MQL4 para Novatos I

por X-Trader
01/02/2008

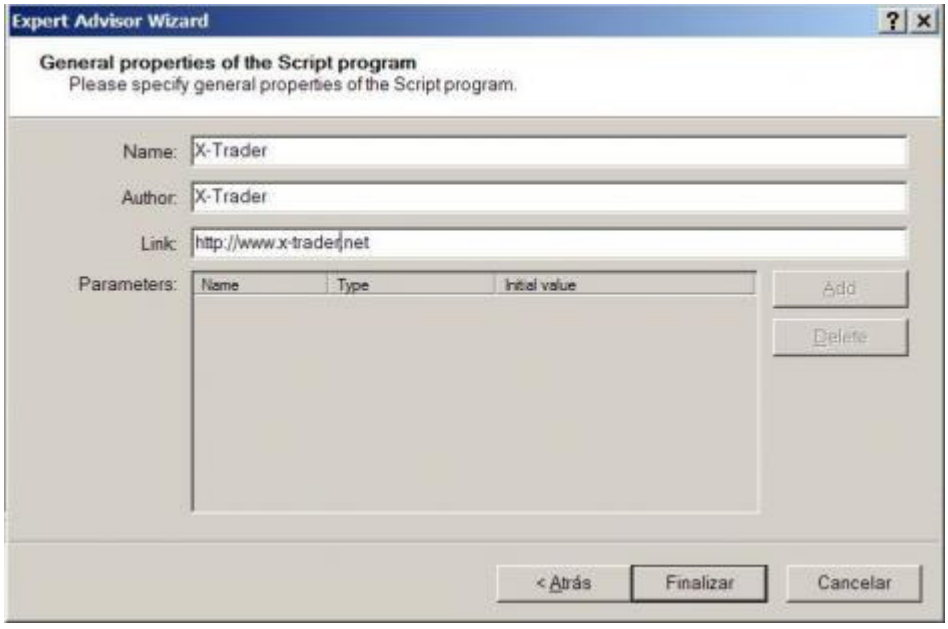
(NOTA: Si no sabe qué es **Metatrader** , le recomiendo que eche un vistazo a este artículo antes de empezar con éste)

Lo primero de todo, recordemos qué podemos hacer utilizando este lenguaje. Con MQL4 podemos crear:

Scripts: se trata de secuencias de comandos que permiten tareas de forma automatizada.
Custom Indicators: se trata de los indicadores que nosotros diseñemos.
Expert Advisors: o lo que es lo mismo, sistemas de trading automáticos.
Libraries: son conjuntos de funciones creadas para desarrollar una tarea específica.

En este primer artículo, vamos a ver cómo se crean sencillos scripts y simultáneamente iremos viendo los fundamentos básicos de este lenguaje. Para ello, utilizaremos el editor de MQL que incorpora Metatrader, denominado MetaEditor 4. Para arrancarlo, basta con entrar en Metatrader y presionar la tecla F4. Una vez dentro de MetaEditor vamos a File->New. Nos aparecerá el siguiente asistente:

Seleccionamos Script y le damos a Siguiente. En la siguiente ventana introducimos los datos que nos pide y le damos a Finalizar:



Con ello inicializamos la ventana del script para empezar a programar en ella:

Observe que hay algunas líneas en gris que vienen precedidas por el símbolo "//". Ello significa que están comentadas y que serán ignoradas de cara a procesar los comandos; en particular, delante de cualquier línea en la que pongamos el símbolo "//" podremos introducir nuestros propios comentarios.

Código Fuente, Compilación y Otras Cosas del Montón
Por si alguno no lo sabe todavía: lo que vamos a escribir en MetaEditor se denomina *código fuente*, es decir, se trata de una secuencia de comandos que nosotros podemos comprender al estar escrito en un lenguaje de programación. Sin embargo, Metatrader no lo comprenderá a menos que lo traduzcamos a su propio idioma. Esta traducción de comandos, denominada *compilación*, se consigue pulsando la tecla F5 en MetaEditor. Una vez compilado el código, se genera un archivo especial que es el que lee Metatrader.

Una vez compilado un script, bastará con que vayamos a Metatrader y entremos en el menú View->Navigator. En la ventana que nos aparecerá a la izquierda del gráfico, haremos click en Scripts y se desplegará la lista de todos los que tengamos. Haciendo doble click en ellos ejecutaremos la secuencia de comandos que hayamos creado.

Bien, volvamos a la ventana de Metaeditor donde teníamos el punto de partida del script. Los comandos que deseemos incluir deben escribirse siempre entre las líneas int start(){ y return(0);}

Variables
Ahora que ya sabemos donde escribir, veamos cómo introducir variables. Básicamente el esquema que debemos usar es el siguiente:

[tipo variable] [nombre variable] = [valor variable];

Dentro de los tipos de variables tenemos los siguientes: int (entero), double (doble precisión), string (cadena de texto) y bool (booleano).

Veamos un ejemplo: si deseamos definir el tamaño de un stop para el EUR/USD. Para ello, podemos escribir:

```
double stoplosseurus = 0.0025;
```

En algunos casos, puede que no sea necesario asignar un valor inicial a la variable; así por ejemplo podemos escribir simplemente:

```
double stoplosseurus ;
```

Y asignarle posteriormente un valor.

Conviene recordar que en MQL4 se distingue el uso de mayúsculas y minúsculas por lo que las siguientes expresiones:
`double HIGHPRICE; double highprice; double HighPrice; double highPrice;`

declararán cuatro variables diferentes.

Asimismo, los nombres de las variables no pueden comenzar por números o símbolos.

Por último hay que tener en cuenta que las palabras del lenguaje MQL4 deben ir siempre en minúsculas. Por ejemplo, las expresiones:

```
DOUBLE hightPrice1;  Double hightPrice2;
```

producirán un error al compilar.

Una vez declaradas las variables, veamos qué se puede hacer con ellas:

Operaciones aritméticas

```
//Declaramos variables para el ejemplo
double a = 50.0;
double b = 2.0;
double c;

// Suma(+): en la variable c sumamos los valores de a y b, obtenemos el valor 52.0
c = a + b;

// Resta(-): en la variable c restamos los valores de a y b, obtenemos el valor 48.0
c = a - b;

// Multiplicación(*): en la variable c multiplicamos a por b, obtenemos el valor 100.0
c = a*b;

// División(/): en la variable c dividimos el valor de a por b, obtenemos el valor 25.0
c = a / b;

// También podemos combinar operaciones tal que c ahora valdría 104.0
c = (a + b)*b;
```

Generación de ventanas con mensajes
Todas las operaciones que acabamos de ver son realizadas correctamente cuando ejecutamos este script en Metatrader. Sin embargo, al ejecutarlo no tendremos ninguna confirmación visual de que todo va bien. Para ello, necesitamos utilizar una función muy útil, MessageBox().

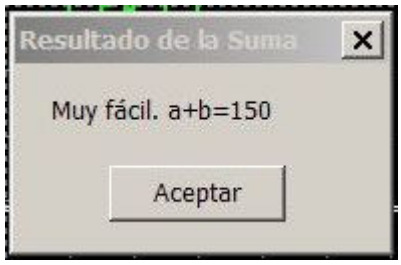
Mediante dicha función podemos generar ventanas con mensajes informativos del estado de una secuencia de comandos. El esquema para utilizar esta función es el siguiente:

MessageBox("Mensaje que queremos mostrar.","Título de la Ventana");

Veamos un ejemplo de cómo podríamos utilizar esta función con el siguiente código:

```
int a = 50;
int b = 100;
MessageBox("Muy fácil. a+b=" + (a + b), "Resultado de la Suma");
```

Como resultado obtenemos la siguiente ventana:



Vectores (Arrays)
Pasamos a ver cómo se declaran vectores en MQL. El esquema para ello es el siguiente:

(Tipo de array) (Nombre del array) [Nº de elementos];

Por ejemplo, la sentencia:

```
double price[5];
```

nos creará un vector de 5 elementos de tipo Double.

¿Cómo podemos hacer referencia a cada uno de los componentes del array y darles un valor? Muy fácil: una vez hemos declarado el array "price" del ejemplo anterior, podemos dar un valor a su primer elemento de la siguiente forma:

```
price[0] = 1.2341;
```

Obsérvese que los índices de los arrays empiezan en 0, de tal forma que dicho valor se corresponde con el primer elemento del

array. Del mismo price[1] = 1.2321; nos asignaría un valor al segundo elemento del array y así sucesivamente.

Por supuesto, podemos declarar un array asignándole valores iniciales:

```
double price[2] = {1.2234, 1.2421};
```

Aunque también es posible no especificar el número de elementos y dejar que Metatrader lo determine:

```
double price[] = {1.2234, 1.2421};
```

Asimismo, con los arrays también podemos realizar las mismas operaciones que hacíamos con las variables como se muestra en los siguientes ejemplos:

```
double price[2];
price[0] = 1.2234;
price[1] = 1.2421;
MessageBox("El precio medios es " + (price[0] + price[1]) / 2.0,"Precio medio");
```

Variables y Arrays de Serie
En el lenguaje MQL, existe una serie de variables y arrays que vienen de serie y que nos serán de gran utilidad a la hora de crear nuestro propio código:

```
double Open[];  // array de precios de apertura
double High[]; // array de máximos
double Low[]; // array de mínimos.
double Close[]; //array de precios de cierre
double Volume[]; // array de volúmenes.
double Bid  // Último precio de compra
double Ask  // Último precio de venta
```

Por supuesto, podemos hacer referencia a un determinado valor del array; así, por ejemplo, High[0] hace referencia al último máximo del precio. El siguiente gráfico seguramente termine de aclararles del todo el asunto:

Cómo pueden observar cada barra está indexada, siendo la barra 0 la última, la barra 1 la penúltima, etc.

Bucles
Veamos ahora como crear una estructura de bucle (*loop*) en MQL. Recordemos que este tipo de estructura puede ser tipo For o While; ésta última la veremos en próximos artículos.

La estructura general de un bucle de tipo For en MQL es la siguiente:

```
for(Declaración de Contador; Condiciones para el bucle; Incremento del contador)
  {
    Código que se ejecutará en el bucle
  }
```

Veamos en detalle la estructura anterior:

Declaración del Contador: siempre utilizaremos una variable de tipo int para el contador, que normalmente siempre será inicializada en 0.
Condiciones para el bucle: aquí indicaremos la condición o condiciones que, en caso de verificarse, permitirán que el bucle continúe ejecutándose; en caso contrario, el bucle finalizará.
Incremento del contador: aquí indicaremos en qué cantidad se incrementará el contador cada vez que termine un bucle. Normalmente, salvo que sea un caso especial, añadiremos una unidad al contador lo que se consigue añadiendo dos signos positivos (++) detrás del nombre de la variable que actúe como contador.

Seguramente un ejemplo aclare todo esto: supongamos que queremos calcular la media de todos los máximos que hay en el gráfico; lo haríamos del siguiente modo:

```
//Declaramos una variable para almacenar los máximos y la media

double CumHigh = 0.0;
double AveragePrice=0.0;

// Iniciamos el bucle: damos valor inicial 0 a la variable contador a, produciéndose el final del bucle se alcance el
// número total de barras que haya en el gráfico (Bars) y sumando una unidad a en cada iteración.
// En la variable CumHigh sumamos la barra de a periodos atras.

for(int a = 0; a < Bars; a++)
{ CumHigh += High[a]; }

// Calculamos la media dividiendo la suma total entre el nº de barras

AveragePrice = CumHigh/Bars;
```

Condiciones
Veamos ahora cómo se construyen condiciones en MQL. Básicamente la estructura es similar a la de otros lenguajes, utilizando la palabra If, por lo que un ejemplo debería ser suficiente para entenderlo:

```
int a = 10;
int b = 0;
if(a > 10 )
{      b = 1;    }
else
{      b = 2;    }
MessageBox("b=" + b,".");
```

En el ejemplo anterior, la condición viene entre paréntesis después de la palabra If. Dado que la condición es falsa (ya que a es igual a 10), al final del script, el valor de b será 2 ya que si no se verifica la condición pasamos al valor Else, que es el que nos determina qué sucederá si a es menor o igual a 10.

En MQL podemos utilizar diferentes operadores para realizar comparaciones: == (igual), != (distinto), > (mayor) , < (menor) , >= (mayor o igual) y <= (menor o igual).

Veamos un ejemplo en el que ahora se incluye un bucle dentro de una condición:

```
int a=0;
double b=0.0;
bool e;
if(a==0)
{ for(int c=0;c<Bars;c++)
{ b+=High[c]; }
if(b>500.0)
{ e=true; }
else
{ e=false; } }
```

El código anterior suma los máximos del gráfico si a es igual a 0; posteriormente si la suma es superior a 500.0 devuelve el valor verdadero mediante la variable e, o falso en caso contrario.

Por supuesto, las condiciones se pueden superponer unas sobre otras como en el siguiente ejemplo:

```
int a=0;
int result;
if(a==0)
{ result=1; }
else if(a==1)
{ result=2; }
else if(a==2)
{ result=3; }
```

```
else { result=4; }
```

Las condiciones se pueden complicar aún más introduciendo los valores lógicos && (AND) y || (OR). Por ejemplo:

```
int a=10;
int b=20;
if ( ((a>5) || (a<15)) && ( b==20))
MessageBox("Me ha salido bien!","Ejemplo del uso de AND y OR");
```

En el próximo artículo veremos cómo construir complejas expresiones así como algunas aplicaciones.

(*Continuará...*)

Un saludo,
X-Trader

MQL4 para Novatos II

por X-Trader
10/02/2008

Bucles de tipo *while*

Las estructuras de tipo *while* pueden sernos útiles en muchas ocasiones. Su forma general es:

```
while(condición)
{   código a ejecutar mientras se dé la condición    }
```

La principal diferencia de este tipo de estructura con los bucles de tipo *for* es que no existe una variable contador. Por tanto, si no necesitamos un contador podemos ahorrarnos trabajar con *for* y usar una bucle *while*. A medida que avancemos en la serie iremos viendo ejemplos de este tipo de estructura.

Un nuevo tipo de condiciones. El operador *switch*.
Con el operador *switch*, podemos ahorrarnos complejas estructuras compuestas por varios *if* y *else*. Fundamentalmente la estructura de tipo *switch* es utilizada cuando necesitamos realizar una tarea en función del valor de una variable. Por ejemplo, supongamos que deseamos crear un sistema cuyo comportamiento varía en función de la situación en qué esté el mercado. Así, declaramos la variable que toma valores enteros, marketState:

```
int marketState
```

Que puede tomar los siguientes valores:

1, si el mercado es alcista
2, si el mercado es bajista
3, si el mercado está sin tendencia

Si ahora creáramos el sistema combinando varios *if* y *else* tendríamos que:

```
if (marketState == 1)
{      // Estrategia Mercado Alcista    }
else       if(marketState == 2)
{          // Estrategia Mercado Bajista      }
else           if(marketState == 3)
{              // Estrategia Mercado sin Tendencia           }
else           {              // Error: no se consideran otras posibilidades           }
```

Como podemos ver, todas las condiciones dependen de una única variable y en todas se compara su valor con un significado asociado a él. En este tipo de situaciones, las estructuras de tipo *switch* son mucho más eficaces:

```
switch(marketState)
{
case 1:  // Estrategia Mercado Alcista
break;
case 2:  // Estrategia Mercado Bajista
break;
case 3:  // Estrategia Mercado sin Tendencia
break;
default: // Error: no se consideran otras posibilidades
break;
}
```

Obsérvese en el ejemplo anterior que hemos definido previamente la variable de estado marketState.

En general, las estructuras de tipo *switch* presentan la siguiente forma:

```
switch(variable de estado)
{      case [valor de la variable]:      // Código a ejecutar para este caso
       break;
       case [otro valor de la variable]: // Código a ejecutar para este caso
       break;
       default:     // Código para el resto de casos
       break;     }
```

Más operadores: *continue* y *break*
Acabamos de ver en la estructura *switch* el uso del operador *break*. Pero *break* puede utilizarse no sólo dentro de una estructura

de ese tipo sino en general para salir de cualquier tipo de bucle. Por ejemplo, supongamos que necesitamos saber cuál es la última barra hasta la que el volumen negociado acumulado ha sido mayor o igual a 1000. Para ello podemos utilizar el siguiente código:

```
int a = 0;
double volume = 0.0;
while(a < Bars)
{
// Si la variable volume supera 1000 points, salimos del bucle while
if(volume > 1000.0)
break;
volume = volume + Volume[a];
// Alternativamente para acumular valores en una variable también
// podemos escribir también volume += Volume[a];
a++;    }
```

El otro operador relacionado con este tipo de tareas es *continue*, el cual se utiliza para omitir iteraciones no deseadas. Supongamos que queremos obtener el volumen total de las barras mostradas en el gráfico pero ignorando aquellas barras cuyo volumen sea superior a 50. Para ello, utilizaríamos el siguiente código:

```
int a = -1;
double volume = 0.0;
while(a < Bars)
{      a++;
// Si el volumen supera el valor 50, omitimos esa barra
if(Volume[a] > 50.0)
continue;
volume += Volume[a];    }
```

Escribiendo nuestras propias funciones
Normalmente nos encontraremos con situaciones en las que veremos que se repite el código varias veces. Para ahorrarnos tiempo tecleando podemos crearnos funciones para llamarlas en los momentos que las necesitemos. Veamos un ejemplo para entenderlo.

Supongamos que deseamos saber el color de la última vela aparecida en el gráfico. Para ello, declaramos una variable booleana (tan sólo hay dos casos posibles, blanco o negro) tal que:

```
bool color;
// Suponemos que color = false se corresponde con una vela negra
if(Close[0] > Open[0])
color = true;
if(Open[0] > Close[0])
color = false;
```

Si quisieramos saber el color de la vela en otra región del gráfico (por ejemplo, hace 5 velas), deberíamos reemplazar [0] por el valor correspondiente. Pero qué pasaría si necesitáramos hacerlo para todas las velas del gráfico? Es en este tipo de situaciones en las que necesitamos recurrir a funciones. Vamos a crear una función que nos permitirá determinar el color de cualquier vela:

```
bool color;
color = GetColor(0);
```

Como podemos ver, en lugar de imponer la condición para obtener el valor de la variable color, lo que hacemos es crear una función que nos devolverá el valor (true/false - blanco/negro) y que se llamará GetColor. Obsérvese que posee un argumento (el cero entre paréntesis) hace referencia a la última barra del gráfico.

La estructura general de una función en MQL4 es la siguiente:

```
[tipo de valor devuelto] [nombre de la función] ([lista de argumentos])
{      // código de la función
       return([valor a devolver]);    }
```

Veamos cómo escribiríamos la función GetColor:

```
bool GetColor(int index)
{      bool color;
if(Close[index] > Open[index])
color = true;
if(Open[index] > Close[index])
color = false;
return(color);    }
```

Seguramente les llame la atención la primera línea, en la que aparece int index entre paréntesis. Sin embargo, cuando llamamos a la función, habíamos escrito simplemente GetColor(0). Qué es lo que ocurre? Sencillamente que cuando hemos escrito la función, hemos definido con la variable index el tipo de argumento que recibirá la función (en este caso de tipo int – Entero).

Por supuesto, si necesitaramos más argumentos porque estuviéramos escribiendo una función más compleja, simplemente bastaría con introducirlos separados por comas. Por ejemplo:

```
bool FuncionComplicada(int Argumento1, int Argumento2, string Argumentostring)
```

Si quisieramos que la función no devuelva ningún valor utilizaremos el comando void:

```
void function()    {      // code    }
```

También podemos establecer valores por defecto para los argumentos de la función. Por ejemplo:

```
void FuncionA(int argumento1, int argumento2, int argumentoespecial = 50)
{      // code    }
```

Si tuvieramos que llamar a la función FuncionA, no sería necesario pasarle un valor al tercer argumento (argumentoespecial) ya que tiene asignado un valor igual a 50 por defecto.

En todo caso, debemos recordar que los argumentos que tengan valores asignados por defecto deben aparecer siempre al final de la lista de argumentos.

Arrays multidimensionales
En MQL4 también podemos trabajar con arrays de varias dimensiones, es decir, matrices de datos. Un ejemplo de array multidimensional sería el siguiente:

[0] 10 20 30

[1] 40 50 60

[2] 70 80 90

[0] [1] [2]

Los números que aparecen entre corchetes indican cómo son los índices utilizados para referenciar filas y columnas en MQL4. Así, el elemento [0] [1] sería 20, el elemento [1] [2] sería 60 y así sucesivamente.

Para crear una matriz como la anterior en MQL4, tenemos el comando array2D[num filas] [num columnas] tal que:

```
int array2D[3][3]=
{10,20,30,
40,50,60,
70,80,90};
```

También podemos crear arrays en 3D mediante el siguiente comando:

```
int array3D[num x][num y][num z] = {lista de números separada por comas};
```

Algunas funciones adicionales para trabajar con arrays serían las siguientes:

Obtener el número de elementos de un array: `int ArraySize(objeto array[]);`
Asignar valores a un array: `int ArrayInitialize(objeto array[],double valor);`
Valor máximo y mínimo de un array:

```
int ArrayMaximum(double array[], int count = WHOLE_ARRAY, int start = 0);
int ArrayMinimum(double array[], int count = WHOLE_ARRAY, int start = 0);
```

Calcular la dimensión de un array: `int ArrayDimension(objeto array[]);`
Ordenar los elementos de un array:

```
int ArraySort(double&array[], int count = WHOLE_ARRAY, int start = 0, int sort_dir = MODE_ASCEND);
```

Copiar un array dentro de otro:

```
int ArrayCopy(object&dest[], object source[], int start_dest = 0, int start_source=0, int count=WHOLE_ARRAY);
```

Preprocesadores
Terminamos la entrega de hoy con lo que se conoce como preprocesador. Veamos en qué consisten con un sencillo ejemplo. Hace algunos párrafos veíamos la siguiente estructura de tipo *switch*:

```
switch(marketState)
{
case 1:  // Estrategia Mercado Alcista
break;
case 2:  // Estrategia Mercado Bajista
break;
case 3:  // Estrategia Mercado sin Tendencia
break;
default: // Error: no se consideran otras posibilidades
break;
}
```

Qué les parecería poder cambiar los valores 1, 2 y 3 por las palabras Alcista, Bajista y Plano? Realmente el código sería aún más comprensible, verdad?

Pues bien, para poder hacer eso simplemente basta con definir preprocesadores de la siguiente forma: al comienzo del código, insertaríamos las siguientes líneas:

```
#define ALCISTA 1
#define BAJISTA 2
#define PLANO 3
```

Como podemos observar, los preprocesadores siempre van precedidos del símbolo #.

Mediante este código lo que hacemos es que, por ejemplo, cada vez que aparezca la palabra ALCISTA en el código a compilar, ésta será reemplazado por el valor 1.

Para comprobarlo, creen un nuevo script e introduzcan el siguiente código para luego ejecutarlo:

```
//+------------------------------------------------------------------+
//|                                                Preprocesador.mq4 |
//|                                                    X-Trader.net  |
//|                                                                  |
//+------------------------------------------------------------------+

#property copyright "X-Trader.net"
#property link      "http://www.x-trader.net"

#define ALCISTA  1
#define BAJISTA 2
#define PLANO    3

//+------------------------------------------------------------------+
//| script program start function                                    |
//+------------------------------------------------------------------+

int start()
  {
   MessageBox("ALCISTA=" + ALCISTA + " BAJISTA=" + BAJISTA +
              " PLANO=" + PLANO);
   return(0);
  }
```

(*Continuará...*)

Un saludo,
X-Trader

Cerrar ventana

MQL4 para Novatos III

por X-Trader
28/03/2008

Retomamos la serie sobre programación en MQL4. En esta tercera entrega pasamos a ver las principales funciones matemáticas de que disponemos en este lenguaje, así como algunas funciones adicionales para mostrar información en pantalla.

FUNCIONES MATEMÁTICAS

MathAbs (Valor Absoluto)
Esquema: double MathAbs(double value)
Ejemplo:

```
int a=-10;
a=MathAbs(a); // ahora a vale 10
```

MathCeil (Redondeo por exceso)
Esquema: double MathCeil(double x)
Ejemplo:

```
double a;
a=MathCeil(1.001); // dado que se redondea por exceso, a valdrá 2.0
```

MathFloor (Redondeo por defecto)
Esquema: double MathFloor(double x)
Ejemplo:

```
double a;
a=MathFloor(1.999); // dado que se redondea por defecto, a valdrá 1.0
```

MathRound (Redondeo normal: si la parte decimal es superior a 0.5, se redondea por exceso, en caso contrario por defecto)
Esquema: double MathRound(double x)
Ejemplo:

```
double a;
a=MathFloor(1.1); // al ser 0.1 menor que 0.5, a valdrá 1.0

double b;
b=MathFloor(1.8); // al ser 0.8 mayor que 0.5, a valdrá 2.0
```

MathMax (Máximo de una serie)
Esquema: double MathMax(double valor1, double valor2)
Ejemplo:

```
double a;
a=MathMax(50.0,1.0); // a toma el valor 50.0
```

MathMin (Mínimo de una serie)
Esquema: double MathMin(double valor1, double valor2)
Ejemplo:

```
double a;
a=MathMin(10.0,12.0); // a toma el valor 10.0
```

MathPow (Potencia de un número)
Esquema: double MathPow(double base, double exponente)
Ejemplo:

```
double a;
a=MathPow(5.0,2.0); // a toma el valor 25, es decir, 5 elevado a 2
```

MathSqrt (Raíz cuadrada de un número)
Esquema: double MathSqrt(double x)
Ejemplo:

```
double a;
a=MathSqrt(9.0); // a es igual a 3, raíz cuadrada de 9
```

MathLog (Logaritmo neperiano de un número)
Esquema: double MathLog(double x)
Ejemplo:

```
double a;
a=MathLog(10.0); // El valor de a es 2.30258509
```

MathExp (Potencia del número e)
Esquema: double MathExp(double d)
Ejemplo:

```
double a;
a=MathExp(1); // El valor de a es 2.7182818
```

MathMod (Resto de una división)
Esquema: double MathMod(double Numerador, double Denominador)
Ejemplo:

```
double a;
a=MathMod(5,2); // Si dividimos 5 entre 2, el resto que queda es a=1.0
```

MathRand (Generación de números aleatorios)
Esquema: int MathRand() (esta función no tiene argumentos)
Ejemplo: si bien por defecto utiliza el rango de enteros comprendidos entre 0 y 32767, podemos fijar el intervalo en el que queremos obtener los números de la siguiente forma:

```
MathRand()%6  // Devuelve enteros aleatoriamente entre 0 (valor 1) y 5 (valor 6).

MathRand()%6+5  // Devuelve enteros aleatoriamente entre 5 (valor 6) y 10 (valor 11)

(MathRand()%11+10)*(-1) // Obtenemos enteros aleatorios entre -10 y -20

(MathRand()%1001/1000.0) // Obtenemos valores entre 0 y 1 con tres decimales
```

MathSrand (Generación de números enteros aleatorios iniciando la secuencia en un valor determinado o semilla)
Esquema: void MathSrand(int semilla)
Ejemplo: Esta función puede combinarse de forma muy efectiva con la función TimeLocal, que devuelve el número de segundos transcurridos desde el 1 de Enero de 1970 tal que:

```
MathSrand(TimeLocal());
```

Con ello, conseguimos dar un valor único a la semilla de partida.

Funciones Trigonométricas
En MQL tenemos disponibles las funciones de seno, coseno y tangente así como sus inversas, esto es, arcoseno, arcocoseno y arcotangente. El esquema de utilización para cada una de ellas es, respectivamente:

double MathSin(double x)
double MathCos(double x)
double MathTan(double x)
double MathArcsin(double x)
double MathArccos(double x)
double MathArctan(double x)

Todas ellas funcionan en base a radianes por lo que conviene convertir cualquier número en grados a radianes antes de introducirlo en estas funciones, de la forma:

Radianes = (Grados * Pi)/180

Suele ser conveniente declarar el número Pi como una variable al comienzo de cualquier código con estas funciones. Para ello escribiremos:

```
#define PI 3.1415926535897
```

OTRAS FUNCIONES PARA MOSTRAR MENSAJES
Hasta ahora sólo habíamos visto la función MessageBox para mostrar información en pantalla. Veamos algunas funciones más.

Alert (Emitir una alerta)

Esquema: void Alert(...)
Ejemplo:

```
Alert("Signal Type:"+signaltype);
```

Recordemos que el sonido de las alertas puede configurarse desde el menú Tools->Options dentro de la pestaña Events.

Comment (Muestra un mensaje en la esquina superior izquierda del gráfico)
Esquema: void Comment(...)
Ejemplo: si escribimos por ejemplo:

```
Comment("Recuerda que tiene soporte en 1.5300");
```

Al ejecutar el código que contiene esta instrucción, nos aparecerá ese comentario en la parte superior izquierda del gráfico.

Print (Permite mostrar mensajes en el registro de un Expert Advisor)
Esquema: void Print(...)
Ejemplo: si queremos que cuando carguemos un Expert Advisor se nos muestre alguna información en pantalla, utilizaremos esta función; por ejemplo:

```
Print("EA cargado correctamente");
```

(Continuará...)

Un saludo,
X-Trader

MQL4 para Novatos IV

por X-Trader
06/04/2008

Continuamos con la serie sobre el lenguaje MQL. En esta ocasión se abordan las principales funciones de indicadores técnicos que trae incorporadas de serie. Todas las funciones que vamos a ver a continuación constan casi siempre de los siguientes argumentos:

symbol – define sobre qué par de divisas o valor se aplicar el indicador. Si queremos que pueda usarse en cualquier gráfico daremos valor NULL (o 0) a este parámetro. Si queremos aplicarlo a un producto determinado deberemos escribir su nombre como una cadena ("EURUSD","GBPUSD" etc.).

timeframe – en el segundo argumento se define la escala temporal en la que se va a utilizar el indicador. Si queremos que se utilice el periodo del gráfico sobre el que insertamos el indicador simplemetnte daremos valor 0 a este parámetro. Si queremos especificar el periodo, deberemos utilizar una de las siguientes constantes:

PERIOD_M1 - 1 minuto
PERIOD_M5 - 5 minutos
PERIOD_M15 - 15 minutos
PERIOD_M30 - 30 minutos
PERIOD_H1 - 1 hora
PERIOD_H4 - 4 horas
PERIOD_D1 - 1 día
PERIOD_W1 - 1 semana
PERIOD_MN1 - 1 mes

shift – el último argumento define cuál será el último valor del indicador que se mostrará en pantalla. Normalmente querremos que nos muestre el último valor del indicador por lo que le asignaremos valor 0, pero si queremos que nos muestre valores anteriores del indicador en la última barra (esto es, queremos retardar el indicador) le asignaremos el valor 1 para mostrar el valor del indicador en la barra anterior, 2 para mostrar su valor hace dos barras, etc.

Asimismo en los indicadores es habitual utilizar valores calculados a partir de los campos que definen una barra, para después promediarlos utilizando un método determinado. Para ello se utilizan los siguientes argumentos:

applied_price – define el campo que utilizaremos del precio; para ello utilizaremos las siguientes constantes:

PRICE_CLOSE - Cierre
PRICE_OPEN - Apertura
PRICE_HIGH - Máximo
PRICE_LOW - Mínimo
PRICE_MEDIAN – Precio Medio definido como (Máx+Mín)/2
PRICE_TYPICAL – Precio Típico, calculado como (Máx+Mín+Cierre)/3
PRICE_WEIGHTED – Precio ponderado, obtenido a partir de la siguiente fórmula: (Máx+Mín+Cierre+Cierre)/4

ma_method – permite definir el método utilizado para calcular una media móvil. Admite los siguientes valores:

MODE_SMA – Media móvil simple
MODE_EMA - Media móvil exponencial
MODE_SMMA - Media móvil suavizada
MODE_LWMA - Media móvil ponderada linealmente

period – define cuántas barras se incluiran para calcular la media

ma_shift – permite desplazar la media un número definido de barras. Si el valor introducido es positivo desplazará la media a la derecha el número de barras que hayamos indicado; por el contrario, si el valor es negativo, la media es desplazada a la izquierda.

Las funciones de indicadores que vamos a ver a continuación pueden agruparse en dos categorías: simples, cuando sólo constan de una línea, o complejas, cuando constan de varias líneas. En este último caso, se utilizará el parámetro mode como veremos más adelante..

Pasamos a continuación a ver todas las funciones de indicadores técnicos que incorpora este lenguaje, la forma de utilizarlos y un ejemplo:

Aceleración/Deceleración (AC)
Este indicador mide la velocidad a la que varían los precios.

Esquema: double iAC(string symbol, int timeframe, int shift)

Ejemplo:
```
double ac;
ac=iAC(0,0,0); // AC aplicado a la última barra en cualquier gráfico de cualquier escala temporal. ac=iAC
("GBPUSD",PERIOD_M30,0);
// AC aplicado a la última barra del gráfico de 30 min. del GBPUSD.
```

Acumulación/Distribución (A/D)
Indicador utilizado para confirmar la tendencia de los precios utilizando el volumen. Debe tener en cuenta que, en el caso de las divisas, Metatrader muestra el volumen negociado por el broker en su red de negociación por lo que no tiene porque ser representativo del volumen negociado a nivel mundial.

Esquema: double iAD(string symbol, int timeframe, int shift)

Ejemplo:
```
double ad;
ad=iAD("GBPUSD",PERIOD_M5,5); // A/D de hace 6 barras en el gráfico de 5 min. del GBPUSD
```

Alligator
Indicador desarrollado por Bill Williams que combina 3 medias móviles

Esquema:

double iAlligator(string symbol, int timeframe, int jaw_period, int jaw_shift, int teeth_period,

int teeth_shift, int lips_period, int lips_shift, int ma_method, int applied_price,

int mode, int shift)

Obsérvese que aparte de los habituales symbol, timeframe y shift esta función para controlar el periodo y el desplazamiento de cada una de las tres líneas - mandíbula (jaw), dientes (teeth) y labios (lips) - de que se compone el Alligator.

Asimismo incorpora los parámetros para seleccionar el tipo de media aplicada (ma_method) y el campo del precio sobre el que se aplicará (applied_price).

Finalmente si miramos la estructura tenemos un parámetro adicional del que por ahora sólo habíamos hecho mención: mode. Las constantes que admite este parámetro varían dependiendo de la función que la utilice y sirve generalmente para seleccionar, en el caso de los indicadores de tipo complejo, la línea con la que queremos trabajar. En el caso del Alligator admite los siguientes valores:

MODE_GATORJAW - Mandíbula del Alligator (línea azul)
MODE_GATORTEETH - Dientes del Alligator (línea roja)
MODE_GATORLIPS - Labios del Alligator (línea verde)

Ejemplo:
```
double teeth;
teeth=iAlligator(“EURUSD”,PERIOD_H1,128,96,64,0,0,0,MODE_EMA,
PRICE_TYPICAL,MODE_GATORTEETH,0);
// Calcula los "dientes" línea roja) en un gráfici horario del EURUSD.
// Para ell utiliza una media móvil exponencial aplicada a precio típico
// Los periodos utilizados para el indicador son 128, 96 y 64. No se aplica desplazamiento.
```

Average Directional Movement Index (ADX)
Este indicador se utiliza para detectar si hay o no tendencia.

Esquema: double iADX(string symbol,int timeframe,int period,int applied_price,int mode,int shift)

En este indicador, el parámetro mode admite los siguientes valores:

MODE_MAIN - Línea principal del indicador
MODE_PLUSDI - Línea +DI
MODE_MINUSDI - Línea –DI

Ejemplo:
```
double plusDi;
plusDi=iADX(“USDCAD”,PERIOD_M1,6,PRICE_OPEN,MODE_PLUSDI,0);
// Calcula el valor de la línea +DI en el gráfico de 1 min del USDCAD utilizando
// un periodo de 6 barras y tomando las aperturas de cada barra.
```

Average True Range (ATR)
Este indicador permite medir la volatilidad del mercado

Esquema: double iATR(string symbol,int timeframe,int period,int shift)

Ejemplo:
```
double atr;
atr=iATR(0,0,15,0); // ATR de 15 periodos del gráfico y escala temporal que deseemos.
```

Awesome Oscillator (AO)
Desarrollado por Bill Williams, permite determinar el momentum del mercado

Esquema: double iAO(string symbol, int timeframe, int shift)

Ejemplo:
```
double ao;
ao=iAO("EURAUD",PERIOD_W1,1);
// AO de la penúltima barra calculado en el gráfico semanal del EURAUD.
```

Bears Power
Este indicador trata de estimar el potencial bajista del mercado

Esquema: double iBearsPower(string symbol,int timeframe,int period,int applied_price,int shift

Ejemplo:
```
double bep;
bep=iBearsPower("EURUSD",PERIOD_M5,32,PRICE_CLOSE,1);
// Bears Power de la penúltima barra calculado sobre 32 cierres
// en un gráfico de 5 min. del EURUSD
```

Bollinger Bands (BB)
Se trata del conocido indicador de John Bollinger, con el que se pretende establecer un rango de movimiento probable para el precio

Esquema:

double iBands(string symbol, int timeframe, int period, int deviation, int bands_shift,int applied_price, int mode, int shift)

Como puede apreciarse tenemos dos parámetros adicionales: deviation, con el que se controla la amplitud de las bandas y bands_shift, con el que es posible desplazar las bandas. Por supuesto, también contamos con el parámetro mode, que admite los siguientes valores:

MODE_UPPER - Línea superior
MODE_LOWER - Línea inferior

Ejemplo:
```
double bb;
bb=iBands(0,0,20,2,0,PRICE_LOW,MODE_LOWER,0);
// Banda inferior de Bollinger calculada hasta la última barra
// de cualquier periodo y escala. Utiliza 20 barras para la media
// que se calcula sobre los mínimos y la desviación utilizada es 2
```

Bulls Power
Este indicador trata de estimar el potencial alcista del mercado

Esquema: double iBullsPower(string symbol, int timeframe, int period, int applied_price, int shift)

Ejemplo:
```
double bup;
bup=iBullsPower(0,0,10,PRICE_CLOSE,1);
// Bears Power de la penúltima barra calculado sobre 10 cierres
// en un gráfico y escala cualesquiera.
```

Commodity Channel Index (CCI)
El CCI se utiliza para medir la desviación del precio con respecto a su media

Esquema: double iCCI(string symbol, int timeframe, int period, int applied_price, int shift

Ejemplo:
```
double cci;
cci=iCCI(0,0,14,PRICE_TYPICAL,0);
// CCI de 14 periodos calculado sobre el precio típico
// para cualquier gráfico y escala.
```

DeMarker (DeM)
Creado por Tom DeMark, este indicador intenta predecir un giro de mercado basándose en la diferencia del precio con respecto a

barras anteriores.

Esquema: double iDeMarker(string symbol, int timeframe, int period, int shift)

Ejemplo:
```
double dm;
dm=iDeMarker("EURJPY",PERIOD_H4,51,1);
// Calcula el DeMarker hasta la penúltima barra en un gráfico de 4 horas
// del EURJPY utilizando un 51 barras.
```

Envelopes
Con este oscilador es posible crear envolventes del precio en base a una media móvil.

Esquema: double iEnvelopes(string symbol, int timeframe, int ma_period, int ma_method, int ma_shift,
int applied_price, double deviation, int mode, int shift)

Aquí aparece el parámetro deviation con el que se controla la amplitud de la envolvente en términos porcentuales (por ejemplo, si asignamos el valor 0.25 diremos que sume y reste el 25% del precio a la media móvil).

Asimismo, dado que el indicador consta de dos líneas, disponemos del parámetro mode para seleccionar la línea que deseemos, utilizando par ello los siguientes valores:

MODE_UPPER - Línea superior
MODE_LOWER - Línea inferior

Ejemplo:
```
double envel;
envel=iEnvelopes(0,0,21,MODE_SMA,0,PRICE_CLOSE,0.05,MODE_LOWER,0);
// Banda inferior de la envolvente calculada usando una media simple de 21 periodos
// y una desviación del 5% para cualquier gráfico y escala
```

Force Index (FRC)
Este indicador trata de medir la fuerza de la tendencia.

Esquema: double iForce(string symbol, int timeframe, int period, int ma_method, int applied_price, int shift)

Ejemplo:
```
double force;
force=iForce("EURGBP",PERIOD_M5,21,MODE_LWMA,PRICE_HIGH,1);
// FRC de 21 periodos calculado hasta la penúltima barra usando una media
// linealmente ponderada de los máximos del gráfico del EURGBP en 5 min.
```

Fractals
Se trata de otro de los indicadores de Bill Williams, utilizado para detectar suelos y techos. Dado que el fractal no se da en todas las barras, esta función devuelve el valor cero mientras no se produzca el patrón.

Esquema: double iFractals(string symbol, int timeframe, int mode, int shift)

Esta función dispone del parámetro mode para seleccionar el tipo de fractal:

MODE_UPPER - Fractales bajistas (techo de mercado)
MODE_LOWER - Fractales alcistas (suelo de mercado)

Ejemplo:
```
double frac;
frac=iFractals("USDJPY",PERIOD_D1,MODE_UPPER,0);
// Fractales bajistas en la última barra de un gráfico diario del USDJPY.
```

Gator Oscillator
El Gator Oscillator se utiliza para mediar el grado de convergencia o divergencia entre las líneas que componen el Alligator.

Esquema:

double iGator(string symbol, int timeframe, int jaw_period, int jaw_shift, int teeth_period,
int teeth_shift, int lips_period, int lips_shift, int ma_method, int applied_price, int mode, int shift)

Dado que se basa en el Alligator, aparecen los parámetros para modificar sus mandíbulas, dientes y labios. Asimismo cuenta con el parámetro mode para seleccionar la parte del histograma que nos interese:

MODE_UPPER - Histograma positivo (verde)
MODE_LOWER - Histograma negativo (rojo)

Ejemplo:
```
double gator;
gator=iGator(0,0,13,8,8,0,0,0,MODE_SMA,PRICE_CLOSE,MODE_UPPER,0);
// Histograma positivo calculado sobre el Alligator de periodos 13,8,8
// utilizando una media móvil simple de los cierres.
```

Ichimoku Kinko Hyo
Se trata del conocido conjunto de líneas japonés

Esquema: double iIchimoku(string symbol, int timeframe, int tenkan_sen, int kijun_sen,
int senkou_span_b, int mode, int shift)

Como puede observarse cuenta con los parámetros para controlar el periodo de cada una de las líneas que compone el indicador y un parámetro mode para seleccionar la línea que nos interese:

MODE_TENKANSEN - Tenkan-sen
MODE_KIJUNSEN - Kijun-sen
MODE_SENKOUSPANA - Senkou Span A
MODE_SENKOUSPANB - Senkou Span B
MODE_CHINKOUSPAN - Chinkou Span

Ejemplo:
```
double ichi;
ichi=iIchimoku(0,0,13,21,53,MODE_KIJUNSEN,0);
// Valor del Kijun-sen tomando periodos 13, 21, 53.
```

Market Facilitation Index (BW MFI)
Creado por Bill Williams, este indicador muestra el cociente entre el rango de la barra y el volumen negociado en dicha barra.

Esquema: double iBWMFI(string symbol, int timeframe, int shift

Ejemplo:
```
double bwmfi;
bwmfi=iBWMFI(0,0,0); // BWMFI calculado para la última barra del chart que tenemos en pantalla.
```

Momentum
El Momentum permite calcular la variación entre dos cierres separados x barras

Esquema: double iMomentum(string symbol, int timeframe, int period, int applied_price, int shift)

Ejemplo:
```
double mom;
mom=iMomentum(0,0,12,PRICE_CLOSE,1);
// Momentum de 12 periodos calculado hasta la penúltima barra.
```

Money Flow Index (MFI)
Este indicador mide la entrada y salida de dinero en el mercado

Esquema: double iMFI(string symbol, int timeframe, int period, int shift

Ejemplo:
```
double mfi;
mfi = iMFI(0,0,14,0);
// Calcula un MFI de 14 periodos para el gráfico actual.
```

Moving Average (MA)
Calcula una media móvil utilizando el método indicado sobre el campo del precio que deseemos.

Esquema: double iMA(string symbol, int timeframe, int period, int ma_shift, int ma_method, int applied_price, int shift)

Ejemplo:
```
double ma;
ma=iMA("EURCHF",PERIOD_M15,21,6,MODE_LWMA,PRICE_LOW,1);
// Calcula la media móvil linealmente ponderada de 21 periodos en el gráfico de 15 min.
// del EURCHF hasta la penúltima barra y la desplaza 6 barras hacia la derecha.
```

Moving Average Convergence/Divergence (MACD)
El MACD es un indicador que trata de determinar la tendencia en base a dos medias exponenciales.

Esquema: double iMACD(string symbol, int timeframe, int fast_ema_period, int slow_ema_period, int signal_period, int applied_price, int mode, int shift)

En este esquema los parámetros fast_ema_period, slow_ema_period y signal_period controlan el valor de las medias moviles utilizadas en el indicador. Asimismo el parámetro mode permite seleccionar la línea que nos interese asignando uno de estos valores:

MODE_MAIN - Línea principal
MODE_SIGNAL - Línea Signal

Ejemplo:

```
double ma;
ma=iMACD(0,0,9,21,9,PRICE_CLOSE,MODE_MAIN,0);
// Valor de la línea principal del MACD en el gráfico actual, utilizando parámetros, 9, 21 y 9.
```

Moving Average of Oscillator (OsMA)
El OsMA permite calcular la diferencia entre la línea principal y la línea signal del MACD

Esquema:

```
double iOsMA( string symbol, int timeframe, int fast_ema_period, int slow_ema_period,
              int signal_period, int applied_price, int shift)
```

Ejemplo:

```
double osma;
osma=iOsMA(0,0,12,26,9,PRICE_CLOSE,0);
// OsMA de un MACD de periodos 12,26,9
```

On Balance Volume (OBV)
Indicador que relaciona el comportamiento del precio con el volumen

Esquema: double iOBV(string symbol, int timeframe, int applied_price, int shift)

Ejemplo:

```
double obv;
obv=iOBV(0,0,PRICE_OPEN,0);
// OBV calculado hasta la última barra utilizando la apertura.
```

Parabolic Stop and Reverse system (Parabolic SAR)
El Parabolic SAR es un indicador que permite definir la tendencia y puntos de entrada y salida.

Esquema: double iSAR(string symbol, int timeframe, double step, double maximum, int shift)

Incorpora los parámetros habituales en este indicador: step para definir la aceleración y maximum para fijar el tope de la aceleración.

Ejemplo:

```
double sar;
sar=iSAR(0,0,0.02,0.2,0);
// Parabolic SAR con aceleración 0.02 y tope 0.2.
```

Relative Strength Index (RSI)
Se trata del conocido indicador de Welles Wilder para detectar puntos de giro en el mercado.

Esquema: double iRSI(string symbol, int timeframe, int period, int applied_price, int shift)

Ejemplo:

```
double rsi;
rsi=iRSI("USDCAD",PERIOD_M1,9,PRICE_OPEN,0);
// Calcula un RSI de 9 periodos en el gráfico de 1 min. del USDCAD utilizando
// datos de apertura.
```

Relative Vigor Index (RVI)
EL RVI es un indicador muy similar al Estocástico que trata de detectar giros de mercado.

Esquema: double iRVI(string symbol, int timeframe, int period, int mode, int shift)

El indicador consta de dos líneas por lo que requiere del parámetro mode, que admite los siguientes valores:

MODE_MAIN - Línea principal
MODE_SIGNAL - Línea signal

Ejemplo:
```
double rvi;
rvi=iRVI(0,0,12,MODE_MAIN,1);
// Calcula la línea principal de un RVI de 12 periodos hasta la penúltima barra.
```

Standard Deviation
Con este indicador se calcula la desviación típica del mercado, con objeto de medir su volatilidad.

Esquema: double iStdDev(string symbol, int timeframe, int ma_period, int ma_shift, int ma_method, int applied_price, int shift)

Ejemplo:
```
double sd;
sd=iStdDev(0,0,10,0,MODE_SMA,PRICE_CLOSE,0);
// Calcula la desviación típica de 10 periodos utilizando una media movil simple.
```

Stochastic Oscillator
El Estocástico es un indicador con el que se intenta detectar puntos de giro del mercado.

Esquema: double iStochastic(string symbol, int timeframe, int %Kperiod, int %Dperiod, int slowing, int method,
int price_field, int mode, int shift)

Aquí aparecen algunos parámetros que controlan el comportamiento del Estocástico: así %Kperiod es el periodo de la línea %K, %Dperiod es el de la línea %D y slowing es el periodo de la media móvil que suaviza la línea %D.

Por su parte, el parámetro price_field admite dos valores: 0, para tomar mínimos y máximos en el cálculo, ó 1 si queremos que sólo utilice los cierres.

Finalmente dado que el Estocástico consta de dos líneas contamos con el parámetro mode que admite los siguientes valores:

MODE_MAIN - Línea principal
MODE_SIGNAL - Línea Signal

Ejemplo:
```
double s;
s=iStochastic(0,0,10,6,6,MODE_SMA,0,MODE_MAIN,0);
// Calcula el valor de la línea principal del Estocástico de parámetros 10,6,6
// Para ello, utiliza una media simple y máximos y mínimos
```

Williams' Percent Range (%R)
Indicador que funciona de manera similar al Estocástico desarrollado por Larry Williams

Esquema: double iWPR(string symbol, int timeframe, int period, int shift)

Ejemplo:
```
double wpr;
wpr=iWPR("USDCHF",PERIOD_D1,9,0);
// Calcula un WPR de 9 periodos en el gráfico diario del USDCHF.
```

(*Continuará...*)

Un saludo,
X-Trader

MQL4 para Novatos V

por X-Trader
11/04/2008

Bien, pasamos a la acción: si Vd. ha seguido el curso desde el principio, no debería tener problemas para crear su primer indicador en MQL4.

Eso sí, antes de ponernos manos a la obra es preciso repasar qué características poseen los indicadores en Metatrader:

Por un lado, existen indicadores que se dibujan sobre el propio gráfico, constan de una o varias líneas, sus valores pueden ser ilimitados y se representan con un color, grosor y estilo de línea definidos. Este sería el caso de una media móvil.
Por otro lado, tenemos indicadores que se dibujan en una nueva subventana mostrando una o varias líneas cuyo color, grosor y estilo pueden ser definidos, y cuyos valores están acotados. Tal sería el caso del RSI.
También tenemos indicadores en forma de histograma o que se representan mediante símbolos (flechas, círculos, etc.)

En definitiva podemos decir que un indicador consta de los siguientes parámetros:

Una o más líneas (que en MQL4 se denominan Buffers), pudiendo definir para cada una su aspecto, color, grosor y estilo.
La ventana en la que se trazará (sobre el gráfico o en una subventana)
En caso de que el indicador sea dibujado en una subventana, habrá que definir si el rango del indicador será limitado o se ajustará automáticamente la escala.

Teniendo en cuenta todo esto, comenzamos a crear nuestro primer indicador. Para ello abrimos el Metaeditor tecleando F4 y vamos a File->New. Se nos abrirá un asistente en el que debemos seleccionar Custom Indicator y darle a Siguiente. A continuación rellenamos los campos de Name, Author y Link. Justamente debajo de esos campos vemos un espacio en blanco que pone Parameters. Ahí podemos añadir variables que después se pueden modificar para que cualquier usuario pueda hacer cambios en los valores que definen el indicador.

Para nuestro ejemplo, pulsamos en Add y pondremos como nombre al parámetro, barsToProcess, y fijando su valor por defecto igual a 100, dejando el tipo de variable como int (entero). Le damos a Siguiente.

Ahora se nos piden dos cosas: por un lado, si queremos tener el indicador en una ventana aparte (para lo cual deberemos marcar Indicator in separate window) y por otro, el número de líneas (Indexes) de que constará el indicador. Debemos tener en cuenta que aunque pongamos que tiene 5 líneas no tienen porque dibujarse todas: podemos por ejemplo usar 3 para almacenar valores y 2 para representar el indicador en sí. Asimismo, el máximo de líneas que podemos poner es de 8. Para añadir una nueva línea haremos click en Add pero por defecto ya tenemos una, cuyo color podemos modificar (ponemos Azul por ejemplo) así como el tipo de representación (que por ahora dejaremos en Line).

Hacemos click en Finalizar y listo, ya tenemos definidos todos los parámetros del indicador. Veamos qué significa el código que acabamos de generar:

El encabezado recoge los datos del autor:

```
#property copyright "X-Trader"
#property link      "http://www.x-trader.net"
```

A continuación se enumeran las propiedades del indicador: debe dibujarse en una subventana, con una línea y en color azul marino (Navy):

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_color1 Navy
```

Observese que el parámetro indicator_color va seguido de un número, que se corresponde con la línea (buffer) a la que afecta (ojo, la enumeración de los buffers comienza aquí en 1 y no en 0 como hasta ahora).

Seguidamente nos muestra los parámetros definidos:

```
//---- input parameters
extern int barsToProcess=100;
```

Definimos el buffer:

```
//---- buffers
double ExtMapBuffer1[];
```

Ahora comienza el código del indicador en sí. Si miramos el código completo veremos que consta de tres secciones diferenciadas: init(), deinit() y start(). Veamos en qué consiste cada una:

init() es una función que sólo será llamada una vez, en el momento de inicializar el indicador. En esta sección se comprueba que los valores de los parámetros son correctos y se prepara el indicador para cargar datos. No obstante no es obligatorio utilizarla por lo que si no lo vamos a usar podemos borrarla.
deinit() se utilizará para indicar qué debe suceder cuando borremos el indicador de un gráfico. De nuevo esta función no es obligatoria.

start() será llamada cada vez que aparezca un nuevo tick en el gráfico por lo que aquí es donde debemos poner las instrucciones que permitirán realizar los cálculos para obtener el valor del indicador.

Veamos que ha escrito el asistente dentro de cada una de las funciones:

En init() se ha especificado como se dibujará el buffer mediante la función SetIndexStyle. En este caso, con DRAW_LINE nos dibujará una línea. Observese que aquí la enumeración de los buffers vuelve a ser desde 0. Por su parte la función SetIndexBuffer asocia un array al buffer que habíamos definido. Finalmente return(0) finaliza la función init():

```
int init()
{
//---- indicators
SetIndexStyle(0,DRAW_LINE);
SetIndexBuffer(0,ExtMapBuffer1);
//----
return(0);
}
```

En el caso de deinit() no tenemos nada (generalmente no utilizaremos demasiado esta función)

Ahora nos falta decirle qué es lo que debe dibujar. Con lo que vamos a poner dentro de start() vamos a dibujar un indicador que represente números aleatorios. El código que debe aparecer en esa sección es el siguiente:

```
int start()
{ int counted_bars=IndicatorCounted();
for(int i=0;i<Bars;i++)
{ ExtMapBuffer1[i]=MathRand()%1001; }
return(0); }
```

Un poco más adelante veremos que significa todo esto. De momento vamos a compilar el indicador tecleando F7. Una vez hecho esto, vamos a Metatrader, abrimos el Navigator (Ctrl+N) y desplegamos Custom Indicators. En esa lista debe aparecer el indicador con el nombre que le hayan dado. Le seleccionamos, pulsamos botón derecho en el ratón y hacemos click en Attach to a chart. En la ventana que nos aparece hacemos click en Aceptar. El resultado debería ser algo como esto:

Bien veamos qué significa el código que hemos introducido en start(). Comenzamos con un bucle de tipo for, comenzando en la última barra (i=0) y finalizando en la primera disponible (i<Bars) incrementando una unidad el contador (i++):

```
for(int i=0;i<Bars;i++)
```

En cada iteración del bucle asignamos a cada elemento del buffer (que se corresponde con la correspondiente barra del gráfico) un número aleatorio entre 0 y 1000:

```
{    ExtMapBuffer1[i]=MathRand()%1001;  }
```

Obsérvese que cada vez que entra un nuevo tick, el oscilador cambia por completo ya que depende de la función MathRand(). Si

no quisieramos que esto sucediera (cosa más que probable ya que el consumo de recursos del ordenador se dispararía) podríamos utilizar el siguiente código:

```
int start()
{ int counted_bars=IndicatorCounted(),
limit;
if(counted_bars>0)
counted_bars--;
limit=Bars-counted_bars;
for(int i=0;i<limit;i++)
{ ExtMapBuffer1[i]=MathRand()%1001; }
return(0); }
```

Donde se ha introducido la función IndicatorCounted() que nos dice cuantas barras no se han modificado desde que se llamó a la función start(). Ahora el bucle for depende de la variable limit que almacena el número de barras que están modificándose en el momento de calcular el indicador.

Finalmente, seguro que recuerdan que hemos introducido un parámetro llamado barsToProcess que todavía no aparece en el código. Pues bien, ha llegado el momento de usarlo, veamos el siguiente código:

```
int start()
{    int counted_bars=IndicatorCounted(),
     limit;
     if(counted_bars>0)
     counted_bars--;
     limit=Bars-counted_bars;
     if(limit>barsToProcess)
     limit=barsToProcess;
     for(int i=0;i<limit;i++)
     {ExtMapBuffer1[i]=MathRand()%1001;}
return(0);  }
```

Ahora lo que hacemos es comprobar si limit es mayor que barsToProcess. En caso afirmativo hacemos que limit sea igual a barsToProcess. De esta forma podemos aplicar el indicador a las últimas n barras que deseemos.

(*Continuará...*)

Un saludo,
X-Trader