Язык программирования MQL5:

Продвинутое использование торговой платформы MetaTrader 5

Тимур Машнин

Тимур Машнин

Язык программирования MQL5: Продвинутое использование торговой платформы MetaTrader 5

http://www.litres.ru/pages/biblio_book/?art=19395560
ISBN 9785447499679

Аннотация

Создание пользовательских индикаторов и советников для торговой платформы MetaTrader 5 с использованием языка программирования MQL5.

Содержание

Введение 5
Общая структура индикатора 7
Свойства индикатора 9
Параметры ввода и переменные индикатора 30
Хэндл индикатора 38
Функции OnInit (), OnDeinit (), OnCalculate () 47
Функция OnInit () 48
Функция OnDeinit () 67
Функция OnCalculate () 69
Пример создания индикатора 84
Графические объекты 105
Функция PlaySound 124
Функция OnChartEvent 133
Конец ознакомительного фрагмента. 135

Язык программирования MQL5: Продвинутое использование торговой платформы MetaTrader 5

Тимур Машнин

Дизайнер обложки Тимур Машнин

© Тимур Машнин, 2020
© Тимур Машнин, дизайн обложки, 2020

ISBN 978-5-4474-9967-9
Создано в интеллектуальной издательской системе Ridero

Введение

Заказать робот или индикатор – пишите tmashnin@gmail.com.

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 позволять себе изредка только его цитирование.

Как сказано в предисловии к справочнику:

Программы, написанные на MetaQuotes Language 5, имеют различные свойства и предназначение

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.

Общая структура 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 ().

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

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

#property copyright «2009, MetaQuotes Software Corp.»

Далее указывается ссылка на сайт разработчика:

#property link http://www.mql5.com

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 индика-

тора 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.

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

Вернемся 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 парамет-

ра 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.

Откроем приложение MetaTrader 5 и в меню Tools (Сервис) выберем MetaQuotes Language Editor (Редактор MetaQuotes Language).

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

В функции OnInit () закомментируем строку:

```
// – — 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);

Теперь массив ExtTmpBuffer является просто динамическим массивом.

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

Терминал выдаст ошибку:

Time	Source	Message
2015.03.19 11:41:42.846	ADX (EURUSD,D1)	array out of range in 'ADX.mq5' (151,19)

Trade | Exposure | History | News 99 | Mailbox | Market | Alerts | Signals | Code Base | Experts | Journal |

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 [], close [].

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

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

ArrayResize (ExtTmpBuffer, rates_total);

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.

Аналогичная ситуация возникает когда значения таких промежуточных массивов заполняются с помощью функции CopyBuffer, которая 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.

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

SetIndexBuffer (4,ExtColorsBuffer, INDICATOR_COLOR_INDEX);

В свойстве #property indicator_color, раскрашиваемого

графического построения, указывается несколько цветов, например:

#property indicator_color1 Green, Red

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

В данном случае, это 0.0 и 1.0.

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

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

//Задаем количество индексов цветов для графического построения

PlotIndexSetInteger (0,PLOT_COLOR_INDEXES,2);

//Задаем цвет для каждого индекса

PlotIndexSetInteger (0,PLOT_LINE_COLOR,0,Blue); // Нулевой индекс цвета – синий цвет

PlotIndexSetInteger (0,PLOT_LINE_COLOR,1,Orange); // Первый индекс цвета – оранжевый цвет

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

Это идентично объявлению:

#property indicator_color1 Blue, Orange

Двинемся дальше по свойствам 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»

Другие свойства можно посмотреть в справочнике.

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

#property indicator_level1 0.0

#property indicator_levelcolor Red

#property indicator_levelstyle STYLE_SOLID

#property indicator_levelwidth 2

В редакторе MQL5, в окне Navigator (Навигатор), в разделе Indicators-> Examples откроем исходный код индикато-

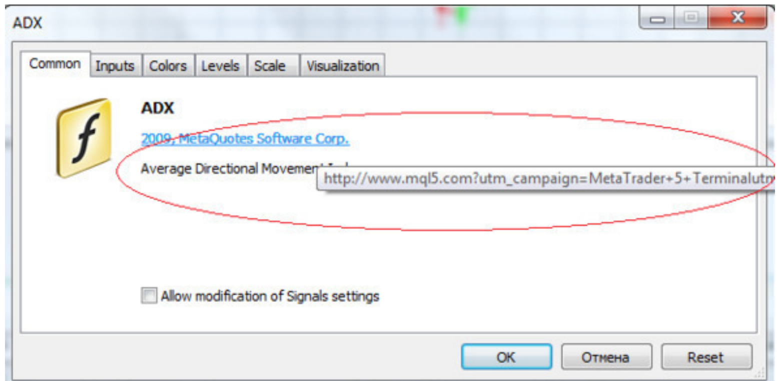
ра ADX.

Блок объявления свойств индикатора выглядит следующим образом:

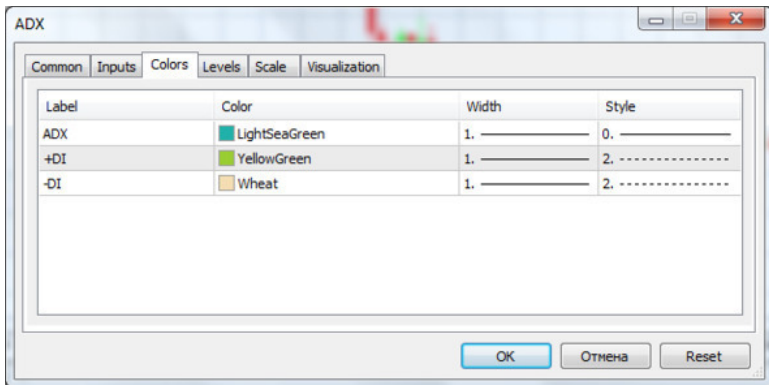
```
#property copyright «2009, MetaQuotes Software Corp.»
#property link "http://www.mql5.com"
#property description «Average Directional Movement Index»
#property indicator_separate_window
#property indicator_buffers 6
#property indicator_plots 3
#property indicator_type1 DRAW_LINE
#property indicator_color1 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»
```

Если мы в MetaTrader 5 попытаемся присоединить дан-

ный индикатор к графику, во-первых, откроется диалоговое окно индикатора, которое во вкладке Common отобразит значения свойств copyright, link и description:



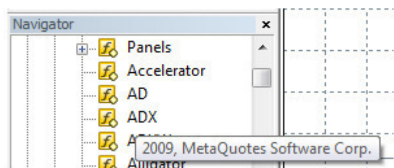
а во вкладке Colors отобразит значения свойств indicator_label, indicator_color, indicator_width, indicator_style:



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.



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) +»)»;

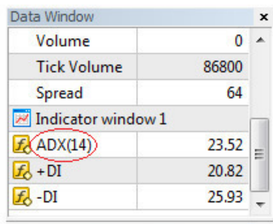
IndicatorSetString (INDICATOR_SHORTNAME, short_name);

изменена подпись индикатора:

А вызовом функции:

PlotIndexSetString (0,PLOT_LABEL, short_name);

изменена метка индикатора в окне Data Window:



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 Data Window.

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

#property indicator_buffers 6

#property indicator_plots 3

Сделано это для того, чтобы использовать три буфера ин-

дикатора для промежуточных расчетов:

SetIndexBuffer (0,ExtADXBuffer);

SetIndexBuffer (1,ExtPDIBuffer);

SetIndexBuffer (2,ExtNDIBuffer);

SetIndexBuffer (3,ExtPDBuffer, INDICATOR_CALCULATIONS);

SetIndexBuffer (4,ExtNDBuffer, INDICATOR_CALCULATIONS);

SetIndexBuffer (5,ExtTmpBuffer, INDICATOR_CALCULATIONS);

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, буфера индикатора для промежуточных вычислений здесь объявляются, так как заранее неизвестен размер загружаемой ценовой истории.

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 опуска-

ется 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.

В качестве первого шага изменим свойство indicator_type1:

#property indicator_type1 DRAW_COLOR_LINE

Далее увеличим на единицу значение свойства indicator_buffers:

#property indicator_buffers 7

Объявим массив для буфера цвета:

double ExtColorsBuffer [];

В функции OnInit () свяжем объявленный массив с буфером цвета:

SetIndexBuffer (0,ExtADXBuffer);

SetIndexBuffer (1,ExtColorsBuffer, INDICATOR_COLOR_INDEX);

SetIndexBuffer (2,ExtPDIBuffer);

SetIndexBuffer (3,ExtNDIBuffer);

SetIndexBuffer (4,ExtPDBuffer, INDICATOR_CALCULATIONS);

SetIndexBuffer (5,ExtNDBuffer,

INDICATOR_CALCULATIONS);

SetIndexBuffer (6,ExtTmpBuffer, INDICATOR_CALCULATIONS);

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.

В свойство indicator_color1 добавим цветов:

#property indicator_color1 LightSeaGreen, clrBlue, clrLightBlue, clrRed, clrLightPink

Увеличим толщину линии:

#property indicator_width1 2

В функции OnCalculate в конце перед закрывающей скобкой цикла for добавим код:

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;
}
```

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

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.

Для этого добавим свойства:

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

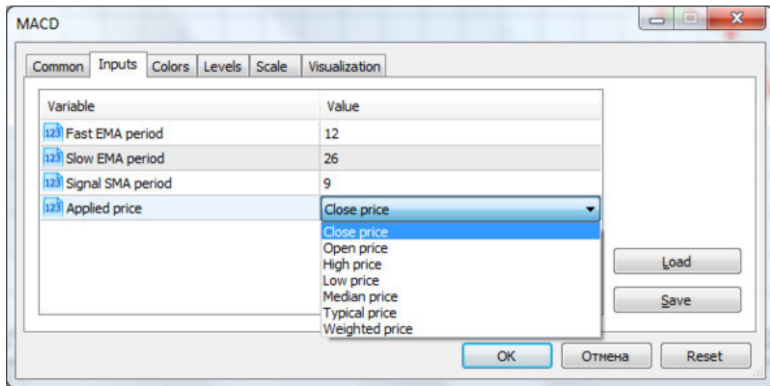
Теперь индикатор будет выглядеть следующим образом:

EURUSD,Daily
1.19150
1.17290
1.15430
1.13570
1.11710
1.09850
1.07990
1.06130
1.05151
RSI(14) 20.70
100.00
70.00
30.00
0.00
9 Jan 2015
19 Jan 2015
28 Jan 2015
6 Feb 2015
16 Feb 2015
25 Feb 2015
6 Mar 2015

Параметры ввода и переменные индикатора

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

Например, для индикатора MACD:



Тут 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.

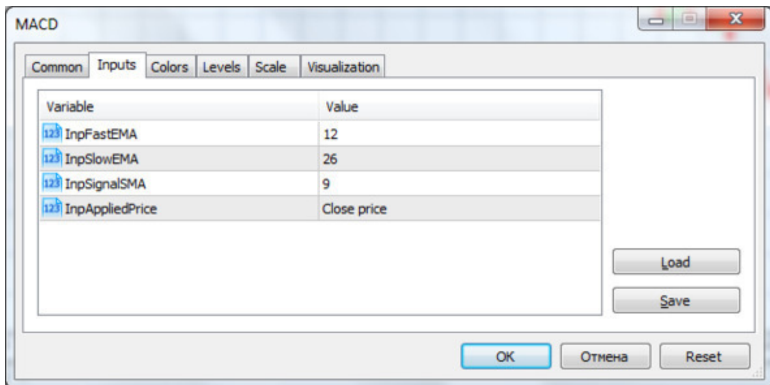
В коде индикатора такие параметры задаются Input переменными с модификатором input, который указывается перед типом данных. Как правило, Input переменные объявляются сразу после свойств индикатора.

Например, для индикатора MACD:

```
// – - 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
```

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.

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

В справочнике приводится соответствующий пример:

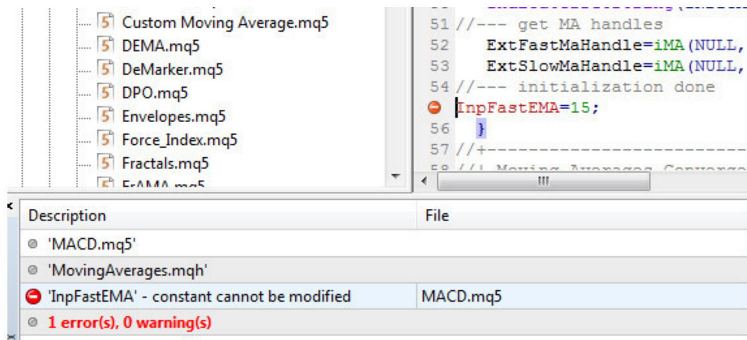
```
#property script_show_inputs
// – - day of week
enum dayOfWeek
{
```

```
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;
```

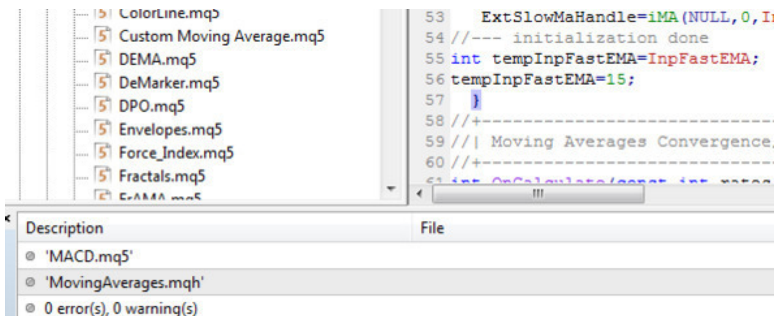
В этом примере команда #property script_show_inputs используется для скриптов, для индикаторов ее можно опустить.

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.

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.

Индикатор MACD имеет включаемый файл MovingAverages:

```
#include <MovingAverages.mqh>
```

расположенный в папке Include.

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

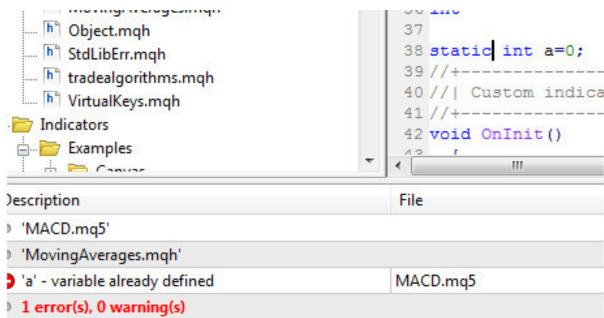
extern int a=0;

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;

то при компиляции обоих файлов возникнет ошибка:



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

#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;
```

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.

Стандартная функция для индикатора скользящего среднего это:

```
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
);
```

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);

где используются свойства индикатора:

// – - input parameters
input int InpFastEMA=12; // Fast EMA period
input int InpSlowEMA=26; // Slow EMA period
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE; // Applied price

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:

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:

// 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:

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:

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;
// – - initialization done
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:

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:

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
);

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. Промежуточные массивы ExtFastMaBuffer [] и ExtSlowMaBuffer [] привязаны к буферам индикатора:

SetIndexBuffer (2,ExtFastMaBuffer, INDICATOR_CALCULATIONS);

SetIndexBuffer (3,ExtSlowMaBuffer, INDICATOR_CALCULATIONS);

И в эти массивы производится копирование буфера индикатора Moving Average на основе его хэндлов:

if (CopyBuffer (ExtFastMaHandle,0,0,to_copy, ExtFastMaBuffer) <=0)
{
Print («Getting fast EMA is failed! Error», GetLastError ());
return (0);
}
if (CopyBuffer (ExtSlowMaHandle,0,0,to_copy,

```
ExtSlowMaBuffer) <=0)
   {
   Print («Getting slow SMA is failed! Error», GetLastError ());
   return (0);
   }
```

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

Time	Source	Message
2015.03.19 19:14:51.657	MACD (EURUSD,D1)	array out of range in 'MACD.mq5' (119,38)

Trade | Exposure | History | News 99 | Mailbox | Market | Alerts | Signals | Code Base | Experts | Journal |

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-посред-

ник, 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.

Функции OnInit (),
OnDeinit (), OnCalculate ()

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

Функция OnInit ()

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:

bool SetIndexBuffer (

int index, // индекс буфера

double buffer [], // массив

ENUM_INDEXBUFFER_TYPE data_type // что будем хранить

);

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

ных 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:

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

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:

```
// – - check for input parameters
if (InpPeriodADX> =100 || InpPeriodADX <=0)
{
ExtADXPeriod=14;
printf («Incorrect value for input variable Period_ADX=%d. Indicator will use value=%d for calculations.», InpPeriodADX, ExtADXPeriod);
}
else ExtADXPeriod=InpPeriodADX;
```

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

При использовании хэндлов индикатора, необходимо ука-

зывать 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.

Пусть определен входной параметр:

```
input string symbol=" "; // символ
```

Объявим глобальную переменную:

```
string name=symbol;
```

В функции OnInit () произведем проверку:

```
// – - удалим пробелы слева и справа
StringTrimRight (name);
StringTrimLeft (name);
// – - если после этого длина строки name нулевая
if (StringLen (name) ==0)
{
// – - возьмем символ с графика, на котором запущен индикатор
name=_Symbol;
}
```

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

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

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.

Свойства индикатора:

//#property indicator_level1 30

//#property indicator_level2 70

//#property indicator_levelcolor Red

//#property indicator_levelstyle STYLE_SOLID

//#property indicator_levelwidth 1

Заменяем на код:

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);

Функция IndicatorSetInteger также позволяет определить точность индикатора, например:

IndicatorSetInteger (INDICATOR_DIGITS,2);

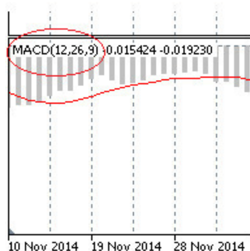
В результате будут отображаться только два знака после запятой значения индикатора.

Для функции IndicatorSetString нет соответствующих ей свойств индикатора.

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

IndicatorSetString (INDICATOR_SHORTNAME,«MACD (»+string (InpFastEMA) +»,»+string (InpSlowEMA) +»,»+string (InpSignalSMA) +»)»);

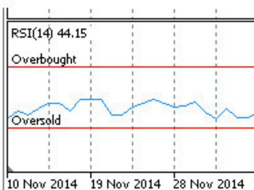
Соответственно имя индикатора будет отображаться в окне индикатора как:



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

IndicatorSetString (INDICATOR_LEVELTEXT, 0,«Oversold»);

IndicatorSetString (INDICATOR_LEVELTEXT, 1,«Overbought»)



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);

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 DataWindow, тип графического построения, признак отображения значений построения в окне DataWindow, 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.

Изменим свойство indicator_type1 индикатора Custom Moving Average:

#property indicator_type1 DRAW_ARROW

В функции OnInit () добавим вызов функции PlotIndexSetInteger, определяя различный код стрелки для стиля DRAW_ARROW:

PlotIndexSetInteger (0,PLOT_ARROW,2);

PlotIndexSetInteger (0,PLOT_ARROW,3);

PlotIndexSetInteger (0,PLOT_ARROW,4);

PlotIndexSetInteger (0,PLOT_ARROW,5);

PlotIndexSetInteger (0,PLOT_ARROW,6);

PlotIndexSetInteger (0,PLOT_ARROW,7);

PlotIndexSetInteger (0,PLOT_ARROW,8);

PlotIndexSetInteger (0,PLOT_ARROW,11);

PlotIndexSetInteger (0,PLOT_ARROW,12);

PlotIndexSetInteger (0,PLOT_ARROW,14);

PlotIndexSetInteger (0,PLOT_ARROW,15);

И так далее. Я думаю, этого будет достаточно для демонстрации этой опции.

В функции OnInit () добавим вызов функции PlotIndexSetInteger, определяя смещение стрелок по вертикали для стиля DRAW_ARROW:

PlotIndexSetInteger (0,PLOT_ARROW_SHIFT,0);

PlotIndexSetInteger (0,PLOT_ARROW_SHIFT,100)

В результате диаграмма индикатора сдвинулась вниз.

В индикаторе Custom Moving Average для определения количества начальных баров без отрисовки и значений в DataWindow используется вызов функции

PlotIndexSetInteger:

PlotIndexSetInteger (0,PLOT_DRAW_BEGIN, InpMAPeriod);

где InpMAPeriod – период скользящей средней.

Идентификатор свойства PLOT_DRAW_TYPE функции PlotIndexSetInteger позволяет программным способом задать свойство индикатора indicator_typeN, например:

PlotIndexSetInteger (0, PLOT_DRAW_TYPE, DRAW_ARROW);

Причем, если одновременно задано свойство indicator_typeN и сделан вызов функции PlotIndexSetInteger с идентификатором PLOT_DRAW_TYPE – действовать будет тип диаграммы, заданный функцией PlotIndexSetInteger.

Убрать отображение текущих значений диаграммы индикатора в окне DataWindow при наведении курсора мышки можно с помощью вызова функции PlotIndexSetInteger с идентификатором PLOT_SHOW_DATA:

PlotIndexSetInteger (0, PLOT_SHOW_DATA, false);

В индикаторе Custom Moving Average для определения сдвига графического построения индикатора по оси времени в барах используется вызов функции PlotIndexSetInteger:

PlotIndexSetInteger (0,PLOT_SHIFT, InpMAShift);

При InpMAShift=0:

При InpMAShift=10:

Такой сдвиг делается для имитации предсказательности индикатора.

Идентификатор свойства PLOT_LINE_STYLE функции PlotIndexSetInteger позволяет программным способом задать свойство индикатора indicator_styleN, стиль линии от-

рисовки, например:

PlotIndexSetInteger (0, PLOT_LINE_STYLE, STYLE_DASHDOT);

Идентификатор свойства PLOT_LINE_WIDTH функции PlotIndexSetInteger позволяет программным способом задать свойство индикатора indicator_widthN, толщину линии отрисовки, например:

PlotIndexSetInteger (0, PLOT_LINE_WIDTH, 2);

Программным способом задать свойство индикатора indicator_colorN позволяет вызов функции PlotIndexSetInteger с идентификаторами PLOT_COLOR_INDEXES и PLOT_LINE_COLOR, например:

#property indicator_color1 clrGreen, clrRed

Или

PlotIndexSetInteger (0,PLOT_COLOR_INDEXES,2);

PlotIndexSetInteger (0,PLOT_LINE_COLOR,0,clrGreen);

PlotIndexSetInteger (0,PLOT_LINE_COLOR,1,clrRed);

Функция PlotIndexSetString позволяет программным способом задать свойство индикатора indicator_labelN. Например, для индикатора MACD это будет выглядеть следующим образом:

#property indicator_label1 «MACD»

#property indicator_label2 «Signal»

Или

PlotIndexSetString (0, PLOT_LABEL, «MACD»);

PlotIndexSetString (1, PLOT_LABEL, «Signal»);

Рассмотренные выше функции программной установки свойств индикатора можно конечно вызывать и в функции обратного вызова OnCalculate, но глубокого смысла в этом нет, так как они не могут быть применены только к части диаграммы индикатора – они применяются сразу ко всей диаграмме индикатора. Поэтому для экономии ресурсов лучше всего вызывать эти функции в функции обратного вызова OnInit ().

Функция OnDeinit ()

Процитируем справочник:

Событие Deinit генерируется для экспертов и индикаторов в следующих случаях:

– перед переинициализацией в связи со сменой символа или периода графика, к которому прикреплена mql5-программа;

– перед переинициализацией в связи со сменой входных параметров;

– перед выгрузкой mql5-программы.

Так как функция OnDeinit () вызывается при деинициализации, то ее основное предназначение, это освобождение занимаемых ресурсов.

Под освобождением занимаемых ресурсов для индикатора подразумевается очищение графика символа от дополнительных графических объектов.

То есть помимо диаграммы индикатора, мы можем присоединять к графику символа различные объекты – линии, графические фигуры треугольник, прямоугольник и эллипс, знаки, подписи и др. Об этом мы поговорим позже.

Соответственно при деинициализации индикатора было бы неплохо все это убрать с графика символа.

Первым делом здесь используется функция Comment, которая выводит комментарий, определенный пользователем,

в левый верхний угол графика:

```
void Comment (
argument, // первое значение
…// последующие значения
);
```

Для очистки от комментариев используются пустые комментарии:

```
Comment (»»);
```

Далее используется функция ObjectDelete, которая удаляет объект с указанным именем с указанного графика:

```
bool ObjectDelete (
long chart_id, // chart identifier
string name // object name
);
```

Позже мы продемонстрируем применение этих функций.

Функция OnCalculate ()

Функция OnCalculate () вызывается клиентским терминалом при поступлении нового тика по символу, для которого рассчитывается индикатор.

Хотя функция OnCalculate () имеет два вида – для индикатора, который может быть рассчитан на основе только одной из ценовых таймсерий:

int OnCalculate (const int rates_total, // размер массива price []

const int prev_calculated, // обработано баров на предыдущем вызове

const int begin, // откуда начинаются значимые данные

const double& price [] // массив для расчета

);

и для индикатора, который рассчитывается с использованием нескольких ценовых таймсерий:

int OnCalculate (const int rates_total, // размер входных таймсерий

const int prev_calculated, // обработано баров на предыдущем вызове

const datetime& time [], // Time

const double& open [], // Open

const double& high [], // High

const double& low [], // Low

```
const double& close [], // Close
const long& tick_volume [], // Tick Volume
const long& volume [], // Real Volume
const int& spread [] // Spread
);
```

Здесь мы будем пользоваться полной версией функции OnCalculate () как наиболее гибкой и предоставляющей наибольшие возможности.

Единственное, что мы должны отметить об усеченной функции OnCalculate (), это то, что она имеет опцию использования в качестве массива price [] рассчитанного буфера другого индикатора.

Продемонстрируем это на примере индикатора MACD и индикатора Custom Moving Average, который использует как раз усеченную функцию OnCalculate ().

Присоединим сначала индикатор MACD к графику символа, а затем перетащим индикатор MA в окно индикатора MACD:

Затем опять перетащим индикатор Custom Moving Average в окно индикатора MACD, при этом снова откроется окно параметров индикатора Custom Moving Average:

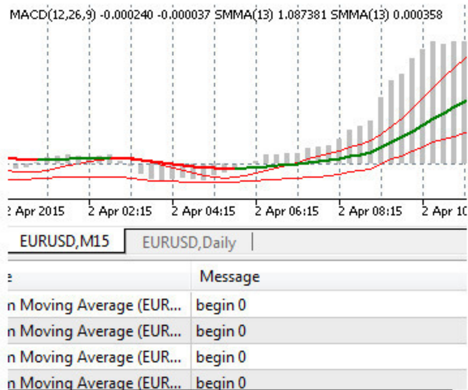
В списке выбора, что использовать в качестве массива price [], будут пункты First Indicator's Data и Previous Indicator's Data.

Здесь пункт First Indicator's Data означает, что в качестве массива price [] будет использоваться массив ExtMacdBuffer буфера индикатора MACD, а пункт Previous Indicator's Data означает, что в качестве массива price [] будет использоваться массив ExtLineBuffer буфера индикатора MA.

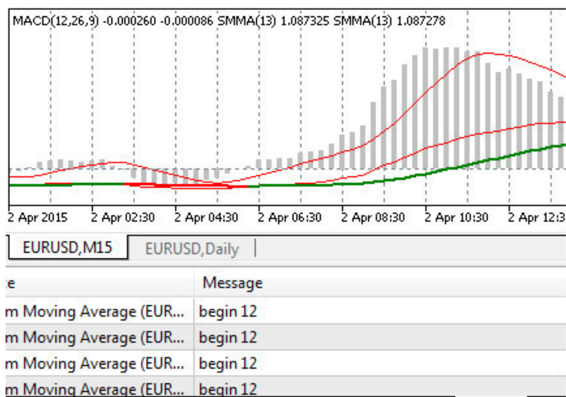
Если в функцию OnCalculate индикатора Custom Moving Average добавить:

Print («begin», begin);

То при выборе First Indicator's Data будет выводиться:



А при выборе Previous Indicator's Data будет выводиться:



В первом случае, begin=0, так как для буфера ExtMacdBuffer индикатора MACD функция

PlotIndexSetInteger с параметром PLOT_DRAW_BEGIN не вызывается. А во втором случае, begin=12, так как для буфера ExtLineBuffer индикатора MA вызывается функция PlotIndexSetInteger:

PlotIndexSetInteger (0,PLOT_DRAW_BEGIN, InpMAPeriod-1+begin);

Тут говорится о том, что массив буфера ExtLineBuffer индикатора MA заполняется, начиная с InpMAPeriod-1 бара, соответственно значение переменной begin функции OnCalculate индикатора Custom Moving Average будет также равно InpMAPeriod-1.

Вернемся к полной версии функции OnCalculate ().

Как правило, код функции OnCalculate () проектируется таким образом, чтобы при загрузке индикатора и первом вызове функции OnCalculate (), буфера индикатора были рассчитаны на основе всей загруженной ценовой истории, а при последующем поступлении нового тика и вызове функции OnCalculate (), рассчитывалось бы только одно новое значение, которое добавляется в конец массива буфера индикатора.

Но в начале кода функции OnCalculate () нужно конечно проверить, достаточный ли размер ценовой истории был загружен при загрузке индикатора.

Для этого проверяется значение переменной rates_total – размер входных таймсерий.

Как правило, в качестве порогового значения для

rates_total принимается значение периода индикатора, например для индикатора ADX:

```
// – - checking for bars count
if (rates_total <ExtADXPeriod)
return (0);
```

Если же в расчете буфера индикатора участвует хэндл другого индикатора, тогда проверяется количество рассчитанных данных для запрашиваемого индикатора:

```
// – - узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated (handle);
if (calculated <=0)
{
PrintFormat («BarsCalculated () вернул %d, код ошибки %d», calculated, GetLastError ());
return (0);
}
```

После проверки первоначальной загруженной истории для расчетов, вычисляется размер данных, которые необходимо рассчитать в этом вызове функции OnCalculate ().

В качестве примера, разберем блок кода, который приводится в справочнике, в разделе Технические индикаторы:

```
int OnCalculate (const int rates_total,
const int prev_calculated,
const datetime &time [],
const double &open [],
```

```
const double &high [],
const double &low [],
const double &close [],
const long &tick_volume [],
const long &volume [],
const int &spread [])
{
// – - количество копируемых значений из индикатора
int values_to_copy;
// – - узнаем количество рассчитанных значений в инди-
каторе
int calculated=BarsCalculated (handle);
if (calculated <=0)
{
PrintFormat («BarsCalculated () вернул %d, код ошибки
%d», calculated, GetLastError ());
return (0);
}
// – - если это первый запуск вычислений нашего индика-
тора или изменилось количество значений в индикаторе
// – - или если необходимо рассчитать индикатор для двух
или более баров (значит что-то изменилось в истории)
if (prev_calculated==0 || calculated!=bars_calculated ||
rates_total> prev_calculated+1)
{
// – - если массив больше, чем значений в индикаторе
```

на паре symbol/period, то копируем не все

// – - в противном случае копировать будем меньше, чем размер индикаторных буферов

if (calculated> rates_total) values_to_copy=rates_total;

else values_to_copy=calculated;

}

else

{

// – - значит наш индикатор рассчитывается не в первый раз и с момента последнего вызова OnCalculate ())

// – - для расчета добавилось не более одного бара

values_to_copy= (rates_total-prev_calculated) +1;

}

// – - запомним количество значений в индикаторе

bars_calculated=calculated;

// – - вернем значение prev_calculated для следующего вызова

return (rates_total);

}

Здесь переменная values_to_copy – количество рассчитываемых значений в вызове функции OnCalculate ().

Переменная prev_calculated – сколько было обработано баров функцией OnCalculate () при предыдущем вызове.

Таким образом, при загрузке индикатора prev_calculated=0, а при каждом следующем поступлении

нового тика prev_calculated= rates_total.

Переменная prev_calculated также обнуляется терминалом, если вдруг изменилось значение переменной rates_total.

Переменная bars_calculated – предыдущее количество рассчитанных данных для запрашиваемого индикатора, на основе которого рассчитывается данный индикатор.

Таким образом, первая проверка здесь:

prev_calculated==0 – индикатор только что загрузился или изменилась ценовая история.

calculated!=bars_calculated – изменилось количество рассчитанных данных для запрашиваемого индикатора.

rates_total> prev_calculated+1 – необходимо рассчитать индикатор для двух или более баров (значит, что-то изменилось в истории).

Последнее условие вступает в противоречие с утверждением справочника:

Если с момента последнего вызова функции OnCalculate () ценовые данные были изменены (подкачана более глубокая история или были заполнены пропуски истории), то значение входного параметра prev_calculated будет установлено в нулевое значение самим терминалом.

Если изменилась история, тогда сработает проверка prev_calculated==0 и проверка последнего условия будет излишней.

Теперь, если срабатывает первое или второе условие, тогда количество рассчитываемых значений – это размер вход-

ных таймсерий или количество рассчитанных данных для запрашиваемого индикатора, на основе которого рассчитывается данный индикатор, что из них меньше.

Если же первое или второе условие не срабатывают, тогда количество рассчитываемых значений:

values_to_copy= (rates_total-prev_calculated) +1;

Опять же, тут есть излишний код, так как, судя по справочнику, переменная prev_calculated может принимать значение либо 0, либо rates_total.

Поэтому, values_to_copy=1.

Таким образом, при поступлении нового тика, будет рассчитываться только одно значение индикатора для этого нового тика.

Рассмотрим другую реализацию вычисления размера данных, которые необходимо рассчитать в вызове функции OnCalculate ().

Для индикатора MACD это реализовано следующим образом:

```
// – - we can copy not all data
int to_copy;
if (prev_calculated> rates_total || prev_calculated <0) to_copy=rates_total;
else
{
to_copy=rates_total-prev_calculated;
if (prev_calculated> 0) to_copy++;
```

}

Опять же, судя по справочнику, здесь будет работать только код:

```
to_copy=rates_total-prev_calculated;
if (prev_calculated> 0) to_copy++;
```

Т.е. при загрузке индикатора to_copy=rates_total, а затем to_copy=1.

После вычисления размера данных, которые необходимо рассчитать в вызове функции OnCalculate (), производится их вычисление и заполнение ими буферов индикатора.

Если индикатор рассчитывается на основе хэндла другого индикатора, тогда производится копирование из буферов используемого индикатора в динамические массивы данного индикатора.

Вот как это реализовано для используемого индикатора ADX:

```
// – - заполняем часть массива ADXBuffer значениями из индикаторного буфера под индексом 0
if (CopyBuffer (ind_handle,0,0,amount, adx_values) <0)
{
// – - если копирование не удалось, сообщим код ошибки
PrintFormat («Не удалось скопировать данные из индикатора iADX, код ошибки %d», GetLastError ());
// – - завершим с нулевым результатом – это означает, что индикатор будет считаться нерассчитанным
```

return (false);

}

// – - заполняем часть массива DI_plusBuffer значениями из индикаторного буфера под индексом 1

if (CopyBuffer (ind_handle,1,0,amount, DIplus_values) <0)

{

// – - если копирование не удалось, сообщим код ошибки

PrintFormat («Не удалось скопировать данные из индикатора iADX, код ошибки %d», GetLastError ());

// – - завершим с нулевым результатом – это означает, что индикатор будет считаться нерассчитанным

return (false);

}

// – - заполняем часть массива DI_plusBuffer значениями из индикаторного буфера под индексом 2

if (CopyBuffer (ind_handle,2,0,amount, DIminus_values) <0)

{

// – - если копирование не удалось, сообщим код ошибки

PrintFormat («Не удалось скопировать данные из индикатора iADX, код ошибки %d», GetLastError ());

// – - завершим с нулевым результатом – это означает, что индикатор будет считаться нерассчитанным

return (false);

}

Здесь ind_handle – это хэндл индикатора ADX, второй параметр – индекс буфера используемого индикатора, из которого производится копирование, третий параметр – стартовая позиция, откуда начинается копирование. Здесь мы помним, что копирование идет от конца к началу, и поэтому нулевая стартовая позиция – это самые свежие данные. Четвертый параметр – это наш размер данных, которые необходимо рассчитать в вызове функции OnCalculate (), и последний параметр – это обычно динамический массив, привязанный к буферу индикатора, куда производится копирование.

Тут есть вопрос, как связать второй параметр функции CopyBuffer с индексом буфера используемого индикатора.

Это определяется вызовом функции SetIndexBuffer в используемом индикаторе. Например, для индикатора ADX:

SetIndexBuffer (0,ExtADXBuffer);

SetIndexBuffer (1,ExtPDIBuffer);

SetIndexBuffer (2,ExtNDIBuffer);

Отсюда нулевой индекс связан с буфером самого индикатора ADX, 1 индекс связан с буфером индикатора направленности +DI, 2 индекс связан с буфером индикатора направленности —DI.

Таким образом, для связывания второго параметра функции CopyBuffer с индексом буфера используемого индикатора, нужно знать код используемого индикатора.

Также для заполнения буфера индикатора значениями,

может использоваться цикл, например:

```
for (int i=start; i <rates_total &&!IsStopped ();i++)
{


}
```

Здесь start – это стартовая позиция, с которой начинается заполнение буфера индикатора.

При значении prev_calculated=0, значение start это, как правило, 0, при значении prev_calculated= rates_total, зачение start=prev_calculated-1.

Если же перед реализацией цикла с помощью функции ArraySetAsSeries поменять порядок доступа к массивам буферов индикатора и цен, тогда цикл примет вид:

```
for (int i=start; i> =0;i – ) {


}
```

Где start=rates_total-1, если prev_calculated=0, и start=0, если prev_calculated= rates_total.

Пример создания индикатора

В качестве примера рассмотрим создание индикатора, который будет реализовывать форекс стратегию «Impulse keeper» (Ловец импульсов) и показывать на графике сигналы на покупку и продажу.

В данной стратегии применяются четыре индикатора:

Экспоненциальная скользящая средняя с периодом 34 для цены High.

Экспоненциальная скользящая средняя с периодом 34 для цены Low.

Экспоненциальная скользящая средняя с периодом 125 для цены Close.

Parabolic SAR.

Сигналы на покупку и продажу в данной стратегии описываются следующим образом.

Сигнал на покупку: зеленая свеча закрывается выше EMA34 High и EMA34 Low, зеленая свеча выше EMA125 и Parabolic SAR.

Сигнал на продажу: красная свеча закрывается ниже EMA34 Low и EMA34 High, красная свеча ниже EMA125 и Parabolic SAR.

Давайте, реализуем эту стратегию в коде, который будет отображать на графике стрелки вверх и вниз сигналов на покупку и продажу.

Откроем MQL5 редактор и в меню File выберем New. В диалоговом окне MQL Wizard выберем Custom Indicator и нажмем кнопку Далее. Введем имя индикатора Impulse keeper, имя автора и ссылку и нажмем два раза Далее, а за-

тем Готово.

В результате мы получим код индикатора с пустыми функциями OnInit и OnCalculate.

Создание индикатора начнем с определения его свойств.

Количество буферов индикатора определим 8.

2 буфера – данные и цвет, для сигналов на покупку. 2 буфера – данные и цвет, для сигналов на продажу. И 4 буфера промежуточных вычислений для скопированных данных из индикаторов EMA34 Low, EMA34 High, EMA125 и Parabolic SAR:

```
#property indicator_buffers 8
```

Определим число графических построений – 2, одно построение для сигналов на покупку и другое построение для сигналов на продажу:

```
#property indicator_plots 2
```

Определим цвет и тип для обоих графических построений:

```
#property indicator_color1 clrGreen, clrBlack
#property indicator_type1 DRAW_COLOR_ARROW
#property indicator_color2 clrRed, clrBlack
#property indicator_type2 DRAW_COLOR_ARROW
```

Далее определим массивы буферов индикатора и хэндлы используемых индикаторов:

```
double IKBuyBuffer [];
double ColorIKBuyBuffer [];
double IKSellBuffer [];
```

```
double ColorIKSellBuffer [];
double EMA34HBuffer [];
double EMA34LBuffer [];
double EMA125Buffer [];
double PSARBuffer [];

int EMA34HHandle;
int EMA34LHandle;
int EMA125Handle;
int PSARHandle;
```

В функции OnInit () для первого графического построения определим тип стрелки – стрелка вверх, пустое значение и сдвиг:

```
int OnInit ()
{
PlotIndexSetInteger (0,PLOT_ARROW,233);
PlotIndexSetDouble (0,PLOT_EMPTY_VALUE,0);
PlotIndexSetInteger (0,PLOT_ARROW_SHIFT, -10);
```

Для второго графического построения определим тип стрелки – стрелка вниз, пустое значение и сдвиг:

```
PlotIndexSetInteger (1,PLOT_ARROW,234);
PlotIndexSetDouble (1,PLOT_EMPTY_VALUE,0);
PlotIndexSetInteger (1,PLOT_ARROW_SHIFT,10);
```

Свяжем массивы с буферами индикатора:

```
SetIndexBuffer (0,IKBuyBuffer, INDICATOR_DATA);
SetIndexBuffer                          (1,ColorIKBuyBuffer,
```

INDICATOR_COLOR_INDEX);

SetIndexBuffer (2,IKSellBuffer, INDICATOR_DATA);
SetIndexBuffer (3,ColorIKSellBuffer, INDICATOR_COLOR_INDEX);

SetIndexBuffer (4,EMA34HBuffer, INDICATOR_CALCULATIONS);
SetIndexBuffer (5,EMA34LBuffer, INDICATOR_CALCULATIONS);
SetIndexBuffer (6,EMA125Buffer, INDICATOR_CALCULATIONS);
SetIndexBuffer (7,PSARBuffer, INDICATOR_CALCULATIONS);

Получим хэндлы используемых индикаторов:

EMA34HHandle=iMA (NULL,0,34,0,MODE_EMA, PRICE_HIGH);
EMA34LHandle=iMA (NULL,0,34,0,MODE_EMA, PRICE_LOW);
EMA125Handle=iMA (NULL,0,125,0,MODE_EMA, PRICE_CLOSE);
PSARHandle=iSAR (NULL,0,0.02, 0.2);

В функции OnCalculate () произведем проверку размера доступной истории для расчета используемых индикаторов, определим количество копируемых значений используемых индикаторов и определим стартовую позицию расчета инди-

катора:

```
int values_to_copy;
int start;
int calculated=BarsCalculated (EMA34HHandle);
if (calculated <=0)
{
return (0);
}
if (prev_calculated==0 || calculated!=bars_calculated)
{
start=1;
if (calculated> rates_total) values_to_copy=rates_total;
else values_to_copy=calculated;
}
else
{
start=rates_total-1;
values_to_copy=1;
}
```

Переменную bars_calculated определим как глобальную int bars_calculated=0; в свойствах индикатора.

Далее произведем копирование из буферов используемых индикаторов в массивы буферов нашего индикатора:

```
if (!FillArrayFromMABuffer (EMA34HBuffer,0,EMA34HHandle, values_to_copy)) return (0); if (!FillArrayFromMABuffer
```

(EMA34LBuffer,0,EMA34LHandle, values_to_copy)) return (0); if (!FillArrayFromMABuffer (EMA125Buffer,0,EMA125Handle, values_to_copy)) return (0);

if (!FillArrayFromPSARBuffer (PSARBuffer, PSARHandle, values_to_copy)) return (0);

Здесь FillArrayFromMABuffer и FillArrayFromPSARBuffer – пользовательские функции, определенные вне функции OnCalculate ():

```
//+ – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – +
bool FillArrayFromPSARBuffer (
double &sar_buffer [], // индикаторный буфер значений Parabolic SAR
int ind_handle, // хэндл индикатора iSAR
int amount // количество копируемых значений
)
{
ResetLastError ();
if (CopyBuffer (ind_handle,0,0,amount, sar_buffer) <0)
{
return (false);
}
return (true);
}
//+ – — – — – — – — – — – — – — – — – — – — – — – — – — – — – —
```

— – — – — – — – — – +

```
bool FillArrayFromMABuffer (
double &values [], // индикаторный буфер значений Moving Average
int shift, // смещение
int ind_handle, // хэндл индикатора iMA
int amount // количество копируемых значений
)
{
ResetLastError ();
if (CopyBuffer (ind_handle,0, -shift, amount, values) <0)
{
return (false);
}
return (true);
}
```

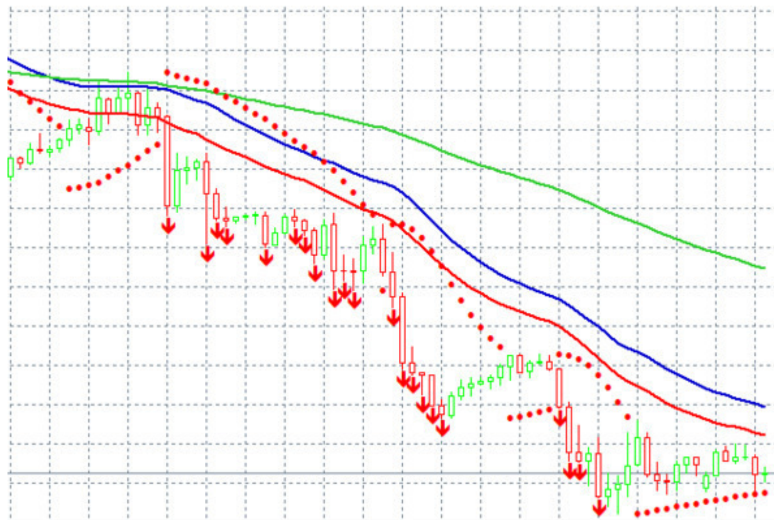
Далее в функции OnCalculate () заполним буфера индикатора данными и цветом:

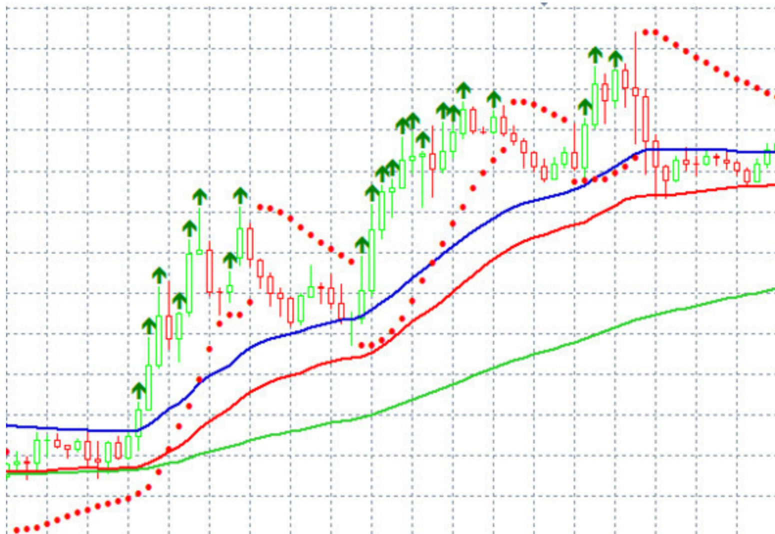
```
for (int i=start; i <rates_total &&!IsStopped ();i++)
{
IKBuyBuffer [i-1] =0;
ColorIKBuyBuffer [i-1] =1;

IKSellBuffer [i-1] =0;
ColorIKSellBuffer [i-1] =1;
```

```
if (close [i-1]> open [i-1] &&close [i-1]> EMA34HBuffer [i-1] &&close [i-1]> EMA34LBuffer [i-1] &&low [i-1]> EMA125Buffer [i-1] &&low [i-1]> PSARBuffer [i-1] &&EMA125Buffer [i-1] <EMA34LBuffer [i-1] &&EMA125Buffer [i-1] <EMA34HBuffer [i-1]) {
IKBuyBuffer [i-1] =high [i-1];
ColorIKBuyBuffer [i-1] =0;
}

if (close [i-1] <open [i-1] &&close [i-1] <EMA34HBuffer [i-1] &&close [i-1] <EMA34LBuffer [i-1] &&high [i-1] <EMA125Buffer [i-1] &&high [i-1] <PSARBuffer [i-1] &&EMA125Buffer [i-1]> EMA34LBuffer [i-1] &&EMA125Buffer [i-1]> EMA34HBuffer [i-1]) {
IKSellBuffer [i-1] =low [i-1];
ColorIKSellBuffer [i-1] =0;
}
}
bars_calculated=calculated;
// – - return value of prev_calculated for next call
return (rates_total);
}
```

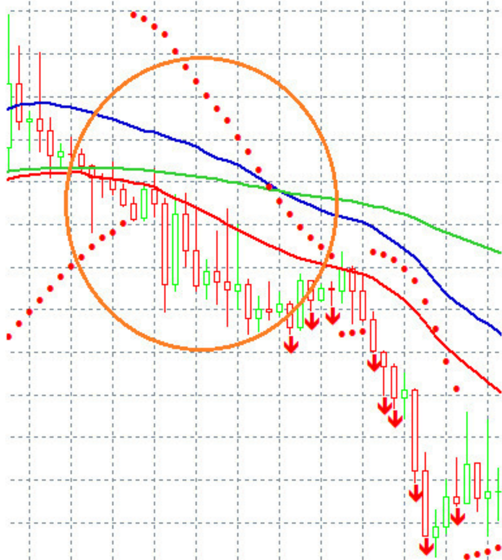
Здесь мы рассчитываем индикатор на предыдущем баре, так как на текущем баре цена close – это текущая цена тика.

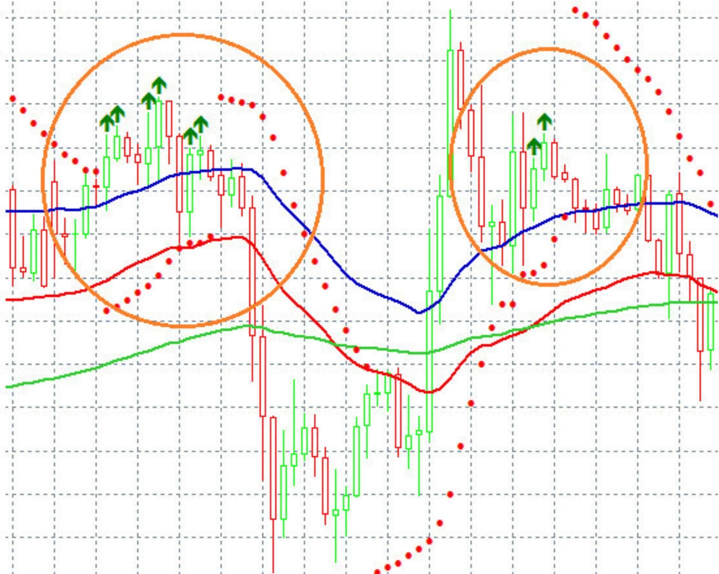
Откомпилируем код и присоединим индикатор к графику:





Мы увидим, что, в общем и целом, индикатор дает верные сигналы на продажу и покупку, хотя в некоторых случаях он запаздывает и дает ложные сигналы:



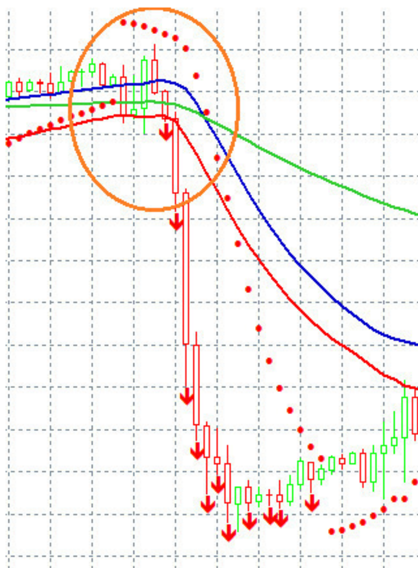


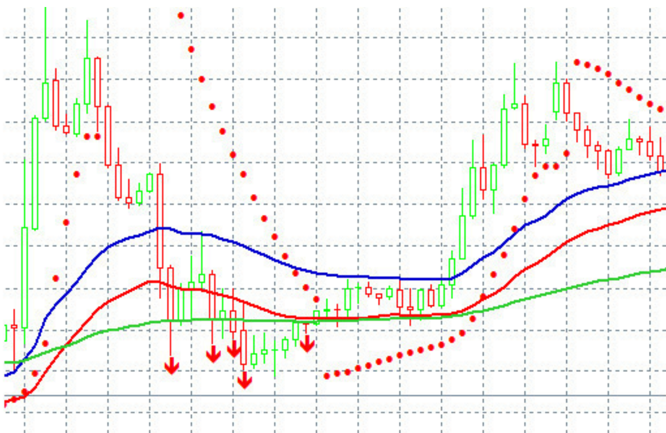
Как мы видим, происходит это из-за трендовой линии EMA125.

Попробуем отвязать ее от текущего периода и попробуем определять тренд, скажем по дневному графику:

EMA125Handle=iMA (NULL, PERIOD_D1,125,0,MODE_EMA, PRICE_CLOSE);

При этом запаздывание, конечно, сократится, но количество ложных сигналов увеличится:





Видимо для улучшения данной стратегии, нужно привлекать дополнительные индикаторы.

Попробуем сделать этот же самый индикатор, но не с помощью графических построений, а помещая графические объекты на график символа.

В свойствах индикатора теперь не нужно объявлять буфера данных и цвета индикатора и графические серии для них. Оставим только буфера индикатора для промежуточных расчетов и хэндлы используемых индикаторов:

```
#property indicator_chart_window
#property indicator_buffers 4
double EMA34HBuffer [];
double EMA34LBuffer [];
double EMA125Buffer [];
```

```
double PSARBuffer [];
int EMA34HHandle;
int EMA34LHandle;
int EMA125Handle;
int PSARHandle;
int bars_calculated=0;
```

В функции OnInit () соответственно оставим только привязку массивов к буферам промежуточных расчетов и получение хэндлов используемых индикаторов:

```
int OnInit ()
{
// – - indicator buffers mapping
SetIndexBuffer (0,EMA34HBuffer, INDICATOR_CALCULATIONS);
SetIndexBuffer (1,EMA34LBuffer, INDICATOR_CALCULATIONS);
SetIndexBuffer (2,EMA125Buffer, INDICATOR_CALCULATIONS);
SetIndexBuffer (3,PSARBuffer, INDICATOR_CALCULATIONS);

EMA34HHandle=iMA (NULL,0,34,0,MODE_EMA, PRICE_HIGH);
EMA34LHandle=iMA (NULL,0,34,0,MODE_EMA, PRICE_LOW);
EMA125Handle=iMA (NULL,0,125,0,MODE_EMA,
```

```
PRICE_CLOSE);
   PSARHandle=iSAR (NULL,0,0.02, 0.2);
   // – —
   return (INIT_SUCCEEDED);
   }
```

В функции OnCalculate () определим создание объектов на графике символа:

```
   int OnCalculate (const int rates_total,
   const int prev_calculated,
   const datetime &time [],
   const double &open [],
   const double &high [],
   const double &low [],
   const double &close [],
   const long &tick_volume [],
   const long &volume [],
   const int &spread [])
   {
   // – —
   int values_to_copy;
   int start;
   int calculated=BarsCalculated (EMA34HHandle);
   if (calculated <=0)
   {
   return (0);
   }
```

```
if (prev_calculated==0 || calculated!=bars_calculated)
{
start=1;
if (calculated> rates_total) values_to_copy=rates_total;
else values_to_copy=calculated;
}
else
{
start=rates_total-1;
values_to_copy=1;
} if (!FillArrayFromMABuffer (EMA34HBuffer,0,EMA34HHandle, values_to_copy)) return (0); if (!FillArrayFromMABuffer (EMA34LBuffer,0,EMA34LHandle, values_to_copy)) return (0); if (!FillArrayFromMABuffer (EMA125Buffer,0,EMA125Handle, values_to_copy)) return (0);
if (!FillArrayFromPSARBuffer (PSARBuffer, PSARHandle, values_to_copy)) return (0);

for (int i=start; i <rates_total &&!IsStopped ();i++)
{
if (close [i-1]> open [i-1] &&close [i-1]> EMA34HBuffer [i-1] &&close [i-1]> EMA34LBuffer [i-1] &&low [i-1]> EMA125Buffer [i-1] &&low [i-1]> PSARBuffer [i-1] &&EMA125Buffer [i-1] <EMA34LBuffer [i-1]
```

```
&&EMA125Buffer [i-1] <EMA34HBuffer [i-1]) {
   if (!ObjectCreate (0,«Buy»+i, OBJ_ARROW,0,time
[i-1],high [i-1]))
   {
   return (false);
   }
   ObjectSetInteger (0,«Buy»+i, OBJPROP_COLOR,
clrGreen);
   ObjectSetInteger (0,«Buy»+i,
OBJPROP_ARROWCODE,233);
   ObjectSetInteger (0,«Buy»+i, OBJPROP_WIDTH,2);
   ObjectSetInteger (0,«Buy»+i, OBJPROP_ANCHOR,
ANCHOR_UPPER);
   ObjectSetInteger (0,«Buy»+i, OBJPROP_HIDDEN, true);
   ObjectSetString (0,«Buy»+i,
OBJPROP_TOOLTIP,»»+close [i-1]);
   }
   if (close [i-1] <open [i-1] &&close [i-1] <EMA34HBuffer
[i-1] &&close [i-1] <EMA34LBuffer [i-1] &&high
[i-1] <EMA125Buffer [i-1] &&high [i-1] <PSARBuffer
[i-1] &&EMA125Buffer [i-1]> EMA34LBuffer [i-1]
&&EMA125Buffer [i-1]> EMA34HBuffer [i-1]) {
   if (!ObjectCreate (0,«Sell»+i, OBJ_ARROW,0,time
[i-1],low [i-1]))
   {
   return (false);
```

```
}
ObjectSetInteger (0,«Sell»+i, OBJPROP_COLOR, clrRed);
ObjectSetInteger (0,«Sell»+i, OBJPROP_ARROWCODE,234);
ObjectSetInteger (0,«Sell»+i, OBJPROP_WIDTH,2);
ObjectSetInteger (0,«Sell»+i, OBJPROP_ANCHOR, ANCHOR_LOWER);
ObjectSetInteger (0,«Sell»+i, OBJPROP_HIDDEN, true);
ObjectSetString (0,«Sell»+i, OBJPROP_TOOLTIP,»»+close [i-1]);
}
}
bars_calculated=calculated;
// – - return value of prev_calculated for next call
return (rates_total);
}
```

Здесь функцией ObjectCreate создаются объекты стрелка, привязанные ко времени и максимальной или минимальной цене.

Функцией ObjectSetInteger со свойством OBJPROP_COLOR определяется цвет стрелки.

Функцией ObjectSetInteger со свойством OBJPROP_ARROWCODE определяется направление стрелки вверх или вниз.

Функцией ObjectSetInteger со свойством OBJPROP_WIDTH определяется размер объекта.

Функцией ObjectSetInteger со свойством OBJPROP_ANCHOR определяется привязка к цене сверху или снизу по центру.

Функцией ObjectSetInteger со свойством OBJPROP_HIDDEN – true определяется отсутствие созданных объектов в списке объектов графика символа.

Функцией ObjectSetString со свойством OBJPROP_TOOLTIP определяется содержание всплывающей подсказки при наведении указателя на объект.

В функции OnDeinit () уберем все добавленные графические объекты:

```
void OnDeinit (const int reason) {
ObjectsDeleteAll (0, -1, -1);
}
```

Более подробно о создании объектов на графике символа мы поговорим далее.

Графические объекты

Как уже было показано ранее, мы можем рисовать на графике символа не только диаграммы индикатора, но и добавлять различные графические объекты с помощью функции ObjectCreate:

```
bool ObjectCreate (
long chart_id, // идентификатор графика
string name, // имя объекта
ENUM_OBJECT type, // тип объекта
int sub_window, // индекс окна
datetime time1, // время первой точки привязки
double price1, // цена первой точки привязки

datetime timeN=0, // время N-ой точки привязки
double priceN=0, // цена N-ой точки привязки

datetime time30=0, // время 30-й точки привязки
double price30=0 // цена 30-точки привязки
);
```

Здесь параметр sub_window это индекс главного окна графика символа со значением 0 или индекс подокна другого индикатора, присоединенного к графику символа.

Например, если в предыдущем примере мы изменим код, и присоединим к графику символа, скажем, индикатор ADX,

мы увидим следующее:

for (int i=start; i <rates_total &&!IsStopped ();i++)

{

if (close [i-1]> open [i-1] &&close [i-1]> EMA34HBuffer [i-1] &&close [i-1]> EMA34LBuffer [i-1] &&low [i-1]> EMA125Buffer [i-1] &&low [i-1]> PSARBuffer [i-1] &&EMA125Buffer [i-1] <EMA34LBuffer [i-1] &&EMA125Buffer [i-1] <EMA34HBuffer [i-1]) {

if (!ObjectCreate (0,«Buy»+i, OBJ_ARROW,0,time [i-1],high [i-1]))

{

return (false);

}

if (!ObjectCreate (0,«Buy1»+i, OBJ_ARROW,1,time [i-1],high [i-1]))

{

return (false);

}

ObjectSetInteger (0,«Buy»+i, OBJPROP_COLOR, clrGreen);

ObjectSetInteger (0,«Buy»+i, OBJPROP_ARROWCODE,233);

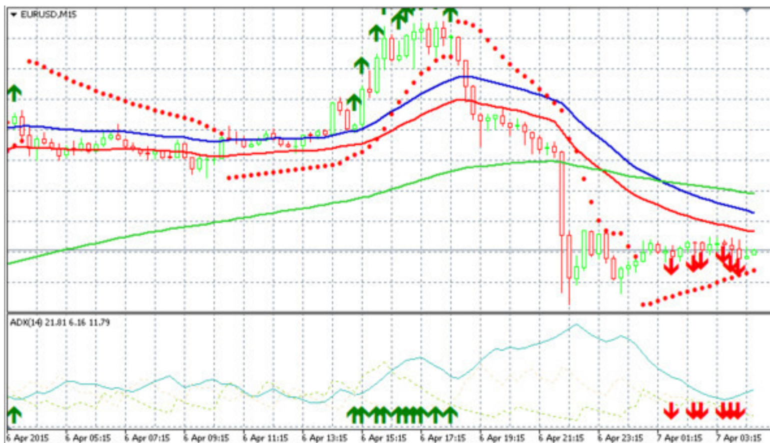
ObjectSetInteger (0,«Buy»+i, OBJPROP_WIDTH,2);

ObjectSetInteger (0,«Buy»+i, OBJPROP_ANCHOR, ANCHOR_UPPER);

ObjectSetInteger (0,«Buy»+i, OBJPROP_HIDDEN, true);

```
ObjectSetString (0,«Buy»+i, OBJPROP_TOOLTIP, close [i-1]);

ObjectSetInteger (0,«Buy1»+i, OBJPROP_COLOR, clrGreen);
ObjectSetInteger (0,«Buy1»+i, OBJPROP_ARROWCODE,233);
ObjectSetInteger (0,«Buy1»+i, OBJPROP_WIDTH,2);
ObjectSetInteger (0,«Buy1»+i, OBJPROP_ANCHOR, ANCHOR_UPPER);
ObjectSetInteger (0,«Buy1»+i, OBJPROP_HIDDEN, true);
ObjectSetString (0,«Buy1»+i, OBJPROP_TOOLTIP, close [i-1]);
}
if (close [i-1] <open [i-1] &&close [i-1] <EMA34HBuffer [i-1] &&close [i-1] <EMA34LBuffer [i-1] &&high [i-1] <EMA125Buffer [i-1] &&high [i-1] <PSARBuffer [i-1] &&EMA125Buffer [i-1]> EMA34LBuffer [i-1] &&EMA125Buffer [i-1]> EMA34HBuffer [i-1]) {
if (!ObjectCreate (0,«Sell»+i, OBJ_ARROW,0,time [i-1],low [i-1]))
{
return (false);
}
if (!ObjectCreate (0,«Sell1»+i, OBJ_ARROW,1,time
```

```
[i-1],low [i-1]))
   {
   return (false);
   }
   ObjectSetInteger (0,«Sell»+i, OBJPROP_COLOR, clrRed);
   ObjectSetInteger (0,«Sell»+i,
OBJPROP_ARROWCODE,234);
   ObjectSetInteger (0,«Sell»+i, OBJPROP_WIDTH,2);
   ObjectSetInteger (0,«Sell»+i, OBJPROP_ANCHOR,
ANCHOR_LOWER);
   ObjectSetInteger (0,«Sell»+i, OBJPROP_HIDDEN, true);
   ObjectSetString (0,«Sell»+i, OBJPROP_TOOLTIP, close
[i-1]);

   ObjectSetInteger (0,«Sell1»+i, OBJPROP_COLOR,
clrRed);
   ObjectSetInteger (0,«Sell1»+i,
OBJPROP_ARROWCODE,234);
   ObjectSetInteger (0,«Sell1»+i, OBJPROP_WIDTH,2);
   ObjectSetInteger (0,«Sell1»+i, OBJPROP_ANCHOR,
ANCHOR_LOWER)
   ObjectSetInteger (0,«Sell1»+i, OBJPROP_HIDDEN, true);
   ObjectSetString (0,«Sell1»+i, OBJPROP_TOOLTIP, close
[i-1]);
    }
    }
```



Нумерация подокон идет сверху вниз в порядке отображения.

Тип отображаемого объекта задается перечислением ENUM_OBJECT, которое можно посмотреть в справочнике.

После добавления графических объектов, не забываем их удалять в функции обратного вызова OnDeinit (), используя функцию ObjectDelete:

```
bool ObjectDelete (
long chart_id, // chart identifier
string name // object name
);
```

Или используя функцию ObjectsDeleteAll:

```
int ObjectsDeleteAll (
```

```
long chart_id, // chart identifier
int sub_window=-1, // window index
int type=-1 // object type
);
```

Помимо вышеупомянутых функций ObjectCreate, ObjectDelete и ObjectsDeleteAll, MQL5 предлагает набор функций для работы с графическими объектами: ObjectName, ObjectFind, ObjectGetTimeByValue, ObjectGetValueByTime, ObjectMove, ObjectsTotal, ObjectGetDouble, ObjectGetInteger, ObjectGetString, ObjectSetDouble, ObjectSetInteger, ObjectSetString, TextSetFont, TextOut, TextGetSize.

Функции ObjectName, ObjectFind, ObjectGetTimeByValue, ObjectGetValueByTime, ObjectsTotal, ObjectGetDouble, ObjectGetInteger, ObjectGetString, TextGetSize – это функции возвращающие информацию.

Функции ObjectSetDouble, ObjectSetInteger, ObjectSetString, TextSetFont – это функции устанавливающие свойства объекта.

Функция ObjectMove перемещает объект в окне.

Функция TextOut выводит текст в пиксельный массив для отображения объектом OBJ_BITMAP_LABEL или OBJ_BITMAP.

После добавления графических объектов рекомендуется принудительно перерисовать график символа с помощью

функции ChartRedraw:

```
void ChartRedraw (
long chart_id=0 // идентификатор графика
);
```

Функция ObjectCreate позволяет создавать программным способом те графические объекты, которые вы можете вручную нарисовать на графике символа, пользуясь панелью инструментов клиентского терминала.

С помощью функции ObjectSetDouble устанавливаются такие свойства графического объекта, как OBJPROP_PRICE – изменение параметра price функции ObjectCreate, OBJPROP_LEVELVALUE – определение уровней для таких объектов, как инструменты Фиббоначи и Вилы Эндрюса, OBJPROP_SCALE – определение масштаба для таких объектов, как инструменты Ганна и Дуги Фибоначчи, OBJPROP_ANGLE – определение угла объекта, т.е. возможность повернуть объект, который изначально не имеет жесткой привязки, например, повернуть текст, OBJPROP_DEVIATION – определение отклонения для объекта Канал стандартного отклонения.

Пример использования OBJPROP_PRICE:

```
int OnCalculate (const int rates_total,
const int prev_calculated,
const datetime &time [],
const double &open [],
const double &high [],
```

```
const double &low [],
const double &close [],
const long &tick_volume [],
const long &volume [],
const int &spread [])
{
// – —
ArraySetAsSeries (time, true);
ArraySetAsSeries (high, true);
ArraySetAsSeries (low, true);
ArraySetAsSeries (close, true);
ObjectDelete (0,«Price»);
if (!ObjectCreate (0,«Price», OBJ_HLINE,0,time [1],close
[1]))
{
return (false);
}
ObjectSetInteger (0,«Price», OBJPROP_COLOR,
clrGreen);
ObjectSetInteger (0,«Price», OBJPROP_WIDTH,1);
ObjectSetString (0,«Price», OBJPROP_TOOLTIP, close
[1]);
if (open [1]> close [1])
ObjectSetDouble (0,«Price», OBJPROP_PRICE, low [1]);
if (open [1] <close [1])
ObjectSetDouble (0,«Price», OBJPROP_PRICE, high [1]);
```

```
// – - return value of prev_calculated for next call
return (rates_total);
}
//+ – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – +
void OnDeinit (const int reason) {
ObjectsDeleteAll (0, -1, -1);
}
```

В этом коде создается горизонтальный уровень, показывающий минимальную или максимальную цену предыдущего бара, в зависимости от того, является ли этот бар бычьим или медвежьим.

Пример использования OBJPROP_ANGLE:

```
int OnCalculate (const int rates_total,
const int prev_calculated,
const datetime &time [],
const double &open [],
const double &high [],
const double &low [],
const double &close [],
const long &tick_volume [],
const long &volume [],
const int &spread [])
{
// – —
ArraySetAsSeries (time, true);
```

```
ArraySetAsSeries (high, true);
ArraySetAsSeries (low, true);
ArraySetAsSeries (close, true);
ObjectDelete (0,«Line»);
ObjectDelete (0,«Price»);
if (!ObjectCreate (0,«Line», OBJ_VLINE,0,time [1],close
[1]))
{
return (false);
}
ObjectSetInteger (0,«Line», OBJPROP_COLOR, clrBlue);
ObjectSetInteger (0,«Line», OBJPROP_WIDTH,1);
ObjectSetString (0,«Line», OBJPROP_TOOLTIP, close
[1]);

if (!ObjectCreate (0,«Price», OBJ_TEXT,0,time [3],high
[1]))
{
return (false);
}
ObjectSetString (0,«Price», OBJPROP_TEXT, close [1]);
ObjectSetInteger (0,«Price», OBJPROP_COLOR, clrBlack);
ObjectSetDouble (0,«Price», OBJPROP_ANGLE,90);
ObjectSetString (0,«Price», OBJPROP_TOOLTIP, close
[1]);
```

```
// – - return value of prev_calculated for next call
return (rates_total);
}
//+ – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – +
void OnDeinit (const int reason) {
ObjectsDeleteAll (0, -1, -1);
}
```

Этот код создает вертикальную линию с подписью цены закрытия предыдущего бара.

С помощью функции ObjectSetInteger устанавливаются такие свойства графического объекта, как цвет, стиль, размер и др.

С помощью функции ObjectSetString можно изменить имя объекта, при этом объект со старым именем будет удален и будет создан объект с новым именем, установить текст для таких объектов, как текст, кнопка, метка, поле ввода, событие, установить текст всплывающей подсказки для объекта, описание уровня для объектов, имеющих уровни, шрифт, имя BMP-файла для объекта «Графическая метка» и «Рисунок», символ для объекта «График».

Функция TextSetFont позволяет установить тип шрифта текста, его размер, стиль и угол наклона для объектов, содержащих текст.

Как уже было сказано, функция TextOut позволяет скомбинировать текст и изображение. Например, следующий код

выводит текст в изображение, залитое одним цветом:

```
uint ExtImg [10000];
//+ – — – — – — – — – — – — – — – — – — – — – — – — –
— – — – — – — – — – — – +
//| Custom indicator initialization function |
//+ – — – — – — – — – — – — – — – — – — – — – — – — –
— – — – — – — – — – — – +
int OnInit ()
{
ObjectCreate (0,«Image», OBJ_BITMAP_LABEL,0,0,0);
ObjectSetString (0,«Image», OBJPROP_BMPFILE,»::
IMG»);
ArrayFill (ExtImg,0,10000,0xffffff);
TextOut («Text», 10,10,TA_LEFT|TA_TOP,
ExtImg,100,100,0x000000,COLOR_FORMAT_XRGB_NOAL
ResourceCreate (»:: IMG»,
ExtImg,100,100,0,0,0,COLOR_FORMAT_XRGB_NOALPHA
ChartRedraw ();
// – —
return (INIT_SUCCEEDED);
}
```

Здесь ExtImg это пиксельный массив, представляющий изображение 100x100 пикселей.

Функция ObjectCreate создает объект «Графическая метка», а функция ObjectSetString устанавливает для этого объекта файл изображения с именем::IMG. По поводу знака «::»

справочник говорит следующее:

Для использования своего ресурса в коде нужно перед именем ресурса добавлять специальный признак "::».

Функция ArrayFill заполняет пиксельный массив пикселями белого цвета.

Функция TextOut выводит в пиксельный массив слово «Text».

Функция ResourceCreate создает из пиксельного массива ресурс с именем::IMG.

В итоге на белом фоне отображается надпись «Text».

Также можно вывести текст на готовое изображение:

```
#resource "\\Images\\image.bmp»
uint ExtImg [10000];
//+ – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – +
//| Custom indicator initialization function |
//+ – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – +
int OnInit ()
{
ObjectCreate (0,«Image», OBJ_BITMAP_LABEL,0,0,0);
ObjectSetString (0,«Image», OBJPROP_BMPFILE,»::
IMG»);
uint width=100;
uint height=100;
ResourceReadImage("::Images\\image.bmp», ExtImg,
```

```
width, height);
   TextOut («Text», 10,10,TA_LEFT|TA_TOP,
ExtImg,100,100,0xffffff,
COLOR_FORMAT_XRGB_NOALPHA);
   ResourceCreate (»:: IMG»,
ExtImg,100,100,0,0,0,COLOR_FORMAT_XRGB_NOALPHA
   ChartRedraw ();
   // – —
   return (INIT_SUCCEEDED);
   }
```

Здесь функция ResourceReadImage считывает существующее изображение из папки Images окна Navigator редактора MQL5 в пиксельный массив::IMG, связанный с объектом «Графическая метка», а функция TextOut выводит в пиксельный массив слово «Text».

То же самое можно проделать и с объектом «Рисунок»:

```
   #resource "\\Images\\image.bmp»
   uint ExtImg [10000];
   //+ – — – — – — – — – — – — – — – — – — – — – — –
— – — – — – — – — – +
   //| Custom indicator initialization function |
   //+ – — – — – — – — – — – — – — – — – — – — – — –
— – — – — – — – — – +
   int OnInit ()
   {
   // – —
```

```
return (INIT_SUCCEEDED);
}
//+ – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – +
//| Custom indicator iteration function |
//+ – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – +
int OnCalculate (const int rates_total,
const int prev_calculated,
const datetime &time [],
const double &open [],
const double &high [],
const double &low [],
const double &close [],
const long &tick_volume [],
const long &volume [],
const int &spread [])
{
// – —
ArraySetAsSeries (time, true);
ArraySetAsSeries (high, true);
ArraySetAsSeries (low, true);
ArraySetAsSeries (close, true);
ObjectDelete (0,«Image»);
ObjectCreate (0,«Image», OBJ_BITMAP,0,time [1],close [1]);
```

```
ObjectSetString (0,«Image», OBJPROP_BMPFILE,»::IMG»);
uint width=100;
uint height=100;
ResourceReadImage("::Images\\image.bmp», ExtImg, width, height);
TextOut («Text», 10,10,TA_LEFT|TA_TOP, ExtImg,100,100,0xffffff, COLOR_FORMAT_XRGB_NOALPHA);
ResourceCreate (»:: IMG», ExtImg,100,100,0,0,0,COLOR_FORMAT_XRGB_NOALPHA
ChartRedraw ();
// – - return value of prev_calculated for next call
return (rates_total);
}
//+ – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – +
void OnDeinit (const int reason) {
ObjectsDeleteAll (0, -1, -1);
}
```

В качестве примера использования графических объектов, рассмотрим создание индикатора, который выводит в небольшое окно на графике символа тот же график, но с другим временным периодом.

Для этого используем графический объект OBJ_CHART.

В качестве входных параметров индикатора используем

символ графика и его период:

```
#property indicator_chart_window

input string InpSymbol=«EURUSD»; // Символ
input ENUM_TIMEFRAMES InpPeriod=PERIOD_CURRENT; // Период
```

В функции OnInit () создадим графический объект График:

```
int OnInit ()
{
if (!ObjectCreate (0,«Chart», OBJ_CHART,0,0,0))
{
return (false);
}
```

По умолчанию точка привязки этого объекта – левый верхний угол графика.

Определим отступ точки привязки объекта, его размеры, символ и период графика, отображение шкалы времени, размер точки привязки, с помощью которой можно перемещать объект, отображение ценовой шкалы, режим перемещения мышкой, цвет рамки графика:

```
ObjectSetInteger (0,«Chart», OBJPROP_XDISTANCE,10);
ObjectSetInteger (0,«Chart», OBJPROP_YDISTANCE,20);
ObjectSetInteger (0,«Chart», OBJPROP_XSIZE,300);
ObjectSetInteger (0,«Chart», OBJPROP_YSIZE,200);
ObjectSetString (0,«Chart», OBJPROP_SYMBOL,
```

InpSymbol);

ObjectSetInteger (0,«Chart», OBJPROP_PERIOD, InpPeriod);

ObjectSetInteger (0,«Chart», OBJPROP_DATE_SCALE, true);

ObjectSetInteger (0,«Chart», OBJPROP_WIDTH,1);

ObjectSetInteger (0,«Chart», OBJPROP_PRICE_SCALE, true);

ObjectSetInteger (0,«Chart», OBJPROP_SELECTABLE, true);

ObjectSetInteger (0,«Chart», OBJPROP_SELECTED, true);

ObjectSetInteger (0,«Chart», OBJPROP_COLOR, clrBlue);

С помощью свойства объектов OBJPROP_CHART_ID функции ObjectGetInteger получим идентификатор графика, используя который мы теперь можем применять функции работы с графиками (https://www.mql5.com/ru/docs/chart_operations) и свойства графиков (https://www.mql5.com/ru/docs/constants/chartconstants/enum_chart_property):

long chartId=ObjectGetInteger (0,«Chart», OBJPROP_CHART_ID);

Откроем наш график символа, к которому мы хотим присоединить индикатор, и нажав правой кнопкой мышки, выберем пункт в контекстном меню Шаблоны и Сохранить шаблон.

Теперь мы можем перенести на наш графический объект все настройки и индикаторы графика символа:

```
ChartApplyTemplate(chartId,"my.tpl»);
ChartRedraw (chartId);
// – —
return (INIT_SUCCEEDED);
}
```

Присоединив индикатор к графику символа, мы можем нажать на нем правой кнопкой мышки и изменить его свойства, включая его период, размеры и др.

Функция PlaySound

Функция PlaySound воспроизводит звуковой файл. Например, это можно делать при появлении сигнала индикатора для напоминания:

```
bool PlaySound (
string filename // имя WAV-файла
);
```

В качестве примера добавим звуковой сигнал в наш индикатор Impulse keeper при появлении первого сигнала на покупку или продажу.

Скачаем какой-нибудь WAV-сигнал из Интернета и поместим его файл в папку Sounds терминала.

Добавим код в индикатор Impulse keeper:

```
#property indicator_chart_window
#property indicator_buffers 4

double EMA34HBuffer [];
double EMA34LBuffer [];
double EMA125Buffer [];
double PSARBuffer [];

int EMA34HHandle;
int EMA34LHandle;
int EMA125Handle;
```

```
int PSARHandle;

int bars_calculated=0;
int countBuy=0
int countSell=0;
//+ – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – —
— – — – — – — – — – +
//| Custom indicator initialization function |
//+ – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – —
— – — – — – — – — – +
int OnInit ()
{
// – - indicator buffers mapping
SetIndexBuffer (0,EMA34HBuffer, INDICATOR_CALCULATIONS);
SetIndexBuffer (1,EMA34LBuffer, INDICATOR_CALCULATIONS);
SetIndexBuffer (2,EMA125Buffer, INDICATOR_CALCULATIONS);
SetIndexBuffer (3,PSARBuffer, INDICATOR_CALCULATIONS);

EMA34HHandle=iMA (NULL,0,34,0,MODE_EMA, PRICE_HIGH);
EMA34LHandle=iMA (NULL,0,34,0,MODE_EMA, PRICE_LOW);
```

```
EMA125Handle=iMA (NULL,0,125,0,MODE_EMA,
PRICE_CLOSE);
PSARHandle=iSAR (NULL,0,0.02, 0.2);

// – —
return (INIT_SUCCEEDED);
}
//+ – — – — – — – — – — – — – — – — – — – — – — – —
— – — – — – — – — – +
//| Custom indicator iteration function |
//+ – — – — – — – — – — – — – — – — – — – — – — – —
— – — – — – — – — – +
int OnCalculate (const int rates_total,
const int prev_calculated,
const datetime &time [],
const double &open [],
const double &high [],
const double &low [],
const double &close [],
const long &tick_volume [],
const long &volume [],
const int &spread [])
{
// – —
int values_to_copy;
int start;
```

```
int calculated=BarsCalculated (EMA34HHandle);
if (calculated <=0)
{
return (0);
}

if (prev_calculated==0 || calculated!=bars_calculated)
{
start=1;
if (calculated> rates_total) values_to_copy=rates_total;
else values_to_copy=calculated;
}
else
{
start=rates_total-1;
values_to_copy=1;
}
if (!FillArrayFromMABuffer (EMA34HBuffer,0,EMA34HHandle, values_to_copy)) return (0); if (!FillArrayFromMABuffer (EMA34LBuffer,0,EMA34LHandle, values_to_copy)) return (0); if (!FillArrayFromMABuffer (EMA125Buffer,0,EMA125Handle, values_to_copy)) return (0);
if (!FillArrayFromPSARBuffer (PSARBuffer, PSARHandle,
```

```
values_to_copy)) return (0);

   for (int i=start; i <rates_total &&!IsStopped ();i++)
   {

   if (close [i-1]> open [i-1] &&close [i-1]> EMA34HBuffer
[i-1] &&close [i-1]> EMA34LBuffer [i-1] &&low
[i-1]> EMA125Buffer [i-1] &&low [i-1]> PSARBuffer
[i-1] &&EMA125Buffer [i-1] <EMA34LBuffer [i-1]
&&EMA125Buffer [i-1] <EMA34HBuffer [i-1]) {
   if (!ObjectCreate (0,«Buy»+i, OBJ_ARROW,0,time
[i-1],high [i-1]))
   {
   return (false);
   }
   ObjectSetInteger (0,«Buy»+i, OBJPROP_COLOR,
clrGreen);
   ObjectSetInteger (0,«Buy»+i,
OBJPROP_ARROWCODE,233);
   ObjectSetInteger (0,«Buy»+i, OBJPROP_WIDTH,2);
   ObjectSetInteger (0,«Buy»+i, OBJPROP_ANCHOR,
ANCHOR_UPPER);
   ObjectSetInteger (0,«Buy»+i, OBJPROP_HIDDEN, true);
   ObjectSetString (0,«Buy»+i, OBJPROP_TOOLTIP, close
[i-1]);
   }
```

```
if (start!=1) {
if (close [i-1]> open [i-1] &&close [i-1]> EMA34HBuffer [i-1] &&close [i-1]> EMA34LBuffer [i-1] &&low [i-1]> EMA125Buffer [i-1] &&low [i-1]> PSARBuffer [i-1] &&EMA125Buffer [i-1] <EMA34LBuffer [i-1] &&EMA125Buffer [i-1] <EMA34HBuffer [i-1]) {
countBuy++;
if (countBuy==1) PlaySound («chime. wav»)
} else {
countBuy=0;
}

if (close [i-1] <open [i-1] &&close [i-1] <EMA34HBuffer [i-1] &&close [i-1] <EMA34LBuffer [i-1] &&high [i-1] <EMA125Buffer [i-1] &&high [i-1] <PSARBuffer [i-1] &&EMA125Buffer [i-1]> EMA34LBuffer [i-1] &&EMA125Buffer [i-1]> EMA34HBuffer [i-1]) {
countSell++;
if (countSell==1) PlaySound («chime. wav»);
} else {
countSell=0;
}
}

if (close [i-1] <open [i-1] &&close [i-1] <EMA34HBuffer
```

```
[i-1] &&close [i-1] <EMA34LBuffer [i-1] &&high
[i-1] <EMA125Buffer [i-1] &&high [i-1] <PSARBuffer
[i-1] &&EMA125Buffer [i-1]> EMA34LBuffer [i-1]
&&EMA125Buffer [i-1]> EMA34HBuffer [i-1]) {
   if (!ObjectCreate (0,«Sell»+i, OBJ_ARROW,0,time
[i-1],low [i-1]))
   {
   return (false);
   }
   ObjectSetInteger (0,«Sell»+i, OBJPROP_COLOR, clrRed);
   ObjectSetInteger (0,«Sell»+i,
OBJPROP_ARROWCODE,234);
   ObjectSetInteger (0,«Sell»+i, OBJPROP_WIDTH,2);
   ObjectSetInteger (0,«Sell»+i, OBJPROP_ANCHOR,
ANCHOR_LOWER);
   ObjectSetInteger (0,«Sell»+i, OBJPROP_HIDDEN, true);
   ObjectSetString (0,«Sell»+i, OBJPROP_TOOLTIP, close
[i-1]);
   }
   }

   bars_calculated=calculated;
   // – - return value of prev_calculated for next call
   return (rates_total);
   }
   //+ – — – — – — – — – — – — – — – — – — – — – — –
```

```
— – — – — – — – — – +
   bool FillArrayFromPSARBuffer (double &sar_buffer [], //
индикаторный буфер значений Parabolic SAR
   int ind_handle, // хэндл индикатора iSAR
   int amount // количество копируемых значений
   )
   {
   ResetLastError ();
   if (CopyBuffer (ind_handle,0,0,amount, sar_buffer) <0)
   {
   return (false);
   }
   return (true);
   }
   //+ – — – — – — – — – — – — – — – — – — – — – — – — – —
— – — – — – — – — – +
   bool FillArrayFromMABuffer (double &values [], // индика-
торный буфер значений Moving Average
   int shift, // смещение
   int ind_handle, // хэндл индикатора iMA
   int amount // количество копируемых значений
   )
   {
   ResetLastError ();
   if (CopyBuffer (ind_handle,0, -shift, amount, values) <0)
   {
```

```
return (false);
}
return (true);
}

void OnDeinit (const int reason) {
ObjectsDeleteAll (0, -1, -1);
}
```

Здесь добавлены счетчики сигналов на продажу и покупку countBuy, countSell, для того, чтобы сигнал звучал только при появлении первого сигнала.

Функция OnChartEvent

Функция OnChartEvent является функцией обратного вызова, которая вызывается при взаимодействии пользователя с графиком символа и событиях, связанных с графическими объектами графика символа.

```
void OnChartEvent (const int id, // идентификатор события
const long& lparam, // параметр события типа long
const double& dparam, // параметр события типа double
const string& sparam // параметр события типа string
);
```

В качестве примера использования функции OnChartEvent рассмотрим наш индикатор Impulse keeper и добавим в него функциональность, позволяющую посмотреть значения используемых индикаторов при клике на сигнале покупки или продажи индикатора.

Для этого добавим в код индикатора функцию OnChartEvent, обрабатывающую событие щелчка мыши на графическом объекте индикатора:

```
#property indicator_chart_window
#property indicator_buffers 4
double EMA34HBuffer [];
double EMA34LBuffer [];
double EMA125Buffer [];
double PSARBuffer [];
```

```
int EMA34HHandle;
int EMA34LHandle;
int EMA125Handle;
int PSARHandle;

int bars_calculated=0;
//+ – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – +
//| Custom indicator initialization function |
//+ – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – — – +
int OnInit ()
{
// – - indicator buffers mapping
SetIndexBuffer (0,EMA34HBuffer, INDICATOR_CALCULATIONS);
SetIndexBuffer (1,EMA34LBuffer, INDICATOR_CALCULATIONS);
```

Конец ознакомительного фрагмента.

Текст предоставлен ООО «ЛитРес».

Прочитайте эту книгу целиком, купив полную легальную версию на ЛитРес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.