MetaTrader 4 Expert Advisors

MetaQuotes Language 4

Особености

В този раздел са представени няколко текста, в които се обясняват различните аспекти на създаването и използването на експертните системи и потребителските индикатори. Преди да започнете да пишете собствени програми на програмния език MQL 4, прочетете внимателно тези текстове, защото в тях можете да намерите някои от отговорите на вашите въпроси.

- **Особености при създаването на експертни системи**
- **Особености при създаването на потребителски индикатори**
- **Strategy Tester: режим за моделиране при тестване**
- **Особености и ограничения при тестването на програмите в MetaTrader 4**

Освен експертни системи и потребителски индикатори, вие имате възможност да създавате и използвате скриптове и библиотеки. Основното различие между скриптовете и експертните системи се състои в начините им на изпълнение: експертните системи се изпълняват въз основа на тиковете, докато скриптовете се изпълняват еднократно, чрез извикване. С помощта на скриптовете, например, могат да се извършват различни последователности от действия на клиентския терминал, извеждане на допълнителни съобщения, както и за очакване на някакви действия от страна на потребителя.

Библиотеките са предназначени за създаване на архиви от потребителски функции, т.е. програмистът може да събере оригиналните функции във файловете на библиотеките и да ги извиква от всяка потребителска програма. При създаване на библиотеките, всички описани в нея функции се компилират и съхраняват за последващо използване. Трябва да се отбележи, че при компилиране в резултат на оптимизирането, функциите, които не се извикват, се премахват от кодовете на експертните системи, потребителските индикатори и скриптовете. В това се състоят особеностите при създаването на библиотеки от други програми на програмния език MQL 4.

Внимание! *Всички права над публикуваните материали принадлежат на* ***СТС Финанс****.*
Препечатването им е разрешено само със задължителна препратка към нашия сайт!

Особености при създаването на експертни системи

Създаването и тестването на експертните системи в търговската система **MetaTrader** имат няколко особености.

- Преди да отворите позиция трябва да проверите, дали имате свободни пари по сметката. Ако нямате достатъчно пари по сметката, откриването на позиция ще завърши неуспешно. Освен това, в процеса на тестване стойността на "FreeMargin" не трябва да е по-малка от 1000, тъй като при тестване цената на един лот възлиза на 1000.

```
if(AccountFreeMargin() < 1000) return(0);   // няма пари - излизаме
```

- Достъп до историческите данни се осигурява с помощта на индексираните масиви Time, Open, Low, High, Close, Volume, които са определени предварително. Индексът в тези масиви расте от края към началото, т.е. най-последните данни имат индекс 0. Индекс 1 означава, че данните са изместени с един период назад, индекс 2 – с два периода назад, 3 – с три периода назад и т.н..

```
// ако Close на предишния бар е по-малък от
// Close на по-предишния бар
if(Close[1] < Close[2]) return(0);
```

- Достъпът до историческите данни може да се осигури и с помощта на други времеви интервали и даже с други валутни двойки. За тази цел трябва предварително да се определи едномерният масив и да се извърши копиране с помощта на функцията "ArrayCopySeries". Освен това при извикването на тази функция може да се предава по-малък брой параметри и да не се посочват параметрите по подразбиране.

```
double eur_close_m1[];
int number_copied = ArrayCopySeries(eur_close_m1, MODE_CLOSE,
"EURUSD", PERIOD_M1);
```

- При създаването на експертна система, както и при създаването на всяка друга програма, понякога се налага да се изведе информация за задаване на допълнителни настройки. Програмният език MQL 4 предоставя няколко възможности за извеждане на такава информация.
  - Функцията "Alert" извежда на екрана диалогов прозорец, който съдържа определени от потребителя данни.

```
Alert("FreeMargin grows to ", AccountFreeMargin(), "!");
```

  - Функцията "Comment" извежда в левия горен ъгъл на графиката определени от потребителя данни. Последователността от символи "\n" се използва за преминаване на нов ред.

```
Comment("FreeMargin is ", AccountFreeMargin(), ".");
```

  - Функцията "Print" отпечатва определените от потребителя данни в системния журнал.

```
Print("FreeMargin is ", AccountFreeMargin(), ".");
```

- Функцията "GetLastError" служи за получаване на информацията за грешките в програмите. Например, операциите, свързани с поръчките винаги връщат номера на тикета. Ако номерът на тикета е равен на 0 (възникнала е някаква грешка при извършването на дадена операция), то за получаване на допълнителна информация за тази грешка трябва да се извика функцията "GetLastError":

```
int iTickNum = 0;
int iLastError = 0;
...
iTickNum = OrderSet (OP_BUY, g_Lots, Ask, 3, 0, Ask + g_TakeProfit *
g_Points, Red);
if (iTickNum <= 0)
  {
   iLastError = GetLastError();
   if (iLastError != ERROR_SUCCESS) Alert("Some Message");
  }
```

Не забравяйте, че извикването на функцията "GetLastError" показва кода на последната грешка и занулява неговата стойност. Затова повторното извикване на тази функция винаги ще връща стойност 0.

- Как да се определи началото на поредния бар? (Това понякога е необходимо, за да се разбере, че предходният бар току що се е образувал). Има няколко начина.

Първият начин се основава на проверката на броя на баровете:

```
int prevbars = 0;
...
if(prevbars == Bars) return(0);
prevbars = Bars;
...
```

Този начин обаче може да не работи при допълнително зареждане на историческите данни, т.е. броят на баровете се е променил, а "предходният" бар все още не се е образувал. В този случай може да се усложни проверката на разликата между стойностите, която е равна на единица.

Другият начин се основава на това, че стойността на "Volume" се определя според броя на тиковете, пристигнали за всеки бар. Първият тик означава, че отново образуващия се бар има стойност на "Volume" равна на 1:

```
if( Volume[0] > 1) return(0);
...
```

Този начин може да не работи при извънредно интензивно постъпване на ценовите тикове. Причината се крие в това, че обработването на пристигащите ценови тикове се осъществява в отделен поток. Ако този поток е зает по време на пристигането на поредния тик, то този тик не се обработва, за да се избегне излишното натоварване на изчислителните ресурси! В този случай също може да се усложни проверката, като се използва съхраняване на предишната стойност на "Volume".

Третият начин се основава на времето на отваряне на бара:

```
datetime prevtime=0;
...
if(prevtime == Time[0]) return(0);
```

```
prevtime = Time[0];
...
```

Това е най-сигурният начин. Той ще се задейства при всякакви обстоятелства.

- Пример за работа с файл от типа "CSV":

```
int h1;
h1 = FileOpen("my_data.csv", MODE_CSV | MODE_WRITE, ";");
if(h1<0)
  {
   Print("Unable to open file my_data.csv");
   return(false);
  }
FileWrite(h1, High[1], Low[1], Close[1], Volume[1]);
FileClose(h1);
```

- Някои пояснения към кода. Първо се отваря файл с формат "CSV". В случай на грешка при отварянето на файла се осъществява изход от програмата. При успешно отваряне на файла съдържанието му се изтрива, данните се записват във файл и този файл се затваря. Ако трябва да се съхрани съдържанието на отворения файл, трябва да се използва режимът за отваряне MODE_READ:

```
int h1;
h1 = FileOpen("my_data.csv", MODE_CSV | MODE_WRITE | MODE_READ, ";");
if(h1<0)
 {
  Print("Unable to open file my_data.csv");
  return(false);
 }
FileSeek(h1, 0, SEEK_END);
FileWrite(h1, High[1], Low[1], Close[1], Volume[1]);
FileClose(h1);
```

- В този пример записването се осъществява в края на файла. В този случай, веднага след отварянето му ние използвахме функцията "FileSeek".

Особености при създаването на потребителски индикатор

Създаването на потребителски индикатори в търговската система MetaTrader също има няколко особености.

- Програмата трябва да има описание, за да се счита за потребителски индикатор:

```
#property indicator_chart_window      // индикаторът се изобразява в главния
                                        прозорец на графиката
```

или

```
#property indicator_separate_window   // индикаторът се изобразява в отделен прозорец
```

- За определяне на мащаба на отделния прозорец на индикатора се използват:

```
#property indicator_minimum Min_Value
#property indicator_maximum Max_Value
```

където "Min_Value" и "Max_Value" са съответните числови стойности. Например, за потребителския индикатор RSI тези стойности трябва да бъдат съответно 0 и 100.

- Броят на индикаторните масиви, необходими за изобразяването на индикатора, трябва да бъде зададен с помощта на:

```
#property indicator_buffer N
```

където N може да приема стойности от 1 до 8.

- Цветовете на линиите на индикатора се задават с помощта на:

```
#property indicator_color1 Blue
#property indicator_color2 Red
...
#property indicator_colorN <SomeColor>
```

където N е броят на индикаторните масиви, определен с помощта на "#property indicator_buffer".

- Съществуват няколко функции, които позволяват да се управлява изчисляването и визуализацията на индикатора. В нашия пример използваме потребителския индикатор Ichimoku Kinko Hyo:

```
//+------------------------------------------------------------------+
//|                                                 Ichimoku.mq4 |
//|                  Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      http://www.metaquotes.net/


#property indicator_chart_window
#property indicator_buffers 7
```

```
#property indicator_color1 Red
#property indicator_color2 Blue
#property indicator_color3 SandyBrown
#property indicator_color4 Thistle
#property indicator_color5 Lime
#property indicator_color6 SandyBrown
#property indicator_color7 Thistle
//---- input parameters
extern int Tenkan=9;
extern int Kijun=26;
extern int Senkou=52;
//---- indicator buffers
double Tenkan_Buffer[];
double Kijun_Buffer[];
double SpanA_Buffer[];
double SpanB_Buffer[];
double Chinkou_Buffer[];
double SpanA2_Buffer[];
double SpanB2_Buffer[];
//---- span_a drawing begin
int a_begin;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
```

MetaTrader е собственост на **MetaQuotes Software Corp.**
Оригинален текст: http://www.metaquotes.ru/experts/mql4/

```
int init()
  {
//----
   SetIndexStyle(0,DRAW_LINE);
   SetIndexBuffer(0,Tenkan_Buffer);
   SetIndexDrawBegin(0,Tenkan-1);
   SetIndexLabel(0,"Tenkan Sen");
//----
   SetIndexStyle(1,DRAW_LINE);
   SetIndexBuffer(1,Kijun_Buffer);
   SetIndexDrawBegin(1,Kijun-1);
   SetIndexLabel(1,"Kijun Sen");
//----
   a_begin=Kijun; if(a_begin<Tenkan) a_begin=Tenkan;
   SetIndexStyle(2,DRAW_HISTOGRAM,STYLE_DOT);
   SetIndexBuffer(2,SpanA_Buffer);
   SetIndexDrawBegin(2,Kijun+a_begin-1);
   SetIndexShift(2,Kijun);
   SetIndexLabel(2,NULL);
   SetIndexStyle(5,DRAW_LINE,STYLE_DOT);
   SetIndexBuffer(5,SpanA2_Buffer);
   SetIndexDrawBegin(5,Kijun+a_begin-1);
   SetIndexShift(5,Kijun);
   SetIndexLabel(5,"Senkou Span A");
```

MetaTrader е собственост на **MetaQuotes Software Corp.**
Оригинален текст: http://www.metaquotes.ru/experts/mql4/

```
//----
   SetIndexStyle(3,DRAW_HISTOGRAM,STYLE_DOT);
   SetIndexBuffer(3,SpanB_Buffer);
   SetIndexDrawBegin(3,Kijun+Senkou-1);
   SetIndexShift(3,Kijun);
   SetIndexLabel(3,NULL);
   SetIndexStyle(6,DRAW_LINE,STYLE_DOT);
   SetIndexBuffer(6,SpanB2_Buffer);
   SetIndexDrawBegin(6,Kijun+Senkou-1);
   SetIndexShift(6,Kijun);
   SetIndexLabel(6,"Senkou Span B");
//----
   SetIndexStyle(4,DRAW_LINE);
   SetIndexBuffer(4,Chinkou_Buffer);
   SetIndexShift(4,-Kijun);
   SetIndexLabel(4,"Chinkou Span");
//----
   return(0);
  }
//+------------------------------------------------------------------+
//| Ichimoku Kinko Hyo                                               |
//+------------------------------------------------------------------+
int start()
  {
```

```
   int    i,k;
   int    counted_bars=IndicatorCounted();
   double high,low,price;
//----
   if(Bars<=Tenkan || Bars<=Kijun || Bars<=Senkou) return(0);
//---- initial zero
   if(counted_bars<1)
    {
     for(i=1;i<=Tenkan;i++)    Tenkan_Buffer[Bars-i]=0;
     for(i=1;i<=Kijun;i++)     Kijun_Buffer[Bars-i]=0;
     for(i=1;i<=a_begin;i++) { SpanA_Buffer[Bars-i]=0; SpanA2_Buffer[Bars-
i]=0; }
     for(i=1;i<=Senkou;i++)  { SpanB_Buffer[Bars-i]=0; SpanB2_Buffer[Bars-
i]=0; }
    }
//---- Tenkan Sen
   i=Bars-Tenkan;
   if(counted_bars>Tenkan) i=Bars-counted_bars-1;
   while(i>=0)
    {
     high=High[i]; low=Low[i]; k=i-1+Tenkan;
     while(k>=i)
      {
       price=High[k];
       if(high<price) high=price;
```

```
         price=Low[k];
         if(low>price)  low=price;
         k--;
        }
       Tenkan_Buffer[i]=(high+low)/2;
      i--;
     }
//---- Kijun Sen
   i=Bars-Kijun;
   if(counted_bars>Kijun) i=Bars-counted_bars-1;
   while(i>=0)
    {
     high=High[i]; low=Low[i]; k=i-1+Kijun;
     while(k>=i)
      {
       price=High[k];
       if(highprice)  low=price;
       k--;
      }
     Kijun_Buffer[i]=(high+low)/2;
     i--;
    }
//---- Senkou Span A
   i=Bars-a_begin+1;
```

MetaTrader е собственост на **MetaQuotes Software Corp.**
Оригинален текст: http://www.metaquotes.ru/experts/mql4/

```
   if(counted_bars>a_begin-1) i=Bars-counted_bars-1;
   while(i>=0)
     {
      price=(Kijun_Buffer[i]+Tenkan_Buffer[i])/2;
      SpanA_Buffer[i]=price;
      SpanA2_Buffer[i]=price;
      i--;
     }
//---- Senkou Span B
   i=Bars-Senkou;
   if(counted_bars>Senkou) i=Bars-counted_bars-1;
   while(i>=0)
     {
      high=High[i]; low=Low[i]; k=i-1+Senkou;
      while(k>=i)
        {
         price=High[k];
         if(high<price) high=price;
         price=Low[k];
         if(low>price)  low=price;
         k--;
        }
      price=(high+low)/2;
      SpanB_Buffer[i]=price;
```

```
      SpanB2_Buffer[i]=price;
      i--;
     }
//---- Chinkou Span
   i=Bars-1;
   if(counted_bars>1) i=Bars-counted_bars-1;
   while(i>=0) { Chinkou_Buffer[i]=Close[i]; i--; }
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

- Функцията "SetIndexStyle" служи за управление на параметрите за изобразяване на индикаторния масив. Типът на изобразяване DRAW_LINE предполага, че линиите се изобразяват между стойностите, определени в съответния индикаторен масив. Типът на изобразяване DRAW_HISTOGRAM има някои особености, когато се използва за индикатора на главния прозорец. Хистограмата се изобразява между съответните стойности на двата индексни масива: четния (в нашия случай - SpanA_Buffer) и нечетния (SpanB_Buffer). Освен това се използва цветът на онзи индексен масив, стойността на който е по-голяма.
- Функцията "SetIndexDrawBegin" показва, от кой елемент започват значимите данни на индикаторния масив.
- Функцията "SetIndexBuffer" позволява да се обяви всеки едномерен масив от типа "double" като индексен масив. Управлението на индексните масиви се извършва от системата. Именно поради тази причина размерът на тези масиви може да не се посочва.

```
//---- indicator buffers
double Tenkan_Buffer[];
double Kijun_Buffer[];
double SpanA_Buffer[];
double SpanB_Buffer[];
double Chinkou_Buffer[];
```

```
double SpanA2_Buffer[];

double SpanB2_Buffer[];
```

- Функцията ArrayResize не може да се използва с индикаторни масиви. Функцията ArrayInitialize също не може да се използва с тях, особено във функция init, когато индикаторните масиви все още не са разпределени. Инициализацията на индикаторните масиви се осъществява автоматично при разпределяне и преразпределяне на паметта. Като начална стойност се използва EMPTY_VALUE или стойността, определена с помощта на функцията SetIndexEmptyValue. "Празните" стойности не се изобразяват.
- Функцията "SetIndexLabel" определя името, което ще се показва в изскачащите подсказки и в прозореца за данни (името по подразбиране е "ValueN", където N е номерът на индексния масив). Ако вместо това име се постави NULL, съответната стойност няма да се показва нито в изскачащите подсказки, нито в прозореца за данни. В нашия случай, облаците се защриховат с помощта на хистограма и се ограничават с помощта на линия. Освен това стойностите на съответните "линейни" и "хистограмни" масиви са еднакви, поради което се показва само една от тях.
- Функцията "IndicatorCounted" позволява да се осъществи икономично изчисляване на индикатора. Тази функция връща броя на баровете, които са налице в момента на предишното стартиране на индикатора, т.е. броя на вече изчислените барове, които не се нуждаят от преизчисляване (ако при предишното стартиране на индикатора не е имало грешки или предварително спиране). При повторна инициализация на потребителския индикатор или при съществено обновяване на историческите данни броят на баровете автоматично става 0.
- Да разгледаме още един пример. Потребителски индикатор - Accelerator/Decelerator Oscillator:

```
//+------------------------------------------------------------------+
//|                                                 Accelerator.mq4 |
//|                  Copyright © 2005, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property  copyright "Copyright © 2005, MetaQuotes Software Corp."
#property  link      http://www.metaquotes.net/
//---- indicator settings
#property  indicator_separate_window
#property  indicator_buffers 3
```

```
#property  indicator_color1  Black
#property  indicator_color2  Green
#property  indicator_color3  Red
//---- indicator buffers
double     ExtBuffer0[];
double     ExtBuffer1[];
double     ExtBuffer2[];
double     ExtBuffer3[];
double     ExtBuffer4[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
//---- 2 additional buffers are used for counting.
   IndicatorBuffers(5);
//---- drawing settings
   SetIndexStyle(0,DRAW_NONE);
   SetIndexStyle(1,DRAW_HISTOGRAM);
   SetIndexStyle(2,DRAW_HISTOGRAM);
   IndicatorDigits(Digits+2);
   SetIndexDrawBegin(0,38);
   SetIndexDrawBegin(1,38);
   SetIndexDrawBegin(2,38);
```

MetaTrader е собственост на **MetaQuotes Software Corp.**
Оригинален текст: http://www.metaquotes.ru/experts/mql4/

```
//---- 4 indicator buffers mapping
   SetIndexBuffer(0,ExtBuffer0);
   SetIndexBuffer(1,ExtBuffer1);
   SetIndexBuffer(2,ExtBuffer2);
   SetIndexBuffer(3,ExtBuffer3);
   SetIndexBuffer(4,ExtBuffer4);
//---- name for DataWindow and indicator subwindow label
   IndicatorShortName("AC");
   SetIndexLabel(1,NULL);
   SetIndexLabel(2,NULL);
//---- initialization done
   return(0);
  }
//+------------------------------------------------------------------+
//| Accelerator/Decelerator Oscillator                               |
//+------------------------------------------------------------------+
int start()
  {
   int    limit;
   int    counted_bars=IndicatorCounted();
   double prev,current;
//---- last counted bar will be recounted
   if(counted_bars>0) counted_bars--;
   limit=Bars-counted_bars;
```

MetaTrader е собственост на **MetaQuotes Software Corp.**
Оригинален текст: http://www.metaquotes.ru/experts/mql4/

```
//---- macd counted in the 1-st additional buffer
   for(int i=0; i<limit; i++)
      ExtBuffer3[i]=iMA(NULL,0,5,0,MODE_SMA,PRICE_MEDIAN,i)-
                    iMA(NULL,0,34,0,MODE_SMA,PRICE_MEDIAN,i);
//---- signal line counted in the 2-nd additional buffer
   for(i=0; i=0; i--)
     {
      current=ExtBuffer3[i]-ExtBuffer4[i];
      prev=ExtBuffer3[i+1]-ExtBuffer4[i+1];
      if(current>prev) up=true;
      if(current<prev) up=false;
      if(!up)
        {
         ExtBuffer2[i]=current;
         ExtBuffer1[i]=0.0;
        }
      else
        {
         ExtBuffer1[i]=current;
         ExtBuffer2[i]=0.0;
        }
      ExtBuffer0[i]=current;
     }
//---- done
```

MetaTrader е собственост на **MetaQuotes Software Corp.**
Оригинален текст: http://www.metaquotes.ru/experts/mql4/

```
                return(0);
               }
              //+------------------------------------------------------------------+
```

- Функцията "IndicatorBuffers" служи за определяне броя на използваните буфери за изчисляването на индикатора. Обикновено тази функция се извиква тогава, когато броят на използваните индексни масиви е по-голям, отколкото е необходимо за изобразяването на индикатора. Управлението на допълнителните масиви се извършва от системата.
- Функцията "SetIndexDigits" се използва за управление на точността на извеждането на информацията. В този случай при изчисляването на разликата между двете пълзящи средни и по-нататъшната разлика между резултата и сигналната линия, стандартната точност от 4 знака след запетаята не е достатъчна.
- Функцията "SetIndexDrawBegin" посочва, от кой елемент започват значимите данни на индикаторния масив. В нашия пример сигналната линия се изчислява като проста пълзяща средна от друга проста пълзяща средна, затова първите 38 стойности на индикатора са празни и не подлежат на изобразяване.
- Функцията "IndicatorShortName" определя така наречената кратко име на индикатора, което ще се показва в левия горен ъгъл на прозореца на индикатора, както и в прозореца "DataWindow". Ако краткото име не е предварително зададено, ще се използва самото название на потребителския индикатор. В нашия пример не е необходимо да се използва функцията SetIndexLabel, тъй като се извежда само една негова стойност. Следователно, за извеждане на една-единствена стойност е достатъчно да се използва името на индикатора.
- Функцията "SetIndexStyle" се използва за управление на параметрите за изобразяване на индикаторния масив. Типът на изобразяване DRAW_NONE показва, че съответната линия може да не се изобразява. В представения индикатор хистограмата трябва да бъде оцветена с 2 различни цвята. Данните от ExtBuffer0 се разпределят между другите два масива - ExtBuffer1 и ExtBuffer2. Функцията SetIndexLabel с параметър NULL се използва за предотвратяване извеждането на дублиращи се данни в изскачащите подсказки и прозореца за данни. Типът на изобразяване DRAW_HISTOGRAM служи за изобразяване на хистограма между нулевата стойност и стойността на съответния масив (ср. с изобразяването на хистограмата в главния прозорец, описано по-горе), когато се използва за индикатора на отделния прозорец.
- Входните параметри, които приемат стойностите на потребителските индикатори и функции, трябва да бъдат определени като "extern" и могат да бъдат от всякакъв тип.
- Ако входните параметри не са определени, извикването на съответния потребителски индикатор се осъществява в най-прост формат.

```
double current_AC = iCustom( NULL, 0, "Accelerator", 0, 0 );
```

Тук предаването на първите две стойности "NULL" и "0" означава, че ще се използва текущата графика. За име на потребителския индикатор се използва името на съответния файл без разширението mq4. Предпоследният параметър 0 означава, че ни интересуват данните от първия индикаторен масив. Последният параметър 0 означава, че ни интересува стойността на последния елемент (т.е. най-прясната, текущата стойност) на съответния индикаторен масив.

- Параметрите са предават на функцията за изчисляване на потребителския индикатор в онзи ред, в който те са описани. Например, извикването на потребителския индикатор "Ichimoku" с параметри (9,26,52) ще изглежда по следния начин:

```
iCustom( NULL, 0, "Ichimoku", 9, 26, 52, 0, shift );
```

Най-общо казано, параметрите на потребителския индикатор могат да не се предават на функцията за изчисляване на потребителския индикатор. Ако в програмата не е определена нито една променлива extern, то параметрите не трябва да се предават. Освен това могат да се използват началните стойности, които се използват при описанието на параметрите. Например, извикването на същия потребителски индикатор без параметри ще изглежда по следния начин:

```
iCustom( NULL, 0, "Ichimoku", 0, shift );
```

Това означава, че ще се използват онези стойности, с които се инициализират променливите "Tenkan", "Kijun", "Senkou", т.е. 9, 26 и 52. Обаче ако в една експертна система се извика потребителски индикатор с различни набори от параметри, тогава не се препоръчва да се използват стойности по подразбиране.

Трябва да се отбележи, че прекалено големият брой и неправилно написани потребителски индикатори могат значително да забавят работата на клиентския терминал.

Strategy Tester: режими за моделиране при тестването на търговските стратегии

Въведение

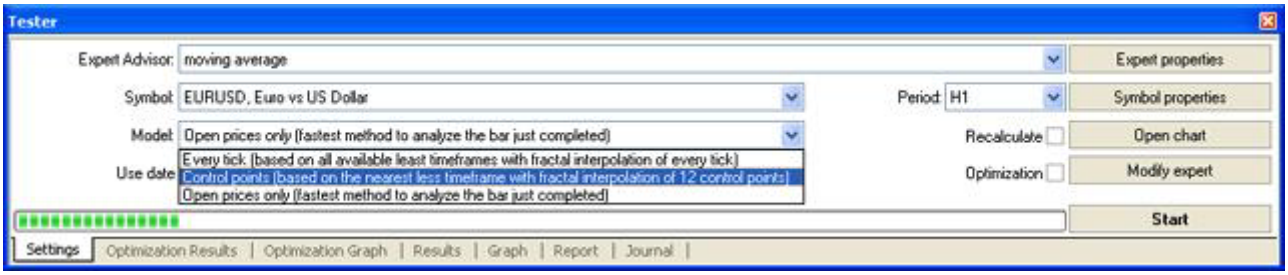
Повечето програми за извършване на технически анализ позволяват провеждането на тестване на търговските стратегии въз основа на историческите данни. В повечето случаи тестването се основава върху предварително получените данни, без опити за моделиране на движението вътре в ценовия бар. По този начин тестването протича бързо, но не е достатъчно точно.

За качествено тестване на търговската стратегия трябва да се избере адекватен начин за моделиране на движението на ценовите барове. Идеален вариант няма, т.е. да са получени пълни исторически данни въз основа на тиковете, които да доведат до абсолютно точно тестване. В повечето случаи е много трудно да се намерят пълни исторически данни въз основа на тиковете, за да се извърши точен анализ.

За решаване на този проблем данните от по-малките периоди от време могат да се използват в качеството на опорни точки, като се моделират промените на цените между тях.

Начини за моделиране на ценовите барове

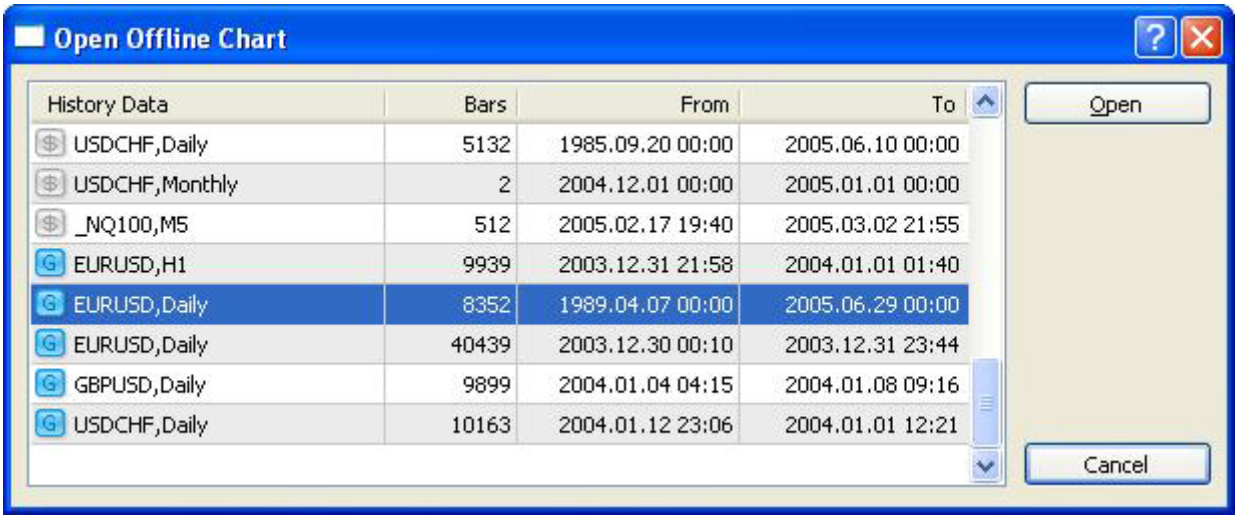
В клиентския терминал MetaTrader 4 се използват три начина за моделиране на движението на баровете:



- Всички тикове (използват се всички достъпни времеви интервали + фрактална интерполация)
- Контролни точки (използва се най-близкият времеви интервал + фрактална интерполация)
- Въз основа на цените на отваряне (бърз метод за моделиране въз основа на образувалите се барове)

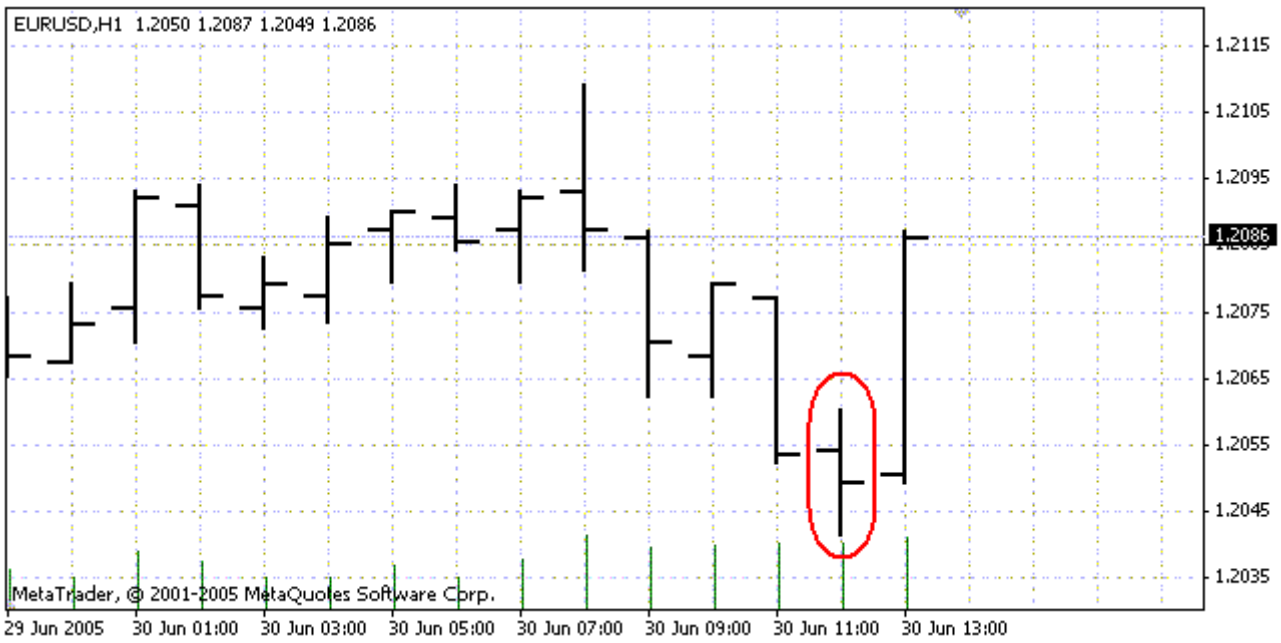
Преди началото на тестването се извършва генерация на междинните ценови барове, като резултатите се записват във файл (например: **/tester/history/eurusd_1440_1.fxt**). Кешираните по този начин данни позволяват впоследствие да се ускори работата на тестера. Отметката "Recalculate" служи за преизчисляване на междинните данни.

Използването на предварително кешираните данни дава възможност за провеждането на тестване въз основа на собствените междинни данни. Достатъчно е в каталога **/tester/history/** да се запише файл в съответния формат (формат *.FXT, напълно отворен). Тези файлове лесно се отварят в терминала като офлайн графики с помощта на **File -> Open offline**.



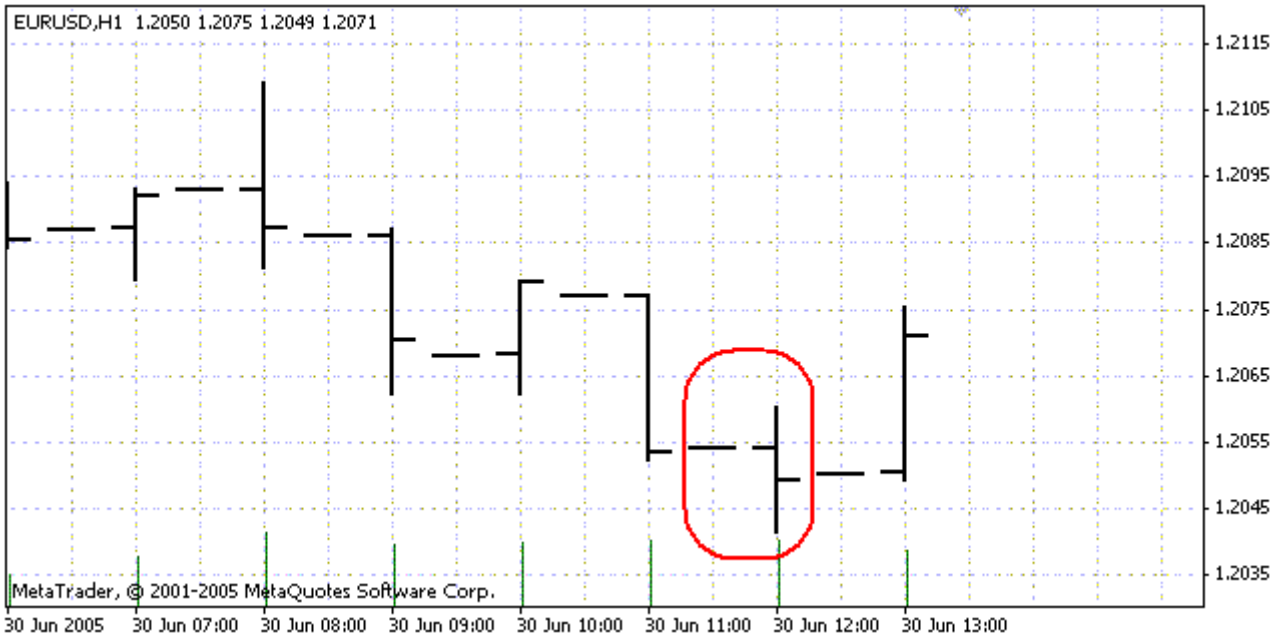
Примери на моделиране

Да започнем с най-простия начин за моделиране въз основа на часова графика. Да разгледаме часов бар (30 юни 2005, 12:00), който е отбелязан с червено:



Въз основа на цените на отваряне

Някои инвеститори не искат да зависят от особеностите на моделирането вътре в самия бар, затова те създават експертни системи, които търгуват въз основа на образувалите се вече барове. Образуването на текущия ценови бар завършва с появяването на следващия бар. Именно за такива експертни системи е предназначен режимът за моделиране "**Въз основа на цените на отваряне**".

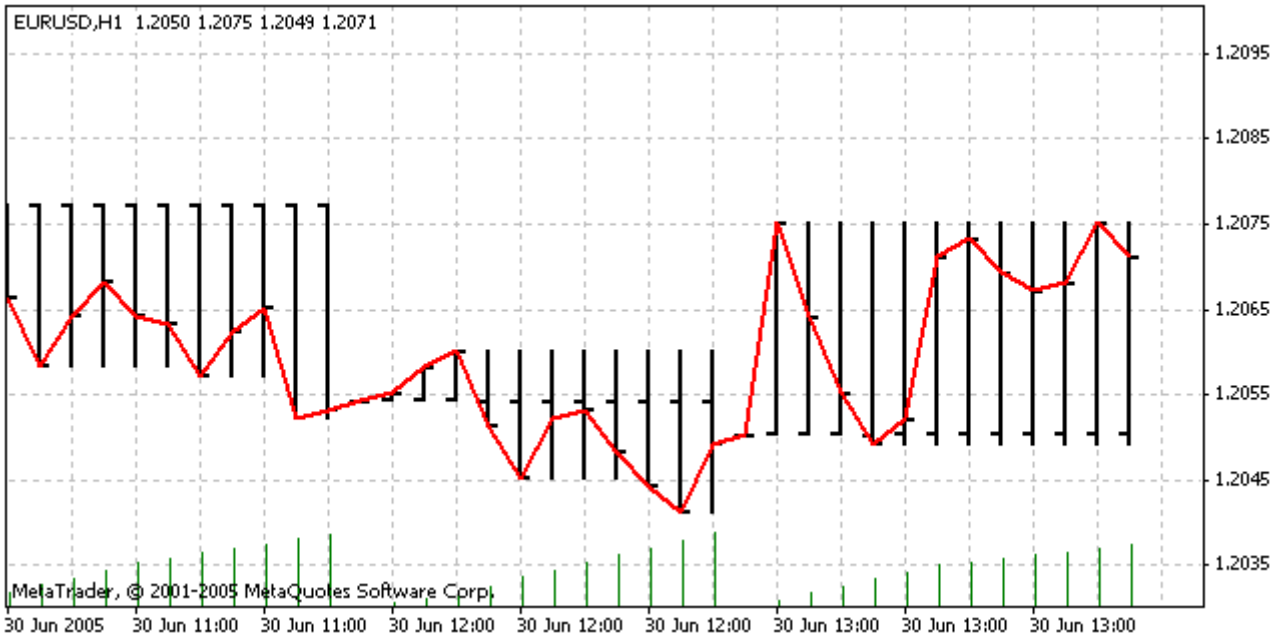


В този режим най-напред се моделира отварянето на бара (Open = High = Low = Close, Volume=1), което дава възможност на експертната система точно да идентифицира края на образуването на предишния ценови бар. Именно върху този появяващ се бар се стартира тестването на експертната система. След това се извежда вече напълно образувал се текущ бар, върху него обаче не се извършва тестване!

Контролни точки (най-близкият малък времеви интервал)

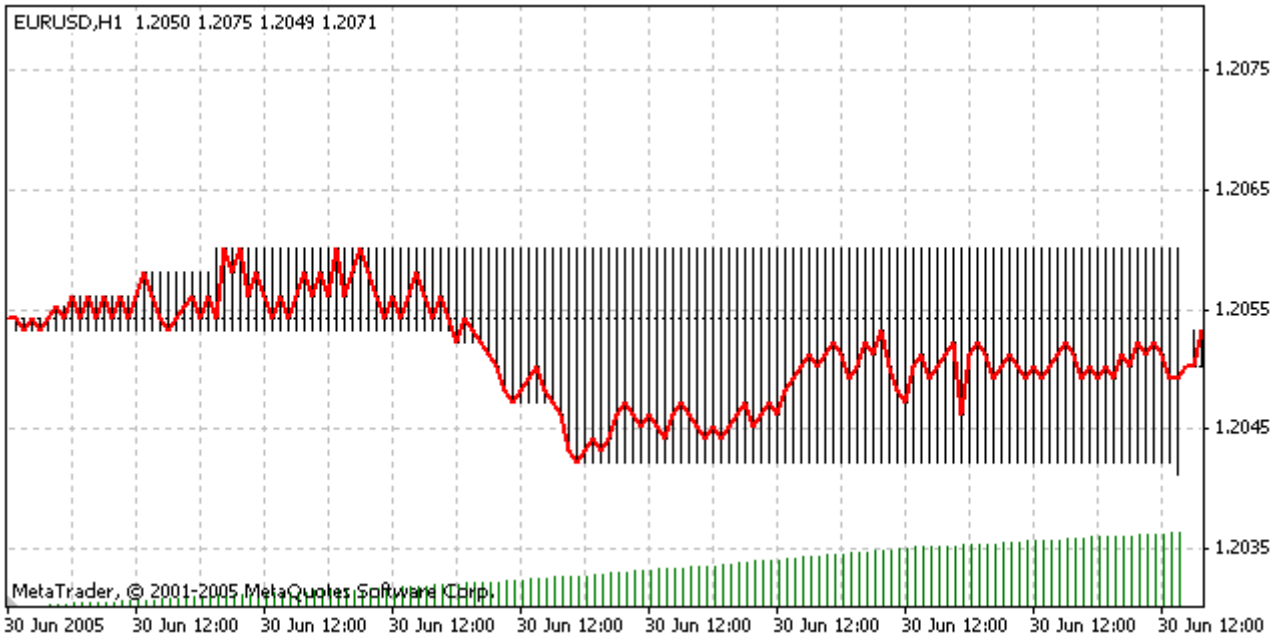
Методът за моделиране въз основа на контролните точки служи за груба оценка на експертните системи, които търгуват вътре в бара. За този метод е необходимо наличието на исторически данни от най-близкия малък времеви интервал. В повечето случаи съответните данни на по-малкия времеви интервал не покриват изцяло диапазона на тествания времеви интервал. При липса на данни от по-малкия времеви интервал движението на бара се генерира въз основа на цените на затваряне на 12 предишни бара, т.е. движението вътре в бара повтаря движението на цената през последните 12 периода. Това се нарича фрактална интерполация.

Този метод значително се различава от представения в предишните версии на терминала вълнов метод, който предоставяше строго определено движение на бара. С появяването на исторически данни за по-малкия времеви интервал, фракталната интерполация се използва вече за тези данни. В този случай обаче се използват вече не 12, а само 6 предишни бара. Това означава, че се възпроизвеждат реално съществуващи цени Open, High, Low, Close плюс още две генерирани цени. Стойностите и местоположението на тези две генерирани цени зависят от движението на цените на 6 предишни периоди.



Всички тикове (всички достъпни по-малки времеви интервали + интерполация)

Този режим позволява най-точно да се моделира движението на цената вътре в бара. За разлика от "контролните точки", този метод използва за генериране историческите данни не само на най-близкия малък времеви интервал, но и на всички достъпни по-малки времеви интервали. Освен това, ако за някой времеви диапазон има данни от повече от един времеви интервал, за генериране се използват данните на най-малкия времеви интервал. По същия начин, както и в предишния метод, контролните точки се генерират фрактално. За генериране движението на цената между контролните точки също се използва фрактална интерполация. Възможна е ситуация, когато се генерират няколко еднакви тика поред. В този случай дублиращите се котировки се филтрират и се фиксира обемът на последната от тези котировки.



Трябва да се има предвид, че генерираните въз основа на тиковете данни могат да имат прекалено голям обем, което, от своя страна, може да окаже значително влияние както на използваните от операционната система ресурси, така и на скоростта на тестването.

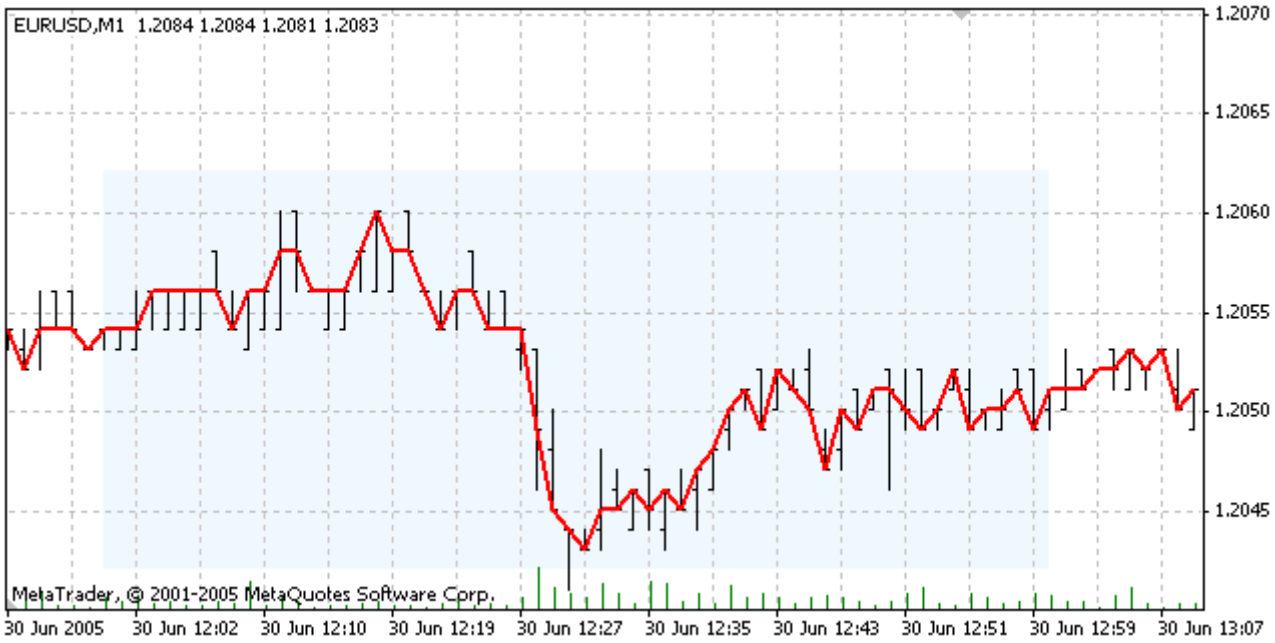
Внимание: при липса на по-малки времеви интервали, които напълно да покриват изследвания период, тестването въз основа на тиковете не трябва да се извършва. Моделирането въз основа на тиковете е предназначено само за тестване въз основа на данните от по-малките времеви интервали!

Използване диапазона на датите при моделиране.

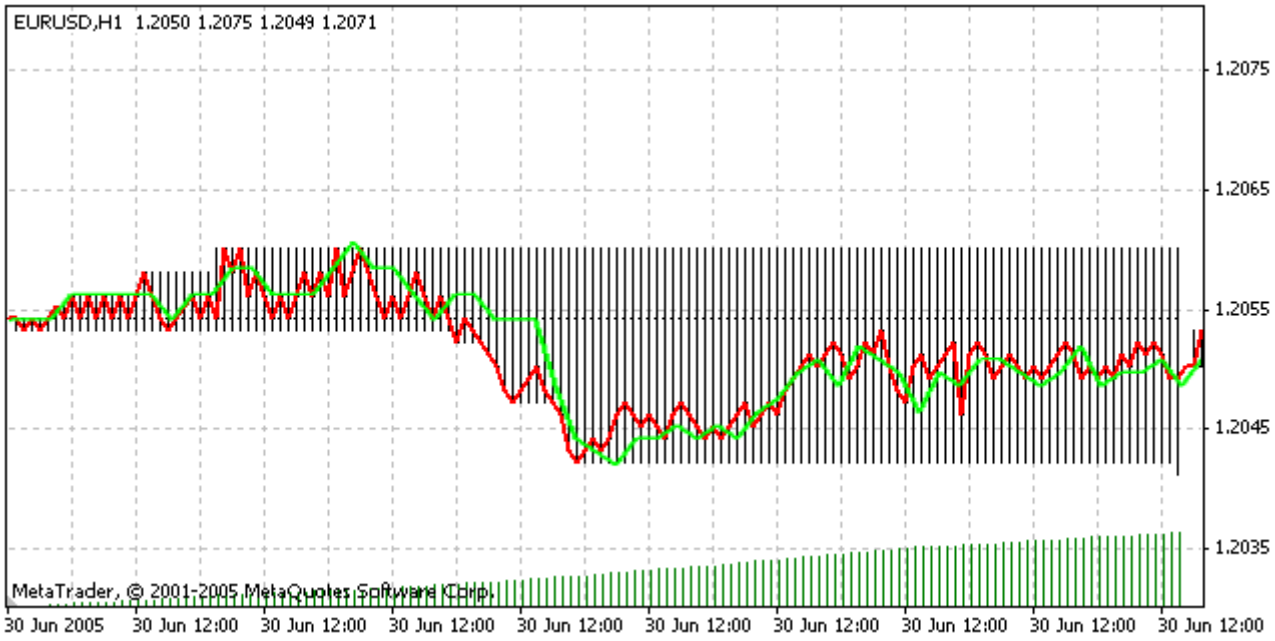
Диапазонът на датите може да се използва не само при тестването на експертната система, но и при генерирането на последователността от барове. Обикновено данните на цялата история може да не се генерират, особено при моделиране въз основа на тиковете, когато обемът на неизползваните данни може да бъде много голям. Затова, ако при първоначалното генериране на последователността от барове е била включена възможността за използване на диапазона на датите, то баровете, които не влизат в пределите на съответния диапазон, не се генерират, а просто се презаписват в изходна последователност. Тези данни не се изключват от съответната последователност, за да има възможност за правилно изчисляване на индикаторите въз основа на цялата получена история. Трябва да се отбележи, че първите 100 бара също не се генерират, това ограничение не зависи от зададения диапазон на датите.

Сравнение с еталон от периода M1

За сравнение на точността на моделирането вътре в бара ще използваме минутна графика от 30 юни 2005, 12:00 до 13:00.



След процеса на мащабиране на оригиналната минутна графика (зелената линия представлява цената Close) и наслагването ѝ върху графиката на моделираните въз основа на тиковете данни се вижда, че те почти напълно съвпадат:



Изводи

Максимално точно тестване и добра гаранция за достоверност на резултатите може да се получи при наличие на спомагателни времеви интервали за по-малките периоди, които напълно покриват изследвания период, т.е. въпросът относно качественото тестване въз основа на тиковете се състои в търсенето на детайлни исторически данни...

Особености и ограничения при тестването на търговските стратегии в MetaTrader 4

Въведение

Този текст ще ви помогне да научите повече за особеностите и ограниченията на тестера на експертните системи в MetaTrader 4.

Особености на работа на тестера

- ***Някои функции се извършват/пропускат без извеждане***
  Това са: Sleep(), Alert(), SendMail(), SpeechText(), PlaySound(), MessageBox(), WindowFind(), WindowHandle(), WindowIsVisible()

- ***Търговията може да се извършва само върху тествания символ, няма портфейлно тестване***
  Опитите да се извърши сделка върху чужд символ ще доведат до грешка.

- ***Размерността и кратността на лотовете, комисионните и суаповете се вземат от настройките на текущия активен акаунт***
  Преди да започнете тестването вие трябва да се уверите, че в терминала има поне един активен акаунт в списъка на прозореца "Навигатор".

- ***Всички суапове, изискванията за гаранционна сума, expirations и GTC ордери се моделират***
  Тестването се извършва максимално близо до условията на търговския сървър. При работа с кръстосани курсове обаче могат да възникнат някои грешки в оценката на изискванията за гаранционна сума, поради липсата на точна информация за курсовете във всеки момент от време.

- ***Нулевият бар на друг период въз основа на същия тестван символ се моделира приблизително***
  Open = правилен Open, Close = правилен Close, Low = min (Open,Close), High = max (Open,Close), Volume = краен Volume (неверен)

- ***За сключването на сделките се използва режимът Instant Execution въз основа на текущите цени***

- ***Отваряне/затваряне на ордерите без пропуски***

- ***Тестването спира след StopOut***

- ***Седмичните, месечните и нестандартните периоди не се тестват***

- ***Валутата на депозита може да се сменя, курсовете на конвертирането обаче се фиксират, като се използват текущите достъпни курсове***

- ***Няма забавяния в извършването на търговските операции***
  Планира се да се внесе определено забавяне в извършването на сделките

- ***Account History е напълно достъпен, в реалния акаунт обаче това зависи от настройките***

- ***При активно използване на други символи и периоди е желателно те да се заредят предварително***

- ***При моделиране въз основа на тиковете тестерът самостоятелно дозарежда всички необходими времеви интервали за тествания символ***

- ***Използването на функцията MarketInfo ще доведе до следната грешка: ERR_FUNCTION_NOT_ALLOWED_IN_TESTING_MODE(4059),*** макар че се извежда коректна информация за текущите цени на тествания символ, за размера на нивата на стоповете, за размера на пипса, както и за размера на спреда на всеки символ, присъстващ в прозореца на котировките.

Особености на работата на оптимизатора

- ***В журнала на логовете не се извежда никаква информация (включително функцията Print())***
  Това е направено с цел ускоряване на процеса на тестването и икономия на свободното пространство на диска. При извеждане на пълни логове, файловете на журналите обикновено заемат стотици мегабайтове.

- ***Графичните обекти реално не се показват***
  Обектите се изключват с цел ускоряване на тестването.

- ***Използва се функцията "Пропускай безполезните резултати"***
  За да не се препълва таблицата и графиката на резултатите от тестването, се използва възможността за пропускане на лошите резултати. Тази опция се изключва от контекстното меню "Резултати на оптимизацията" -> "Пропускай безполезните резултати".