не позволит изменять значение этой переменной в процессе выполнения программы.

Статические переменные определяются модификатором static, который указывается перед типом данных.

Со статическими переменными все немного сложнее, но легче всего их понять, сравнивая статические переменные с локальными и глобальными переменными.

В принципе, статическая переменная, объявленная там же, где и глобальная переменная, ничем не отличается от глобальной переменной.

Хитрость начинается, если локальную переменную объявить с модификатором static.

В этом случае, после выполнения блока кода, память, выделенная под статическую переменную, не освобождается. И при следующем выполнении того же блока кода, предыдущее значение статической переменной можно использовать.

Хотя область видимости такой статической переменной ограничивается те же самым блоком кода, в котором она была объявлена.

extern переменные это аналог статических глобальных переменных. Нельзя объявить локальную переменную с модификатором extern.

Отличие extern переменных от статических глобальных переменных проще всего продемонстрировать на индикаторе MACD.

```
//+------------------------------------------------------------------+
//|                                                        MACD.mq5  |
//|                  Copyright 2009-2017, MetaQuotes Software Corp.  |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Moving Average Convergence/Divergence"
#include <MovingAverages.mqh>
//--- indicator settings
#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   2
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_type2   DRAW_LINE
#property indicator_color1  Silver
#property indicator_color2  Red
#property indicator_width1  2
#property indicator_width2  1
#property indicator_label1  "MACD"
#property indicator_label2  "Signal"
```

extern int a=0;

static int a=0;

Индикатор MACD имеет включаемый файл MovingAverages, обозначенный с помощью директивы #include и расположенный в папке Include.

Если в файле MovingAverages и файле MACD одновременно объявить extern-переменную:

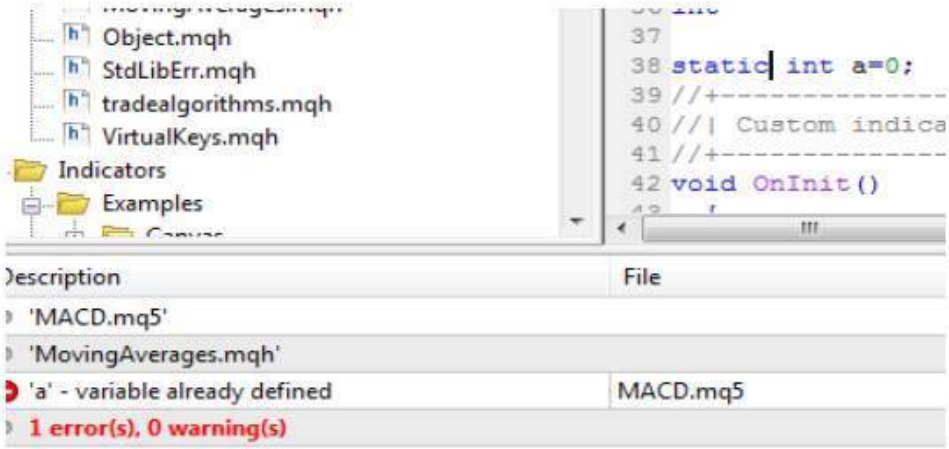
extern int a=0;

то при компиляции обоих файлов все пройдет удачно, и переменную можно будет использовать.

Если же в файле MovingAverages и файле MACD одновременно объявить статическую глобальную переменную:

static int a=0;

тогда при компиляции обоих файлов возникнет ошибка.



Помимо команды #include полезной является также директива #define, которая позволяет делать подстановку выражения вместо идентификатора, например:

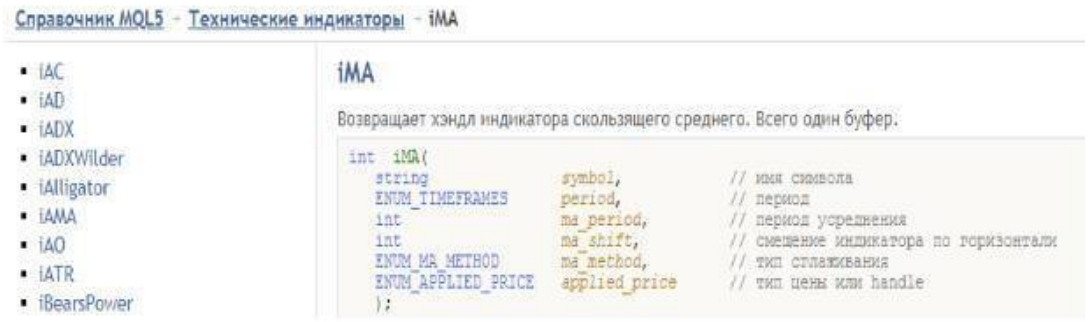
```
#define ABC                 100
#define PI                  3.14
#define COMPANY_NAME        "MetaQuotes Software Corp."
...
void ShowCopyright()
  {
   Print("Copyright  2001-2009, ",COMPANY_NAME);
   Print("https://www.metaquotes.net");
  }
```

#define PI 3.14

Хэндл индикатора

Начнем с цитаты:

HANDLE идентифицирует объект, которым Вы можете манипулировать. Джеффри РИХТЕР "Windows для профессионалов".



Переменные типа handle представляют собой указатель на некоторую системную структуру или индекс в некоторой системной таблице, которая содержит адрес структуры.

Таким образом, получив хэндл некоторого индикатора, мы можем использовать его данные для построения своего индикатора.

Хэндл индикатора представляет собой переменную типа int и объявляется, как правило, после объявления массивов буферов индикатора, вместе с глобальными переменными, например в индикаторе MACD:

```
//--- indicator buffers
double                  ExtMacdBuffer[];
double                  ExtSignalBuffer[];
double                  ExtFastMaBuffer[];
double                  ExtSlowMaBuffer[];
//--- MA handles
int                     ExtFastMaHandle;
int                     ExtSlowMaHandle;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ExtMacdBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ExtSignalBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,ExtFastMaBuffer,INDICATOR_CALCULATIONS);
   SetIndexBuffer(3,ExtSlowMaBuffer,INDICATOR_CALCULATIONS);
//--- sets first bar from what index will be drawn
   PlotIndexSetInteger(1,PLOT_DRAW_BEGIN,InpSignalSMA-1);
//--- name for Dindicator subwindow label
   IndicatorSetString(INDICATOR_SHORTNAME,"MACD("+string(InpFastEMA)+","+string(InpSlowEMA)+","+string(InpSignalSMA)+")");
//--- get MA handles
   ExtFastMaHandle=iMA(NULL,0,InpFastEMA,0,MODE_EMA,InpAppliedPrice);
   ExtSlowMaHandle=iMA(NULL,0,InpSlowEMA,0,MODE_EMA,InpAppliedPrice);
//--- initialization done
  }
```

Объявляются два хэндла – int ExtFastMaHandle и int ExtSlowMaHandle.

Здесь хэндлы индикаторов – это указатели на индикатор скользящего среднего с разными периодами 12 и 26.

Объявив эти переменные, мы, естественно, реально ничего не получаем, так как объекта индикатора, данные которого мы хотим использовать, еще не существует.

Создать в глобальном кеше клиентского терминала копию соответствующего технического индикатора и получить ссылку на нее можно несколькими способами.

Если это стандартный индикатор, проще всего получить его хэндл можно с помощью стандартной функции для работы с техническими индикаторами.

iMA

Возвращает хэндл индикатора скользящего среднего. Всего один буфер.

```
int  iMA(
   string               symbol,           // имя символа
   ENUM_TIMEFRAMES      period,           // период
   int                  ma_period,        // период усреднения
   int                  ma_shift,         // смещение индикатора по горизонтали
   ENUM_MA_METHOD       ma_method,        // тип сглаживания
   ENUM_APPLIED_PRICE   applied_price     // тип цены или handle
   );
```

Стандартная функция для индикатора скользящего среднего это функция iMA.

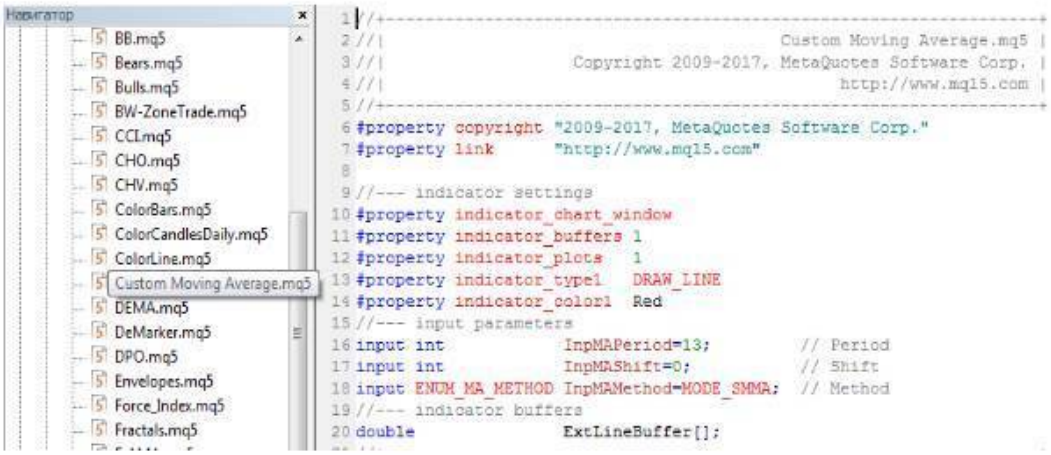
И в индикаторе MACD хэндлы индикатора скользящего среднего получаются с помощью вызова функции iMA в функции OnInit().

```
//--- get MA handles
   ExtFastMaHandle=iMA(NULL,0,InpFastEMA,0,MODE_EMA,InpAppliedPrice);
   ExtSlowMaHandle=iMA(NULL,0,InpSlowEMA,0,MODE_EMA,InpAppliedPrice);
```

```
//--- input parameters
input int                InpFastEMA=12;               // Fast EMA period
input int                InpSlowEMA=26;               // Slow EMA period
input int                InpSignalSMA=9;              // Signal SMA period
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE; // Applied price
```

где используются свойства индикатора – InpFastEMA, InpSlowEMA и InpAppliedPrice.

Предположим, что мы хотим использовать не стандартный, а пользовательский индикатор.



В папке Indicators/Examples редактора MQL5 есть нужный нам индикатор – это файл Custom Moving Average.mq5.

Для вызова того индикатора воспользуемся функцией iCustom.

iCustom

Возвращает хэндл указанного пользовательского индикатора.

```
int  iCustom(
   string           symbol,     // имя символа
   ENUM_TIMEFRAMES  period,     // период
   string           name        // папка/имя_пользовательского индикатора
   ...                          // список входных параметров индикатора
   );
```

В функции OnInit() индикатора MACD изменим код, где для получения хэндлов вместо стандартной функции, используем функцию iCustom.

```
// ExtFastMaHandle=iMA(NULL,0,InpFastEMA,0,MODE_EMA,InpAppliedPrice);
// ExtSlowMaHandle=iMA(NULL,0,InpSlowEMA,0,MODE_EMA,InpAppliedPrice);
ExtFastMaHandle=iCustom(NULL,0,"Examples\\Custom Moving Average", InpFastEMA,0,MODE_EMA,InpAppliedPrice);
ExtSlowMaHandle=iCustom(NULL,0,"Examples\\Custom Moving Average", InpSlowEMA,0,MODE_EMA,InpAppliedPrice);
```

После компиляции индикатора мы увидим, что его отображение никак не изменилось.

Еще один способ получить хэндл пользовательского индикатора, это использовать функцию IndicatorCreate.

IndicatorCreate

Возвращает хэндл указанного технического индикатора, созданного на основе массива параметров типа MqlParam.

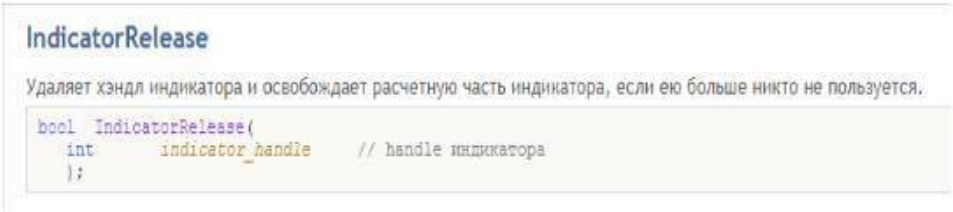
```
int  IndicatorCreate(
   string           symbol,                 // имя символа
   ENUM_TIMEFRAMES  period,                 // период
   ENUM_INDICATOR   indicator_type,         // тип индикатора из перечисления ENUM_INDICATOR
   int              parameters_cnt=0,       // количество параметров
   const MqlParam&  parameters_array[]=NULL, // массив параметров
   );
```

В функции OnInit() индикатора MACD изменим код, где для получения хэндлов используем функцию IndicatorCreate.

```
// ExtFastMaHandle=iMA(NULL,0,InpFastEMA,0,MODE_EMA,InpAppliedPrice);
// ExtSlowMaHandle=iMA(NULL,0,InpSlowEMA,0,MODE_EMA,InpAppliedPrice);
// ExtFastMaHandle=iCustom(NULL,0,"Examples\\Custom Moving Average", InpFastEMA,0,MODE_EMA,InpAppliedPrice);
// ExtSlowMaHandle=iCustom(NULL,0,"Examples\\Custom Moving Average", InpSlowEMA,0,MODE_EMA,InpAppliedPrice);

MqlParam params[];
 ArrayResize(params,5);
 params[0].type         =TYPE_STRING;
 params[0].string_value="Examples\\Custom Moving Average";
//--- set ma_period
   params[1].type         =TYPE_INT;
   params[1].integer_value=InpFastEMA;
//--- set ma_shift
   params[2].type         =TYPE_INT;
   params[2].integer_value=0;
//--- set ma_method
   params[3].type         =TYPE_INT;
   params[3].integer_value=MODE_EMA;
//--- set applied_price
   params[4].type         =TYPE_INT;
   params[4].integer_value=InpAppliedPrice;

ExtFastMaHandle=IndicatorCreate(NULL,NULL,IND_CUSTOM,4,params);
params[1].integer_value=InpSlowEMA;
ExtSlowMaHandle=IndicatorCreate(NULL,NULL,IND_CUSTOM,4,params);

```

После компиляции индикатора мы опять увидим, что его отображение никак не изменилось.

После получения хэндла индикатора, если он используется в коде один раз, для экономии памяти неплохо использовать функцию IndicatorRelease.



IndicatorRelease

Удаляет хэндл индикатора и освобождает расчетную часть индикатора, если ею больше никто не пользуется.

```
bool  IndicatorRelease(
   int      indicator_handle     // handle индикатора
   );
```

Которая удаляет хэндл индикатора и освобождает расчетную часть индикатора.
Хорошо, хэндл индикатора мы получили. Как же теперь извлечь его данные?
Делается это в функции OnCalculate с помощью функции CopyBuffer.

Обращение по начальной позиции и количеству требуемых элементов

```
int  CopyBuffer(
   int       indicator_handle,     // handle индикатора
   int       buffer_num,           // номер буфера индикатора
   int       start_pos,            // откуда начнем
   int       count,                // сколько копируем
   double    buffer[]              // массив, куда будут скопированы данные
   );
```

Обращение по начальной дате и количеству требуемых элементов

```
int  CopyBuffer(
   int       indicator_handle,     // handle индикатора
   int       buffer_num,           // номер буфера индикатора
   datetime  start_time,           // с какой даты
   int       count,                // сколько копируем
   double    buffer[]              // массив, куда будут скопированы данные
   );
```

Обращение по начальной и конечной датам требуемого интервала времени

```
int  CopyBuffer(
   int       indicator_handle,     // handle индикатора
   int       buffer_num,           // номер буфера индикатора
   datetime  start_time,           // с какой даты
   datetime  stop_time,            // по какую дату
   double    buffer[]              // массив, куда будут скопированы данные
   );
```

При этом функция CopyBuffer() распределяет размер принимающего массива под размер копируемых данных.

Напомним, что это работает, если принимающий массив является просто динамическим массивом.

Если же принимающий массив связан с буфером индикатора, тогда клиентский терминал сам заботится о том, чтобы размер такого массива соответствовал количеству баров, доступных индикатору для расчета.

В индикаторе MACD именно такая ситуация.

```
35 //+------------------------------------------------------------------+
36 //| Custom indicator initialization function                         |
37 //+------------------------------------------------------------------+
38 void OnInit()
39   {
40 //--- indicator buffers mapping
41    SetIndexBuffer(0,ExtMacdBuffer,INDICATOR_DATA);
42    SetIndexBuffer(1,ExtSignalBuffer,INDICATOR_DATA);
43    SetIndexBuffer(2,ExtFastMaBuffer,INDICATOR_CALCULATIONS);
44    SetIndexBuffer(3,ExtSlowMaBuffer,INDICATOR_CALCULATIONS);
```

Промежуточные массивы ExtFastMaBuffer и ExtSlowMaBuffer привязаны к буферам индикатора с помощью функции SetIndexBuffer.

```
118 //--- get Fast EMA buffer
119    if(IsStopped()) return(0); //Checking for stop flag
120    if(CopyBuffer(ExtFastMaHandle,0,0,to_copy,ExtFastMaBuffer)<=0)
121      {
122       Print("Getting fast EMA is failed! Error",GetLastError());
123       return(0);
124      }
125 //--- get SlowSMA buffer
126    if(IsStopped()) return(0); //Checking for stop flag
127    if(CopyBuffer(ExtSlowMaHandle,0,0,to_copy,ExtSlowMaBuffer)<=0)
128      {
129       Print("Getting slow SMA is failed! Error",GetLastError());
130       return(0);
131      }
```

И в эти массивы производится копирование буфера индикатора Moving Average на основе его хэндлов с помощью функции CopyBuffer.

```
40 //--- indicator buffers mapping
41    SetIndexBuffer(0,ExtMacdBuffer,INDICATOR_DATA);
42    SetIndexBuffer(1,ExtSignalBuffer,INDICATOR_DATA);
43 //    SetIndexBuffer(2,ExtFastMaBuffer,INDICATOR_CALCULATIONS);
44 //    SetIndexBuffer(3,ExtSlowMaBuffer,INDICATOR_CALCULATIONS);
```

Время	Источник	Сообщение
2018.11.08 11:14:38.678	MACD (EURUSD,H1)	array out of range in 'MACD.mq5' (139,39)

Торговля | Активы | История | Новости 75 | Почта 6 | Календарь | Компания | Маркет | Алерты | Сигналы

```
120    if(CopyBuffer(ExtFastMaHandle,0,0,to_copy,ExtFastMaBuffer)<=0)
121      {
122       Print("Getting fast EMA is failed! Error",GetLastError());
123       return(0);
124      }
```

Если убрать привязку массивов ExtFastMaBuffer и ExtSlowMaBuffer к буферам индикатора, тогда клиентский терминал выдаст ошибку.

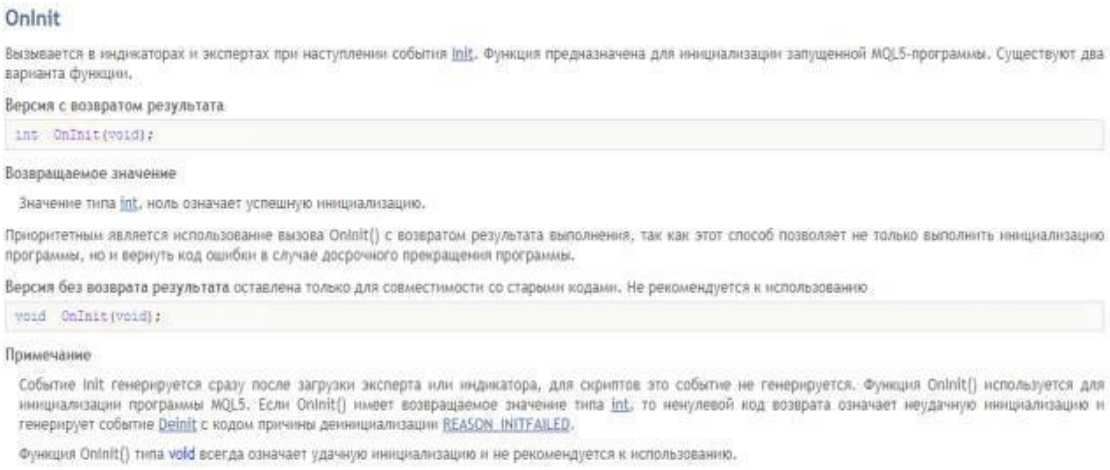
Происходит это потому, что при загрузке индикатора значение to_copy равно размеру ценовой истории, а дальше to_copy=1 и производится частичное копирование в массивы ExtFastMaBuffer и ExtSlowMaBuffer, при этом их размеры становятся равны 1.

В этом случае применением функции ArrayResize проблему не решить, так как функция CopyBuffer все равно будет уменьшать размер массива до 1.

Можно, конечно, использовать еще один массив-посредник, в который копировать один элемент. И уже из этого массива-посредника производить копирование в промежуточный массив, но проще всего, конечно, просто привязать промежуточный массив к буферу индикатора.

Функция OnInit

Как уже говорилось, функции OnInit(), OnDeinit(), OnCalculate() вызываются клиентским терминалом при наступлении определенных событий.



OnInit

Вызывается в индикаторах и экспертах при наступлении события Init. Функция предназначена для инициализации запущенной MQL5-программы. Существуют два варианта функции.

Версия с возвратом результата

```
int  OnInit(void);
```

Возвращаемое значение

Значение типа int, ноль означает успешную инициализацию.

Приоритетным является использование вызова OnInit() с возвратом результата выполнения, так как этот способ позволяет не только выполнить инициализацию программы, но и вернуть код ошибки в случае досрочного прекращения программы.

Версия без возврата результата оставлена только для совместимости со старыми кодами. Не рекомендуется к использованию

```
void  OnInit(void);
```

Примечание

Событие Init генерируется сразу после загрузки эксперта или индикатора, для скриптов это событие не генерируется. Функция OnInit() используется для инициализации программы MQL5. Если OnInit() имеет возвращаемое значение типа int, то ненулевой код возврата означает неудачную инициализацию и генерирует событие Deinit с кодом причины деинициализации REASON_INITFAILED.

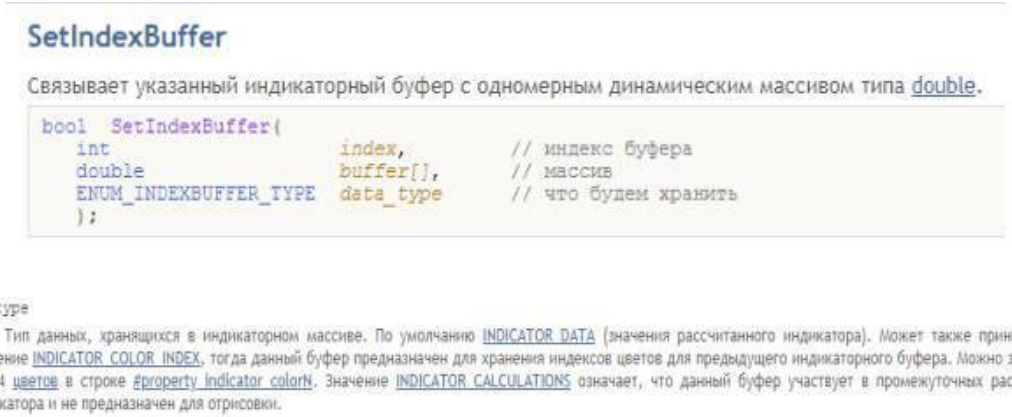
Функция OnInit() типа void всегда означает удачную инициализацию и не рекомендуется к использованию.

Функция OnInit() вызывается сразу после загрузки индикатора и соответственно используется для его инициализации.

Инициализация индикатора включает в себя привязку массивов к буферам индикатора, инициализацию глобальных переменных, включая инициализацию хэндлеров используемых индикаторов, а также программную установку свойств индикатора.

Давайте разберем некоторые из этих пунктов более подробно.

Как уже было показано, привязка массивов к буферам индикатора осуществляется с помощью функции SetIndexBuffer.



SetIndexBuffer

Связывает указанный индикаторный буфер с одномерным динамическим массивом типа double.

```
bool  SetIndexBuffer(
   int                  index,        // индекс буфера
   double               buffer[],     // массив
   ENUM_INDEXBUFFER_TYPE data_type    // что будем хранить
   );
```

data_type

[in] Тип данных, хранящихся в индикаторном массиве. По умолчанию INDICATOR_DATA (значения рассчитанного индикатора). Может также принимать значение INDICATOR_COLOR_INDEX, тогда данный буфер предназначен для хранения индексов цветов для предыдущего индикаторного буфера. Можно задать до 64 цветов в строке #property indicator_colorN. Значение INDICATOR_CALCULATIONS означает, что данный буфер участвует в промежуточных расчетах индикатора и не предназначен для отрисовки.

Где data_type может быть INDICATOR_DATA (данные индикатора для отрисовки, по умолчанию, можно не указывать), INDICATOR_COLOR_INDEX (цвет индикатора), INDICATOR_CALCULATIONS (буфер промежуточных расчетов индикатора).

После применения функции SetIndexBuffer к динамическому массиву, его размер автоматически поддерживается равным количеству баров, доступных индикатору для расчета.

Каждый индекс массива типа INDICATOR_COLOR_INDEX соответствует индексу массива типа INDICATOR_DATA, а значение индекса массива типа INDICATOR_COLOR_INDEX определяет цвет отображения индекса массива типа INDICATOR_DATA.

Значение индекса массива типа INDICATOR_COLOR_INDEX, при его установке, берется из свойства #property indicator_colorN как индекс цвета в строке.

Индекс буфера типа INDICATOR_COLOR_INDEX должен следовать за индексом буфера типа INDICATOR_DATA.

После привязки динамического массива к буферу индикатора можно поменять порядок доступа к массиву от конца к началу, т.е. значение массива с индексом 0 будет соответствовать последнему полученному значению индикатора. Сделать это можно с помощью функции ArraySetAsSeries.

ArraySetAsSeries

Устанавливает флаг AS_SERIES указанному объекту динамического массива, индексация элементов массива будет производиться как в таймсериях.

```
bool  ArraySetAsSeries(
   const void&  array[],    // массив по ссылке
   bool         flag        // true означает обратный порядок индексации
   );
```

Параметры

array[]

[in][out] Числовой массив для установки.

flag

[in] Направление индексирования массива.

При применении функции ArraySetAsSeries физическое хранение данных массива не меняется, в памяти, массив, как и прежде, хранится в порядке от первого значения до последнего значения.

Функция ArraySetAsSeries меняет лишь программный доступ к элементам массива – от последнего элемента массива к первому элементу массива.

В функции OnInit() также может осуществляться проверка входных параметров на корректность, так как пользователь может ввести все, что угодно.

При этом значение входного параметра переназначается с помощью глобальной переменной, и далее в расчетах используется уже значение глобальной переменной.

Например, для индикатора ADX это выглядит так:

```
47 void OnInit()
48   {
49 //--- check for input parameters
50    if(InpPeriodADX>=100 || InpPeriodADX<=0)
51      {
52       ExtADXPeriod=14;
53       printf("Incorrect value for input variable Period_ADX=%d. Indicator will use value=%d for calculations.",
54       InpPeriodADX,ExtADXPeriod);
55      }
56    else ExtADXPeriod=InpPeriodADX;
```

здесь ExtADXPeriod – глобальная переменная, а InpPeriodADX – входной параметр.

При использовании хэндлов индикатора, можно указывать символ (финансовый инструмент), для которого индикатор будет создаваться.

При этом такой символ может определяться пользователем.

В функции OnInit() также полезно проверить этот входной параметр на корректность.

```
35 //------------------------------------
36 input string symbol=" ";  // символ
37 string name=symbol;
```

```
54 //--- удалим пробелы слева и справа
55    StringTrimRight(name);
56    StringTrimLeft(name);
57 //--- если после этого длина строки name нулевая
58    if(StringLen(name)==0)
59      {
60       //--- возьмем символ с графика, на котором запущен индикатор
61       name=_Symbol;
62      }
63
64 ExtFastMaHandle=iMA(name,0,InpFastEMA,0,MODE_EMA,InpAppliedPrice);
65 ExtSlowMaHandle=iMA(name,0,InpSlowEMA,0,MODE_EMA,InpAppliedPrice);
```

Например, в коде индикатора MACD пусть определен входной параметр:

input string symbol=" ";

Объявим глобальную переменную:

string name=symbol;

И в функции OnInit() произведем проверку – удалим пробелы слева и справа с помощью функции StringTrimRight, и если после этого длина строки name нулевая, возьмем символ с графика, на котором запущен индикатор.

Пользовательские индикаторы

Группа функций, используемых при оформлении пользовательских индикаторов. Данные функции нельзя использовать при написании советников и скриптов.

Функция	Действие
SetIndexBuffer	Связывает указанный индикаторный буфер с одномерным динамическим массивом типа double
IndicatorSetDouble	Задает значение свойства индикатора, имеющего тип double
IndicatorSetInteger	Задает значение свойства индикатора,имеющего тип int
IndicatorSetString	Задает значение свойства индикатора, имеющего тип string
PlotIndexSetDouble	Задает значение свойства линии индикатора, имеющего тип double
PlotIndexSetInteger	Задает значение свойства линии индикатора, имеющего тип int
PlotIndexSetString	Задает значение свойства линии индикатора, имеющего тип string
PlotIndexGetInteger	Возвращает значение свойства линии индикатора, имеющего целый тип

Программная установка свойств индикатора осуществляется с помощью функций IndicatorSetDouble, IndicatorSetInteger, IndicatorSetString, PlotIndexSetDouble, PlotIndexSetInteger, PlotIndexSetString.

IndicatorSetDouble

Задаёт значение соответствующего свойства индикатора. Свойство индикатора должно быть типа double.

Вызов с указанием идентификатора свойства.

```
bool  IndicatorSetDouble(
   int     prop_id,           // идентификатор
   double  prop_value         // устанавливаемое значение
   );
```

Вызов с указанием идентификатора и модификатора свойства.

```
bool  IndicatorSetDouble(
   int     prop_id,           // идентификатор
   int     prop_modifier,     // модификатор
   double  prop_value         // устанавливаемое значение
   )
```

IndicatorSetDouble(INDICATOR_LEVELVALUE, 0, 50)

property indicator_level1 50

Функция IndicatorSetDouble позволяет программным способом определять такие свойства индикатора как indicator_minimum, indicator_maximum и indicator_levelN, например:

IndicatorSetDouble(INDICATOR_LEVELVALUE, 0, 50)

является аналогом:

property indicator_level1 50

IndicatorSetInteger

Задает значение соответствующего свойства индикатора. Свойство индикатора должно быть типа int или color.

Вызов с указанием идентификатора свойства.

```
bool  IndicatorSetInteger(
   int  prop_id,           // идентификатор
   int  prop_value         // устанавливаемое значение
   );
```

Вызов с указанием идентификатора и модификатора свойства.

```
bool  IndicatorSetInteger(
   int  prop_id,           // идентификатор
   int  prop_modifier,     // модификатор
   int  prop_value         // устанавливаемое значение
   )
```

Функция IndicatorSetInteger позволяет программным способом определять такие свойства индикатора как indicator_height, indicator_levelcolor, indicator_levelwidth, indicator_levelstyle.

При этом для уровней необходимо определить их количество, используя функцию IndicatorSetInteger. Например, для индикатора RSI это выглядит следующим образом.

```
//--- indicator settings
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
//#property indicator_level1 30
//#property indicator_level2 70
#property indicator_buffers 3
#property indicator_plots   1
#property indicator_type1   DRAW_LINE
#property indicator_color1  DodgerBlue

//#property indicator_levelcolor Red
//#property indicator_levelstyle STYLE_SOLID
//#property indicator_levelwidth 1
```

```
IndicatorSetInteger(INDICATOR_DIGITS,2);
```

```
IndicatorSetInteger(INDICATOR_LEVELS,2);
IndicatorSetDouble(INDICATOR_LEVELVALUE,0,30);
IndicatorSetDouble(INDICATOR_LEVELVALUE,1,70);
IndicatorSetInteger(INDICATOR_LEVELCOLOR,0,0xff0);
IndicatorSetInteger(INDICATOR_LEVELCOLOR,1,0xff0);
IndicatorSetInteger(INDICATOR_LEVELSTYLE,0,STYLE_SOLID);
IndicatorSetInteger(INDICATOR_LEVELSTYLE,1,STYLE_SOLID);
IndicatorSetInteger(INDICATOR_LEVELWIDTH,0,1);
IndicatorSetInteger(INDICATOR_LEVELWIDTH,1,1);
```

Свойства индикатора, связанные с уровнями, заменяем на код, используя функцию IndicatorSetInteger.

Функция IndicatorSetInteger также позволяет определить точность индикатора, например:

IndicatorSetInteger(INDICATOR_DIGITS,2);

В результате будут отображаться только два знака после запятой значения индикатора.

Для функции IndicatorSetString нет соответствующих ей свойств индикатора property.

IndicatorSetString

Задает значение соответствующего свойства индикатора. Свойство индикатора должно быть типа string.

Вызов с указанием идентификатора свойства.

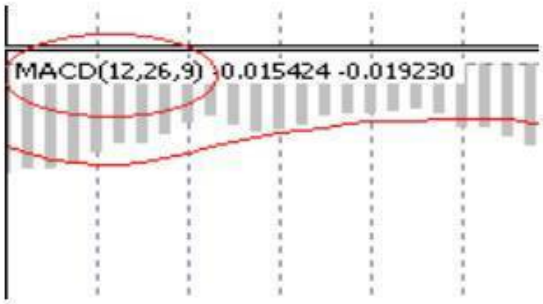
```
bool  IndicatorSetString(
   int     prop_id,          // идентификатор
   string  prop_value        // устанавливаемое значение
   );
```

Вызов с указанием идентификатора и модификатора свойства.

```
bool  IndicatorSetString(
   int     prop_id,          // идентификатор
   int     prop_modifier,    // модификатор
   string  prop_value        // устанавливаемое значение
   )
```

С помощью функции IndicatorSetString можно определить короткое наименование индикатора, например для индикатора MACD:

```
//--- name for Dindicator subwindow label
  IndicatorSetString(INDICATOR_SHORTNAME,"MACD("+string(InpFastEMA)+","+string(InpSlowEMA)+","+string(InpSignalSMA)+")");
```



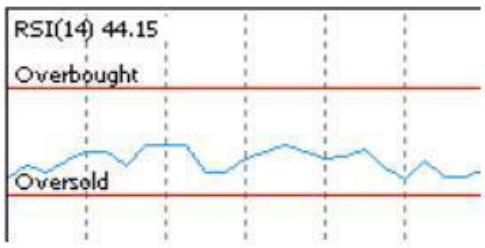
Это выглядит следующим образом.

И соответственно имя индикатора будет отображаться в окне индикатора как:

MACD (12, 26, 9)

Кроме того, функция IndicatorSetString позволяет установить подписи к уровням индикатора, например для индикатора RSI:

```
IndicatorSetString(INDICATOR_LEVELTEXT,
0,"Oversold");
IndicatorSetString(INDICATOR_LEVELTEXT,
1,"Overbought");
```



Можно отобразить подписи к уровням Oversold и Overbought.

PlotIndexSetDouble

Задает значение соответствующего свойства соответствующей линии индикатора. Свойство индикатора должно быть типа double.

```
bool  PlotIndexSetDouble(
   int     plot_index,     // индекс графического стиля
   int     prop_id,        // идентификатор свойства
   double  prop_value      // устанавливаемое значение
   );
```

PlotIndexSetDouble(индекс_построения,PLOT_EMPTY_VALUE,0);

С помощью функции PlotIndexSetDouble определяют, какое значение буфера индикатора является пустым и не участвует в отрисовке диаграммы индикатора.

Диаграмма индикатора рисуется от одного непустого значения до другого непустого значения индикаторного буфера, пустые значения пропускаются. Чтобы указать, какое значение следует считать "пустым", необходимо определить это значение в свойстве PLOT_EMPTY_VALUE. Например, если индикатор должен рисоваться по ненулевым значениям, то нужно задать нулевое значение в качестве пустого значения буфера индикатора:

PlotIndexSetDouble(индекс_построения,PLOT_EMPTY_VALUE,0);

PlotIndexSetInteger

Задает значение соответствующего свойства соответствующей линии индикатора. Свойство индикатора должно быть типа int, char, bool или color. варианта функции.

Вызов с указанием идентификатора свойства.

```
bool  PlotIndexSetInteger(
   int  plot_index,        // индекс графического стиля
   int  prop_id,           // идентификатор свойства
   int  prop_value         // устанавливаемое значение
   );
```

Вызов с указанием идентификатора и модификатора свойства.

```
bool  PlotIndexSetInteger(
   int  plot_index,        // индекс графического стиля
   int  prop_id,           // идентификатор свойства
   int  prop_modifier,     // модификатор свойства
   int  prop_value         // устанавливаемое значение
   )
```

Функция PlotIndexSetInteger позволяет программным способом, динамически, задавать такие свойства диаграммы индикатора, как код стрелки для стиля DRAW_ARROW, смещение стрелок по вертикали для стиля DRAW_ARROW, количество начальных баров без отрисовки и значений в Окне Данных, тип графического построения, признак отображения значе-

ний построения в Окне Данных, сдвиг графического построения индикатора по оси времени в барах, стиль линии отрисовки, толщина линии отрисовки, количество цветов, индекс буфера, содержащего цвет отрисовки.

Давайте разберем каждое из этих свойств по порядку на примере индикатора Custom Moving Average.

Конец ознакомительного фрагмента.