Программный комплекс QUIK -
фронт-офисная система прямого доступа

Создание индикаторов технического анализа с помощью скриптов Lua

Инструкция

Содержание

Начиная с версии 6.9, в рабочем месте QUIK появилась возможность создавать собственные индикаторы технического анализа. В данном документе рассматривается процесс создания индикатора на примере «скользящей средней» (Moving Average)

1. Как устроены индикаторы в QUIK

Основой для построения всех индикаторов в QUIK является источник данных (далее ИД). ИД представляет собой массив, в котором элементы являются структурами и имеют 6 полей:

1. Open;
2. High;
3. Low;
4. Close;
5. Volume;
6. Time.

Фактически, это значит, что все элементы массива в источнике данных представляют собой свечи. В случае тиковых данных поля с 1-го по 4-е буду иметь одно значение, и оно будет совпадать со значением параметра в этот момент времени. Источники данных могут быть интервальными графиками (тики, 1 минута, 5 минут и т.д.), рассчитанными по таблице всех сделок или по изменениям параметра торгуемого инструмента

Индикатор представляет собой функцию, которая для элемента массива ИД может вернуть одно или несколько чисел, в зависимости от количества линий, отображаемых на графике.

Индикатор не может выступать источником данных для другого индикатора.

2. *Минимальный код индикатора*

Пример 1 (ex1.lua)

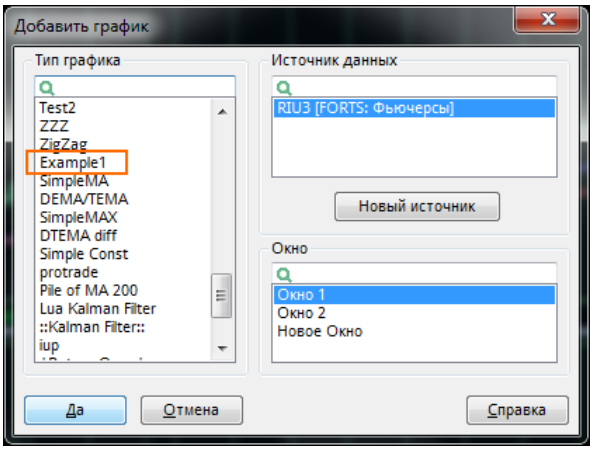
```
Settings=
{
        Name = "Example1"
}
function Init()
   return 1
end

function OnCalculate(index)
   return nil
end
```

Рассмотрим подробнее, что происходит при добавлении такого индикатора на график.

При создании нового индикатора (пункт ***Добавить график (индикатор)...*** контекстного меню графика) терминал сканирует папку LuaIndicators в директории Рабочего места QUIK на наличие в ней скриптов, отвечающих следующим требованиям:

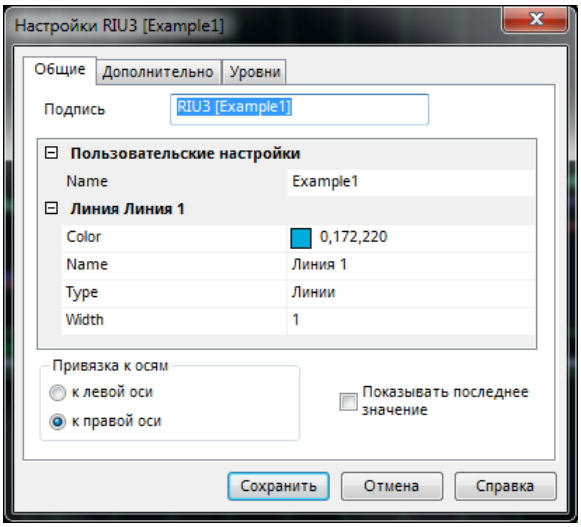
1. В скрипте определена глобальная таблица Lua с именем Settings;
2. Определена функция Init();
3. Определена функция OnCalculate().

Поле Name в таблице Settings будет определять имя индикатора, с которым он будет отображаться в диалоге:



Если скрипт не удовлетворяет перечисленным выше требованиям или содержит синтаксические ошибки языка Lua, то он не будет отображаться в этом диалоге.

Выбрав индикатор Example1 и нажав на кнопку «Да», мы увидим диалог настройки отображения индикатора:



Как видно, значение поля Settings.Name попало в подпись нового индикатора и отображается в поле Name группы «Пользовательские настройки». Также в диалоге свойств индикатора присутствуют параметры одной линии с именем «Линия 1». Функция Init вернула 1, это говорит терминалу, что индикатор будет состоять из одной линии. Так как мы не описали в коде параметры этой линии, то значения полей Color, Name, Type, Width инициализируются значениями по умолчанию.

После нажатия кнопки «Сохранить» мы не увидим никаких новых линий на графике. Потому что функция OnCalculate всегда возвращает nil. Это значение говорит терминалу, что значение индикатора для указанной свечи источника данных не определено.

3. *Изменяем свойства индикатора*

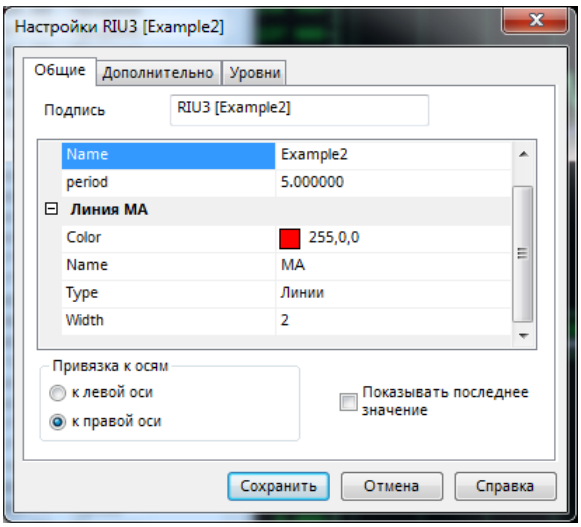
В качестве примера рассмотрим код:

```
Settings=
{
                Name = "Example2",
                period = 5,
                line =
                {
                        {
                          Name = "MA",
                          Color = RGB(255, 0, 0),
                          Type = TYPE_LINE,
                          Width = 2
                        }
                }
}
function Init()
   return 1
end

function OnCalculate(index)
   return nil
end
```

Здесь в таблицу Settings добавились поля period и line. Поле line является массивом таблиц с индексным доступом. Это значит, что все её элементы доступны через численные индексы – line[1], line[2] и т.д.

Наш индикатор возвращает только одну линию, поэтому и свойства описаны только для одной – линии с индексом 1.

Вот как это будет выглядеть в диалоге настроек:



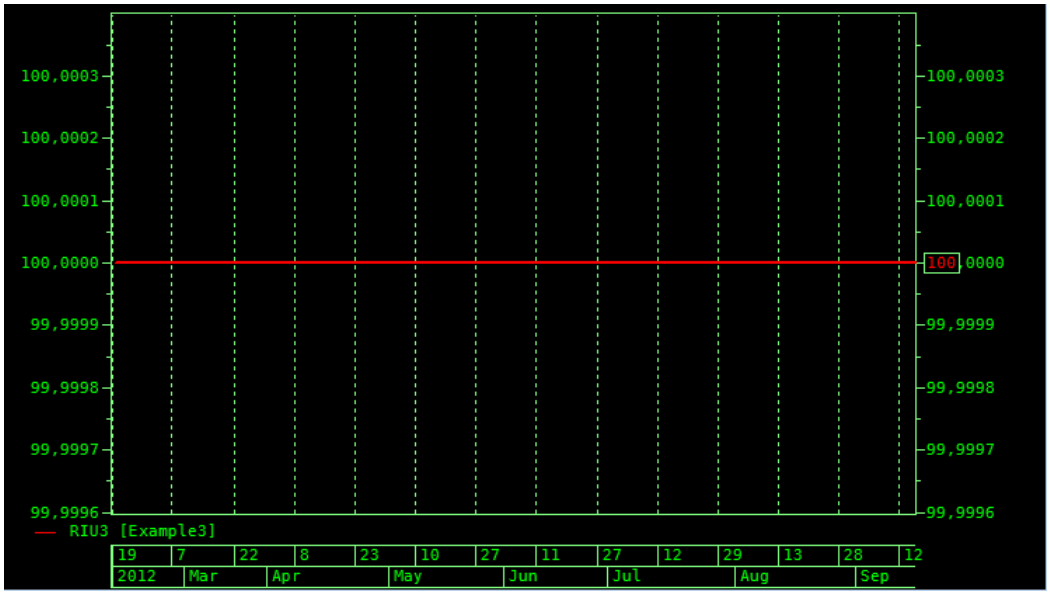
Все, что не относится к описанию параметров линий, попадает в группу «Пользовательские настройки». Тип параметра определяется начальным значением. Поле Name имеет строковый тип, поле period – числовой, так как в коде мы его инициализировали значением 5.

Параметры линии теперь отличаются от значений по умолчанию. Например, цвет линии определяет функция RGB(255, 0 ,0).

4. *Рисуем прямую линию*

Для этого будет достаточно изменить только функцию OnCalculate следующим образом:

```
function OnCalculate(index)
   return 100
end
```

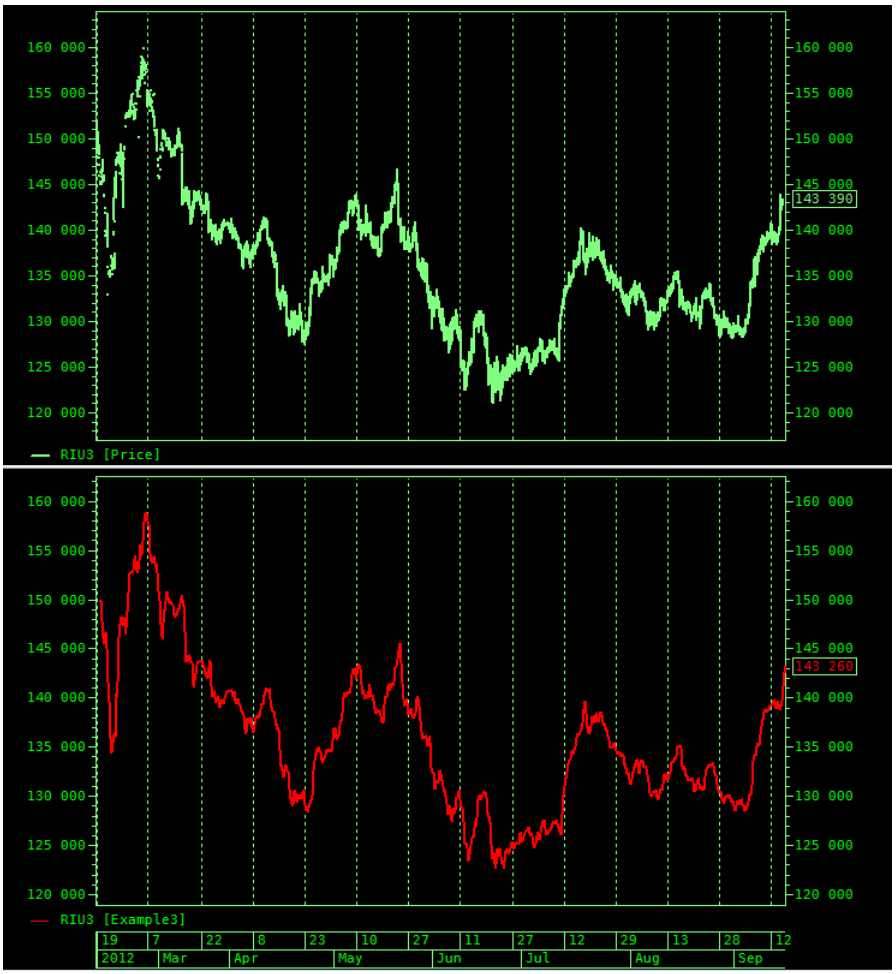


5. *Считаем среднее*

Усложним код, посчитаем среднее значение на заданном нами интервале по ценам закрытия свечи:

```
Settings=
```

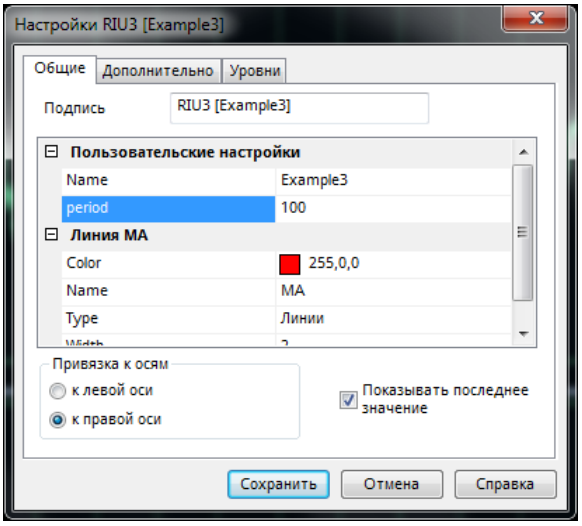
```
{
        Name = "Example3",
        period = 5,
        line =
        {
                {
                        Name = "MA",
                        Color = RGB(255, 0, 0),
                        Type = TYPE_LINE,
                        Width = 2
                }
        }
}
function Init()
   return 1
end

function OnCalculate(index)
   if index < Settings.period then
                return nil
        else
                local sum = 0
                for i = index-Settings.period+1, index do
                        sum = sum +C(i)
                end
                return sum/Settings.period
        end
end
```



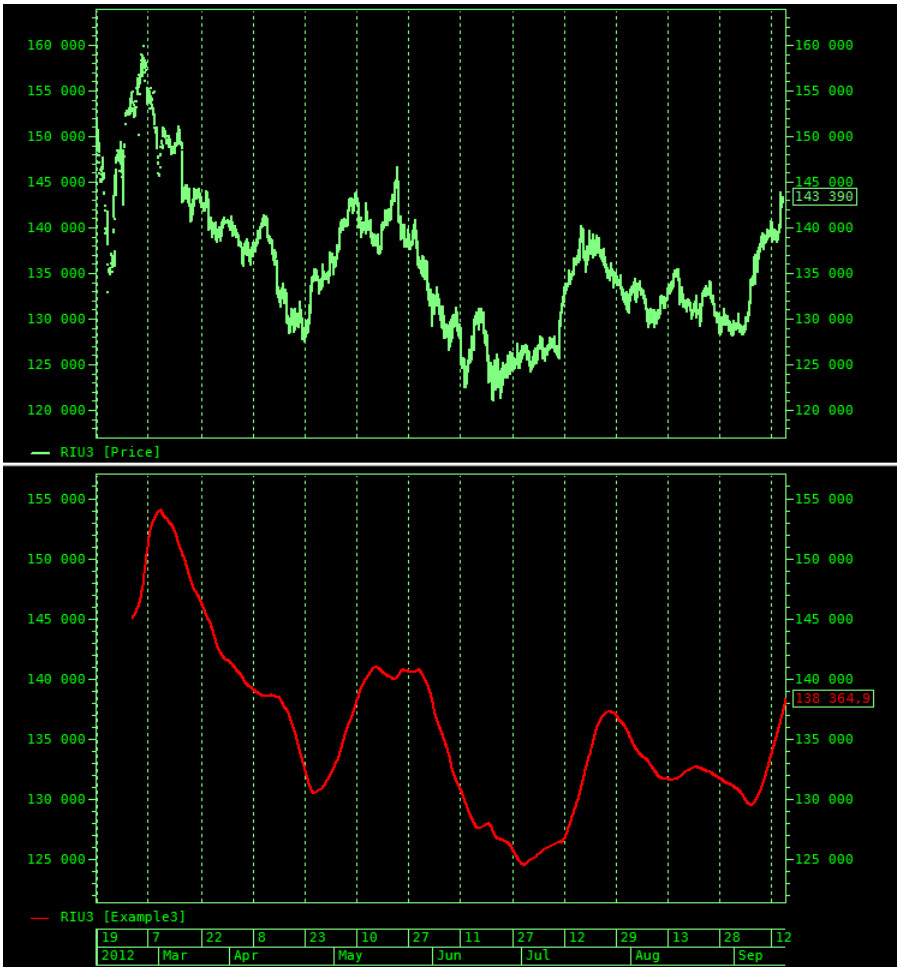
В коде примера есть несколько важных моментов:

1. Проверяем переданный нам индекс свечи. Если он меньше заданного нами периода, то возвращаем nil. Данных для расчёта недостаточно, поэтому значение индикатора не определено на индексах свечек меньших, чем задано в Settings.period.
2. Для расчёта индикатора мы везде используем поле таблицы Settings.period.

Откроем свойства нашего индикатора и поменяем значение поля period:



Сохраняем и видим уже другую картину:



Этот пример иллюстрирует тот факт, что все значения из диалога настроек индикатора после нажатия кнопки «Сохранить» попадают в работающую виртуальную машину Lua и становятся доступны в функциях скрипта. При этом никак не затрагивается исходный код скрипта на диске и

индикаторы уже созданные с его помощью. Если мы ещё раз добавим этот индикатор на график, то получим предыдущую картинку.

6. Доступ к данным

Как говорилось выше, каждый индикатор привязан к источнику данных. Для доступа к данным скрипт может использовать следующие функции:

- O(i);
- H(i);
- L(i);
- C(i);
- V(i);
- T(i).

За исключением функции T(), все они возвращают соответствующее значение для указанного бара – Open, High, Low, Close и Volume. Функция T() возвращает таблицу, которая содержит время указанного бара.

Среднее значение можно рассчитать не только по цене закрытия. Немного усложним код, добавив функцию:

```
Settings=
{
	Name = "Example3",
	period = 5,
	value_type = "C",
	line =
	{
		{
			Name = "MA",
			Color = RGB(255, 0, 0),
			Type = TYPE_LINE,
			Width = 2
		}
	}
}

function dValue(i,param)
	local v = param or "C"

	if		v == "O" then
			return O(i)
	elseif	v == "H" then
			return H(i)
	elseif	v == "L" then
			return L(i)
	elseif	v == "C" then
			return C(i)
	elseif	v == "V" then
			return V(i)
	elseif	v == "M" then
			return (H(i) + L(i))/2
	elseif	v == "T" then
			return (H(i) + L(i)+C(i))/3
	elseif	v == "W" then
			return (H(i) + L(i)+2*C(i))/4
	else
		return C(i)
	end
```

```
end

function Init()
   return 1
end

function OnCalculate(index)
   if index < Settings.period then
                        return nil
                else
                        local sum = 0
                        for i = index-Settings.period+1, index do
                                sum = sum +dValue(i, Settings.value_type)
                        end
                        return sum/Settings.period
                end
end
```

Теперь в диалоге настроек графика видим ещё одну переменную – value_type. Меняя её значение, мы изменяем поведение функции dValue() и, соответственно, входные значения для расчёта среднего значения

7. Рассчитываем EMA

Как известно EMA вычисляется по следующей итерационной формуле:

$$EMA_i = \alpha * P_i + (1 - \alpha) * EMA_{i-1}$$

Мы не можем обратиться напрямую к предыдущим рассчитанным значениям индикатора. Это значит, что для вычислений текущего значения нам придётся хранить и предыдущие значения. Для таких целей в Lua можно использовать очень удобный механизм замыканий. Хорошей идеей будет вынести определение такой функции в отдельный файл и даже в отдельный каталог, который не будет сканироваться при создании индикатора.

Пример файла с функцией расчёта EMA (ma.lua):

```
function round(num, idp)
                if num == nil then return nil end
                local mult = 10^(idp or 0)
                return math.floor(num * mult + 0.5) / mult
end

function dValue(index, v_type)
                v_type = v_type or BAR_CLOSE
                if          v_type == BAR_OPEN then
                        return O(index)
                elseifv_type ==BAR_HIGH then
                        return H(index)
                elseifv_type ==BAR_LOW then
                        return L(index)
                elseifv_type ==BAR_CLOSE then
                        return C(index)
                elseifv_type ==BAR_VOLUME then
                        return V(index)
                end
                return 0
end

function average(_start, _end, v_type)
                local sum=0
                for i = _start, _end do
                        sum=sum+dValue(i, v_type)
                end
```

```
        return sum/(_end-_start+1)
end

function cached_EMA()
        local cache={}
        return function(ind, _p, v_t, kk)
            local n = 0
            local p = 0
            local period = _p
            local v_type = v_t
            local index = ind
            local k = kk or 2/(period+1)
            if index == 1 then
               cache = {}
            end
            if index < period then
               cache[index] = average(1,index, v_type)
               return nil
            end
            p = cache[index-1] or dValue(index, v_type)
            n = k*dValue(index, v_type)+(1-k)*p
            cache[index] = n
            return n
        end
end

function cached_DTEMA()
        local cache_EMA={}
        local cache_DMA={}
        local cache_TMA={}
        return function(ind, _p, v_t, kk)
            local n_ema = 0
            local p_ema = 0
            local n_dma = 0
            local p_dma = 0
            local n_tma = 0
            local p_tma = 0
            local period = _p
            local v_type = v_t
            local index = ind
            local dv = dValue
            local k = kk or 2/(period+1)
            if index == 1 then
               cache_DMA = {}
               cache_EMA = {}
               cache_TMA = {}
            end
            if index < period then
               cache_EMA[index] = average(1,index, v_type)
               return nil
            end
            p_ema = cache_EMA[index-1] or dv(index, v_type)
            n_ema = k*dv(index, v_type)+(1-k)*p_ema
            cache_EMA[index] = n_ema

            p_dma = cache_DMA[index-1] or cache_EMA[index-1]
            n_dma = k*n_ema + (1-k)*p_dma
            cache_DMA[index] = n_dma

            p_tma = cache_TMA[index-1] or cache_DMA[index-1] or cache_EMA[index-1]
            n_tma = k*n_dma + (1-k)*p_tma
            cache_TMA[index] = n_tma
            return round(n_dma, 2), round(n_tma, 2)
        end
end
```

В этом примере, кроме функции cached_EMA, присутствует функция для расчёта DEMA и TEMA.

В папке с терминалом создадим папку Include, куда и сохраним файл ma.lua.

Вот какой вид примет код нашего индикатора с использованием этого файла:

```
dofile(getWorkingFolder() .. "\\Include\\ma.lua")

Settings =
{
                Name = "EMA",
                period = 50,
                value_type = "C",
                line=
                {
                        {
                          Name = "1",
                          Color = RGB(255, 0, 0),
                          Type = TYPE_LINE,
                          Width = 2
                        }
                }
}
function Init()
                myEMA = cached_EMA()
                return 1
end
function OnCalculate(index)
                return myEMA(index, Settings.period, Settings.value_type)
end
```

Функция getWorkingFolder возвращает нам путь папки с файлом info.exe. В примере последний параметр у нас не используется и по умолчанию в функции инициализируется значением 2/(Setting.period+1).

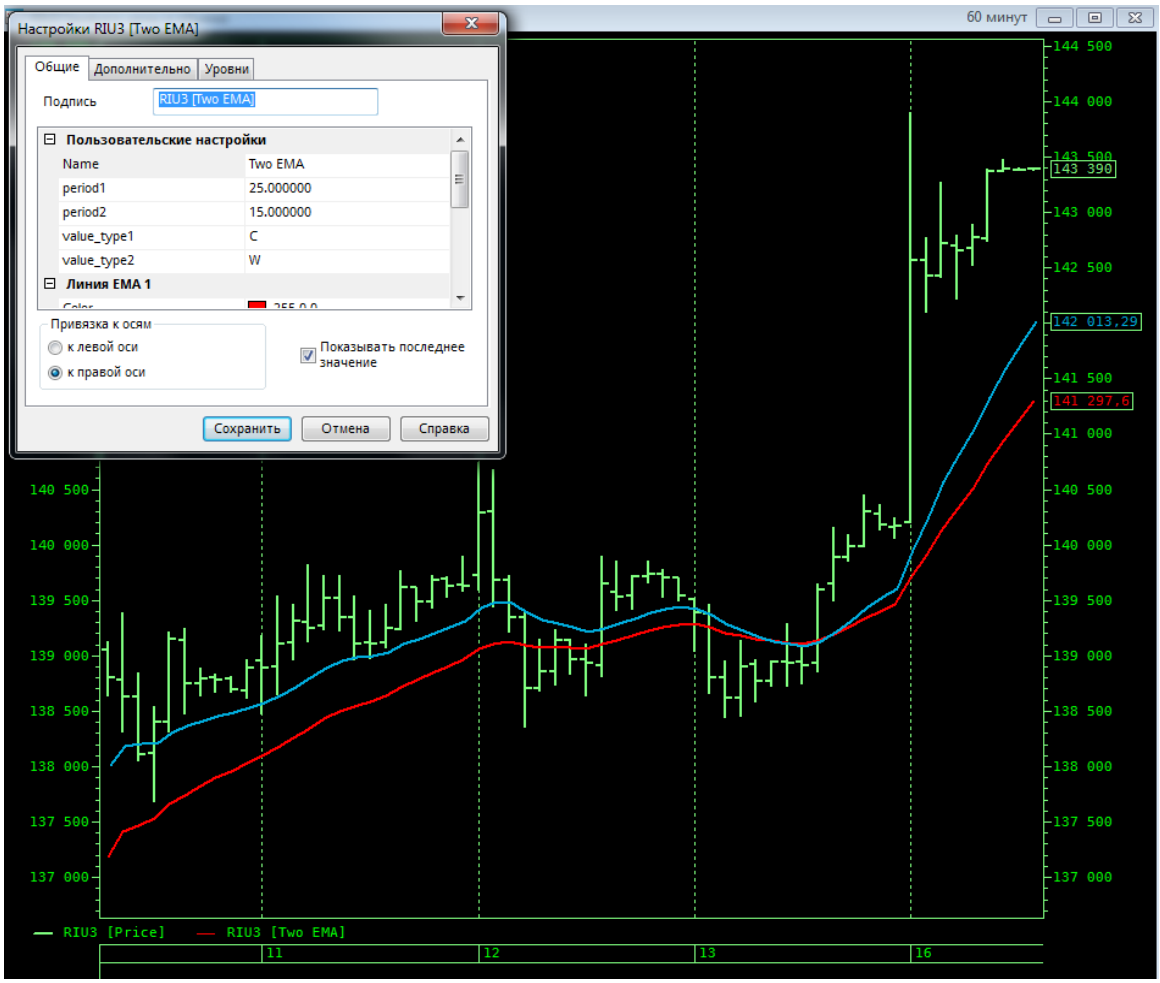
Пример полученного индикатора:

8. Индикатор с несколькими линиями

Изменим код нашего индикатора. Добавим туда ещё одну линию EMA с собственными параметрами:

```
dofile(getWorkingFolder() .. "\\Include\\ma.lua")

Settings =
{
                Name = "Two EMA",
                period1 = 50,
                value_type1 = "C",
                period2 = 50,
                value_type2 = "C",

                line=
                {
                        {
                           Name = "EMA 1",
                           Color = RGB(255, 0, 0),
                           Type = TYPE_LINE,
                           Width = 2
                        },
                        {
                           Name = "EMA 2",
                           Type = TYPE_LINE,
                           Width = 2
                        }

                }
}
function Init()
                myEMA1 = cached_EMA()
                myEMA2 = cached_EMA()
                return 2
end
function OnCalculate(index)
                ema1 = myEMA1(index, Settings.period1, Settings.value_type1)
                ema2 = myEMA2(index, Settings.period2, Settings.value_type2)
                return round(ema1,2), round(ema2,2)
end
```

Вот как это может выглядеть в терминале:



9. *Функция OnDestroy*

Иногда в коде индикатора бывает необходимо не только заниматься расчётами, но и использовать какие-либо системные ресурсы, например файлы. Для того, чтобы понять когда их нужно освободить существует функция OnDestroy.

Если функция OnDestroy определена в скрипте, то она будет вызываться при удалении индикатора или закрытии окна с графика данного индикатора. Пример кода:

```
Settings={}
Settings.Name = "FileOp"
Settings.mode = 0

file = nil

function Log(s)
	local x = tostring(Settings.Name)
	if file ~=nil then
		file:write(x.. " : " .. s .. "\n")
		file:flush()
	end
end
function Init()
	file = io.open(getScriptPath() .. "\\zigzag.log", "a+t")
	Log("Init return 1")
	return 1
end
function OnCalculate(i)
	Log("OnCalculate(" .. i .. ")")
end
```

```
function OnDestroy()
	Log("OnDestroy()")
	if file~=nil then
		file:close()
	end
end
```

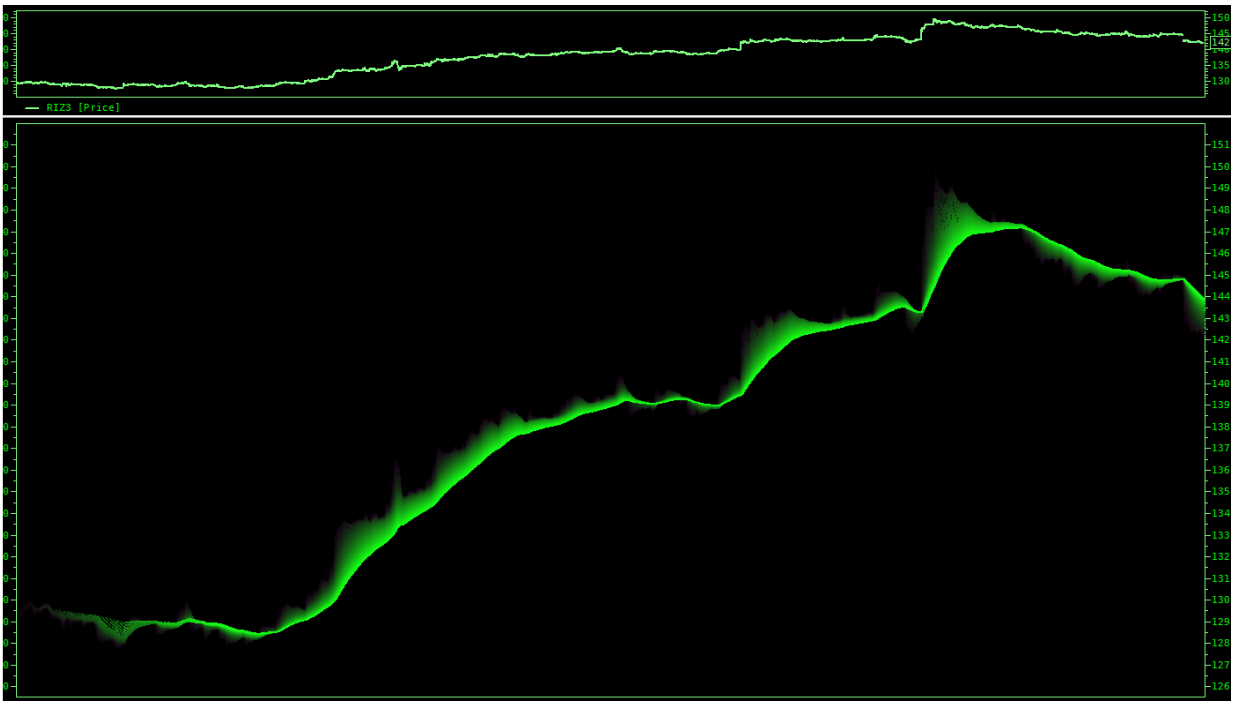
10. Magician birthday

Немного модифицируем функцию cached_EMA:

```
function cached_EMA_Ex(__period, __k)
	local cache={}
	local period = __period
	local k = __k or 2/(period+1)
	return function(ind, v_t)
		local n = 0
		local p = 0
		--local period = _p
		local v_type = v_t
		local index = ind
		if index == 1 then
			cache = {}
		end
		if index < period then
			cache[index] = average(1,index, v_type)
			return nil
		end
			p = cache[index-1] or dValue(index, v_type)
			n = k*dValue(index, v_type)+(1-k)*p
			cache[index] = n
		return n
	end
end
```

и напишем индикатор:

```
dofile(getWorkingFolder() .. "\\Include\\ma.lua")

Settings={}
Settings.StartN = 5
Settings.Nstep = 5
Settings.N = 100
Settings.Name = Settings.N .. " MA"

gtMA={}
function Init()
	Settings.line = {}
	for i = 1, Settings.N  do
		gtMA[i] = cached_EMA_Ex(Settings.StartN + (i-1)*Settings.Nstep, 1/i)
		Settings.line[i] = {}
		Settings.line[i] = {Color = RGB(20, 255/Settings.N*i, 20), Type = TYPE_LINE, Width = 1}
	end
	return Settings.N
end
function OnCalculate(idx)
	local res={}
	for i=1, Settings.N do
		res[i] = gtMA[i](idx,"C")
	end
	return unpack(res)
end
```


RIZ3 [Price]