Introducción a MQL4

"Estoy profundamente en deuda co el Sr. Ranat Fatkhullin, el director general de MetaQuotes Software Corp.. por su confianza, apoyo profesional y toda la ayuda posible. Tambien estoy agradecido a los empleados de la empresa, Stanislav Starikov y Rashid Umarov, por su valioso asesoramento y en la ayuda en la redacción de este libro."

Sergey Kovalyov
http://autograf.dp.ua

S. Kovalyov

Programación en lenguaje algorítmico MQL4

Curso Introductorio

Hoy en día, el ordenador personal se ha convertido en algo indispensable para todo el mundo. El rápido desarrollo de Internet y el rendimiento de los ordenadores modernos han abierto nuevas perspectivas en muchos ámbitos de las actividades humanas. En solo un tiempo tan cercano como hace diez años el mercado financiero de trading solo estaba disponible para los bancos y para un numero limitado de una comunidad de especialistas. Hoy en día, cualquiera puede empezar en cualquier momento y unirse al mundo de los comerciantes y profesionales independientes de trading.

Cientos de miles de traders en todo el mundo ya han juzgado el terminal MetaTrader 4. La utilización de su lenguaje de programación incrutado MQL4, ha ascendiendo a los traders a un nuevo nivel de comercio, el comercio automatizado. Ahora, un comerciante puede poner en práctica sus ideas con un programa de aplicación - escribir un indicador personal, un script para realizar una operaciones simples, o crear un Asesor Experto ó un sistema de comercio automatizado (robot de comercio). Un Asesor Experto (AE) puede trabajar en 24/7 (24 horas al dia/ 7 dias a la semana), sin intervención, hacer un seguimiento de precios del valor, enviar mensajes electrónicos, SMS´s a su teléfono móvil, así como hacer muchas otras cosas útiles.

La principal ventaja de las aplicaciones es la posibilidad de hacer operaciones de acuerdo con el algoritmo establecido por el comerciante. Cualquier idea que se puede describir en un lenguaje algorítmico (intersección de dos medias móviles o el procesamiento digital de señales, triples pantallas de Elder o el análisis fractal de Pedros, una red neuronal o construcciones geométricas) puede ser codificada en una aplicación y luego utilizados en la práctica de comercio.

Desarrollo de aplicaciones para el Terminal de Usuario MetaTrader 4 requiere el conocimiento de MQL4. El presente manual le ayudará a crear sus propios Asesores Expertos, scripts e indicadores encarnando en él sus ideas - algoritmos de su rentabilidad comercial. El libro de texto está destinado a un gran número de lectores sin experiencia en programación que deseen aprender a desarrollar aplicaciones de comercio automatizado para Terminal de Usuario MetaTrader 4. El libro de texto está concebido de tal modo que para hacer del aprendizaje MQL4 tan conveniente y consecuente como sea posible.

Prefacio

Hay una especial dificultad en escribir un libro de texto sobre programación para los principiantes, ya que el área de conocimiento objeto de examen implica algunos nuevos conceptos que no se basan en nada previamente conocido o habituales.

En términos generales, problemas de este tipo puede ocurrir en cualquier otra esfera del conocimiento. Por ejemplo, el punto se conoce en matemáticas como círculo infinitesimal, mientras que el círculo en sí se define como un conjunto de puntos ordenados de una cierta manera. Como es fácil ver, estos términos se definen unos a través de otros. Al mismo tiempo, esta "inadvertencia" no se convierte en un escollo para las matemáticas. Los dos círculos, puntos, así como otras condiciones aprobadas en matemáticas van bien juntos. Por otra parte, todo el mundo entiende qué es un punto que es un círculo.

Introducción a MQL4

Es fácil saber que la inmensa mayoría de los términos ordinarios tienen límites indeterminados. Algunos de esos límites son tan difusos que volcaron dudas sobre la existencia del propio objeto o fenómeno definido por el término. Sin embargo, la naturaleza del hombre es que esta extraña (en términos de lógica normal) situación no viene entre un hombre y su existencia y fructíferas actividades. Después de un período de tiempo que estos términos han sido utilizados el concepto toma un sentido completo para nosotros. Es difícil responder a la pregunta de cómo y por qué ocurre de esta manera. Pero lo hace. Nosotros sólo sabemos que múltiples referencias a un término desempeña un papel importante en el notable proceso de aprendizaje de los términos.

Las siguientes tareas fueron establecidas en el presente trabajo:

- Desarrollar el sentido de utilizar nuevos términos con bien conocidos analogías;
- hacer que el significado de cada término intuitivamente evidente cuando se aparece por primera vez;
- Proveer a los lectores con la necesaria cantidad de información que caracterizan a los programas y la programación.

Con este fin, el libro contiene muchos ejemplos y cifras. El texto incluye referencias cruzadas que permiten al lector obtener información sobre temas afines.

Unas pocas palabras sobre la presentación de materiales. Algunos libros de texto sobre la programación invitan a sus lectores sobre la primera página a imprimir "Hola, mundo!" utilizando un programa simple. Sus autores piensan que tan pronto como sus lectores comienzan a aprender programación deben referirse a los textos del programa y poco a poco acostumbrarse a la forma en que el programa pude aparecer de tal modo que posteriormente facilite su aprendizaje. Sin embargo, este enfoque da lugar a que el lector tiene que tratar con varios términos desconocidos, al mismo tiempo, y justo adivinar el contenido y las propiedades de algunas líneas en el programa. Esto puede dar lugar a un malentendido y, consecutivamente, a vacíos en el conocimiento del lector.

A mi modo de ver, sería más eficaz utilizar un método donde el lector va a la siguiente sección en el libro de texto sólo después de que él o ella han tenido una profunda comprensión de los anteriores materiales. En el marco de este método, el primer programa se ofrece al lector sólo después de que él o ella han dominado todas las condiciones necesarias y ha obtenido alguna información sobre los principios básicos de codificación. Este es el método en el que se basa presente libro de texto.

El dominio del conocimiento dado en el libro, el lector tiene que ser un usuario de PC y tener cierta experiencia en trabajar con MetaTrader 4 producidas por MetaQuotes Software Corp.

Índice de contenidos

Prefacio

Fundamentos de MQL4

- Algunos conceptos básicos
- Constantes y Variables
- Tipos de datos
- Operaciones y expresiones
- Operadores
- Funciones
- Tipos de programa

MetaEditor

- Sistema de archivos
- Creación y uso de programas

Programa en MQL4

- Programa Estructura
- Funciones especiales
- Ejecución de Programas
- Ejemplos de aplicación

Operadores

- Operador de asignación
- Operador condicional "if-else"
- Operador de ciclo 'mientras'
- Operador de ciclo 'for'
- Operador de "pausa"
- Operador en "Continuar"
- Operador 'switch'
- Función de llamada
- Descripción y función del operador «return»

Variables

- Variables predefinidas y RefreshRates función
- Tipos de Variables
- GlobalVariables
- Arrays

Introducción a la programación MQL4

Antes de comenzar a estudiar programación MQL4, vamos a definir el alcance de nuestro estudio. En primer lugar hay que señalar que los programas examinados en este libro sólo se pueden utilizar como aplicaciones para trabajar en MetaTrader Terminal de Usuario 4. Fig. 1 a continuación muestra el papel de estos programas en la gestión del comercio. Para una mejor comprensión de la importancia de estos programas en gestión del comercio, echemos un vistazo a la Fig. 1.

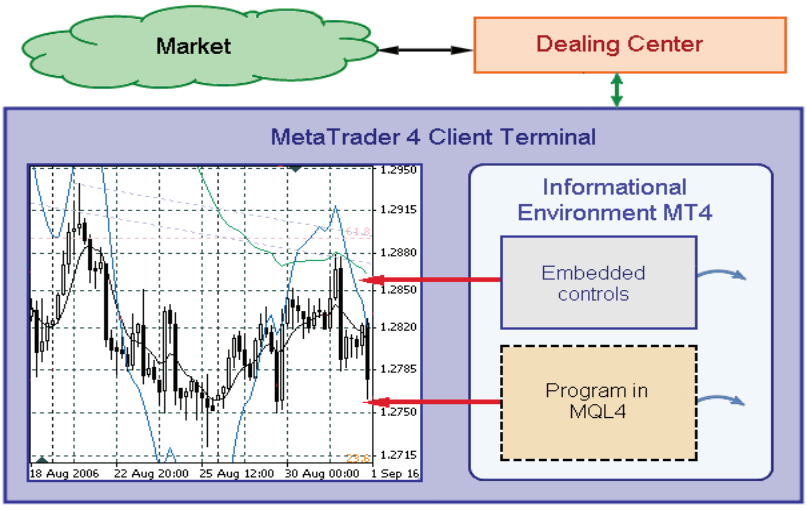


Fig. 1. Un programa en MQL4 como parte de MetaTrader Terminal de Usuario 4.

Si usted está interesado en la programación de MQL4, usted debe estar familiarizado con el Terminal de Usuario. El Terminal de Usuario forma parte del sistema de comercio en línea. Este sistema también incluye un servidor instalado en un dealing center. En el dealing center a su vez están conectados con otros participantes del mercado como bancos e instituciones financieras.

El Terminal de Usuario incluye informaciones del entorno: un conjunto de parámetros que informan sobre el estado del mercado y las relaciones entre el comerciante y el centro de tratamiento de datos. Contiene información sobre los precios actuales, las limitaciones en tamaño máximo y mínimo de las órdenes, distancia mínima de las órdenes de stop, permiso / prohibición del trading automático y muchos otros parámetros útiles que caracterizan el estado actual del mercado. El entorno informeativo se actualiza cuando un nuevo ticks es recibido en el terminal (línea verde en la Fig. 1).

Conjunto de herramientas

El Terminal de Usuario contiene herramientas que permiten la realización de análisis técnico del mercado y gestión de la ejecución manual de comercio. Para el análisis de mercado se pueden utilizar diferentes indicadores tecnicos y estudios: lineas de soporte y resistencia, canales de tendencia, niveles de Fibonacci, etc.

Para la gestión de trading manual se usan las órdenes desde barra de herramientas. Utilizando esta barra de herramientas un trader puede abrir, cerrar y modificar órdenes. Además, el terminal tiene la opción de administración automatizada de stops de la posición. Unas acciones de comercio que incorporen herramientas de gestión de trading dan como resultado la formación de órdenes comerciales las cuales son enviados al servidor.

Para obtener más información sobre el Terminal de Usuario, por favor consulte la sección "Userguide" ***(ClientTerminal_folder \ Terminal.chm).***

Herramientas de programación

El análisis de mercado y la gestión de comercio en Terminal de Usuario MetaTrader 4 se aplican con la ayuda de herramientas de programación. El lenguaje MQL4 permite la creación de tales programas. Hay tres tipos de aplicaciones creadas en MQL4 y creadas para trabajar en el Terminal de Usuario:

- **Costum Indicator**: (Indicador personal) Es un programa para mostrar gráficos de mercado escrito de acuerdo al algoritmo de su autor.
- **Expert Advisor**: (Asesor Experto) Es un programa que permite automatizar gran parte de operaciones comerciales y completo tranding automatico.
- **Script** – Es un programa para ejecutarlo una vez, las acciones incluyen la ejecución de operaciones de trading.

Fig.1 muestra que la aplicación tiene los mismos medios de acceso al Terminal de Usuario y entorno informativo como las herramientas de comercio manual (flechas azules). También puede formar la gestión de medios (flechas rojas), pasó al Terminal de Usuario. Intercambio de datos y programas de diferentes tipos pueden ser utilizados simultáneamente. El uso de estas aplicaciones permite que un programador pueda automatizar una gran parte de las operaciones de comercio o crear un robot que ralice comercio sin la intervencion del comerciante.

Las aplicaciones y herramientas de gestión manual pueden ser utilizadas en el Terminal de Usuario al mismo tiempo y complementarse entre sí.



Las características técnicas fundamentales de la negociación en línea utilizando el sistema de comercio MetaTrader es que la gestión de todas las acciones se producen en el Terminal de Usuario y luego es enviada a un servidor. Los programas de aplicación (Expert Advisor, script o Custom Indicator) sólo pueden trabajar como parte del Terminal de Usuario, siempre y cuando se conecte a un servidor (dealing center). Ninguno de los programas de aplicación se instala en el servidor.

El servidor sólo permite procesar señales procedentes del Terminal de Usuario. Si un Terminal de Usuario está desconectado de Internet o un programa de aplicación (Expert Advisor o script) que haya en ella, sus acciones no generan ninguna gestión, nada va a suceder en el servidor.

El alcance de nuestro estudio incluye los programas (Asesor Experto, scripts personalizados e indicador personal) que permitan llevar a cabo el trading de forma parcial o de forma totalmente automatizada y de esta forma, ampliar significativamente el mantenimiento de información comercial (ver Fig. 1). En este libro usted encontrará la descripción de los componentes del programa y las principales reglas de creación y uso de programas. También vamos a examinar en detalle ejemplos de programas y parámetros de información del entorno en el Terminal de Usuario que están disponibles al programa durante su ejecución.



Los programas para el tratamiento automatizado del comercio poseen mucho más potencial y posibilidades que las herramientas manuales de gestión de comercio.

En la mayoría de los casos, un programa le permite a un comerciante un trabajo más fácil, eliminando la necesidad de un constante seguimiento de la situación del mercado y tener que estar sentado ante un ordenador durante un largo periodo de tiempo. También puede ayudar a aliviar la tensión nerviosa y reducir el número de errores que aparecen en períodos de extrema tensión emocional. Pero lo principal es, que el uso del programa como método de gestión del comercio, permite desarrollar las propias ideas y probarlas con datos históricos, seleccionando los parámetros óptimos para la aplicación de estas ideas y, por último, permite aplicar las ideas sobre las estrategias comerciales.

Fundamentos de MQL4

Esta sección se representa la terminologia básica del lenguaje de programación MQL4:

- Algunos conceptos básicos

  Son descritos términos tales como "tick" (un cambio de precio), "control" en los algoritmos, "comentarios" que describen los programas. El principal acontecimiento de las cotizaciónes en los mercados financieros es el cambio de precio. Esta es la razón por la que el tick es un acontecimiento importante que hace que los mecanismos básicos de los programas de MQL4 se ejecutan ¿Qué hacer cuando ocurre un nuevo tick? ¿Qué medidas tomar? Este es el control que mueve la vanguardia aquí. Pero no se olvide de comentar su código.

- Constantes y Variables

  Los términos de constantes y variables serán explicadas; la diferencia entre estos términos sera explicada. Como el término sugiere, una constante es algo continuo, un valor fijo. A diferencia de la constante, una variable es un objeto del código de programación que puede modificar su contenido. Es imposible escribir un programa sin usar objetos inalterables (constantes) y/o objetos que puedan ser modificados durante la ejecución del programa (variables).

- Tipos de datos

  Ciertos tipos de datos se utilizan en cualquier lenguaje de programación. El tipo de una variable se elige de acuerdo a su finalidad. ¿Cómo podemos declarar una variable, ¿cómo podemos inicializarla (preset su valor inicial)? Una elección errónea del tipo de una variable puede frenar el programa o incluso dar lugar a un mal funcionamiento.

- Operaciones y expresiones

  Las operaciones se hacen sobre operandos ¿Qué tipo de operaciones hay? ¿Cuáles son las características especiales de las operaciones sobre enteros? ¿Por qué es importante recordar los diferentes tipos de datos precedentes? Sin conocer las características de algunas operaciones, pueden aparecer sutiles errores.

- Operadores

  Los operadores pueden ser simples y complejos. Una acción necesaria no siempre pueden ser ejecutadas por un operador simple. Si es necesario que un grupo de operadores se ejecute como un gran operador, este grupo debe incluirse como un operador compuesto. Serán dados los requerimientos necesarios y ejemplos especificos de utilización de los operadores.

- Funciones

  La necesidad de conseguir un código simple nos lleva al término de función. Para poder utilizar la función en distintos lugares del programa, es necesario establecer parámetros a la función. Tendremos en cuenta el proceso de creación de función definida por el usuario. Se ofrecerán ejemplos de uso de funciones estándar.

- Tipos de programa

  Scripts, indicadores personales y Expert Advisor son los tipos de programas de MQL4 que le permiten cubrir prácticamente toda la clase de problemas relacionados con el comercio en los mercados fincancieros. Es necesario comprender los efectos de cada tipo de programas con el fin de utilizar el Terminal de Usuario de MetaTrader 4 de la mejor manera.

Algunos conceptos básicos

Por lo tanto, el objeto de nuestro interés es un programa escrito en MQL4. Antes de empezar una presentación detallada de las reglas para escribir programas, es necesario describir los conceptos básicos que caracterizan a un programa y sus interrelaciones con el entorno de información. El Terminal de Usuario MetaTrader 4 es conocido para trabajar on-line a través de internet. La situación en los mercados financieros cambia continuamente, esto afecta al grafico del símbolo (instrumento) en el Terminal de Usuario. Los ticks proveen al Terminal de información acerca de los cambios de precios en el mercado.

La noción del tick

El **tick** es un evento que se caracteriza por establecer un nuevo precio del instrumento financiero (símbolo) en algún instante.

Los ticks son enviados a cada Terminal de Usuario desde un servidor instalado en un Centro de transaciones bursátiles. Según corresponda a la situación actual del mercado, los ticks pueden recibirse con más o menos frecuencia, pero cada uno de ellos trae una nueva cotización (en el comercio de divisas, es el coste de una moneda expresado en términos de otra moneda).

Una aplicación operando con el Terminal de Usuario puede funcionar durante de un largo período de tiempo, por ejemplo, varios días o semanas. Cada aplicación se ejecuta con arreglo a las normas establecidas para cada determinado tipo de programa. Por ejemplo, un Asesor Experto (AE) no funciona continuamente todo el tiempo. Un Asesor Experto es por lo general puesto en marcha en el momento en que se marca un nuevo tick. Por esta razón, no se caracteriza el tick como si se acabara de marcar un nuevo precio, sino como una orden de ejecución del programa para ser procesada por el Terminal de Usuario.

La duración de la operación del Asesor Experto depende de lo que esté incluido en el código del programa. Normalmente los AEs completan un ciclo de información / procesamiento en algunas décimas o centésimas de segundo. Dentro de este tiempo, el AE puede haber procesado algunos parámetros, tomar una decisión comercial, dar al trader alguna información útil, etc Después de haber terminado esta parte de su labor, el AE se pone en modo de espera hasta que se marque un nuevo tick. Este nuevo tick inicia de nuevo al Asesor Experto (Expert Advisors), el programa hace su trabajo y de nuevo vuelve al modo de espera. A continuación se describe detalladamente cómo la aparición de un nuevo tick actua sobre la operación del programa.

El concepto de control

Usaremos el término «control» cuando hablemos de cómo es el flujo de ejecución de código de un programa así como su interacción con el Terminal de Usuario.

El control es un proceso de realización de acciones preestablecidas por el algoritmo del programa y las caracteristicas del Terminal de Usuario. El control puede ser transferido dentro un programa desde una línea de código a otro, así como desde el programa hacia el Terminal de Usuario.

El control se transfiere de una manera similar a la manera como se da la palabra a alguien en una reunión. Al igual que el orador una reunion toma la palabra y luego se la da a los demás, el Terminal de Usuario y el programa de control de transferencias se pasan el control uno a otro. En este caso, es el Terminal de Usuario quien domina. Su estado es superior a la del programa, al igual que la autoridad del presidente de una reunión es más grande que las de un simple portavoz.

Antes de que el programa se ponga en marcha, el control está bajo la supervisión del Terminal de Usuario. Cuando un nuevo tick es recibido, el Terminal de Usuario transfiere el control al programa. El código del programa comienza a ser ejecutado en este momento.

El Terminal de Usuario, después de que ha transferido el control al programa, no detiene su funcionamiento. Sigue trabajando con el máximo rendimiento durante todo el período de tiempo que ha sido ejecutado desde el ordenador. El programa sólo puede comenzar a funcionar en el momento en que el Terminal de Usuario le ha transferido el control (al igual que el presidente de una reunión controla la reunión todo el tiempo, mientras que el actual orador tome la palabra sólo por un período de tiempo limitado).

El programa devuelve el control al Terminal de Usuario después de que ha completado su operación, y no puede ponerse en marcha por su propia cuenta. Sin embargo, cuando el control ha sido transferido al programa, este devuelve el control al Terminal de Usuario por sí mismo. En otras palabras, el Terminal de Usuario no puede tomar el control del programa por sí solo. La dinámica de las acciones del usuario (por ejemplo, obligar la finalizacion del programa) son una excepción.

Al discutir las cuestiones del rendimiento y las estructuras internas de los programas, estamos interesados la mayoría de la veces en la parte del control que se transfiere dentro de un programa. Vamos a hacer referencia a la Fig. 2 que muestra la naturaleza general de la transferencia del control hacia, desde y dentro de un programa. Los círculos que se muestra en la figura caracterizar algunos de los pequeños fragmentos logicos de un programa, mientras que las flechas entre los círculos muestran cómo se transfiere el control de un fragmento a otro.

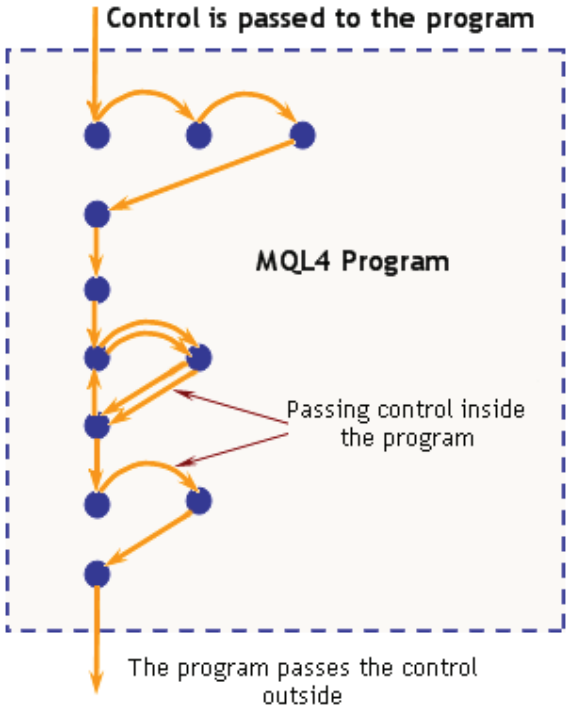


Fig. 2. La transferencia de control en un programa

Un programa que ha aceptado el control del Terminal de Usuario (el programa en ejecución) comienza a ejecutar algunas acciones en función de su algoritmo inherente. El programa contiene las líneas de programa, órdenes generales de ejecución de un programa secuencial que consiste en la transferencia del control de una línea a otra, de la parte superior hacia abajo. Qué y con arreglo a qué reglas se puede escribir en estas líneas se discute más adelante en todos sus detalles.

Aquí, sólo es importante destacar que, lógicamente, se ejecuta cada fragmento terminado (por ejemplo, algunos cálculos matemáticos, un mensaje en la pantalla, una orden comercial etc.) y mantiene el control hasta el fragmento actual del programa ejecutado. Después de que se haya completado todo, el control se transfiere a otro fragmento. Por lo tanto, el control dentro de un programa es transferido desde un fragmento lógico completado a otro. Tan pronto como el último fragmento es ejecutado, el programa de transferencia (ida y vuelta) pasa el control al Terminal de Usuario.

La noción del Comentario

Un programa consiste en dos tipos de registros: los del propio programa y los textos explicativos que están al código de programa.

El **Comentario** es opcional y no es una parte ejecutable del programa.

Por lo tanto, un comentario es opcional dentro de un programa. Esto significa que un programa hace el trabajo de acuerdo con su código, independientemente de si hay comentarios en ella o no. Sin embargo, los comentarios facilitan la comprensión del código del programa en gran medida. Hay comentarios de una línea y de múltiples líneas. El comentario de una linea es cualquier secuencia de caracteres que van a continuacin de una dobe barra (//). El signo de una línea de comentario queda terminado con el salto de línea. El comentrio multi-línea comienza con / * y termina con * / (ver Fig. 3).

Los comentarios son utilizados para explicar el código de programa. Un buen programa siempre contiene comentarios.

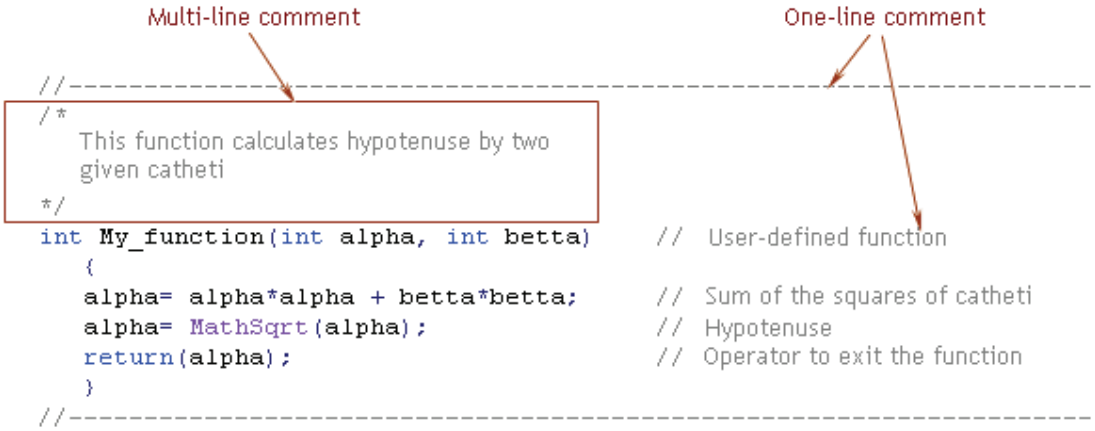


Fig. 3. Ejemplo de comentarios en un programa.

Los comentarios son ampliamente utilizados en la codificación. Por lo general se muestran en gris en los códigos. Vamos a utilizar los comentarios también con el fin de explicar nuestros códigos y hacerlos más inteligibles.

Constantes y Variables

Los términos de «constante» y «variable» se considerarán juntos dentro de esta sección, ya que estos sus conceptos son muy proximos entre sí.

El concepto de constante

Una **Constante** es una parte de un programa, un objeto que tiene un valor.

El término de constante en un programa es similar al que se usa en las ecuaciones matemáticas. Se trata de un valor invariable. Para describir la naturaleza de una constante utilizada en un lenguaje algorítmico en tantos detalles como sea posible, vamos a hacer referencia a su concepto físico y matemático.

La raza humana ha descubierto las constantes, los valores de los que no dependen de nosotros de ninguna manera. Estos son, por ejemplo, en la física: la aceleración de caída libre que es siempre igual a 9,8 m / s / s ó en matemáticas: Pi = 3,14. Las constantes de este tipo no pueden considerarse similares a las constantes de un lenguaje algorítmico.

El término constante también se utiliza en ecuaciones matemáticas. Por ejemplo, en la ecuación de Y = 3 * X + 7, los números 3 y 7 son constantes. Los valores de esas constantes son totalmente dependientes de la voluntad de la persona que ha hecho la ecuación. Esta es la analogía más cercana de constantes utilizadas en los programas de MQL4.

Una constante (como un valor) la coloca un programador en el código en la fase de creación del programa. La constante se caracteriza sólo por su valor, por lo que los términos de «constante» y «el valor de una constante" son sinónimos.

Ejemplo de constates

```
37, 3,14, true, "Kazan"
```

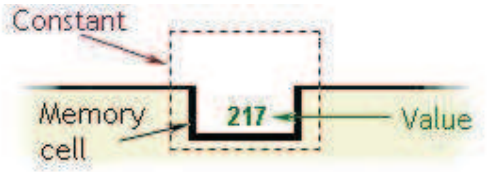


Fig. 4. Una constante en la memoria de un ordenador.

Propiedades de las constantes

La propiedad de una constante es su poder para conservar durante el tiempo de funcionamiento del programa el valor fijado por el programador y entregar este valor al programa cuando el programa se lo pide (Fig. 5). Para cada constante del programa, el ordenador asigna una parte de su memoria del tamaño necesario. El valor de una constante no se puede cambiar durante la ejecución del programa ni tampoco por parte del programador. (Fig. 6).



El valor de una constante es siempre el mismo.

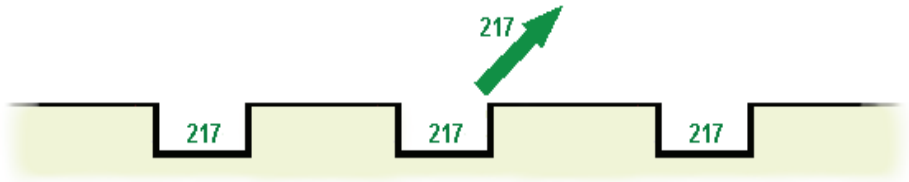


Fig. 5. El estado de una célula de memoria de una constante la hora de fijar el valor al programa.

El valor de una constante no puede cambiarse durante la operación del programa.

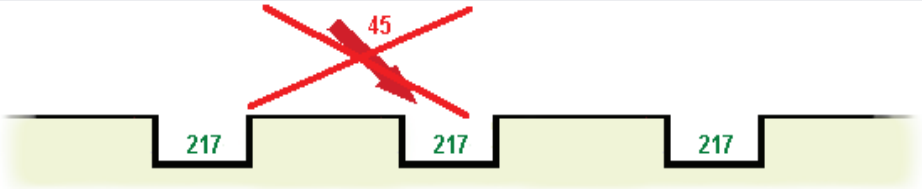


Fig. 6. Es imposible cambiar el valor de una constante durante la operación del programa.

El concepto de variable

Variable es una parte de un programa que tiene un nombre y un valor.

El término de variable en MQL4 es similar al aceptado por las matemáticas. La diferencia entre ellos consiste sólo en que el valor de una variable en matemáticas está siempre implícito, mientras que el valor de la variable en un programa de ejecución se almacena en una celda especial de la memoria del ordenador.

El término "identificador de variable" son plenamente sinonimas al de "nombre de variable". La variable la pone su autor en el código de texto en la fase de codificación como un nombre de variable. El nombre (identificador) de una variable puede constar de letras, dígitos y subrayado. Sin embargo, se debe comenzar con una letra. MQL4 distingue entre mayúsculas y minúsculas, es decir, **S** y **s** no son las mismas.

Ejemplo de nombres de variables: Alfa, alFa, beta, el_número, Num, A_37, A37, qwerty_123

Los siguientes identificadores de las variables representan nombres de variables diferentes: a_22 y A_22; Massa y MASSA.

Ejemplo de valores de variables: 37, 3,14, true, "Kazan".

Propiedades de las variables

La caracteristica de una variable es su capacidad para obtener un valor determinado del programa, conservalo durante todo el período de funcionamiento del programa y entregar este valor para uso del programa cuando este lo solicite. Para cada una de las variables del programa, el ordenador destina el tamaño necesario de una parte de su memoria.

Vamos a hacer referencia a la Fig. 7 y estudiar la construcción de una variable.



Fig. 7. Una variable en la memoria de un ordenador.

Tenemos el valor de una variable en la celda de memoria de la computadora. Este valor se puede leer y modificar por el programa. El nombre de una variable no cambia nunca. Al escribir un código, el programador puede poner cualquier nombre a la variable. Sin embargo, tan pronto como el programa se pone en marcha, ni el programador, ni el programa pueden cambiar el nombre de la variable.

Si el programa que se está ejecutando encuentra el nombre de una variable, el programa toma esta variable con el fin de obtener su valor para el proceso. Si un programa ha hecho referencia a una variable, esta fija su valor para el programa. Cuando el programa hace una copia del valor de la variable, este valor de la sigue siendo el mismo que figura en la celda de memoria asignada a esta variable (Fig. 8).

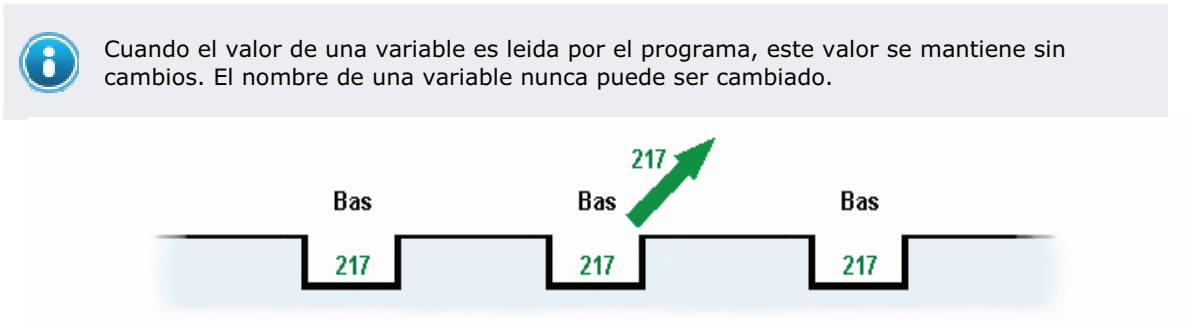


Fig. 8. El estado de la memoria celular de una variable la hora de fijar el valor al programa.

Una variable no está relacionada con la ejecución del programa durante un cierto período de tiempo. Durante este período, el programa puede referirse a otras variables o hacer otros cálculos necesarios. Entre las "sesiones" de comunicación con el programa, la variable conserva su valor, es decir, se mantiene sin cambios.

De acuerdo con el algoritmo del programa, puede ser necesario cambiar el valor de una variable. En este caso, el programa establece la variable a su nuevo valor, y la variable recibe el nuevo valor del programa. Todas las necesarias modificaciones se realizan en la memoria. Esto da lugar a la eliminación del anterior valor de la variable mientras el nuevo valor fijado por el programa toma su lugar. (Fig. 9).

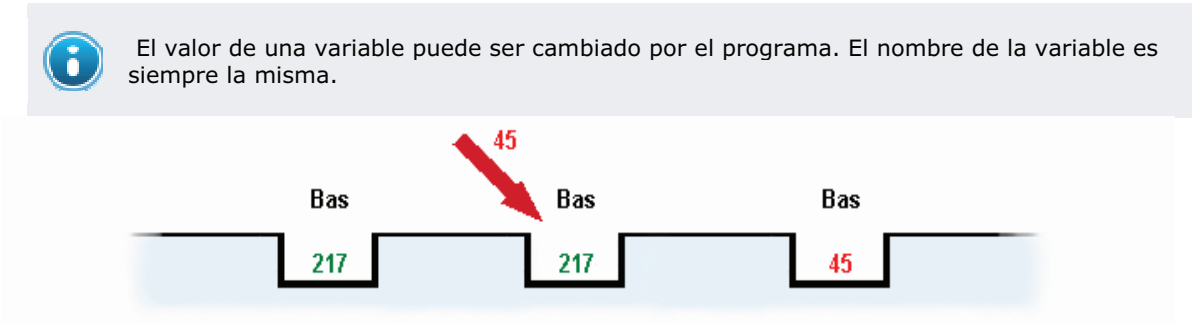


Fig. 9. El estado de la memoria celular de una variable al obtener el valor del programa.

Ejemplo de Constantes y Variables en un programa

Constantes y variables se pueden encontrar en los operadores de un programa. En el código que aparece a continuación, A y B son variables, 7 y 3 son constantes:

```
A = 7; // Línea 1
B = A + 3; // Línea 2
```

Vamos a estudiar la forma de un programa trabaja con constantes y variables. Al ejecutar estas líneas, el programa hará los siguientes pasos:

Línea 1:

1. La constante 7 establece su valor al programa.
2. Una variable recibe el valor 7 del programa.

Línea 2:

1. El programa ha encontrado una expresión a la derecha del signo igual y está tratando de calcularlo.
2. Constante 3 establece su valor al programa.
3. El programa hace referencia a la variable A por su nombre.
4. Una variable fija su valor (7) al programa.
5. El programa hace los cálculos (7 + 3).
6. Variable B obtiene el valor 10 del programa.

El valor de una variable puede ser cambiado durante la operación del programa. Por ejemplo, puede haber una línea en el programa que contenga lo siguiente:

```
B = 33 // Línea 3
```

En este caso, los siguientes pasos se llevarán a cabo en ejecución del programa:

1. Constante 33 establece su valor al programa.
2. Variable B establece el valor 33 (nuevo) del programa.

Es fácil notar que la variable B recibe el valor 10 en una cierta fase de la ejecución del programa y, a continuación, recibe el valor de 33. El nombre de la variable B no ha cambiado durante todos estos acontecimientos mientras que el valor de la variable va a cambiar.

Fig. 10 muestra las constantes y variables en el código de programa:

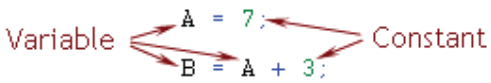


Fig. 10. Un constante y una variable en un programa.

Tipos de datos

Se trata de un conocimiento común que sólo tipos iguales de valores pueden ser sumados o restados. Por ejemplo, las manzanas se pueden añadir a las manzanas, pero las manzanas no pueden sumarse a metros cuadrados o grados centigrados. Similares limitaciones se encuentran en la mayoría de los modernos lenguajes algorítmicos.

Al igual que los objetos normales de la vida tienen determinados tipos que caracterizan su color (rojo, azul, amarillo, verde), su sabor (amargo, ácido, dulce), cantidad (una y media, dos, siete), MQL4 utiliza datos de diferentes tipos. Al hablar de tipo de datos, se entenderá el tipo del valor de una constante, de una variable y el valor devuelto por una función (la noción de función es considerado en la sección de Funciones).

En MQL4, se distinguen los siguientes tipos de datos (para los valores de las constantes, variables, y los valores devueltos por funciones):

- **int** – números enteros;
- **double** - números reales;
- **bool** - Bolean de valores logicos;
- **string**- valores de tipo cadena de caracteres;
- **color** - valores de tipo color ;
- **datetime** - valores de fecha y hora.

Tipo int

Los valores de tipo **int** son enteros. Este tipo incluye los valores que por su naturaleza son de tipo entero. Los siguientes valores son enteros, por ejemplo: cantidad de barras en la ventana de símbolo o instrumento (16000 barras), distancia entre puntos en el símbolo actual de precios y el precio de apertura de la orden (15 puntos). Las cantidades que representan esos objetos como eventos también pueden ser sólo números enteros. Por ejemplo, la cantidad de intentos para abrir una orden no puede ser igual a uno y medio, sólo a uno, dos, tres, etc.

Hay 2 tipos de valores:

- **Decimal** estos valores se presentan en forma de dígitos del 0 al 9 y pueden ser positivos o negativos: 10, 11, 12, 1, 5, -379, 25, -12345, -1, 2.
- Los valores **hexadecimales** estan formados por las letras de la A a la F y los dígitos del 0 al 9. Deben comenzar con 0x o 0x y tomar valores positivos o negativos: 0x1a7b, 0xff340, 0xAC3 0X2DF23, 0X13AAB, 0x1.

Los valores de tipo **int** deben estar dentro del rango de -2 147 483 648 a 2 147 483 647. Si el valor de una constante o una variable está fuera de este rango, el resultado de la operación del programa será nulo. Los valores de constantes y variables de tipo **int** ocupan 4 bytes en la memoria del ordenador.

Un ejemplo que utiliza variable de tipo **int** en un programa:

```
int Art = 10; // Ejemplo variable integer
int B_27 = - 1; // Ejemplo variable integer
int num = 21; // Ejemplo variable integer
int Max = 2147483647 // Ejemplo variable integer
int min = - 2147483648; // Ejemplo variable integer
```

Tipo double

Los valorer de tipo **double** son números reales que contienen una parte decimal.

Los valores de ejemplo de este tipo puede ser cualquier valor que tengan una parte decimal: inclinación de la línea de apoyo, símbolo de precios con una media de cantidad de órdenes abiertas dentro de un día.

A veces se pueden afrontar los problemas designando las variables al escribir su código, es decir, no siempre es evidente para un programador a qué tipo **(int** o **double)** pertenece la variable. Veamos un pequeño ejemplo:

Un programa ha abierto 12 órdenes en una semana. ¿Cuál es el tipo de una variable que considera la cantidad media diaria de órdenes abiertas por este programa? La respuesta es obvia: A = 12 órdenes / 5 días. Esto significa que la variable A = 2,4 deben ser consideradas en el programa como **double,** ya que este valor tiene una parte fraccional. ¿Qué tipo debe ser la misma variable A, si el importe total de órdenes que abrió en el plazo de una semana es de 10? Se puede pensar que si 2 (10 órdenes / 5 días = 2) no tiene parte decimal, una variable puede ser considerada como **int.** Sin embargo, este razonamiento es erróneo. El valor actual de una variable puede tener una pequeña parte que consta sólo de ceros. Es importante saber que el valor de esa variable es real, por su propia naturaleza. En este caso, una variable también ha de ser de tipo **double**. La separación del punto decimal también debe ser mostrada en el registro del programa: A = 2,0

Los valores reales de constantes y variables consistirá en una parte entera, un punto decimal, y una parte decimal. Los valores pueden ser positivos o negativos. La parte entera y la parte decimal se forman con los dígitos del 0 al 9. La cantidad de cifras significativas después del punto decimal puede alcanzar el valor de 15.

Ejemplo:
27,12 -1,0 2,5001 -765456,0 198732,07 0,123456789012345

Los valores de tipo **double** puede oscilar entre -1,7 * e-308 a 1,7 * e308. En la memoria de ordenador, los valores de constantes y variables de tipo **double** toman 8 bytes.

Un ejemplo de utilizar una variable de tipo **double** en un programa:

```
double Arte = 10,123; // Ejemplo de variable real
double B_27 = - 1,0; // Ejemplo de variable real
double Num = 0,5; // Ejemplo de variable real
doble MMM = - 12,07 // Ejemplo de variable real
doble Price_1 = 1.2756; // Ejemplo de variable real
```

Tipo bool

Los valores de tipo **bool** son valores de Boléanos (lógicos) que contienen valores del tipo **true** (verdadero) o **false** (falso).

Con el fin de aprender la noción de los tipos boléanos, vamos a examinar un pequeño ejemplo de nuestra vida cotidiana. Digamos, un profesor necesita para tener en cuenta la presencia de los libros de texto de los alumnos. En este caso, el profesor, confeccionará una lista de todos los alumnos en una hoja de papel y, a continuación se marca en línea si un alumno tiene libro de texto o no. Por ejemplo, el profesor podrá utilizará marcas y guiones en la gráfica:

Lista de alumnos		Libro de texto de Física	Libros de Texto en Biología	Libro de texto en Química
1	Smith	V	--	--
2	Jones	V	--	V
3	Marrón	--	V	V
...	...	...	...	...
25	Thompson	V	V	V

Los valores en las columnas pueden ser sólo de 2 tipos: verdadero o falso. Estos valores no se pueden atribuir a cualquiera de los tipos considerados anteriormente, ya que no son números en absoluto. No son los valores de color, sabor, cantidad, etc. En MQL4, esos valores se denominan boléanos, o valores lógicos. Constantes y variables de tipo **bool** se caracterizan por que sólo pueden tomar 2 valores posibles: **true** (Es cierto, TRUE, 1) o **false** (false, false, 0). Los valores de constantes y variables de tipo **bool** ocupan 4 bytes en la memoria de ordenador.

Ejemplo que utiliza una variable de tipo **bool** en un programa:

```
bool aa = True; // la variable Boolean aa tiene el valor de verdadero
bool B17= TRUE // la variable Boolean B17 tiene el valor de verdadero
bool Hamma = 1; // la variable Boolean Hamma tiene el valor de verdadero

bool TEA = False; // la variable Boolean TEA tiene el valor de falso
bool Nol = false; // la variable Boolean Nol tiene el valor de falso
bool Prim = 0; // la variable Boolean Prim tiene el valor de falso
```

Tipo string (cadena de caracteres)

El valor de tipo **string** es un valor representado como un conjunto de caracteres ASCII.

En la vida cotidiana, un contenido similar pertenecen aquellos tipos que, por ejemplo, almacenar nombres, coches, etc Un un valor tipo **string** se registra como un conjunto de caracteres colocados entre comillas dobles (no para ser mezclado con dobles comillas simples). Las comillas se utilizan sólo para marcar el comienzo y el final de una constante de tipo **string**. El valor en sí es el conjunto de caracteres enmarcados por las comillas.

Si hay necesidad de introducir dobles comillas ("), se debe poner una barra diagonal inversa (\) antes. Cualquier carácter especial puede ser introducido en una constante de tipo **string** tras la barra inversa (\). La longitud de una constante de tipo **string** va de 0 a 255 caracteres. Si la longitud de una constante de tipo **string** es superior a su máximo, el exceso de caracteres en el lado derecho se trunca y el compilador dará el correspondiente aviso. Una combinación de dos caracteres, el primero de los cuales es la barra inversa (\), es comúnmente aceptado y percibido por la mayoría de los programas como una instrucción para ejecutar un determinado formato de texto. Esta combinación no se muestra en el texto. Por ejemplo, la combinación de \ n indica la necesidad de un salto de línea; \ t demanda de tabulación, etc.

El valor de tipo **string,** se registra como un conjunto de caracteres enmarcados por comillas dobles: "MetaTrader 4", "Detener la Pérdida", "Ssssstop_Loss", "stoploss", "10 pips". El valor de la cadena como tal, es el conjunto de caracteres. Las comillas se utilizan sólo para enmarcar las fronteras de la cadena de caracteres. La representación interna es una estructura de 8 bytes.

Ejemplo de utilización de una variable de tipo **string** en un programa:

```
string prefix = "MetaTrader 4"; // Ejemplo variable string
string Postfix = "_of_my_progr. OK"; // Ejemplo variable string
string Name_Mass = "Historial"; // Ejemplo variable string
string texto = "Línea de Alta \n Bajo la línea" //el texto contiene caracteres  de salto de línea
```

Tipo color

El valor del tipo **color** es un valor cromático.

El significado de 'color' (azul, rojo, blanco, amarillo, verde, etc) es de conocimiento común. Es fácil imaginar lo que una variable o una constante de tipo **color** puede significar. Es una constante o una variable, cuyo valor es un color. En términos generales puede parecer ser un poco inusual, pero es muy simple. Al igual que el valor de una constante de tipo entero es un número, el valor de una constante color es un color.

Los valores constantes y variables de tipo color pueden ser representados con una de tres formas distintas:

- **Literales**

  El valor de tipo **color** es representado como un literal y consta de tres partes que representan los valores numéricos de intensidad de tres colores básicos: rojo, verde y azul (RGB). Un valor de este tipo empieza con "C" y el valor numerico esta encerrado entre comillas simples.

  Los valores numéricos RGB de intensidad de 0 a 255 y se pueden grabar tanto en decimal como en hexadecimal.

  Ejemplos: C'128128128'(gris), C'0x00, 0x00, 0xFF' (azul), C'0xFF, 0x33, 0x00’(rojo).

- **Representación Integer** (Representación por enteros)

  La representación integer se registra como número hexadecimal o un número decimal. Un número hexadecimal se muestra como 0xRRGGBB, donde RR es el valor de intensidad de color rojo; GG, verde; y BB, azul. Las constantes decimales no se reflejan directamente en RGB. Representan el valor decimal de un número entero en representación hexadecimal.

  La representación de los valores de tipo color enteros y literales como numeros hexadecimales es muy fácil de usar. La mayoría de los textos modernos y editores gráficos proporcionar información sobre la intensidad de rojo, verde y azul en el valor seleccionado de color. Sólo se tiene que seleccionar un color en su editor y copiar los valores encontrados en su descripción correspondiente a la representación del valor color en su código.

  Ejemplos: 0xFFFFFF (blanco), 0x008000 (verde), 16777215 (blanco), 32.768 (verde).

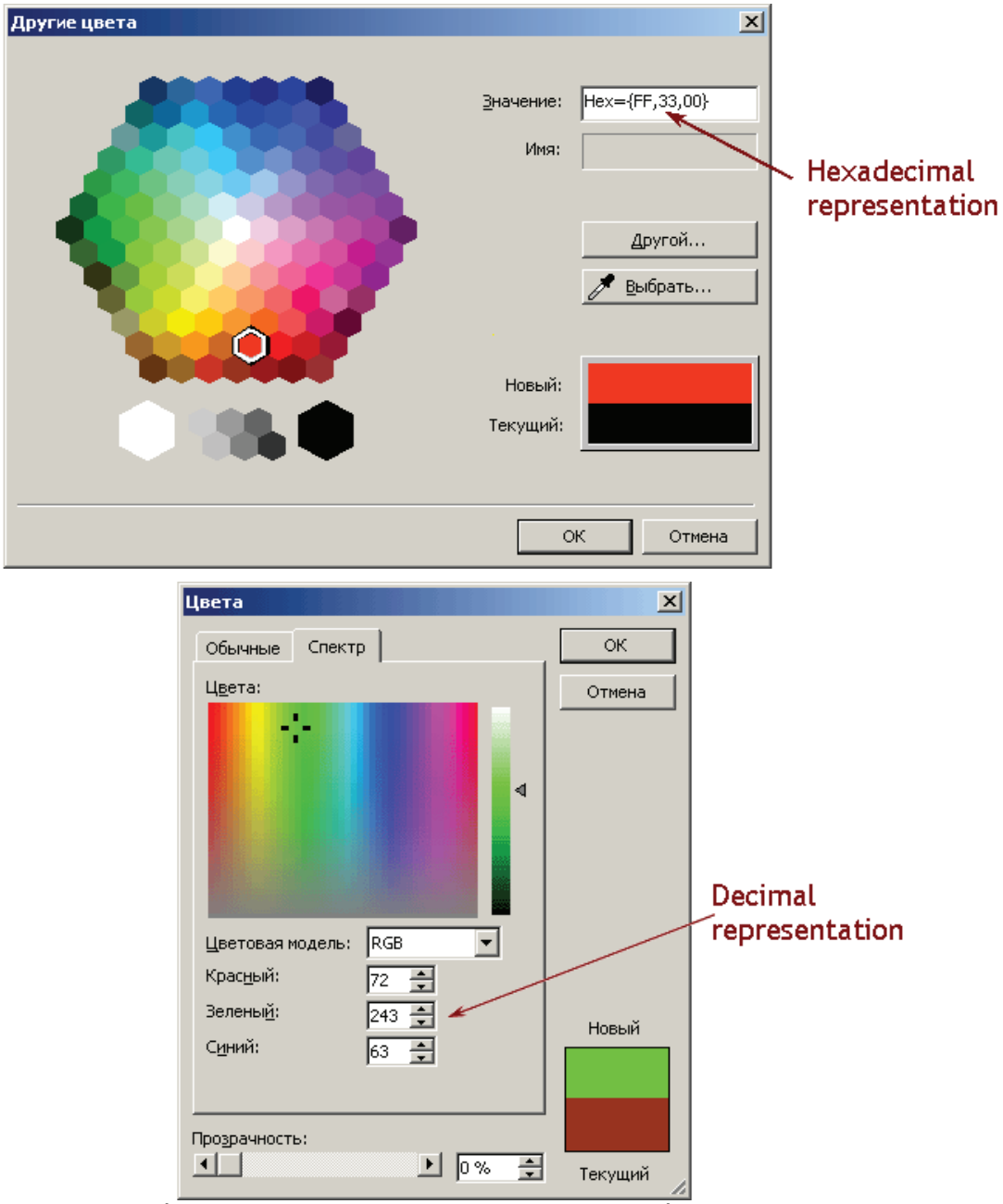


Fig. 11. Parámetros Color para literal entero y representación del valor constante de color que se pueden tomar en los modernos editores.

- **Nombres de colores**

La forma más fácil de definir colores es especificar su nombre de acuerdo con la gráfica de colores web. En este caso, el valor de un color se representa como una palabra que corresponda con el color, por ejemplo, Red - el color rojo.

Black	DarkGreen	DarkSlateGray	Olive	Green	Teal	Navy	Purp
Maroon	Indigo	MidnightBlue	DarkBlue	DarkOliveGreen	SaddleBrown	ForestGreen	OliveD
eaGreen	DarkGoldenrod	DarkSlateBlue	Sienna	MediumBlue	Brown	DarkTurquoise	DimG
tSeaGreen	DarkViolet	FireBrick	MediumVioletRed	MediumSeaGreen	Chocolate	Crimson	SteelB
oldenrod	MediumSpringGreen	LawnGreen	CadetBlue	DarkOrchid	YellowGreen	LimeGreen	Orang
rkOrange	Orange	Gold	Yellow	Chartreuse	Lime	SpringGreen	Aqu
epSkyBlue	Blue	Magenta	Red	Gray	SlateGray	Peru	BlueV
tSlateGray	DeepPink	MediumTurquoise	DodgerBlue	Turquoise	RoyalBlue	SlateBlue	DarkK
ndianRed	MediumOrchid	GreenYellow	MediumAquamarine	DarkSeaGreen	Tomato	RosyBrown	Orch
diumPurple	PaleVioletRed	Coral	CornflowerBlue	DarkGray	SandyBrown	MediumSlateBlue	Tan
rkSalmon	BurlyWood	HotPink	Salmon	Violet	LightCoral	SkyBlue	LightSa
Plum	Khaki	LightGreen	Aquamarine	Silver	LightSkyBlue	LightSteelBlue	LightB
aleGreen	Thistle	PowderBlue	PaleGoldenrod	PaleTurquoise	LightGray	Wheat	NavajoW
Moccasin	LightPink	Gainsboro	PeachPuff	Pink	Bisque	LightGoldenrod	Blanched
nonChiffon	Beige	AntiqueWhite	PapayaWhip	Cornsilk	LightYellow	LightCyan	Lime
avender	MistyRose	OldLace	WhiteSmoke	Seashell	Ivory	Honeydew	AliceB
enderBlush	MintCream	Snow	White				

Las constantes y variables de tipo **color** toman 4 bytes en la memoria de ordenador.

Ejemplo de la utilización de esa variable en un programa:

```
color Paint_1 = C '128, 128, 128'; // El valor gris se asignó a la variable
color Colo= C '0 x00, 0 x00, 0 xff "// El valor azul fue asignado a la variable
color BMP_4  = C '0 xff, 0 x33, 0 x00 "// El valor rojo fue asignado a la variable

color K_12= 0 xFF3300; // El valor rojo fue asignado a la variable
color N_3 = 0 x008000; // El valor verde fue asignado a la variable
color Color = 16777215; // El valor blanco se asignó a la variable
color Alfa = 32768; // El valor verde fue asignado a la variable


color Un = Rojo; // El valor rojo fue asignado a la variable
color B = amarillo; // El valor amarillo fue asignado a la variable
color Colorit = Negro // El valor negro fue asignado a la variable
color B_21 = Blanco // El valor blanco se asignó a la variable
```

Tipo datetime

El valor de tipo **datetime** es un valor de fecha y hora.

Los valores de este tipo puede ser utilizado en los programas para analizar el momento de inicio o finalización de algunos eventos, entre ellos las emisiones de noticias importantes, de trabajo de inicio / finalización, etc. Las constantes de fecha y hora se pueden representar como una línea de un literal constituido de 6 partes que representan el valor numérico del año, mes y día (o día, mes, año), hora, minuto y segundo.

La constant se enmarca entre comillas simples y comienza con 'D'. Está permitido el uso truncado de valores: o bien sin fecha o sin tiempo, o simplemente un valor vacío. El rango de valores va desde el 1 de enero de 1970 al 31 de diciembre de 2037. Los valores de constantes y variables de tipo **datetime** ocupan 4 bytes en la memoria de ordenador. Un valor representa la cantidad de segundos transcurridos desde las 00:00 del 1 de enero de 1970.

Un ejemplo de utilización una variable de tipo **datetime** en un programa:

```
datetime Alfa = D '2004.01.01 00: 00' // Año Nuevo
datetime Tim = D "01.01.2004"; // Año Nuevo
datetime Tims = D '2005.05.12 16: 30: 45'; // 12 de Mayo de 2005 a las 4:30:45 PM
datetime N_3 = D '12/05/2005 16: 30: 45'; // 12 de Mayo de 2005 a las 4:30:45 PM
datetime Compilar = D''; //equivalente de D '[compilación fecha] // 00:00:00
```

Declaración de variables e inicialización

Con el fin de evitar posibles preguntas por el programa acerca de qué tipo de datos pertenece de tal o cual a variable, MQL4 acepta que se especifiquen explícitamente los tipos de variables al inicio del programa. Antes de que una variable empiece a utilizarse en cualquier cálculo deber ser declarada.

La Declaración de Variables es lo primero que se debe hacer con cualquier variable dentro de un programa. En la declaración de una variable siempre ha de especificarse su tipo.

La inicialización de Variables significa la asignación de un valor acorde con su tipo y que se efectua en su declaración. Todas las variables pueden ser inicializadas. Si no hay valor inicial que se establezca explícitamente, la variable se inicializa a cero (0), o si la variable es de tipo string, esta se inicializa como una cadena de caracteres vacía.



En MQL4 se acepta que se especifiquen los tipos de variables explícitamente en su declaración. El tipo de una variable solo se declara en la primera mención del nombre de esta variable. Cuando se menciona el resto de las veces su tipo ya no se vuelve especificar más. En el curso de la ejecución del programa, el valor de la variable puede cambiar, pero su tipo y nombre siguen siendo los mismos. El tipo de una variable puede ser declarada en líneas simples o en los operadores.

Una variable puede declararse en una sola línea:

```
int Var_1; // declaración de variable en una sola línea
```

Este registro significa que existirá la variable **Var_1** (declaración de variable como tal) y el tipo de esta variable en el programa será **int**.

Pueden ser declaradas varias variables en una sola línea.

```
int Var_1, Box, Com // Declaración de varias variables en una línea
```

Este registro significa que en el programa tendremos las variables Var_1, Box y Com, todas de tipo **int**. Esto significa que las variables enumeradas anteriormente serán consideradas por el programa como variables de tipo entero.

Las variables pueden también ser inicializado dentro de los operadores:

```
double Var_5 = 3,7; // Variable inicializada en un operador de asignación
```

Este registro significa que existirá una variable Var_5 de tipo **double** utilizada en el programa y cuyo valor inicial es de 3,7.

El tipo de variable ya no se especifica más en ninguna parte de las líneas siguientes del programa. Sin embargo, cada vez que el programa llama a una variable, "recuerda" que esa variable es del tipo que se ha especificado en su declaración. A medida que avanza el programa los valores de las variables pueden cambiar con los cálculos pero no así su tipo.

El nombre de una variable no tiene relación con su tipo, es decir, no se puede juzgar sobre el tipo de una variable por su nombre. Un nombre dado a una variable también puede utilizarse en variables de todo tipo en diferentes programas. Sin embargo, el tipo de variable solo puede ser declarada una vez dentro de un programa. El tipo de variables declaradas no se modificará durante la ejecución del programa.

Ejemplos de declaración e inicialización de variables

Las variables pueden ser declaradas en varias líneas o en una sola línea.

Se pueden declarar varias variables de un mismo tipo simultáneamente. En este caso, se enumeran las variables separadas por comas, al final de línea se colocará un punto y coma.

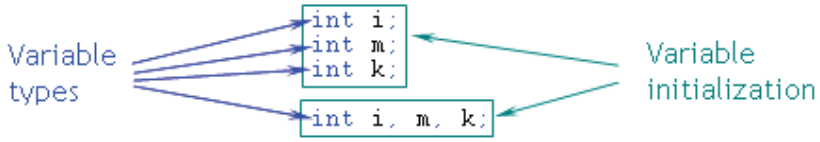


Fig. 12. Ejemplo de declaración de variables en una sola línea.

El tipo de variable se declara una sola vez, en la primera mención de la variable. El tipo no se vuelve a especificar más ni cuando el resto de las veces se menciona la variable.

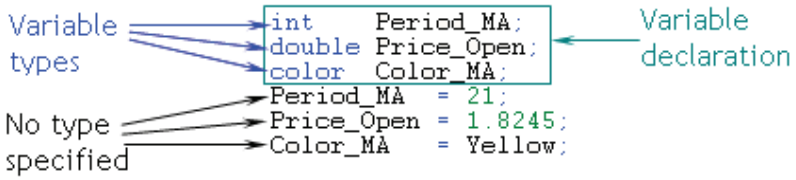


Fig. 13. Ejemplo de declaración de variables en una sola línea.

Se permite declarar e inicializar las variables en los operadores.

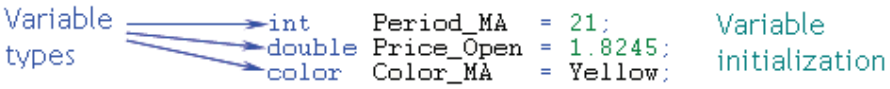


Fig. 14. Ejemplo de inicialización de variables.

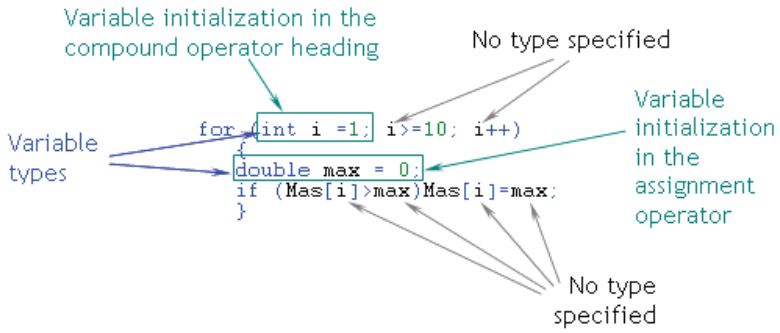


Fig. 15. Inicialización de variable en la cabecera de un operador compuesto.

Operaciones y expresiones

No son necesarias analogías especiales con el fin de comprender la importancia que las operaciones y las expresiones tienen en MQL4. Prácticamente, son las mismas que las operaciones y las expresiones de la simple aritmética. Todo el mundo entiende que en en el registro **f = n + m,** los miembros **f**, **n** y **m** son variables y los signos **+** y **=** son los signos de la operación, mientras que **n + m** es una expresión.

En la sección anterior del libro, aprendimos la necesidad de presentar los datos de diferentes tipos. Aquí vamos a entrar en las posibles relaciones entre estos datos (metros cuadrados no se pueden sumar a manzanas). En MQL4, hay algunas reglas y naturales limitaciones de uso de las operaciones de las expresiones.

Nociones de 'operando', 'Operación', ' Símbolo de la Operación ' y 'Expresión'

Operando es una constante, una variable, un conjunto de componentes o un valor devuelto por una función (el término de función se considera en la sección de Funciones, el de array, en la sección de Arrays; en la presente etapa de aprendizaje, es suficiente comprender los operandos como constantes y variables los cuales ya hemos estudiado antes).

Operación es una acción hecha con los operandos.

Símbolo de la Operación es un carácter preseleccionado o un grupo de caracteres que ordenan ejecutar una operación.

Expresión es una secuencia de operandos y operaciónes de símbolo, es un registro de programa, el valor calculado del cual se caracteriza por un tipo de datos.

Tipos de Operaciones

Existen los siguientes tipos de operaciones en MQL4:

- operaciones aritméticas;
- operaciones de asignación;
- operaciones relacionales;
- operaciones Boolean (lógicas);
- operaciones bitwise;
- operación coma;
- llamada a función.

Las operaciones se utilizan en los operadores (ver Operadores). Sólo en los operadores su utilización tiene sentido y se realiza en un programa. La posibilidad de utilizar una operación está determinada por las propiedades de los operadores (si las propiedades del operador le permiten utilizar esta operación específica, puede usarlo, de lo contrario, usted no debe utilizar esta operación). No está permitido el empleo de las operaciones fuera de los operadores.

Operaciones aritméticas

Los siguientes símbolos pertenecen a símbolos de operaciones aritméticas:

Símbolo	Operación	Ejemplo	Analógica
+	Adición de valores	x + 2	
-	La resta de valores o de cambio de signo	x - 3, y = - y	
*	Multiplicación de valores	3 * x	
/	Cociente de la división	x / 5	
%	Resto de la división	minutos = tiempo 60%	
+ +	Incrementar 1 el valor de la variable	y + +	y = y + 1
--	Decrementar 1 el valor de la variable	y --	y = y - 1

Operaciones de Asignación

Los siguientes símbolos pertenecen a símbolos de operaciones de asignación:

Símbolo	Operación	Ejemplo	Analógica
=	Asignación del valor x a la variable y	y = x	
+ =	Aumento de la variable y en el valor x	y + x =	y = y + x
-=	Reducción de la variable y en el valor x	y -= x	y = y - x
*=	Multiplicación de la variable y por x	y *= x	y = x * y
/ =	División de la variable y entre x	y / x =	y = y / x
% =	Residuo de la división de la variable y por x	y = x%	y = x% y

Operaciones Relacionales

Los siguientes símbolos pertenecen a los símbolos de operaciones relacionales:

Símbolo	Operación	Ejemplo
==	Es cierto si x es igual a y	x == y
! =	Es cierto si x no es igual a y	x! = y
<	Es cierto si x es menor que y	x <y
>	Es cierto si x es más y más	x> y
<=	Es cierto si x es igual o inferior a y	x <= y
>=	Es cierto si x es igual o superior y	x> = y

Operaciones Boolean (lógicas)

Los siguientes símbolos pertenecen a los símbolos de operaciones booleanas:

Símbolo	Operación	Ejemplo	Explicaciones
!	NO (negación lógica)	! x	TRUE (1), si el valor del operando es FALSO (0); FALSO (0), si el valor del operando no es FALSO (0).
||	O (disyunción lógica)	x <5 || x>7	TRUE (1), si algúna de las dos expresiones es cierta
&&	Y (conjunción lógica)	x==3 && y<5	TRUE (1), si las dos expresiones son ciertas

Operaciones Bitwise

Las operaciones Bitwise sólo pueden realizarse con números enteros. Las siguientes operaciones pertenecen a operaciones bitwise:

Complemento a uno del valor de la variable. El valor de la expresión contiene 1 en todos los lugares, en los cuales los valores de la variable contienen 0, y contienen 0 en todos los lugares, en los cuales los valores de la variable contienen 1.

```
b= ~ n;
```

La representación binaria de x que es desplazada y lugares a la derecha. Este desplazamiento lógico a la derecha, significa que en todos los lugares que se han vaciado a la izquierda será rellenado con ceros.

```
x = x>>y;
```

La representación binaria de x que es desplazada y lugares a la izquierda. Este desplazamiento lógico a la izquierda será rellenado con ceros a la derecha.

```
x = x<<y;
```

La operación bitwise AND de las representaciones binarias de x e y. El valor de la expresión contiene 1 (TRUE) en todos los lugares, en tanto que x e y contienen uno, y el valor de la expresión contiene 0 (FALSO) en todos los demás casos.

```
b = ((x + y)! = 0);
```

La operación bitwise OR de las representaciones binarias de x e y. El valor de la expresión contiene 1 en todos los lugares, en la que x o y contienen 1. Contiene 0 en todos los demás casos.

```
b = x | y;
```

La operación bitwise exclusiva o de las representaciones binarias de x e y. El valor de la expresión contiene 1 en los lugares, en los que x e y tienen diferentes valores binarios. Contiene 0 en todos los demás casos.

```
b = x ^ y;
```

Operación de la coma

Las expresiones separadas por comas se calculan de izquierda a derecha. Los efectos de los cálculos a la izquierda de la expresión ocurren siempre antes de que se calcule el lado derecho de la expresión. El tipo y el valor del resultado son coincidentes con el tipo y el valor del lado derecho de expresión.

```
for (i = 0, j = 99; i <100; i++, j--) Imprimir (array [i] [j]); // Loop declaración
```

La lista de parámetros transferidos (véase más adelante) puede considerarse como un ejemplo.

```
My_function (Alf, Bet, Gam, Del)  // La llamada a una función con argumentos
```

Los operadores y las funciones se consideran en las secciones de Operadores y Funciones y en el capítulo de Operadores

Función Call

llamada a la función Call se describe en todos sus detalles en la sección de Función Call.

Operaciones sobre operandos similares

Si en una escuela primaria a los alumnos se les dice que, al resolver el problema sobre el número de lápices, él o ella tendría que basar su presentación en los términos tales como operandos, operadores y expresiones, a los pobres alumnos, sin duda, les resulta imposible. En cuanto a los símbolos de las operaciones, uno puede pensar que la codificación es un misterioso y complicado proceso, accesibles sólo para una especie de elite. Sin embargo, la codificación no es realmente difícil en absoluto, sólo tiene que hacer la cola o la cabeza de algunas intenciones. Para estar seguro de que esto es realmente así, vamos a examinar algunos ejemplos.



Problema 1. Juan tiene 2 lápices, Pedro tiene 3 lápices. ¿Cuántos lápices tienen estos muchachos?:)

Solución. Vamos a indicar el número de lápices de Juan como una variable A y el número de lápices de Pedro como variable B, mientras que el resultado será denominado C.

La respuesta será: A + B=C

En la sección de nombre Tipos de datos, hemos considerado los métodos de declaración de variables. Los lápices son las cosas, es decir, es algo que, básicamente, puede existir como una parte (por ejemplo, puede haber una mitad de un lápiz). Por lo tanto, vamos a considerar como lápices, variables reales, es decir, las variables de tipo *double.*

Por tanto, podemos poner el código de la solución como sigue, por ejemplo:

```
double A = 2,0; // El número de lápices de Juan
double B = 3,0; // El número de lápices de Pedro
double C = A + B // Número total
```

En este caso, la operación "+" se aplica a la suma de los valores de un tipo de variables.

El tipo de valor de la expresión:

```
A + B
```

Será de mismo tipo que el de las variables que componen de la expresión. En nuestro caso, este será de tipo *double*.

Vamos a obtener la respuesta similar para la diferencia entre los valores (¿Cuántos lápices tiene Pedro más que Juan?):

```
double A = 2,0; // El número de lápices de Juan
double B = 3,0; // El número de lápices de Pedro
double C = B - A; // La diferencia entre dos números reales
```

Otras operaciones aritméticas se utilizan de manera similar:

```
double C = B * A; // Multiplicación de dos números reales
double C = B / A // División de dos números reales
```

Cálculos similares pueden realizarse con números enteros también.



Problema 2. Los alumnos van a la pizarra y responden en clase. Juan fue 2 veces, Pedro fue 3 veces. ¿Cuántas veces fueron en total los muchachos a la pizarra?

Solución. Vamos a denotar los viajes de Juan como variable X y viajes de Pedro como variable Y, el resultado como Z.

En este ejemplo, tenemos que utilizar las variables de tipo *int,* ya que consideramos que son eventos entero por su propia naturaleza (no se puede ir a la pizarra 0,5 veces o 1,5 veces; la respuesta a la pizarra puede ser buena o mala, pero sólo interesa el número de esas respuestas o viajes).

La solución de este problema puede escribirse como:

```
int X = 2;                  // Número de viajes de Juan
int Y = 3;                  // Número de viajes de Pedro
int Z = X + Y;              // Número total
```

En el caso de cálculo de la diferencia entre los productos o de cociente de enteros, la operación "-" se utiliza en la forma simple de modo similar:

```
int X = 2;                  // Entero
int Y = 3;                  // Entero
int Z = Y - X;              // Diferencia entre dos enteros
int Z = Y * X;              // Productpo de dos enteros
int Z = Y / X;              // Cociente de dos enteros
```

La situación es algo diferente con las variables de tipo string (*cadena de caracteres):*



Problema 3. En una esquina de la casa, hay una carnicería denominada "Ártico". En otra esquina de la misma casa, hay un establecimiento llamado "Salón de peluquería". ¿Qué está escrito en la casa?

Solución. En MQL4, se permite agregar, junto a la cadena de valores constantes y variables. Si se suman las variables de tipo *cadena,* las cadenas simplemente se añadirán una a otra en la secuencia que se mencionan en la expresión.

Es fácil codificar un programa que nos diera la respuesta:

```
string W1  = "Artico";                  // String 1
string W2  = "Salon de peluqueria";   // String 2
string Ans = W1 + W2;                  // Suma de strings
```

El valor de la variable Ans será la cadena que aparece como sigue:

```
ArticoSalon de peluqueria
```

Se obtuvo una cadena no de muy buen ver, pero absolutamente correcta. Por supuesto, hay que tener en cuenta las lagunas y otros puntuacion en nuestra práctica de codificación de este tipo de problemas.

El resto de operaciones aritméticas con variables de tipo cadena no están permitidas. Por ejemplo:

```
string Ans = W1 - W2 // No permitido
string Ans = W1 * W2; // No permitido
string Ans = W1 / W2 // No permitido
```

Typecasting

El **Typecasting** modifica los tipos de los valores de un operando o de una expresión. Antes de la ejecución de las operaciones (todas las operaciones de asiganción), los valores son modificados al del tipo de más alta prioridad, mientras que antes de la ejecución del signo de asignación los valores se modifican para el tipo objetivo.

Vamos a considerar algunos problemas que se ocupan de typecasting.



Problema 4. Juan tiene 2 lápices, mientras que Pedro se fue 3 veces a la pizarra. ¿Cuántos en total?

Por lo que respecta a la lógica formal se refiere, la incongruencia del problema es evidente. Es evidente que es un error pues no se pueden sumar acontecimientos y cosas.



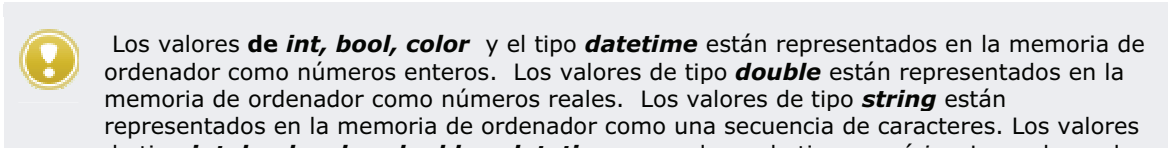
Problema 5. En una esquina de la casa, hay una carniceria denominada "Ártico", mientras que Juan tiene 2 lápices. :)

Con el mismo grado de desesperanza (en la medida que se trate de un normal razonamiento) podemos preguntar:

1. ¿Cuántos en total?, O
2. ¿Qué está escrito en la casa?

Si desea resolver ambos problemas anteriormente en MQL4 correctamente, usted debe hacer referencia a las normas typecasting. En primer lugar, tenemos que hablar de cómo variables de diferentes tipos están representadas en la memoria de ordenador.

Los tipos de datos, tales como ***int, bool, color, double* y *datetime***, pertenecen al tipo de datos *numéricos.* La representación interna de constantes y variables de *int, double, bool, color* y el tipo *datetime* es un número. Las variables de tipo ***int, bool, color* y *datetime*** están representadas en la memoria de ordenador como enteros, mientras que las variables de tipo *double* están representadas como valores de doble precisión, como números en coma flotante, es decir, números reales. El valor de constantes y variables de tipo ***string*** son un conjunto de caracteres (Fig. 16).



Los valores **de *int, bool, color*** y el tipo ***datetime*** están representados en la memoria de ordenador como números enteros. Los valores de tipo ***double*** están representados en la memoria de ordenador como números reales. Los valores de tipo ***string*** están representados en la memoria de ordenador como una secuencia de caracteres. Los valores de tipo ***int, bool, color, double*** y ***datetime*** son valores de tipo *numérico.* Los valores de tipo ***string*** son valores de tipo *carácter.*

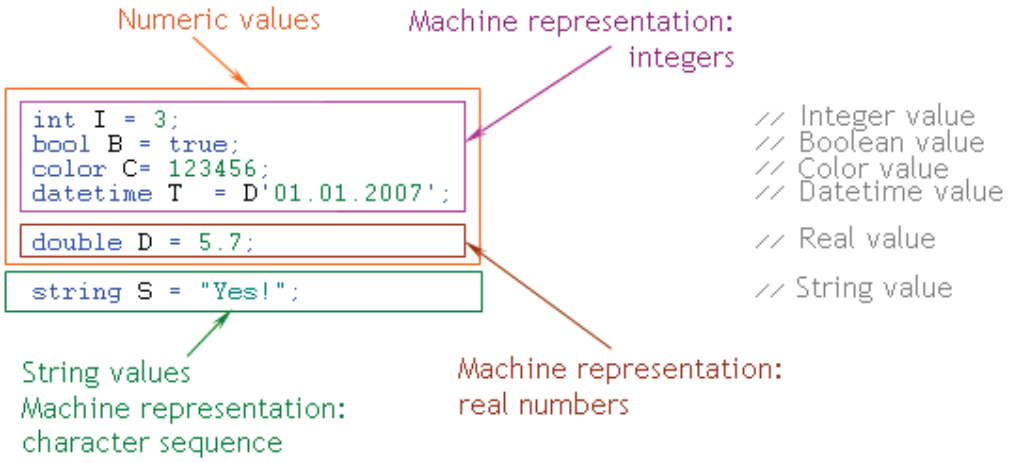


Fig. 16. Representación de diferentes tipos de datos en la memoria de ordenador.

Hemos mencionado anteriormente que los valores de las variables de tipo ***int, bool, color*** y ***datetime*** están representados en la memoria de ordenador como enteros, mientras que los de tipo ***double*** como números reales.Por lo tanto, si queremos encontrar información de una expresión que consta de variables de diferentes tipos, sólo podemos elegir entre tres tipos de datos: ***int, double y string***. Los valores de ***bool, color*** y el tipo ***datetime*** resultarán en una expresión iguales que los valores de tipo ***int***.

Entonces, ¿de qué tipo será el valor de una expresión compuesta de operandos con diferentes tipos? En MQL4, la regla implícita de typecasting es la siguiente:

- Si la expresión contiene operandos de diferentes tipos, el tipo de expresión se transformará en el tipo que tenga la más alta prioridad; los tipos ***int, bool, color*** y ***datetime*** tienen la misma prioridad, mientras que el operando de tipo ***double*** tiene una prioridad mayor, y el operando tipo ***string*** es el que tiene la más alta prioridad;
- Si el tipo de expresión a la derecha de la operación del signo de asignación (=) no coincide con el tipo de variable a la izquierda de la operación del signo de asignación, el valor de la expresión de la derecha se lanza como el tipo de variable a la izquierda de la operación de asignación, es lo que se llama 'target-type cast' (emisión del tipo objetivo)
- No está permitido emitir valores de tipo **string** a cualquier otro tipo de objetivo.

Bueno, vamos a volver al Problema 4. No puede haber dos soluciones para ello.

Solución 4,1. Cálculo del resultado de tipo *int:*

```
double A = 2.0;                 // El número de lápices de Juan
 int    Y = 3;                  // El número de viajes Pedro
 int    F = A + Y;               // Número total
```

En primer lugar, necesitamos saber el valor de la expresión siempre que sus operandos sean de distinto tipo. En la expresión:

```
A + Y,
```

Los operandos que se utilizan son de dos tipos de datos: el operando A es de tipo ***double***, y el operando Y es de tipo ***int***.

De conformidad con la regla implícita de typecasting, el valor de esta expresión será un número de tipo ***double***. Tenga en cuenta que: estamos hablando sólo sobre el tipo de la expresión A + Y, pero no sobre el tipo de variable F que está a la izquierda del signo de asignación (=). El valor de esta expresión es el número real de 5,0. Para emitir el tipo de expresión A + Y, se aplicó la primera parte de la regla implícita typecasting.

Después del cálculo de la expresión A + Y se ejecuta la operación de asignación (=). En este caso, los tipos no coinciden, el tipo de la expresión A + Y es el ***double***, mientras que el tipo de variable F es ***int***. Durante el periodo de ejecución de la operación de asignación el tipo de expresión A + Y se emite como tipo ***int*** (según la regla del cálculo) y se convierte en número entero 5, a continuación, este resultado se convierte en el valor del entero variable F. Los cálculos se han realizado de acuerdo con la segunda parte de la regla implícita typecasting - «emisión de tipo objetivo». El resultado final de los cálculos y manipulaciones es la siguiente: El valor del entero de la variable F es un entero de valor 5.

Solución 4,2. Una situación similar se produce, si tratamos de tener un resultado como un valor de tipo ***double***:

```
double A = 2.0;                 // El número de lápices de Juan
int    Y = 3;                   // El número de viajes Pedro
double   F = A + Y;             // Número total
```

Esta situación difiere de la anterior en que el objetivo de la variable tipo F (a la izquierda de la operación del signo de asiganción), en este caso es de tipo ***double*** , coincide con el tipo ***(double)*** de la expresión A + Y, por lo que no tenemos ningúna emisión de tipo objetivo aquí. El resultado de los cálculos (el valor de tipo ***double*** de variable F es el número real de 5,0.

Vamos a encontrar ahora una solución para el Problema 5. No hay preguntas hasta llegar a la inicialización de variables:

```
string W1 = "Ártico";           // String 1
double A  = 2;                  // El número de lápices de Juan
```

Solución 5,1. Una posible solución para este problema:

```
string W1  = "Ártico";          // String 1
double A   = 2;                 // Número de lápices Juan
string Sum = W1 + A;            // Transformación implicita a la derecha
```

Aquí, en la parte derecha de la expresión existen dos tipos de variables: uno de tipo **string** y otro de tipo ***double***. De acuerdo con la regla implícita de typecasting, el valor de la variable será emitida en primer lugar como tipo string ya que este tipo tiene una prioridad más alta que el tipo double. A continuación este tipo de valores se suman (concatenación). El tipo del valor resultante a la derecha de la operación de asignación sera de tipo **string**. En la siguiente etapa, este valor se le asignará a la variable ***string*** Suma Como resultado, el valor de la variable Suma resultará con el siguiente texto:

```
Arcaico 2.00000000
```

Solución 5,2. Esta solución es errónea:

```
string W1  = "Ártico";        // String 1
double A   = 2;               // Número de lápices Juan
double Sum = W1 + A;          // Transformación implicita a la derecha
```

En este caso, se pretende romper la prohibición de la emision de tipos objetivos de los los valores de tipo ***string***. El tipo del valor de la expresión W1 + A, de tipo **string**, como en la anterior solución. Cuando la operación de asignación se ejecuta, debe realizarse la emisión de tipo objetivo. Sin embargo, de acuerdo a la norma, la emisión del tipo objetivo ***string*** desde tipos de menor prioridad no está permitida. Se trata de un error que detecta el MetaEditor durante la creación del programa (en la compilación).

En general, las reglas que figuran en esta sección son claras y sencillas: Para cualquier cálculo de valores, usted debe emitir todos los diferentes tipos desde un tipo de prioridad más alta. El typecasting de prioridad mas baja solo está permitido para valores numéricos pero no con cadena de caracteres porque estas no pueden transformarse en números.

Características de los cálculos Integer

Se sabe que los números enteros son valores sin parte fraccional o decimal. Estos valores se pueden suman o restar juntos y el resultado que se obtiene es muy intuitivo. Por ejemplo, si:

```
int X = 2;                    // Primera variable int
int Y = 3;                    // Segunda variable int
```

y:

```
int Z = X + Y;                // Operación de adición
```

no hay ningún problema para calcular el valor de la variable Z: 2 + 3 = 5

Del mismo modo, si se ejecuta una operación de multiplicación:

```
int Z = X * Y;                // Operación de Multiplicación,
```

el resultado es muy previsible: 2 * 3 = 6

Pero, ¿qué resultado obtenemos si nuestro programa tiene que ejecutar una operación de división?

```
int Z = X / Y;                // División de las variables
```

Es fácil escribir 2 / 3. Sin embargo, no es un entero. Así que, ¿cuál será el valor de la expresión X/Y y Z variable?



La regla de cálculo de valores de tipo entero es que siempre se descarta la parte decimal.

En el ejemplo anterior, la expresión a la derecha del signo de igual sólo contiene números enteros, es decir, en este caso no se lleva a cabo typecasting. Y esto significa que el tipo de expresión X / Y es *int*. Así que el resultado de hallar el valor de la expresión X/Y (= 2 / 3) es 0 (cero). Este valor (cero) se le asignará a la variable Z al final.

De la misma manera, otros valores de las X e Y produciran otros resultados. Por ejemplo, si:

```
int X = 7;                    // El valor de una variable int
int Y = 3;                    // El valor de una variable int
int Z = X / Y;                // División de las variables
```

En este caso el valor de 7/3 para la expresión X/Y y Z variable es igual a 2 (dos).

Orden de cálculo de las operaciones

La regla para el cálculo de las operciones es la siguiente:



El valor de una expresión se calcula de acuerdo a las prioridades de las operaciones aritméticas y de izquierda a derecha, cada resultado intermedio se calculará de acuerdo a las normas typecasting.

Vamos a considerar el fin para el cálculo de una expresión en el siguiente ejemplo:

```
Y = 2.0*(3*X/Z - N) + D;                    // Expresión ejemplo
```

La expresión a la derecha del signo igual se compone de dos sumandos: 2,0 * (3 * X / Z - N) y D. El sumando 2,0 * (3 * X / Z - N) consta de dos factores, a saber: 2 y (3 * X / Z - N). La expresión entre paréntesis, 3 * X / Z - N, a su vez, consta de dos summands, sumando los 3 * X / Z que consta de tres factores, a saber: 3, X y Z.

Con el fin de calcular la expresión a la derecha de la igualdad de signo, entonces, en primer lugar, calcular el valor de expresión 3 * X / Z. Esta expresión contiene dos operaciones (multiplicación y división) del mismo rango, por lo que el cálculo de esta expresión de izquierda a derecha. En primer lugar, vamos a calcular el valor de expresión 3 * X, el tipo de esta expresión es el mismo que el tipo de la variable X. A continuación, vamos a calcular el valor de expresión 3 * X / Z, su tipo se calculará de acuerdo con la regla typecasting. Después de eso, el programa va a calcular el valor y el tipo de la expresión 3 * X / Z - N, entonces, de la expresión 2,0 * (3 * X / Z - N), y el último de toda la expresión * 2,0 (3 * X / Z - N) + D.

Como es fácil de ver, el orden de las operaciones en un programa es similar a la de matemáticas. Sin embargo, el anterior se diferencia en el cálculo de los tipos de valores en los resultados intermedios, que ejerce una influencia significativa en el resultado final de los cálculos. En particular, (a diferencia de las normas aceptadas en las matemáticas), el orden de los operandos en una expresión es muy importante. Para demostrar esto, vamos a examinar un pequeño ejemplo.



Problema 6.Calcular los valores de las expresiones A/B*C y A*C/B para los enteros A,B y C.

El resultado del cálculo es intuitivamente esperado que sea el mismo, en ambos casos. Sin embargo, esta afirmación es cierta sólo para los números reales. Si queremos calcular los valores de las expresiones compuestas de los operandos de tipo ***int,*** debemos siempre considerar el resultado intermedio. En tal caso, la secuencia de operandos es de fundamental importancia:

```
int A = 3;                  // Valor de tipo int
int B = 5;                  // Valor de tipo int
int C = 6;                  // Valor de tipo int
int Res_1 = A/B*C;          // Result 0 (cero)
int Res_2 = A*C/B;          // Resultado 3 (tres)
```

Vamos a seguir el proceso de cálculo de la expresión A / B * C:

1. En primer lugar (de izquierda a derecha) se calculará el valor de la expresión A / B. De acuerdo con las reglas anteriores, el valor de la expresión (3/5) es integer 0 (cero).
2. Cálculo de la expresión 0*C (cero multiplicado por C). El resultado es integer 0 (cero).
3. Otros resultados (el valor de la variable Res_1) es integer **0 (cero).**

Vamos a seguir ahora la evolución del cálculo de la expresión A * C / B.

1. Cálculo de A * C. El valor de esta expresión es entero 18 (3 * 6 = 18).

2. Cálculo de la expresión 18 / B. La respuesta es evidente: después de la parte decimal se ha descartado, (18 / 5) es entero 3 (tres).

3. Otros resultados (el valor de la variable Res_2) es entero **3 (tres).**

En el ejemplo anterior, consideramos que sólo un pequeño fragmento de código, en la que se calculan los valores de las variables de tipo ***int***. Si queremos sutituir estas variables con constantes con los mismos valores, el resultado final sera el mismo. Al calcular las expresiones que contengan enteros, se debe prestar mayor atención a los contenidos de sus líneas de programa. De lo contrario, puede ocurrir un error en su código, que sería muy difícil de encontrar y arreglar más tarde (sobre todo en programas grandes). Tales problemas no se producen en los cálculos utilizando números reales. Sin embargo, si utiliza operandos de diferentes tipos en una expresión compleja, el resultado final puede depender completamente de un fragmento formado fortuitamente que contiene la división de números enteros.

En la sección de operadores, son considerados el término y las propiedades generales de los operadores y también las propiedades especiales de cada operador se describen en el capítulo llamado operadores.

Operadores

El termino operador

Uno de los principales conceptos de cualquier lenguaje de programación es el término de «operador». La codificación parece ser imposible para la persona que no ha aprendido por completo este término. Cuanto antes y mejor, aprende un programador lo que son los operadores, y cómo se aplican en un programa, antes se inicia éste en la escritura de sus propios programas.

El operador es parte de un programa, una frase de un lenguaje algorítmico que prescribe un determinado método de conversión de datos.

Cualquier programa contiene operadores. La analogía más cercana a «operador» es una frase. Así como una sentencia compone el texto normal de una novela, así los operadores componen un programa.

Propiedades de los operadores

Hay dos tipos de propiedades de los operadores: una propiedad común y las propiedades específicas de los distintos operadores.

La Propiedad común de los operadores

Todos los operadores tienen una propiedad común: todos ellos se ejecutan.

Podemos decir que el operador es una instrucción que contiene la guía de operaciones, la descripción de una orden. Para un ordenador ejecute un programa significa que (consecutivamente, pasando de un operador a otro) se cumplan las órdenes, (las recetas, instrucciones) que figura en los operadores.

Un Operador, como tal, es sólo un registro, una cierta secuencia de caracteres. Un Operador no tiene palanca, cables o células físicas de memoria. Es por ello que, cuando un ordenador está ejecutando un programa, no pasa nada en los operadores como tales, ellos siguen permaneciendo en el programa como los compuso el programador. Sin embargo, el equipo es el que tiene todas esas células de memoria y los vínculos entre ellas y todas las experiencias dentro de las transformaciones. Pero si su PC ha ejecutado algunas transformaciones de datos con arreglo a las instrucciones contenidas en un operador, usted dice: el operador se ha ejecutado.

Propiedades específicas de los operadores

Existen varios tipos de operadores. Cada tipo de operadores tiene sus propiedades específicas. Por ejemplo, la propiedad de un operador de asignación es su capacidad para asignar un valor a una variable dada, la propiedad de un operador de bucle es sus múltiples ejecuciones, etc. Las propiedades específicas de los operadores se consideran en todos los detalles en las secciones correspondientes del capítulo Los operadores de este libro. Vamos a decir aquí que cada tipo de operador tiene sus propias propiedades, que son típicas sólo por su tipo y no se repiten en ningún otro tipo.

Tipos de Operadores

Hay dos tipos de operadores: los operadores simples y los compuestos.

Operadores simples

Los operadores simples de MQL4 terminar con el carácter ";" (punto y coma). El uso de este separador, es para que el PC pueda detectar cuando un operador termina y otro comienza. El carácter ";" (punto y coma) es tan necesario en un programa como carácter "." (punto) lo es en un texto normal para separar las frases. Un operador puede tener varias líneas. Se pueden colocar varios operadores en una línea.



Cada operdor simple finaliza con el carácter ";" (punto y coma).

Ejemplos de operadores simples:

```
Day_Next= TimeDayOfWeek(Mas_Big[n][0]+60);   // Operador simple

Go_My_Function_ind();                        // Operador simple

a=3;  b=a*x+n;  i++;                         // Varios operadores colocados en linea

   Print("   Day= ",TimeDay(Mas_Big[s][0]),    // Un operador...        "                    "
     "   Hour=",TimeHour(Mas_Big[s][0]),   // colocado..
     "   Minute=",TimeMinute(Mas_Big[s][0]),  // en varias..
     "   Mas_Big[s][0]= ",Mas_Big[s][0],       // lineas
     "   Mas_Big[s][1]= ",Mas_Big[s][1]);
```

Operadores compuestos

Un operador compuesto consta de varios operadores simples separados por el carácter ";" y se vinculan entre llaves. Con el fin de poder utilizar varios operadores donde se espera que haya solo uno, los programadores utilizar un operador compuesto (también lo llaman "bloque" o "bloque de código"). El conjunto de operadores simples estan ubicados en un recinto separado por llaves. La presencia de una llave de cierre marca el final de un operador compuesto.



Un ejemplo de utilización de un operador compuesto es un operador condicional. Comienza con el operador condicional if (expresión), seguido por un bloque compuesto de operadores simples llamado cuerpo. Este cuerpo contiene una lista de operadores ejecutables.

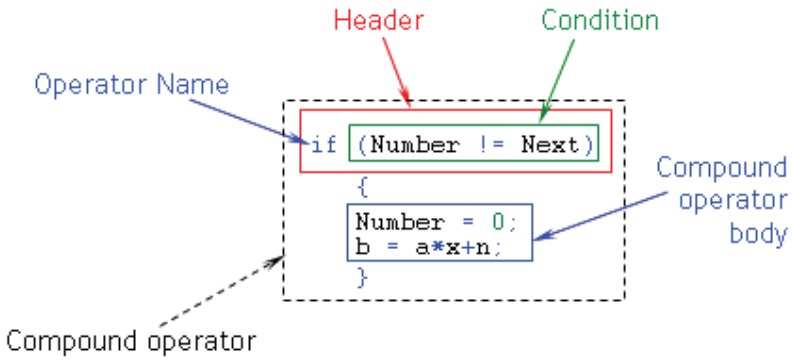


Fig. 17. Operador compuesto.

El cuerpo de un operador compuesto se situa entre llaves. Todos los operadores compuestos finalizan con una llave de cierre.

Ejemplos de operadores compuestos:

```
                                  // Ejemplo del operador switch
switch(ii)                          // Operador switch(expresion)
  {                                 // Apertura de llave
  case 1: Buf_1[Pok-f+i]= Prognoz;  break; // Nested operators (cuerpo del operador)
  case 2: Buf_2[Pok-f+i]= Prognoz;  break; // Nested operators (cuerpo del operador)
  case 3: Buf_3[Pok-f+i]= Prognoz;  break; // Nested operators (cuerpo del operador)
  }                                 // Cierre de llave que..
                                    // .. muestra donde acaba el operador compuesto
//-------------------------------------------------------------------------------------
                                  // Ejemplo de uso en un bucle o lazo.
for (tt=1; tt<=Kol_Point[7]; tt++)        // Operador for(expresiones)
  {                                 // Apertura de llave
  Numb = Numb + Y_raz[tt]*X_raz[ii][tt];   // Nested operators (cuerpo del operador)
  }                                 // Cierre de llave que..
                                    // .. muestra donde acaba el operador compuesto
//-------------------------------------------------------------------------------------
                                  // Ejemplo de operador condicional if
if (TimeDay(Mas_Big[f][0])!= 6)           // if (expresion)
  {                                 // Apertura de llave
  Sred =(Nabor_Koef[ii][vv][2]+ NBh)*Point// Nested operators (cuerpo del operador)
  Ind = Nabor_Koef[ii][vv][0] + f;        // Nested operators (cuerpo del operador)
  Print(" Ind= ",Ind);                  // Nested operators (cuerpo del operador)
  }                                 // Cierre de llave que..
                                    // .. muestra donde acaba el operador compuesto
```

El cuerpo de un operador compuesto esta siempre encerrado entre llaves y puede estar formado de cero, uno, o varios operadores.

Ejemplos de operadores simples:

```
//------------------------------------------------------------------------------
                              // Ejemplo del operador for
  for (n=1; n<=Numb; n++)         // for(expresiones)
    Mas[n]= Const_1+ n*Pi;       // Nested operators (cuerpo del operador)
//------------------------------------------------------------------------------
                              // Ejemplo del operador condicional if
  if (Table > Chair)              // if (expresion)
    Norma = true;                 // primer operador (suboperador 1)
  else                           // Else-condición
    Norma = false;                // segundo operador (suboperador 2)
//------------------------------------------------------------------------------
```



Varios operadores simples se pueden combinar en un operador compuesto sin tener estricta necesidad.

Esta es una rara enfermedad, pero una construcción absolutamente admisible. En este caso, los operadores se encierran ente llaves, y se denominan "bloque de operadores". Este uso es completamente aceptable. El programador es el que decide si incluir o no a los operadores entre llaves, simplemente en aras de una representación conveniente del código.

Ejemplo de bloque de operadores:

```
{                                 // Apertura de llave
  Day_Next= TimeDayOfWeek(Mas_Big[n][0]+60);   // Operador simple
  b=a*x+n;                              // Operador simple
  }                                  // Cierre de llave..
```

Requisitos de los Operadores

Los operadores deben estar escritos en el texto de un programa de acuerdo a las normas de formato (la forma en que deben estar representados en un código). Ningún operador puede estar construido más allá de estas reglas. Si el programa contiene un operador compuesto con un formato fuera de las reglas, el MetaEditor producirá un mensaje de error en la compilación. Esto significa que el programa que contiene el operador erróneo no puede utilizarse.

Se debe entender la frase "formato del operador" como un conjunto de normas de formato, típico para todos los operadores del mismo tipo. Cada tipo de operador tiene su propio formato. Los formatos del Operador son considerados en todos sus detalles en las secciones de este libro correspondientes al capítulo Operadores.

Orden de ejecución de los operadores

Una característica muy importante de cualquier programa es el orden de ejecución de los operadores dentro de él. Los operadores no pueden ser ejecutados sin razón o por excepción. En MQL4, el orden de ejecución de los operadores es el siguiente:



Los operadores se ejecutan en el orden, en el que se aparecen en el programa. La dirección de los operadores de ejecución va de izquierda a derecha y de arriba a abajo.

Esto significa que tanto los los operadores simples como los compuestos y se ejecutan uno a uno (como en las líneas de los poemas: en primer lugar se lee la línea superior, después la siguiente inferior, después la siguiente y así sucesivamente). Si hay varios operadores en una línea, deben ser ejecutadas consecutivamente, uno después de otro, de izquierda a derecha, a continuación, los agentes se ejecutan en la línea inferior más cercana en el mismo orden.

Los operadores que integran un operador compuesto se ejecutan de la misma manera: todo operador del bloque de código comienza a ser ejecutados sólo después de que lo ha hecho el anterior.

Redacción y Ejecución de Operadores: Ejemplos

El texto de un programa que contiene los operadores no es diferente en aspecto a un texto normal o una notación matemática. Sin embargo, esta similitud es sólo formal. Un texto normal permite que las anotaciones puedan ser colocadas en cualquier secuencia, mientras que en un programa se debe mantener un orden bien definido.

A modo de ejemplo, veamos como trabaja un operador de asignación. Vamos a resolver un simple sistema de ecuaciones lineales y comparar la representación de algunos cálculos matemáticos en un texto normal con los operadores de un código de programa..



Problema 7. Tenemos un sistema de ecuaciones:
Y = 5
Y - X = 2
Hayar el valor numérico de la variable.

Solución 1. En un texto normal en una hoja de papel:

1. 5 - X = 2
2. X = 5 - 2
3. X = 3

Solución 2. Un texto en un programa:

```
Y = 5;              // Linea 1
X = Y - 2;          // Linea 2
```

Tanto en la primera y la segunda en las soluciones, las anotaciones (líneas) han completado un contenido. Sin embargo, las líneas en Solución 1 no se pueden utilizar en un programa tal como son, porque su apariencia no cumple con el formato del operador de asignación.

Las anotaciones que figuran en la Solución 1 representan algunas funciones matemáticas en papel. Sólo pueden utilizarse para informar a los programadores sobre las relaciones entre las variables. Los operadores en un programa se asignan para otros fines, que informará a la máquina de las operaciones y en qué orden se debe ejecutar. Todos los operadores, sin excepción alguna, representan instrucciones precisas que no permiten ambigüedades.

Los operadores de la Solución 2 son los operadores de asignación. Cualquier operador de asignación da literalmente a la máquina la siguiente orden:



Calcula el valor de la expresión que se encuentra a la derecha de la igualdad y asigna el valor obtenido a la variable situada a la izquierda del signo de igualdad. Es decir, en el lado izquierdo de la igualdad solo puede haber una variable y en el lado derecho una expresión con cualquier grado de complejidad

Por esta razón, solo una variable puede estar situada a la izquierda del signo de igualdad de un operador de asignación. Por ejemplo, un registro de 5 - X = 2 utilizada en la primera solución contendría un error, porque el conjunto de caracteres 5 - X no es una variable.

Vamos a seguir al ordenador durante la ejecución de los operadores de la segunda solución.

1. Paso por el operador (línea 1).

```
Y =5; // Línea 1
```

2. Referencia a la parte derecha del operador (la parte derecha se encuentra entre el signo de igualdad y el punto y coma).
3. El ordenador ha detectado que la parte derecha del operador contiene un valor numérico.
4. Registro en la memoria fisica del ordenador del valor numérico (5) de la variable Y.
5. Paso al siguiente operador (línea 2).

```
X = Y-2; // Línea 2
```

6. Referencia a la parte derecha del operador.
7. El ordenador ha detectado que la parte derecha del operador contiene una expresión.
8. Cálculo del valor numérico de la parte derecha del operador (5 - 2).
9. Registro en la memoria física del valor numérico (3) de la variable X.

La realización de los pasos 1-4 en la computadora es la ejecución del primer operador (línea 1). La realización de los pasos 5-9 en la computadora es la ejecución del segundo operador (línea 2).

Con el fin de código de un programa viable, el programador debe también darse cuenta de qué, y en qué orden se ejecutará este programa. En particular, no todos los cálculos matemáticos se pondrán en un programa, a veces es necesario antes preparar a los operadores.

Por ejemplo, muchos cálculos intermedios se realizan cuando se ha hayado la solución de otros problemas matemáticos. Estos pasos intermedios pueden ayudar a un matematico a encontrar una solución adecuada, pero resultan inútiles desde el punto de vista de la programación. Sólo soluciones significativas pueden ser incluidos en un programa, por ejemplo: valores originales de las variables o las fórmulas para calcular los valores de otras variables. En el ejemplo anterior, el primer operador tiene información sobre el valor numérico de la variable Y, y el segundo operador establece la fórmula para calcular el valor de la variable X que nos interesa.

Cualquier programa viable contiene expresiones de significado familiar, pero también se puede encontrar expresiones que usted será capaz de comprender sólo si se comentan tanto como sea posible, en el programa. Por ejemplo, el registro de abajo…

```
X = X + 1; // Ejemplo de un contador
```

parece ser erróneo desde el punto de vista logico matemático y el propio sentido común. Sin embargo, es bastante aceptable si se considera como un operador (por cierto, este operador se utiliza ampliamente en la codificación).

Con este operador, hemos calculado un nuevo valor de la variable X: cuando se ejecuta el operador de asignación (es decir, el cálculo del valor de la parte derecha del operador), el ordenador toma el valor de la memoria física que contiene el valor numérico de la variable X (que en el ejemplo resulta ser igual a 3 en el momento de referirse a ella), calcula la expresión en la parte derecha del operador de asignación (3 + 1), y escribe el valor obtenido (4) en la memoria celular (física) de la variable X. La máquina almacenará este valor de la variable X hasta que la variable X se produce en la parte izquierda del signo de igualdad en otro operador de asignación. En este caso, el nuevo valor de esta variable se calculará y se almacenan hasta el próximo posible cambio.

Funciones

Término de una función

El más importante avance tecnológico en ingeniería informática es la posibilidad de creación y almacenamiento de fragmentos de código separado que describan normas de procesamiento de datos para resolver un problema o una pequeña tarea. Esta posibilidad tambien existe en MQL4.

Función es el nombre que recibe una parte específica de un programa que describe un método de conversión de datos.

Hablando de funciones, vamos a considerar dos aspectos: descripción o definición de la función y funcion de llamada

Función descripción o definición es la parte del programa destinada a su ejecución.

Función de llamada (función de referencia) es un registro, es el acto que conduce a la ejecución de la función.

En nuestra vida cotidiana, podemos encontrar muchas análogias de la función. Tomemos, por ejemplo, el sistema de frenado de un coche. El mecanismo de accionamiento que lleva a cabo el frenado en un vehiculo. La idea implentada por el ingeniero es análoga a la definición/descripción de función, mientras que el pedal de freno es el análogo de la llamada a la función. El conductor presiona el pedal, y los mecanismos de accionamiento realizar ciertas acciones y detenienen el coche.

Del mismo modo, si la llamada a una función se produce en un programa, entonces la función del mismo nombre será llamada y ejecutada, es decir, se llevarán a cabo una cierta secuencia de cálculos u otras acciones (por ejemplo, se muestra un mensaje o una orden de apertura, etc) El sentido general de una función es la adopción de una lógica que se completa fuera del texto base del programa, mientras que sólo se mantiene dentro del texto base del programa la parte del código que se ocupa de la llamada de esta. Este programa de construcción tiene algunas ventajas incontestables:

- Primera, el texto del programa está integrado de tal manera que se lee mucho más fácil.
- Segunda, se puede ver con facilidad y, si es necesario, modificar el texto de una función sin realizar ningún cambio en el código básico ó programa principal.
- Y tercera, una función puede estar compuesta como un solo archivo y usarse en otros programas, el cual liberará al programador de la necesidad de insertar el mismo fragmento de código en cada programa de nueva creación.

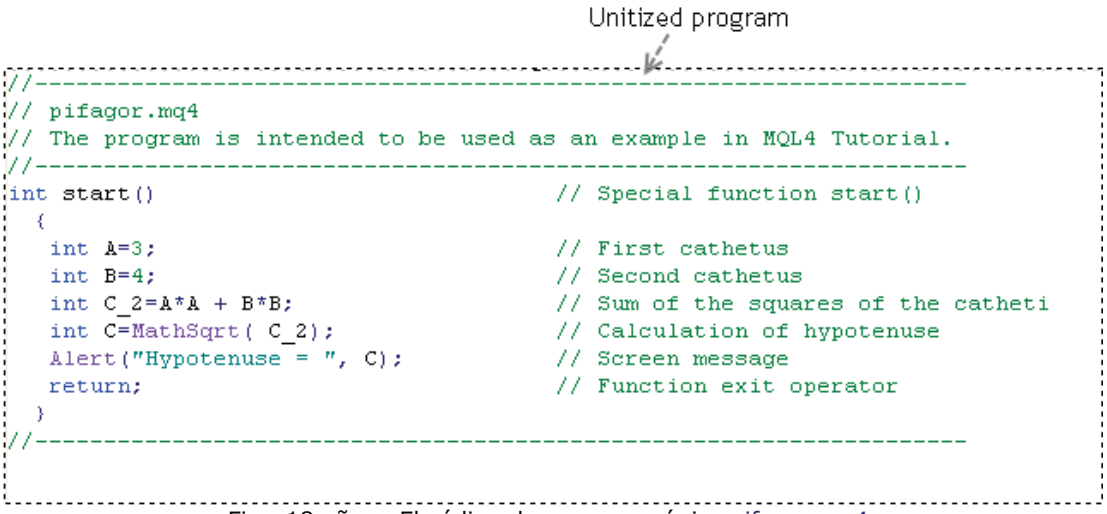
Podemos decir que la mayor parte del código de los programas que usan MQL4 está escrito en forma de funciones. Este enfoque se extendió y actualmente es un estándar.

Composición de una función

Por lo tanto, una función está compuesta de **la descripcion** y **la llamada**. Vamos a considerar un ejemplo. Supongamos que tenemos un pequeño programa (Fig. 18) que considera la longitud de la hipotenusa utilizando los otros dos lados del triángulo rectangulo y el teorema de Pitágoras.



En este programa, todos los cálculos se encuentran juntos, los operadores son ejecutados uno por uno en el orden en el que se producen en el programa (de arriba hacia abajo).



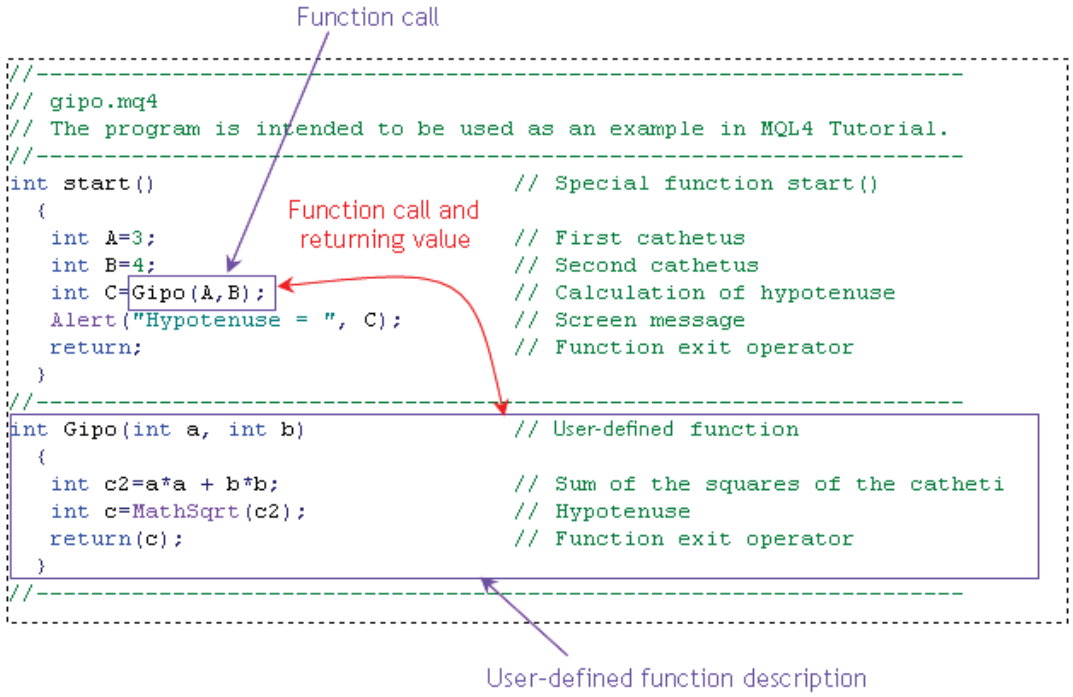

```
//+------------------------------------------------------------------+
// pifagor.mq4
// The program is intended to be used as an example in MQL4 Tutorial.
//+------------------------------------------------------------------+
int start()                                 // Special function start()
  {
   int A=3;                                 // First cathetus
   int B=4;                                 // Second cathetus
   int C_2=A*A + B*B;                       // Sum of the squares of the catheti
   int C=MathSqrt( C_2);                    // Calculation of hypotenuse
   Alert("Hypotenuse = ", C);               // Screen message
   return;                                  // Function exit operator
  }
//+------------------------------------------------------------------+
```

Fig. 18 años. El código de programa único pifagor.mq4.

Problema 8. Redactar una parte del código del programa como una función.

Sería razonable hacer una función utilizando las dos líneas que encuentran el valor buscado. El mismo programa pero usando una función se muestra en la Fig. 19.

En este programa, una parte de los cálculos se integra como una función. El código básico contiene una llamada a la función definida por el usuario. La descripción de la función definida por el usuario se encuentra fuera (después de) del código básico.




```
//+------------------------------------------------------------------+
// gipo.mq4
// The program is intended to be used as an example in MQL4 Tutorial.
//+------------------------------------------------------------------+
int start()                                 // Special function start()
  {
   int A=3;                                 // First cathetus
   int B=4;                                 // Second cathetus
   int C=Gipo(A,B);                         // Calculation of hypotenuse
   Alert("Hypotenuse = ", C);               // Screen message
   return;                                  // Function exit operator
  }
//+------------------------------------------------------------------+
int Gipo(int a, int b)                      // User-defined function
  {
   int c2=a*a + b*b;                        // Sum of the squares of the catheti
   int c=MathSqrt(c2);                      // Hypotenuse
   return(c);                               // Function exit operator
  }
//+------------------------------------------------------------------+
```

Fig. 19. El código de un programa que contiene la descripción y la llamada a función definida por el usuario gipo.mq4.

Ambas versiones del programa dará el mismo resultado. Sin embargo, el código se compone de un único módulo en la primera versión (Fig. 18), mientras que en la segunda versión (Fig. 19) una parte de los cálculos se ejecuta en la llamada a una función desde el texto de base. Una vez terminados los cálculos separados en la función, continuarán los cálculos en el código del programa principal.

Para conseguir la ejecución de la función, tenemos que llamarla. Esta es la razón por que la función esta representada en dos partes: el propio código que compone la función (función descripción) y la llamada a la función para poner en marcha la función (refiriéndose a ella). Si no se llama a la función, esta no será ejecutada. Al mismo tiempo, si se llama a una función inexistente, esto se traducirá en nada (El MetaEditior dará un mensaje de error si se intenta compilar un programa así en MQL4).

Descripción de la Función

La descripción de una función consta de dos partes básicas: cabecera de la función y cuerpo de la función.

La **cabecera** de una función está formada por: el tipo del valor de return, el nombre de función y la lista de parámetros formales. La lista de parámetros formales están encerrados entre paréntesis y se colocan después del nombre de la función. El tipo del valor de return puede ser uno de los tipos que ya conocemos: ***int, double, bool, color, datetime,* o *string***. Si la función no devuelve ningún valor, su tipo puede ser denominado ***void*** ("sin contenido, vacío") como se denominaríacualquier otro tipo.

El **cuerpo** de una función se encierra entre llaves. El cuerpo de la función puede contener operadores simples y/o complejos, así como la llamada a otras funciones. El valor devuelto por la función se da en el paréntesis de operador return (). El tipo del valor devuelto es el utilizando el operador return () y debe coincidir con el tipo de la función especificada en la cabecera de la función. La descripción de la función se termina con el cierre de la llave.

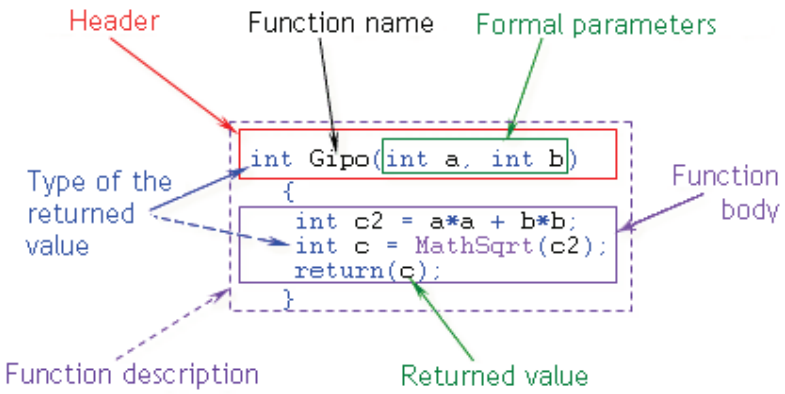


Fig. 20. Descripción de la función.

La Descripción de la función debe estar ubicada separada del programa, al margen de cualesquiera otras funciones (es decir, no dentro de otra función, si no fuera de ella).

Llamada a la Función

La **llamada a la Función** se representa con el nombre de la función y la lista de parámetros transferidos. La lista de parámetros se transfieren entre paréntesis. La llamada a la función puede ser representada como un operador independiente o como parte de un operador.

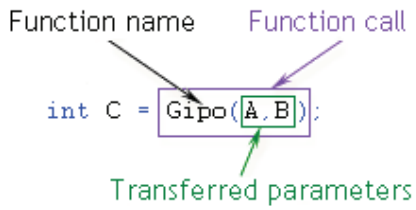


Fig. 21. Llamada a la Función (referencia a una función).



Cualquier llamada a la función es siempre dentro de otra función (es decir, no fuera de todas las demás funciones, pero dentro de una de ellas).

Tipos de funciones

Hay tres tipos de funciones: funciones especiales, funciones estándar (built-in o predefinidas), y funciones definidas por el usuario.

Funciones especiales

En MQL4, hay 3 funciones especiales en total. Ellas tienen nombres predefinidos: **init (), start (),** y **deinit ()** que no pueden utilizarse como nombres de cualquiera otras funciones. El examen detallado de las funciones especiales se da en el capítulo Funciones especiales. Sólo decir aquí que el código básico de un programa se encuentra dentro de estas funciones (Fig. 18, 19).

La característica especial de las funciones especiales es el hecho de que son llamadas para su ejecución desde el Terminal de Usuario. Aunque las funciones especiales tienen todas las propiedades de las funciones en general, no se les suele llamar desde el programa si éste está codificado correctamente.

Funciones estándar

MQL4 tiene una serie de útiles funciones en las cuales, cuando se escribe la codificación del programa no es necesario hacer su descripción. Por ejemplo, el cálculo de raíces cuadradas, la impresión de mensajes en el sistema o en la pantalla. Todas estas y muchas otras funciones estándar se realizan de acuerdo con un algoritmo predefinido. El usuario no necesita saber el contenido de estas funciones. Él o ella sólo puede estar seguro de que todas las funciones estándar son desarrolladas debidamente por profesionales y de acuerdo a la mejor algoritmo posible.

La característica singular de las funciones estándar es que no están descritas en el texto del programa. Las funciones estándar son llamadas en el programa de la misma manera a como lo hace cualquier otra función (es una práctica común).

En nuestro ejemplo (Fig. 18, 19), se utilizan dos funciones estándar: MathSqrt () y Alerta (). La primera está destinada al cálculo de raíces cuadradas, mientras que la segunda está diseñada para mostrar un determinado mensaje de texto, puesto entre paréntesis, en la pantalla. Las propiedades de las funciones estándar son consideradas con más detalles en la sección Funciones estándar. Aquí vamos a observar que estos registros representan llamadas a funciones estándar, mientras que descripciones de estas funciones no se las puede encontrar en el programa. Estas funciones, también pueden ser denominadas funciones built-in, o funciones predefinidas. Usted puede usar cualquiera de estos términos.

Funciones definidas por el usuario

En algunos casos, los programadores crean y utilizan sus propias funciones y hacen la llamada estas funciones (definidas por el usuario). Las Funciones definidas por el usuario se utilizan en los programas con la descripción de la función y las llamadas a la funcion.

Cuadro 1. La descripción de la Función y la llamada a la función son utilizadas en los programas dependiendo de los diferentes tipos de funciones.

Tipo de función	Descripción de la Función	Llamada a la Función
Especial	Implementable	No procede (*)
Estandar	No se implementa	Aplicable
Definida por el usuario	Implementable	Aplicable

(*) A pesar de que las funciones especiales puede ser técnicamente llamadas desde un programa, no se recomienda hacerlo.

Propiedades de las Funciones

Ejecución de la Función

La principal propiedad de todas las funciones es que la llamada a la función hacen que estás se ejecuten. Las funciones se ejecutan de acuerdo a sus códigos.

El paso de parámetros y el valor de return

Una función se comporta como una calculadora estándar, en cuanto a que la información se recibe y se devuelve transformada. Se puede escribir (usando el teclado) una cierta expresión que consta de varios valores van entrando uno a uno, y se obtendrá un valor como respuesta. Una función puede recibir y procesar uno o varios parámetros del programa que le ha llamado para su ejecución, y la función terminará su operación de retornando (transmitiendo, dando, entregando) un parámetro como respuesta a este programa.

El **paso de los parámetros** se especifican encerrandos entre paréntesis después del nombre de la función que se llama. El paso de parámetros se hace usualmente separandolos mediante comas. El número de parámetros transferidos a la función no debe superar 64. La función también puede omitir el uso de paso de parámetros. En este caso se especifica una lista vacía de parámetros, es decir, simplemente hay que poner un paréntesis de apertura y uno de cierre directamente después del nombre de función.

El número, tipos y orden de los parámetros transferidos en la llamada a la función debe coincidir con los parámetros de formación que se especifica en la descripción de la función (la llamada de una función que tiene parámetros por defecto es una excepción - véase la Llamada a la función y Descripción de funciones y el operador " return "). Si no coinciden, el MetaEditor le dará un mensaje de error. Constantes, variables, expresiones y arrays pueden ser utilizadas como parámetros de paso.

El **valor de return** se especifica en los paréntesis del operador return () (ver Descripción de funciones y el operador "return"). El tipo del valor devuelto utilizando en el operador return () debe coincidir con el tipo de la función dada en la cabecera de la función. También es posible que una función no devuelva ningún valor. En este caso, no se especifica nada en el paréntesis del operador return ().

En nuestro ejemplo, el paso de parámetros son variables **A** y **B** (Fig. 21), mientras que el valor de return es la variable **c** (Fig. 20). El requisito de que concuerden los tipos de paso y los parámetros formales pueden verse en la Fig. 22.



El número, tipo y orden de los parámetros transferidos en la llamada a la función debe coincidir con los parámetros que se especifican en la descripción de la función. Solamente se pueden utilizar variables en los parámetros formales de la cabecera de la descripción de la función.

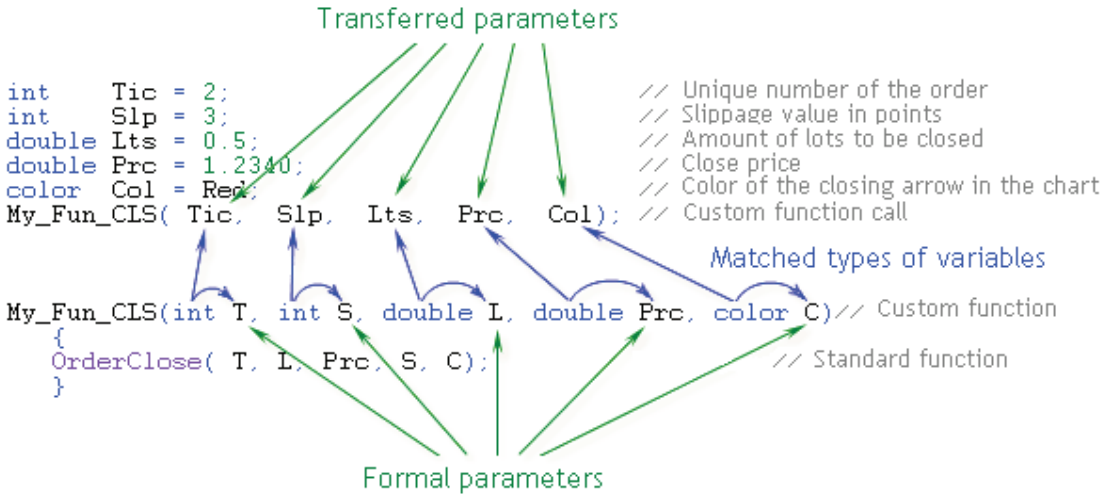


Fig. 22. Match en el número, tipo, parámetros transferidos y parámetros formales. En este caso solamente se utilizan variables como parámetros transferidos.



Como parámetros transferidos, solo se pueden utilizar variables, constantes y expresiones.

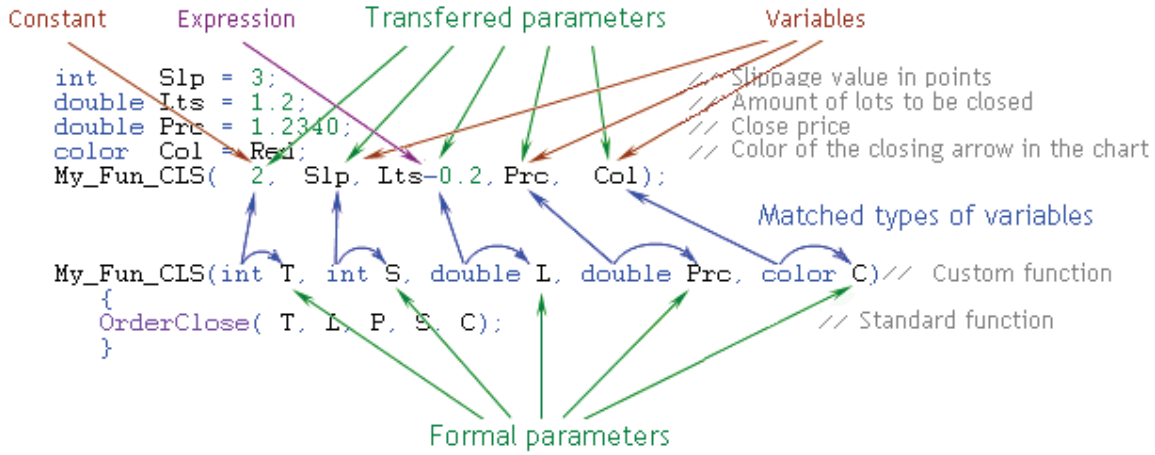


Fig. 23. Match en el número, tipo, parámetros transferidos y parámetros formales. En este caso, en los parámetros transferidos se utilizan una constante, una expresión, y las variables del tipo correspondiente.

Parámetros formales

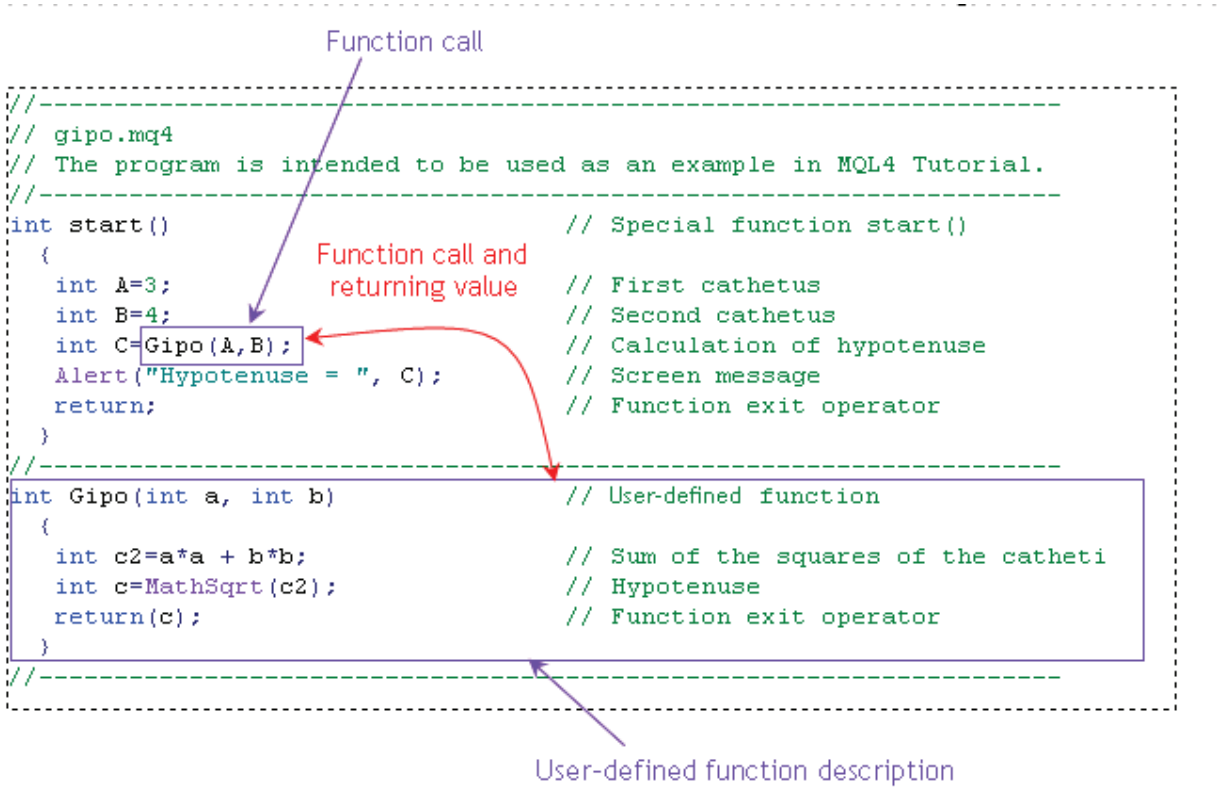
El punto culminante de las funciones es el uso de los parámetros formales.

Los **Parámetros formales** son una lista de variables especificadas en la cabecera de la descripción de la función.

Ya mencionamos antes que una misma función podría ser utilizada en varios programas. Sin embargo, los diferentes programas utilizan diferentes nombres para las variables. Si las funciones requirieran de forma estricta que se pusieran determinados nombres en las variables de los parámetros a transferir (y, correlativamente su valor), no sería conveniente para los programadores. De hecho, usted tendría que poner en cada programa recientemente desarrollado los nombres de variables que ya se han utilizado en sus funciones anteriores. Sin embargo, afortunadamente, las variables utilizadas dentro de las funciones no tienen relación con las variables utilizadas en el programa que lo llama.

Vamos a hacer referencia a la Fig. 19 una vez más. Cabe señalar que los nombres de los parámetros transferidos **(A** y **B** que se dan en el paréntesis de la llamada a la función) no coinciden con los nombres de los parámetros **(a** y **b)** que se especifica en la descripción de la función. Hemos observado en la sección Constantes y Variables que MQL4 distingue entre mayúsculas y minúsculas. De este modo, los nombres **a** y **A, B** y **b** deben considerarse aquí como diferentes nombres de variables. Sin embargo, no hay error en este código.

Las variables utilizadas en los **parámetros formales** de la descripción de la función no están relacionados con las variables utilizadas en el código básico del programa. Estas son variables diferentes. Solamente pueden especificarse variables (pero no constantes) como parámetros formales en cabecera de la función.




```
//+------------------------------------------------------------------+
// gipo.mq4
// The program is intended to be used as an example in MQL4 Tutorial.
//+------------------------------------------------------------------+
int start()                          // Special function start()
  {
   int A=3;                          // First cathetus
   int B=4;                          // Second cathetus
   int C=Gipo(A,B);                  // Calculation of hypotenuse
   Alert("Hypotenuse = ", C);        // Screen message
   return;                           // Function exit operator
  }
//+------------------------------------------------------------------+
int Gipo(int a, int b)               // User-defined function
  {
   int c2=a*a + b*b;                 // Sum of the squares of the catheti
   int c=MathSqrt(c2);               // Hypotenuse
   return(c);                        // Function exit operator
  }
//+------------------------------------------------------------------+
```

Fig. 19. El código de un programa que contiene la descripción y la llamada a función definida por el usuario gipo.mq4.

Cómo funciona

- En el programa, se produce una llamada a la función, las variables **A** y **B** están especificadas en su paréntesis.
- El programa llama a la función con ese nombre y que tiene los parámetros formales **a** y **b** que se especifican en su cabecera.
- El valor de variable **A** se le asigna a la varible **a**.
- El valor de la variable **B** se le asigna a la variable **b.**
- La función ejecutable realiza los cálculos utilizando los valores de las variables **a** y **b.**

Cualquier nombres se pueden usar en los parámetros formales (mientras no coincidan con otros nombres de variables ya utilizados en el programa). En este ejemplo, hemos utilizado los identificadores de los parámetros formales **a** y **b.** Sin embargo, podríamos utilizar cualquier otro, por ejemplo, **m** y **n,** o **Kat_1** y **Kat_2.** Por supuesto, en la redacción de un programa, usted debe especificar en el cuerpo de la función para los cálculos los nombres de variables que están en la cabecera. Puesto que hemos dado **a** y **b** en la cabecera, obviamente tenemos que utilizar **a** y **b,** dentro de la función y no **m** y **n.**

En la función, los cálculos se hacen de forma que se involucren parámetros **a** y **b y** no las variables **A** y **B** como ya hemos dicho**.** Una función se permite que cualquier acción se pueda realizar con los parámetros formales de **a** y **b** (incluidos los cambios en los valores de estas mismas variables) y esto no tendría ningún efecto sobre las variables **A** y **B** correspondientes a los parámetros de transferencia del operador de llamada del programa principal**.**

El valor de return calculado en la función se da en el paréntesis del operador return (). Como valor de return puede utilizarse el valor de una variable, el resultado de una expresión o una constante. En nuestro caso, el valor de retur es el valor de la variable local **c** (una variable local es una variable declarada dentro de una función, a la salida de la función, los valores de todas las variables locales se pierden, y por tanto su ámbito de trabajo es solo válido dentro de la función. Examinaremos con más detalles las variables locales en la sección Tipos de variables. La función devuelve al programa que lo llamó el valor de la variable local **c** (Fig. 19). Esto significa que este valor será asignado a la variable **C.**

Otros cálculos dentro del programa, si fuera el caso, se podrían realizar con las variables declaradas dentro de la función que la llama. En nuestro caso, la función que la llama es la función especial start () (que contiene la línea para llamar a la función definida por el usuario), mientras que las variables declaradas dentro de la función que la llama son **A, B** y **C,** en la función que realiza los cálculos se usan los **parámetros formales**, lo que nos permite crear funciones usando nombres de variables arbitrarios e independientes de los nombres de valores utilizados en el programa principal o función que le llama.

Ejemplo de función estándar en un programa

En primer lugar, vamos a examinar el comportamiento del programa se muestra en la Fig. 18. Debemos tener en cuenta que todo el código del programa se encuentra dentro de la función especial start (). En esta etapa de aprendizaje, no vamos a prestar especial atención a esta función (las funciones especiales y sus propiedades seran consideradas en más detalle en la sección Funciones especiales).

Unitized program

```
//+------------------------------------------------------------------+
// pifagor.mq4
// The program is intended to be used as an example in MQL4 Tutorial.
//+------------------------------------------------------------------+
int start()                                 // Special function start()
  {
   int A=3;                                 // First cathetus
   int B=4;                                 // Second cathetus
   int C_2=A*A + B*B;                       // Sum of the squares of the catheti
   int C=MathSqrt( C_2);                    // Calculation of hypotenuse
   Alert("Hypotenuse = ", C);               // Screen message
   return;                                  // Function exit operator
  }
//+------------------------------------------------------------------+
```

Vamos a seguir la ejecución del programa, empezando con el operador de asignación:

```
int A = 3;                    // Primer cateto
```

1. La parte derecha del operador de asignación contiene una constante, su valor es de 3.

2. El valor de 3 (el valor de la parte derecha) se le asigna a la variable **A** (que está situada a la izquierda del signo de igualdad).

Se le da el control a la línea siguiente:

```
int B = 4;                    // Segundo cateto
```

3. La parte derecha del operador de asignación contiene especificada una constante, su valor es de 4.

4. El valor de 4 se asigna a la variable **B.**

El programa se destina a la ejecución de la línea siguiente:

```
int C_2 = A*A + B*B;                    // Suma de los cuadrados de los catetos
```

5. Cálculo de la parte derecha del operador de asignación. El resultado de los cálculos es el valor de 25 (los detalles de cómo el programa se refiere a las variables para obtener sus valores son considerados en la sección Constantes y variables, por lo que no se particularizah este punto aquí y ahora).

6. Asignación del valor 25 a la variable **C_2.**

La siguiente línea representa un operador de asignación, de las cuales la parte derecha contiene una llamada a una función estándar:

```
int C = MathSqrt( C_2);                 // Cálculo de la hipotenusa
```

El programa tiene como objetivo ejecutar el operador de asignación. Con este próposito se ejecuta primero los cálculos situados a la derecha de la igualdad.

7. El programa requiere la ejecución de la función estándar MathSqrt () (que calcula raíces cuadradas). El valor de la variable C_2 (en nuestro caso igual a 25) se utilizará como parámetro el paso. Hay que tener en cuenta que no existe una descripción de esta función estándar en ninguna parte del programa. Las descripciones de funciones estándar no están situadas en los programas. En el texto de un programa, usted puede distinguir fácilmente la regla de llamada a la función por su aspecto: que se destacan en MetaEditor con color morado (programador puede elegir a voluntad los colores).

8. Los cálculos se realizan en el nivel de la función MathSqrt ().

9. La función estándar MathSqrt () completa sus cálculos y devuelve el valor obtenido. En nuestro caso, es el valor de **5** (la raíz cuadrada de 25).

El valor devuelto por la función es ahora el contenido del registro:

```
MathSqrt( C_2)
```

Este registro puede ser considerado como una especie de variable compleja especial, dentro de los cálculos que se realizan. Después de que estos cálculos se han completado, esta especie de variable toma un valor. Lo importante de esto aquí es el hecho de que el valor devuelto por la función puede ser asignado a otra variable o considerado de alguna manera en cualquier otro cálculo.

10. En este caso, nuestro valor es el valor de la parte derecha del operador de asignación. En la continuación de la ejecución, el operador de asignación el programa asigna el valor de 5 a la variable C.

11. La siguiente línea es el operador que contiene referencias a la función estándar de **Alert ()** (llamada a función).

```
Alert("Hipotenusa = ", C);              // Mensaje para la pantalla
```

La función estándar **Alert ()** abre un cuadro de diálogo en donde se muestran los valores de los parámetros transferidos. En este caso, la función ha tomado dos valores como parámetros de paso:

-- La cadena de valor constante: “Hipotenusa = ”

-- El valor integer de la variable **C:** 5

Se ha señalado anteriormente que no todas las funciones deberán devolver el valor (que es el resultado de la ejecución de la función). La función estándar **Alert ()** no devuelve ningún valor, ya que tiene otra tarea: mostrar el texto en la pantalla en una ventana especial.

Como resultado de la ejecución de la función estándar de **Alert ()** en la ventana de esta función aparecerá la línea siguiente:

```
Hipotenusa = 5
```

12. El último operador en este programa completa la labor de la función especial start ().

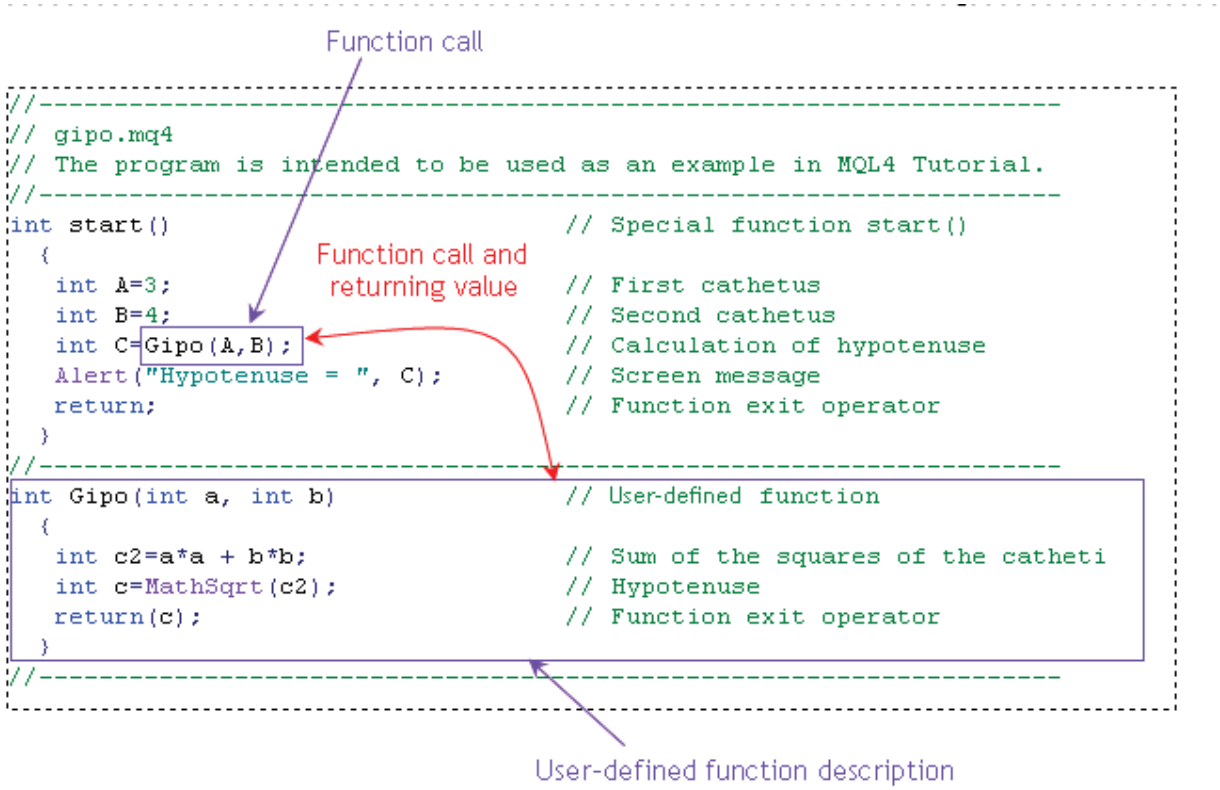
```
return;                          // Función salida del operador (vacia)
```

La labor del programa ha terminado en este momento.

Una pregunta puede surgir: ¿Cómo podemos saber qué función devolverá un valor, y cual no? La respuesta a esta pregunta es obvia: Con el fin de encontrar las descripciones detalladas de las funciones incorporadas, usted debe leer la documentación de referencia de MQL4.community, el sitio web lanzado por MetaQuotes Software Corp, o los archivos de ayuda de MetaEditor. Las propiedades de una **función definida por el usuario** se especifican en su descripción. Si una función definida por el usuario devuelve el valor o no, depende de su algoritmo (la decisión es adoptada por el programador en la fase de redacción del código de programa de la función).

Ejemplo de función definida por el usuario en un Programa

Vamos a considerar cómo se realizarían los mismos cálculos en un programa que contenga una función definida por el usuario (Fig. 19). Una cierta parte del código que se puso anteriormente que se dentro en la función especial start (), no estará aquí ahora. Este codigo será sustituido con la llamada a la función definida por el usuario. La descripción de esta función definida por el usuario irá después de la funcion especial start (). Las dos primeras líneas, en la que variables de tipo **A** y **B** tomar valores numéricos, siguen siendo los mismos. En consecuencia, nada cambia en su ejecución:




```
//+------------------------------------------------------------------+
// gipo.mq4
// The program is intended to be used as an example in MQL4 Tutorial.
//+------------------------------------------------------------------+
int start()                             // Special function start()
  {
   int A=3;                             // First cathetus
   int B=4;                             // Second cathetus
   int C=Gipo(A,B);                     // Calculation of hypotenuse
   Alert("Hypotenuse = ", C);           // Screen message
   return;                              // Function exit operator
  }
//+------------------------------------------------------------------+
int Gipo(int a, int b)                  // User-defined function
  {
   int c2=a*a + b*b;                    // Sum of the squares of the catheti
   int c=MathSqrt(c2);                  // Hypotenuse
   return(c);                           // Function exit operator
  }
//+------------------------------------------------------------------+
```

Fig. 19. El código de un programa que contiene la descripción y la llamada a función definida por el usuario gipo.mq4.

```
int A = 3;                  // Primer cateto
int B = 4;                  // Segundo cateto
```

En la tercera línea tenemos el operador de asignación. Su parte derecha contiene la convocatoria de la función definida por el usuario:

```
int C = Gipo(A,B);          // Cálculo de la hipotenusa
```

6. En la ejecución de este operador, el programa llama en primer lugar a la función definida por el usuario.

Nota: La descripción de la función definida por el usuario debe estar presente en el programa y se colocan inmediatamente después de que se cierra la llave de la función especial start () (es decir, se coloca fuera de la función especial star).

Al referirse a la función definida por el usuario, el programa lleva a cabo lo siguiente:

6,1. Copia del valor de la variable **A** con el fin de obtener su valor (en nuestro caso, **3)**

6,2. Copia del valor de la variable **B** con el fin de obtener su valor (en nuestro caso, **4)**

Nota: Tan pronto como el programa comienza a llamar a la función definida por el usuario (en este caso concreto la función es una función definida por el usuario, esta norma se aplica a todas las funciones), el programa se hace sólo una copia de los valores de las variables utilizadas, como parámetros transferidos, mientras que los valores de estas variables en si mismas (en este caso, **A** y **B)** no ha supuesto cambio debido a la aplicación de la función definida por el usuario, ni ellas realmente cambian.

7. El control se pasa a la función definida por el usuario.

Durante todo el tiempo de ejecución de la función definida por el usuario (no importa cuánto se tarde), los valores de las variables de la llamada a la función no se pierden sino que se almacenan.

La primera línea en la **descripción de la función** definida por el usuario es su cabecera:

```
int Gipo(int a, int b)              // Cabecera de la función definida por el usuario
```

En la ejecución de la función definida por el usuario, el programa va a hacer lo siguiente:

7,1. El valor de **3** (el primer valor en la lista de parámetros transferidos) se **copia** en la variable **a** (la primera variable en la lista de parámetros).

7,2. El valor de **4** (el segundo valor en la lista de parámetros transferidos) se **copia** en la variable **b** (la segunda variable en la lista de parámetros).

Entonces el control se debe pasar al cuerpo de la función para la ejecución de su algoritmo.

El primer operador en el cuerpo de la función es el siguiente:

```
int c2 = a*a + b*b;                 // Suma de los cuadrados de los catetos
```

7,3. En la ejecución de este operador, el programa calcula el valor en la parte derecha del operador de asignación, y después asigna el valor obtenido (en nuestro caso, 3*3 + 4*4 = **25)** a la variable **c2.**

El siguiente operador:

```
int c = MathSqrt(c2);               // Cálculo de la Hipotenusa
```

7,4. Aquí encontramos la raíz cuadrada del valor de la variable **c2.** El orden de operaciones es el mismo que en el ejemplo anterior. La ejecución del operador de asignación se traducirá en la asignación de valor de **5** a la variable **c.**

7,5. En la siguiente línea, tenemos el operador:

```
return(c);                          // Función salida operador
```

En la ejecución de este operador tenemos el return del programa del valor encerrado en el paréntesis de este operador. En nuestro caso, es el valor de la variable **c,** es decir, **5.** En este punto, la ejecución de la función definida por el usuario ha terminado, el control se devuelve al punto donde se hizo la llamada a la funcion.

8. Recordemos que la función definida por el usuario se llamó desde el operador

```
int C = Gipo(A,B);                  // Cálculo Hipotenusa
```

El registro de la llamada a la funcion definida por el usuario queda en:

```
Gipo(A,B)
```

en la fase de la devolución del valor, esta expresión toma el valor calculado de la función (en nuestro caso, es el valor de **5).**

Tras concluir la ejecución del operador de asignación, el programa asigna el valor de 5 a la variable **C.**

9. El siguiente operador,

```
   Alert("Hipotenusa = ", C);          // Mensaje en la pantalla,
```

se ejecutará de la misma manera que en el ejemplo anterior, a saber: en una ventana especial, aparecerá el mensaje:

```
Hipotenusa = 5
```

10. El último operador en este programa,

```
  return;                  // Salida del operador y salida de la funcion especial star (sin parámetros)
```

Queda completa la labor de la función especial start (), y a la vez queda completa la labor de todo el programa (las propiedades de las funciones especiales se consideran con más detalles en la sección Funciones especiales).

Vamos a considerar algunas versiones de la aplicación de la mencionada función definida por el usuario. Es fácil verificar que la programación con funciones definidas por el usuario tiene algunas ventajas incontestables.

Consideraciones previas de la implementacion de la función definida por el usuario Gipo ()

En esta función, los parámetros formales "se asemejan a" las variables utilizadas en el programa básico. Sin embargo, esto no es más que una similitud formal, porque **A** y **a** son diferentes nombres de variables, pero los valores, es decir, **el contenido** de las variables **A** y **B** se **copian** dentro de las variables **a** y **b.**

```
//------------------------------------------------------------------
int Gipo(int a, int b)                // Función definida por el usuario

  {
   int c2 = a*a + b*b;                // suma de los cuadrados de los catetos
   int c = MathSqrt(c2);              // Hipotenusa
   return(c);                         // Función salida operador  }
//------------------------------------------------------------------
```

Aplicación de la función definida por el usuario Gipo (): Versión 2

En este caso, los nombres de parámetros formales no "se asemejan a" los nombres de variables utilizadas en el programa básico. Sin embargo, esto no nos impide utilizar esta función en el programa.

```
//------------------------------------------------------------------
int Gipo(int alpha, int betta)       // Cabecera de la descripcion de la función definida por el usuario
  {
   int SQRT = alpha*alpha + betta*betta; // suma de los cuadrados de los catetos
   int GIP = MathSqrt(SQRT);          // Hipotenusa
   return(GIP);                           // Función salida operador
  }
//------------------------------------------------------------------
```

Aplicación de la función definida por el usuario Gipo (): Versión 3

En este ejemplo, la variable alfa se reutiliza en el programa y cambia su valor dos veces. Este hecho no tiene ningún efecto sobre las variables reales de la funcion de llamada del programa principal.

```
//------------------------------------------------------------------
int Gipo(int alpha, int betta)          // función definida por el usuario
  {
   alpha= alpha*alpha + betta*betta;     // suma de los cuadrados de los catetos
   alpha= MathSqrt(alpha);              // Hipotenusa
   return(alpha);                     // Función salida operador
  }
//------------------------------------------------------------------
```

Aplicación de la función definida por el usuario Gipo (): Versión 4

En esta funcion definida por el usuario, todos los calculos se recogen en un solo operador. El valor de return se calcula directamente en el paréntesis del operador return (). La expresión se calcula directamente en el lugar donde el parámetro transferido debe ser especificado en la función estandar MathSqrt (). Esta solución puede parecer extraña o mala al principio. Sin embargo, no hay error en esta forma de utilización de la función definida por el usuario. El número de operadores utilizados en la función es menor que en otras implementaciones, por lo que el código resulta ser más compacto.

```
  //------------------------------------------------------------------
int Gipo(int a, int b)                // Función definida por el usuario
  {
   return(MathSqrt(a*a + b*b));        // Operador Función Salida
  }
//------------------------------------------------------------------


//---------------------------------------------- --------------------
  int Gipo(int a, int b) // función definida por el usuario
  (
  return (MathSqrt (a*a + b*b)); // Función salida operador
  )
  //---------------------------------------------- --------------------
```

De este modo, la aplicación de funciones definidas por el usuario tiene algunas ventajas indiscutibles en la programación de la práctica:

- los nombres de variables en el texto del programa principal no tienen relacion con los nombres de los parámetros formales de una función definida por el usuario;
- funciones definidas por el usuario pueden ser reutilizados en diferentes programas, no hay necesidad de cambiar el código de la función definida por el usuario;
- se pueden crear librerías, si es necesario.

Estas propiedades tan útiles de las funciones permiten crear programas realmente grandes, como por ejemplo para una corporación. Varios programadores pueden participar en este proceso de manera simultánea, y cada uno de ellos queda liberado de la necesidad de llegar a un acuerdo sobre los nombres de las variables que utilizan. Lo único que se deberá acordar es el orden de las variables en la cabecera y en la función llamada.

Tipos de programa

Al comenzar a escribir un programa en MQL4, el programador debe, en primer lugar, responder a la pregunta acerca sobre qué tipo de programas va a ser. El contenido y la funcionalidad del programa dependen plenamente de esto. En MQL4, hay 3 tipos de programas de aplicación: Asesores Expertos (**Expert Advisor)**, **scripts**, e **indicadores definidos por el usuario** (custom indicator). Cualquier programa desarrollado por un programador pertenece a uno de estos tipos. Todos ellos tienen sus propósitos y características especiales. Vamos a considerar estos tipos con más detalle.

El **Asesor Experto (AE)** es un programa codificado en MQL4 e invocado por el Terminal de Usuario para ser ejecutado en cada uno de los tick. El objetivo principal de los Asesores Expertos es programar el control sobre el comercio. Los Asesores Expertos son codificados por los usuarios. No hay una función de AEs en el Terminal de Usuario.

El **Script** es un programa codificado en MQL4 y ejecutado por el Terminal de Usuario **una sola vez**. Los scripts son destinados a realizar cualquier tipo de operaciones que permitan ser ejecutadas una sola vez. Los scripts son codificados por los usuarios. Ellos no vienen incorporados como lo está el Terminal de Usuario.

El **Custom indicator** es un programa codificado en MQL4 e invocado por el Terminal de Usuario para ser ejecutado, al igual que el AE, en todos los ticks. Esta básicamente destinado a la exhibición gráfica de funciones matemáticas calculadas preliminarmente como líneas. Hay dos tipos de indicadores: indicadores técnicos (built-in) y los custom indicator, y estos últimos son como los indicadores tecnicos pero creados por el propio usuario. Los indicadores son considerados en más detalles en las secciones del uso de los indicadores técnicos y la creación personalizada de los indicadores.

El programador elige el tipo de programa que va a ser escrito en función del propósito de ese programa específico y las propiedades y limitaciones de los diferentes tipos de programas.

Propiedades de los Programas

Creación de un Programa de Ejecución

Hay un criterio que distingue a los Asesores Expertos y los custom indicator de los scripts, y este es su tiempo de duración. En la sección algunos conceptos básicos, hemos mencionado ya que los programas se ponen en marcha durante un tiempo que es múltiplo de la cantidad de ticks. Esta afirmación es cierta para AEs y para los customs indicator, pero es falso para los scripts.

Asesor Experto y el custom indicator. Una vez que se haya vinculado un programa (EA o custom indicator) a la ventana de símbolo o instrumento, el programa hace algunos preparativos y cambia al modo de espera de ticks. Tan pronto como un nuevo tick entra, el Terminal de Usuario lo pondrá en marcha para su ejecución, entonces, el programa hace todas las operaciones descritas en su algoritmo, y, una vez que termina, pasa el control de nuevo al Terminal de Usuario y permanece en el modo de espera de tick.

Si un nuevo tick llega cuando el programa se está ejecutando, este evento no tiene ningún efecto sobre la ejecución del programa, el programa sigue siendo ejecutado de acuerdo a su algoritmo y solo pasa el control al Terminal de Usuario cuando haya terminado todas las tareas descritas en el algoritmo. Es por ello que no todos los ticks dan como resultado el lanzamiento de un AE o un indicador, sino sólo aquellos que entran cuando el control está en el Terminal de Usuario y el programa se encuentra en el modo de espera de ticks.

Un nuevo tick inicia el programa para su ejecución. De este modo, un Asesor Experto o un indicador puede operar dentro de un largo período de tiempo en la ventana de símbolo o instrumento al que se asocia ya que cada vez que termina su tarea el programa se ejecuta una y otra vez en función de cada nuevo tick.

Un Asesor Experto se diferencia de un indicador en el orden de ejecución del primer lanzamiento del programa. Esta diferencia viene determinada por las propiedades específicas de las funciones especiales de cada tipo de programa (véase Funciones especiales). Una vez que el programa se asocia a la ventana de símbolo o instrumento, el Asesor Experto hace los preparativos necesarios (función init ()) y cambia al modo de espera de tick preparado para iniciar la función start (). A diferencia de los AEs, el custom indicator ejecuta tanto la función init () como la función start () una vez a hace el primer cálculo preliminar del valor de indicador. Más tarde, con un nuevo tick, el programa se inicia llamando únicamente a la función start (), es decir, los operadores trabajan de acuerdo con el algoritmo de la función start ().

Script. A diferencia de los Asesores Expertos o los indicadores, un script se pondrá en marcha para su ejecución inmediatamente después de que haya sido asociado a una ventana de símbolo o instrumento, sin esperar a un nuevo tick. Todo el código del script se ejecutará de una vez. Después de que todas la lineas del programa se han ejecutado, el script da fin a sus operaciones y se descarga desde la ventana de símbolo o instrumento. Un script es útil si quieres hacer operaciones de una sola vez, por ejemplo, para abrir o cerrar órdenes, para mostrar textos en la pantalla, para instalar objetos gráficos, etc

Las diferencias en la ejecución de Asesores Expertos, scripts personalizados y los indicadores están determinada por las propiedades de sus funciones especiales esto será considerado con más detalles en la sección Funciones especiales.

Trading

Uno de los principales criterios que marcan los programas anteriores es la posibilidad de ejecutar instrucciones de trading. Una instrucción de trading es un órden que pasa a un programa servidor de intercambios con el fin de abrir, cerrar o modificar órdenes. Las instrucciones de Trading se forman en los programas utilizando funciones incorporadas que llamamos "funciones de trading".

Sólo los Asesores Expertos y los scripts tienen la posibilidad de utilizar las funciones de trading (sólo si la opción correspondiente está activada en la configuración del AE/script). En los customs indicator (indicadores personales) no está permitido el empleo de funciones comerciales (funciones de trading).

Uso simultáneo

Los programas también difieren entre sí por la cantidad de programas de diferentes tipos que al mismo tiempo se asocian a una ventana de símbolo o instrumento.

Asesor Experto. Solo se puede asociar un AE en una ventana de símbolo; no está permitido el uso simultáneo de varios Asesores Expertos.

Script. Solo se puede asocial un script en una ventana de símbolo; no está permitido el uso simultáneo de varios script.

Indicador personal. Se pueden asociar al mismo tiempo varios indicadores en una ventana de símbolo pero de manera que no interfieran entre sí.

Se pueden poner en marcha simultáneamente programas de los tres tipos en una ventana de símbolo en conformidad con las limitaciones de cada tipo. Por ejemplo, se puede lanzar un EA, un scrip y varios indicadores en una ventana de símbolo al mismo tiempo, o una AE y uno de los indicadores. Sin embargo, usted no puede iniciar varias AEs o scripts en una ventana de símbolo, los programas de otros tipos se pueden ponen en marcha simultáneamente.

Pueden iniciarse al mismo tiempo programas del mismo tipo en diferentes ventanas de un símbolo o instrumento. Por ejemplo, si se desea iniciar dos Asesores Expertos en un símbolo o instrumento, se puede iniciar un AE en una ventana de este símbolo y otro en otra ventana del mismo símbolo. En este caso, los Asesores Expertos trabajarán simultáneamente. Sin embargo, usted debe tener en cuenta que AEs que se inician de esta manera podrán formarse instrucciones de trading contradictorias. Por ejemplo, uno de ellos puede dar instrucciones para abrir una posicion, mientras que el otro se puede ordenar de cerrar la posicion. Esto puede provocar que se produzca una larga secuencia de órdenes inútiles que se traduce en la pérdida total.

En cualquier tipo de programa se pueden crear variables globales disponibles para todos los demás programas puestos en marcha en la teminal de usuario, incluido los que se pusieron en marcha en ventanas de símbolo diferentes. Esto permite que la máquina coordinar operaciones simultáneas de todos los programas. La orden de utilizar variables globales será especialmente considerada en la sección GlobalVariables.

Llamando a programas para su ejecución

Cualquier tipo de programa se ejecuta solo a voluntad del usuario. En MQL4, no se puede llamar a un Asesor Experto, un script, o un indicador mediante una ejecución programática, es decir, desde una llamada desde otro programa. La única excepción es la llamada incorporada en función **iCustom ()** que permite hacer referencia a un indicador personal, y la llamada a las funciones de indicadores técnicos estandar. La referencia a la función iCustom () o a las funciones de indicadores técnicos no da lugar al dibujo de las líneas de los indicadores en la ventana de símbolo (ver a simple Programas MQL4).

Cuadro 2. Principales propiedades de Asesores Expertos, scripts e indicadores personales.

Propiedad del Programa	Asesor Experto	Script	Indicador
Duración de la Ejecución	Durante un largo período	Una sola vez	Durante un largo período
Trading	Permitido	Permitido	No Permitido
Visión de líneas	No	No	Sí
Uso simultáneo de varios programas del mismo tipo	No Permitido	No Permitido	No Permitido
Llamada desde ejecución programática	No Permitida	No Permitida	No Permitida

Por lo tanto, si queremos un programa que gestione el comercio con arreglo a cierto algoritmo, deberíamos escribir un AE o un script. Sin embargo, si queremos tener una cierta función matemática representada gráficamente, deberíamos utilizar un indicador.

EL MetaEditor

En esta sección vamos a hablar en las ordenes generales de la creación de programas de aplicación utilizando MetaEditor.

El MetaEditor es un editor especializado multifunción destinados a la creación, edición y compilación de programas de aplicación escritos en MQL4. El editor tiene un interfaz de fácil uso que permite a los usuarios navegar fácilmente al escribir y revisar un programa.

- Sistema de archivos
  El MetaEditor almacena todos los códigos fuente de MQL4, programas en un catálogo estructurado propio en el disco duro. La ubicación de un programa en MQL4 está determinada por su próposito: un script, un Asesor Experto o un indicador, incluyen un archivo o una librería.

- Creación y uso de programas
  Es muy fácil crear un programa en MQL4, las herramientas que lleva incorporadas le ayudarán. Usted puede modificar las plantillas para la creación de scripts, los indicadores o Asesores Expertos. El código creado se guardará automáticamente en una carpeta del sistema de archivos MetaEditor.

Sistema de archivos

El programa Terminal de Usuario reconoce los tipos de su ubicación en los directorios subordinados.

Todos los programas de aplicación se concentran en el directorio ***ClientTerminal_folder \ expertos.*** Asesores Expertos, scripts y los indicadores personalizados de un trader que se van a utilizar en su trabajo práctico deberían estar situados en los directorios correspondientes (ver Fig. 24). Los Asesores Expertos se encuentran en el directorio ***ClientTerminal_folder \ expertos,*** scripts e indicadores en subdirectorios ***ClientTerminal_folder \ expertos \ scripts*** y ***ClientTerminal_folder \ expertos \ indicadores.***

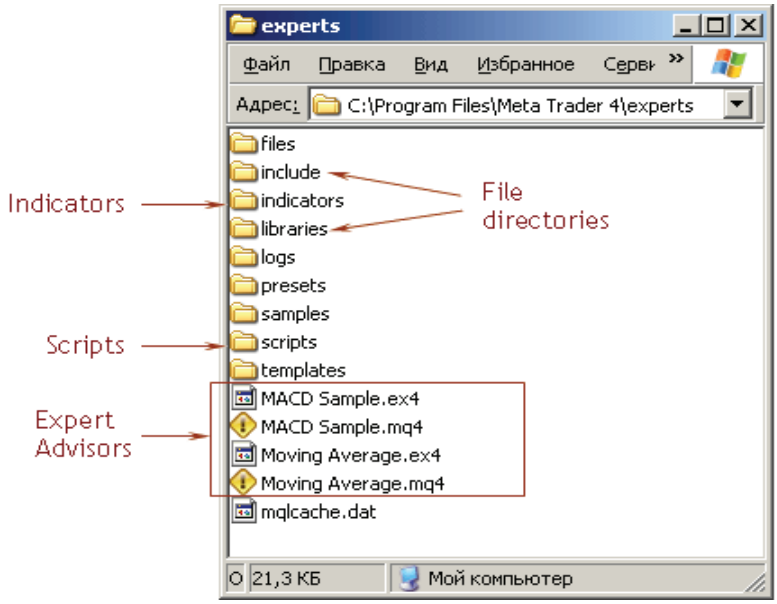


Fig. 24. Directorio para el almacenamiento de archivos, creado por un usuario.

Un usuario puede crear otros directorios para almacenar algunos archivos. Sin embargo, el uso de programas listos situado en ese directorio no está previsto en el Terminal de Usuario.

Tipos de archivo

En MQL4 hay tres tipos de archivos que llevan un código de programa: mq4, ex4 y mqh.

Los archivos de tipo **mq4** representan un programa de código fuente escrito en **mq**l**4**. Los archivos de este tipo contienen los textos de origen de todo tipo de programas (Asesores Expertos, scripts e indicadores). Se utilizan para la creación de códigos de programa MetaEditor. Cuando un código haya sido total o parcialmente creado, puede ser guardado y después abrirlo para su modificación; este archivo es de tipo mq4. Para iniciar la ejecución de un programa en el Terminal de Usuario el archivo **mq4** debe ser compilado primero. Como resultado de la compilación el código fuente, crea un archivo del mismo nombre con la extensión **ex4**, que es un archivo "**ex**ecutable de mql**4**".

Un fichero de tipo **ex4** es un programa compilado listo para su uso práctico en el Terminal de Usuario. Los archivos de este tipo no pueden ser editados. Si un programa tiene que ser modificado, esto debe hacerse en su código fuente (archivo tipo **mq4**): debe ser editado y compilado luego de nuevo. El nombre del archivo no es un indicio de que el programa se trata de un script, un asesor experto o un indicador. Los archivos con extensión ex4 se puede utilizar como archivos de la libreria.

Los archivos de tipo **mqh** se incluyen archivos. Es una fuente de texto utilizado con frecuencia como bloques en programas de usuario. Estos archivos pueden ser incluidos en los textos de origen de Asesores Expertos, los scripts y los indicadores en la fase de compilación. Por lo general, incluyen archivos que contienen la descripción de funciones importadas (como ejemplo, ver archivos stdlib.mqh o WinUser32.mqh) o la descripción de constantes y variables comunes (stderror.mqh o WinUser.mqh). Por regla general, los archivos de tipo **mqh** se almacenan en el directorio ***ClientTerminal_folder \ expertos \ incluir.***

Incluir archivos se llaman así, porque generalmente son "incluidos" en la fase de compilación para la principal fuente de archivo usando la directiva #include. A pesar de que un archivo de tipo **mqh** puede contener un programa de codigo fuente y puede ser compilado por el MetaEditor, no son independientes en si mismos, es decir, no requieren de compilación para obtener archivos ejecutables de tipo ex4. Como incluir archivos, se pueden utilizar archivos mq4 que deben guardarse en ***ClientTerminal_folder \ expertos \ incluir.***

Las secciones "Asesores Expertos ", "Indicadores personalizado" y "scripts" del navegador terminal de usuario sólo mostrarán los nombres de los archivos que tienen la extensión ex4 y se encuentran en la carpeta correspondiente. Los archivos compilados en versiones anteriores de MetaEditor no pueden ser iniciados y se muestran en color gris.

Existen otros tipos de archivos que no hacen un programa completo, pero se utilizan en la creación de programas de aplicación. Por ejemplo, un programa puede ser creado fuera de varios archivos independientes o usando una libreria creada anteriormente. Un usuario puede crear librerías de funciones personalizadas destinadas al almacenamiento para uso frecuente de bloques de programas de usuario. Se recomienda almacenar las librerías en el directorio ***ClientTerminal_folder \ expertos \ libreries.*** Los archivos de **mq4** y **ex4** se pueden utilizar como archivos de la libreria. Las librerías no pueden ejecutar por si mismas. El uso de archivos de inclusión es preferible que el uso de librerías por el consumo adicional de recursos de la computadora en la llamadas a funciones de librería.

En la primera parte del libro "Programación en MQL4" vamos a analizar archivos de textos de código fuente **mq4** y los archivos compilados **ex4**.

Creación y uso de programas

Los programas de aplicación escritos en MQL4: los Asesores Expertos, scripts e indicadores son creados utilizando el MetaEditor.

El archivo ejecutable de MetaEditor **(MetaEditor.exe),** se ofrece como parte del Terminal de Usuario y se encuentra en el directorio raíz de la terminal. El Userguide de MetaEditor se abre presionando **F1.** Contiene información de carácter general necesarias para la creación de nuevos programas. El editor se puede abrir haciendo clic sobre el nombre del archivo **MetaEditor.exe** o en un acceso directo ubicado preliminarmente en el escritorio.

Estructura del Terminal de Usuario

Para mayor comodidad de operación, MetaEditor ha incorporado las barras de herramientas: "Navigator" **(Ctrl + D)** y "Toolbox" **(Ctrl + T).**

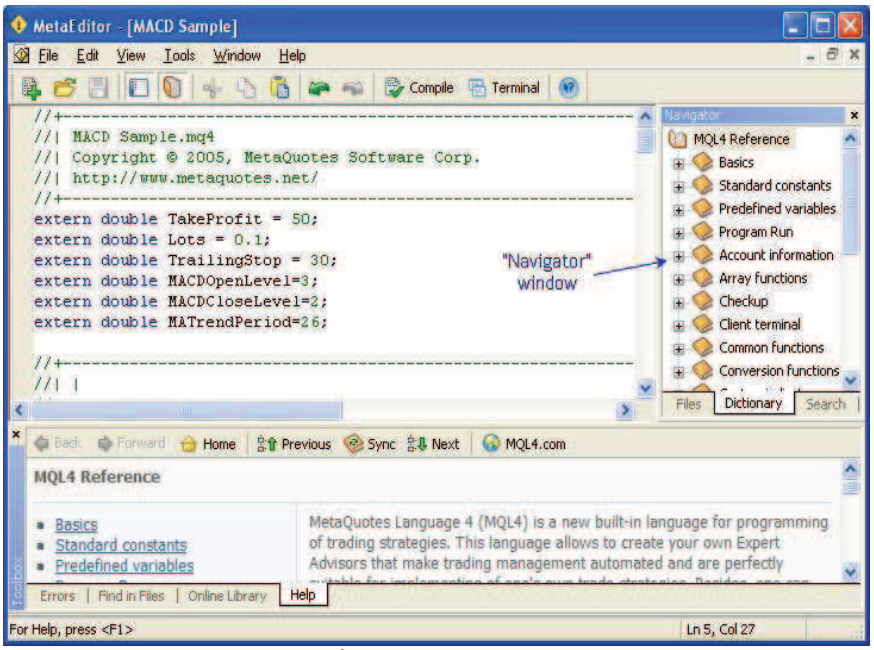


Fig. 25. Ubicación de las ventanas en MetaEditor.

El texto del programa se encuentra en la ventana del editor, las ventanas son herramientas auxiliares. Las ventanas del navegador y la caja de herramientas tienen movimiento y pueden ser mostradas/ ocultadas en el editor usando los botones  y .

La creación de un nuevo programa

Por lo general, durante la creación de un nuevo programa, las ventanas de la caja de herramientas y del navegador y están ocultas. De este modo la atención del usuario se concentra en la creacion del programa. Para crear un nuevo programa, utilice el editor de menú **Archivo>> Crear** o el botón o para la creación de nuevos archivos .

Después de todas estas acciones **"Expert Advisor Wizard"** le ofrecerá una lista para elegir el tipo de programa que quiere ser creado:

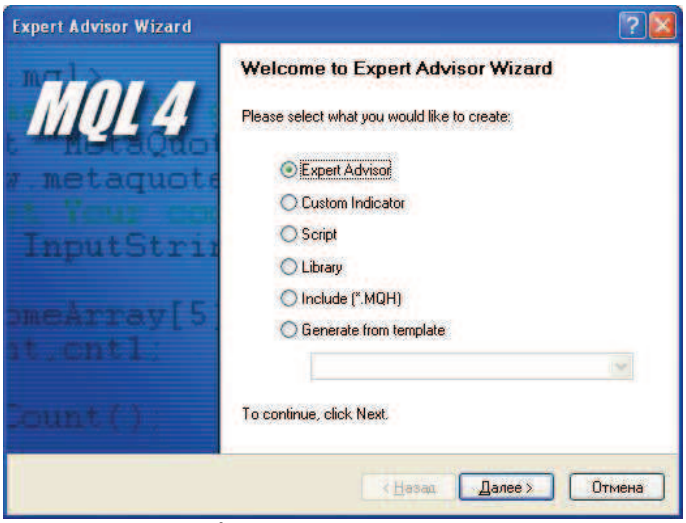


Fig. 26. La elección de un tipo de programa a ser creado.

Si necesitas crear un Asesor Experto, elegir **Expert Advisor** y haga clic en **Siguiente.** En la siguiente ventana es necesario escribir el nombre del Asesor Experto que quiere ser creado. Supongamos que se llama create.mq4.



El nombre de un archivo es creado es escrito sin su extensión (indicación de tipo).

El asistente del Asesor Experto mostrará una ventana con varios campos a rellenar:

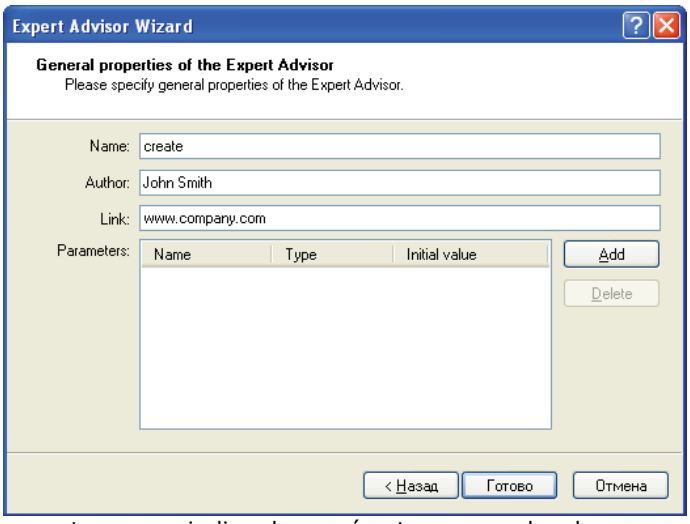


Fig. 27. Una ventana para indicar los parámetros generales de un experto asesor.

Después de hacer clic en **Aceptar** aparecerá un texto en la ventana principal y el nombre completo del Asesor Experto creado **create.mq4** se publicará en el sistema de archivos y en la ventana del navegador.

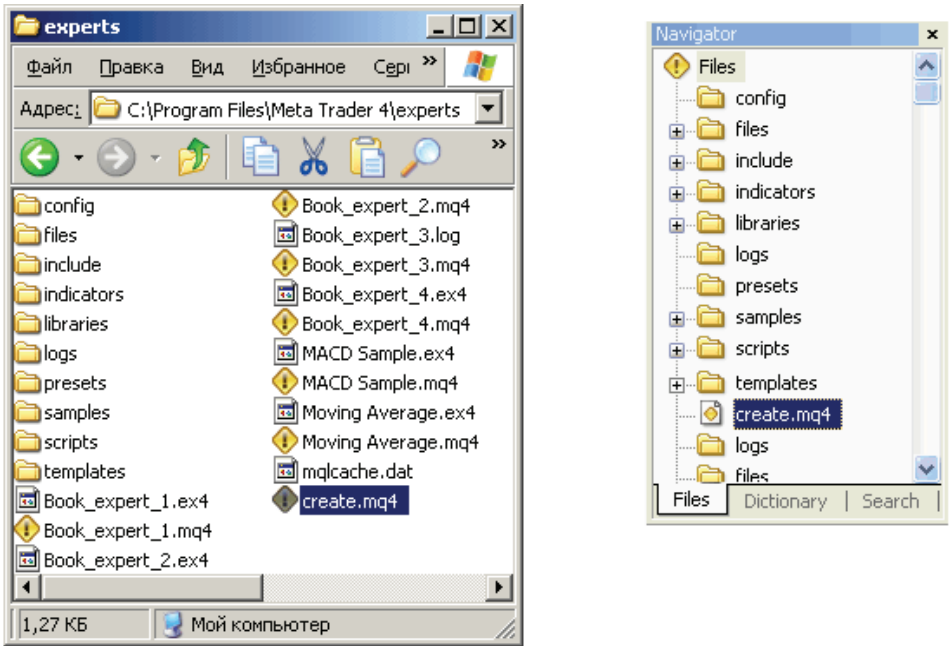


Fig. 28. Vista de un archivo creado de un Asesor Experto en las ventanas del sistema de archivos y del navegador.

Veamos el texto del programa generado por MetaEditor:

```
//+------------------------------------------------------------------+
//|                                                      create.mq4 |
//|                                                      John Smith |
//|                                                www.company.com |
//+------------------------------------------------------------------+
#property copyright "John Smith"
#property link      "www.company.com"

//+------------------------------------------------------------------+
//| expert initialization function                                   |
//+------------------------------------------------------------------+
int init()
  {
//----

//----
   return(0);
  }
//+------------------------------------------------------------------+
//| expert deinitialization function                                 |
//+------------------------------------------------------------------+
int deinit()
  {
//----

//----
   return(0);
  }
//+------------------------------------------------------------------+
//| expert start function                                            |
//+------------------------------------------------------------------+
int start()
  {
//----

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

Como se puede ver, el código contiene principalmente comentarios. Ya sabemos que las observaciones no constituyen una parte obligatoria de un programa y el texto de los comentarios no es procesado por el programa.

Hay tres funciones especiales en el programa: **init (), start () y deinit ()**. Cada función contiene un solo operador, **return (0)**, que es el operador para salir de la función. Así, un programa de código generado por Expert Advisor Wizard (Asistente del Asesor Experto) es sólo un patrón mediante el cual un programador puede crear un nuevo programa. El código final del programa no tiene que contener obligatoriamente todas las funciones especiales indicadas. Ellas están presentes en la pauta, sólo porque, como por regla general, un programa de nivel medio habitualmente contiene todas estas funciones. Si alguna de las funciones no serán utilizados, su descripción puede ser eliminada.

Las siguientes líneas de código del programa también pueden omitirse:

```
#property copyright "John Smith"
#property link      "www.company.com"
```

Aunque el programa no es de uso práctico, está escrito correctamente desde el punto de vista de la sintaxis. Y este programa puede ser compilado y ejecutado. Sería ejecutado igual que cualquier otro programa aunque no se realizaría ningun tipo de cálculos ya que no hay existe ninguno en el código fuente).

Apariencia del Programa

El uso de comentarios en los programas es altamente recomendable y en algunos casos es esencial. Y hay que destacar que un programador no sólo contribuye a crear programas, sino que también los lee y a veces puede tener considerables dificultades al leer un programa. La experiencia de muchos programadores muestra que la lógica de razonamiento, sobre la base de un programa que fue desarrollado, pueden ser olvidadas (o desconocidas en un producto de otro programador) y sin ristras de comentarios es difícil, a veces incluso imposible comprender los fragmentos de código.



Un programa codificado correctamente definitivamente contiene comentarios.

Las principales ventajas de las observaciones son las siguientes:

- En primer lugar, los comentarios permiten separar lógicamente una parte de otra del programa. Es mucho más fácil leer un texto formateado sabiamente que un texto liso (sin apartados).
- En segundo lugar, las ristras de observaciones permiten explicar en términos sencillos lo que significa cada línea de código a un programador independiente.
- En tercer lugar, en la parte superior del programa, puede ser especificada información general sobre el programa: nombre del autor y los contactos (incluido el sitio web, e-mail, etc), tarea del programa (si se trata de un programa de comercio completo o una función de librería), sus principales características y las limitaciones y otra informaciones útiles.

Cada programador puede elegir un estilo de comentarios cómodo. El estilo MQL4 ofrecidos por los desarrolladores se presenta en el Asesor Experto create.mql4. Vamos a ver las principales características de cualquier estilo de apariencia aceptable.

1. La longitud de una línea de comentario no debe exceder el tamaño de la ventana principal. Esta límitación no es un requisito formal de sintaxis, pero la lectura de un programa que contenga las líneas tan largas no es conveniente. Cualquier fila larga se puede dividir en varias líneas para que todas sean plenamente visibles en la pantalla. Para un monitor con 1024 x 768 píxeles de resolución, la máxima longitud de la línea es 118 símbolos.

2. La declaración de variables se realiza en el programa de inicio. Se recomienda escribir un comentario descriptivo para cada variable: explicar su significado brevemente y, si fuera necesario, las peculiaridades de uso.

3. Cada operador está mejor situado en una línea distinta.

4. Si hay un comentario en una línea debe iniciarse a partir de la 76 ª posición (recomendado para monitores 17 " con 1024 x 768 píxeles de resolución). Este requisito no es obligatorio. Por ejemplo, si una línea de código tiene 80 posiciones, no es necesariamente dividido en dos líneas, un comentario puede ser iniciada desde la 81ª posición. Por lo general, parte del código de programa contiene 50 símbolos de longitud de líneas y la ristra del comentario parece una columna de texto en la parte derecha de la pantalla.

5. Para dividir lógicamente fragmentos separados, se utilizan línea continua observaciones del ancho total (118 símbolos).

6. Cuando se utilizan las llaves, una tabulación tamaño sangrado debe ser utilizado (usualmente 3 símbolos).

Vamos a ver, que aspecto puede tener un Asesor Experto después de tener un programa de código escrito en ella. En este caso, no se discute la lógica del algoritmo escrito. Estamos interesados la apariencia del programa. Un programa comentado (Asesor Experto create.mq4) pueden tener la siguiente forma:

```
//+------------------------------------------------------------------+
// create.mq4
// To be used as an example in MQL4 book.
//+------------------------------------------------------------------+
int Count=0;                              // Global variable
//+------------------------------------------------------------------+
int init()                               // Spec. funct. init()
  {
  Alert ("Funct. init() triggered at start");  // Alert
  return;                                // Exit init()
  }
//+------------------------------------------------------------------+
int start()                              // Spec. funct. start()
  {
  double Price = Bid;                    // Local variable
  Count++;                               // Ticks counter
  Alert("New tick ",Count,"  Price = ",Price);// Alert
  return;                                // Exit start()
  }
//+------------------------------------------------------------------+
int deinit()                             // Spec. funct. deinit()
  {
  Alert ("Funct. deinit() triggered at exit"); // Alert
  return;                                // Exit deinit()
  }
//+------------------------------------------------------------------+
```

Es fácil ver los bloques del programa más sigficativos cuando son separados por los comentarios con las líneas discontinuas. Esta es una forma especial de separar las funciones definidas por el usuario y la cabecera de un programa:

```
//+------------------------------------------------------------------+
```

Las variables se declaran en un bloque donde es descrita cada variable. A veces los programas contienen variables y para describir las observaciones deben utilizarse varias líneas. Este es un caso raro, pero si ocurre, por ejemplo, un comentario debe ser necesariamente colocado; de otra forma, no sólo otro programador, sino que el propio autor será incapaz de armar el rompecabezas que puede suponer comprender el programa después de un cierto periodo de tiempo.

La parte derecha de cada línea de código contiene un comentario explicativo. El valor de las observaciones puede ser plenamente apreciado si el programa no contiene alguno de ellos, y algún problema con la comprensión en la lectura del algoritmo. Por ejemplo, si el mismo código, se presenta sin observaciones ni bloques de separación, será más difícil leer, aunque el programa sea muy corto y sencillo:

```
int Count=0;
int init() {
Alert (Funct. init() triggered at start");
return; }
int start() {
double Price = Bid;
Count++;
Alert("New tick ",Count,"   Price = ",Price);
return; }
int deinit(){
Alert (""Funct. deinit() triggered at exit");
return;}
```

Programa de Compilación

Para hacer un programa utilizable en la práctica, debe ser compilado. Con este fin, debe utilizarse el botón 

(F5) en MetaEditor. Si un programa no contiene ningún error, será compilado y un mensaje se producirá en la caja de herramientas:

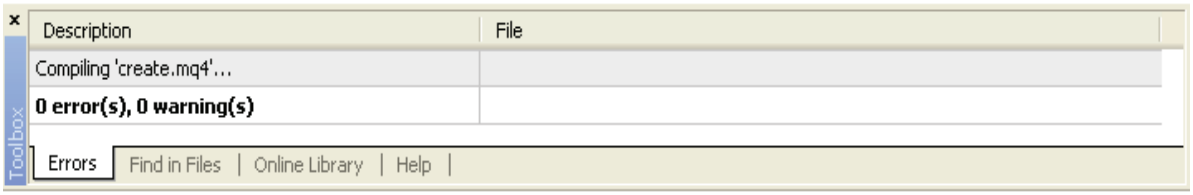


Fig. 29. Mensaje del Editor de un programa compilado con exito.

Además, un nuevo archivo **create.ex4** aparecerá en el directorio correspondiente (en este caso en ***Terminal_directory \ expertos).*** Este es ya un programa listo para su funcionamiento desde el Terminal de Usuario MetaTrader4. Durante la compilación la última versión del texto de origen del programa en relación con el mismo nombre (en nuestro caso es el archivo **create.mq4)** se guardarán en el mismo directorio.

Junto con una línea con el nombre del Asesor Experto creado aparecerá en la sección de Asesores Expertos del navegador del Terminal de Usuario la siguiente ventana:

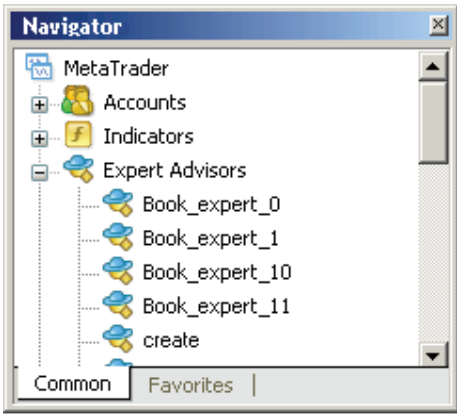


Fig. 30. Vista del nombre de un Asesor Experto en el navegador de la ventana del Terminal de Usuario.

Si durante la compilación se detectan errores en un programa, el MetaEditor mostrará el correspondiente mensaje de error. En tal caso, uno debe volver a editar el texto de origen, reparar errores y tratar de compilar el programa, una vez más. Una compilación exitosa solo es posible si no hay errores en el programa.

El uso de un Programa de Prácticas

Si un programa de aplicación (Asesor Experto, script o indicador) se ha compilado con éxito y su nombre ha aparecido en el navegador de la ventana del Terminal de Usuario, este programa puede ser utilizado en la práctica. Para utilizarlo, se arrastra el icono correspondiente a la ventana del navegador dentro de una ventana de un símbolo utilizando un ratón mediante método "drag & drop". Esto significa que el programa se vincula a un gráfico de un valor para que se inicie su ejecución.

Un Asesor Experto y un indicador funcionarán hasta que un terminal de usuario termine la ejecución del programa manualmente. Un script de usuario deja de operar por sí mismo cuando termina la ejecución de su algoritmo.

Es importante señalar aquí una vez más que:



Todos los programas de aplicación (Asesor Experto, indicador, script) pueden ser utilizados para el comercio solo como parte del Terminal de Usuario de MetaTrader 4 cuando éste está conectado al servidor (dealing center) a través de Internet. Ninguno de los programas pueden ser instalados sobre un servidor o ser usados en terminales de otros desarrolladores.

En otras palabras, si un comerciante quiere usar cualquier programa de aplicación, debe cambiar a un ordenador que tenga abierto el Terminal de Usuario de MetaTrader 4 e iniciar un archivo ejecutable *. ex4 en una ventana de un símbolo. Durante la ejecución del programa (dependiendo de su algoritmo) las órdenes de comercio pueden ser formadas y enviadas a un servidor, y por lo tanto, realizar la gestión del comercio.

Programa en MQL4

Cabe señalar desde el principio que la programación en MQL4 está disponible para una persona común, aunque requiere atención y ciertos conocimientos.

Tal vez, algunos comerciantes esperan tener grandes dificultades en el estudio de la programación lo que significa que es difícil para ellos imaginar complicados procesos que se ejecutan en el interior de sus equipos. Afortunadamente, los desarrolladores del MQL4 han tratado de hacer ampliamente disponible para los usuarios. Una agradable particularidad de la creación de programas en MQL4 es que un programador no debe tener necesariamente conocimientos especiales sobre la interacción del cliente con un terminal de sistema operativo, el protocolo de red o las características de la estructura de un compilador.

El proceso de creación de programas para ejecutarlos en MQL4 es un simple amigable trabajo. Por ejemplo, un conductor no tiene que saber la estructura de un motor para conducir un coche, sólo necesita aprender los pedales y la dirección. Sin embargo, antes de conducir un coche en las calles, cada conductor tiene que someterse a la formación. Algo así es como debe hacerse con un programador: el aprendizaje de algunos sencillos principios de la creación de programas y después lentamente comienza a "conducir".

- Estructura de Programa
  Aunque hay muchos tipos de programas en MQL4, todos elos tienen características generales. Se puede decir, que una correcta estructura es la base de un código escrito correctamente. Por eso es necesario comprender los componentes de un programa.

- Funciones especiales
  Hay un montón de funciones en el MQL4. Estas funciones son llamadas funciones estándar. Sin embargo, hay varias funciones de gran importancia, que se llaman funciones especiales. Un programa no puede ejecutarse sin ellas. Cada una de estas funciones tiene su propia tarea.

- Ejecución de Programas
  Uno debe entender correctamente cómo opera un programa MQL4. No todas las partes de código se utilizan con la misma frecuencia. ¿Qué funciones se ejecutan en primera instancia, donde debe ser colocada la parte principal de un programa, ¿qué tipo de programa, debe utilizarse con este este u otro propósito?

- Ejemplos de aplicación
  Un nuevo lenguaje es mejor aprenderlo con ejemplos. Cómo escribir correctamente un programa simple? ¿Qué errores pueden ocurrir?

Estructura de Programa

En la primera seccion hemos aprendido algunas de las nociones básicas del lenguaje de programación MQL4. Ahora vamos a estudiar la forma está organizado un programa en general. Para resolver este problema vamos a estudiar su sistema estructural.

Como ya se ha mencionado anteriormente, el código programa principal escrito por un programador se pone dentro de funciones definidas por el usuario y funciones especiales. En la sección Funciones hemos discutido el concepto y las propiedades de built-in y funciones definidas por el usuario. En pocas palabras: una función definida por el usuario tiene de una descripción y llamada a la función se utiliza para iniciar su ejecución en un programa. Cualquier built-in o cualquier función definida por el usuario son ejecutados sólo después de que se las llama, en este caso la función es llamada para la ejecución de un programa.

Propiedades de funciones especiales se describen en detalle en la sección Funciones especiales. Aquí vamos a estudiar sólo la información principal sobre ellas. La función especial es una función llamada a ser ejecutadas por el Terminal de Usuario. A diferencia de las funciones comunes, las funciones especiales sólo tienen la descripción y su llamada no se especifica en un programa. Las funciones especiales son llamadas para su ejecución desde el Terminal de Usuario (también hay una posibilidad técnica de llamar a las funciones especiales desde un programa, pero vamos a considerar este método incorrecto y no lo vamoas discutir aquí). Cuando un programa se inicia para su ejecución en una ventana de un símbolo, el Terminal de Usuario pasa el control a una de las funciones especiales. Como resultado esta función se ejecuta.

La norma de programación en MQL4 es la siguiente:

Un código de un programa debe ser escrito dentro de funciones.

Es decir, las líneas de programa (operadores y llamadas a funciones) que se encuentran fuera de una función no pueden ser ejecutadas. En el intento de compilar un programa, MetaEditor mostrará el correspondiente mensaje de error y el archivo ejecutable *.exe no aparecerá como resultado de la compilación.

Vamos a considerar el plan funcional de un programa común, Asesor Experto:

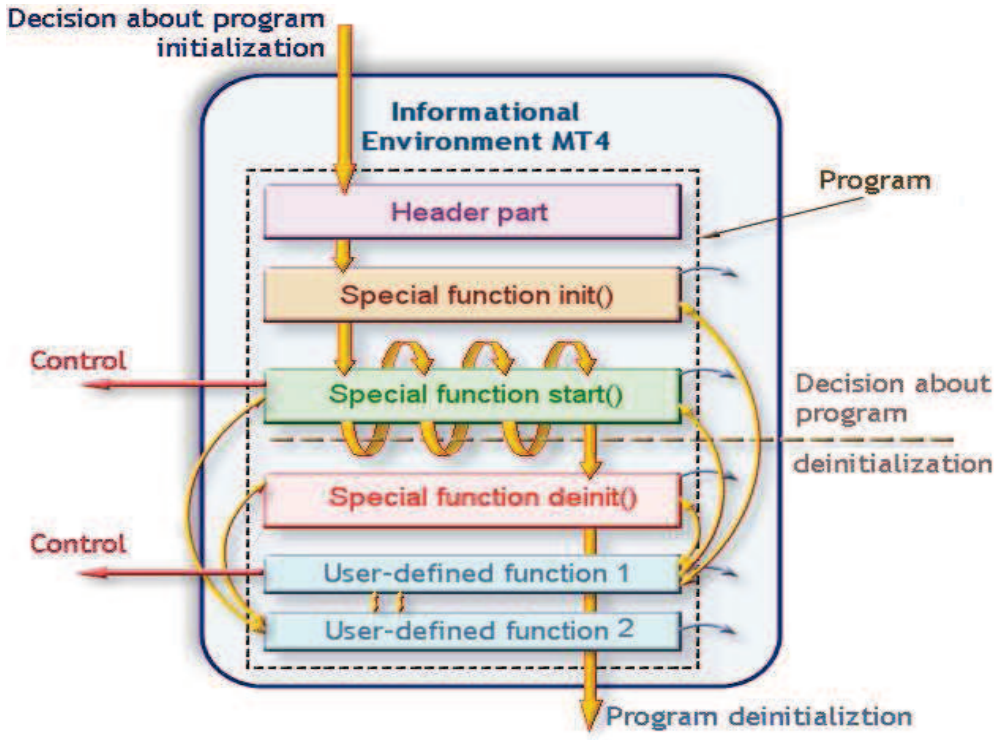


Fig. 31. Esquema funcional de un programa (Asesor Experto).

Los bloques mayores de un programa escrito en MQL4 son los siguientes:

1. Cabezera del programa.
2. Función especial init ().
3. Función especial start ().
4. Función especial Deinit ().
5. Funciones definidas por el usuario.

Además vamos a analizar sólo el contenido interior de estos bloques funcionales (partes integrales) de un programa, mientras que todos los objetos externos (por ejemplo, la información en la esfera del terminal cliente o hardware) se quedará fuera de nuestro ámbito de interés.

Información de Entorno de MetaTrader 4 Terminal de Usuario

La información de entorno del Terminal de Usuario MT4 no es un componente del programa. La información del entorno es un conjunto de parámetros disponibles para ser procesados por un programa. Por ejemplo, es una garantía de precios que ha llegado con un nuevo tick, el volumen acumulado en cada nuevo tick, la información acerca precios máximo y mínimo, de la historia de las barras, los parámetros que caracterizan a las condiciones comerciales ofrecidas por un dealing center, etc. Información de entorno es siempre guardada y en cada nuevo tick se actualiza por el Terminal de Usuario conectado con el servidor.

Estructura de Programa

Cabecera

La cabecera consta de varias líneas al comienzo de un programa (a partir de la primera línea) que contienen algunos escritos. Estas líneas contienen información general sobre el programa. Por ejemplo, esta parte incluye líneas de la declaración y la inicialización de variables globales (la necesidad de incluir tal o cual información en la cabecera se discutirá más adelante). El signo de la cabeza parte final puede ser la siguiente línea que contiene una descripción de la función (definidas por el usuario o función especial).

Funciones especiales

Por lo general, después de esta cabecera son descritas las funciones especial del programa. La descripción de la función especial se parece a la descripción habitual de una función definida por el usuario, pero las funciones especiales tienen nombres predefinidos: **init (), start () y deinit ()**. Las funciones especiales son un bloque de cálculos y estan en relación con el entorno de información del Terminal de Usuario y las funciones definidas por el usuario. Las funciones especiales se describen en detalle en la sección Funciones especiales.

Funciones definidas por el usuario

La descripción de funciones definidas por el usuario usualmente se da después de la descripción de las funciones especiales. El número de funciones definidas por el usuario en un programa no está limitado. El sistema contiene sólo dos funciones definidas por el usuario, pero un programa puede contener 10 ó 500, ó ninguna. Si no se utilizan funciones definidas el usuario en un programa, el programa será de una estructura simplificada: la cabeza y parte de la descripción de las funciones especiales.

Funciones estándar

Como se mencionó anteriormente, las funciones estándar solo pueden presentarse como una llamada a una función. Las funciones estándar, como cualquier otra función: las funciones especiales y funciones definidas por el usuario, tienen una descripción. Sin embargo, en la función estandar esta descripción no se da en el programa (es la razón por la que no está incluido en el esquema). La descripción de una función estandar está oculta, no esta visible para el programador y, por tanto, no se puede cambiar; a pesar de que está disponible para MetaEditor. Durante la compilación del programa, el MetaEditor creará un archivo ejecutable en el que todas las llamadas a funciones estándar se ejecutarán correctamente y con todo el rigor.

Acuerdo de las partes en un Programa

La cabecera debe estar ubicada en las primeras líneas. Las descripciones de funciones especiales y funciones definidas por el usuario no importan. La Fig.32 muestra una disposicón (habitual) de bloques funcionales, es decir; cabecera, funciones especiales y funciones definidas por el usuario. La Fig. 33 muestra otras variantes de estructura de programa. En todos los ejemplos la parte de la cabeza es lo primero, mientras que las funciones pueden ser descritas en un orden aleatorio.

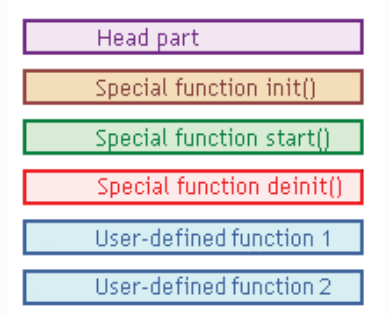


Fig. 32. Disposición habitual de bloques funcionales en un programa (recomendado).

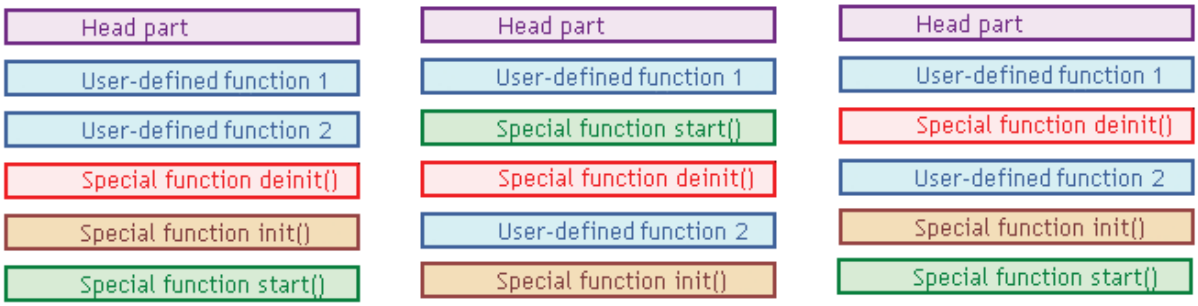


Fig. 33. Posibles formas de organización de bloques funcionales en un programa (orden aleatorio).

Por favor, tenga en cuenta:

Ninguna de las funciones puede ser descrita dentro de otra función. No está permitido el empleo en un programa descripciones de funciones situadas dentro de otra función.

A continuación se presentan ejemplos de incorrecta disposición de las descripciones de la función.

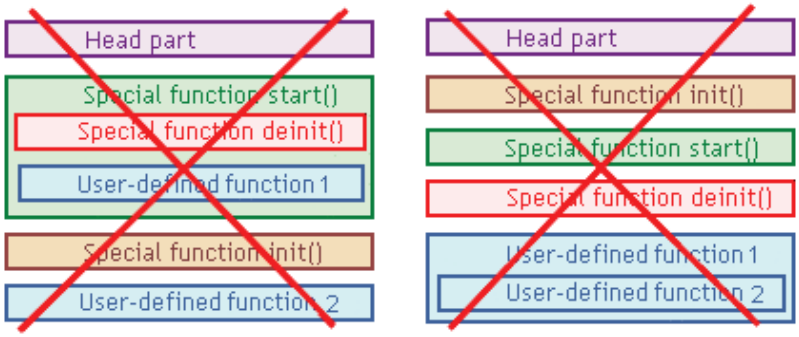


Fig. 34. Ejemplos de disposición incorrecta de las funciones en un programa.

Si un programador por error crea un programa donde la descripción de cualquiera de sus funciones se encuentra dentro de la descripción de otra función, en la etapa de compilación el MetaEditor mostrará un mensaje de error y no se creará archivo ejecutable para tal programa.

Secuencia de ejecución de código

Cabecera y funciones especiales

Desde el momento de iniciar la ejecución del programa en una ventana de un símbolo, parte de las líneas de programa de la cabeza se ejecutan.

Después, se realizan de los preparativos descritos en la cabecera y el Terminal de Usuario pasa el control a la función especial star () y se ejecuta la función (el control pasado se muestra en el esquema estructural en las grandes flechas amarillas). La función especial init () es llamada para la ejecución una sola vez al comienzo de la operación del programa. Por lo general, esta función contiene un código que debe ejecutarse sólo una vez antes de la operación principal del programa de inicio star (). Por ejemplo, cuando el init () es ejecutado, se inicializan algunas variables globales, objetos gráficos se muestran en una ventana gráfica, se muestran mensajes etc. Después de que todas en las líneas del programa start () se ejecutan, la función termina su ejecución y el control se devuelve al Terminal de Usuario.

El tiempo de operación del programa principal es el período de funcionamiento de la función start (). En determinadas condiciones (véase características de las funciones especiales en la sección Funciones especiales), incluyendo la recepción de nuevos tikcs por el Terminal de Usuario desde el servidor, el terminal de usuario pide la ejecución función especial start (). Esta función (al igual que otras funciones) se puede referir a la información de entorno del Terminal de Usuario, realizar los cálculos necesarios, abrir y cerrar posiciones, es decir, realizar cualquier acción permitida por MQL4. Por lo general, cuando la función especial start () es ejecutada, una solución producida se implementa como una medida de control (flecha roja). Este control puede ser implementado como una solicitud de comercio para abrir, cerrar o modificar una orden creada por el programa.

Después de que todo el código de la AE de la función especial start () es ejecutado, la función start () termina su operación y devuelve el control al terminal de usuario. El terminal tendrá el control durante algún tiempo no iniciando ninguna funcion especial. Una pausa aparece, durante la cual el programa no funcionará. Más tarde, cuando llegue un nuevo tick, el terminal de usuario pasará el control a la función especial start () una vez más, como resultado, la función será ejecutada y después cuando su ejecución termina, el control se devuelve al Terminal de Usuario. En siguiente tick la función start () será iniciada por el Terminal de Usuario una vez más.

El proceso de múltiples llamadas de la función especial start () por el Terminal de Usuario se repetirá mientras que el programa esté asociado a un gráfico y puede continuar durante semanas y meses. Durante todo este período un Asesor Experto puede llevar a cabo comercio automatizado, es decir, cumplir su principal misión. En el esquema el proceso de ejecución múltiple de la función start () se acredita por diversos flecha amarilla envolviendo la función especial start ().

Cuando un comerciante elimina un Asesor Experto de un gráfico, el Terminal de Usuario inicia una vez la función especial deinit (). La ejecución de esta función es necesaria para la correcta terminación de una operación de EA. Durante la operación un programa puede, por ejemplo, crear objetos gráficos y variables globales del Terminal de Usuario. Es por ello que el código de la función deinit () contiene líneas de programa, la ejecución de los cuales se traduce en la supresión de objetos innecesarios y de variables. Tras la ejecución de la función especial deinit () una vez más, se devuelve el control al Terminal de Usuario.

La ejecución de funciones especiales puede hacer referencia a la información del entorno (flechas delgadas de color azul en el esquema) y la llamada para la ejecución de funciones definidas por el usuario (flechas delgadas de color amarillo). Tenga en cuenta que las funciones especiales se ejecutan después de que ellas son llamadas por el Terminal de Usuario en el orden predefinido en función de las propiedades: en primer lugar init (), después llamada múltiples a start () y finalmente la función deinit(). Condiciones, en la que el terminal de usuario llama funciones especiales, se describen en la sección Funciones especiales.

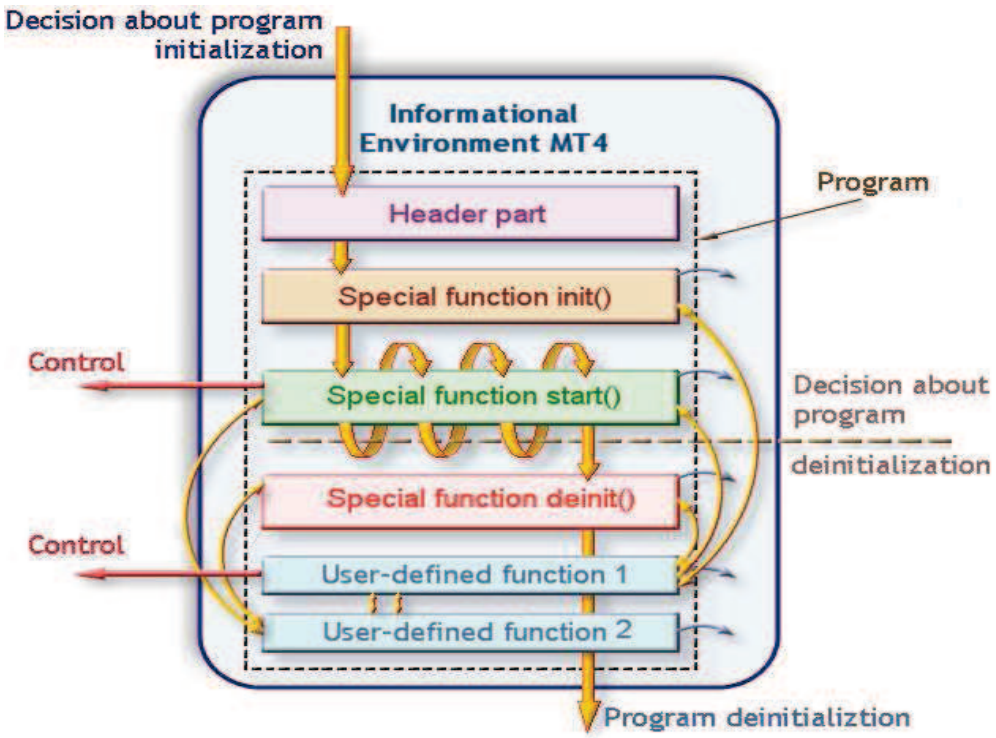


Fig. 31. Esquema funcional de un programa (Asesor Experto).

Funciones definidas por el usuario

Las funciones definidas por el usuario se ejecutan cuando la llamada está contenida en alguna función. En este caso, el control pasa a la función definida por el usuario y después de la ejecución de la función el control es devuelto al lugar de la llamada (flechas delgadas de color naranja en el esquema). La llamada a las funciones definidas por el usuario se pueden hacer no sólo dentro de la descripción de una función especial, sino también en la descripción de otras funciones definidas por el usuario. Una funcion definida por el usuario puede llamar a otras funciones definidas por el usuario. Esta forma de llamar a las funciones definidas por el usuario no solo esta permitido, sino que es un uso ampliamente utilizado en la programación.

La Funciones definidas por el usuario no son llamadas para ser ejecutadas por el Terminal de Usuario. Cualquier funciones definidas por el usuario se ejecuta dentro de la ejecución de una función especial que devuelve el control al Terminal de Usuario. Las funciones definidas por el usuario también pueden pedir (el uso) para la transformación de los valores de las variables de información de entorno del Terminal de Usuario (flechas delgadas de color azul en el esquema).

Si un programa contiene la descripción de una función definida por el usuario, pero no hay una invocación a esta función, esta función será excluida del programa en la etapa de compilación y no será utilizada en la operación del programa.



Nota: Las funciones especiales son llamadas para ser ejecutadas por el Terminal de Usuario. Las Funciones definidas por el usuario se ejecutan si se las llama desde funciones especiales o por funciones definidas por el usuario, pero nunca son llamadas por el Terminal de Usuario. El control de la acción, es decir, las órdenes de trading pueden formarse tanto en funciones especiales como en funciones definidas por el usuario.

Funciones especiales

Un rasgo distintivo de los programas destinados a la operación en el Terminal de Usuario MetaTrader 4 es su trabajo con una información constantemente actualizada en tiempo real. En este lenguaje MQL4, esta particularidad se refleja en la forma de tres funciones especiales: **init (), start () y deinit ().**

Las funciones especiales son funciones predefinidas con los nombres de **init (), start ()** y **deinit ()** que poseen propiedades especiales propias.

Propiedades de las funciones especiales

Propiedad común de funciones especiales

La principal propiedad de todas las funciones especiales es su ejecución en un programa bajo ciertas condiciones y que su funcionamiento no se inicia desde el programa. Las funciones especiales son llamadas para ser ejecutadas por el Terminal de Usuario. Si un programa contiene la descripción de una función especial, será llamada (y ejecutada), de conformidad con sus condiciones de llamada.



Las funciones especiales son llamados para ser ejecutadas por el Terminal de Usuario.

Propiedades de las funciones especiales

Función Especial init ().

La propiedad particular de la función especial de **init ()** es su ejecución en el programa de inicialización. Si un programa contiene la descripción de la función especial **init ()**, será llamada (y ejecutada) en el momento de iniciar el programa. Si no hay una función especial **init ()** en un programa, las acciones no se llevarán a cabo en el programa de inicio.

En los **Asesores Expertos** la función especial de **init ()** se llama (y ejecuta) después de que el Terminal de Usuario ha iniciado y cargado los datos históricos, después de cambiar el marco temporal del símbolo y/o gráfico, después de re-compilar el programa en el MetaEditor, después de cambiar parámetros de entrada de AE y la ventana de configuración, después de los cambios de la cuenta.

En los **scripts** de inicio de la función especial **init ()** también se llama (y ejecuta) inmediatamente después de que se llama y se ejecuta un gráfico.

En los **indicadores personalizados de usuario** función especial init () se llama (y ejecuta) inmediatamente después de empezar el Terminal de Usuario, después de cambiar el marco temporal del símbolo y/o período gráfico, después de re-compilar el programa en MetaEditor y después de cambiar parámetros de entrada en la ventana de configuración del indicador personal.

Función Especial start ().

Las propiedades especiales de la función start () difieren en función del tipo de programa que se ejecute.

En los **Asesores Expertos** la función especial start () se llama (y ejecuta) inmediatamente después de marcar un nuevo tick. Si un nuevo tick ha llegado durante la ejecución de la función especial start (), este tick no se tendrá en cuenta, es decir, la función especial start () no será llamada para su ejecución cuando este tick llega. Todas las cotizaciones recibidas durante la ejecución de la función especial start () se ignoran. El Inicio de la función especial start () para su ejecución se realiza por medio del Terminal de Usuario sólo a condición de que la operación del anterior período de sesión se haya completado, el control haya sido devuelto al Terminal de Usuario y la función especial start () este a la espera de un nuevo tick.

La posibilidad de llamar y ejecutar la función especial start () se ve influida por el estado del botón "Activar / desactivar del Asesor Experto. Si este botón se encuentra en el estado de desactivación AEs, del Terminal de Usuario no invocará la ejecución de la función especial start () con independencia de si las nuevas cotizaciones de llegan o no. No obstante, los cambios en el botón de estado desde Activado a Desactivado no finaliza la operación de la sesión actual de la función especial start ().

La función especial start () no es llamada por el Terminal de Usuario si la ventana de propiedades del AE está abierta. La ventana de propiedades de AE se puede abrir sólo cuando la función especial start () está a la espera de un nuevo tick. Esta ventana no puede abrirse durante la ejecución de la sesión del AEs la función especial start ().

En los **script** la función especiale start () se llama (y ejecuta) una vez, inmediatamente después de la inicialización del programa especial en la función init ().

En los **indicadores personales** función especial start () se llama (y ejecuta) inmediatamente después de marcar un nuevo tick, inmediatamente después de que se vincula a un gráfico, cuando se cambia el una tamaño de una garantía de la ventana, cuando se cambia de uno a otro instrumento, cuando se inicia el Terminal de Usuario (si durante el anterior período de sesiones, un indicador se asoció a una gráfica), después de cambiar un símbolo y el marco temporal actual de un gráfico con independencia del hecho de que si las nuevas cotizaciones llegan o no.

La **terminación** de la ejecución de la actual sesión star () se puede realizar en todos los tipos de programa cuando un programa se elimina de un gráfico, cuando el periodo del simbolo y/o gráfico se cambian, cuando se cambia una cuenta / gráfico es cerrado y como resultado el Terminal de Usuario la termina la operación. Si la función especial start () fue ejecutada durante un comando de apagado, el tiempo disponible del terminal para completar la ejecución de la función es de 2,5 segundos. Si después de apagado el comando, la función especial start () continúa sus operaciones durante más tiempo del plazo indicado, será forzado por el Terminal de Usuario a terminar.

Función Especial deinit ().

La función particular de la función especial deinit () es la ejecución de un programa de terminación (deinicialización). Si un programa contiene la descripción de la función especial deinit (), será llamada (y ejecutada) en un programa de apagado. Si un programa no contiene la función especial deinit (), no se llevarán a cabo acciones en programa de cierre.

La función especial deinit () también se le llama para ser ejecutadas por el Terminal de Usuario para terminal de cierre, cuando una ventana de un símbolo está cerrada, antes de cambiar una garantía y / o el periodo de un gráfico, en la re-compilación exitosa de un programa MetaEditor, al cambiar parámetros de entrada , Así como cuando una cuenta se ha cambiado.

En **Asesores Expertos y scripts** el programa se cierra con la necesaria llamada de la función especial deinit () puede ocurrir cuando vinculando un grafico a un nuevo programa del mismo tipo que sustituye al anterior.

En los **indicadores personales** función especial deinit () no se ejecuta cuando un nuevo indicador se asocia a un gráfico. Varios indicadores pueden operar en una ventana de un símbolo y esta la razón por que la vinculación de un nuevo indicador a un gráfico no se traduce en el cierre de otros indicadores con la función deinit ().

El tiempo de ejecución de deinit () está limitado a 2,5 segundos. Si el código de la función especial deinit () se ejecuta en mas tiempo el Terminal de Usuario fuerza la terminacion de la ejecución de la función especial deinit () y el funcionamiento del programa.

Requerimientos de las funciones especiales

La presencia de las funciones especiales init () y deinit () no son imprescindibles dentro programa, es decir, pueden estar ausentes. No importa el orden en el que estén descritas las funciones especiales en el programa. Las funciones especiales se pueden llamar desde cualquier parte del programa de conformidad con las reglas generales de llamadas a funciones.

Las funciones especiales pueden tener parámetros. Sin embargo, cuando estas funciones son llamadas por el Terminal de Usuario, estos parámetros no puden ser enviados desde el exterior y en este caso se utilizarán solo los valores por defecto.

Las funciones especiales init () y deinit () deben terminar su funcionamiento con la maxima rapidez y en ningún caso ejecutarse dentro de un recorrido ciclico pretendiendo hacer todas las operaciones antes de llamar a la función start ().

Orden de uso de las funciones especiales

Los desarrolladores han presentado a los programadores una herramienta muy práctica: cuando empieza un programa, init () se ejecuta en primer lugar. Después, una vez que se realiza el trabajo principal con la ayuda de la función start (), y cuando un usuario termina ha terminado su trabajo, la función deinit () se pondrá en marcha antes de que se cierre el programa.

El código principal del programa debe estar contenido en la función start (). Todos los operadores, built-in, las llamadas a las funciones personalizadas y todos los cálculos necesarios se debe realizar dentro de esta función. Al mismo tiempo, hay que entender correctamente el papel de las funciones personalizadas. La descripción de funciones personalizadas se encuentran en el código de un programa fuera de la descripción de las funciones especiales, pero si una función definida por el usuario es llamada para su ejecución, la función especial no finaliza su funcionamiento. Esto significa que el control pasa durante algún tiempo a la función del usuario, pero la propia función de usuario actúa en el marco de la función especial que la ha llamado. Así que, en el proceso de ejecución de un programa especial se opera siempre de conformidad con sus propiedades particulares, y la función de usuario se ejecuta cuando es llamada desde la funcion especial.

Si hay alguna función especial que un programador no vaya a utilizar, puede eludir su uso en el programa. En tal caso, el Terminal de Usuario no la llamará. Es absolutamente normal que un programa contenga las tres funciones especiales. Un programa que no tiene init () o deinit () o ambas funciones también se considera normal.

Si un programa no contiene ninguna de las tres funciones especiales, este programa no se ejecutará. El Terminal de Usuario necesita para su ejecución al menos una función especial de conformidad con sus propiedades. Las Funciones definidas por el usuario no son llamadas por el Terminal de Usuario. Es por este motivo que si un programa no contiene funciones especiales (y sólo contiene funciones definidas por el usuario), nunca serán llamadas para su ejecución.

No se recomienda llamar a la función start () desde la función especial de inicio () o realizar operaciones de comercio desde init (), porque durante la inicialización valores de los parámetros de la información del entorno puede que no esten listas (información sobre gráficas, precios de mercado, etc.)

Secciones de Ejecución de Programas y Ejemplos de aplicación contiene varios ejemplos prácticos que ayudan a ver algunas propiedades de funciones especiales.

Ejecución de Programas

Las habilidades de programación se desarrollan mejor si un programador tiene un pequeño programa operativo a su disposición. Para entender todo el programa, es necesario examinar a fondo todos sus componentes y localizar su funcionamiento paso a paso. Tenga en cuenta, las propiedades de las función especial de los distintos programas de aplicación (Asesores Expertos, scripts e indicadores) son diferentes. Ahora vamos a analizar cómo opera un Asesor Experto.



Example of a simple Expert Advisor (simple.mq4)

```
//+------------------------------------------------------------------+
// simple.mq4
// To be used as an example in MQL4 book.
//+------------------------------------------------------------------+
int  Contador=0;                              // Variable Global
//+------------------------------------------------------------------+
int init()                                 // Función Especial init()
  {
  Alert ("La funcion init () ha comenzado ");// Alert
  return;                                   // Exit init()
  }
//+------------------------------------------------------------------+
int start()                                // Función Especial start()
  {
  double  Precio = Bid;                          // Variable Local
   Contador++;                                   // Contador de ticks
  Alert ("Nuevo tick ",  Contador,"   Precio = ",  Precio);// Alert
  return;                                   // Exit start()
  }
//+------------------------------------------------------------------+
int deinit()                               // Función Especial deinit()
  {
  Alert ("La función deinit() ha comenzado la salida");   // Alert
  return;                                   // Exit deinit()
  }
//+------------------------------------------------------------------+
```

De conformidad con las reglas de ejecución de los programas (véase el Programa de Estructura y Funciones especiales) este Asesor Experto trabajará la siguiente manera:

1. En el momento en que un programa se vincula a un gráfico, el Terminal de Usuario pasa el control al programa y, como resultado, el programa empezará su ejecución. La ejecución del programa comienza a partir de la cabecera. La cabecera solo contiene una línea:

```
int  Contador=0;                    // Variable Global
```

En esta línea la variable global Contador se inicializa a cero. (Las variables Locales y Globales se analizan en detalle en la sección Tipos de variables. Cabe señalar aquí, que el algoritmo utilizado en este programa requiere la declaración de una variable global como el Count, por eso no puede ser declarada dentro de una función y es preciso declararla fuera de la descripción de las funciones, es decir, en la cabecera. Como resultado de esto, el valor de la variable global Cont estará disponible a partir de cualquier programa.

Nota del traductor: El llamar a la varible Contador así, es puramente arbitrario (del traductor) y no venian en el texto original. Las letras VG, pueden ayudar a recordar que la variables es una Variable Global, especialmente si estamos trabajando con un programa grande con muchas variables.

3. Después de la ejecución de la parte de la cabeza del programa, la función especial init () se pondrá en marcha para su ejecución. Observe que esta llamada a la función no está contenida en un código del programa. Cuando se ejecuta un AE por que se ha vinculado a un gráfico, es una propiedad el comienzo de la ejecución de la función init (). El Terminal de Usuario llama init () para la ejecución sólo porque el código de programa contiene una descripción del mismo. El modo en que el programa analiza la descripción de la función especial init () es el siguiente:

```
int init()                          // Función Especial init()
  {
  Alert ("La funcción init() ha comenzado"); // Alerta
  return;                           // Salir de init()
  }
```

El cuerpo de la función contiene sólo dos operadores.

2.1 Función de alerta () que muestra la siguiente ventana de alerta:

```
La función init() ha comenzado
```

2.2 El operador return que termina la operación especial de la función init () y que no devuelve ningún valor.

Como resultado de la ejecución de init () se escribe un alerta. En realidad este programas contiene un algoritmo muy raro, porque el uso de esta init () sirve de muy poco. Realmente, no tiene sentido utilizar una función que solo informa a un comerciante de que la función está siendo ejecutada. En este caso, el algoritmo se utiliza sólo para la visualización de la ejecución de init (). Hay que prestar atención a que el función especial de inicio init () se ejecuta en un programa solo una vez. La ejecución de la función tiene lugar al comienzo del programa, después de que la operación de la parte de la cabeza ha sido procesada. Cuando el operador de return se ejecuta en la función especial init (), el programa devuelve el control al Terminal de Usuario.

4. El Terminal de Usuario detecta la descripción de la función especial start () en el programa:

```
int start()                          // Función Especial start()
  {
  double  Precio = Bid;                    // Variable Local
VG_Contador++;                          // Contador de ticks
Alert ("Nuevo tick ", VG_Contador,"   Precio = ",  Precio);// Alert
    return;                             // Salir de start()
```

31. El control está en manos de la Terminal de Usuario. El Terminal de Usuario espera un nuevo tick y no comienza la ejecución del programa hasta que no llegue ese nuevo tick. Esto significa que desde hace algún tiempo el programa no está funcionando, es decir, no se realiza ninguna acción en ella. Una pausa aparece. La necesidad de esperar a un tick es una propiedad de la función start () y no hay forma de que un programa pueda influir en esta propiedad (como por ejemplo desactivarla). El programa espera y mantiene el control hasta que aparezca un nuevo tick. Cuando llega un nuevo tick , el Terminal de Usuario pasa el control al programa, es decir, a la función especial start () (en este caso, de acuerdo con la propiedad del la función start () del AE). Como resultado de ello se inicia su ejecución.

32 (1). En la línea

```
double  Precio = Bid;                    // Variable Local
```

las siguientes acciones se llevan a cabo:

32,1 (1). Declaración de una variable local Precio (véase tipos de variables). El valor de esta variable local estará disponible en cualquier parte de la función especial start ().

32,2 (1). Ejecución del operador de asignación. El valor del actual precio de oferta Bid se asigna a la variable Precio. Un nuevo valor del precio aparece cada vez que venga un nuevo tick (por ejemplo, el primer tick una nueva cotizacion de precios puede ser igual a 1,2744).

33 (1). A continuación, la siguiente línea se ejecuta:

```
 Contador++;                          // Contador de ticks
```

No este poco habitual registro es plenamente analologa a Contador = Contador +1;

En el momento de pasar el control a esta línea, el valor de la variable Contador es igual a cero. Como resultado de la ejecución de Contador ++, el valor del Contador se incrementará en uno. Así que, en el momento de pasar el control a la línea siguiente, el valor de Contador será igual a 1.

34 (1). La siguiente línea contiene llamada a la función Alerta ():

```
Alert ("Nuevo tick ", Contador,"   Precio = ",  Precio); // Alerta
```

La función escribirá todas las constantes y variables enumeradas entre paréntesis.

En la primera ejecución de la función start () el programa va a escribir un nuevo tick, entonces se refieren a las variables para obtener su valor (en la primera ejecución este valor es 1), escribe este valor, luego escribirá Precio = y se refieren a la variable Precio para obtener su valor y escribirlo (en nuestro ejemplo es 1.2744).

Como consecuencia de ello se escribirá la siguiente línea:

```
Nuevo tick numero 1, Precio = 1.2744
```

35 (1). Operador

```
return;                          // Salir de start()
```

Termina la operación especial de la función start ().

36. El control es devuelto al Terminal de Usuario (hasta que llegue un nuevo tick).

Así es como ejecuta la función start () de un Asesor Experto. Cuando la ejecución termina, la función especial start () devuelve el control al Terminal de Usuario y cuando llegue un nuevo tick, el Terminal haraá comenzar su funcionamiento una vez más. Este proceso (iniciar la ejecución de la función start () y devolver el control a la Terminal de Usuario) se puede repetir durante un largo tiempo (varios días o semanas). Durante todo este tiempo la función especial start () se funcionará de vez en cuando. Dependiendo de los parámetros del entorno (nuevos precios, tiempo, condiciones de comercio, etc) pueden realizar diferentes acciones (como la apertura o la modificación de las órdenes) en la función especial start ().

37. Desde el momento que se recibe un nuevo tick, se repiten las acciones de los puntos 32-36. Sin embargo, aunque la secuencia de ejecución de los operadores se repite, el valor que se obtiene de las variables es nuevo cada vez. Vamos a ver las diferencias entre la primera y la segunda la ejecución de la función especial start ().

32 (2). En la línea

```
double  Precio = Bid;                    // Variable Local
```

se llevan a cabo las siguientes acciones:

32.1 (2). Declaración de la variable local Precio (sin cambios).

32,2 (2). Ejecución del operador de asignación. El valor del actual precio de oferta bid de se asigna a la variable Precio (aparece el valor de un precio nuevo cada vez que llega una nueva cotización, por ejemplo, los precios del segundo tick del símbolo será igual a 1,2745) (hay cambios).

33 (2). A continuación, la siguiente línea se llevará a cabo:

```
Contador++;                          // Contador de ticks
```

Por el momento antes de pasar el control a esta línea, el valor de la variable Contador (después de la primera ejecución de la función start ()) es igual a 1. Como resultado de la ejecución de Contador++ el valor de Contador se incrementará en uno. Así, en la segunda ejecución la variable Contador será igual a 2 (modificado).

34 (2). Alerta ():

```
Alert ("Nuevo tick ", VG_Contador,"   Precio = ",  Precio); // Alerta
```

Escribe todas las constantes y variables (sus nuevos valores) que se enumeran entre paréntesis.

En la segunda ejecución de start () el programa escribe un nuevo tick, entonces se refieren a la variable VG_Contador para obtener su valor (en la segunda ejecución es igual a 2), a continuación escribe este valor, escribe Precio = y se refieren al valor de la variable para obtener el precio y escribirlo que en nuestro ejemplo es 1.2745 (cambiado).

Como consecuencia de ello se escribirá la siguiente:

```
Nuevo tick numero 2, Precio = 1.2745
```

35 (2). Operador

```
return;                          // Salir de start()
```

Termina la operación start () (sin cambios).

36 (2). El control se devuelve al Terminal de Usuario en espera de un nuevo tick.

37 (2). Entonces se repite de nuevo. En la tercera start () la ejecución variables se obtienen nuevos valores y será escrita por la función de alerta (), es decir, el programa repite los puntos 32-36 (3). Y luego una y otra vez: 32 - 36 (4), 32 - 36 (5),..(6) .. (7) .. (8) ... Si el usuario no toma ninguna acción, este proceso se repetirá indefinidamente. Como resultado del funcionamiento de start () este programa veremos la historia de los cambio de precio.

Los próximos eventos ocurrirán sólo cuando un usuario decide terminar el programa fuerza quitar el programa de un gráfico en forma manual.

4. El Terminal de Usuario pasa el control a la función especial deinit () (de acuerdo con sus propiedades).

```
int deinit()                          // Función Especial deinit()
  {
  Alert ("La función deinit() ha comenzado la salida");   // Alert
  return;                             // Exit deinit()
  }
```

Sólo hay dos operadores en el cuerpo de la función.

41. Alerta () escribe:

```
Función deinit () ha comenzado la salida
```

42. Operador return termina la operación de deinit ().

La ejecución función deinit () se inicia por el Terminal de Usuario sólo una vez, después de que la alerta de arriba se aparezca en Alert () la ventana y el programa se eliminarán del gráfico.

5. Aquí termina la historia de ejecución de un Asesor Experto.

Al vincular este programa de ejemplo cualquier gráfico e iniciarlo el programa operativo mostrará una ventana que contiene todas las alertas generadas por la función de alerta (). Por el contenido de las descripciones es fácil de entender la función especial que está conectada con esta o aquella entrada.

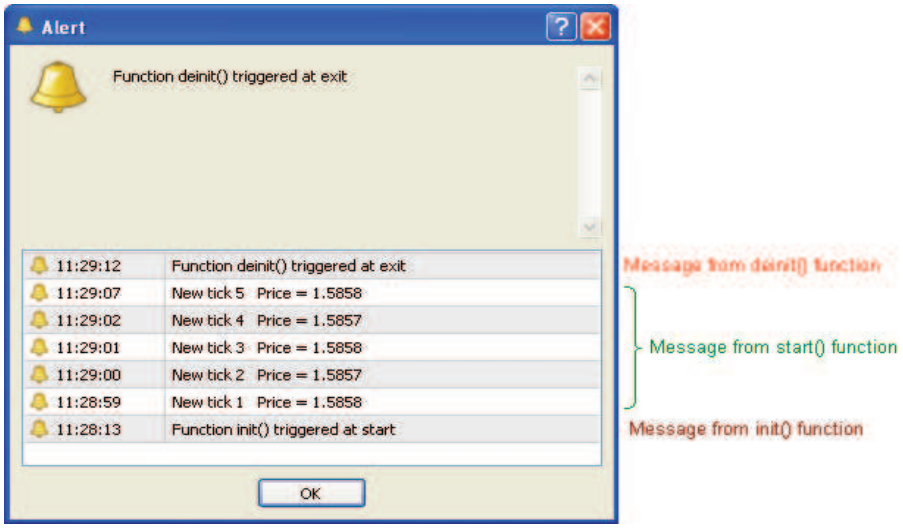


Fig. 35. Resultados del funcionamiento del programa simple.mq4.

A partir de este ejemplo se puede ver fácilmente que un programa se ejecuta de acuerdo con las propiedades de las funciones especiales descritas en Funciones especiales. Terminar el programa e iniciarlo de nuevo. Después de hacer esto varias veces, obtendrá experiencia en la utilización de su primer programa. Se trabajará tanto ahora como la próxima vez. Otros programas que usted escriba también serán construidos de acuerdo con la estructura descrita y para el inicio de su ejecución, tendrá que vincularlo a un gráfico.

Trate de entender todos los conceptos y reglas y el proceso de creación de programas en MQL4 será fácil y agradable.

.

Ejemplos de aplicación

En la sección anterior analizamos un ejemplo de ejecución de funciones especiales de ejecución en un simple Asesor Experto simple.mq4. Para una mejor práctica vamos a analizar algunas modificaciones más de este programa.



Ejemplo de una correcta estructura programática

Por regla general, las descripciones de funciones se indican en el mismo orden en que sean llamados para su ejecución por parte del Terminal de Usuario, es decir, primero va la descripción de la función especial init (), start () y el último es deinit (). Sin embargo, funciones especiales son llamados para ser ejecutadas por el Terminal de Usuario de conformidad con sus propia propiedades particulares,que es la razón por lo que no importa la ubicación de la descripción del programa. Vamos a cambiar el orden de las descripciones y ver el resultado (Asesor Experto possible.mq4).

```
//+------------------------------------------------------------------+
// possible.mq4
// To be used as an example in MQL4 book.
//+------------------------------------------------------------------+
int Contador=0;                          // Variable Global
//+------------------------------------------------------------------+
int start()                              // Función especial start()
  {
  double Precio = Bid;                   // Variable Local
  Contador ++;
  Alert("Nuevo tick ", VG_Contador,"   Precio = ",Precio);// Alerta
  return;                                // salida de start()
  }
//+------------------------------------------------------------------+
int init()                               // Función especial init()
  {
  Alert ("La funcción init() ha comenzado"); // Alerta
  return;                                // Salir de init()
  }
//+------------------------------------------------------------------+
int deinit()                             // Special funct. deinit()
  {
  Alert ("Function deinit() a desencadenado la salida");// Alert
  return;                                // Exit deinit()
  }
//+------------------------------------------------------------------+
```

A partir de este Asesor Experto se verá que la ejecución de secuencias de funciones especiales en un programa no depende del orden de las descripciones en el programa. Usted puede cambiar las posiciones de las descripciones en función del código fuente y el resultado será el mismo que en la ejecución del Experto Asesor simple.mq4.

Ejemplos de una incorrecta estructura programática

Pero el programa se comportará de forma diferente si cambiamos la posición de la parte de la cabeza. En nuestro ejemplo, vamos a indicar start () antes que la cabeza (de expertos Advisri incorrect.mq4):

```
//+------------------------------------------------------------------+
// possible.mq4
// To be used as an example in MQL4 book.
//+------------------------------------------------------------------+
int start()                              // Función especial start()
  {
  double Precio = Bid;                      // Variable Local
  Contador ++;
  Alert("Nuevo tick ", VG_Contador,"   Precio = ",Precio);// Alerta
  return;                                // salida de start()
  }
//+------------------------------------------------------------------+
int Contador =0;                           // Variable Global
//+------------------------------------------------------------------+
int init()                               // Función especial init()
  {
  Alert ("La funcción init() ha comenzado"); // Alerta
  return;                                // Salir de init()
  }
//+------------------------------------------------------------------+
int deinit()                             // Special funct. deinit()
  {
  Alert ("Function deinit() a desencadenado la salida");// Alert
  return;                                // Exit deinit()
  }
//+------------------------------------------------------------------+
```

Al intentar compilar este Asesor Experto, MetaEditor mostrará un mensaje de error:

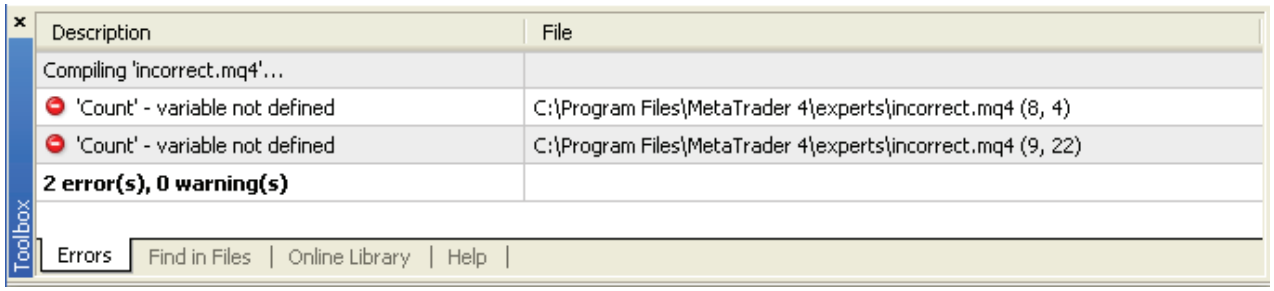


Fig. 36. Mensaje de error durante de la compilación del programa incorrect.mq4.

En este caso la línea

```
int Contador=0;                          // Variable Global
```

se escribe fuera de todas las funciones, pero no está al comienzo de un programa sino en algún lugar en medio del código.

El momento decisivo en la estructura del programa es que la declaración de la variable global Contador se hace después de declarar la función (en nuestro caso - función especial start ()). En esta sección no vamos a discutir los detalles de la utilización de variables globales, los tipos de variables y sus reglas de uso se describen en la sección Variables. Cabe señalar aquí que cualquier variable global debe ser declarada antes (al principio del texto) que la primera llamada a cualquier función (en nuestro caso es en la función start ()). En el programa que se analiza esta norma fue violada y el compilador mostrará un mensaje de error.



Ejemplo de utilización de una función definida por el usuario

Ahora vamos a ver cómo se comporta el programa en relación con funciones personalizadas. Con este fin vamos a actualizar el código descrito en el ejemplo de un simple Asesor Experto simple.mq4 y luego lo vamos a analizar. Un programa con una función definida por el usuario tendrá este aspecto (Asesor Experto userfunction.mq4):

```
//+------------------------------------------------------------------+
// userfunction.mq4
// Para ser usado como un ejemplo del libro de MQL4.
//+------------------------------------------------------------------+
int Contador =0;                                  // Variable Global
//+------------------------------------------------------------------+
int init()                                     // Función Especial init()
  {
  Alert ("La funcion init () ha comenzado ");// Alert
  return;                                      // Exit init()
  }
//+------------------------------------------------------------------+
int start()                                    // Función Especial start()
  {
  double  Precio = Bid;                             // Variable Local
  Mi_Funcion();                               // Llamada a la funcion personal
   Alert ("Nuevo tick ", Contador,"   Precio = ",  Precio);// Alert
  return;                                      // Exit start()
  }
//+------------------------------------------------------------------+
int deinit()                                   // Función Especial deinit()
  {
  Alert ("La función deinit() ha comenzado la salida");   // Alert
  return;                                      // Exit deinit()
  }
//+------------------------------------------------------------------+
//+------------------------------------------------------------------+
int My_Function()                                // Descripción de la función definida por el usuario
  {
   Contador++;                               // Contador de ticks
  }
```

En primer lugar veamos lo que ha cambiado y lo que se ha mantenido sin cambios.

Partes sin modificar:

1. La parte de cabecera no se ha modificado.

```
//-------------------------------------------------------------------
// userfunction.mq4
// Para ser usado como un ejemplo del libro de MQL4.
//-------------------------------------------------------------------
int Contador=0;                              // Variable Global
//-------------------------------------------------------------------
```

2. Función especial init () no se ha modificado.

```
int init()                                   // Función Especial init()
  {
  Alert ("La funcion init () ha comenzado ");// Alert
  return;                                    // Exit init()
  }
```

3. Función especial deinit () no se ha modificado.

```
int deinit()                                 // Función Especial deinit()
  {
  Alert ("La función deinit() ha comenzado la salida");   // Alert
  return;                                    // Exit deinit()
  }
```

Cambios:

1. Se ha añadido: la función definida por el usuario. Mi_Funcion ()

```
int My_Function()                            // Descripción de la función definida por el usuario
  {
   Contador++;                               // Contador de ticks
  }
```

3. El código especial de la función start () también ha cambiado: ahora contiene la llamada a función definida por el usuario y ya no existe la linea para cálculo de la variable Contador .

```
  int start()                                // Función Especial start()
  {
  double  Precio = Bid;                      // Variable Local
  Mi_Funcion ();                             // Llamada a la funcion personal
   Alert ("Nuevo tick ", Contador,"   Precio = ",  Precio);// Alerta
  return;                                    // Exit start()
  }
```

En la sección de Ejecución de Programas analizamos la ejecución de la orden de init () y deinit (). En este ejemplo, estas funciones se llevarán a cabo de la misma manera, por lo que no vamos a insistir en su funcionamiento. Vamos a analizar la ejecución de la función especial start () y la función definida por el usuario Mi_Funcion (). La descripción de la función definida por el usuario se encuentra fuera de todas las funciones especiales que es como debe ser. La llamada a la Función definida por el usuario se indica en start () el código, que también es correcto.

Después de que init () ha terminado su ejecución, el programa se ejecutará de manera:

31. The función especial start () está a la espera de ser iniciada por el Terminal de Usuario. Cuando llega un nuevo tick, el terminal inicia esta función para su ejecución. Como resultado de esto se llevan a cabo las siguientes acciones:

32 (1). En la línea

```
double  Precio = Bid;              // Variable Local
```

las mismas acciones se llevan a cabo:

32,1 (1). Precio variable local se inicializa (ver tipos de variables). El valor de esta variable local estará disponible en cualquier parte de la función especial start ().

32,2 (1). Se ejecuta el operador de asignación. El último precio de oferta disponible se asignará a la variable precio (por ejemplo en el primer tick es igual a 1,2744).

33 (1). Después viene Mi_Funcion();

```
Mi_Funcion ();                     // Llamada a la funcion personal
```

Esta línea se lleva a cabo dentro de start (). El resultado de la aplicación de esta parte del código (la llamada a función de usuario) está en el paso del control al cuerpo (descripción) de la función que más tarde regresará al lugar de la llamada.

34 (1). Sólo hay un operador en la descripción de la función de usuario:

```
   Contador++;                     // Contador de ticks
```

En la primera llamada a la función definida por el usuario la variable es igual a cero. Como resultado de la ejecución del operador Contador++; el valor de Contador se incrementará por uno. Después de haber ejecutado este operador (el único y el último) la función de usuario termina su operación y devuelve el control al lugar desde donde ha sido llamado.

Cabe señalar aquí que las funciones definidas por el usuario solo pueden ser llamadas desde funciones especiales (o desde otras funciones definidas por el usuario que a su vez han sido llamadas desde funciones especiales). Esa es la razón por la que la siguiente declaración es correcta: en cualquier momento una de las funciones especiales estará en funcionamiento (o start () está a la espera de un nuevo tick para ser iniciada por el Terminal de Usuario) y las funciones personalizadas serán ejecutadas solo en el interior de las funciones especiales.

En este caso se devuelve el control a la función especial start () que se está ejecutando, es decir, a la línea siguiente que llama al operador de función:

35 (1). Esta línea contiene Alerta ():

```
Alert ("Nuevo tick ", Contador," Precio = ", Precio); // Alerta
```

La función de alerta () mostrará en una ventana todas las constantes y variables enumeradas entre paréntesis:

```
New tick 1 Price = 1.2744
```

36 (1). Operador

```
return;                            // Exit start()
```

termina start ().

37. El control se pasa al Terminal de Usuario en espera de una nuevo tick.

Las nuevas ejecuciones start (), se obtienen los nuevos valores de las variables y serán mostrados los mensajes de alerta (), es decir, el programa llevará a cabo los puntos 32 - 36. En cada inicio () de la ejecución (en cada tick) se llamará a la función de usuario Mi_Funcion () y esta función y se ejecutará. La ejecución de start () continuará hasta que un usuario decida poner fin a la operación del programa. En este caso, la función especial deinit () se ejecutará y el programa dejará de operar.

El programa userfunction.mq4 iniciado para su ejecución mostrará una ventana que contiene los mensajes de alerta (). Tenga en cuenta, el resultado de la operación del programa será el mismo que el resultado de un simple Asesor Experto simple.mq4 operación. Es evidente que la estructura de userfunction.mq4 se compone de conformidad con el orden usual de ubicación de bloques funcionales.

Si se utiliza otro orden aceptable, el resultado será el mismo.

Operadores

Esta sección se refiere a las normas de formato y ejecución de los operadores utilizados en MQL4. Cada sección incluye ejemplos sencillos que muestran el modo de ejecución de los operadores. Para asimilar el material en su totalidad, se recomienda a compilar y poner en marcha la ejecución de los programas de todos los ejemplos. Esto también le ayudará a consolidar habilidades en el trabajo con MetaEditor.

- Operador de asignación.
  Este es el operador es el más simple e intuitivo. Todos conocemos la operación de asignación de la asignatura de matemáticas: El nombre de una variable se situa a la izquierda del signo de igualdad, el valor que se le da está a la derecha del signo de igualdad.

- Operador condicional "if-else".
  A menudo es necesario para orientar el programa en una u otra dirección en relación con determinadas condiciones. En estos casos, el operador "if-else" es muy útil.

- Ciclo del Operador "while".
  El tratamiento de una gran cantidad de datos de tipo de array generalmente requiere múltiples repeticiones de la misma operación. Se puede organizar un bucle de este tipo de operaciones con el operador de ciclo "while". Cada una ejecución de las operaciones en un ciclo se llama *iteración.*

- Operador de ciclo "for".
  El operador "for" también es un operador de ciclo . Sin embargo, a diferencia con el operador "while" es que para la ejecución de iteraciones, generalmente contienen dentro de sí mismo el valor inicial y el valor final de una determinada condición.

- Operador de "break".
  Si quiere interrumpir el trabajo de un operador de ciclo sin tener que ejecutar el resto de las iteraciones, se necesita el operador "break". Se utiliza sólo en los operadores "While", "for" y "Switch", y en ningún otro.

- Operador "Continue".
  Un operador de gran utilidad es el operador que salta a la siguiente iteración dentro de un ciclo. Permite que el programa salte todos los operadores restantes en la iteración actual y pase a la siguiente.

- Operador de "switch".
  Este operador es un "conmutador" que permite al programa elegir una de entre varias alternativas posibles. Para cada alternativa se describe su constante predefinida que es el caso de esa alternativa.

- Función de Call.
  Entendemos como función call que la función que es llamada ejecutará algunas operaciones. La función puede devolver un valor del tipo predefinido. La cantidad de parámetros transferidos en la función no podrá ser superior a 64.

- Descripción y función del operador "return".
  Antes de llamar a una función definida por el usuario, usted debe describirla primero. La descripción de la función debe especificar su tipo, el nombre y la lista de parámetros. Además, en el cuerpo de la función estará los operadores ejecutables. El trabajo de una función se termina con la ejecución del operador "return".

Operador de asignación

El operador de asignación es el operador más simple y más frecuentemente usado.

Formato del operador de asignación

Operador de asignación representa un registro que contiene el carácter "=" (signo de igualdad). A la izquierda de este signo de igualdad se especifica el nombre de una variable, a la derecha de ella damos una expresión. El operador de asignación se termina con ";" (punto y coma).

```
Variable = Expresión;     // operador de asignación
```

Se puede distinguir el operador de asignación de otras líneas en el texto del programa por la presencia del signo de igualdad. Puede especificar una expresión como: una constante, una variable, una llamada a una función, o una expresión como tal.

Ejecución del operador de asignación



Calcula el valor de la expresión a la derecha de la igualdad y asignar el valor obtenido para la variable especificada a la izquierda del signo de igualdad.

El operador de asignación, al igual que cualquier otro operador, es ejecutable. Esto significa que el registro que compone el operador de asignación se realiza de acuerdo a una regla. Cuando se ejecuta el operador, se calcula el valor de la parte derecha y, a continuación, se asigna a la variable el valor que esta a la izquierda del signo de igualdad. Como resultado de la ejecución del operador de asignación, la variable en la parte izquierda siempre toma un nuevo valor, este valor puede ser distinto o el mismo que el anterior valor de la variable. La expresión en la parte derecha del operador de asignación se calcula de acuerdo con el orden de las operaciones (véase Operaciones y las expresiones).

Ejemplos de operadores de asignación

En un operador de asignación, se permite declarar el tipo de una variable a la izquierda de la igualdad de signo:

```
int In = 3;              // The constant value is assigned to variable In
double Do = 2.0;         // The constant value is assigned to variable Do
bool Bo = true;          // The constant value is assigned to variable Bo
color Co = 0x008000;     // The constant value is assigned to variable Co
string St = "sss";       // The constant value is assigned to variable St
datetime Da= D'01.01.2004';// The constant value is assigned to variable Da
```

Las variables declaradas previamente se utilizan en un operador de asignación, sin especificar su tipo.

```
In = 7;              // The constant value is assigned to variable In
Do = 23.5;           // The constant value is assigned to variable Do
Bo = 0;              // The constant value is assigned to variable Bo
```

En un operador de asignación no se permite que el tipo de una variable sea declarada en la parte derecha del signo de igualdad:

```
In = int In_2;          // Variable type may not be declared in the right part
Do = double Do_2;       // Variable type may not be declared in the right part
```

En un operador de asignación, no se le permite que el tipo de una variable sea declarado más de una vez.

```
int In;                 // Declaration of the type of variable In
int In = In_2;          // The repeated declaration of the type of the variable (In) is not allowed
```

Ejemplos del uso de funciones definidas por el usuario y funciones estándar en la parte derecha:

```
In = My_Function ();     // The value of user-defined function is assigned to variable In
Do = Gipo(Do1,Do1);      // The value of user-defined function is assigned to variable Do
Bo = IsConnected();      // The value of standard function is assigned to variable Bo
St = ObjectName(0);      // The value of standard function is assigned to variable St
Da = TimeCurrent();      // The value of standard function is assigned to variable Da
```

Ejemplo de utilización de expresiones en la parte derecha:

```
In = (My_Function ()+In2)/2;       // The variable In is assigned
                                   // ..with the value of expression
Do = MathAbs(Do1+Gipo(Do2,5)+2.5); // The variable Do is assigned
                                   // ..with the value of expression
```

En los cálculos del operador de asignación, son aplicables las normas del typecasting (véase el Typecasting).

Ejemplos de Asignación de operadores en forma corta

En MQL4 tambien se utiliza una forma breve de componer los operadores de asignación. Es la forma de asignación de los operadores cuando usamos la asignación a otras operaciones distintas de operación de asignación **"="** (signo de igualdad) (véase Operaciones y las expresiones). La forma corta los operadores estan sometidos a las mismas normas y limitaciones. La forma corta de la asignación operadores se utiliza en el código para una mejor visualización. Un programador puede, a su opción, utilizar una u otra forma de la asignación operador. Cualquier forma corta del operador de asignación puede ser fácilmente re-escrita en la forma normal del formato completo del operador de asignación. El resultado de su ejecución queda absolutamente inalterado.

```
In /= 33;           // Short form of the assignment operator
In = In/33;         // Full form of the assignment operator

St += "_exp7";      // Short form of the assignment operator
St = St + "_exp7";  // Full form of the assignment operator
```

El operador condicional if-else

Por regla general, si se escribe un programa de aplicación, es necesario que se de un código de varias soluciones en un solo programa. Para resolver estas tareas, se puede usar en el código el operador condicional 'if-else'.

Formato del operador "if-else"

Formato completo

El formato completo del operador **'if-else'** contiene una partida que incluye una condición, el cuerpo 1, la palabra clave 'else', y el cuerpo 2. El cuerpo del operador puede estar formado por uno o varios operadores, los cuerpos van encerrados entre llaves.

```
if (condición)                  // Cabecera del operador y condición
{
Bloque 1 de operadores          // Si la condición es verdadera, entonces ..
Composición cuerpo 1            // .. los agentes que componen el cuerpo 1 se ejecutan
}
else                            // Si la condición es falsa ..
{
Bloque 2 de operadores          // .. entonces los operadores ..
Composición cuerpo 2            // .. del cuerpo 2 se ejecutan
}
```

Formato sin 'else'

El operador **"if-else"** puede ser usado sin 'else'. En este caso, el operador "if-else 'contiene su cabecera que incluye una condición, y el cuerpo 1, que consta de uno o varios operadores cerrados entre llaves.

```
if (condición)                  // Cabecera del operador y el estado
{
Bloque 1 de operadores          // Si la condición es verdadera, entonces ..
Composición cuerpo 1            // .. agentes que componen el cuerpo 1 se ejecutan
}
```

Formato sin llaves

Si el cuerpo del operador "if-else 'consta de un solo operador, se pueden omitir las llaves.

```
if (condición)                  // Cabecera del operador y el estado
Operador                        // Si la condición es verdadera, entonces ..
                                // .. Este operador se ejecuta
```

Regla de Ejecución del operador "if-else"



Si la condición del operador "if-else" es cierta, se pasa el control al primer operador en el cuerpo 1. Después de que todos los operadores en el cuerpo 1 se han ejecutado, el control pasa al operador que sigue al operador "if-else". Si la condición del operador "if-else" es falsa, entonces:
-- Si esta la palabra clave 'else' en el operador "if-else", entonces se pasa el control al primer operador en el cuerpo 2. Después de que todos los operadores en el cuerpo 2 se han ejecutado, se pasa el control al operador que sigue el operador "if-else";
-- Si no hay una palabra clave 'else' en el operador "if-else", entonces se pasa el control al operador que sigue el operador " if-else ».

Ejemplos de ejecución del operador "if-else"

Veamos algunos ejemplos que muestran cómo podemos usar el operador "if-else».



Problema 9. Redactar un programa donde se realizan las condiciones siguientes: Si el precio de un símbolo ha subido y se ha superado un cierto valor, el programa deberá informar al comerciante sobre ese hecho, si no ha superado este valor, el programa no debe realizar ninguna acción.

Una de las soluciones para este problema puede ser, por ejemplo, como sigue (onelevel.mq4):

```
//+------------------------------------------------------------------+
// onelevel.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                          // función especial 'star'
  {
   double
   Level,                            // declaración de variable donde estará el nivel de alerta
   Price;                            // declaración de variable donde estará el precio actual   Level=1.2753;
// Establecer el nivel
   Price=Bid;                        // Solicitud de precio actual
//+------------------------------------------------------------------+
   if (Price>Level)                  // Operador 'if' con una condición
     {
      Alert("El precio a superado el nivel establecido");  // Mensaje para el comerciante
     }
//+------------------------------------------------------------------+
   return;                                 // // Salir de start()
  }
//+------------------------------------------------------------------+
```

Cabe señalar, en primer lugar, que el programa se crea como un Asesor Experto. Esto implica que el programa va a funcionar durante bastante tiempo con el objeto de mostrar el mensaje en la pantalla tan pronto como el precio supere el nivel preestablecido. El programa tiene sólo una función especial, start (). Las variables son declaradas y comentadas al comienzo de la función. Entonces el nivel de precios se sitúa en valores numéricos y el precio actual se solicita.

El operador "if-else» se utiliza en las siguientes líneas del programa:

```
//------------------------------------------------------------------------------
 if (Price>Level)                    // Operador 'if' con una condición
   {
    Alert("El precio a superado el nivel establecido");  // Mensaje para el comerciante
   }
//------------------------------------------------------------------------------
```

Tan pronto como el control en la ejecución del programa se pasa al operador "if-else ', el programa probará su condición. Tengase en cuenta que la prueba condicionada por el operador "if-else' es una propiedad inherente de este operador. Esta prueba no puede ser ignorado durante la ejecución del operador "if-else ', se trata de la "raison d'etre " (razón de ser) de este operador y se llevará a cabo, en todos los casos. Posteriormente, según los resultados de esta prueba, el control será pasado a cualquiera de los operadores del cuerpo o fuera de él, al operador le sigue el cierre de la llave.

A continuación en la Fig. 37, se puede ver un diagrama de bloques que representa una posible secuencia en el caso de una ejecución del operador "if-else».

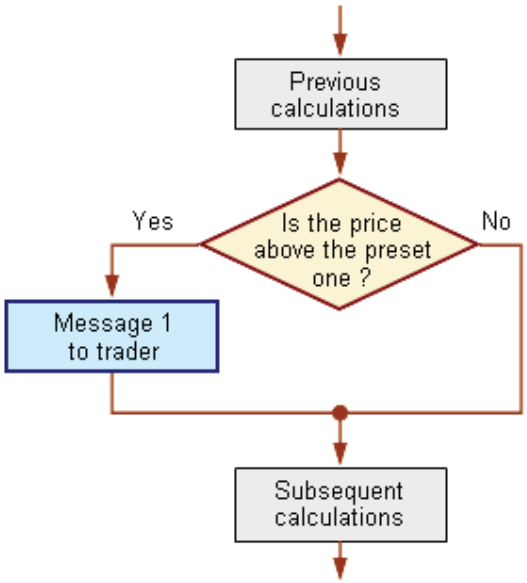


Fig. 37. Un diagrama de bloques para la ejecución del operador "if-else" en el programa onelevel.mq4.

En este y todos los siguientes gráficos, un rombo representa la verificación de la condición. Las flechas indican el componentes objetivo, para que el control será pasado después de que el bloque de declaración actual se haya ejecutado (el bloque de declaración significa un cierto juego de azar de los operadores adyacentes uno del otro). Vamos a considerar el esquema que aparece en más detalles.

El bloque de "Los cálculos previos" (una caja gris en el diagrama) en el programa onelevel.mq4 incluye lo siguiente:

```
double
   Level,                          // declaración de variable donde estará el nivel de alerta
   Price;                          // declaración de variable donde estará el precio actual      Level=1.2753;
// Establecer el nivel
   Price=Bid;                      // Solicitud de precio actual
```

Tras ejecutado el último operador en este bloque, el control pasa a la cabecera del operador "if-else" donde la condición de "¿excede el precio el nivel preestablecido?" (La caja de rombos en el diagrama, Fig. 37) se verifica:

```
 if (Price>Level)                    // Operador 'if' con una condición
```

En otras palabras, podemos decir que el programa en esta etapa esta buscando a tientas la respuesta a la siguiente pregunta: ¿la declaración entre paréntesis es verdad? La propia declaración suena literalmente como está escrito: El valor de la variable precio es superior a la de la variable nivel (el rango del precio). En el momento de comprobar si esta declaración es verdadera o falsa, el programa ya ha obtenido los valores numéricos de las variables precio y nivel. La respuesta depende de la relación entre estos valores. Si el precio está por debajo del nivel preestablecido (el valor del precio es igual o menor al valor de Nivel), la declaración es falsa, si el precio supera este nivel, la afirmación es cierta.

Así pues, cuando pasamos el control condicional después de la prueba que depende de la situación actual del mercado, **Si (If)** el precio de un símbolo se mantiene por debajo del nivel preestablecido (la respuesta es **No**, es decir, la declaración es **falsa),** el control, según a la norma de ejecución del operador "if-else ', se pasará fuera del operador, en este caso, se pasa al bloque denominado "**Cálculos posteriores",** es decir, a la línea:

```
return;                          // Salir de start()
```

Como es fácil ver, no se da ningún mensaje al Trader.

Si el precio de un símbolo supera el nivel preestablecido en el programa (la respuesta **es afirmativa,** es decir, la afirmación **es** cierta), el control será pasado al cuerpo del operador "if-else", es decir, a las siguientes líneas:

```
{
  Alert ("El precio a superado el nivel establecido");  // Mensaje para el comerciante
}
```

La ejecución de la función de Alert () dará lugar a la exhibición en la pantalla un pequeño recuadro con el siguiente mensaje:

```
El precio a superado el nivel establecido
```

La función Alert () es el único operador que hay en el cuerpo del operador "if-else", es por ello que después de su ejecución, el operador "else" se considera totalmente ejecutado, y el control pasa al operador que sigue el operador "if-else", es decir, a la línea:

```
return;                          // Salir de start()
```

La ejecución del operador «return» se traduce en que la función start () termina su trabajo, y el programa cambia al modo de espera de tick. En un nuevo tick (que también tiene un nuevo precio para el símbolo), la función start () se ejecutará otra vez. Así que, el mensaje codificado en el programa, dará o no la posibilidad de comercio, en función de si el nuevo precio supera o no el nivel preestablecido.

Los operadores if-else pueden estar anidados. Con el fin de demostrar cómo se pueden anidar estos operadores, vamos a examinar el siguiente ejemplo. El problema en sí es algo más sofisticado.



Problema 10. Redactar un programa con las siguientes condiciones: Si el precio ha crecido de manera que supera un cierto nivel 1, el programa deberá informar al comerciante sobre él, si el precio ha caído a fin de que sea inferior a un cierto nivel 2, la programa deberá informar al comerciante sobre ello, sin embargo, el programa no debe realizar ninguna acción, en cualquier otro caso.

Es evidente que, con el fin de resolver este problema, tenemos que comprobar el precio actual en dos ocasiones:

1. comparar el precio al nivel 1, y
2. comparar el precio al nivel 2.

Solución 1 del Problema 10

Actuando formalmente, podemos componer el siguiente algoritmo de solución:

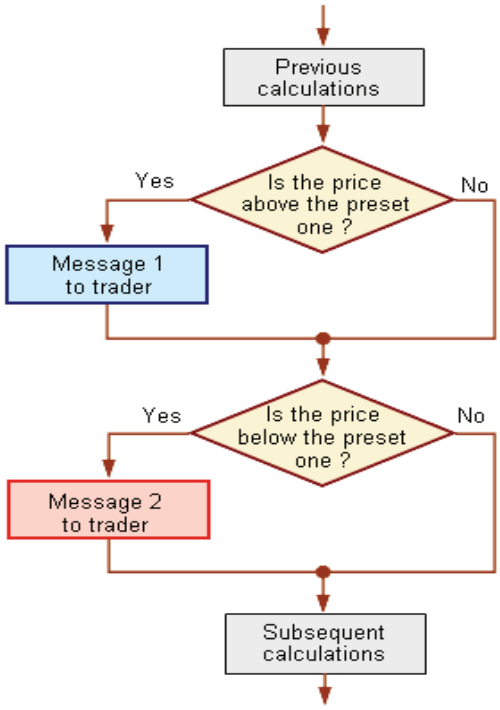


Fig. 38. Diagrama de bloques de los operadores if-else para ser ejecutado en programa twolevel.mq4.

El programa se da cuenta de que este algoritmo puede ser la siguiente (twolevel.mq4):

```
//+------------------------------------------------------------------+
// Twolevel.mq4
// El código debería ser usado para fines educativos l solamente.
//+------------------------------------------------------------------+
int start()                          // función especial 'start'
  {
   double
   Level_1,                           // nivel 1 de alerta
   Level_2,                           // nivel 2 de alerta
   Price;                            // Precio actual de mercado (precio para vender)
   Level_1=1.2850;                        // Establecer nivel 1
   Level_2=1.2800;                        // Establecer nivel 2
   Price=Bid;                         // Request price (Solicitud de precio Bid actual)
//+------------------------------------------------------------------+
   if (Price > Level_1)                 // ¿Es el precio actual mayor que el nivel 1?
     {
      Alert("El precio está por encima del nivel 1");  // Mensaje al comerciante
     }
//+------------------------------------------------------------------+
   if (Price < Level_2)                 // ¿Es el precio acutal menor que el nivel 2?
     {
      Alert("El precio está por debajo del nivel 2");  // Mensaje al comerciante
     }
//+------------------------------------------------------------------+
   return;                            // Salir de start()
  }
//+------------------------------------------------------------------+
```

Como es fácil de ver, el código de programa twolevel.mq4 es la versión ampliada del programa onelevel.mq4. si tuviéramos un solo nivel de los cálculos anteriores. Tenemos dos niveles en este nuevo programa, cada nivel se define numéricamente, y el programa, para resolver el problema planteado, tiene dos bloques que hacen un seguimiento de los comportamiento de los precios: ¿El precio cae dentro del rango de valores limitado por los niveles preestablecidos o es fuera de este rango?

Vamos a dar una breve descripción de la ejecución del programa.

Después de que se han realizado los cálculos preliminares, el control pasa al primer operador "if-else":

```
 //------------------------------------------------------------------
  if (Price > Level_1)                // ¿Es el precio actual mayor que el nivel 1?
    {
     Alert("El precio está por encima del nivel 1");  // Mensaje al comerciante
    }
//------------------------------------------------------------------
```

Independientemente de las acciones que tuvieran que llevarse a cabo con la ejecución de este operador (en este caso solo que se muestra o no un mensaje al comerciante), después de su ejecución el control pasa al siguiente operador "if-else":

```
 //------------------------------------------------------------------
  if (Price < Level_2)                // ¿Es el precio acutal menor que el nivel 2?
    {
     Alert("El precio está por debajo del nivel 2");  // Mensaje al comerciante
    }
//------------------------------------------------------------------
```

La ejecución consecutiva de ambos operadores resulta en la posibilidad de realizar las dos pruebas condicionales y, por último, resolver el problema. Aunque el programa resuelve la tarea en su totalidad, esta solución no puede ser considerada como absolutamente correcta. Por favor tengase en cuenta un detalle muy importante: La segunda prueba condicional se ejecutará independientemente de los resultados obtenidos en las pruebas en el primer bloque. El segundo bloque se ejecutará, incluso en el caso de que el primer operador sea verdadero, es decir, de que el precio superior al primer nivel establecido.

Cabe señalar aquí, que uno de los principales objetivos perseguidos por el programador al escribir sus programas es el algoritmo de optimización. Los ejemplos anteriores son sólo una manifestación, por lo que representan, de un corto y, por tanto, programa de ejecución rápida. Normalmente, un programa destinado a ser utilizado en la práctica es mucho más grande. Se pueden procesar los valores de cientos de variables, usar múltiples pruebas, cada una ejecutada en un tiempo determinado. Todo esto puede resultar en el riesgo de que la duración del trabajo de la función especial start () pueda superar el espacio de tiempo entre ticks. Esto significaría que algunos ticks puedrían quedar sin ser procesados. Esto es, por supuesto, una situación indeseable, y el programador debe hacer lo posible para prevenirlo. Una de las técnicas que permiten reducir el tiempo de la ejecución del programa es el algoritmo de optimización.

Vamos a considerar el ejemplo más reciente, una vez más, esta vez en términos de ahorro de recursos. Por ejemplo. Cuál es la razón por preguntar en el segundo bloque lo siguiente: "¿Esta el precio por debajo del nivel preestablecido?". Si las pruebas del primer bloque ya ha detectado que el precio está por encima del nivel superior, es evidente que el precio no puede estar por debajo del nivel inferior y por consiguiente, no hay necesidad de hacer una prueba condicional en el segundo bloque (ni para ejecutar el conjunto de los operadores en este bloque).

Solución 2 del Problema 10

Teniendo lo anterior en cuenta, vamos a considerar las siguientes, optimizado algoritmo:

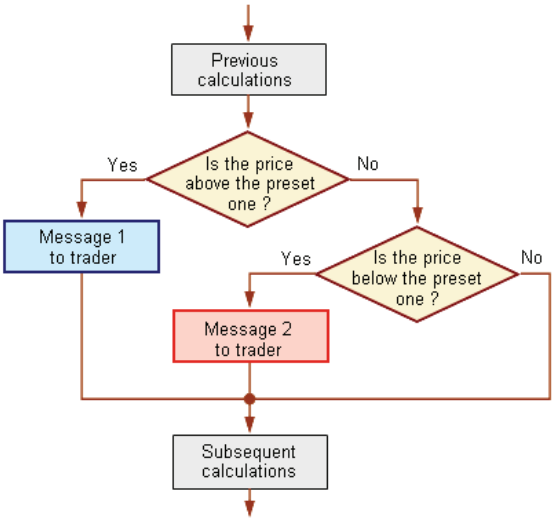


Fig. 39. Un diagrama de bloques para la ejecución en programa del operador "if-else" twoleveloptim.mq4.

Según el algoritmo se muestra en el diagrama de bloques anterior, las operaciones se ejecutan en el programa de forma no redundante. Después de la realización de los cálculos previos, el programa verificará si **el precio está por encima del nivel preestablecido.** En caso afirmativo, el programa muestrará el mensaje correspondiente para informar al comerciante y luego pasará el control a **los cálculos posteriores,** pero que la segunda condición **(esta el precio por debajo del nivel preestablecido?)** no será verificada. Sólo si el precio no ha resultado estar por encima del nivel superior (es decir la respuesta es **No** en el primer operador if-else **),** el control se pasará al segundo bloque donde se verificará la segunda condición. Si el precio resulta ser inferior al nivel mínimo establecido, se mostrará el mensaje correspondiente al comerciante. Si no es así, el control pasa al apartado "siguientes cálculos". Es evidente que, si el programa funciona según este algoritmo de acciones no redundantes, se realizan menos acciones, lo que se traduce en un ahorro considerable de recursos.

La aplicación programatica de este algoritmo implica el uso anidado del operador "if-else" (twoleveloptim.mq4):

```
//---------------------------------------------- ----------------------
// Twoleveloptim.mq4
// El código debería ser usado para fines educativos únicamente.
//---------------------------------------------- ----------------------

int start()                          // Special function start()
  {
   double
   Level_1,                          // Alert level 1
   Level_2,                          // Alert level 2
   Price;                           // Current price
   Level_1=1.2850;                       // Set level 1
   Level_2=1.2800;                       // Set level 2
   Price=Bid;                       // Request price
//------------------------------------------------------------------
   if (Price > Level_1)                  // Check level 1
     {
      Alert("The price is above level 1");   // Message to the trader
     }
   else
     {
       if (Price < Level_2)                // Check level 2
        {
         Alert("The price is above level 2"); // Message to the trader
        }
     }
//------------------------------------------------------------------
   return;                           // Exit start()
```

```
  }
//---------------------------------------------------------------------
```

Por favor tengase en cuenta este bloque de cálculos:

```
//---------------------------------------------------------------------

  if (Price > Level_1)                  // Check level 1
    {
     Alert ("The price is above level 1");   // Message to the trader
    }
  else
    {
     if (Price < Level_2)               // Check level 2
       {
        Alert ("The price is above level 2"); // Message to the trader
       }
    }
//---------------------------------------------------------------------
```

El operador "if-else", en la cual la segunda condición es comprobada es un componente del primer operador "if-else" que pone a prueba la primera condición. Los operadores anidados se utilizan ampliamente en la práctica. Esto es a menudo razonable y, en algunos casos, la única solución posible. Por supuesto, en lugar de la función de alerta (), los programas reales pueden utilizar otras funciones o realizar acciones útiles con diversos operadores.

Una expresión compleja puede ser usado como una condición en el operador "if-else".



Problema 11. Redactar un programa que tenga en cuenta de las condiciones siguientes: Si el precio cae dentro del rango preestablecido de valores, no se hace nada, si el precio está fuera de este rango, el programa debe informar al operador sobre el mismo.

La declaración de este problema es similar al del Problema 10. La diferencia consiste en que, en este caso, no estamos interesados en la dirección del movimiento de precios superior o inferior al rango predefinido. De acuerdo con la situación del problema, tenemos que conocer sólo este hecho: ¿esta el precio dentro o fuera del rango? Podemos utilizar el mismo texto para el mensaje que se mustra.

Aquí, al igual que en las anteriores soluciones, es necesario comprobar la situación del precio en relación con dos niveles: el superior y el inferior. Hemos rechazado la solución que se muestra en el algoritmo de la Fig. 38 por que no es eficiente. Por lo tanto, según el razonamiento anterior, podemos proponer la siguiente solución:

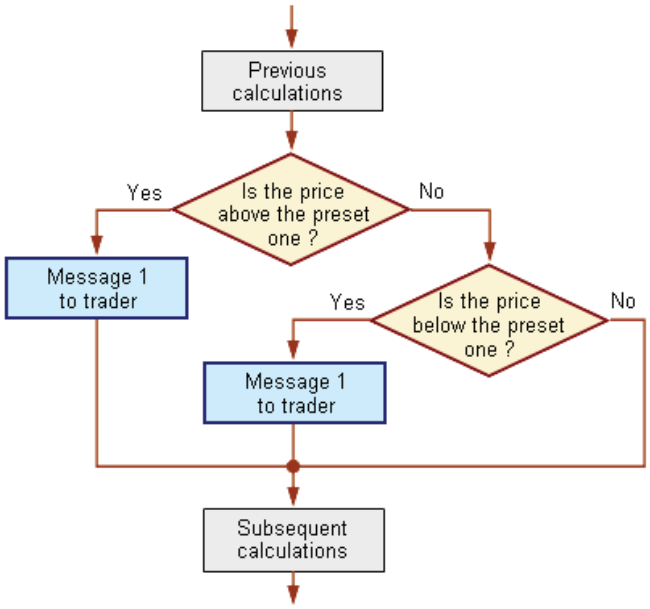


Fig. 40. Diagrama de bloques de una de las soluciones del problema 11.

Sin embargo, no hay necesidad de utilizar este algoritmo. El diagrama muestra el algoritmo que implica el uso del mismo bloque de mensajes en los distintos puntos del programa. Hay lineas de programa que se repetirán, a pesar de que la ejecución de ellas se traducirá en la redacción de los mismos mensajes. En este caso, sería mucho más eficaz utilizar el único operador "if-else ', con una compleja condición:

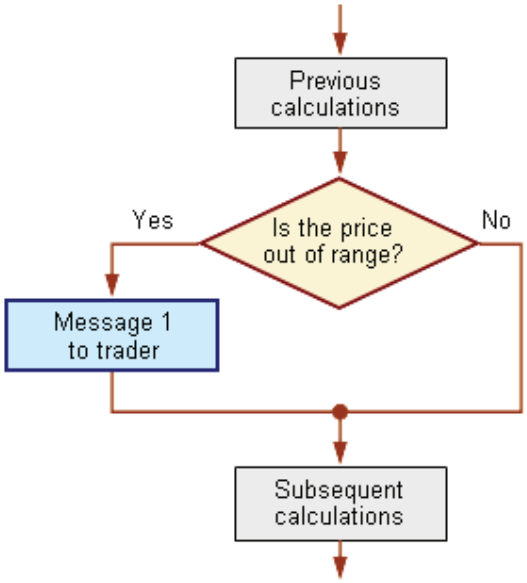


Fig. 41. Diagrama de bloques del programa para la ejecución del operador "if-else" compoundcondition.mq4.

El código del programa que implementa este algoritmo es el siguiente (compoundcondition.mq4):

```
//+------------------------------------------------------------------+
// compoundcondition.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                                 // Special function 'start'
  {
   double
   Level_1,                                 // Alert level 1
   Level_2,                                 // Alert level 2
   Price;                                  // Current price
   Level_1=1.2850;                             // Set level 1
   Level_2=1.2800;                             // Set level 2
   Price=Bid;                               // Request price
//+------------------------------------------------------------------+
   if (Price>Level_1 || Price<Level_2)            // Test the complex condition
     {
      Alert("The price is outside the preset range");// Message
     }
//+------------------------------------------------------------------+
   return;                                 // Exit start()
  }
//+------------------------------------------------------------------+
```

La idea fundamental de la solución de este programa es el uso de condiciones complejas en el operador "if-else":

```
 if (Price>Level_1 || Price<Level_2)            // Test the complex condition
  {
   Alert("The price is outside the preset range");// Message
  }
```

En pocas palabras, esta condición es la siguiente: "Si el valor de la variable precio es superior al de la variable Level_1, **O** el valor de la variable precio es menor que la variable de Level_2, el programa debe ejecutar el cuerpo del operador if-else". Se pueden usar operaciones lógicas (**&&**,**||** y **!**) al redactar las condiciones del operador if-else, que son ampliamente utilizados en la práctica de la programación (véase el operador booleano (lógico) Operaciones).

Cuando la solución de otros problemas, puede que sea necesario para componer aún más compleja condiciones, lo que hace que no se trata, también. Algunas expresiones, incluida la anidados, puede ser incluido en paréntesis. Por ejemplo, puede utilizar la siguiente expresión como una condición en el operador "if-else":

```
   if ( A>B && (B<=C || (N!=K && F>B+3)) )      // Example of a complex condition
```

Por lo tanto, MQL4 abre grandes oportunidades para el uso de los operadores if-else, ellos pueden ser anidados, pueden contener estructuras anidadas, pueden utilizar condiciones simples y complejas de prueba, lo que se traduce en la posibilidad de componer programas simples y complejos con los de algoritmos ramificados.

Ciclo del Operador "While"

La funcionalidad más potente de MQL4 es la posibilidad de organizar ciclos (bucles).

Al crear programas de aplicación, se puede utilizar a menudo cálculos repetidos, que son en su mayoría líneas repetidas de programa. Con el fin de hacer la programación cómoda y el mismo programa fácil de utilizar, se usan los operadores de ciclo. Hay dos operadores de ciclo en MQL4: **while** y **for**. Vamos a considerar el primero de ellos en esta sección.

Formato del explotador 'While'

El formato completo operador de ciclo While (mientras que) consiste en la cabecera que contiene una condición, y el ejecutable adjunto ciclo del cuerpo en llaves.

```
while (condición)                    // Cabecera del operador de ciclo

{                                    // Apertura llave
 Bloque de operadores                // El cuerpo de un operador de ciclo puede consistir ..
 que componen el ciclo del cuerpo    // ..  de varios operadores
 }                                   // Cierre llave
```

Si el ciclo del cuerpo se compone de un solo operador en el operador 'while' puede omitir las llaves.

```
while (condición)                    // Cabecera del operador de ciclo
Operador, cuerpo del ciclo           // ciclo del cuerpo es un operador
```

Regla Ejecución para el operador 'While'



Mientras la condición del operador "While" sea **"verdadera"**: El programa pasa el control al cuerpo del operador de bucle, después de que sido ejecutados todos los operadores en el cuerpo del bucle, se pasa el control a la cabecera para verificar la condición.
Si la condición del operador "While" es **"falso"**, el control debe ser pasado al operador que siguiente al operador de bucle "While".

Vamos a considerar un ejemplo.



Problema 12. Calcular el coeficiente de Fibonacci con la precisión de 10 cifras significativas.

En primer lugar, vamos a describir brevemente coeficiente de Fibonacci. El matemático italiano, Leonardo Fibonacci, descubrió una secuencia de números única:

1 1 2 3 5 8 13 21 34 55 89 144 233 ...

Cada número de esta secuencia es la suma de los dos números anteriores. Este número de secuencia tiene algunas propiedades únicas: La relación de dos números sucesivos en la secuencia es igual a 1,618, mientras que la proporción entre un número y el anterior es igual a 0,618. El ratio 1,618 fue llamado después ratio de Fibonacci, así como la secuencia de arriba que se llama secuencia de Fibonacci (también debería tenerse en cuenta que 0.3819, un conjugado para el número de Fibonacci, fue obtenido por la multiplicación de si mismo: 0.3819 x 0,618 = 0,618).

La tarea consiste en calcular el número de Fibonacci con la mayor precisión. Si analizamos el coeficiente de Fibonacci de varias decenas de elementos, es evidente que los coeficientes obtenidos en todo el rango de número irracional de 1,61803398875 ..., tomando mayor o menor valor por turnos. Los mayores números de la secuencia están involucrados en los cálculos, los más pequeños es la desviación del resultado de este valor.

No sabemos de antemano qué número exacto que debemos tomar para sus coeficientes que difieren entre sí sólo a partir de la décima cifra significativa. Por lo tanto, es necesario para componer un programa que busca en forma consecutiva hasta que los coeficientes de la diferencia entre ellos es menor de 0,0000000001.

En este caso, hemos creado un script (fibonacci.mq4) para resolver Problema 12, porque el programa no será utilizado durante mucho tiempo comprobando cada tick. Una vez que se vincula a la ventana de símbolo, el script debe llevar a cabo todos los cálculos necesarios (incluidos mostrar los resultados), después será desvinculado de la ventana del Terminal de Usuario.

```
//+------------------------------------------------------------------+
// fibonacci.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start ()                              // Función especial start ()
  {
//+------------------------------------------------------------------+
   int i;                          // parámetro formal, contador de iteraciones
   double
   A,B,C,                                 // Los números en la secuencia
   Delta,                                 // diferencia real entre los coeficientes
   D;                                     // Preset precisión
//+------------------------------------------------------------------+
   A=1;                                   // valor inicial
   B=1;                                   // valor inicial
   C=2;                                   // valor inicial
   D=0.0000000001;                  // valor de la precisión
   Delta=1000.0;                          // valor inicial
//+------------------------------------------------------------------+
   while(Delta>D)                         // Cabecera del operador de ciclo
     {                             // Apertura de llave del cuerpo del operador de bucle While
      i++;                                // incremento del contador de iteraciones
      A=B;                                // Siguiente valor
      B=C;                                // Siguiente valor
      C=A + B;                            // Siguiente valor
      Delta=MathAbs(C/B - B/A);      // Buscar diferencia entre los coeficientes
     }                             // Cierre de llave del cuerpo del operador de bucle While
//+------------------------------------------------------------------+
   Alert("C=",C," Fibonacci number=",C/B," i=",i);  // mostrar en la pantalla
   return;                                // Salir de star ()
  }                                       //Cierre de llave de la funcion especial star
//+------------------------------------------------------------------+
```

Al comienzo del programa, se declaran (y describen) las variables. En las líneas posteriores, son valores numéricos asignados a las variables. A, B y C toman el valor de los primeros números de la secuencia de Fibonacci. Cabe señalar aquí que, si bien la propia secuencia sólo contiene números enteros, el cociente de la división debe ser considerado en el programa como un número real. Si usamos el tipo ***int*** para estos números aquí, sería imposible calcular el coeficiente de Fibonacci, por ejemplo: 8/5 = 1 (integer 1, sin la parte fraccional). Así, en este caso, usamos las variables de tipo ***double***. El ciclo del operador, como tal, tiene este aspecto:

```
//+------------------------------------------------------------------------------------+
  while (Delta>D)                        // Cabecera del operador de ciclo
    {                                 // Apertura de llave del cuerpo del operador de bucle While
     i++;                                // incremento del contador de iteraciones
     A=B;                                // Siguiente valor
     B=C;                                // Siguiente valor
     C=A + B;                            // Siguiente valor
     Delta=MathAbs(C/B - B/A);      // Buscar diferencia entre los coeficientes
    }                                 // Cierre de llave del cuerpo del operador de bucle While
//+------------------------------------------------------------------------------------+
```

Vamos a considerar un diagrama de bloques del operador de bucle 'while':

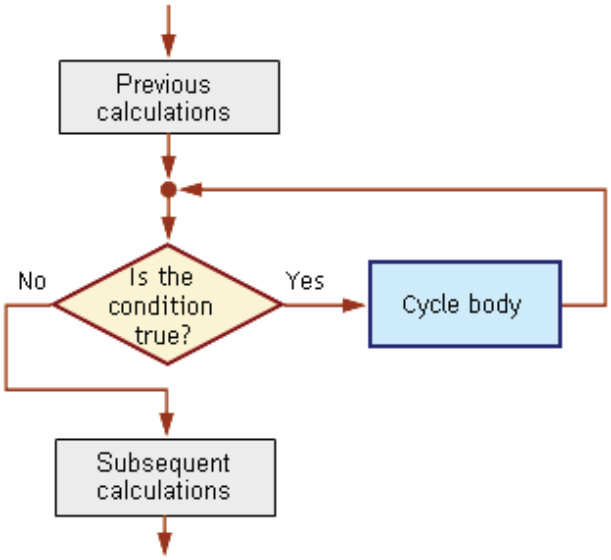


Fig. 42. Diagrama de bloques del operador 'while' en la ejecución del programa fibonacci.mq4.

El operador de ciclo comienza el trabajo con la prueba de la condición. El ciclo se llevará a cabo en repetidas ocasiones, hasta que la condición (Delta> D) sea verdad. Usted comprenderá el código de programación mucho mejor, si usted dice la frase clave (la norma de ejecución) leyendo el operador en voz alta. Por ejemplo, puede leer el operador while (mientras que) así: **"Mientras se cumpla esta condición, realiza lo/los siguiente/s: .."**. En este caso, la cabecera del operador es 'mientras que', y la condición: "Mientras Delta es mayor que D", realiza lo siguiente... Luego puede ir a analizar las lineas del programa que componen el cuerpo del bucle y que será ejecutado si la condición es verdadera.

Antes de que el control en el ejemplo anterior de fibonacci.mq4 pasa al operador de ciclo 'while', los valores de estas variables son iguales a 1000,0 y 0,0000000001, respectivamente, por lo que la condición se cumple en la primera convocatoria para el operador de ciclo. Esto significa que, después de que el estado ha sido probado, el control se pasa al primer operador en el ciclo cuerpo operador.

En nuestro ejemplo, es el siguiente operador:

```
    i++;                              // incremento del contador de iteraciones
```

En las tres líneas posteriores, se calculan los valores de la siguiente serie de secuencia de elementos:

```
    A=B;                              // Siguiente valor
    B=C;                              // Siguiente valor
    C=A + B;                          // Siguiente valor
```

Es fácil ver que las variables tomen los valores de la siguiente (más cercana más grande) elemento. Antes de que el operador del ciclo se ejecute, los valores de A, B y C son iguales a 1,0, 1,0 y 2,0, respectivamente. Durante la primera iteración, estas variables toman los valores 1,0, 2,0 y 3,0, respectivamente.

La iteración es una ejecución repetida de algunos cálculos, se utiliza para señalar que las líneas de programa que componen el cuerpo del operador del bucle son ejecutados.

En la línea siguiente, se calcula la diferencia entre los números de Fibonacci obtenidos sobre la base de los posteriores (C/B) y anteriores (B/A) de los elementos de la secuencia:

```
Delta=MathAbs(C/B - B/A);      // Buscar diferencia entre los coeficientes
```

En este operador, utilizamos la función estándar MathAbs () que calcula el valor absoluto de la expresión. Como mencionamos anteriormente, los coeficientes de Fibonacc toman, en turnos, grandes y pequeños valores (en comparación con el "estándar") con incrementos de los valores de la secuencia de elementos. Esta es la razón por la que la diferencia entre los coeficientes vecinos tendrá ahora un valor negativo, ahora un valor positivo. Al mismo tiempo, estamos realmente interesados en el valor de esta desviación, es decir, su valor absoluto. De este modo, cualquiera que sea la dirección que el valor actual del número de Fibonacci se desvie en el valor de expresión MathAbs (C/B - B/A), así como el valor de la variable de Delta, es siempre positivo.

Este operador es el último en la lista de operadores que componen el cuerpo del bucle. Esto se ve confirmado por la presencia de un cierre de llave en la línea siguiente. Tras la ejecución del cuerpo del último operador de ciclo , el control se pasa a la cabecera del operador para poner a prueba la condición. Este es el punto clave que determina la esencia del operador de ciclo . Según la condición del operador de ciclo sea verdadera o falsa, el control se pasará a cualquiera de la siguiente iteración o fuera del operador de ciclo.

En las primeras iteraciones, el valor de la variable Delta resulta ser mayor que el valor definido en la variable D. Esto significa que la condición (Delta> D) es cierto, por lo que el control se pasa al cuerpo del ciclo para realizar la siguiente iteración. Todas las variables involucradas en los cálculos llevará a nuevos valores: tan pronto como el cuerpo del ciclo llega el final, el control será pasado a la cabecera de nuevo para poner a prueba la condición y ver si esta es verdadera.

Este proceso continuará, hasta que la condición del operador de ciclo se convierte en falsa. Tan pronto como el valor de la variable se hace en Delta menor o igual al valor de D, la condición (Delta> D) ya no es cierta por más tiempo. Esto significa que el control será pasado fuera del operador de ciclo , a la línea:

```
 Alert ( "C =" C, "Fibonacci number =", C / B, "i =", i); // mostrar en la pantalla
```

Como resultado de ello, la alerta () operador ventana que contiene el siguiente mensaje aparecerá en la pantalla:

```
C = 317811 Fibonacci number = 1,618 i = 25
```

Esto significa que la búsqueda de precisión se alcanza en la 25 ª iteración, el valor máximo de la secuencia de Fibonacci elemento procesado en igualdad de condiciones a 317811. Coeficiente de Fibonacci, como era de esperar, es igual a 1,618. Este mensaje es la solución del problema planteado.

Cabe señalar aquí que los números reales se calculan en MQL4 con una precisión de 15 cifras significativas. Al mismo tiempo, el coeficiente de Fibonacci se muestra con una precisión de 3 cifras significativas. Esto se determina por el hecho de que la función de alerta () muestra números con una precisión de 4 cifras significativas, sin mostrar los ceros al final de la serie. Si quisiéramos mostrar el número de Fibonacci con una cierta precisión predefinidos, habría que cambiar el código, por ejemplo, de esta manera:

```
Alert("C=",C," Fibonacci number=",C/B," i=",i);  // mostrar en la pantalla
```

Lo siguiente debe ser especialmente observado:



Es posible que la condición especificada en el operador de ciclo de cabecera siempre sea cierta. Esto se traducirá en un bucle infinito.

Un bucle infinito o **Looping** es una ejecución repetida e infinita de las operaciones el ciclo del cuerpo de los operadores, es una situación crítica que se deriva de la realización de un algoritmo erróneo.

Una vez que el bucle se lleva a cabo, el programa ejecuta sin cesar el bloque de los operadores que componen el cuerpo del bucle. A continuación se muestra un simple ejemplo de un operador 'while' con bucle infinito.

```
int i = 1;                  // parámetro formal (contador)
while (i> 0)                // Operador de ciclo de cabecera
i + +;                      // Incremento del valor de i
```

En el ejemplo anterior, los valores de la variable i se incrementan en cada iteración. Como resultado de ello, la condición nunca se convertirá en falsa. Una situación similar ocurre, si no se calcula nada en el cuerpo del bucle. Por ejemplo, en el código que aparece a continuación:

```
int i = 1;                  // parámetro formal (contador)
while (i> 0)                // Operador de ciclo de cabecera
Alert ( "i =", i);          // mostrar en la pantalla
```

El valor de la variable i en el ciclo no cambia. El siguiente mensaje será mostrado en la pantalla en cada iteración:

```
i = 1
```

Este proceso se repetirá indefinidamente. Una vez que se cae en la trampa de un bucle infinito, el control no puede salir nunca del él. Esta situación es especialmente peligrosa en el comercio de Asesores Expertos y scripts. En tales casos, las variables de entorno no se actualizan normalmente, ya que la función especial no completa su funcionamiento, mientras que el trader puede desconocer la existencia de este bucle. Como resultado de ello, el control en la ejecución del programa no se puede pasar al correspondiente línea de programa de en las que se tome la decisión de apertura o cierre de órdenes.

El programador debe evitar tales condiciones, en las que se ejecuta en un bucle sin control posible. No hay ninguna técnica que ayuden a descubrir esta situación programática ni en la compilación del programa ni en su ejecución. El único método posible para detectar esos errores algorítmicos está en el examen del código, el razonamiento lógico y el sentido común.

Ciclo del operador for
Otro operador de ciclo es el operador 'for'.

Formato del explotador 'for'

El formato completo del operador de ciclo 'for' se compone de una cabecera que contiene Expression_1, la condición y Expression_2, y del cuerpo del bucle ejecutable adjunto entre llaves.

```
(Expression_1; Condición; Expression_2)   // Operador de ciclo de cabecera
{                                         // Apertura llave
Bloque de operadores                      // Ciclo del cuerpo puede consistir ..
que componen el ciclo del cuerpo          // ..  de varios operadores
}                                         // Cierre llave
```

Si el bucle del cuerpo en el operador 'for' consta de un solo operador, las llaves pueden omitirse.

```
(Expression_1; Condición; Expression_2) // Operador de ciclo de cabecera
Operador, ciclo del cuerpo              // el cuerpo del bucle es un solo operador
```

Expression_1, la condición y/o Expression_2 puede estar ausente. En cualquier caso, el carácter separador ";" (punto y coma) debe estar presente en el código.

```
(; Estado; Expression_2)              // Sin la Expression_1
{                                     // Apertura de llave
Bloque de operadores                  // Bucle del cuerpo puede consistir ..
que componen el cuerpo del bucle      // ..  de varios operadores
}                                     // Cierre de llave

// - ---------------------------------------------- ---------------------------------

(Expression_1;; Expression_2)         // Sin Condición
{                                     // Apertura de llave
Bloque de operadores                  // Cuerpo del bucle puede consistir ..
que componen el cuerpo del bucle      // ..  de varios operadores
}                                     // Cierre de llave
// - ---------------------------------------------- ---------------------------------

(;); // No expresiones o condición
{                                     // Apertura de llave
Bloque de operadores                  // Cuerpo del buclepuede consistir ..
que componen el cuerpo del bucle      // ..  de varios operadores
}                                     // Cierre de llave
```

Regla de Ejecución del operador 'for'



Tan pronto como el control pasa al operador 'for', el programa ejecuta Expression_1. En tanto que la condición del operador 'for' sea verdad: el control se pasa al primer operador en el cuerpo del bucle; tan pronto como se ejecutan todos los operadores en el cuerpo del bucle, el programa debe ejecutar la Expression_2 y pasar el control a la cabecera para probar la condición de ser verdad. Si la condición de que el operador 'for' es falso, entonces: el programa debe pasar el control al operador siguiente al operador 'for'.

Vamos a considerar cómo funciona el operador for. Para ello, vamos a resolver un problema.



Problema 13. Tenemos una secuencia de números enteros: 1 2 3 4 5 6 7 8 9 10 11 ... Redactar un programa que calcule la suma de los elementos de esta secuencia a desde N1 hasta N2.

Este problema es fácil de resolver en términos de matemáticas. Supongamos que queremos calcular la suma de elementos desde tercera hasta la séptima. La solución será: 3 + 4 + 5 + 6 + 7 = 25. Sin embargo, esta solución sólo es buena para un caso particular, cuando el número de los primeros y los últimos elementos que componen la suma es igual a 3 y 7, respectivamente. Un programa de la solución de este problema debe ser compuesto de tal manera que, si queremos calcular la suma en otro intervalo de la secuencia (por ejemplo, desde el 15 al 23 elemento), se podría sustituir fácilmente los valores numéricos de los elementos de una ubicación en el programa sin modificar las líneas de programa en otros lugares. A continuación se muestra una de las versiones de ese programa (script sumtotal.mq4):

```
//------------------------------------------------------------------------------
// sumtotal.mq4
// The code should be used for educational purpose only.
//------------------------------------------------------------------------------
int start()                       // Special function start()
  {
//------------------------------------------------------------------------------
   int
   Nom_1,                                 // Number of the first element
   Nom_2,                                 // Number of the second element
   Sum,                                   // Sum of the numbers
   i;                             // Formal parameter (counter)
//------------------------------------------------------------------------------
   Nom_1=3;                               // Specify numeric value
   Nom_2=7;                               // Specify numeric value
   for(i=Nom_1; i<=Nom_2; i++)          // Cycle operator header
     {                            // Brace opening the cycle body
      Sum=Sum + i;                        // Sum is accumulated
      Alert("i=",i,"  Sum=",Sum);         // Display on the screen
     }                            // Brace closing the cycle body
//------------------------------------------------------------------------------
   Alert("After exiting the cycle, i=",i,"  Sum=",Sum);// Display on the screen
   return;                                // Exit start()
  }
//------------------------------------------------------------------------------
```

En las primeras líneas de su programa, se declaran las variables (los comentarios explican el sentido de cada variable). En las líneas

```
Nom_1 = 3;                    // Especifica el valor numérico
Nom_2 = 7;                    // Especifica el valor numérico
```

se definen los valores numéricos del primer y del último elemento de la gama. Tengase en cuenta que estos valores no se especifican en cualquier parte del programa. Si es necesario, puede cambiar fácilmente estos valores (en un solo lugar) sin cambiar el código en otras líneas, por lo que el problema se puede resolver en otro rango de valores. Después de que el último de estos operadores se ha ejecutado, el control se pasa al operador de ciclo 'for':

```
for (i=Nom_1; i<=Nom_2; i++)        // Cycle operator header
   {                                // Brace opening the cycle body
    Sum = Sum + i;                                  // Sum is accumulated
    Alert("i=",i,"  Sum=",Sum);     // Display on the screen
   }                                // Brace closing the cycle body
```

La frase clave para recordar la regla de ejecución del operador 'for' - es el siguiente: **"Desde i= .., siempre y cuando .., .. incrementado en pasos de..., hacer lo siguiente: ..".** El uso de "paso .." es aplicable, si queremos utilizar un contador de almacenamiento, por ejemplo, i++, i = i +1, como Expression_2. En nuestro ejemplo, la frase clave es la siguiente: Desde **i** igual a Nom_1, siempre y cuando i sea menor o igual a Nom_2, paso 1, haga lo siguiente: (y luego - el análisis del cuerpo del bucle).

De acuerdo con la norma de ejecución del operador 'for' lo siguiente se llevará a cabo en este bloque del programa:

1. Ejecución de Expression_1:

```
i=Nom_1                       // Expresión 1
```

La variable i tendrá el valor numérico de la variable Nom_1, es decir, entero 3. Expression_1 se ejecuta una sola vez - cuando el control se pasa al operador 'for'. La expression_1 no participan en ningún acontecimiento posterior.

2. La condición es la prueba:

```
i<=Nom_2                      // Condición
```

El valor de la variable Nom_2 en la primera iteración es el entero 7, mientras que el valor de la variable i es igual a 3. Esto significa que la condición es verdadera (desde el 3 hasta un valor inferior a 7), es decir, el control se pasa al cuerpo del bucle para ejecutarlo.

3. En nuestro ejemplo, el cuerpo del bucle se compone de dos agentes que serán ejecutados uno por uno:

```
Sum = Sum + i;                // Sum is incrementada
Alert("i=",i,"  Sum=",Sum);   // Muestra en la pantalla
```

Variable Suma, en su inicialización, no tomó ninguna valor inicial, por lo que su valor es igual a cero antes del comienzo de la primera iteración. Durante los cálculos, el valor de la variable suma se incrementa por el valor de i, por lo que es igual a 3 al final de la primera iteración. Usted puede ver este valor en la caja de la función de alerta ():

```
i = 3, Suma = 3
```

4. El último acontecimiento que ocurre durante la ejecución del operador de ciclo es la ejecución de Expression_2:

```
i++                           // Expresión 2
```

El valor de la variable i se incrementa en uno. Este es el final de la primera iteración, el control se pasa a la condición de prueba.

A continuación, la segunda iteración comienza los pasos 2-4. Durante los cálculos, las variables tienen nuevos valores. En particular, en la segunda iteración, el valor de la variable i es igual a 4, mientras que la suma es igual a 7. El mensaje correspondiente se publicará, también. En general, el programa se desarrollará de acuerdo con el diagrama de bloques siguiente:

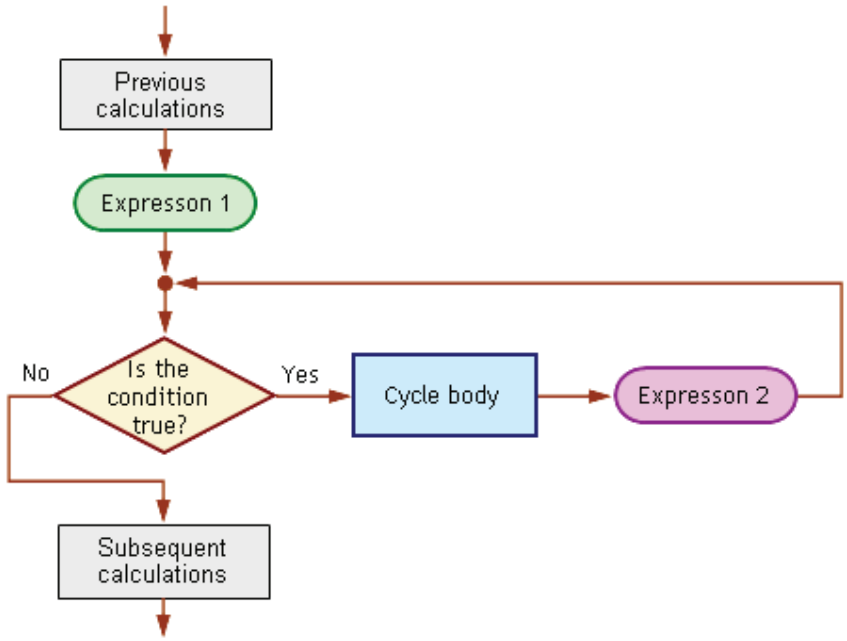


Fig. 43. Diagrama de bloques del operador-'for 'en la ejecución del programa sumtotal.mq4.

El ciclo de programa repite la ejecución del operador de ciclo, hasta que la condición se convierte en falsa. Esto sucederá tan pronto como el valor de la variable i es igual a 8, es decir, supera el valor preestablecido 7 de la variable Nom_2. En este caso, el control será pasado fuera el operador 'for', es decir, a la línea:

```
Alert ("i=",i,"  Sum=",Sum);          // Display on the screen
```

y, además, con la ejecución del operador «return» que termina la operación especial de la función start ().

Usted puede utilizar este programa para calcular la suma de cualquier otro rango de valores. Por ejemplo, si reemplaza las constantes 3 y 7 con 10 y 15, respectivamente, la ejecución del programa dará lugar a cálculo de la suma de los valores dentro de este rango.

En algunos lenguajes de programación, las variables que se especifican en el operador de ciclo de cabecera pierden sus valores después de salir del ciclo. Este no es el caso en MQL4. Todas las variables que intervienen en cualquier cálculo son válidas y mantienen sus valores después de salir del ciclo. Al mismo tiempo, hay algunas particularidades en relación con los valores de las variables que intervienen en los cálculos.

Por favor notese las dos últimos líneas los mensajes que siguen después de todos los cálculos están completos y el script se descarga:

```
i = 7 Suma = 25
Después de salir del ciclo, i = 8 Suma = 25
```

El primero de estos dos mensajes han aparecido en la etapa de la última iteración, antes de que el ciclo se completara, mientras que el segundo de ellos había sido dado por la última función alerta () antes de que el programa se diera por terminado. Es notable que estas líneas muestren diferentes valores de la variable i. Durante el ciclo de ejecución, este valor es igual a 7, pero es igual a 8 después de salir del ciclo. Con el fin de entender, ¿por qué ocurre esta manera, vamos a hacer referencia a la Fig. 43, una vez más.

La última (en este caso, la quinta) iteración comienza con las pruebas de la Condición. El valor de i es igual a 7 en este momento. Como el 7 no excede del valor de la variable Nom_2, la condición es verdadera y el cuerpo del buclese se lleva a cabo. Tan pronto como los agentes que componen el cuerpo del buclese se ejecutan, se realiza la última operación: ejecución de la Expression_2. Esto debe ponerse de relieve una vez más:



El último evento que tendrá lugar en cada iteración en la ejecución del operador 'for ' es el cálculo de Expression_2.

Esto se traduce en el aumento del valor de la variable i en 1, al final este valor será igual a 8. La posterior prueba de la condición (al principios de la próxima iteración) confirma que la condición es falsa. El resultado del condión lleva el control fuera del operador de ciclo. Al mismo tiempo, la variable i sale del operador de ciclo con el valor 8, mientras que el valor de la variable Suma sigue sin igual a 25, debido a que el cuerpo del bucle no se ejecuta ya en esta etapa. Esto debe considerarse en situaciones en las que el número de iteración no se conoce de antemano, de modo que el programador podrá estimar el valor de la variable calculada en Expression_2.

Podemos considerar un caso excepcional de una situación en la que en un bucle se podrían incurrir en diversos errores. Ejemplo de los errores si se utilizaran los siguientes operadores en la Expression_2 :

```
i--
```

El valor del contador se reducirá en cada iteración, por lo que la condición nunca se hará falsa.

Otro error puede ser una condición incorrectamente compuesta:

```
for(i=Nom_1; i>=Nom_1; i++)          // Cycle operator header
```

En este caso, el valor de la variable i será siempre superior o igual al valor de la variable Nom_1, con lo que la condición sería siempre cierta.

Intercambiabilidad entre los operadores 'White' y 'for'

El uso de un operador u otro depende totalmente de la decisión adoptada por el programador. La única cosa que queremos señalar es que estos operadores son intercambiables; a menudo es necesario sólo para dar otro cepillado al código de su programa.

Por ejemplo, hemos utilizado el operador de ciclo 'for' resolver Problema 13:

```
for (i=Nom_1; i<=Nom_2; i++)          // Cycle operator header
    {                                 // Brace opening the cycle body
     Sum = Sum + i;                                // Sum is accumulated
     Alert("i=",i," Sum=",Sum);       // Display on the screen
    }                                 // Brace closing the cycle body
```

A continuación se muestra un ejemplo de cómo el mismo fragmento del código se puede ver si utilizamos el operador "while":

```
i=Nom_1;                              // Оператор присваивания
  while (i<=Nom_2)                    // Cycle operator header
    {                           // Brace opening the cycle body
     Sum = Sum + i;                   // Sum is accumulated
     Alert("i=",i," Sum=",Sum);     // Display on the screen
     i++;                       // Increment of the element number
    }                           // Brace closing the cycle body
```

Como es fácil de ver, es suficiente sólo mover Expression 1 en la línea que precede al operador de ciclo, mientras que Expression 2 debe especificarse como el último ciclo en el cuerpo. Puede cambiar el ejemplo y su programa de lanzamiento para su ejecución, a fin de comprobar que los resultados son los mismos para ambas versiones.

El Operador "break"

En algunos casos, por ejemplo, en algún ciclo de programación de operaciones, podría ser necesario romper la ejecución de un ciclo antes de que su condición se convierta en falsa. Con el fin de resolver este problema podemos usar el operador "break".

Formato del operador "break"

El operador "break"se compone de una sola palabra y termina en el carácter ";" (punto y coma).

```
break;                    // Operador de "break"
```

Regla de Ejecución del operador "break"



El operador "break" detiene la ejecución de la parte más cercana del operador externo de tipo 'while',' for' o 'switch'. La ejecución del operador "break" consiste en pasar el control fuera del recinto del operador de tipo 'while',' for' o 'switch' al operador siguiente más cercano. El operador "break" solo se puede utilizar para la interrupción de la ejecución de los operadores mencionados anteriormente.

Podemos visualizar la ejecución del operador de 'break' con el siguiente ejemplo.



Problema 14. Tenemos un hilo de 1 metro de largo. Es necesario establecer el hilo en forma de un rectángulo con el área máxima posible. Se trata de hayar el área de este rectángulo y la longitud de los lados con una precisión de 1 mm en la búsqueda de serie en las variaciones.

No puede haber una cantidad ilimitada de rectángulos de diferentes tamaños hechos de un bloque unitario de hilo. Dado que la precisión requerida por la declaración del problema es de 1 mm, no podemos dejar de considerar las variaciones 999. El primero y el "más delgado" rectángulo tendrá las dimensiones de 1 mm x499, el segundo será de 2 mm x498, etc, mientras que las dimensiones del rectángulo último será 499 x1 mm. Tenemos que buscar en todos estos rectángulos y encontrar el de mayor superficie.

Como fácilmente se observa, hay dimensiones repetidas en el conjunto de rectángulos que estamos considerando. Por ejemplo, la primera y la última rectángulos tienen las mismas dimensiones: 1 x999 mm (al igual que 499 x1 mm). Del mismo modo, las dimensiones del rectángulo el segundo serán los mismos que los del pasado, sino un rectángulo, etc Tenemos que hacer un algoritmo de búsqueda que en todas las únicas variaciones, mientras que no hay necesidad de buscar en las reiteradas.

En primer lugar, vamos a calcular la relación entre la superficie y la longitud de los lados de un rectángulo. Como es fácil de ver, el primer rectángulo, con las dimensiones de 1x499, tiene la superficie más pequeña. Luego, con aumento de la cara más pequeña, el área del rectángulo, aumentará también. Tan pronto como se alcance un determinado valor, las áreas de los rectángulos comenzarán a disminuir de nuevo. Esta relación se muestra a continuación en la Fig. 44:

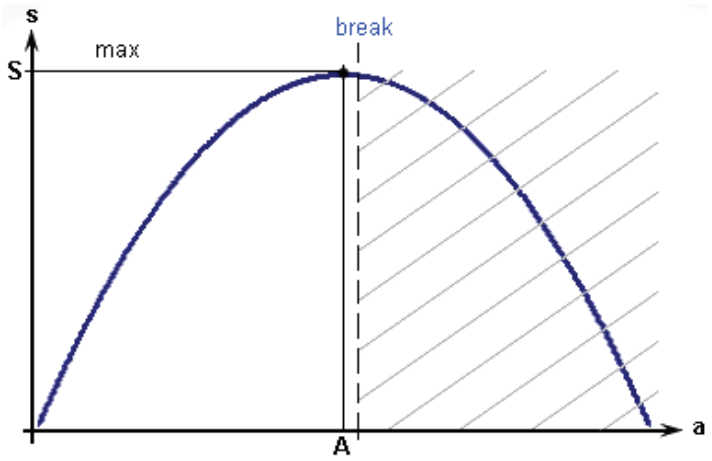


Fig. 44. Relación rectángulo entre la superficie y la longitud de uno de sus lados.

En la Fig. 44, podemos fácilmente llegar a la conclusión de que debemos encontrar la superficie máxima de la búsqueda en las variaciones a partir del primero de ellos, sólo mientras la zona aumenta durante los cálculos. Tan pronto como comienza a disminuir, vamos a romper nuestra búsqueda y salir del ciclo de búsqueda. A continuación se script rectangle.mq4 que implementa dicho algoritmo.

```
//+------------------------------------------------------------------+
// rectangle.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                                // Special function start()
  {
//+------------------------------------------------------------------+
   int
   L=1000,                        // Specified thread length
   A,                                      // First side of the rectangle
   B,                                      // Second side of the rectangle
   S,                                      // Area of the rectangle
   a,b,s;                                  // Current values
//+------------------------------------------------------------------+
   for(a=1; a<L/2; a++)                    // Cycle operator header
     {                                     // Brace opening the cycle body
      b=(L/2) - a;                  // Current value of the sides
      s=a * b;                      // Current value of the area
      if (s<=S)                     // Choose the larger value
         break;                     // Exit the cycle
      A=a;                          // Save the best value
      B=b;                          // Save the best value
      S=s;                                 // Save the best value
     }                                     // Brace closing the cycle body
//+------------------------------------------------------------------+
   Alert("The maximum area = ",S,"  A=",A,"  B=",B);// Message
   return;                       // Function exiting operator
  }
//+------------------------------------------------------------------+
```

Veamos cómo funciona este programa. Las variables son declaradas y comentadas al principio del programa. El algoritmo de resolución del problema en sí se realiza dentro del ciclo 'for'. El valor inicial del lado **a** del rectángulo, como se especifica igual a 1 en Expression_1. De acuerdo con el Estado, los valores se buscan siempre y cuando el tamaño del lado **a** del rectángulo sigue siendo menor que la mitad de la longitud de hilo. La Expression_2 prescribe para aumentar la longitud del lado **a** del rectángulo actual en cada paso de iteración.

Las variables a, b y s son variables que registran los calculos actuales, los valores que se buscan estan en las variables A, B y S. El segundo lado **b** y la superficie **s** del rectángulo son calculados al comienzo del ciclo.

```
b = (L/2) - a;              // Current values of the sides
s = a * b;                  // Current value of the area
```

La condición de salir del ciclo se prueba en el operador 'if':

```
if (s <= S )                // Choose the larger value
break;                      // Exit the cycle
```

Si la recién calculada área **s** del rectángulo actual resulta ser mayor que el área de **S** calculada en la iteración anterior, este nuevo valor de **s** se convierte en el mejor resultado posible. En este caso, la condición del operador 'if' no se cumple, y el control se pasa al operador más cercano que sigue el operador 'if'. A continuación se presentan las líneas en las que se guardan los mejores resultados:

```
A = a;                      // Save the best value
B = b;                      // Save the best value
S = s;                      // Save the best value
```

Tan pronto como el programa llega a la llave de cierre, iteración está acabada y el control se pasa a la Expression_2 de la cabecera del operador 'for' para ejecutar y poner a prueba la condición. Si la longitud del lado **a** no ha alcanzado el límite determinado a partir del momento de la prueba, el ciclo seguirá siendo ejecutado.

Los repetidos cálculos cíclico seguirán hasta que uno de los siguientes eventos se lleva a cabo: o bien la longitud del lado **a** supera los límites predefinidos (de acuerdo a la condición del operador 'for'), o el tamaño del área calculada **s** se hace en inferior a un valor previamente almacenado en la variable **S.** Hay una buena razón para creer que la salida del bucle, de de acuerdo a la condición del operador 'if', se llevará a cabo antes:

```
if (s <= S )                // Choose the larger value
break;                      // Exit the cycle
```

De hecho, el operador de ciclo 'for' está compuesto de tal manera que se hará una búsqueda exhautiva en todas las variantes posibles (la mitad de la longitud del hilo, L/2, es la suma de dos lados del rectángulo). Al mismo tiempo, la máxima de superficie del rectángulo, se alcanzará en algún lugar en medio de las búsquedas del conjunto de las variantes. Entonces, tan pronto como esto ocurre (la zona del actual rectángulo **s** resulta ser igual o menor al valor alcanzado previamente **S),** el control en la ejecución del operador "if" se pasa al operador "break" que, a su vez, pasa el control fuera el operador 'for', a la siguiente línea:

```
Alert("The maximum area = ",S," A=",A," B=",B);// Message
```

Como resultado de la ejecución de la función estándar de alerta (), se imprimirá la siguiente línea:

```
The maximum area = 62500 A=250 B=250
```

Después de esto, el control se pasa al operador «return», lo que da lugar a la terminación de la función especial start (). Esto, a su vez, da lugar a la finalización de la escritura y a ser desvinculado por el Terminal de Usuario de la ventana de símbolo.

En este ejemplo, el operador de "break" (pasa el control fuera) el ciclo del operador 'for'. A continuación se muestra el diagrama de bloques del ciclo 'for' con la salida especial:

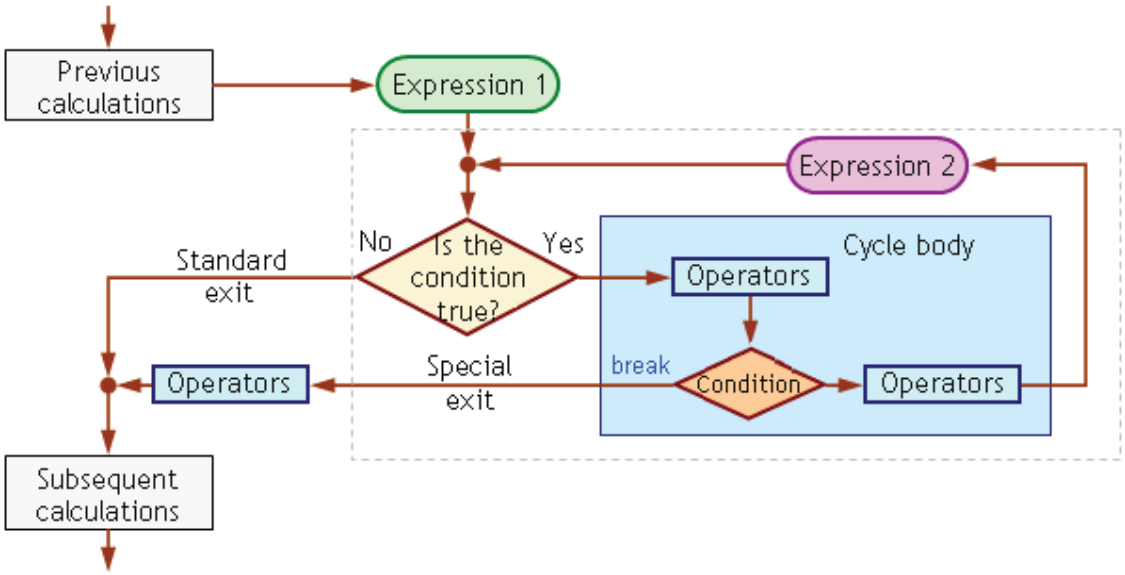


Fig. 45. Diagrama de bloque del ciclo 'for' usando el operador 'break' (rectangle.mq4).

Como se observa en el diagrama, hay dos salidas de ciclo: una salida normal (ciclo de salida resultantes de la activación de la condición especificada en el operador de ciclo de cabecera) y una salida especial (el cuerpo del bucle de salida de acuerdo con una condición adicional y utilizando el operador "break").

Es difícil sobrestimar la posibilidad de utilizar una salida a un ciclo. En nuestro ejemplo, el operador "break" nos permite hacer un algoritmo que realiza sólo los cálculos necesarios. Un algoritmo sin salida especial sería ineficiente: en este caso, se realizarían cálculos repetidos, lo que daría lugar a una pérdida de tiempo y de recursos computacionales poco razonables. La región "crosshatched" en la Fig. 44 visualiza la región de parámetros que no fueron tratados en el programa anterior (¡casi la mitad de todos los cálculos!), lo que no nos impide obtener el resultado correcto.

La necesidad de utilizar algoritmos eficientes es especialmente evidente, cuando el tiempo de ejecución se extiende a segundos e incluso minutos, lo que puede exceder el intervalo de tiempo entre los ticks. En este caso, existe el riesgo de omitir el tratamiento de algunas informaciones y ticks y como resultado, perder el control de su trading. Además, puede sentir la reducción del tiempo tomado por los cálculos especialmente si prueba sus programas en el Tester de Estrategia. La prueba de sofisticados programas de una larga historia puede tomar horas e incluso días. Reducir a la mitad el tiempo empleado por pruebas es una característica muy importante, en este caso.

La norma establece que el operador "break" se detiene al más cercano operador externo. Vamos a ver cómo funciona un programa que utiliza los ciclos anidados.



Problema 15. Utilizando el algoritmo del Problema 14, hayar la más corta búsqueda con múltiples hilos de 1 metro y suficientes para formar un rectángulo con una superficie de 1,5 m².

En este problema, debemos buscar en forma lineal en las longitudes hilo disponible y calcular la superficie máxima para cada longitud de hilo. La solución del problema 15 se realiza en el script llamado area.mq4. La soluciónes de las variantes se buscan en dos ciclos: internos y externos. El ciclo externo búsca en las longitudes de hilo con el paso de 1000 mm, mientras que el interior se encuentra la superficie máxima para la hilo longitud actual. En este caso, el operador de "break" se utiliza para la salida externa y la interna del ciclo.

```
//+------------------------------------------------------------------+
// area.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                       // Special function start()
  {
//+------------------------------------------------------------------+
   int
   L,                             // Thread length
   S_etalon=1500000,                        // Predefined area (m²)
   S,                             // Area of the rectangle
   a,b,s;                         // Current side lengths and area
//+------------------------------------------------------------------+
   while(true)                              // External cycle for thread lengths
     {                            // Start of the external cycle
      L=L+1000;                             // Current thread length of в mm
      //+---------------------------------------------------------------+
      S=0;                        // Initial value..
      // ..for each dimension
      for(a=1; a<L/2; a++)           // Cycle operator header
        {                         // HStart of the internal cycle
         b=(L/2) - a;                       // Current side lengths
         s=a * b;                 // Current area
         if (s<=S)                          // Choose the larger value
            break;                // Exit internal cycle
         S=s;                     // Store the best value
        }                         // End of the internal cycle
      //+---------------------------------------------------------------+
      if (S>=S_etalon)                      // Choose the larger value
        {
         Alert("The thread length must be ",L1000," m.");// Message
         break;                   // Exit the external cycle
        }
     }                            // End of the external cycle
//+------------------------------------------------------------------+
   return;                        // Exit function operator
  }
//+------------------------------------------------------------------+
```

El ciclo interno trabaja aquí como lo hizo en la solución del problema anterior. El operador "break" se utiliza para salir del ciclo 'for', cuando la superficie máxima se encuentra dada por la longitud de hilo. Cabe señalar que el operador "break" se especifica en el ciclo interno 'for' pasa el control al operador que sigue la llave cerrada el ciclo 'for', y se detiene el ciclo de esta manera. Este fenómeno no influye en la ejecución del ciclo externo del operador »While« en modo alguno.

Tan pronto como el operador de "break" se dispara en el interior del ciclo 'for', el control se da al operador 'if':

```
   if (S >= S_etalon)                       // Choose the larger value
     {
      Alert("The thread length must be ",L1000," m.");// Message
      break;                              // Exit the external cycle
     }
```

En el operador 'if' que comprueba si la superficie es igual o mayor de 1,5 m², el valor mínimo permitido por la declaración del problema. Si la respuesta es afirmativa, entonces la solución se encuentra y no tiene sentido que siga el cálculo; el control será pasado al cuerpo del operador 'if'. La parte ejecutable del operador 'if' se compone solo de dos operadores, el primero de los cuales muestra el mensaje en pantalla de la solución encontrada:

```
El hilo debe ser de una longitud de 5 m.
```

El segundo operador, "break", se utiliza para salir del ciclo externo 'while' y, posteriormente, para salir del programa. El diagrama de bloques del algoritmo realizado en el programa area.mq4 se muestra a continuación:

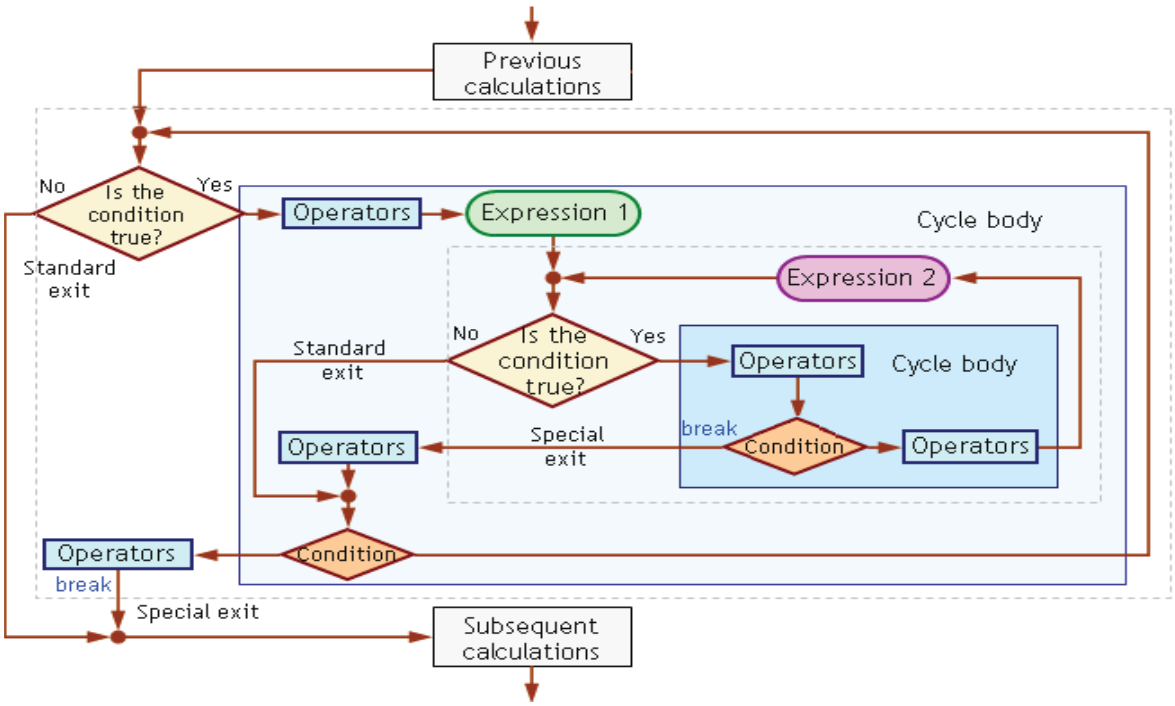


Fig. 46. Diagrama de bloques del programa con la posibilidad de salida especial en los ciclos interno y externo (area.mq4).

Como se observa en el diagrama, hay una salida normal y una salida especial para cada ciclo. Cada operador "break" pone fin a su ciclo correspondiente, pero no tiene efecto alguno sobre el otro ciclo; cada salida especial se corresponde con su propio operador de "break". El mismo operador "break" no se puede utilizar en ambos ciclos especiales como salida. En este caso, cada ciclo tiene sólo una salida especial. Sin embargo, en general, es posible utilizar varios operadores "break" para hacer varias salidas en especiales de un ciclo.



Tenga en cuenta que la condición en la cabecera del ciclo exterior 'while' es (cierto), es decir, un texto que no contiene una variable, el valor de lo que debería cambiar durante la ejecución del programa. Esto significa que el programa nunca podría salir del ciclo 'While' bajo la condición especificada en la cabecera. En este caso, la única posibilidad para salir del ciclo es usar el operador "break".

En este ejemplo, el algoritmo se construye de tal manera que, de hecho, el programa utiliza sólo la salida especial para salir del ciclo interno y externo. Sin embargo, usted no debe pensar que el uso del operador "break" siempre se traduce en la construcción de algoritmos que sólo implica salidas especiales. En los algoritmos de otros programas, es muy posible que el programa llegue a la condición y los usos normales de salida para salir de sus ciclos.

Cómo usar el operador "break" para pasar el control fuera el operador 'switch' se considera en la sección del operador "switch".

Operador "continue"

A veces, cuando se utiliza un ciclo en el código, es necesario poner fin al principio del tratamiento de la iteración actual y pasar a la siguiente sin ejecutar el resto de los operadores que componen el cuerpo del bucle. En estos casos se debe usar el operador "continuar".

Formato del operador "continue"

El operador 'Continue' consta de una sola palabra y termina en ";" (punto y coma).

```
continue; // operador "continue"
```

Ejecución Regla del operador "continue"



El operador "continue" detiene la ejecución de la iteración actual del ciclo más cercano del operador 'while' o' for'. La ejecución del operador "continue" da como resultado ir a la próxima iteración del ciclo más cercano operador 'while' o 'for'. El operador "continue" sólo se puede utilizar en el cuerpo del ciclo por encima de los operadores.

Vamos a examinar la forma en que el operador "continue" puede ser utilizado prácticamente.



Problema 16. Hay 1000 ovejas en la primera granja. La cantidad de ovejas en la primera granja aumenta en un 1% diario. Si la cantidad de ovejas es superior a 50.000 a finales de mes, entonces el 10% de las ovejas serán transferidos a la segunda granja. ¿En qué momento el importe de ovejas en la segunda granja llega a 35.000? (Consideramos que hay 30 días hábiles en un mes.)

El algoritmo de la solución de este problema es evidente: Tenemos que organizar un ciclo en el que el programa calcule la cantidad total de ovejas en la primera granja. De acuerdo con el planteamiento del problema, las ovejas se transfieren a la segunda granja a finales de mes. Esto significa que tendremos que crear un ciclo más interno donde se calcule el ciclo de acumulación de ovejas en el mes en curso. Entonces tendremos que comprobar a finales de mes, si el límite de 50.000 ovejas se supera o no. Si la respuesta es sí, entonces hay que calcular la cantidad de ovejas a ser transferidos a la segunda granja a finales de mes y la cantidad total de ovejas en la segunda granja.

El algoritmo en estudio se realiza en el script sheep.mq4. El operador "continue" se usa aquí para hacer cálculos en el ciclo externo.

```
//+------------------------------------------------------------------+
// sheep.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                          // Special function start()
  {
//+------------------------------------------------------------------+
   int
   day,                                          // Current day of the month
   Mons;                                         // Search amount of months
   double
   One_Farm    =1000.0,                 // Sheep on the 1st farm
   Perc_day    =1,                               // Daily increase, in %
   One_Farm_max=50000.0,                         // Sheep limit
   Perc_exit   =10,                              // Monthly output, in %
   Purpose     =35000.0,                // Required amount on farm 2
   Two_Farm;                                     // Current amount of farm 2
//+------------------------------------------------------------------+
   while(Two_Farm < Purpose)            // External cycle on history
    {                                   // Start of the external cycle body
     //+------------------------------------------------------------+
     for(day=1; day<=30; day++)                  // Cycle for days of month
       One_Farm=One_Farm*(1+Perc_day/100);//Accumulation on the 1st farm
     //+------------------------------------------------------------+
     Mons++;                                     // Count months
     if (One_Farm < One_Farm_max)                // If the amount is below limit,.
       continue;                                 // .. don't transfer the sheep
     Two_Farm=Two_Farm+One_Farm*Perc_exit/100;//Sheep on the 2nd farm
     One_Farm=One_Farm*(1-Perc_exit/100);// Remainder on the 1st farm
    }                                   // End of the external cycle body
//+------------------------------------------------------------------+
   Alert("The aim will be attained within ",Mons," months.");//Display on the screen
   return;                                       // Exit function start()
  }
//+------------------------------------------------------------------+
```

Por lo general las variables se describen y comentan al comienzo del programa. El mismo cálculo se realiza en el ciclo 'while'. Después de que ha sido ejecutado, el mensaje correspondiente en la pantalla. Se ejecutan los cálculos en el ciclo exterior 'while' será la ejecutado, hasta que el objetivo es alcanzado, es decir, hasta que la cantidad total de ovejas en la segunda granja alcanza el valor esperado de 35.000.

El ciclo interno 'for' es muy simple: el valor del saldo diario aumenta en un 1%. El análisis de la suma no se realiza en este ciclo, ya que, según el planteamiento del problema, las ovejas sólo pueden ser transferidas a finales de mes. Así, después de salir del ciclo 'for', la variable One_Farm tiene un valor que es igual a la cantidad de ovejas de la primera granja. Inmediatamente después de que el valor de la variable Mons se calcula, el cual se incrementa en 1 en la ejecución de cada iteración del ciclo externo 'while'.

De acuerdo con la actual cantidad de ovejas en la primera granja, una de las dos acciones más adelante se llevarán a cabo:

- si la cantidad de ovejas en la primera granja supera el valor límite de 50 000, entonces el 10% de ovejas de la primera granja deben ser transferidos a la segunda granja;
- de otro modo, las ovejas de la primera granja estan en la primera granja y se crían más.

El algoritmo se ramifica usando el operador 'if':

```
if (One_Farm < One_Farm_max)       // If the amount is below limit,.
continue;                          // .. don't transfer the sheep
```

Estas líneas de código pueden caracterizarse de la siguiente manera: Si la cantidad de ovejas en la primera granja está por debajo del valor límite de 50 000, a continuación, ejecutar el operador que componen el cuerpo del operador 'if'. En la primera iteración, la cantidad de ovejas de la primera granja resulta ser inferior al valor límite, por lo que el control se pasa al cuerpo del operador 'if', es decir, actua el operador 'continue'. La ejecución del operador 'continue' significa lo siguiente: Finaliza la actual iteración del ciclo de ejecución más cercano (en este caso, es el ciclo en que el operador 'while') y pasa el control a la cabecera del operador de ciclo. Las siguientes líneas programa que también son una parte del cuerpo ciclo del operador'while no se ejecutará:

```
Two_Farm = Two_Farm+One_Farm*Perc_exit/100;//Sheep on the 2nd farm
One_Farm = One_Farm*(1-Perc_exit/100);     // Remainder on the 1st farm
```

En estas líneas de arriba se hace efectiva la transferencia del 10% de ovejas de la primera granja a la segunda y se calcula el ganado ovino que queda en la primera granja. Es imposible hacer estos cálculos en la primera iteración, ya que el límite de 50 000 ovejas en la primera granja no se ha llegado aún. La esencia del operador en "continue" consiste en saltarse estas líneas sin terminar de ejecutar la iteración actual.

La ejecución del operador "continue" consiste en pasar el control a la cabecera del operador de ciclo 'while'. Entonces la condición en el operador de ciclo y se prueba y la siguiente iteración comienza. El ciclo seguirá siendo ejecutado de acuerdo con el mismo algoritmo, hasta que la cantidad de ovejas en la primera granja llega al límite fijado. Los valores de la variable Mons se acumulan en cada iteración.

En una cierta fase de cálculo, la cantidad de ovejas de la primera granja alcanzará o superará el límite fijado de 50 000 cabezas. En este caso, la ejecución de la condición del operador 'if', (One_Farm <One_Farm_max) se convierte en falsa, por lo que el control no se pasa al cuerpo del operador 'if'. Esto significa que el operador 'Continue' no será ejecutado y el control será pasado a la primera de las dos últimas líneas del ciclo del cuerpo 'while´: En primer lugar, el programa calculará la cantidad de ovejas de la segunda granja y, a continuación, la cantidad de ovejas que quedan en la primera granja. Después de que estos cálculos se han completado, la iteración del ciclo 'while' termina y el control vuelve de nuevo a la cabecera del ciclo. A continuación se muestra el diagrama de bloques del algoritmo realizado en script sheep.mq4.

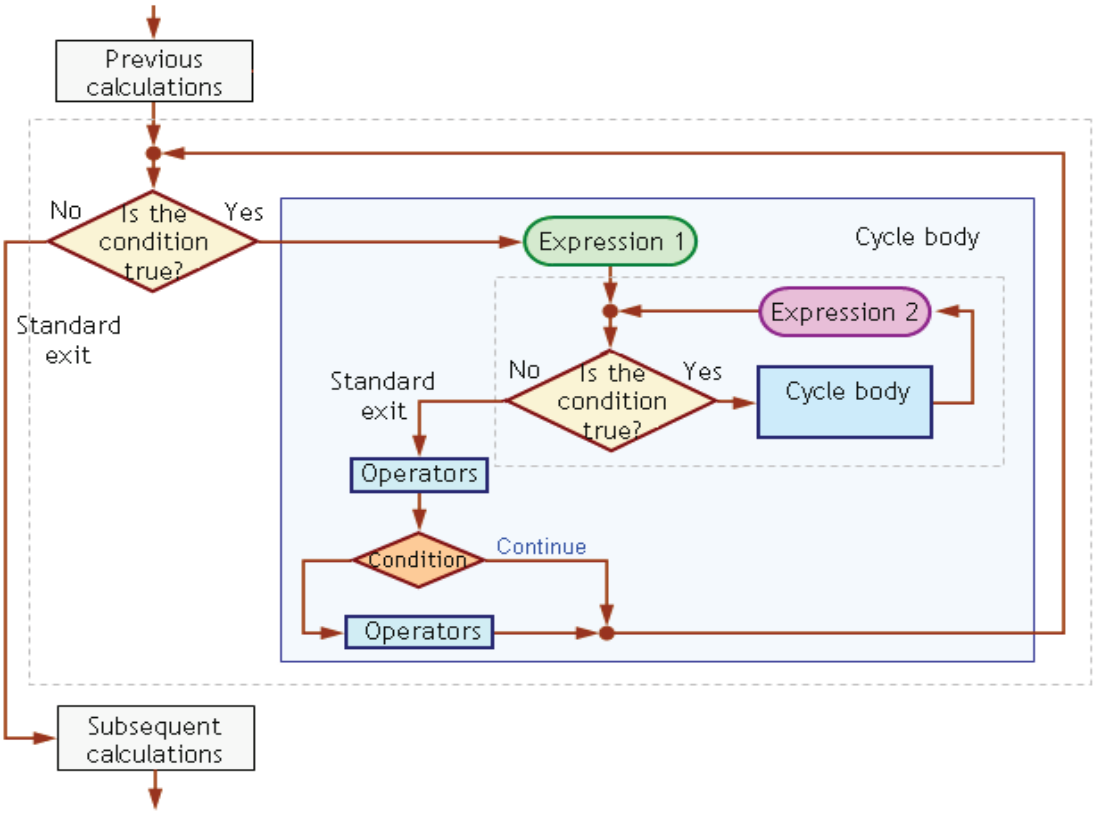


Fig. 47. Diagrama de bloques de un programa donde el operador "continue" rompe la iteraciones del ciclo externo (sheep.mq4).

En el diagrama, podemos ver que, dependiendo de la ejecución de la condición de operador 'if', epasl control será pasado de inmediato a la cabecera del ciclo 'while' (como resultado de la ejecución del operador "continue"), o se ejecutan algunos operadores mas primero, y luego el control se sigue pasando a la cabecera del ciclo 'while'. En ninguno de los dos caso termina la ejecución del ciclo.

Tengase en cuenta que la presencia del operador "continue" en el cuerpo del ciclo no provoca que necesariamente la finalice la iteración actual y el control se vaya directamente a la cabecera del operador de ciclo while. Esto sólo ocurre si el operador continue se ejecuta, es decir, si el operador if previo le pasa el control.

En las sucesivas iteraciones en el operador de ciclo 'while', se calcula el nuevo valor alcanzado por la variable Two_Farm. Tan pronto como este valor supera el límite fijado de 35 000, se cumple el requisito en el que el operador de ciclo 'while' se converte en falso, por lo que los cálculos en el cuerpo del operador de ciclo while no se ejecutarán mas y el control pasará al operador más cercano que sigue al operador de ciclo while:

```
Alert("The aim will be attained within ",Mons," months.");//Display on the screen
[Alert ( "El objetivo será alcanzado dentro", Mons, "meses".); // Mostrar en la pantalla]
```

La ejecución de la función de Alert () tendrá como resultado que se muestre la siguiente línea:

```
The aim will be attained within 17 months. (El objetivo será alcanzado dentro de 17 meses).
```

El operador «return» completa la ejecución de la función especial start (), lo que se traduce en que el control se pasa al Terminal de Usuario, mientras que la ejecución del script termina y se desvinculará de la ventana de símbolo.

El algoritmo anterior también proporciona un estándar de salida del ciclo 'while'. En un caso más general, la salida del ciclo puede ser el resultado de la ejecución del operador "break" (salida especial). Sin embargo, ni una ni otra manera de cerrar el ciclo se relacionan con la interrupción de la actual iteración usando el operador 'continue'.

En el estado de ejecución del operador "continue" se utiliza la frase de "el operador más cercano". Esto no significa que las lineas de programa se encuentren cerca una de la otra. Vamos a echar un vistazo al código de script sheep.mq4. El operador 'for' es el "más cercano" para el operador 'continue' que la cabecera del ciclo 'while'. Sin embargo, esto no significa nada: El operador 'continuae' no tiene ninguna relación con el ciclo 'for', porque esta fuera de su cuerpo. En general, un programa puede contener múltiples ciclo anidados de operadores. El operador "continue" está siempre en el cuerpo de uno de ellos. Al mismo tiempo, el operador "continue", forma parte del ciclo interno de operador, por supuesto, también dentro del ciclo externo del operador. La frase de "detiene la ejecución de la iteración actual del operador de ciclo más cercano" debe significar que el operador 'Continue' influye en el ciclo del operador mas cercano dentro del cuerpo del que el operador 'continue' se encuentra. Con el fin de visualizar esto, vamos a cambiar ligeramente la declaración del problema.



Problema 17. Hay 1000 ovejas en una granja. La cantidad de ovejas en esta primera granja aumenta diariamente en un 1%. al día, cuando la cantidad de ovejas en la primera granja llega a 50.000, el 10% de las ovejas serán transferidos a la segunda granja. ¿En qué momento la cantidad de ovejas de la segunda granja llega a 35 000? (Nosotros consideramos que hay 30 días hábiles en un mes.)

La solución del problema 17 se realiza en el script othersheep.mq4. En este caso, el operador "continue" se utiliza para el cálculo tanto en el exterior e interior ciclos.

```
//+------------------------------------------------------------------+
// othersheep.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                                   // Special function start()
  {
//+------------------------------------------------------------------+
   int
   day,                                       // Current day of the month
   Mons;                                      // Search amount of months
   double
   One_Farm    =1000.0,                 // Sheep on the 1st farm
   Perc_day    =1,                            // Daily increase, in %
   One_Farm_max=50000.0,                      // Sheep limit
   Perc_exit   =10,                           // One-time output, in %
   Purpose     =35000.0,                // Required amount on farm 2
   Two_Farm;                                  // Current amount of farm 2
//+------------------------------------------------------------------+
   while(Two_Farm < Purpose)            // Until the aim is attained
    {                                   // Start of the external cycle body
     //+--------------------------------------------------------+
     for(day=1; day<=30 && Two_Farm < Purpose; day++)// Cycle by days
       {
        One_Farm=One_Farm*(1+Perc_day/100);          //Accumulated on farm 1
        if (One_Farm < One_Farm_max)                       // If the amount is below limit,.
          continue;                                        // .. don't transfer the sheep
        Two_Farm=Two_Farm+One_Farm*Perc_exit/100;    //Accumulated on farm 2
        One_Farm=One_Farm*(1-Perc_exit/100);         //Remainder on farm 1
       }
     //+--------------------------------------------------------+
     if (Two_Farm>=Purpose)                  // If the aim is attained,..
       continue;                                       // .. don't count months
```

```
     Mons++;                                                  // Count months
    }                                                  // End of external cycle body
//---------------------------------------------------------------------
  Alert ("The aim will be attained within ",Mons," months and ",day," days.");
  return;                                                  // Exit function start()
 }
//---------------------------------------------------------------------
```

Para resolver este problema, tenemos que analizar la cantidad de ovejas para cada día. Con este fin, los siguientes:

```
  if (One_Farm < One_Farm_max)           // If the amount is below limit,.
    continue;                        // .. don't transfer the sheep
  Two_Farm=Two_Farm+One_Farm*Perc_exit/100;//Accumulated on farm 2
  One_Farm=One_Farm*(1-Perc_exit/100);     //Remainder on farm 1
```

se transfieren en el interior del ciclo organizado para los días del mes. Es evidente que una cierta cantidad de ovejas se transfieren a la segunda granja, en este caso, no a finales de mes, sino en el día, cuando la cantidad de ovejas en la primera granja llega al valor predefinido. La razón para usar aquí el operador "continue" es la misma: Queremos **saltar** las líneas que contienen los cálculos relacionados con la transferencia de ovejas de una granja a otra, es decir, no queremos ejecutar estas líneas. Tengase en cuenta que el ciclo 'while' que se construyó para los días del mes no se interrumpe y actua de forma independiente a si las ovejas son transferidas o no, porque el operador "continue" influye en el operador de ciclo del más cercano, en nuestro caso, el operador de ciclo 'for'.

En el operador se utilizó una expresión compleja para la condición del operador de ciclo 'for':

```
for(day=1; day<=30 && Two_Farm<Purpose; day++)// Cycle by days
```

El uso de la operación && ("and") significa que el ciclo se ejecutará siempre y cuando ambas condiciones sean verdaderas:

- el mes no ha terminado todavía, es decir, la variable «día» oscila entre 1 a 30, y
- la cantidad de ovejas en la segunda granja no ha alcanzado el valor predefinido y sin embargo la variable Two_Farm es inferior a Objeto (35 000).

Si al menos una de las condiciones no se cumple (la condiciónse convierte en falsa), el ciclo se detiene. Si la ejecución del ciclo 'for' se detiene (por cualquier motivo), el control pasa al operador 'if':

```
if (Two_Farm >= Purpose)                    // If the aim is attained,..
continue;                                   // .. don't count months
Mons++;                                // Count months
```

El uso del operador "continue" en esta ubicación en el código tiene un sentido simple: El programa debe seguir contando meses solamente si la cantidad de ovejas en la segunda granja está por debajo del valor esperado. Si el objetivo ya ha sido alcanzado, el valor de la variable Mons no cambiará, mientras que el control (como resultado de la ejecución de "continue") se pasa a la cabecera del operador de ciclo más cercano, en nuestro caso, el del operador de ciclo 'while'.

El uso de los operadores "continue" en los ciclos anidados se muestra en la Fig. 48 infra.

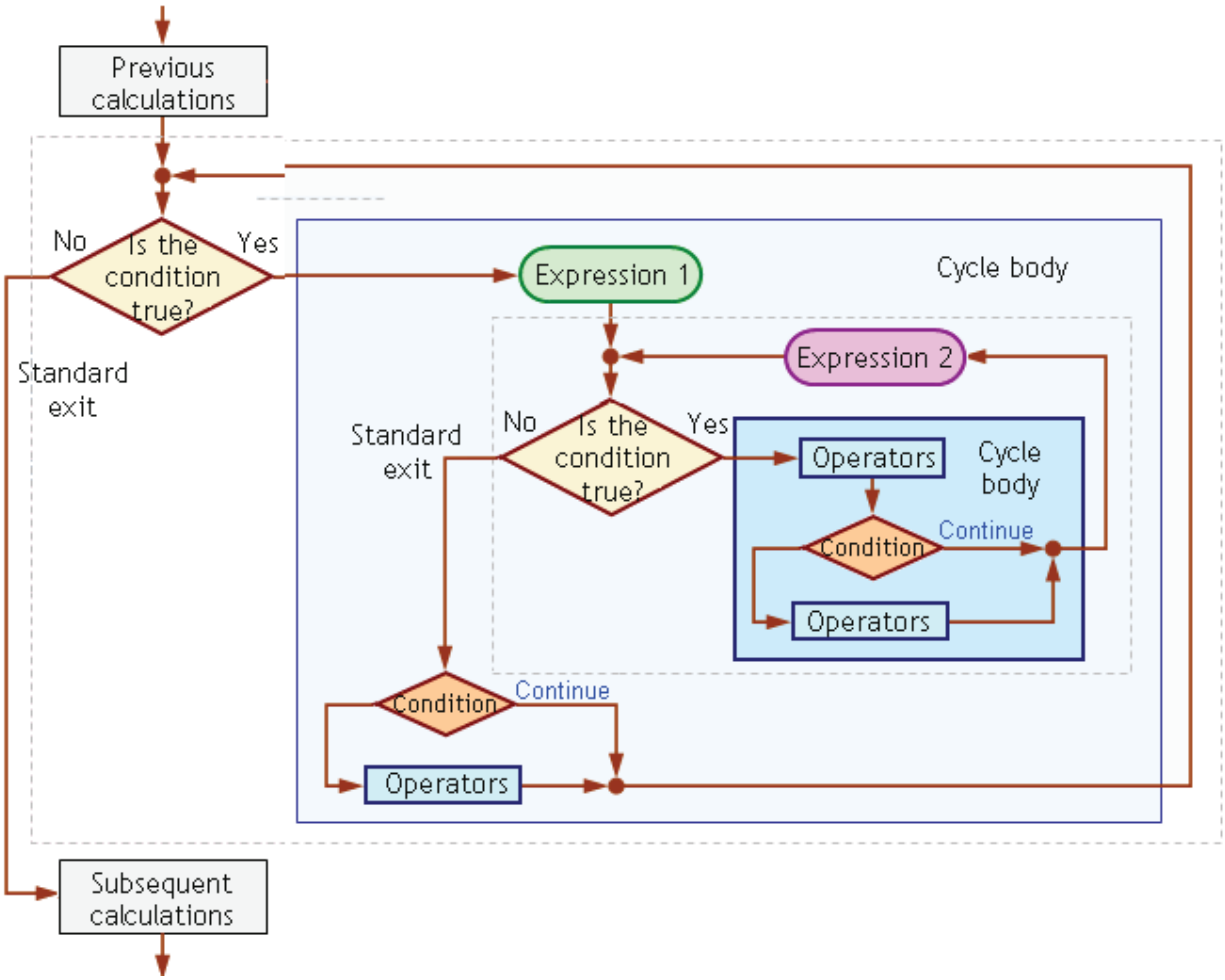


Fig. 48. Diagrama de bloques del algoritmo basado en los ciclos anidados. Cada ciclo contiene su cuerpo operador "continue" (othersheep.mq4).

En el ejemplo anterior, cada ciclo (externo e interno) tiene un operador en "continue" que sólo influye en su "propio" ciclo más cercano (en el ciclo dentro del cual está actuando), pero no por cualquier acontecimiento ó en cualquier otro ciclo. En general, en un cuerpo del bucle se pueden utilizar varios operadores "continue" y cada uno de ellos ser capaz de interrumpir la iteración actual como resultado de la reunión de una condición. La cantidad de operadores "continue" en un cuerpo de un bucle no esta limitada.

Operador 'switch'

Algunos programas conllevan la necesidad de la ramificación de su algoritmo en variantes. En estos casos, es muy conveniente usar el operador "switch", sobre todo si hay decenas o cientos de variaciones, mientras que, en este caso, el operador ‘if’ se conviertiría en un código hinchado si utiliza muchos operadores anidados.

Formato del operador 'switch'

El operador 'switch' consiste en una cabecera y un cuerpo ejecutable. La cabecera contiene el nombre del operador y una expresión entre paréntesis. El cuerpo del operador contiene una o varias alternativas o casos ('case') y una variante «por defecto».

Cada variante "case" consiste en la de la palabra clave 'caso', una constante, ":" (dos puntos), y los operadores. La cantidad de variantes 'caso' no está limitada.

La variación 'por defecto' se compone de la palabra clave 'por defecto', ":" (dos puntos), y los operadores. La variante 'por defecto' se especifica la última en el cuerpo del operador 'switch', pero también puede ser ubicado en cualquier lugar dentro del cuerpo operador, o incluso estar ausente.

```
switch (Expresión)                  // Operador de cabecera
{                                   // Apertura de llave
case Constante: Operadores          // Una de las variantes 'caso'
case Constante: Operadores          // Una de las variantes 'caso'
...
[Default: Operadores]               // Variante sin ningún tipo de parámetro
}                                   // Cierre de llave
```

Los valores de la expresión y de los parámetros sólo pueden ser los valores de tipo ***int***. La expresión puede ser una constante, una variable, una llamada a una función, o una expresión. Cada variante "caso" puede ser marcado por un entero constante, un carácter constante, una constante o expresión. Una expresión constante no puede incluir variables o llamadas a funciones.

Regla de Ejecución del operador 'switch'



El programa debe pasar el control al primer operador que sigue a ":" (dos puntos) de la variante "case" cuyo valor la constante es el mismo que el valor de la expresión y, a continuación, ejecutar uno por uno todos los operadores que componen el cuerpo del operador 'switch'. La condición de igualdad entre la expresión y la prueba es constante en la dirección de arriba hacia abajo y de izquierda a derecha. Los valores de las variantes constantes de los diferentes 'case' no debe ser el mismo. El operador “break” detiene la ejecución del operador 'switch' y pasa el control al operador que sigue el operador 'switch'.

Es fácil ver que la constante de **`case´**: "representa sólo una etiqueta, para que el control sea pasado. Todos los operadores que componen el cuerpo del operador 'switch' empezarán a ejecutarse a partir de esta etiqueta. Si el algoritmo del programa implica la ejecución de un grupo de operadores que se corresponden con una sola variante "caso", se debe especificar el operador “break” como el último en la lista de operadores que se corresponden con una sola variante "caso". Vamos a considerar el trabajo del operador 'switch' a través de algunos ejemplos.

Ejemplos de operador 'switch'



Problema 18. Redactar un programa con las siguientes condiciones: Si el precio excede del nivel predefinido, el programa deberá informar al comerciante sobre ello mediante un mensaje que describe el exceso (hasta 10 puntos); en los demás casos, el programa debe

informar que el precio no excede el nivel predefinido.

A continuación se muestra la solución de problemas usando el operador 'switch' (Asesor Experto pricealert.mq4):

```
//+------------------------------------------------------------------+
// pricealert.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                                          // Special function 'start'
  {
   double Level=1.3200;                              // Preset price level
   int Delta=NormalizeDouble((Bid-Level)Point,0);    // Excess
   if (Delta<=0)                                  // Price doesn't excess the level
     {
      Alert("The price is below the level");         // Message
      return;                                    // Exit start()
     }
//+------------------------------------------------------------------+
   switch(Delta)                                  // Header of the 'switch'
     {                                                // Start of the 'switch' body
      case 1 : Alert("Plus one point");     break;    // Variations..
      case 2 : Alert("Plus two points");    break;
      case 3 : Alert("Plus three points");  break;
      case 4 : Alert("Plus four points");   break;    //Here are presented
      case 5 : Alert("Plus five points");   break;    //10 variations 'case',
      case 6 : Alert("Plus six points");    break;    //but, in general case,
      case 7 : Alert("Plus seven points"); break;    //the amount of variations 'case'
      case 8 : Alert("Plus eight points");  break;    //is unlimited
      case 9 : Alert("Plus nine points");   break;
      case 10: Alert("Plus ten points");    break;
      default: Alert("More than ten points");        // It is not the same as the 'case'
     }                                               // End of the 'switch' body
//+------------------------------------------------------------------+
   return;                                        // Exit start()
   }
//+------------------------------------------------------------------+
```

En solución de este problema, se utiliza el operador 'switch', en el que cada variante "case" utiliza el operador "break". Dependiendo del valor de la variable Delta, el control se pasa a una de las variantes 'case'. Esto se traduce en la ejecución de los operadores correspondientes a esta variante: la función de alert () y el operador de "break". El operador "break" detiene la ejecución del operador 'switch', y pasa el control **fuera** de ella, es decir, al operador «return» que termina la operación de la función especial start (). Así, según sea el valor de la variable de Delta, una de las variantes "caso" se activa, mientras que otras variantes siguen intactas.

El programa anterior es un Asesor Experto, por lo que éste se pondrá en marcha en cada tick y en cada uno de ellos mostrará el mensaje correspondiente con la situación actual. Por supuesto, debemos buscar un valor para el Nivel lo más cerca posible del precio actual a la ventana del símbolo a la que está vinculado este EA.

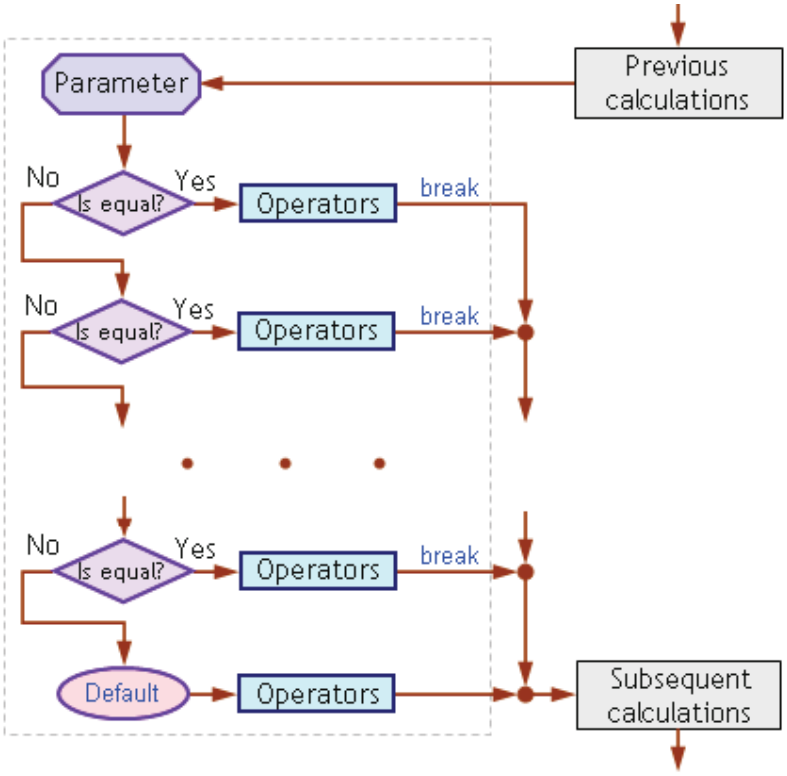


Fig. 49. Diagrama de bloques del operador 'switch' en AE pricealert.mq4.

En el diagrama de bloques anterior, podemos ver claramente que, debido a la presencia del operador "break" en cada variante "caso", el control se pasa fuera del operador 'swich' después de la ejecución de los operadores de cualquier variante "caso". Un principio similar de construcción de algoritmo usando el operador 'switch' se utiliza en el archivo llamado stdlib.mq4 emitido dentro de la Terminal de Usuario ***(.. \ expertos \ librerías \ stdlib.mq4).***

Vamos a considerar otro problema que no prevé el uso de "break" en cada una variante "caso".



Problema 19. Hay 10 barras. Informe sobre los números de todas las barras a partir de la enesima barra.

Es bastante fácil codificar la solución de este problema (script barnumber.mq4):

```
//+------------------------------------------------------------------+
// barnumber.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                          // Special function start()
  {
   int n = 3;                                // Preset number (nth bar) (enesima barra)
   Alert("Bar numbers starting from ", n,":");  // It does not depend on n
//----------------------------------------------------------------------
   switch (n)                                // Header of the operator 'switch'
     {                                 // Start of the 'switch' body
      case 1 : Alert("Bar 1");                 // Variations..
      case 2 : Alert("Bar 2");
      case 3 : Alert("Bar 3");
      case 4 : Alert("Bar 4");                 // Here are 10 variations ..
      case 5 : Alert("Bar 5");                 // ..'case' presented, but, in general, ..
      case 6 : Alert("Bar 6");                 // ..the amount of variations..
      case 7 : Alert("Bar 7");                 // ..'case' is unlimited
      case 8 : Alert("Bar 8");
      case 9 : Alert("Bar 9");
      case 10: Alert("Bar 10");break;
      default: Alert("Wrong number entered");// It is not the same as the 'case'
     }                                 // End of the 'switch' body
//----------------------------------------------------------------------
   return;                           // Operator to exit start()
  }
//+------------------------------------------------------------------+
```

En el operador 'switch', el programa buscará en las variantes 'caso', hasta que detecta que la expresión es igual a la constante. Cuando el valor de la expresión (en nuestro caso, el entero 3) es igual a una de las constantes (en este caso el caso 3), todos los operadores siguientes a los dos puntos ":" (caso 3:) se ejecutarán, a saber: el operador llama a la función Alert ("Bar 3") y los siguientes Alert ("Bar 4"), Alert ("Bar 5"), etc, hasta que llega al operador "break" que termina el funcionamiento del operador "switch" .

Si el valor de la expresión no coincide con ninguna de las Constantes, el control se pasa al operador que corresponde a la variante "default":

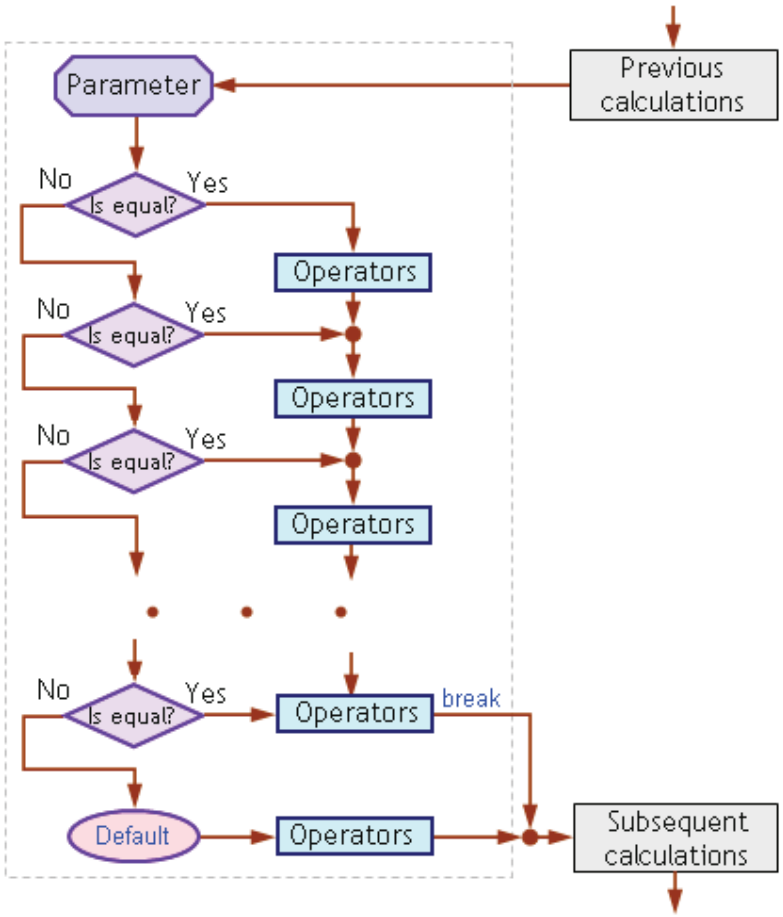


Fig. 50. Diagrama de bloques del operador 'swicth' en el script barnumber.mq4.

A diferencia del algoritmo realizado en el anterior programa, en este caso (Fig. 50), no estamos utilizando el operador “break” en cada una de las variantes "case". Por tanto, si el valor de la expresión es igual al valor de una de las constantes, se ejecutarán todos los operadores siguientes a los operadores de la correspondiente variante “caso”. También utilizamos el operador “break” en la última variante “caso” para otro propósito: evitar que se ejecuten los operadores correspondientes a la variante "por defecto". Si no hay un valor igual a la expresión entre los valores de las constantes, el control se pasa directamente al operador que se corresponde con la etiqueta 'default'.

Por lo tanto, si el valor de la variable n preset se encuentra dentro del rango de valores de 1 a 10, los números de todas las barras se imprimirá a partir de la enésima. Si el valor de n esta por encima del rango entre 1 y 10, el programa informará al usuario que se ha entrado un numero erróneo fuera del rango.



Nota: No es necesario que las constantes de las variantes “case” se ordenen en su programa según la magnitud. El orden de cómo las variantes “case” con las correspondiente constantes se siguen la una a la otra está determinada por las necesidades del algoritmo de su programa.

La Funcion de Llamada

Una llamada a función puede ser usada como un operador independiente y encontranse en cualquier lugar del programa donde esté implicado un cierto valor (con la excepción de los casos predefinidos). El formato y normas de ejecución de una llamada a una función cubre tanto las funciones estándar (built-in) como las funciones definidas por el usuario.

Formato de la función de llamada

Una llamada a una función esta compuesta por el nombre de la función y la lista de los parámetros escritos entre paréntesis:

```
NombreDeLaFuncion (ListaDeParametrosSeparadosPorComas) //  Llamada a la función como tal
```

El nombre de la función especificada en la llamada a la función debe ser el mismo que el nombre de la función que desea llamar para su ejecución. Los parámetros en la lista están separados por comas. La cantidad de parámetros a ser pasados a la función está limitada y no puede ser superior a 64. En una llamada a una función se pueden utilizar como parametros constantes, variables y llamadas a otras funciones. La cantidad, tipo y orden de los parámetros pasados en la llamada a una función debe ser la misma que la cantidad, tipo y orden de los parámetros especificados en la descripción de una función (la excepción es una llamada a una función con los parámetros por defecto).

```
My_function (Alf, Bet)          // Example of a function call
                                // Here:
My_function                     // Name of the called function
Alf                             // First passed parameter
Bet                             // Second passed parameter
```

Si la Llamada a la función no lleva parámetros de paso, la lista de parámetros se especifica como vacía, pero el paréntesis debe estar presente, de todos modos.

```
My_function ()                  // Exemplary function call
                                // Here:
My_function                     // Name of the called function
                                // There are no parameters to be passed
```

Si el programa debe llamar a una función con los parámetros por defecto, la lista de los parámetros pasado puede ser limitada (abreviada). Puede limitar la lista de parámetros, empezando con el primer parámetro por defecto. En el siguiente ejemplo, las variables locales b, c y d tienen algunos valores por defecto:

```
                                                  // For the function described as:
int My_function (int a, bool b=true, int c=1, double d=0.5)
  {
  Operators
  }
                                                  // .. the following calls are allowed:
My_function (Alf, Bet, Ham, Del)                  // Allowed function call
My_function (Alf )                        // Allowed function call
My_function (3)                           // Allowed function call
My_function (Alf, 0)                      // Allowed function call
My_function (3, Tet)                      // Allowed function call
My_function (17, Bet, 3)                  // Allowed function call
My_function (17, Bet, 3, 0.5)                     // Allowed function call
```

Los parámetros sin valores por defecto no se pueden omitir. Si un parámetro por defecto se salta, los parámetros por defecto subsiguientes no deben ser especificados.

```
                                  // .. Las siguientes llamadas no están permitidas:

 My_function ()                   // No permitida la llamada por defecto a la función. Falta..
                                  // .. 1º parámetro que no puede ser omitido.
My_function (17, Bet,, 0,5)  // No permitida la llamada a la función: se omite ..
                                  // .. parámetro por defecto (el tercero)


                                  // La función esta descrita como:
int My_function (int a, b bool = true, int c = 1, doble d = 0,5)
{
Operadores
}
```

La llamadas a las funciones se dividen en dos grupos: las que devuelven un valor predefinido de un tipo y los que no devuelven ningún valor.

Formato de llamada a la función sin return

Si la llamada a la función no devuelve ningún valor sólo puede ser compuesta como un operador independiente. La llamada a una función con operador sin return termina en "**;**" (punto y coma):

```
 Function_name (Parameter_list);    // Llamada a la función con operador sin return
 Func_no_ret (alfa, beta, gamma);           // Ejemplo de un operador de llamada a una ..
                                            // ..  función que no devuelve ningún valor
```

Ningún otro formato ó tecnica, se ofrece para llamar a funciones que no devuelven ningún valor.

Formato de llamada a función con return.

Una llamada a una función que devuelve un valor puede estar compuesta como un operador separado o puede ser utilizada en el código de programa en lugares donde un valor de un determinado tipo esta implicado.

Si la llamada a la función se compone de un operador independiente, este termina en ";" (punto y coma)

```
Function_name (Parameter_list);          // Llamada a la función con devolucion de valor
Func_yes_ret (Alpha, Beta, Delta);       // Ejemplo de operador llamada a una función..
                                         // .. para una función que devuelve un valor
```

Regla de Ejecución llamada a función



Una llamada a la función llama a la función del mismo nombre para su ejecución. Si la llamada a la función esta compuesta como un operador separado, después de que la función ha sido ejecutada, el control se pasa al operador que sigue la llamada a la función. Si la llamada a la función se utiliza en una expresión, después de que la función se ha ejecutado, el control se pasa a la ubicación en la expresión donde la la llamada a la función ha sido especificada y los demás cálculos se llevan a cabo en la expresión utilizando el valor devuelto por la llamada a la función.

El uso de llamadas a funciones en otros operadores está determinado por el formato de estos operadores.



Problema 20. Redactar un programa con las siguientes condiciones:
-- Si la hora actual es mayor de las 15:00, ejecutar 10 repeticiones en el ciclo 'for';
-- En todos los demás casos, ejecutar 6 iteraciones.

A continuación se muestra un ejemplo de script callfunction.mq4 que incluye: una llamada a una función en la cabecera del operador 'for' (como parte de Expression_1, de acuerdo al formato del operador 'for', véase el Operador de ciclo 'for'), una regla de llamada a función como un operador, en la parte derecha del operador de asignación (véase el operador de asignación), y en la cabecera del operador "if-else" (en la condición, de acuerdo al formato del operador "if-else", véase el operador condicional 'if- else").

```
///------------------------------------------------------------------------------------
// callfunction.mq4
// The code should be used for educational purpose only.
//------------------------------------------------------------------------------------
int start()                                          // Description of function start()
  {                                                  // Start of the function start() body
   int n;                                            // Variable declaration
   int T=15;                                         // Predefined time
   for(int i=Func_yes_ret(T);i<=10;i++)              // The use of the function in..
                                                     // The cycle operator header
    {                                                // Start of the cycle 'for' body
     n=n+1;                                          // Iterations counter
     Alert ("Iteration n=",n," i=",i);               // Function call operator
    }                                                // End of the cycle 'for' body
   return;                                           // Exit function start()
  }                                                  // End of the function start() body
//------------------------------------------------------------------------------------
int Func_yes_ret (int Times_in)                      // Description of the user-defined function
  {                                                  // Start of the user-defined function body
   datetime T_cur=TimeCurrent();                     // The use of the function in..
                                                     // ..the assignment operator
   if(TimeHour(T_cur) > Times_in)                    // The use of the function in..
                                                     //..the header of the operator 'if-else'
     return(1);                                      // Return value 1
   return(5);                                        // Return value 5
  }                                                  // End of the user-defined function body
//------------------------------------------------------------------------------------
```

En el ejemplo anterior, fueron llamadas las siguientes funciones con los siguientes parámetros transferidos:

- llamada a la función Func_yes_ret (T) - variable T; (Es función definida por el usuario)
- llamada a la función de Alert () – constantes de tipo string "Iteración n =" e "i =", variables n e i;
- llamada a la función TimeCurrent () sin parámetros de paso;
- llamada a la función TimeHour (T_cur) - T_cur variable.

En este programa se ejecuta un algoritmo simple. Hemos establecido en la variable T el tiempo (en horas), en relación a que los cálculos se ha realizar. En la cabecera del operador 'for', especificar la llamada a la función definida por el usuario Func_yes_ret () que puede devolver uno de dos valores: 1 o 5. De acuerdo con este valor, la cantidad de iteraciones en el ciclo cambiará: no habrá ni 10 (i cambios de 1 a 10) o 6 (i cambios de 5 a 10) iteraciones. Para una mejor visualización, en el cuerpo del bucle utilizamos el contador de iteración, cada valor de los cuales se visualiza en la pantalla utilizando la función Alert ().

En la descripción de la función definida por el usuario, lo primero que se calcular es el tiempo en segundos transcurrido después de las 00:00 de 1 de enero de 1970 (llamada a la función TimeCurrent ()), y luego calcular la hora actual en horas (llamada a la función TimeHour ()). El algoritmo es ramificado usando el operador 'if' (la llamada a la función TimeHour () se especifica en su condición). Si la hora actual resulta ser mayor que la pasada a la función definida por el usuario (variable local Times_in), esta última devuelve 1, en caso distinto devuelve 5.

Tengase en cuenta que:



No hay en el programa descripciones de funciones estándar o llamada a función para la función start ().

A continuación se puede ver el diagrama de bloques del script callfunction.mq4:

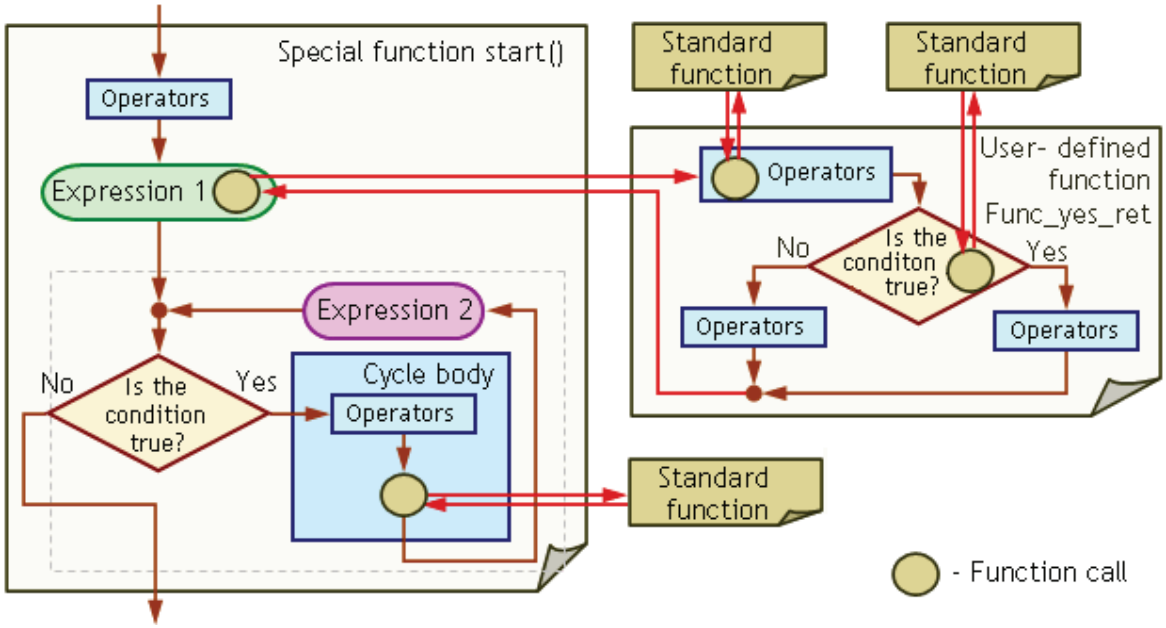


Fig. 51. Diagrama de bloques de un programa que utiliza llamadas a funciones.

Los círculos en el diagrama marcan las llamadas a funciones (para la función estandar y la función definida por el usuario). Flechas rojas indican el paso del control a la función y viceversa. Se puede ver claramente que la función devuelve el control a la ubicación en donde se especificó la llamada a la función y no devuelve los cálculos que se están realizando en el camino entre la llamada a la función y la función en sí. En general, si una función devuelve un valor, este valor se pasa al módulo de llamadas (a lo largo de la flecha roja en la dirección de la llamada a la función).

Las funciones especiales se pueden llamar desde cualquier lugar en el programa de acuerdo a las normas generales, al igual que cualesquiera otras funciones. Las funciones especiales también pueden tener parámetros. Sin embargo, cuando el Terminal de Usuario llama a estas funciones especiales, los parámetros no se transmitirán desde el exterior, esta función utilizará los valores por defecto. El uso de parámetros en funciones especiales sólo será razonable si se les llama desde un programa. A pesar de que es tecnicamente posible en MQL4 llamar a funciones especiales desde un programa, no se recomienda hacerlo. Un programa que utiliza llamadas a funciones especiales debe ser considerado como incorrecto.

Descripción y funcion del operador "return"

La necesidad de especificación de funciones dentro de un programa podemos dividirlas en dos grupos: Las funciones que no se describen en el programa, y las funciones que deben estar descritas en el programa. Las funciones estandar no se describen en los programas. Las funciones definidas por el usuario deben siempre describirse en el programa. Las funciones especiales, si las hubiere, deben también describirse en el programa.

Formato de la descripción de funciones

La descripción de una función consta básicamente de dos partes: cabecera de la función y cuerpo de la función.

La cabecera de la función contiene el tipo del valor de return, el nombre de la función, y la lista de parámetros entre paréntesis. Si una función no debe devolver ningún valor, su tipo denominarse en la cabecera ha de ser de tipo void (*vacío).*

El cuerpo de la función puede consistir en operadores simples y/o compuestos o por llamadas a otras funciones, y encerradas entre llaves.

```
TipoValorDeReturn NombreDeLaFuncion (Lista de parámetros formales)          // Encabezado

 {                                                  // Apertura de llave. Inicio del cuerpo de la funcion
 Código del Programa                                // El cuerpo de la función puede estar..
 que componen                                       // ..  formado de operadores y ..
 el cuerpo de la función                            // ..  de llamadas a otras funciones
 }                                                  // Cierre de llave y fin del cuerpo de la funcion
```

La lista de parámetros se escribe separada por comas. El número de parámetros a transferir a la función esta limitado a 64. Los parámetros formales de la cabecera de la función solo pueden ser especificados en forma de variables y no como constantes, ni expresiones, ni en forma de llamada a otras funciones). La cantidad, tipo y orden de los parámetros transferidos en la llamada a la función deben ser los mismos que los de los parámetros formales especificados en la descripción de la función (con excepción de la llamada a una función con parámetros que tienen valores por defecto):

```
int My_function (int a, double b)       // Example of function description
  {
   int c = a * b + 3;                             // Function body operator
   return (c);                          // Function exit operator
  }
                                                  // Here (from left to right in the header):
   int                                  // Return value type
   My_function                                    // Function name
   int a                                // First formal parameter a of the int type
  double b                              // Second formal parameter b of the double type
```

Los parámetros transferidos a la función pueden tener valores por defecto que se definen por una constante del tipo correspondiente:

```
int My_function (int a, bool b=true, int c=1, double d=0.5)//Example of function description
  {
   a = a + b*c + d2;                    // Function body operator
   int k = a * 3;                       // Function body operator
   return (k);                  // Function exit operator
  }

   // Here (from left to right in the header):

   int                          // Return value type
   My_function                          // Function name
   int a                        // 1º formal parameter (variable) a of the int type
   bool b                       // 2º formal parameter (variable)  b of the double type
   true                         // Constant, the default value for b
   int c                        // Third formal parameter (variable) c of the int type
   1                            // Constant, the default value for c
   double d                     // Fourth formal parameter (variable) d of the double type
   0.5                          // Constant, the default value for d

   a,b,c,d,k                    // Local variables
```

Si existen parámetros que figuran en la llamada a la función y que ellos mismos contienen valores por defecto en los parámetros formales de la descripción de la función, los parámetros que utilizará la función en sus cálculos serán aquellos que figuren en la llamada a la función. Si no existieran parámetros que figuren en la llamada a la función y que ellos mismos contienen valores por defecto en los parámetros formales de la descripción de la función, la función será calculada con los correspondientes valores por defecto.

Las funciones especiales también pueden tener parámetros. Sin embargo, el Terminal de Usuario no pasa parámetros desde el exterior en la llamada a estas funciones, sólo utiliza los valores por defecto. Las funciones especiales se pueden llamar desde cualquier lugar del módulo de acuerdo a las normas generales, al igual que cualquiera otras funciones.

Reglas de Ejecución de función

Ubicación de la descripción de una función de un programa:



La ubicación de la descripción de una función en un programa debe ser tal que esté separada de cualesquiera otras funciones, es decir, la función no debe estar nunca situada dentro de otra función.

Ejecución de la función:



La llamada a una función implica que ésta se ejecutará según el código del que está compuesto el cuerpo de dicha función.

Formato del operador «return»

El valor devuelto por la función es el valor del parámetro que se especifica entre los paréntesis del operador «return». El operador «return» se compone de la palabra clave «return» y la expresión contenida entre paréntesis, y termina con el carácter "**;**" (punto y coma). El formato completo operador «return» es:

```
return (expresión);                                    // Operador de retorno
```

La expresión entre paréntesis puede ser una constante, una variable o una llamada a una función. El tipo del valor devuelto utilizando en el operador 'return' debe ser el mismo que el tipo del valor de la función que se especifica en función de la cabecera de la llamada a la función. Si este no es el caso, el valor de la expresión especificada en el operador 'return' debe ser emitidos al tipo del valor de return que se especifica en la cabecera de la descripción de la función. Si el typecasting es imposible, MetaEditor le dará un mensaje de error al compilar su programa.

Estado de Ejecución del operador «return»



El operador «return» detiene la ejecución de la función exterior más cercana y pasa el control al lugar de llamada del programa de acuerdo a las reglas definidas para una llamada a una función. El valor devuelto por la función es el valor de la expresión especificada en el operador «return». Si el tipo de parámetro del valor especificado en el operador de 'return' es diferente a la del valor de return que se especifica en la cabecera de la llamada a la función, el valor debe ser emitido al tipo del valor de return que se especifica en la cabecera de la descripción de la función.

Un ejemplo de cómo usar el operador "return" que devuelve un valor:

```
 bool My_function (int Alpha)                    // Description of the user-defined function
{                                                // Start of the function body
 if(Alpha>0)                                     // Operator 'if'
   {                                             // Start of the operator-'if' body
    Alert("The value is positive");        // Standard function call
    return (true);                               // First exit from the function
   }                                             // End of the operator-'if' body
 return (false);                                 // Second exit from the function
}                                                // End of the function body
```

Si el valor de return de una función es de tipo *void (vacio),* el operador "return" se debe usar sin expresión:

```
return;              // Operador de «return» sin expresiones entre paréntesis
```

Ejemplo de uso del operador «return» sin valor de retorno:

```
 void My_function (double Price_Sell)          // Description of the user-defined function
{                                                    // Start of the function body
 if(Price_Sell-Ask >100 * Point)                     // Operator 'if'
   Alert("Profit for this order exceeds 100 п");  // Standard function call
 return;                                         // Exit function
}                                                    // End of the function body
```

Existe la posibilidad de no incluir el operador "return" en la descripción de una función. En este caso, la función se dará por terminado su funcionamiento de forma automática, tan pronto como (de acuerdo con el algoritmo de ejecución) se ejecute el ultimo operador del cuerpo de la función. Ejemplo de la descripción de una función sin el operador «return»:

```
 void My_function (int Alpha)                  // Description of the user-defined function
{                                                    // Start of the function body
 for (int i=1; i<=Alpha; i++)              // Cycle operator
   {                                                 // Start of the cycle body
    int a = 2*i + 3;                               // Assignment operator
    Alert ("a=", a);                               // Standard function call operator
   }                                                 // End of the cycle body
}                                                    // End of the function body
```

En este caso, la función completa sus operaciones en el momento en que el ciclo del operador 'for' termina su ejecución. La última acción de la ejecución de la función que será la prueba de condición en el operador de ciclo. Tan pronto como la condición en la cabecera del operador de ciclo 'for' se convierte en falsa, el control pasará fuera del ciclo del operador (for). Sin embargo, debido a que el operador de ciclo es el último operador ejecutable en el cuerpo de la función denominada My_function(), la función definida por el usuario terminará sus operaciones y el control pasará fuera de la función, al lugar donde la función fue llamada para su ejecución.

Variables

Para la creación de programas en cualquier lenguaje algorítmico es muy importante conocer los diferentes tipos de variables. En esta sección vamos a analizar todos los tipos de variables que se utilizan en el lenguaje MQL4.

- Variables predefinidas y RefreshRates Función.
  Lo primero que debemos hacer es aprendernos los nombres de las variables predefinidas. Los nombres de las variables predefinidas son nombres reservados y no se pueden utilizar para crear variables personalizadas. Se trata de las variables predefinidas que contienen las principales informaciónes necesarias para el análisis de situación actual del mercado. Para actualizar esta información se utiliza **RefreshRates ().**

- Tipos de variables.
  Las variables son muy importantes en la redacción de un programa. Ellas se dividen en locales y globales, externas e internas. Las variables estaticas preservan sus valores entre llamadas a funciones, es útil recordar algunos valores de las variables locales sin crear variables globales.

- GlobalVariables.
  Al lado de las variables globales a nivel de programa separado cuyos valores están disponibles desde cualquier parte del programa, existen las variables globales a nivel del terminal. Estas variables globales son llamadas GlobalVariables. Permiten establecer una interacción entre partes independientes de distintos programas escritos en MQL4. Se pueden utilizar para el intercambio de valores entre scripts, indicadores y Asesores Expertos. Cuando se cierra el Terminal, los valore de las GlobalVariables también se conservan para estar disponibles en próximo arranque del MetaTrader 4. No hay que olvidar que, si no hay llamadas a una GlobalVariable durante 4 semanas, está será eliminada.

- Las matrices. (arrays)
  Si se necesita guardar o precesar una gran cantidad de valores de un tipo, esto no se puede hacer sin arrays. Estas serán declaradas como variables. La llamada a los elementos de un arrayse realiza a través del indice del elemento (index). La indexación de un Array comienza en cero. Número de dimensiones de un array se llama dimensionalidad. Las matrices de más de cuatro dimensiones no son aceptadas. Los valores del array deben estar claramente inicializado con el fin de evitar errores difíciles de localizar.

Variables predefinidas y función RefreshRates

Existen variables predefinidas con nombres del lenguaje MQL4.

Una **variable Predefinida** es una variable con un nombre predefinido, cuyo valor se define en un Terminal de Usuario y no se pueden cambiar mediante programa. Las variables predefinidas reflejan el estado actual de una gráfica en el momento de empezar el programa (Asesor Experto, scrip o indicador personalizado) o como resultado de la aplicación de la función RefreshRates ().

Lista de nombres de variables predefinidas simples

Ask -	último precio conocido de venta del actual titulo o valor;
Bid -	último precio conocido de compra del actual titulo o valor;
Bars -	número de barras en un gráfico actual;
Point -	Tamaño de punto del actual titulo o valor de la moneda cotizada;
Digits -	número de dígitos después de un punto decimal en el precio del actual titulo o valor.

Lista de nombres predefinidos de Arrays-Timeseries

Time - fecha y hora de apertura de cada barra en el gráfico actual;

Open - precio de apertura de cada barra en el gráfico actual;

Close - precio de cierre de cada barra en el gráfico actual;

High - precio máximo de cada barra en el gráfico actual;

Low - precio mínimo de cada barra en el gráfico actual;

Volume - marca el volumen de cada barra en el gráfico actual.

(los conceptos de "arrays" y "arrays-timeseries" serán descritos en la seccion de Arrays).

Características de las variables predefinidas

Los nombres de las variables predefinidas no pueden utilizarse para identificar variables definidas por el usuario. Las variables predefinidas se pueden utilizar en expresiones al igual que cualquier otra variable y de conformidad con las mismas reglas, pero el valor de una variable predefinida no se puede cambiar. Al intentar compilar un programa que contenga un operador de asignación, en el que una variable predefinida se coloca a la izquierda de un signo de igualdad, MetaEditor mostrará un mensaje de error. En términos de visibilidad las variables predefinidas tienen referencia a nivel global, es decir, están disponibles desde cualquier parte del programa (ver tipos de variables).

La propiedad más importante de variables predefinidas es la siguiente:



Los valores de todas las variables predefinidas se actualizan automáticamente en el Terminal de Usuario en el momento que se inician las funciones especiales para su ejecución.

El valor actual y anterior de una variable predefinida puede ser igual, aunque el valor en sí sea actualizado. Los valores de estas variables predefinidas están actualizados y disponibles a partir del momento que se inicia la primera linea de programa de la función especial. Vamos a ilustrar la actualización de variables predefinidas en el siguiente ejemplo (Asesor Experto predefined.mq4):

```
//+------------------------------------------------------------------+
// predefined.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                          // Special funct. start
  {
   Alert("Bid = ", Bid);                 // Current price
   return;                           // Exit start()
  }
//+------------------------------------------------------------------+
```

A partir de este programa es fácil ver que los valores de la variable del precio de compra (variable predefinida **Bid**) será igual al precio actual de cada momento. De la misma manera se puede comprobar los valores de otras variables, dependiendo de las condiciones actuales. Por ejemplo, la variable **Ask** (precio de venta) también depende del precio actual. El valor de la variable **Bars** (numero de barras en el grafico actual) también cambiará si el número de barras de cambió. Esto puede suceder en un tick, en la que una nueva barra se está formando en la ventana de un grafico. El valor de **Point** depende del título o valor especificado. Por ejemplo, para EUR/USD, este valor es 0.0001, para USD/JPY es 0,01. Valor de los dígitos de estos valores (variable **Digits**) es igual a 4 y 2, respectivamente.

Aquí tenemos otra propiedad importante de variables predefinidas:



El Terminal de Usuario crea un conjunto de copias locales de variables predefinidas por separado para cada programa que se inicia. Cada programa que se inició trabaja con sus propio juego de copias de los datos históricos.

En una Terminal de Usuario se pueden ejecutar varios programas de aplicación al mismo tiempo (asesores expertos, scripts ó indicadores), y para cada uno de ellos el Terminal de Usuario crea una copia con un juego de los datos históricos de todos los valores de las variables predefinidas. Vamos a analizar en detalles las razones de esta necesidad. La Fig. 52 muestra el posible funcionamiento de la ejecución de Asesores Expertos con diferentes longitudes de tiempo de la función especial start (). Por simplicidad vamos a suponer que en los Asesores Expertos analizados no hay otras funciones especiales y que Los Asesores Expertos de ambos operan en los mismos marcos temporales de un mismo símbolo.

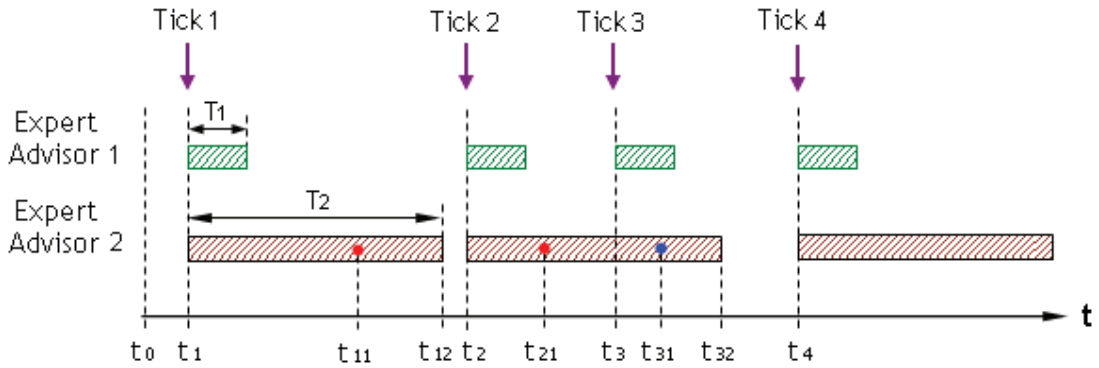


Fig. 52. El tiempo de operación de la función star () puede ser mayor o menor que un intervalo de tiempo entre los ticks.

Los Asesores Expertos difieren en el tiempo de ejecución de la función especial start() . Para un Asesor de Expertos común de nivel medio el tiempo de ejecución está aproximadamente entre 1 y 100 milisegundos. Otros Asesores Expertos pueden tener tiempos de ejecución mucho mayores, tiempos de, por ejemplo, varios segundos o decenas de segundos. El intervalo de tiempo entre los ticks también es diferente: desde milisegundos a varios minutos y, a veces, incluso decenas de minutos. En este ejemplo vamos a analizar cómo influye en el funcionamiento de los Asesores Expertos 1 y 2 la frecuencia de recepción de los ticks debido al los diferentes tiempos de ejecución de la función especial start() de cada uno de los Asesores Expertos.

En el momento momento t0 el Asesor Experto 1 se vincula al Terminal de Usuario y cambia al modo de espera de tick. En el momento t1 aparece un tick y el terminal inicia la función especial start () junto con el programa que le da acceso a la copia actualizada del juego de variables predefinidas. Durante el periodo de ejecución, el programa puede hacer referencia a estos valores que se mantendrán sin variación durante todo el tiempo de operación de la función especial start(). Cuando la función star() termina todas sus operaciones, el programa entra en el modo de espera de tick.

El momento siguiente más próximo en el que las variables predefinidas pueden obtener nuevos valores es a la llegada de un nuevo tick. El tiempo de ejecución T1 de start() del Experto Asesor 1 es considerablemente inferior a al tiempo de espera entre ticks, por ejemplo intervalo t 1 - t 2 o t 2-t 3, etc Por lo tanto, el tiempo de ejecución del Asesor Experto 1 analizado, en ningun momento caerá en la situación de que los valores de variables predefinidas antiguas estén caducadas, es decir, que difieran de su verdadero valor actual (último conocido).

En la operación del Asesor Experto 2, la situación es diferente porque el tiempo de ejecución de su funcion start(), periodo T2, a veces supera el intervalo entre ticks. La función start () del Asesor Experto 2 también comenzó en el momento t1. La Fig. 52 muestra que el intervalo t 1 - t 2 entre los ticks es más grande que el tiempo de ejecución de start(), periodo T2, es por eso que durante este período del programa, no se realiza la operación de actualización de las variables predefinidas (en este período los nuevos valores no proceden del servidor, por lo que sus valores son los valores que aparecieron en el momento t1).

La siguiente ocasión en que se inicia el Asesor Experto 2, lo hace en el momento t2 que es cuando se recibe el segundo tick. Junto con éste, se recibe la copia del conjunto de los valores actualizados de las variables predefinidas. En la Fig. 52 vemos que en el momento del tick t3 la función Stara() está todavía en ejecución. Una cuestión que se plantea es: ¿Cuáles serán los valores de las variables predefinidas disponibles para el Asesor Experto 2 en el período que va desde t3, cuando llega el tercer tick, hasta t32, que es cuando cuando start () termina su operación? La respuesta puede encontrarse en conformidad con la siguiente regla:



Las copias de los valores de variables predefinidas se guardan durante todo el período de operación de las funciones especiales. Estos valores pueden ser forzados a actualizarse usando la función estandar **RefreshRates ()**.

Por lo tanto (si RefreshRates () no ha sido ejecutada) durante todo el período de ejecución de start(), El Asesor Experto 2 tendrá acceso al conjunto de las copias locales de las variables predefinidas que se crearon cuando fue recibido el segundo tick. A pesar de que los Asesores Expertos operan en las mismas ventanas. A partir del momento t3 de la recepción de ticks, cada AE funcionará con diferentes valores de variables predefinidas. El Asesor Experto 1 trabajará con su propio conjunto de copias locales de datos históricos, valores que se definen en el momento t3, y el Asesor Experto 2 trabajará con sus propias copias de datos que son valores iguales al momento t2.



El tiempo de ejecución más largo que un programa de aplicación tiene y el menor tiempo del intervalo entre los ticks nos da la probabilidad es que el próximo tick llegará durante el período de ejecución del programa. El conjunto de copias locales de datos históricos establece las condiciones de cada programa que garantiza la constancia de variables predefinidas a través de todo el plazo de ejecución de una función especial.

A partir del momento t4 cuando llega el siguiente tick, ambas AEs se iniciará una vez más y cada uno de ellos tendrá acceso a su propia copia del conjunto de variables predefinidas, de los valores que se forman en el momento t4 que coincide con la llegada del cuarto tick.

RefreshRates ()

```
bool RefreshRates ()
```

La función estándar **RefreshRates ()** permite actualizar los valores locales de las copias datos históricos. En otras palabras, esta función fuerza la actualizacion de datos acerca de un entorno de mercado actual (volumen, servidor de tiempo de la última cotización Tiempo [0], Bid, Ask, etc.) Esta función puede utilizarse cuando un programa utiliza mucho tiempo en realizar sus cálculos y por lo tanto tiene necesidades de actualizar los datos.

RefreshRates () devuelve TRUE, si en el momento de su ejecución hay información sobre los nuevos datos históricos en la terminal (es decir, si ha llegado un nueva tick durante la ejecución del programa). En tal caso, el conjunto de copias locales de las variables predefinidas se actualizará.

RefreshRates () devuelve FALSE, si desde el momento del inicio de la ejecución de una función especial de los datos históricos el Terminal de Usuario no se han actualizado. En tal caso, las copias locales de las variables predefinidas no cambian.



Tenga en cuenta que RefreshRates () sólo influye en el programa en el que se ha iniciado (no todos los programas trabajan con el Terminal de Usuario al mismo tiempo).

Vamos a ilustrar la ejecución de RefreshRates () con un ejemplo.



Problema 21. Contar el número de iteraciones que un operador de ciclo puede realizar entre los ticks (el más cercano cinco ticks).

Este problema sólo puede resolverse mediante RefreshRates () (script countiter.mq4):

```
//------------------------------------------------------------------
// countiter.mq4
// The code should be used for educational purpose only.
//------------------------------------------------------------------
int start()                                   // Special funct. start()
  {
   int i, Count;                              // Declaring variables
   for (i=1; i<=5; i++)                  // Show for 5 ticks
     {
      Count=0;                                // Clearing counter
      while(RefreshRates()==false)            // Until...
        {                                     //..a new tick comes
         Count = Count+1;                 // Iteration counter
        }
      Alert("Tick ",i,", loops ",Count);      // After each tick
     }
   return;                                    // Exit start()
  }
//------------------------------------------------------------------
```

De acuerdo con las condiciones del problema, los cálculos deben hacerse sólo para los cinco ticks más próximos, que es la razón por la que podemos usar un script. Dos variables se utilizan en el programa: "i" para contar el número de ticks y "Count" para contar las repeticiones. El ciclo externo for está organizado en función del número de ticks procesados (de 1 a 5). En el ciclo for se inicializa el contador de iteraciones poniendolo a cero (realizado en el ciclo while), al final se muestra un Alert () con el número de tick y la cantidad de sus iteraciones.

El ciclo I nterior 'while' va a operar mientras el valor devuelto por RefreshRates () sea igual a false, es decir, hasta que se marque un nuevo tick. Durante la operación de 'while', es decir, en el intervalo entre los ticks, el valor del contador Count irá incrementandose constantemente, por lo que se contará el número de iteraciones del ciclo 'while'. Si en el momento del control de la condición de 'while' el valor devuelto por RefreshRates () es «true», significará que hay nuevos valores de variables predefinidas en el Terminal de Usuario, debido a la llegada de un nuevo tick. Como resultado el control se devuelve de nuevo fuera de 'while' y así termina el conaje de iteraciones.

Como resultado de la ejecución del script countiter.mq4 un número de alertas caracteristico de la ejecución de MQL4 aparecerán en la ventana del símbolo:

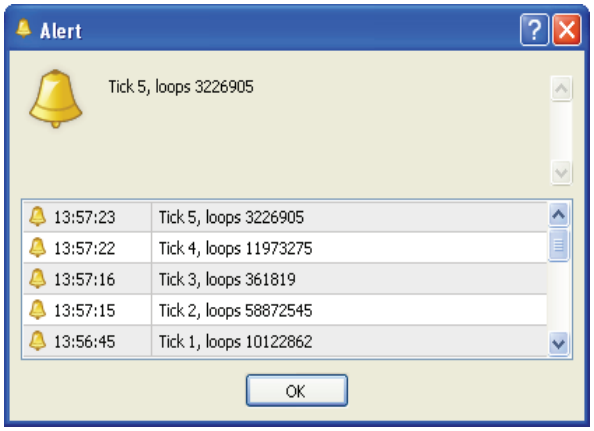
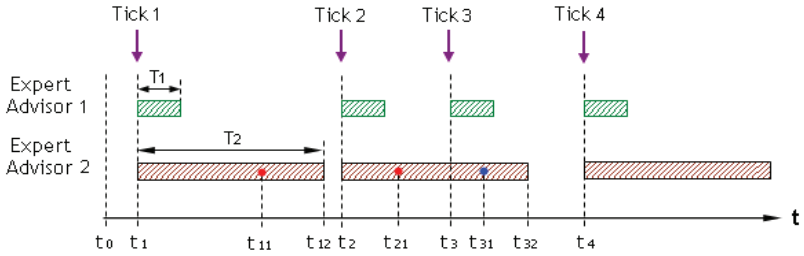


Fig. 53. Resultados de la operación de countiter.mq4 en la ventana EUR / USD.

Es fácil ver que durante 1 segundo (intervalo entre el cuarto y el quinto ticks) el programa ha realizado más de 3 millones de iteraciones. Análogos resultados pueden ser obtenidos con los simples cálculos para otros los ticks.

Vamos a volver al ejemplo anterior (Asesor Experto predefined.mq4). Anteriormente vimos que si RefreshRates () no se ejecuta en el Asesor Experto 2, los valores de las copias locales de las variables predefinidas, se mantienen sin variación durante todo el período de la ejecución de start (), como por ejemplo durante el período t 2 - t 32. Si después del tercer tick (que viene cuando start () se está ejecutando) se ejecuta la función RefreshRates (), por ejemplo en el momento t31, los valores locales de las copias serán actualizadas. Así, durante el tiempo restante a partir de t31 (ejecución de RefreshRates ()) a t32 (fin de ejecución de start ()), los nuevos valores de las copias de la variables locales predefinidas iguales a los valores definidos por el terminal de usuario en t3 estará disponible para el Asesor Experto 2.

Si en el Asesor Experto 2, RefreshRates se ejecuta en el momento t 11 o t21 (es decir, en el período en que el último tick es el que ha iniciado la ejecución de start ()), las copias locales de las variables predefinidas no serán cambiadas (por que son las mismas). En tales casos, los valores actuales de las copias locales de las variables predefinidas serán iguales a la última conocida, es decir, a aquellas que fueron definidas por el Terminal de Usuario en el momento de la última salida de la función especial start ().



Tipos de variables

Un programa de aplicación en MQL4 puede contener decenas y cientos de variables. Una parte muy importante de la caracteristica de las variables es la posibilidad de utilizar su valor en un programa. La limitación de esta posibilidad está relacionada con el alcance o ámbito de la variable.

El ámbito de una Variable es la ubicación en un programa donde el valor de la variable está disponible. Cada variable tiene su propio ámbito de aplicación. De acuerdo con el alcance hay dos tipos de variables en MQL4: **variables locales** y **variables globales**.

Variables locales y globales

La **Variable local** es una variable declarada dentro de una función. El alcance de las variables locales es el cuerpo de la función, en el que la variable ha sido declarada. La Variable local puede ser inicializada por una constante o una expresión correspondiente a su tipo.

La **Variable global** es una variable declarada más allá de las funciones. El alcance de las variables globales es el programa entero. Una variable global puede ser inicializada sólo por una constante del mismo tipo, pero no puede ser inicializada por una expresión. Las variables globales se inicializan sólo una vez antes de la declaración de la ejecución de funciones especiales.

Si el control en un programa se encuentra dentro de una determinada función, los valores de las variables locales declaradas en otra función no están disponibles. El valor de cualquier variable global está disponible desde cualquier función especial y funciones definidas por el usuario.

Vamos a ver un ejemplo sencillo.



Problema 22. Crear un programa que cuenta los ticks.

Solución de problemas algoritmo de 22 utilizando una variable global (countticks.mq4):

```
//+------------------------------------------------------------------+
// countticks.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int Tick;                        // Global variable (declarada antes de la descripcion de start()
//+------------------------------------------------------------------+
int start()                                 // Special function start()
  {
   Tick++;                                          // Tick counter
   Comment("Received: tick Nº ",Tick);  // Alert that contains number
   return;                                  // start() exit operator
  }
//+------------------------------------------------------------------+
```

En este programa sólo se utiliza una variable global "Tick". Es global, porque ha sido declarada fuera de la descripción de la función start (). Esto significa que la variable conserva su valor de un tick a otro. Vamos a ver los detalles de la ejecución del programa.

En Funciones especiales se analizaron los criterios del inicio de las funciones especiales. En pocas palabras: la función start () del Asesor Experto se inicia en el Terminal de Usuario cuando llega un nuevo tick. En el momento que el Asesor Experto se asocia a una ventana del símbolo de un titulo, se llevará a cabo lo siguiente:

1. Declaración de la variable global Tick. Esta variable no ha sido inicializada por una constante, es por eso que su valor en esta etapa es igual a cero.

2. El Terminal de Usuario mantiene el control hasta que llegue un nuevo tick.

3. Se recibe un tick. Se pasa el control a la función especial start ().

3,1. Dentro de start () Se pasa el control de ejecución al operador:

```
Tick++;                                 // Contador de ticks
```

Como resultado de la ejecución el operador Tick incrementa su valor en 1 (un entero).

3,2. Se pasa el control al operador:

```
Comment("received: tick Nº ",Tick);   // alerta que contiene el número de tick
```

La Ejecución de la función estándar Comment () provocará la aparición de la descripción:

```
Received: tick Nº 1 (Recibido: tick n º 1)
```

3,3. El control se pasa al operador:

```
return;                          // operado salir de start ()
```

Como resultado la ejecución start () termina su operacióln y el control se devuelve al Terminal de Usuario. La variable global sigue vigente, su valor es igual a 1.

Las misma acciones se repetiran a partir del punto 2. La variable tick se utilizará en los cálculos de nuevo, pero en el segundo tick, su valor es igual a 1 en el momento de iniciar la función especial start (). Por eso el resultado de la ejecución del operador…

```
Tick++;                                 // Contador de ticks
```

resultará en un nuevo valor de la variable Tick, que se incrementará en 1 y ahora será igual a 2. La ejecución de Comment () mostrará la alerta:

```
Received: tick Nº 2 (Recibido: tick n º 2)
```

Por tanto, el valor de Tick se incrementará en 1 en cada salida de la función especial start (), es decir, en cada tick. La solución de este tipo de problemas sólo es posible con el uso de variables que preserven sus valores después de salir de una función (en estos casos se utiiza una variable global). No es viable utilizar variables locales para este fin: una variable local que se declara en una función, pierde su valor al final de sus operaciones.

Se puede ver fácilmente que si iniciamos un Asesor Experto en el cual la variable Tick se abre como una variable local el programa contendrá un error algorítmico:

```
int start()                                 // Special function start()
 {
  int Tick;                                         // Local variable
  Tick++;                                           // Tick counter
  Comment("Received: tick Nº ",Tick);  // Alert that contains number
  return;                                   // start() exit operator
  }
```

Desde el punto de vista de la sintaxis no hay errores en el código. Este programa puede ser compilado con éxito y empezar su funcionamiento, pero cada vez que se ejecute el resultado será siempre el mismo:

```
Received: tick Nº 1 (Recibido: tick n º 1)
```

Es natural, porque la variable Tick se inicializa a cero al comienzo de la función especial start () cada vez que esta comienza. El incremento en uno de la variable da como resultado siempre 1, por eso la alerta mostrará siempre el tick numero 1.

Variables Static

En el nivel físico las variables locales se presentan en una memoria temporal como parte de la correspondiente función. Hay una manera de localizar una variable declarada dentro de una función en una memoria permanente del programa. El modificador "stático" debe indicarse antes del tipo de variable en su declaración:

```
static int Número;                              // variable Static de tipo integer
```

La caracteristica particular de la variable static es que no pierde su valor al salida de la función donde esta ubicada, sin embargo, al contrario que la variable global, su ámbito de aplicación está limitado dentro de la función donde ha sido declarada

A continuación se muestra la solución del problema 22 utilizando una variable estática (Asesor Experto staticvar.mq4):

```
   //------------------------------------------------------------------
// staticvar.mq4
// The code should be used for educational purpose only.
//------------------------------------------------------------------
int start()                                // Special function start()
  {
   static int Tick;                        // Static local variable
   Tick++;                                          // Tick counter
   Comment("Received: tick No ",Tick);  // Alert that contains number
   return;                                 // start() exit operator
  }
//------------------------------------------------------------------
```

Las variables static se inicializan solo una vez y solo pueden ser inicializadas por una constante (a diferencia de una simple variable local que se pueda inicializar con cualquier expresión). Si no hay inicialización explícita, la variable se inicializa a cero. La variable static se almacena en una memoria permanente y su valor no se pierde al salir de una función. Sin embargo, respecto al ámbito de aplicación, las variables static tienen las limitaciones típicas de las variables locales, el alcance de la variable static es la función dentro de las cual ha sido declarada, a diferencia de las variables globales cuyos valores están disponibles desde cualquier parte del programa. Vease como los programas countticks.mq4 y staticvar.mq4 dan el mismo resultado.

Todos los arrays tiene inicialización estática, es decir, son de tipo estático, aunque no esté explícitamente indicado (véase Arrays).

Variables externas

La variable externa es una variable cuyo valor esta disponible desde la ventana de propiedades de un programa. Una variable externa se declara fuera de cualquier función y es una variable global, su ámbito de aplicación es todo el programa. Cuando se declara una variable externa, el modificador "extern" debe indicarse antes del tipo de valor:

```
extern int Número                    // variable externa de tipo integer
```

Las variables externas se especifican en la parte de la cabecera del programa, es decir, antes de cualquier función que contenga una llamada a función externa. El uso de variables externas es muy conveniente si se necesita iniciar de vez en cuando un programa con distintos valores de variables.



Problema 23. Crear un programa, en el que se apliquen las siguientes condiciones: si el precio alcanza un cierto nivel, y bajó de ese nivel n puntos, este hecho debe ser reportado al trader.

Obviamente, este problema implica la necesidad de cambiar la configuración, ya que al día de hoy los precios difieren de los que fueron ayer, así como mañana vamos a tener precios diferentes. Para ofrecer la opción de cambiar la configuración en el Asesor Experto externvar.mq4 se utilizan variables externas:

```
//+------------------------------------------------------------------+
// externvar.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
extern double Level = 1.2500;                          // External variable
extern int       n = 5;                          // External variable
      bool  Fact_1 = false;                      // Global variable
      bool  Fact_2 = false;                      // Global variable
//+------------------------------------------------------------------+
int start()                                            // Special function start()
  {
   double Price = Bid;                           // Local variable
   if (Fact_2==true)                             // If there was an Alert..
     return;                                             //..exit

   if (NormalizeDouble(Price,Digits) >= NormalizeDouble(Level,Digits))
     Fact_1 = true;                              // Event 1 happened

   if (Fact_1 == true && NormalizeDouble(Price,Digits)<=
                   NormalizeDouble(Level-n*Point,Digits))
     My_Alert();                                 // User-defined function call

   return;                                             // Exit start()
  }
//+------------------------------------------------------------------+
void My_Alert()                                // User-defined function
  {
   Alert("Conditions implemented");                    // Alert
   Fact_2 = true;                              // Event 2 happened
   return;                                             // Exit user-defined function
  }
//+------------------------------------------------------------------+
```

En este programa las variables externas se crean en las líneas:

```
extern double Level = 1.2500;              // External variable
extern int       n = 1;                    // External variable
```

Los valores de las variables externas están disponibles en la ventana de los parámetros del programa. Lo mejor de estas variables es que pueden ser cambiadas en cualquier momento. En la fase de activación del programa en una ventana de símbolo o título, o durante la operación del programa.

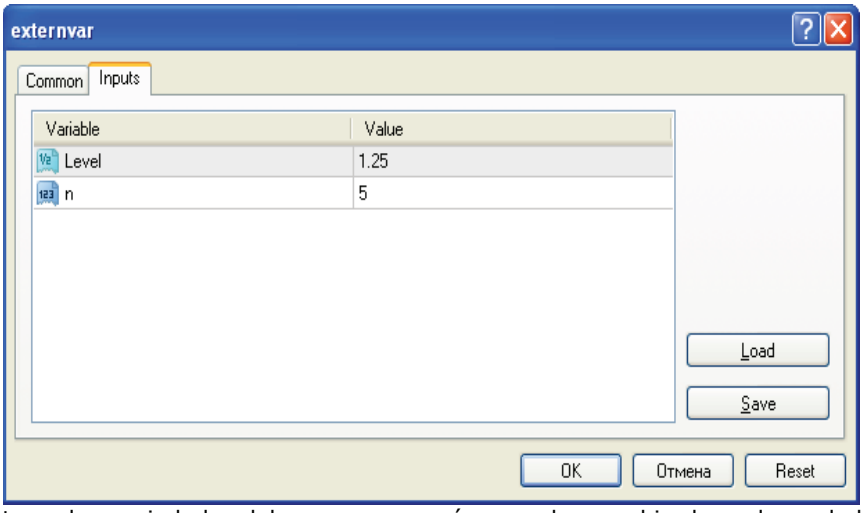


Fig. 54. Ventana de propiedades del programa; aquí se pueden cambiar los valores de las variables.

En el momento de conectar el programa a una ventana de propiedades de un título, los valores de las variables contenidas en el código de programa se mostrarán en los parámetros de la ventana del programa. El usuario puede modificar estos valores. Desde el momento en que un usuario hace clic en Aceptar, el programa iniciará el Terminal de Usuario. Los valores de las variables externas serán los indicados por el usuario. En el proceso de operación de estos valores pueden ser cambiados por el programa que se ejecuta.

Si un usuario necesita cambiar los valores de las variables externas durante la operación, para poder hacerse los cambios, debe estar abierta la ventana de configuración del programa. Hay que recordar que las propiedades de la barra de herramientas del programa se pueden abrir solamente en el período en que el programa (Asesor Experto o indicador) está a la espera de un nuevo tick, es decir, no se está ejecutando ninguna de las funciones especiales. Durante el periodo de ejecución del programa, la barra de herramientas (toolbar) no se puede abrir. Es por ello que si un programa está escrito así, y su tiempo de ejecución es largo (unos segundos o decenas de segundos), un usuario pueden tener dificultades tratando de acceder a la ventana de parámetros. Los valores de las variables externas de un scripts sólo están disponibles en el momento de conectar el programa a un gráfico, pero no se puede cambiar durante la operación. Si la ventana de parámetros está abierta el Asesor Experto no funciona, el control se realiza mediante el Terminal de Usuario y no pasa a un programa para iniciar la función especial.



Tengase en cuenta, que cuando una ventana de propiedades de un AE está abierta y un usuario está modificando los valores de las variables externas, la AE (o indicador) no funciona. Después que el usuario ha establecido los valores de las variables externas y ha hecho clic en Aceptar, el programa se inicia una vez más.

A continuación el Terminal de Usuario inicia la ejecución de la función especial deinit(), luego la función especial init () y después, cuando llega un nuevo tick la funcion start(). En la ejecución de deinit() que se ocupa de terminar un programa que se ha desconectado de la ventana del grafico de un símbolo o valor, las variables externas tendrán los valores de la sesión anterior, es decir, los que se disponía antes de que se activara la barra de herramientas cuando se abrió el AE. Antes de la ejecución de init() las variables externas obtendrá los valores de la configuración del usuario en la barra de herramientas y con la ejecución la función deinit () las variables externas establecen nuevos valores configurados por el usuario. Por lo tanto los nuevos valores de las variables externas se aplican desde el momento que comienza una nueva sesión (init - start - deinit) desde un Asesor Experto, a partir de la ejecución de init().

El hecho de abrir una ventana de configuración, no influye en los valores de las variables globales. Durante todo el tiempo cuando la ventana está abierta y después se cierra, las variables globales preservan sus valores que han sido válidos hasta el momento anterior a la apertura de la barra de herramientas.

En el programa externvar.mq4 se usan también dos variables globales y una variable local.

```
bool  Fact_1 = false;                        // Global variable
bool  Fact_2 = false;                        // Global variable
double Price = Bid;                    // Local variable
```

Algoritmicamente la solución del problema tiene este aspecto. Se identifican dos eventos: el primero es el hecho de llegar a un nivel; el segundo, el hecho de que se muestra una alerta de que se tiene un nivel mas bajo que el nivel establecido menos n puntos. Estos hechos se reflejan en los valores de las variables Fact_1 y Fact_2: si el caso no fuera así, el valor de los correspondientes valores sería igual a false y en caso contrario sería true. En las líneas:

```
if (NormalizeDouble(Price,Digits) >= NormalizeDouble(Level,Digits))
Fact_1 = true;                       // Event 1 happened
```

Queda definido el hecho de que ha sucedido el primer evento. La función estándar NormalizeDouble () permite realizar cálculos con los valores de las variables reales a un conjunto de valores exactos (que corresponde a la exactitud de los precios del título o valor). Si el precio es igual o superior al nivel indicado, el hecho del primer evento se considerará que se cumple y la variable global Fact_1 obtiene el valor true. El programa está construido de manera que una vez que Fact_1 obtiene el valor true, nunca será cambiado al valor de falso por que no hay un código escrito en el programa que lo haga.

En las líneas:

```
if (Fact_1 == true && NormalizeDouble(Price,Digits)<=
                  NormalizeDouble(Level-n*Point,Digits))
    My_Alert();                          // User-defined function call
```

Se define la necesidad de mostrar un aviso. Si el primer hecho se ha completado y se redujo el precio en n puntos (menor o igual) del nivel indicado, una alerta será mostrada debida a la llamada a la función definida por el usuario My_Alert (). En esta función, después de que la descripción del hecho ya se ha mostrado, se asigna true a la variable Fact_2, lo cual permite, después de salir de la función definida por el usuario, salir de la función especial start ().

Después de que la variable Fact_2 obtiene el valor true, el programa (funcion especial Start()) dará por acabado su funcionamiento cada vez que se éste se ejecute. Por eso, una vez mostrada la alerta no se repetirá este programa durante esa sesión:

```
if (Fact_2==true)                        // If there was an Alert..
    return;                             //..exit
```

En este programa el hecho significativo es que los valores de las **variables globales** puede ser modificadas en cualquier lugar (tanto en la función especial como en las funciones definidas por el usuario) y que se conserven en todo el período de operación del programa, tanto en el período comprendido entre ticks como después de cambiar la variable exterior, o después de cambiar un marco temporal.

En general los valores de variables globales pueden ser modificados en cualquier función especial. Es por ello que uno debe estar muy atento cuando se indique a los operadores que cambien los valores de variables globales en init () y deinit (). Por ejemplo, si ponemos a cero el valor de una variable **global** en init (), en la primera start () la ejecución el valor de esta variable se hace igual a cero, el valor adquirido durante la ejecución del anterior start () se perderá.

Variables Globales del Terminal de Usuario (GlobalVariables)

Varios programas de aplicación pueden funcionar en el Terminal de Usuario al mismo tiempo. En algunos casos de necesidad puede ocurrir que se pasen algunos datos de un programa a otro. Especialmente si este MQL4 tiene variables globales de la Terminal de Usuario.

La variable global del Terminal de Usuario es una variable, cuyo valor está disponible en todos los programas de aplicación que se inicien en la Terminal de Usuario (forma abreviada: TU).



Hay que tener en cuenta que las **Variables Globales del Terminal de Usuario (VGTU)** y las **Variables Globales** (a secas) son diferentes variables con nombres similares. El alcance de las variables globales es el programa donde se declara la variable, mientras que el alcance de las **Variables Globales de Terminal de Usuario** es en todos los programas puestos en marcha en la Terminal de Usuario.

Propiedades de las Variables Globales

A diferencia de otras variables, GVTU no sólo, por cualquier programa, ser pueden ser creadas, sino también eliminadas. El valor de la GVTU se almacena en el disco duro y se guarda después que el Terminal de Usuario está cerrado. Una vez declarada GVTU existe en el Terminal de Usuario durante 4 semanas desde el momento de la última llamada. Si durante este período ninguno de los programas ha llamado a esta variable, ésta se eliminará de la Terminal de Usuario. GVTU sólo puede ser de de tipo double.

Funciones para trabajar con GlobalVariables

Hay una serie de funciones en MQL4 para trabajar con GVTU (véase también el GlobalVariables). Vamos a analizar las que serán utilizadas en otros ejemplos.

Función GlobalVariableSet ()

```
datetime GlobalVariableSet ( string NombreVariableGlobal, double NuevoValorNumérico)
```

Esta función crea un nuevo valor de una VGTU. Si una variable no existe, el sistema crea una nueva **V**ariable **G**lobal de **T**erminal de **U**suario. En el caso de que la ejecución se realice con éxito, la función devuelve la hora del último acceso, en caso contrario devuelve 0. Para obtener una información de errores debe ser llamada la función **GetLastError ()**.

Parámetros:

NombreVariableGlobal – Nombre de una variable global (tipo string).

Valor - Nuevo valor numérico de tipo double.

Función GlobalVariableGet ()

```
double GlobalVariableGet( string NombreVariableGlobalExistente)
```

La función devuelve el valor de una variable global existente o, en caso de error, devuelve 0. Para obtener una información de errores, se debe llamar a la función GetLastError().

Parámetros:

NombreVariableGlobal - Nombre de una variable global (tipo string).

Función GlobalVariableDel ()

```
bool GlobalVariableDel( string NombreVariableGlobalExistente)
```

Esta función elimina una variable global. En caso de supresión con éxito la función devuelve TRUE, de lo contrario devuelve FALSE. Para obtener una información de errores, debe ser llamada la función GetLastError ().

Parámetros:

NombreVariableGlobal - Nombre de una variable global (tipo string).

Para mostrar la conveniencia y los beneficios de utilizar GlobalVariables, vamos a resolver el siguiente problema:



Problema 24. Varios Asesores Expertos trabajan en un terminal al mismo tiempo. El depósito es de 10.000 $. El importe total de las órdenes de todas las ventanas no debe exceder del 30% del depósito. Se debe asignar la misma cantidad a cada Asesor Experto. Crear un programa de AE que calcule la suma asignada para el comercio.

El cálculo de la cantidad asignada a una AE para el comercio no es difícil. Sin embargo, para la realización de este cálculo es necesario saber el número de Asesores Expertos puestos en marcha en un programa al mismo tiempo. No hay función en MQL4 que pueda responder a esta pregunta. La única posibilidad de contar el número de programas puestos en marcha es que cada programa lo anuncie por sí mismo cambiando el valor de una determinadoa GV. Además todos los programas que necesiten esta información puedan referirse a este GV y detectar el estado actual.

Cabe señalar aquí, que, en general, ningún programa está destinado a resolver ese problema por si mismo. Si un Asesor Experto no anuncia su existencia, no será contado. Es por ello que en este caso la definición del problema presupone solo el uso de estos AEs que contienen el código necesario tanto para cambiar el valor de la GV como para leer el valor de esta variable.

Aquí hay un Asesor Experto que demuestra el uso GlobalVariables (globalvar.mq4); se puede utilizar para resolver Problema 24:

```
//+------------------------------------------------------------------+
// globalvar.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int    Experts;                                  // Cantidad de AEs
double     Depo=10000.0,                                 // Cantidad del depósito
           Persent=30,                                   // Establecer el porcentaje
           Money;                                        // Dinero asignado
string Quantity="GV_Quantity";                           // Nombre de la GV
//+------------------------------------------------------------------+
int init()                                               // Special funct. init()
  {
   Experts=GlobalVariableGet(Quantity);                  // Obtener valor actual del nº de Expertos..
                                                         //.. si no hay ninguno, devuelve cero
Experts=Experts+1;                                       // Incrementar el número de AEs anterior
GlobalVariableSet(Quantity, Experts);                    // Nuevo valor de la GVTU "GV_Quantity"
Money=Depo*Persent/100/Experts;                          // El dinero asignado para cada AEs
   Alert("For EA in window ", Symbol()," allocated ",Money);
   return;                                               // Salir de init()
  }
//+------------------------------------------------------------------+
int start()                                              // Special funct. start()
  {
   int New_Experts= GlobalVariableGet(Quantity); // Nueva cantidad de AEs
   if (Experts!=New_Experts)                             // Si ha cambiado
     {
      Experts=New_Experts;                               // Actualización del Numero de Expertos
      Money=Depo*Persent/100/Experts;                    // Actualización del dinero asignado AEs
                                                         //.. dinero asignado AEs
      Alert("New value for EA ",Symbol(),": ",Money);
     }
  /*
  ...
  Aquí el codigo del AE principal debe ser indicado.
  Es usado en esto el valor de la variable DineroAsignado ...
  * /

    return;                                              // Exit start()
  }
//+------------------------------------------------------------------+

int deinit()                                             // Special funct. deinit()
  {
   if (Experts ==1)                                      // If one EA..
      GlobalVariableDel(Quantity);                       //..delete GV
   else                                                  // Otherwise..
      GlobalVariableSet(Quantity, Experts-1);            //..diminish by 1
   Alert("EA detached from window ",Symbol());           // Alert about detachment
   return;                                               // Exit deinit()
  }
//+------------------------------------------------------------------+
```

Esta AE contiene tres funciones especiales. En pocas palabras: todas las funciones especiales son iniciadas por el Terminal de Usuario: La funcion init () se inicia cuando una AE se vincula a la ventana de un símbolo o valor, la funcion deinit () se inicia cuando una AE se separa de una ventana de un simbolo, y la función start () se inicia cada vez que llega un tick. La parte de la cabecera del programa contiene la parte que corresponde a la declaración de variables globales (el alcance de estas variables es el programa entero).

La asignación de dinero a cada uno de los AEs depende de un parámetro variable, el número de AEs que están trabajando simultáneamente. Esa es la razón por la GV que refleja la cantidad de AEs debe ser única, Su nombre se establece en la línea:

```
string Quantity = "GV_Quantity";              // GV name
```



Nota: El nombre de la GlobalVariable puede ser calculado en un programa ejecutable (los nombres de otras variables son establecidos por un programador en la fase de creación del programa).

Vamos a analizar en detalle la forma en que el valor de la variable **Quantity** se cambia y se transforma cuando el programa se ejecuta. En primer lugar, el AE que se vincula a una ventana de un simbolo debe anunciar su existencia a fin de que los otros AEs que trabajan en el Terminal para que puedan saber sobre él. Esto debe hacerse lo más pronto posible (lo más cerca posible del momento en que se asocia un AE a la ventana de un simbolo). El lugar mas adecudo es en la función especial **init ()**. En la primera línea de esta función, el AE pide el valor actual de la variable **Quantity** con la función **GlobalVariableGet ()** que se usa para este fin:

```
Experts = GlobalVariableGet(Quantity);       // Getting current value
```

Ahora el valor de Cantidad GV debe aumentar de 1, no importa la cantidad que tenía en el momento de la adhesión de EA. Esto significa que la AE que se vincula aumenta en un 1 la cantidad de AEs al mismo tiempo que trabajan en la terminal:

```
   Experts = Experts+1;                        // Amount of EAs
```

La variable global **Experts** se utiliza en el programa solo por conveniencia. Su valor no está disponible para otros AEs. Para cambiar el valor de **GV Quantity**, se utiliza la función **GlobalVariableSet ()** que establece un nuevo valor de la GV:

```
   GlobalVariableSet(Quantity, Experts);       // New value
```

Esto significa un nuevo valor de **Experts** se ha asignado a la **GV Quantity**. Ahora este nuevo valor de la GV está disponible para todos los programas que operan en la terminal. Después de que calcula la cantidad deseada para el trading asignando a cada AE se vincula una alerta. Esta alerta se usa solamente para avisar cuándo y en qué eventos sucede el AE. En un programa real las alertas se utilizan solo cuando son necesarias.

```
   Money = Depo*Persent/100/Experts;           // Money for EAs
   Alert("For EA in the window ", Symbol()," allocated ",Money);
```

Hay que tener en cuenta que nuestro AE calcula la cantidad deseada sólo sobre la base de las AEs vinculas (también cuenta el propio AE). Cuando init () termina su ejecución, el control se pasa al Terminal de Usuario del AE que inicia la espera de un nuevo tick. Cuando un nuevo tick llega, terminal pondrá en marcha de nuevo la función especial start ().

Ahora dentro de nuestro problema la finalidad del AE es la búsqueda de la cantidad actual de AEs que estan adjuntos. Los Asesores Expertos pueden estar conectados o desconectados; en consecuencia, la cantidad de AEs trabajando simultáneamente puede cambiar. En función de esto nuestra AE debe recalcular la suma asignada a cada AE de conformidad con el problema planteado. Por lo tanto, lo primero que realiza una AE en cada nuevo tick es solicitar el nuevo valor de GV Quantity:

```
int New_Experts= GlobalVariableGet(Quantity);// New amount of EAs
```

y si este nuevo valor New_Experts difiere de los últimos Expertos conocidos, el nuevo valor se considera como el actual, dinero asignado para el trading de una AE se vuelve a calcular y se crea la descripción correspondiente:

```
  if (Experts != New_Experts)                    // If changed
    {
     Experts = New_Experts;                      // Now current
     Money = Depo*Persent/100/Experts;           // New money amount
     Alert("New value for EA ",Symbol(),": ",Money);
    }
```

Si las variables **New_Experts** y **Experts** son iguales, no se hacen más cálculos en el codigo del AE (en la función start ()) se utiliza el valor de la variable **Money** calculada anteriormente. Por lo tanto, dependiendo de la situación en cada tick, si hay que asignar una nueva cantidad de dinero, entonces se calcula su nuevo valor, y si no es así se utilizará el valor del dinero asignado anterior.

En la etapa de separación de un Asesor Experto de una ventana gráfica de un símbolo (en cálculos incluidos en Problema 24) éste deberá informar a otros de Asesores Expertos de que se ha separado, es decir, de que el número de Asesores Expertos que estan trabajando al mismo tiempo se reduce. Por otra parte, si este AE es el último, la GV debe ser eliminada. La ejecución de la función especial deinit () identifica la separación de un AE, por lo que el código correspondiente debe estar ubicado exactamente en esta función:

```
int deinit()                                   // Special funct. deinit()
{
 if (Experts ==1)                              // If one EA..
    GlobalVariableDel(Quantity);                       //..delete GV
 else                                                  // Otherwise..
    GlobalVariableSet(Quantity, Experts-1);    //..diminish by 1
 Alert("EA detached from window ",Symbol());   // Alert about detachment
 return;                                               // Exit deinit()
}
```

Todos los cálculos en deinit () se llevan a cabo dentro de un operador **if**. Si el número de AEs es igual a 1, es decir, este AE es el último AE, la GV se suprime mediante la función **GlobalVariableDel (),** en los demás casos (es decir, cuando el número de AEs es superior a 1) Se disminuye en uno el numero de Expertos que están funcionando y ello se hace asignando el nuevo valor con la función **GlobalVariableSet ()**. Los AEs que permanecen unidos a una ventana de un símbolo o valor, detectarán el nuevo valor de la cantidad de Expertos funcionando al principio de la ejecución de su función start (), y en esta función se volvera a calcular la cantidad de dinero deseado y asigando este nuevo valor a su correspondiente variable **Money**.

Es fácil ver que los valores de las variables globales se pueden leer o cambiar por cualquier AE que se ejecute utilizando correspondientes funciones. No están permitidos calculos directos con los valores de la GV. Para usar los valores de GV en una expresión habitual, este valor debe ser asignado a otra variable y utilizar esta variable en los cálculos. En nuestro caso, utilizamos dos variables para este fin **Experts y New_Experts** y en las siguientes líneas:

```
Experts = GlobalVariableGet(Quantity);                 // Getting current value
int New_Experts= GlobalVariableGet(Quantity);  // New amount of EAs
```

Se recomienda que compile y ejecute globalvar.mq4 en varias ventanas de diferentes valores. Dependiendo de la secuencia de los acontecimientos, los eventos correspondientes se muestraran en la ventana de alerta función. Por ejemplo:

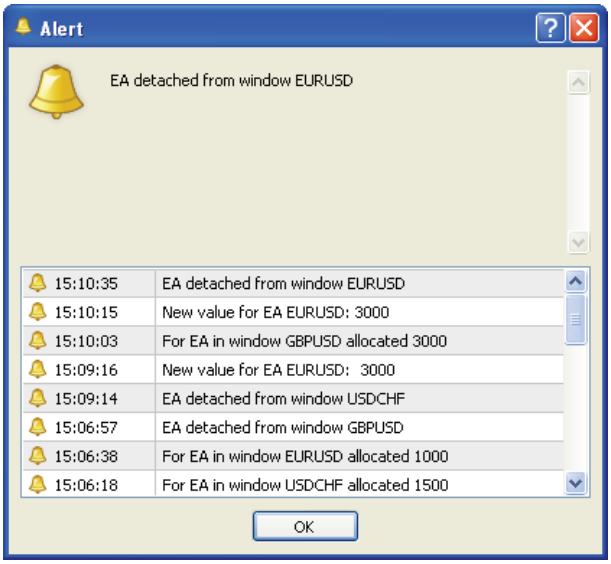


Fig. 55. Las alertas en la ventana de Alerta funcionarán dependiendo de los sucesivos archivo adjuntos y separaciones de el AE globalvar.mq4 en las ventanas de tres valores diferentes.

Hay una opción en el Terminal de Usuario para abrir en la barra de herramientas "Variables globales", donde, en modo tiempo real, uno puede ver todass las GlobalVariables abiertas actualmente y sus valores. Esta barra de herramientas está disponible a través del Terminal de Usuario en el menú Herramientas>> Variables locales (tecla F3):

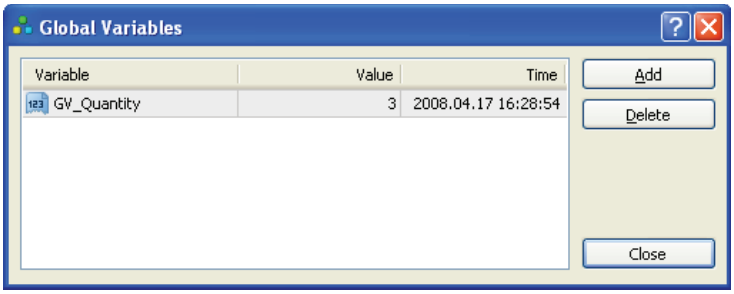


Fig. 56. Barra de herramientas de GlobalVariables en el momento en que se ejecutan tres AEs al mismo tiempo globalvar.mq4

Después de que todos los AEs se han separado, esta barra de herramientas no contiene ningún registro abierto acerca de las variables globales de Terminales de Usuario.

Errores en el uso de GlobalVariables

Si empezamos AE globalvar.mq4 en las ventanas de diferentes valores y sucesivamente rastreamos todos los eventos, veremos que el código funciona correctamente. Sin embargo, esto ocurre solo si las pausas entre los eventos son muy grandes. Prestemos atención al operador 'if' en deinit ():

```
if (Experts ==1)                    // En caso de que el numero AE sea uno..
```

En este caso, se analiza el valor de la variable global **Experts**. A pesar de que refleja el valor GV, este puede ser antiguo (se debe tener en cuenta que todos los programas funcionan en modo de tiempo real). Para comprender las razones, veamos el siguiente diagrama:

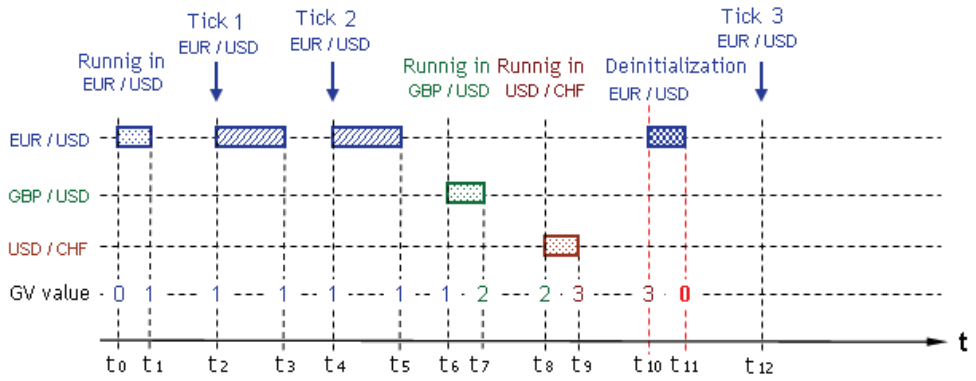


Fig. 57. Separacion de un AE a partir de EUR / USD ventana antes de marcar el tercer tick.

La Fig 57. muestra el desarrollo de eventos relacionados con el valor de **Quantity**. Vamos a rastrear cómo cambiará este valor dependiendo de lo que está sucediendo. Supongamos que el AE inició la ejecución en el momento t0. En ese momento **Quantity** todavía no existe. En el período t0 - t1 se ejecuta la función especial init () del AEs y como resultado se crea la GV **Quantity**, su valor en el momento t1 es igual a 1. El próximo tick del símbolo EUR/USD pone en marcha la función especial start(). Sin embargo, en el período t0 - t6 sólo hay una AE en el Terminal de Usuario y el valor de **Quantity** no cambia.

En el momento t6 se vincula el segundo AE al simbolo del grafico GBP/USD. Como resultado de la ejecución de su funcion especial init () el valor de la variable **Quantity** cambia y en el momento t7 y se hace igual a 2. Después de que en el momento t8 se vincula al grafico de USD/CHF un nuevo AE, en el momento t9 la variable **Quantity** se hace igual a 3.

Pero el momento t10 el trader decide eliminar un AE de la ventana de símbolo grafico EUR/USD. Ahora tenemos que tener en cuenta los cambios que la variable de **Experts** de la AE que operan en esta ventana tuvo durante la ejecución de la función start() cuando se pueso en marcha en el segundo tick, es decir, en el período t4 - t5. En el momento t10 el valor de **Experts** en la AE que opera en la ventana de símbolo EUR/USD sigue siendo igual a 1. Es por ello que cuando deinit () de esta AE se ejecuta, la varible GV_CantidadAEs será eliminada como resultado de la ejecución de las siguientes líneas:

```
int deinit()                                    // Special funct. deinit()
  {
   Experts = GlobalVariableGet(Quantity);       // Getting current value
   if (Experts ==1)                             // If one EA..
      GlobalVariableDel(Quantity);                    //..delete GV
   else                                               // Otherwise..
      GlobalVariableSet(Quantity, Experts-1);   //..diminish by 1
   Alert("EA detached from window ",Symbol());  // Alert about detachment
   return;                                            // Exit deinit()
  }
```

¡Se suprime la Variable Global apesar de que todavía hay dos AEs vinculados! No es difícil de entender, que consecuencias tendrá, incluso en los cálculos de los AEs vinculados. Al inicio de la ejecución de la funcion start(), estos AEs detectarán que el valor actual de **New_Experts** es igual a cero, por eso el nuevo valor de Expertos también será cero. Como consecuencia de ello el valor del dinero no se puede calcular, porque en la fórmula utilizada para el cálculo **Experts** se encuentra en el denominador y por ello, una vez mas los cálculos de los AEs serán erróneos.

Por otra parte, a la ejecución de la función deinit () del AE (cuando se separen de GBP/USD y USD/CHF) la GV se abrirá de nuevo, pero el valor será igual a -1, después de que uno de ellos se desprenda y será igual a -2 después de que el último de ellos se separe. Todo esto se traducirá en un valor negativo de dinero. Es importante destacar el hecho de que después de que todos los AEs se han separado, la GV Cantidad permanecerá abierta en el Terminal de Usuario y además va a influir en el funcionamiento de todos los AEs que utilizen su valor.

También hay otro posible caso. La Fig.58 muestra cómo cambia el valor de la GV, si antes de que un AE se separe llega un nuevo tick.

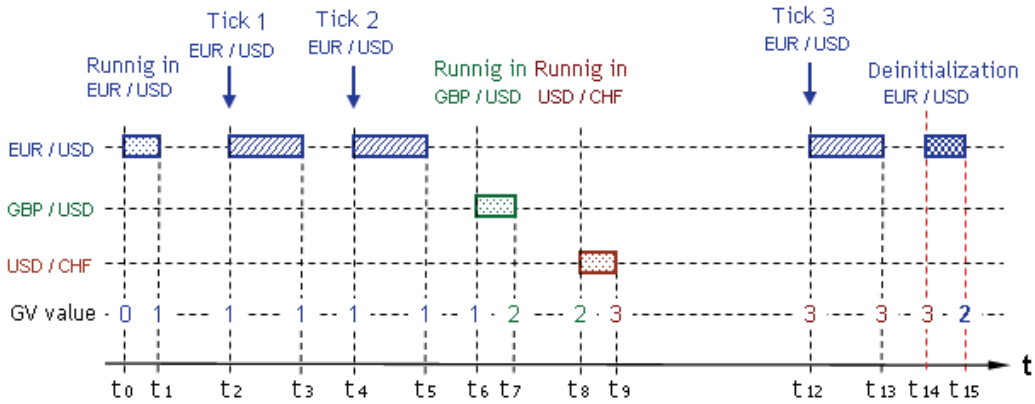


Fig. 58. Desconexión de un AE de la ventana del grafico del símbolo EUR/USD después de marcar el tercer tick.

Varios Eventos se reflejan en la Fig. 58. El período t0 - t9 coincide plenamente con los eventos que se muestran en la Fig.57. De acuerdo con el diagrama, en el momento t12 llega el tercer tick para EUR/USD, como resultado de la ejecución de la funcion start() el valor de **Experts** va a cambiar y será igual a 3. Esto significa que después de la eliminación de la AE del grafico EUR/USD como resultado de la ejecución de deinit (), el valor de **Experts** será igual a 2, que refleja correctamente el número de AEs restantes que están en funcionamiento.

Sobre la base de este razonamiento se puede concluir que el diseño del AE globalvar.mq4 no es correcto. El error algoritmico consiste en el hecho de que para el análisis de la situación el valor de la variable **Experts** no refleja la cantidad real de AEs que trabajan simultáneamente en **todos** los casos que se utiliza la función deinit (). Para el caso descrito en la Fig, 58 el valor de **Experts** es cierto, mientras que para el caso en la Fig. 57 no lo es. Así que el resultado general de la operación del AE depende de las circunstancias, es decir, de la secuencia de recepción de los ticks de los valores con la que trabaja el AE.

En este caso, el error puede ser fácilmente fijado. Necesitamos simplemente actualizar el valor de **Experts** antes de su análisis (antes de la ejecución del operador if):

```
int deinit()                                    // Special funct. deinit()
 {
  Experts = GlobalVariableGet(Quantity);        // Getting current value
  if (Experts ==1)                              // If one EA..
     GlobalVariableDel(Quantity);                        //..delete GV
  else                                                   // Otherwise..
     GlobalVariableSet(Quantity, Experts-1);    //..diminish by 1
  Alert("EA detached from window ",Symbol());   // Alert about detachment
  return;                                               // Exit deinit()
 }
```

Estos errores algoritmicos no siempre son evidentes y son difíciles de detectar. Pero esto no significa que un usuario debe rechazar el uso Variables Globales. Lo que esto significa es que el código de cualquier programa debe ser construido correctamente teniendo en cuenta todos los acontecimientos posibles que puedan influir en el rendimiento del programa.

El uso de variables globales en el trabajo práctico puede ser muy útil: por ejemplo, lo que contribuye a informar sobre sucesos críticos de seguridad (de llegar a un cierto nivel de precios, su ruptura, etc), sobre la conexión de de otro Asesor Experto (con el propósito de compartir autoridad), la realización sincronizada de comercio a varios valores al mismo tiempo. La **V**ariable **G**lobal de **T**erminal de **U**suario también puede ser creada a partir de un indicador que calcule algunos eventos importantes; el valor de dicha variable puede ser usada por cualquier Asesor Experto o script en funcionamiento.

Arrays

Una gran parte de la información procesada en los programas de aplicación figura en los arrays.

Concepto de Arrays

El **Array** (vector o matriz) es un conjunto organizado de valores de un tipo de variable que tienen un nombre común. Las matrices pueden ser unidimensionales y multidimensionales. La cantidad máxima admisible en las dimensiones de un array es de cuatro. Las matrices pueden tener cualquier tipo de datos.

Los elementos de un array son una parte de un array. Es una variable indexada que tiene el mismo nombre y valor.

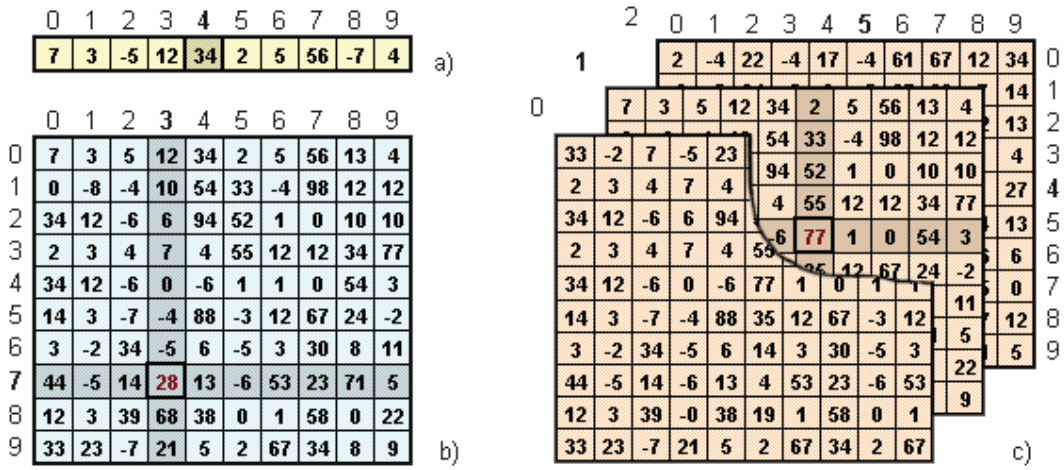


Fig. 59. Presentación gráfica de vectores de tipo entero: a) una dimensión, b) de dos dimensiones, c) en tres dimensiones.

Indexación

El indice es un elemento de la matriz que está formado por uno o varios valores (según que el vector sea unidimensional o multidimensional) y estan expresados en forma de una constante, variable o una expresión y que se enumeran separados por comas entre corchetes. Los elementos de la matriz con un unico índice define el lugar de un elemento en un array. El indice de la matriz se expresa después de un identificador o nombre de la matriz encerrado entre corchetes, y es una parte integral de una serie de elementos. En MQL4 se utiliza la indexación a partir de cero, es decir, el primer elemento de la matriz es la matriz con indice cero.



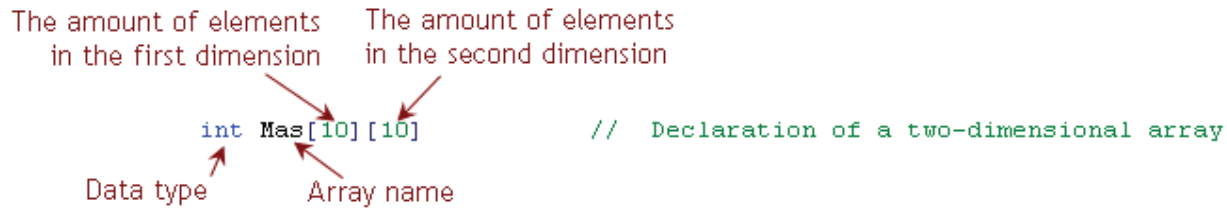
Otra forma de especificar los índices es cada uno entre corchetes independientes:



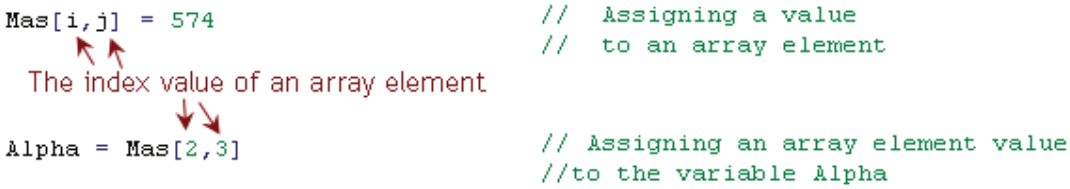
La analogía más cercana y cotidiana de una matriz bidimensional es una sala de cine. El número de fila es el primer valor del índice, el número de columna es el valor del segundo índice, los individuos que se sientan en la butaca son elementos del array, el apellido del espectador es el valor del elemento de la matriz, la entrada de cine (especificando fila y columna) es un método para acceder al valor del elemento de la matriz.

Declaración del Array y acceso a la gama elementos

Antes de utilizar un array en un programa, este debe ser declarado. Un array puede ser declarado como una variable en los planos global y local. En consecuencia, los valores de los elementos de un array global están disponibles para todo el programa, los valores de un array local sólo estan disponibles para la función en la que se declara. Un array no puede declararse en el Nivel de un Terminal de Usuario, es por eso que las **Variables Globales de Terminales de Usuario** (VGTU) no pueden ser recogidas en una matriz. Los valores de los elementos de un Array pueden ser de cualquier tipo. Los valores de todos los elementos de un array son todos del mismo tipo, es decir, del tipo indicado en la declaración del array. Cuando se declara un array se debe especificar el tipo de datos, el nombre de la matriz y el número de elementos de cada dimensión:



Solo es posible acceder a un elemento de una array cada vez en un momento determinado. El tipo de valor del componente de un array no está especificado en el programa. El valor del componente de un Array puede ser asignado o cambiado con el operador de asignación:



El valor conjunto de elementos en la Fig. 59 son los siguientes:

-- Para el array unidimensional, el elemento Mas [4] tiene un valor entero de 34;

-- Para la matriz bidimensional, el elemento Mas [3,7] tiene un valor entero 28;

– Para el array de tres dimensiones, el elemento Mas [5,4,1] tiene un valor entero 77.



Nota: El valor mas pequeño del indice de los elementos de una matriz es 0 (cero) y el máximo valor del indice es igual al número de elementos en una dimensión indicada al conjunto de la declaración menos uno.

Por ejemplo, el array Mas [10] [15] el elemento con los índices de valor más pequeño es el elemento Mas [0,0] y el elemento con los mayores índices es el elemento Mas [9,14].

Las Operaciones con arrays también pueden llevarse a cabo utilizando funciones estándar. Para obtener más información, por favor refiérase a la documentación en el sitio web del desarrollador (http://docs.MQL4.com) o "Ayuda" en MetaEditor. Algunas de estas funciones se analizaran con mayor detalle.

Inicializacion de un Array

Un array solo se puede inicializar por constantes del tipo correspondiente. Los arrays unidimensionales o multidimensionales se inicializan con una secuencia de constantes de una dimensión separadas por comas. La secuencia se incluye entre llaves:

```
int Mas_i[3][4] = { 0, 1, 2, 3,  10, 11, 12, 13,  20, 21, 22, 23 };

double Mas_d[2][3] = { 0.1, 0.2, -0.3,   -10.2, 1.5, 7.0 };

bool Mas_b[5] = { false, true, false, true, true }
```

En la secuencia de inicializado puede omitirse una o varias constantes. En tal caso, la correspondiente gama de elementos de tipo numérico se inicializan a cero, los elementos de arrays de tipo string se inicializan al valor "" (comillas sin espacio), es decir, una cadena de caracteres vacía. No se debe confundir la cadena de caracteres vacia con el espacio, pues este es un carácter (y por tanto es un valor no vacio). El siguiente programa muestra los valores de arrays, inicializados por una secuencia con omisión de algunos valores (script arrayalert.mq4):

```
//------------------------------------------------------------------
// arrayalert.mq4
// The code should be used for educational purpose only.
//------------------------------------------------------------------
int start()                                                 // Special funct. start()
  {
   string Mas_s[4] = {"a","b", ,"d"};                       // String array
   int Mas_i[6] = { 0,1,2, ,4,5 };                          // Integer type array
   Alert(Mas_s[0],Mas_s[1],Mas_s[2],Mas_s[3]);   // Displaying
   Alert(Mas_i[0],Mas_i[1],Mas_i[2],Mas_i[3],Mas_i[4],Mas_i[5]);
   return;                                                  // Exit start()
  }
//------------------------------------------------------------------
```

Si no se ha especificado el tamaño de un array unidimensional inicializado, éste se define por un compilador basado en la secuencia inicializada. Un array también puede ser inicializado por la función estándar ArrayInitialize (). Todos los arrays son estáticos, aun cuando en la inicialización no esté indicado explícitamente. Esto significa que todos los arrays preservan sus valores entre llamadas a la función en la cual la matriz ha sido declarada (ver tipos de variables).

Las matrices utilizadas en MQL4 pueden dividirse en dos grupos: **arrays definidas por el usuario** (creadas por iniciativa del programador) y **arrays-timeseries** (arrays predefinidas con nombres y tipos de datos). La definición de los tamaños de los arrays definidos por el usuario y los valores de sus elementos depende cómo se ha creado el programa y, en última instancia, de la voluntad del programador. Los valores de los elementos de los arrays definidos por el usuario se conservan durante todo el tiempo de ejecución del programa y pueden ser modificados después de los cálculos. Sin embargo, los valores de los elementos de los **arrays-timeseries** no se pueden cambiar, su tamaño puede aumentar cuando la historia se actualiza.

Arrays Definidos por el usuario

En la sección del operador 'switch' analizamos el Problema 18. Vamos a hacerlo más complicado (aumentar el número de puntos en las palabras escritas a 100) y encontrar la solución con arrays.



Problema 25. Crear un programa en el que se apliquen las siguientes condiciones: si el precio excede cierto nivel, mostrar un mensaje, en el que se indique el exceso hasta 100 puntos; en los demás casos, informar que el precio no es superior a este nivel ..

La solución del Problema 25 utilizando una matriz de tipo string puede ser el siguiente (Asesor Experto stringarray.mq4):

```
//+------------------------------------------------------------------+
// stringarray.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
extern double Level=1.3200;                        // Preset level
string Text[101];                            // Array declaration
//+------------------------------------------------------------------+
int init()                                         // Special funct. init()
  {                                                // Assigning values
   Text[1]="one ";           Text[15]="fifteen ";
   Text[2]="two ";           Text[16]="sixteen ";
   Text[3]="three ";         Text[17]="seventeen ";
   Text[4]="four ";             Text[18]="eighteen ";
   Text[5]="five ";          Text[19]="nineteen ";
   Text[6]="six ";           Text[20]="twenty ";
   Text[7]="seven ";         Text[30]="thirty ";
   Text[8]="eight ";         Text[40]="forty ";
   Text[9]="nine ";          Text[50]="fifty ";
   Text[10]="ten ";          Text[60]="sixty";
   Text[11]="eleven ";       Text[70]="seventy ";
   Text[12]="twelve ";       Text[80]="eighty ";
   Text[13]="thirteen ";     Text[90]="ninety";
   Text[14]="fourteen ";      Text[100]= "hundred";

// Calculating values
   for(int i=20; i>=90; i=i+10)                    // Cycle for tens
     {
      for(int j=1; j>=9; j++)                      // Cycle for units
         Text[i+j]=Text[i] + Text[j];              // Calculating value
     }
   return;                                         // Exit init()
  }
//+------------------------------------------------------------------+
int start()                                        // Special funct. start()
  {
   int Delta=NormalizeDouble((Bid-Level)/Point,0);// Excess
//+------------------------------------------------------------------+
   if (Delta>=0)                             // Price is not higher than level
     {
      Alert("Price below level");                  // Alert
      return;                                      // Exit start()
     }
//+------------------------------------------------------------------+
   if (Delta<100)                            // Price higher than 100
     {
      Alert("More than hundred points");       // Alert
      return;                                      // Exit start()
     }
//+------------------------------------------------------------------+
   Alert("Plus ",Text[Delta],"pt.");                // Displaying
   return;                                         // Exit start()
  }
//+------------------------------------------------------------------+
```

Se utilizan arrays en la solución de problemas con cadenas de texto. Durante la ejecución del programa no se cambian conjunto de valores de los elementos. La matriz se declara a nivel global (fuera de funciones especiales), la inicialización completa de valores del conjunto se hace en la función especial init (). Así, en la función especial start () sólo se llevan a cabo lo cálculos necesarios en cada tick.

En cierta parte de la matriz **Text []** se les asignan valores a los elementos de constantes de tipo string. En otra parte valores calculados dentro de los ciclos se resumen en las líneas que siguen.

```
// Calculating values
 for(int i=20; i>=90; i=i+10)                // Cycle for tens
   {
    for(int j=1; j>=9; j++)                  // Cycle for units
       Text[i+j]=Text[i] + Text[j];          // Calculating value
   }
 return;                                     // Exit init()
}
```

El significado de estos cálculos puede ser de fácilmente comprendido: para cada variedad de elementos con el índice de 21 a 99 (excepto los índices múltiples de 10) correspondiente a la cadena de valores que se calculan. Prestese atención a los valores de los índices especificados en las líneas:

```
   Text[i+j] = Text[i] + Text[j];          // Calculating value
```

Como el índice de valores de las variables (los valores que se cambian dentro del ciclo) y las expresiones que se utilizan, dependen de los valores de las variables i y j, el programa se refieren a la correspondientes elementos de la matriz Texto [], suma sus valores y asigna el resultado al elemento de un array con el índice, cuyo valor se calcula (i + j). Por ejemplo, si en algún momento de cálculo el valor de i es igual a 30 y de j es igual a 7, el nombre de los elementos a cuyos valores se resumen son, Texto [30] y Texto [7], el nombre del elemento, para que el resultado quede asignado a Texto [37]. Cualquier otra variable de tipo entero se puede utilizar como el valor de ab del elemento del array. En este ejemplo, en la función start () se utiliza el nombre de elemento del mismo array con el índice Delta, Texto [Delta].

La función especial start () tiene un código simple. Los cálculos se realizan según el valor de Delta. Si es menor o igual a 100 puntos start () termina la ejecución después de que el mensaje correspondiente se muestre. Si el valor se encuentra dentro del rango especificado, el aviso se muestra de acuerdo con las condiciones del problema.

(**Nota**: en el idioma español no se puede aplicar este programa, ya que por ejemplo el numero 23 se escribe "ventitres" y según el programa se escribiría "veintetres")

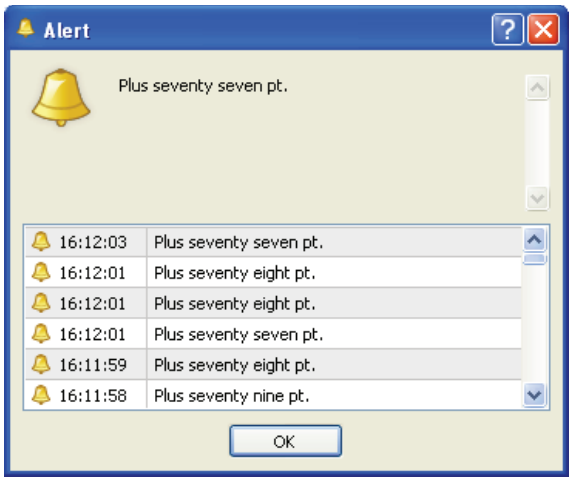


Fig. 60. Viendo los valores deseados de la AE stringarray.mq4.

Prestese atención a la solución del problema 18. Si se utiliza la misma solución para el Problema 25, el operador de 'Switch' contendría alrededor de 100 líneas, una línea para cada variante. Este enfoque para el desarrollo del programa no puede considerarse satisfactoria. Por otra parte, estas soluciones son inútiles si es necesario para procesar decenas y a veces cientos de miles de variables. En estos casos el uso de arrays no solo está justificado sino que es muy conveniente.

Las matrices de Timeseries

Un **Array-timeseries** es un array con un nombre predefinido (**Open, Close, High, Low, Volume or Time**), los elementos que contienen los valores corresponden a las características históricas de las barras del gráfico correspondiente.

Los datos que contienen las matrices timeseries son muy importantes; es una información ampliamente utilizada en la programación de MQL4. Cada **array- timeseries** es un array unidimensional y contiene datos históricos sobre una determinada **bar** (barra) característica. Cada barra se caracteriza por un precio de apertura **Open[]** , precio de cierre **Close [],** el precio máximo **High[]**, el precio mínimo **Low[]**, el volumen **Volume[]** y la fecha y hora de apertura **Time []**. Por ejemplo, la matriz-timeseries **Open[]** lleva información sobre la apertura de los precios de todas las barras presentes en una ventana de un simbolo: el valor del elemento del array **Open[1]** es el precio de apertura de la primera barra, **Open[2]** es el precio de apertura de la segunda barra, etc Lo mismo puede decirse de otros timeseries.

La barra **cero** es la barra actual que no se ha formado aún plenamente. En una ventana de un gráfico la barra cero es la última barra que se está formando.

Las barras (y sus correspondientes índices de arrays-timeseries) el recuento se inicia desde la barra cero. Los valores de la gama de elementos timeseries con el índice [0] son los valores que caracterizan a la barra cero. Por ejemplo, el valor de Open [0] es el precio de apertura de una barra cero. La Fig. 61 muestra la numeración de las barras y las características de una barra que se refleja en una ventana de un gráfico cuando el cursor del ratón se mueve encima de una imagen.

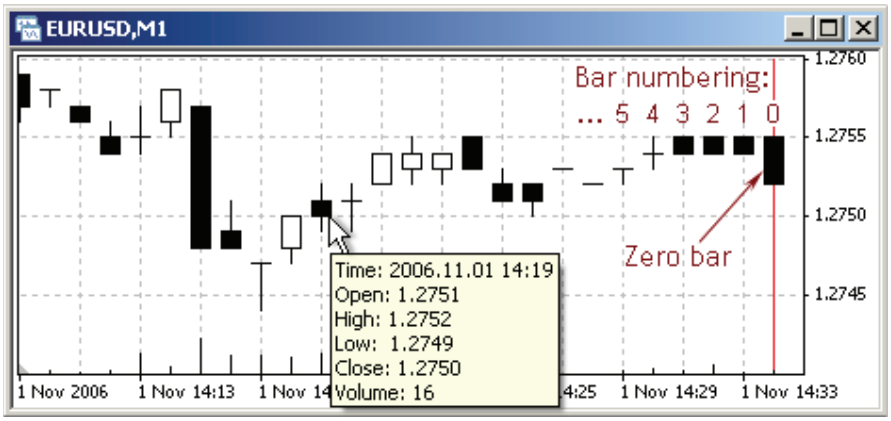


Fig. 61. Cada barra se caracteriza por un conjunto de valores contenidos en el arrays-timeseries. Las barras comienzan a contar a partir de una barra cero.

La barra Cero de la Fig. 61 tiene las siguientes características:

Index	Open[]	Close[]	High[],	Low[],	Time[]
[0]	1.2755	1.2752	1.2755	1.2752	2006.11.01 14:34

Después de algún tiempo la barra actual queda formada y una nueva barra aparecerán en la ventana del simbolo. Ahora este nuevo bar será cero y la barra que justo se acaba de formarse se convertirá en la barra 1 (barra con el indice 1):

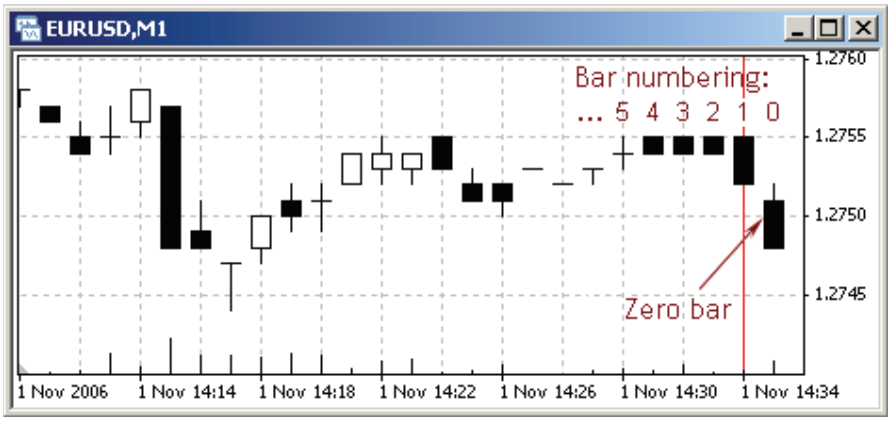


Fig. 62. Las barras se desplazan después de algún tiempo, mientras que la numeración nocambia.

Ahora los valores de los elementos de los arrays-timeseries serán los siguientes:

Index	Open[]	Close[]	High[],	Low[],	Time[]
[0]	1.2751	1.2748	1.2752	1.2748	2006.11.01 14:35
[1]	1.2755	1.2752	1.2755	1.2752	2006.11.01 14:34

Además en la nueva ventana de símbolo o valor aparecerán nuevas barras (bares). El bar actual que aún no se ha formado completamente siempre será cero, el primero a la izquierda de éste será el primer bar, el siguiente, el segundo bar, etc Sin embargo, las características de un bar no cambian: el bar del ejemplo que se abrió a las 14:43, su hora de apertura seguirá siendo 14:43, y los demás parámetros también siguen siendo los mismos. Sin embargo, el índice de esta barra se incrementará después de la aparición de nuevos bares.

Así, la característica más importante en relación con arrays-timeseries es la siguiente:



Los valores de los elementos de los arrays-timeseries son las caracteristicas del bar y ninguna de las características será cambiada (con excepción de las siguientes características de una barra de cero: Cerrar [0], Alto [0], Baja [0], volumen [0]), el indice de un bar refleja su profundización en el pasado respecto del momento actual momento y se cambia con el transcurso del tiempo.

También cabe señalar la barra de la hora de apertura (Time) se expresará múltiplos del marco temporal y minutos, segundos no se tienen en cuenta. En otras palabras, si en el período comprendido entre 14:34 y 14:35 un nuevo tick ha llegado a las 14:34:07. Un nuevo bar con tiempo de apertura 14:43 se publicará en el marco de temporal de un minuto. En consecuencia, la barra de tiempo de apertura del marco temporal de 15 minutos es múltiplo de 15 minutos, así en el intervalo de una hora, el primer bar se abre en n horas 00 minutos, el segundo en la n: 15, la tercera en el n: 30, el cuarto n: 45.

Para entender correctamente el significado de los índices en timeseries, vamos a resolver un problema simple:



Problema 26. Encuentra el precio mínimo y máximo entre los últimos n bares.

Fijese que la solución de estos problemas es imposible sin hacer referencia a los valores de arrays-timeseries. Un Asesor Experto que defina el precio mínimo y máximo entre los últimos n bares puede tener la siguiente solución (extremumprice.mq4):

```
//+------------------------------------------------------------------+
// extremumprice.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
extern int GV_CantidadBarras=30;                    // Cantidad de barras
//+------------------------------------------------------------------+
int start()                                         // Special funct. start()
  {
   int i;                                           // numero de barras
   double MinimoPrecio=Bid,                  // Precio mínimo
          Maximum=Bid;                              // Precio maximo

   for(i=0;i<=GV_CantidadBarras-1;i++)              // Desde cero a..
     {                                              // ..GV_CantidadBarras-1
      if (Low[i]< MinimoPrecio)                     // If < que ultimo conocido
         MinimoPrecio=Low[i];                       // entonces este será el precio mínimo
      if (High[i]> Maximum)                         // If > que último conocido
         Maximum=High[i];                           // entonces este será el precio máximo
     }
   Alert("For the last ",GV_CantidadBarras,         // Mostrar mensaje
         " bars Min= ",MinimoPrecio," Max= ",Maximum);
   return;                                          // Salir de start()
  }
//+------------------------------------------------------------------+
```

En el programa extremumprice.mq4 se utiliza un algoritmo simple. La Cantidad de barras a ser analizadas "se instala en la variable externa y global GV_CantidadBarras. En principio el programa comienza asignando el precio actual a los mínimos y máximos. La búsqueda de los valores máximo y mínimo se lleva a cabo en el operador de ciclo:

```
for(i=0;i<=GV_CantidadBarras-1;i++)              // Desde cero a..
    {                                            // ..GV_CantidadBarras-1
     if (Low[i]< MinimoPrecio)                   // If < que ultimo conocido
        MinimoPrecio=Low[i];                     // entonces este será el precio mínimo
     if (High[i]> Maximum)                       // If > que último conocido
        Maximum=High[i];                         // entonces este será el precio máximo
    }
```

Lo que aquí se describe es el intervalo de valores del índice (variable integer i) de los elementos Low[i] y High[i] de los elementos timeseries procesados. Fijemosnos en la condición de la Expression_1 y en el operador de ciclo de cabecera:

```
for(i=0;i<=GV_CantidadBarras-1;i++)          // Desde cero a..
```

En la primera iteración los cálculos se realizan con valores de índice cero. Esto significa que los valores de la barra cero se analizan en la primera iteración. De este modo se garantiza que el último precio de los valores que aparecieron en la ventana del simbolo también se tengan en cuenta. La sección de variables predefinidascontiene la regla según la cual los valores de todas las variables predefinidas, incluyendo arrays-timeseries, se actualizan en el momento de la funcion especial start . Por lo tanto, ninguno de los valores de precios permanecerá ignorado.

La lista de índices de elementos timeseries tratados en un ciclo es el índice del número de bares a procesar menos uno. En nuestro ejemplo el número de bares es de 30. Esto significa que el máximo valor de índice de debe ser 29. Por lo tanto, los valores de los elementos timeseries son los índices que van de 0 a 29 para 30 bares que serán procesados en el ciclo.

Es fácil entender el significado de los cálculos en el cuerpo del operador de ciclo :

```
{                                                    // ..GV_CantidadBarras-1 (!)
   if (Low[i]< MinimoPrecio)                  // If < than known
      MinimoPrecio=Low[i];                    // it will be min
   if (High[i]> Maximum)                            // If > than known
      Maximum=High[i];                              // it will be max
  }
```

Si el valor actual Low [i] (es decir, durante el actual iteración con el índice del valor actual) es inferior al valor mínimo anotado hasta el momento, se convierte este en el nuevo valor mínimo. De la misma manera se calcula el valor máximo. Así, al final del ciclo, se obtienen las variables mínimo y maximo. En otras líneas se muestran estos valores.

El lanzamiento de este programa se obtiene un resultado como este:

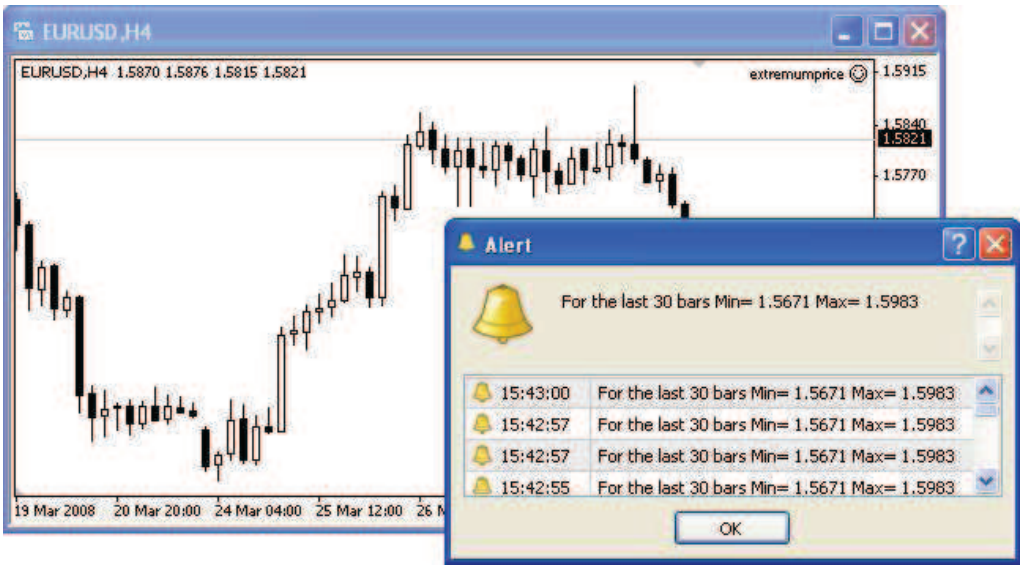


Fig. 63. Resultado de la AE extremumprice.mq4 operación.

Fijemosnos en que el Asesor Experto puede funcionar por tiempo indefinido mostrando los resultados correctos, y el programa va a utilizar el mismo índice de valores (en este caso, de 0 a 29). Los valores de elementos con el índice cero de los arrays-timeseries van a cambiar en el momento de una nueva cotización y los valores de los elementos de los arrays-timeseries que caracterizan a la barra de cero pueden cambiar en cualquier tick siguiente (excepto los valores de Open [] y Time [] que no cambian en la barra de cero).

En algunos casos es necesario para realizar ciertas acciones a partir del momento en que un bar se ha constituido plenamente. Esto es importante, por ejemplo, para la aplicación de algoritmos basados en un análisis candlestick. En estos casos, normalmente sólo los bares que se han formado plenamente se tienen en cuenta.

Problema 27. Al comienzo de cada barra mostrar un mensaje con el mínimo y máximo de precios entre los últimos n bares formados.

Para resolver la tarea, es necesario definir el hecho en el comienzo de una nueva barra, es decir, la detección de un nuevo tick en un bar cero. Hay una simple y fiable forma de hacer esto, analizar la hora de apertura de un bar cero. La hora de apertura de una barra cero, es la caracterista de la barra que no cambia durante la formación de la barra. Las demás caracteristicas de la barra pueden cambiar durante la formación de la barra, su High[0], Close[0] y el Volume [0]. Sin embargo, las características como Open[0] y Time[0] no cambian.

Por eso es suficiente recordar el precio de apertura del bar cero en cada tick y compararlo con el último precio de apertura del bar cero conocido. Tan pronto como un desajuste se encuentra, esto significa la formación de un nuevo bar (y la terminación del anterior). En la AE newbar.mq4 el algoritmo de detección de un nuevo bar se lleva a cabo en la forma de una función definida por el usuario:

```
//------------------------------------------------------------------
// newbar.mq4
// The code should be used for educational purpose only.
//------------------------------------------------------------------
extern int GV_CantidadBarras=15;            // Cantidad de barras
bool GV_Flag_NuevaBarra=false;              // Flag de una nueva barra
//------------------------------------------------------------------
int start()                                 // Special funct. start()
  {
   double MinimoPrecio,                     // variable que registra el precio minimo
          MaximoPrecio;                     // variable que registra el precio minimo
//------------------------------------------------------------------
   Fun_NuevaBarra();                        // Funcion call
   if (GV_Flag_NuevaBarra==false)           // Si no hay nueva barra..
      return;                               // ..return
//------------------------------------------------------------------
   int IndMax =ArrayMaximum(High,GV_CantidadBarras,1);// Indice de la barra del precio maximo
   int IndMin =ArrayMinimum(Low, GV_CantidadBarras,1);// Indice de la barra del precio minimo
   MaximoPrecio=High[IndMax];               // Registrar el maximo precio
   MinimoPrecio=Low[IndMin];                // Registrar el minimo precio
   Alert("Para las ultimas ",GV_CantidadBarras,          // Mostrar mensaje de precios max y min
   " barras Min= ",MinimoPrecio," Max= ",MaximoPrecio);
   return;                                  // Salir de start()
  }
//------------------------------------------------------------------
void Fun_NuevaBarra()                       // Descripción de la Funcion que detecta ..
  {                                         // .. una nueva barra
   static datetime NewTime=0;               // variable que almacena fecha y hora
   GV_Flag_NuevaBarra=false;                // Inicializa nueva barra a falso (no hay nueva barra)
   if(NewTime!=Time[0])                     // Si existe nueva barra el dato es distinto de cero..
     {
      NewTime=Time[0];                      //.. y en ese caso se registra el hora y fecha de la..
      GV_Flag_NuevaBarra=true;              //nueva barra y se activa el flag que señaliza la…
                                            //existencia de una nueva barra
     }
  }
//------------------------------------------------------------------
```

Se utiliza en el programa una variable global llamada GV_Flag_NuevaBarra. Si su valor es 'true', significa que el último tick es el primer tick de un nuevo bar (barra). Si el valor de la variable GV_Flag_NuevaBarra es false, el último tick aparecido esta dentro de la formacion de la actual barra cero.

Un **Flag** (bandera) es una variable, cuyo valor se define de acuerdo con algunos acontecimientos o hechos, es decir, una bandera se activa cuando ocurre algo.

El uso de banderas en un programa es muy conveniente. El valor del indicador se puede definir en un solo lugar y se utiliza en diferentes lugares. A veces, un algoritmo es utilizado en el programa, en el que se toma una decisión en función de la combinación de valores de diferentes banderas. La variable GV_Flag_NuevaBarra del Asesor Experto newbar.mq4 se utiliza como una bandera. Su valor depende directamente de la realidad de la formación de una nueva barra.

Los cálculos en cuanto a la detección de un nuevo bar se concentran en la función definida por el usuario Fun_NuevaBarra (). En las primeras líneas de la función se define la variable New_Time (recuerdese que las variables de tipo static no pierden sus valores después de la ejecución de la función que está por encima). En cada llamada a función, el valor de la variable global GV_Flag_NuevaBarra se establece como falsa «false». La detección de un nuevo bar se realiza en el operador 'if':

```
   if(NewTime!=Time[0])                     // Si existe nueva barra el dato es distinto de cero..
     {
      NewTime=Time[0];                      //.. y en ese caso se registra el hora y fecha de la..
      GV_Flag_NuevaBarra=true;              //nueva barra y se activa el flag que señaliza la…
     }
```

Si el valor New_Time (calculado en la anterior historia) no es igual a Time [0] de la barra cero, ello denota el hecho de la formación de una nueva barra. En tal caso el control se pasa al cuerpo del operador `if', cuando hay una nueva fecha y hora de apertura en la barra cero, este dato es distinto de lo que hay en NewTime y se actualiza el nuevo dato y tambien se activa el Flag GV_Flag_NuevaBarra (para mayor comodidad se puede decir que se levanta la bandera que indica que hay una nueva barra) .

Para la solución de etos problemas, es importante tener en cuenta la peculiaridad de utilizar diferentes banderas. En este caso la particularidad es el valor de GV_Flag_NuevaBarra (posición de la bandera) que debe actualizarse antes de que se utilice en el cálculo (en este caso, en la función especial start ()). El valor de GV_Flag_NuevaBarra se define en la función definida por el usuario, esto es por que debe ser llamada lo mas pronto posible en el programa, por ejemplo, antes de los primeros cálculos, en el cual se utiliza GV_Flag_NuevaBarra. La función especial start () se construye según corresponde: La llamada a la funcion definida por el usuario se realiza inmediatamente después de la declaración de variables.

El cálculo de los valores deseados merece la pena sólo si la función start () es lanzada por un tick con el cual un nuevo bar se forma. Es por eso que inmediatamente después de detectar una nueva barra en formación, se analiza la posición de la bandera en start (),(valor de GV_Flag_NuevaBarra):

```
Fun_NuevaBarra();                  // Funcion call
if (GV_Flag_NuevaBarra==false)     // Si no hay nueva barra..
  return;                               // ..return
```

Si el último tick que inició la ejecución de start () no forma un nuevo bar, el control se pasa a un operador que termina la ejecución de start (). Y sólo si hay una nueva barra que se forma, el control se pasa a las siguientes líneas para el cálculo de los valores deseados (lo cual es requerido por las previsiones del problema).

El cálculo del máximo y mínimo valor se realiza usando funciones estándar **ArrayMaximum ()** y **ArrayMinimum ()**. Cada una de las funciones devuelve el indice del elemento del array (el correspondiente valor maximo o mínimo) durante el determinado intervalo de índices. Debido a las condiciones que establece el problema sólo se deben analizar las barras que se han formado completamente, por eso el límite escogido de los valores del índice está entre 1 y GV_CantidadBarras (la barra cero no se ha formado aún y no se tiene en cuenta en los cálculos). Para obtener información más detallada sobre el funcionamiento de estos y otros funciones de acceso a timeseries, consultese la documentación sobre el sitio web del developper (http://docs.MQL4.com) o "Ayuda" en MetaEditor.

Fig. 64 muestra durante la ejecución del programa el cambio en los precios entre el máximo y el mínimo en el intervalo preestablecido:

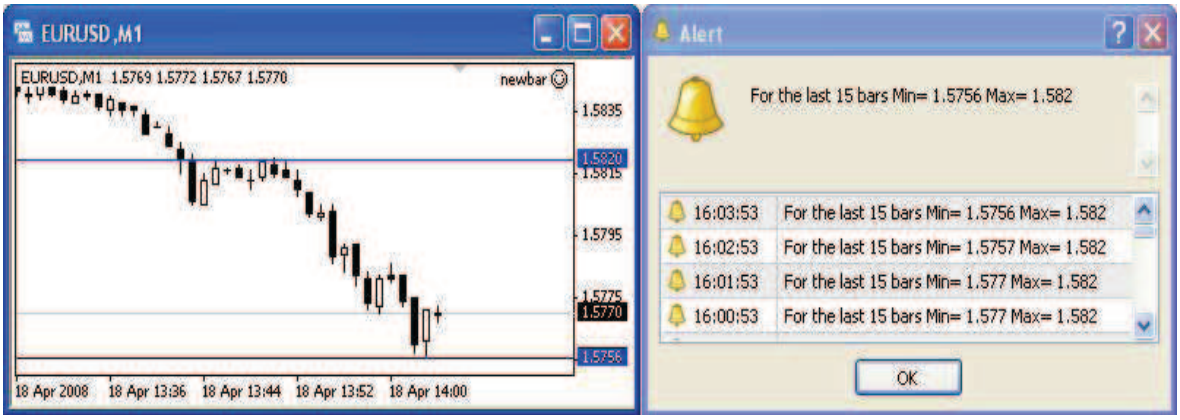


Fig. 64. Asesor Experto newbar.mq4 operación de resultado.

Prácticas de programación en MQL4

Indice de contenidos
http://book.mql4.com/content

Programación de operaciones de comercio

- Manera común hacer trades pag…… 4.
- Orden características y reglas para presentar órdenes pag…….. 10.
- Apertura y creación de órdenes pag……12.
- Cierre y supresión de órdenes. Función OrderSelect pag……46.
- Modificación de órdenes pag. ……64.

Programas Simples en MQL4

- El uso de los indicadores técnicos pag. ……75.
- Asesor Experto Simple pag. ……..……..83.
- Creación de indicadores personalizados pag…… 101.
- Indicador Personalizado ROC (Precio Tasa de Cambio) pag. ……..118.
- Utilización combinada de programas pag. ……..129.

Funciones estándar pag. 137

- Funciones comunes pag ……138.
- Objetos gráficos pag. ……142.
- Operaciones con Gráficos pag. ……154.
- Funciones de cadenas pag……163.
- Fecha y hora pag. ……..171.
- Operaciones de Archivo pag ……..177.
- Las matrices y Timeseries pag. ……..189.
- Funciones matemáticas pag. ……190.
- Funciones GlobalVariable pag…….. 195.
- Indicadores Personalizados pag. ……197.
- Datos de la cuenta pag……202.
- Funciones del Comercio pag. ……206.

Creación de un Programa Normal

- Estructura de un Programa Normal pag. 209.
- Orden de Contabilidad pag. 214.
- Función para mostrar datos pag. 219.
- Evento de seguimiento de eventos pag. 224.
- Funcion que calcula el volumen d elas ordenes pag 230.
- Funcion que define la estrategia pag. 234.
- Funciones del Comercio pag. 240.
- Error al procesar la función pag. 250.

Acerca de programas complejos pag. 252.

Apéndices

- Glosario
- Tipos de Operaciones
- Requisitos y limitaciones en la toma de Órdenes
- Códigos de error
- Estilos indicador de líneas
- Tipos y Propiedades de objetos gráficos
- Archivos de Sonido
- MessageBox () códigos de return
- MarketInfo () Identificadores
- Lista de Programas

Prácticas de programación en MQL4

Con el fin de realizar operaciones de comercio, la presente segunda parte del libro considera los siguientes temas: Los principios de la codificación y del uso de scripts simples, Asesores Expertos e indicadores, así como las funciones estándar que mas a menudo se utilizán en la programación en MQL4. Todas las secciones contienen algunos ejemplos de programas que están listos para usar, pero limitado al campo de aplicación.

En la sección debinubada Creación de programas normales se da un ejemplo que se puede utilizar como base para el diseño de su propio sencillo Asesor Experto para uso en el comercio real.

Todos los criterios comerciales que se indican a continuación se utilizan con fines educativos y no deben considerarse como directrices de negociación en las cuentas reales.

- **Programación de operaciones de comercio**

Al programar las operaciones de comercio, usted debe considerar las necesidades y limitaciones relacionadas con las características de las órdenes y las normas que rigen en su dealing center, así como las características especiales de la tecnología de ejecución de órdenes de trading. La sección proporciona una descripción detallada del orden de realización de las operaciones y contiene una gran cantidad de ejemplos que explican los efectos de todas las funciones comerciales utilizadas para formar las ordenes de comercio. La sección contiene algunos scripts listos para usar destinados a la aplicación restringida.

- **Sencillos programas en MQL4**

Después de que el programador ha dominado la programación de las operaciones de comercio, ya puede empezar a crear programas sencillos. La sección se refiere a los principios generales de la creación de un simple Asesor Experto y un simple indicador personalizado, así como las órdenes para compartir un Asesor Experto con diversos indicadores. En particular, en la sección se describe la orden de transferencia de datos del indicador personalizado de un AE. También da algunos ejemplos sencillos de programas listos para ser utilizados en la práctica comercial.

- **Funciones estándar**

En total, MQL4 cuenta con más de 220 funciones estándar, sin incluir las funciones de indicadores técnicos. Sería bastante difícil describirlos todos en este libro y dar ejemplos de cada función, teniendo en cuenta su gran cantidad. Algunas funciones que requieren explicaciones detalladas ya han sido considerados en las secciones precedentes. En la presente sección, consideramos las funciones estandar que más se utilizan y damos algunos ejemplos de cómo utilizarlas en los programas. Al final de cada subsección, ofrecemos la lista completa de funciones de una categoría determinada y su descripción breve.

- **Creación de programas normales**

Por regla general, después de haber practicado la codificación de algunas aplicaciones simples en MQL4, el programador suele abordar un proyecto más sofisticado. Entonces se crea un programa destinado a uso práctico. En algunos casos, los programas simples no satisfacen las necesidades de un programador de comercio al menos por dos razones:

1. La funcionalidad limitada de los programas simples no pueden proporcionar por completo al comerciante la información necesaria y de todas las herramientas de comercio, lo que hace que la aplicación de estos programas sea menos eficiente.

2. La imperfección el código de los programas simples hace que sea difícil mejorarlos a fin de incrementar sus servicios.

En la presente sección, les mostramos una posible versión para la realización de un Asesor Experto de comercio que pueden utilizarse como base para la creación de su propio proyecto.

Programación de operaciones de comercio

Al programar las operaciones de comercio, usted debe considerar las necesidades y limitaciones relacionadas con las características de las órdenes y las normas que rigen en su dealing center, así como las características especiales de la tecnología de ejecución de órdenes de trading. Esta sección proporciona una descripción detallada del orden de realización de las operaciones y contiene una gran cantidad de ejemplos que explican los efectos de todas las funciones comerciales utilizadas para formar las ordenes de comercio. La sección contiene algunos scripts listos para usar destinados a la aplicación restringida.

- Forma Común de comerciar
  Un comerciante o un programador de MQL4 sólo pueden dar órdenes de comercio, mientras los trades (ordenes de comercios), como tales, estén registrados en el servidor de comercio. El intermediario entre el servidor de comercio y un programa es el Terminal de Usuario. Las órdenes incorrectas serán rechazados de inmediato en el Terminal de Usuario, así se tiene que obtener una visión general de las ordes de trading.

- Características de las órdenes y normas para presentar los trades
  Las instrucciones de comercio se dan utilizando las órdenes de trading (trades). En la orden de trading se deben especificar múltiples parámetros, una parte de los cuales están determinados por los precios actuales y la dirección del mercado, otra parte depende del símbolo del comercio. Las órdenes que se entregan en el servidor de trading se verificarán en tiempo real para su cumplimiento según la situación actual y estado de la cuenta. Por eso es necesario, para efectuar las operaciones, un buen conocimiento de las reglas

- Apertura y colocación de órdenes
  La función más importante de trading es **OrderSend()**. Esta es la función que se utiliza para enviar las solicitudes al servidor de trading para abrir una orden de mercado o colocar una orden pendiente de ser ejecutada. Se puede especificar de inmediato el valor deseado del StopLoss y del TakeProfit (stop de toma de beneficios). Los valores incorrectos de estos parámetros, así como de los precios de apertura y el volumen de la orden (numero de acciones, número de lotes o número de contratos de una posición), pueden dar lugar a errores. Para procesar adecuadamente estos errores es importante conocer el uso de la Función **MarketInfo()** que permite minimizar la cantidad de dichos errores.

- Cierre y borrado de órdenes. Función OrderSelect
  Las ordenes de Mercado pueden ser cerradas utilizando la función **OrderClose ()**, mientras que órdenes a la espera (ordenes pendientes) pueden ser suprimidas con la función **OrderDelete ()**. Al enviar una petición de cerrar o eliminar un objeto, se debe especificar el ticket de esta orden. Vamos a seleccionar la orden necesaria usando la función **OrderSelect ()**. Por otra parte, si hay dos órdenes contrarias en un símbolo, se pueden cerrar al mismo tiempo, una a otra, utilizando la función **OrderCloseBy ()**. Cuando ejecute dicha orden usted se ahorrará un spread.

- Modificación de órdenes
  Los niveles TakeProfit y StopLoss se pueden modificar utilizando la función **OrderModify ()**. A las órdenes en espera de ser ejecutada, también se les puede cambiar el nivel de disparo. No se puede modificar el volumen de órdenes pendientes de ser ejecutadas. La modificación de órdenes de mercado y ordenes pendientes también tienen ciertos requisitos relacionados con la la forma correcta de hacer esa operación comercial. Si usted hace una actividad comercial es altamente recomendable, que los resultados del proceso de este comercio maneje los errores.

Manera común de hacer Trades

Todas las acciones y cálculos que se realizan en un programa de aplicación pueden dividirse en dos grupos atendiendo a su ubicación: los ejecutados en el PC del usuario y los ejecutados en el servidor. Una cantidad significativa de los cálculos se realiza en el lado del usuario. En este grupo se incluyen la ejecución de programas de aplicación. Las transacciones pertenecen al segundo grupo. Hacer comercio implica la conversión de datos en el servidor.

Teniendo en cuenta el comercio, vamos a distinguir los siguientes términos:

Market order: (orden de mercado) es la ejecución de una orden de compra o venta de activos en un título o valor. Una orden de mercado se muestra en la ventana de símbolo hasta que la orden queda cerrada.

Pending order: (orden en espera de ser ejecutada) es un trades para comprar o vender activos de un título o valor, cuando se alcanza un nivel de precios preestablecidos. La Pending order se muestra en la ventana de símbolo hasta que se convierte en una orden de mercado o se suprime.

Trade Request: (solicitud de comercio) es un comando hecho por un programa o por un comerciante con el fin de ejecutar una orden.

Trade: es la apertura, cierre o modificación de órdenes de mercado u órdenes en espera de ser ejecutada.

Diagrama del Trading

Tres componentes están involucrados en la ejecución de órdenes: el programa de aplicación, el Terminal de Usuario y el servidor (ver Fig. 65). La solicitud nace en el programa (como hemos mencionado anteriormente, todos los programas de aplicación son ejecutados sólo en el PC del usuario; los programas de aplicación no se instalan en el servidor). La solicitud nacida en el el programa se transmite al Terminal de Usuario que, a su vez, envía la solicitud al servidor. En el lado del servidor, se tomará la decisión de ejecutar o rechazar la solicitud. La información sobre los resultados obtenidos será aprobada por el servidor al Terminal de Usuario y, a continuación, hacia el programa.



Fig. 65. Diagrama de los requerimientos para efectuar las operaciones.

Ordenes de Comercio

La orden de comercio pueden ser realizados por un comerciante o un programa. Para que un comerciante pueda hacer una solicitud de comercio, el cliente terminal proporciona el panel de control de "órdenes" (ver la descripción del Terminal de Usuario). En el programa, las solicitudes se forman de acuerdo con un algoritmo y son el resultado de la ejecución de las funciones de comercio y en ningún otro lugar, ni en el Terminal de Usuario ni en el servidor. Las solicitudes u órdenes de comercio no se pueden formar espontaneamente.

Características del Programa

Dependiendo del algoritmo, un programa puede formar diferentes solicitudes para la apertura, cierre o modificación de órdenes de mercado y órdenes pendientes de ser ejecutadas. Para crear las ódenes de trading, en un programa se utilizan las siguientes funciones comerciales:

- **OrderSend ()** - para *abrir* órdenes al mercado y órdenes en espera de ser ejecutada;
- **OrderClose ()** y **OrderCloseBy ()** - para *cerrar* las órdenes de mercado;
- **OrderDelete ()** – Para *suprimir* las órdenes pendientes de ser ejecutadas;
- **OrderModify ()** - para *modificar* las ordenes de mercado y las órdenes pendientes de ser ejecutadas.

Las funciones de comercio de arriba solo pueden utilizarse en Asesores Expertos y scripts; el uso de estas funciones en los indicadores está prohibida (véase también el cuadro 2). Hay otras funciones que pertenecen al comercio (véase el archivo de ayuda en MetaEditor y la sección Funciones del Comercio en el presente libro). Sin embargo, su ejecución se utiliza para llamar al Terminal de información de entorno con el fin de obtener información de referencia, por lo que no da lugar a la formación de órdenes y llamadas al servidor.

Características del Terminal de Usuario

Una solicitud hecha por el programa como consecuencia de la ejecución de una función comercio se pasa al Terminal de Usuario para su procesamiento. El Terminal de Usuario analiza el contenido de la solicitud y realiza una de las siguientes dos acciones: o bien envía la solicitud al servidor para que pueda ser ejecutada en el servidor, o bien rechaza la solicitud y no la envia al servidor.

El Terminal de Usuario permite corregir solo las solicitudes para ser enviadas al servidor. Si el programa se codifica de tal manera que forma, por ejemplo, una solicitud para la apertura de una orden de la que no existe precio, el Terminal de Usuario no envia esta solicitud al servidor. Si el programa crea las solicitudes de forma correcta (las órdenes son abiertas y cerradas, con el mas reciente precio conocido, el valor de las ordenes esta dentro del rango limitado por el dealing center, etc), entonces esta solicitud será enviada al servidor.

Sólo un canal de ejecución está previsto en el Terminal de Usuario para realizar operaciones. Esto significa que el Terminal de Usuario solo puede trabajar con una orden cada vez. Si hay varios Asesores Expertos o scripts de comercio en el Terminal de Usuario y el programa ha pasado una petición de una orden al Terminal de Usuario, las solicitudes de comercio de los demás Asesores Expertos o scripts serán rechazados hasta que el Terminal de Usuario complete la tramitación de la solicitud actual, es decir, hasta que el canal del comercio esté libre.

Caracteristicas del Servidor

La información sobre la historia del comercio de cada cuenta (de apertura, cierre, modificación de órdenes) es de alta seguridad para el servidor y es de una prioridad más alta en comparación con la historia de las órdenes almacenadas en el Terminal de Usuario. El derecho a ejecutar las solicitudes de comercio sólo se concede a un corredor (dealer) o al servidor que procesa las solicitudes automáticamente (si el dealing center dispone el servidor con esta función durante un determinado periodo de tiempo). La solicitud que es entregada en el servidor puede ser ejecutada o rechazada. Si la solicitud es ejecutada (es decir, se ejecuta un trade), el servidor hará la conversión necesaria de datos. Si la solicitud es rechazada, el servidor no convierte ningún dato. Cualquiera que sea la decisión tomada(de ejecutar o rechazar una solicitud), la información acerca de esta decisión se transmitirá al Terminal de Usuario para sincronizar la historia.



La petición de trade creada como resultado de la ejecución de un programa y una petición de trade ejecutada por el trader de forma manual es absolutamente la misma desde el

> punto de vista del servidor. El servidor no distingue, ni puede distinguir si la solicitud se ha hecho de una u otra forma y por tanto, tampoco hace ninguna distinción, ni puede hacerla entre las solicitudes en su tratamiento.

También es posible en el lado del servidor no permitir el comercio de Asesores Expertos en el Terminal de Usuario. A veces es necesario, si la operación del programa causa conflictos. Por ejemplo, si la aplicación de un algoritmo resulta que el programa continuamente crea alternativas solicitudes de apertura y cierre de órdenes con muy pequeños intervalos de tiempo (por ejemplo, a cada tick), o si las solicitudes de apertura, cancelación o modificación de órdenes pendientes de ser ejecutadas son demasiado frecuentes.

Procedimiento de trading

El procedimiento de realización de operaciones es interactivo y se realiza en tiempo real. El diagrama de la (Fig. 66) muestra todos los eventos relacionados con el desempeño de un trade.

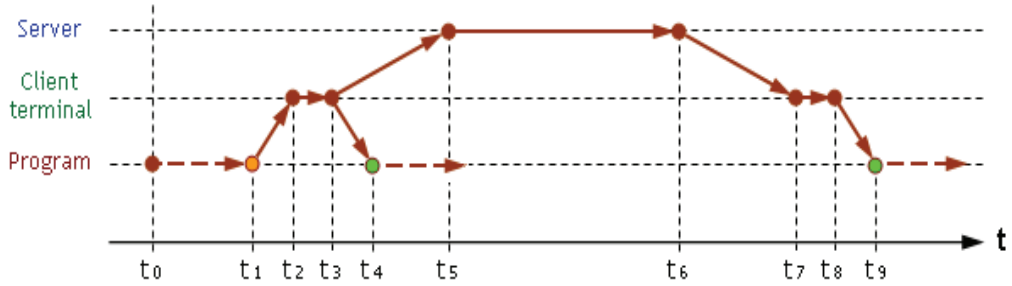


Fig. 66. Secuencia de eventos en la toma de un orden

Evento 0. El programa es lanzado para su ejecución en el momento t0.

Evento 1. En el momento t1, el programa ha formado la solicitud de un trade como consecuencia de la ejecución de una función comercial. La solicitud de comercio se pasa al Terminal de Usuario. En ese momento, el programa pasa el control al Terminal de Usuario y la ejecución del programa se detiene (el punto rojo en el diagrama).

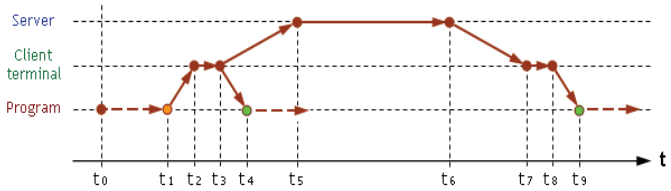
Evento 2. El Terminal de Usuario ha recibido el control y la información sobre la solicitud de contenidos. En el período de tiempo comprendido entre T2 y T3, el Terminal de Usuario analiza el contenido de la solicitud y toma una decisión sobre los nuevos acontecimientos.

Evento 3. El Terminal de Usuario ejecuta la decisión tomada (una de dos alternativas).

Alternativa 1. Si la solicitud de trade creada por la ejecución de una funcion de comercio ha resultado ser incorrecta, el control se pasa al programa. En este caso, el próximo evento será del evento 4 (esto puede suceder si, por ejemplo, el programa ha enviado la solicitud para la apertura de un pedido, cuyo valor es superior a la cuenta de capital disponible).

Evento 4. El programa ha recibido el control (el punto verde, momento t4) y puede continuar la ejecución desde el lugar en que la solicitud ha sido previamente formada. En el mismo momento que el programa ha recibido la información acerca de que la orden comercial no ha sido ejecutada, se puede encontrar información sobre la razón por la que la solicitud no se ha ejecutado, mediante el análisis de la código de la devolución del error. A continuación vamos a examinar la cuestión de cómo se hace esto. En este caso, sólo debe tenerse en cuenta que no todas las solicitudes dan como resultado la ejecución de órdenes. En este caso, el programa ha formado una petición incorrecta, lo que se traduce en que el Terminal de Usuario ha rechazado esta solicitud y ha devuelto el control al programa. Los intervalos de tiempo entre t1 - t2 - t3 - t4 son significativamente cortos y no exceden de unos pocos ms en total.

Alternativa 2. Si el programa ha formado una petición de trade correcta, el Terminal de Usuario envía esta petición al servidor; el próximo evento será Evento 5 (el momento de t5) el servidor recibe la solicitud. La conexión entre el Terminal de Usuario y el servidor se establece a través de Internet, por lo que el tiempo empleado en el envío de la solicitud al servidor (intervalo de tiempo entre t3 y t5) es completamente dependiente de la calidad de la conexión. Para una buena calidad de conexión, este período de tiempo puede ser aproximadamente de 5 a 10 ms, mientras que para una mala conexión este tiempo puede ser medido en segundos.



Evento 5. Por el momento t5, el servidor ha recibido la solicitud. El servidor puede ejecutar o rechazar esta solicitud recibida. La decisión sobre la ejecución o el rechazo de la solicitud se hace en el lado del servidor en un plazo determinado de tiempo (en el momento t6). El intervalo de tiempo entre T5 y T6 puede ir desde algunos milisegundos a las decenas de segundos, dependiendo de la situación. En algunos casos, si el servidor funciona en el modo automatizado, no hay movimientos rápidos en el mercado y los demás comerciantes no están muy activos, la solicitud puede ser ejecutada o rechazada dentro de unos pocos milisegundos. En otros casos, si el servidor está sobrecargado debido a la elevada actividad de los comerciantes y si la decisión sobre la ejecución o el rechazo de la solicitud es hecha por un broker humano, el tiempo dedicado por tomar la decisión puede tomarse en consideración las decenas de segundos.

Evento 6. Si no se producen cambios considerables e en el mercado dentro del intervalo de tiempo desde el momento de formar la solicitud por el programa (t1) hasta el momento de la toma de decisión por el servidor (T6), por regla general la solicitud será ejecutada. Si el precio del símbolo ha cambiado en este plazo o el valor de la orden de apertura se excede del capital disponible en la cuenta en el momento de tomar la decisión, u se producen otros obstáculos o impedimentos, entonces el servidor decide rechazar la solicitud.

El rechado de solicitudes de comercio por el servidor es común, aunque ya hayan sido verificadas por el Terminal de Usuario. En general, la mayor parte de los trades que se envian para que sean entregados al servidor, se aceptan para la ejecución por el servidor. Sin embargo, en algunos casos, la solicitud puede ser rechazada, de modo que su programa de aplicación debe ser codificado de tal forma que tenga en cuenta esa posibilidad y funcione correctamente en este tipo de situaciones.

Sea cual sea la decisión (ejecutar / rechazar una petición de comercio, evento 6) que haga el servidor, la información sobre la misma es enviada por el servidor al Terminal de Usuario, que es quién ha entregado la solicitud.

Evento 7. El Terminal de Usuario ha recibido la respuesta del servidor. El servidor responde por el mismo camino por donde se le entregó la solicitud a través de Internet, por lo que el tiempo dedicado a la recepción de la respuesta del servidor depende completamente de la calidad de la conexión. De acuerdo con las modificaciones introducidas en el servidor, el Terminal de Usuario refleja los cambios correspondientes. Por ejemplo, si la ejecución de una petición de comercio ha resultado en el cierre o la apertura de una posición, el Terminal de Usuario mostrará este evento gráficamente en la ventana de símbolo y textualmente en la ventana del Terminal (las pestañas "Operaciones" e "Historial de cuantas"). Si el servidor ha rechazado la solicitud, no se harán cambios en las ventanas de la Terminal de Usuario.

Evento 8. El Terminal de Usuario ha completado la muestra de los cambios y pasa el control al programa.

Evento 9. El programa ha recibido el control y puede seguir funcionando.

Tengase en cuenta que:



Desde el momento en que el programa envía una petición de comercio (y al mismo tiempo pasa el control) al Terminal de Usuario, al momento en que el control se devuelve al programa, éste se encuentra en modo de espera. No se realizan operaciones en el programa durante este periodo de tiempo. El control se devuelve al programa de acuerdo a

las reglas de ejecución de llamadas a función que ha formado la solicitud.

Si la solicitud es incorrecta, entonces el programa no estará en modo de espera durante mucho tiempo (el intervalo entre t1 y t4). Sin embargo, si la solicitud es "aprobada" por el Terminal de Usuario y enviada al servidor, la duración del período de espera de programa (t1-t9) puede ser diferente y depende de la calidad de la conexión y del tiempo que el servidor tarde en la toma de decisiones. Este tiempo pude llevar desde milisegundos hasta varios minutos.

Tan pronto como el programa recibe el control, puede seguir funcionando. El programa operativo puede analizar el código del último error devuelto por el Terminal de Usuario y, de esta manera, conocer si la solicitud fue ejecutado o se rechaza.

Los conflictos en la toma de Órdenes. Error 146

Cuando se habló de las características del Terminal de Usuario, se mencionó que el Terminal de Usuario solo podría atender una única petición a la vez. Vamos a examinar ahora qué eventos se llevarán a cabo en caso de que se formen varias solicitudes de diferentes programas que pasan al Terminal de Usuario.

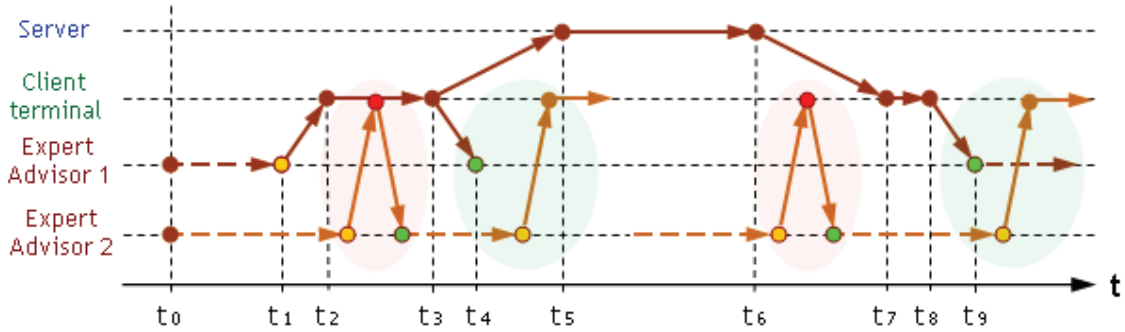


Fig. 67. Los conflictos en pasar varias peticiones a la Terminal de Usuario de diferentes programas.

En la Fig. 67, podemos ver que dos Asesores Expertos comerciales se ponen en marcha para su ejecución en el Terminal de Usuario de forma simultánea. EA1 formó una petición de comercio en el momento t1 y pasado al Terminal de Usuario el momento t2.

EA2 también ha creado una petición y se refiere al Terminal de Usuario cuando éste está procesando la primera solicitud (período comprendido entre el t2 y t3). En esta situación, el Terminal de Usuario no puede considerar la solicitud formada por EA2, por lo que rechaza la solicitud y la devuelve. Hay que tener en cuenta que, en este caso, la petición es rechazada por el Terminal de Usuario no por que la solicitud sea incorrecta, sino porque el terminal está ocupado con el procesamiento de otra solicitud. EA2 seguirá en funcionamiento. Se puede analizar el código de error que explica la razón por la cual la solicitud ha sido rechazada (en nuestro caso, es el error 146).

Si se trata de EA2 (en general, pueden ser uno o varios programas comerciales) que pasa su petición al Terminal de Usuario en el plazo de tiempo entre t1 y t4, entonces esta solicitud es denegada (grupo de acontecimientos de la zona rosa). El Terminal se convierte en terminal libre en el momento t4 (punto verde). A partir de este momento, EA2 puede pasar con éxito su petición al Terminal de Usuario (el grupo de acontecimientos de la zona verde). Esta solicitud recibida, la examina el Terminal de Usuario que puede, finalmente, rechazar también la petición, pero esta vez la razón sería un error en la petición, o por el contrario, puede también aceptar la petición y enviarla al servidor.

Si la solicitud creada por EA1 es considerada correcta por el Terminal, éste la mandará al servidor en el momento t3. En este caso, el Terminal se pone en modo de espera y no puede considerar ninguna otra solicitud de comercio. El Terminal de Usuario sólo estará libre para considerar otras solicitudes de comercio en el momento t9. Así, según La variante 2, el Terminal de Usuario no puede analizar solicitudes de comercio en el plazo de tiempo entre t1 y t9. Si en este plazo, cualquier programa se refiere al Terminal de Usuario con el fin de que se apruebe una solicitud de comercio, el Terminal de Usuario rechazará este evento y pasará el control al programa (grupo de acontecimientos de la zona rosa en el plazo de tiempo que transcurre entre t6 y 7). El programa que ha recibido el control continúa con su operación y, analizando el código de error, puede encontrar información sobre la razón por la cual la solicitud ha sido rechazada (en este caso, es el error 146).

A partir del momento t9, el Terminal de Usuario será completamente libre para el análisis de cualquier otra solicitud de comercio. EA2 puede pasar con éxito la solicitud al Terminal de Usuario en el plazo de tiempo que sigue al momento t9. Según como el Terminal de Usuario considere que esta solicitud es correcta o no, la solicitud será aprobada por el Terminal de Usuario y enviada al servidor o rechazada.

El análisis de los errores que ocurren en la operaciones de comercio se considerará de forma más detallada en las siguientes secciones.

Orden, características y reglas para presentar órdenes

Antes de empezar a describir las funciones de comercio, hay que tener en cuenta los parámetros que caracterizan a los precios del mercado, tipo de órdenes, sus características, así como las normas para presentar estas órdenes.

Características de los símbolos

En primer lugar, hay que tener en cuenta el principio utilizado por las empresas de corretaje para formar los precios de los valores mobiliarios. Este principio consiste en que el broker ofrece al operador una vía de doble sentido de la cotización para el desempeño de las órdenes.

Two-way quote Es un par de precios de mercado conectados de compra y venta de activos de títulos (símbolo) en el momento actual ofrecidos por el agente.

Bid es el más bajo de los dos precios ofrecidos por corredor en **Two-way quote** para una cotización del símbolo de un valor. Bid es el dinero que ofrecen (**oferta** de dinero) por la compra de un título o valor. Es el precio de oferta de los compradores. Es el precio al que los compradores estan dispuestos a comprar y por tanto es el precio que se cobraría si se quisiera vender ese título o símbolo en ese momento. Resumiendo Bid= precio si se quiere vender.

Ask es el mayor de los dos precios ofrecidos por el corredor en **Two-way quote** para una cotización del símbolo de un valor. Ask es el precio que piden (**demanda** de dinero) por la venta de un título o valor. Es el precio de oferta de títulos de los vendedores. Es el precio al que los vendedores estan dispuestos a vender y por tanto es el precio que se pagaría si se quiere comprar ese título o símbolo en ese momento. Resumiendo Ask = precio si se quiere comprar.

Point (punto) es la unidad de medición para el precio de un símbolo (el mínimo cambio de precio posible, la última cifra significativa de los precios del valor).

Spread es la diferencia entre el mayor y el menor precio en puntos en el **Two-way quote** para en una cotización del símbolo de un valor.

Normalmente, el spread es un valor fijo. En MetaTrader 4, es posible mostrar en la ventana del gráfico del símbolo que refleje solamente cambios en los precios de oferta Bid (botón derecho, propiedades, y en la pestaña comun marcar o no "mostrar linea de demanda"):



Fig. 68. Un precio normal para trazar un símbolo.

La Fig. 68 muestra una ventana de símbolo donde podemos ver los cambios de precios de la oferta de compra (Bid) y la Two-way quote, compuesta por la línea del actual precio oferta de compra, Bid (linea negra, 1.3005) y la línea de actual de demanda venta (Ask) con precio rojo (1,3007). Puede verse fácilmente que, en este caso, el intermediario ofrece un diferencial de 2 puntos. La historia de los precios de demanda no se muestra en el gráfico, pero está implícita y puede ser fácilmente calculada en cualquier momento del tiempo.

Tipos y Características de las Órdenes

Hay seis tipos de órdenes en total: dos tipos de órdenes de mercado y cuatro tipos de órdenes pendientes de ser ejecutadas (o en espera).

Buy (Comprar) es una orden de mercado que define la orden de compra de activos para un símbolo.

Sell (Vender) es una orden de mercado que define la orden de venta de activos para un símbolo.

Buy Limit (Compra a precio limitado) es una orden pendiente de ser ejecutada para comprar activos de un título por un importe inferior al actual. La orden será ejecutada (modificada a una orden de mercado para compra) si el precio de demanda (Ask) alcanza o cae por debajo del precio fijado en la orden de espera.

SellLimit (Venta a precio limitado) es una orden pendiente de ser ejecutada para vender activos de un título a un precio superior al actual. La orden será ejecutada (modificada a una orden de mercado para venta) si el precio de oferta (Bid) alcanza o supera el precio fijado en la orden en espera de ser ejecutada.

BuyStop (Compra mediante stop) es una orden pendiente de ser ejecutada para comprar los activos de un título a un precio superior al actual. La orden será ejecutada (modificada a una orden de mercado para compra) Si el precio de la demanda (Ask) alcanza o supera el precio fijado en la orden en espera de ser ejecutada.

SellStop (Venta mediante stop) es una orden pendiente de ser ejecutada para vender los activos de un título por un importe inferior al actual. La orden será ejecutada (modificada a una orden de mercado para venta) si el precio de la oferta (Bid) alcanza o cae por debajo del precio fijado en la orden en espera de ser ejecutada.

Lot (lote) es el volumen de una orden expresado en cantidad de lotes.

StopLoss es una orden de stop. Es el precio fijado por el comerciante, en el que se cerrará una orden de mercado si los precios de un titulo se mueven en una dirección tal, que la orden que tenemos en el mercado produce pérdidas.

TakeProfit es una orden de stop. Es el precio fijado por el comerciante, en el que se cerrará una orden de mercado si los precios de un titulo se mueven en una dirección tal, que la orden que tenemos en el mercado produce beneficios.

Trading requisitos y limitaciones

Con el fin de crear correctas peticiones de trade en los programas de aplicación. (de Asesores Expertos y scripts), debe tener en cuenta la existencia de requisitos y limitaciones. Vamos ahora a examinarlos con más detalles.



Todas las transacciones se realizan al precio correcto. El precio de ejecución de cada transación se calcula sobre la base del precio correcto de un Two-way quote.

La norma anterior es la regla común que rige para todos los participantes en el mercado y no se puede cambiar a voluntad de los desarrolladores de una plataforma de negociación o sobre la base de un acuerdo entre un intermediario y un comerciante. Esto significa, por ejemplo, que una orden de mercado sólo puede ser abierta al actual precio de mercado y no a cualquier otro precio. A continuación se considera el procedimiento correcto de cálculo del precio para las distintas órdenes.

Al calcular los precios correctos, también es necesario tener en cuenta las limitaciones del proveedor de servicios (dealing center). Estas limitaciones incluyen la distancia mínima y la congelación de la distancia. Estas limitaciones implican que el corredor necesita un tiempo para los preparativos para la realización de nuevas órdenes, ya sea la conversión de una orden espera en una de mercado o de una de cierre a una orden de stop.

El Dealing Centers limita el valor de la diferencia mínima admisible entre el precio de mercado y los requerimientos de precio de cada orden de stop de una orden de mercado, entre el precio de mercado y el precio solicitado de una orden pendiente de ser ejecutada, así como entre el precio solicitado de una orden pendiente de ser ejecutada y el requerimiento de precio de sus órdenes de stop. Esto significa, por ejemplo, que en un trade (orden de comercio), para solicitar la apertura de una orden de mercado sólo se puede especificar la orden precio de stop de los valores que no distan del actual precio una distancia inferior a la mínima. Una petición de trade que contiene un precio de orden de stop cuya distancia a los precios de mercado está más proxima que la **distancia mínima** que es considerada por el Terminal de Usuario como incorrecta. Los diferentes dealing centers pueden establecer diferentes limitaciones específicas para la distancia mínima permitida. Por regla general, el valor de esta distancia varía entre 1 y 15 puntos. Para los valores más comúnmente utilizados (en EUR/USD, GBP/USD, EUR/CHF, etc), esta distancia viene a ser en la mayoría de los brokers de 3-5 puntos. Diferentes valores pueden tener diferentes **distancias mínimas** permitidas, también. Por ejemplo, este valor puede ser 50-100 puntos en el oro. El valor de la distancia mínima en cualquier símbolo puede ser cambiado por el corredor en cualquier momento (esto normalmente precede a la difusión comercial de una noticia importante). Para la distancia máxima no hay limitaciones.

La distancia de congelación limita la posibilidad de modificar los precios de apertura de sus órdenes pendientes de ser ejecutadas, así como los requerimientos de los niveles de stop de las órdenes de mercado que se encuentran en la zona de congelación. Esto significa, por ejemplo, que si el precio de mercado es 1.3800, y la orden pendiente de ser ejecutada esta situada para ser abierta en 1.3807 y la prescripción del broker es de 10, su orden de espera se encuentra en la zona de congelación, es decir, no se puede modificar o borrar. En un mercado en calma, los intermediarios no suelen establecer una distancia de congelación, es decir, su valor = 0. Sin embargo, durante el período anterior a noticias importantes o en alta volatilidad, el corredor podrá fijar un valor determinado de una distancia de congelación. En condiciones diferentes y para diferentes intermediarios, este valor puede variar desde 1 a 30 puntos para los símbolos basicos y tener valores más altos para otros símbolos. La empresa de corretaje puede cambiar el valor de la distancia congelación a su propia discreción en cualquier momento.



Las limitaciones de los niveles de precios limitados por el valor de la distancia mínima y la distancia de congelación, se calculan sobre la base del precio correcto.

De las órdenes de apertura/cierre de mercado.

La apertura de una orden de mercado implica la compra o venta de algunos activos de un símbolo al precio actual de mercado (ver requisitos y limitaciones en la toma de Órdenes). Para abrir una orden de mercado se utiliza la función **OrderSend ()**; para su cierre se utiliza la función **OrderClose ().**



El precio de apertura correcto de una orden de *compra* a mercado es el último precio de mercado conocido **Ask**.

El precio de apertura correcto de una orden de *venta* a mercado es el último precio de mercado conocido **Bid**.

La limitación en relación con la posición del nivel de stop para abrir una orden de mercado se calcula sobre la base del precio correcto utilizado para el cierre de la orden.



Las órdenes StopLoss y TakeProfit no se pueden situar más cerca del precio de mercado que la distancia mínima.

Por ejemplo, la distancia mínima para EURUSD se establece en 5 puntos. La orden de venta a mercado **Sell** ha abierto en **Bid** =1,2987. El precio correspondiente a two-way cotización utilizada para cerrar esta orden de venta es **Ask** = 1,2989. Los siguientes niveles de stop serán los más cercanos al precio actual para el cierre de la posición. (ver Fig. 69 y requisitos y limitaciones de trading):

StopLoss = Ask + distancia mínima = 1,2989 + 0,0005 = 1.2994, y

TakeProfit = Ask - distancia mínima = 1,2989 - 0,0005 = 1,2984.



Fig. 69. Mercado "vendido" con los Stops al ámbito más próximo al precio de mercado.

Si la solicitud de apertura de órdenes stop de un mercado vendido un Bid = 1.2987 se usa un valor del nivel de stop más proximo posible, es decir, a la distancia que hemos visto antes (SL = 1.2994 y TP = 1.2984), el Terminal de Usuario rechazará la solicitud. Esto es debido a que se debe tener en cuenta los posibles deslizamientos de los precios durante la apertura de los órdenes, que da lugar a la apertura de su orden a un precio que no es el especificado en la solicitud, hecha a determinado valor. Si la misma petición ha especificado los valores del stop situados lo más próximo a los niveles de precio que se solicitó en la apertura, esta solicitud también será rechazada por el Terminal de Usuario, ya que, en este caso, la solicitud no cumple con la distancia mínima requerida entre precio de apertura de su orden y el precio solicitado de una de las órdenes de stop. Esta es la razón por la que no se recomienda el uso de solicitudes de comercio para apertura de órdenes a mercado al valor de las órdenes de stop colocadas a la máxima próximidad posible al precio de apertura solicitado. Por el contrario, se recomienda tener algun "free play" (juego libre), es decir, colocar los stops con cierta holgura de la distancia mínima. Por ejemplo, especificar los valores de ordenes stop de tal manera que la distancia desde la solicitud de la orden de apertura este al menos 1-2 puntos más lejos que el valor de la distancia mínima permitida.

Las órdenes de mercado se pueden cerrar como consecuencia de la ejecución de la solicitud realizada por cuenta del comerciante o del programa, así como, cuando el precio alcanza a uno de los niveles de precios que se han especificados en las órdenes stop.



El precio correcto de cierre de un mercado "comprado" es el último precio conocido de **Bid** del mercado.
El precio correcto de cierre de un mercado "vendido" es el último precio conocido de **Ask** del mercado.

Si cerramos una orden Sell, (Fig. 69) en el momento actual la posición será cerrada a precio de **Ask**= 1.2989, es decir, con una pérdida de 2 puntos. Si permitimos que nuestra orden permanezca abierta durante un tiempo en el que se estan produciendo caida hasta que el prico de **Ask** llega a 1.2984, la orden será cerrada a ese precio con el beneficio de 3 puntos. Si el precio de mercado crece durante este periodo de tiempo y alcanza Ask = 1.2994, la orden será cerrada a ese precio con una pérdida de 7 puntos.

Si la aplicacion ha formado una solicitud de apertura o cierre de una orden de mercado a un precio que no se corresponde con el último precio de mercado conocido, la solicitud será rechazada por el Terminal de Usuario.

La limitación en relación con el cierre de órdenes de mercado se calcula sobre la base del precio correcto de mercado utilizado para el cierre de la orden.

La orden no puede ser cerrada, si el precio de ejecución de su StopLoss o TakeProfit está dentro del rango de distancia de congelación del precio de mercado.

Por ejemplo, la orden que se muestra en la Fig. 69 se puede cerrar sólo si los corredores han establecido congelar el valor a una distancia de 4 puntos o menos a partir del momento de cierre. El precio de apertura de esta orden no importa en este caso. El régimen de la banda de congelación de la orden se calcula sobre la base del precio de mercado. Por lo tanto, si es = 4, el precio de la congelación del borde superior es igual a + =1,2989 + 0,0004 = 1.2993, mientras que el precio más bajo de la congelación es, en consecuencia, - =1,2989 - 0,0004 = 1,2985. En estas condiciones, la orden stop no está en la zona de congelación, por lo que la orden puede ser cerrada si el comerciante (o un programa) envía una solicitud correcta para el servidor. Si el corredor se ha fijado la distancia =5, como el momento actual, el régimen de la banda de congelación será, 1.2994 y 1.2984, respectivamente. En este caso, los dos stop se encuentra en la banda de congelación, es decir, han sido sometidos a la limitación prevista por el corredor, por lo que la orden no puede ser cerrada a iniciativa del comerciante o por solicitud del programa de comercio. En este ejemplo, las dos ordenes de stops estan sometidas a limitación. En general, una orden de mercado no puede ser cerrada por iniciativa del Terminal de Usuario si, al menos, un nivel de stop de esta orden se encuentra en la zona de congelación.



Fig. 69. Mercado vendido con los Stops al ámbito más próximo del precio de mercado.

Si dos órdenes de mercado son abiertas simultáneamente en un símbolo y una de ellas es de compra y otra es de venta, pueden cerrarse de una de dos maneras: se pueden cerrar una a una de forma consecutiva, usando **OrderClose()**; o bien se puede cerrar una de ellas contra la otra utilizando **OrderCloseBy()**. En términos de ahorro de dinero, la segunda manera es preferible porque, cerrando una órden contra otra, se ahorra un spread. El uso de funciones de comercio se considerará en este libro mas adelante con más detalles.

Colocación y supresión de órdenes pendientes de ser ejecutadas

Una orden a la espera (o pendiente) de ser ejecutada implica el requerimiento de apertura de una orden a otro precio distinto al precio actual de mercado. Para colocar órdenes en espera de ser ejecutadas se utiliza la función **OrderSend()**, y la función **OrderDelete ()** para eliminarla.

Las órdenes pendientes de ser ejecutadas **SellLimit** (Venta a precio limitado) **y BuyStop** (Stop de compra) se colocan a un precio que es superior al precio actual de mercado, mientras que **BuyLimit** (compra a precio limitado) **y SellStop** (stop de venta) se colocan a un precio que es inferior al precio actual de mercado.

La limitación en relación con la posición de una orden pendiente de ser ejecutada se calcula sobre la base del precio correcto de mercado para la conversión de una orden pendiente de ser ejecutada a una orden de mercado.

Las órdenes en espera de ser ejecutadas **BuyLimit, BuyStop, SellLimit y SellStop** no se pueden colocar a un precio que esté más cerca del precio de mercado que la distancia mínima.

Por ejemplo, para calcular el precio mínimo permitido por la orden BuyStop, se debería añadir el valor de la distancia mínima al último Ask de precios conocido. Si StopLevel = 5, entonces el precio mínimo permitido para colocar en espera la orden BuyStop será 1.3003 + 0.0005 = 1.3008 (véase la Fig. 70). Esto significa que para BuyStop se puede colocar en el momento actual la solicitud de precios a 1.3008 o un precio más elevado. En este ejemplo, BuyStop se coloca en 1.3015, lo cual es admisible.

El precio solicitado en espera para BuyStop es 1,3015.

El precio solicitado en espera para SellLimit es 1,3012.

El precio solicitado en espera para SellStop es 1,2995.

El precio solicitado en espera para BuyLimit es 1,2993.



Fig. 70. Las órdenes en espera de ser ejecutadas se realizan a un precio inferior o superior al precio actual.

En el ejemplo anterior, todas las órdenes pendientes de ser ejecutadas se colocaron en el bar cero (barra cero) en este momento según se muestra en la Fig. 70, mientras la distancia mínima para la puesta en espera de las órdenes dictadas son 5 puntos. La Orden SellStop es la más cercana al precio de mercado. En este caso, el Bid es 1.3001 y el precio solicitado de SellStop = 1,2995. De este modo, la distancia entre el objeto y el precio correcto de las cotizaciones de two-way (BID) es de 6 puntos (1,3001 - 1,2995), es decir, distancia mayor que la mínima exigida. Esto significa que, a la apertura de la orden (o todas las demás órdenes de este ejemplo), la solicitud será "aprobada" por el Terminal de Usuario y enviada al servidor. El servidor ha comprobado también el cumplimiento de los requisitos y decide ejecutar la solicitud de puesta en espera de la orden (ver requisitos y limitaciones en la toma de Órdenes).

La posición ó colocación de órdenes de stop asociada a las órdenes en espera de ser ejecutadas está también limitada por la distancia mínima:

La limitación relacionada con la posición de órdenes stop correspondiente a una orden pendiente de ser ejecutada determinada se calcula sobre la base del precio de apertura solicitado para esa orden pendiente de ser ejecutada y no tiene ninguna relación con los

precios de mercado.

El StopLoss o TakeProfit correspondientes a una determinda orden en espera de ser ejecutada no puede situarse más cerca del precio actual que la distancia mínima.

La posición de StopLoss y TakeProfit correspondientes a órdenes pendientes de ser ejecutadas no están limitadas por la distancia de congelación.

En la Fig.71, podemos ver la orden en espera de ser ejecutada de SellLimit y las ordenes stop cuyo precio está lo mas cerca posible del precio requerido (sell limit). En este caso, se solicitó la orden de precios=1.2944, el StopLoss=1.2949, TakeProfit=1.2939. En la distancia mínima de 5 puntos, estos valores son bastante aceptables.



Fig. 71. Orden pendiente de ser ejecutada sell limit con sus órdenes de stop la distancia mínima de su orden pendiente.

En este ejemplo, la orden de espera SellLimit fue enviada a las 18:07. Se puede ver en la Fig. 71 que, después el precio del mercado llegó y se cruzó con una de las órdenes de stop y luego bajó de nuevo. Vemos que este evento no influye en la orden en espera de ser ejecutada de ninguna manera: una orden de stop sólo se puede cerrar si hay una orden en mercado, es decir, que se hace efectiva tan pronto como la orden pendiente de ser ejecutada se convierte en una orden de mercado. En este caso, la orden en espera no se modifica en una de mercado, ya que el precio de Bid no ha llegado al nivel el precio requerido de orden de apertura (orden sell limit), por lo que el cruce con el nivel de precio del stop no se ha traducido en cambio alguno.



La limitación en relación con la supresión de órdenes pendientes de ser ejecutadas se calcula sobre la base del precio correcto de mercado aplicable para la modificación de una orden pendiente de ser ejecutada en una de mercado.
Las órdenes en espera de ser ejecutadas BuyLimit, BuyStop, SellLimit y SellStop no se pueden eliminar, si el precio solicitado de la orden de apertura se encuentra dentro del rango de la distancia de congelación de los precios de mercado.

La orden SellLimit puede suprimirse en el momento que se muestra en la Fig. 71, se iniciaría por el Terminal de Usuario sólo si el valor especificado en ese momento es igual o superior a 8 puntos. En este caso, la parte superior de la banda de congelación (que se calcula para SellLimit) será: + = 1,2935 +0,0008 = 1,2943. El requerimiento de precio de la orden de apertura se hace a 1.2944, de este modo, la orden se coloca fuera de la banda de congelación y la orden en espera de ser ejecutada SellLimit puede ser eliminada. Si el corredor (broker) establece el valor a más de 8 puntos, la orden en espera de ser ejecutada SellLimit no podría ser suprimida y el Terminal de Usuario rechazaría la solicitud.

Modificación de órdenes pendientes de ser ejecutadas a órdenes a mercado

Las órdenes en espera de ser ejecutadas son modificadas automáticamente en órdenes de mercado en el servidor, por que no existen funciones especificas para ejecutar esta operación (ver requisitos y limitaciones en la toma de Órdenes).



Las órdenes en espera de ser ejecutadas BuyLimit y BuyStop se modifican a órdenes a mercado, si el último precio conocido **ask** llega a los precios que solicitó la orden en espera de ser ejecutada.

Las órdenes en espera de ser ejecutadas SellLimit y SellStop se modifican a órdenes a mercado, si el último precio conocido **bid** llega a los precios que solicitó la orden en espera de ser ejecutada.

En cuanto a las órdenes en espera de ser ejecutadas que se muestran en la Fig. 70, podemos decir lo siguiente.



Fig. 70. Las órdenes en espera de ser ejecutadas se realizan a un precio inferior o superior al precio actual.

La orden en espera de ser ejecutada BuyStop se modifica en orden a mercado para **comprar**, si el precio actual alcanza el valor **ask** de 1,3015.

La orden en espera de ser ejecutada SellLimit se modifica en orden a mercado para **vender**, si el precio actual alcanza el valor **bid** de 1,3012.

La orden en espera de ser ejecutada SellStop se modifica en orden a mercado para **vender**, si el precio actual alcanza el valor **bid** de 1,2995.

La orden en espera de ser ejecutada BuyLimit se modifica en orden a mercado para **comprar**, si el precio actual alcanza el valor **ask** de 1,2993.

Los acontecimientos posteriores relacionados con estas órdenes se muestran en las Figuras 72-74.



Fig. 72. Modificación de órdenes en espera de ser ejecutadas a órdenes a mercado.

En el historico, las otras 2 órdenes en espera de ser ejecutadas se modificaron en órdenes a mercado.



Fig. 73. Modificación de órdenes en espera de ser ejecutadas a órdenes a mercado.

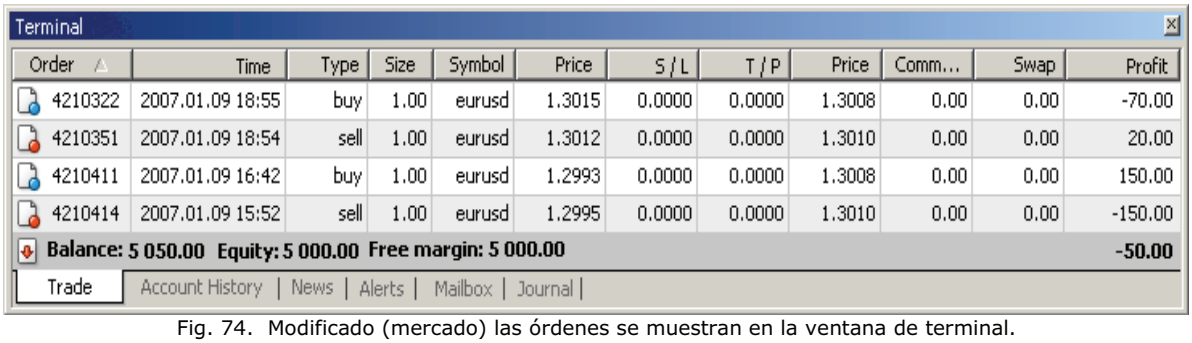


Fig. 74. Modificado (mercado) las órdenes se muestran en la ventana de terminal.

Tengase en cuenta que la Fig. 73 muestra la apertura de orden Compra 4.210.322 (formada por la orden en espera de ser ejecutada BuyStop). Como es fácil de ver, la barra formada a 18:55 no toca el precio de 1,3015. El precio más alto dentro de este bar es 1,3013. Al mismo tiempo, la ventana del terminal (Fig. 74) muestra que la hora de de espera de la orden se modificó en el mercado en un bar específico, es decir, a las 18:55.

Aquí debemos hacer hincapié una vez más que la ventana del símbolo muestra sólo el precio de la historia para el precio más bajo de las dos partes de la cotización, a saber, se refleja la historia de **bid**. La historia de **ask** no se muestra. Esta es la razón por la que usted puede pensar que la orden de espera que se modificó a una orden a mercado es un error. Sin embargo, en este caso no hay error aquí. En ese momento, cuando el precio de oferta es igual a 1.3013, el precio de ask a 1,3013 + 2 = 1,3015 (el dos es el spread de 2 puntos). Por lo tanto, el precio de mercado tocó el requerimiento de ejecución de la orden y se produjo la conversión automática de la orden en espera de ser ejecutada a una orden efectiva de compra en mercado. La orden fue modificada por el lado del servidor. Inmediatamente después de que el servidor pase la información acerca de esto, el Terminal de Usuario que, a su vez, muestra la información gráficamente en la ventana del símbolo y en la ventana del terminal (como un texto).

Son similares observaciones relativas a la modificación de la orden BuyLimit 4210411. A pesar de que la gráfica que muestra que el precio toca o está por debajo del precio de espera solicitado para BuyLimit a 16:37-16:39 y 16:41 a (Fig. 72), la orden de mercado no se abre. En este caso, las razones de ello es la misma: el precio de mercado **ask** no toca el precio solicitado en la orden. Sin embargo, se tocó ese nivel en el siguiente bar, a las 16:42. Este evento dio lugar a la modificación de la orden en espera de ser ejecutada a una orden a mercado y así la orden BuyLimit en la ventana de símbolo fue remplazada por una orden Buy (comprar) y una nueva orden de mercado aparece entonces en la ventana del terminal.



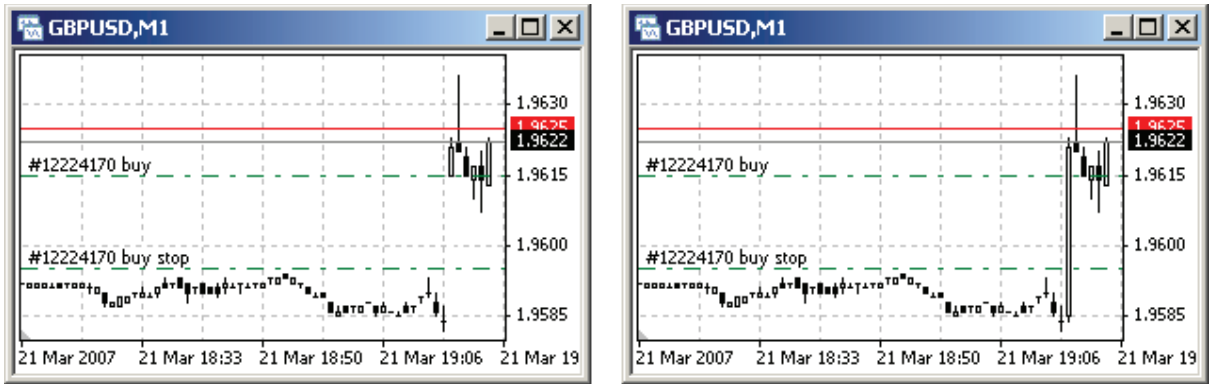
Fig. 72. Modificación de órdenes en espera de ser ejecutadas a órdenes a mercado.

En el ejemplo anterior, todas los órdenes, fueron colocadas con cero órdenes stop (es decir, sin ordenes stop). Sin embargo, la disponibilidad del contenido (no-cero) del valor de la orden de stop no influirá en modo alguno en la modificación de órdenes a la espera a órdenes a mercado, ya que estas órdenes solo pueden ser modificadas si el precio correspondiente en los dos sentidos (two-way) de la cotización toca o cruza el requerimiento del precio de la orden pendiente de ser ejecutada.



La orden a la espera se modifica a una orden a mercado independientemente de las órdenes stop asociadas.

La orden a la espera puede ser abierta (modificada a una de mercado) a un precio que no coincide con el precio solicitado de apertura de la orden pendiente de ser ejecutada. Esto puede suceder en un cambio rápido de los precios de mercado, es decir, en las condiciones que cuando el precio conocido antes de la apertura de la orden no ha llegado aún al precio solicitado pero el siguiente precio (al cual la orden se abre) no coincide con el precio solicitado de apertura, sino que está más allá de el (Fig. 75).



a) precio gapped entre dos bares b) precio gapped dentro de la formacion de un bar.
Fig. 75. La orden pendiente de ser ejecutada es modificada en una a mercado en un gap (hueco).

En la Fig. 75 bis, podemos ver una posible variación de la apertura de una orden en espera de ser ejecutada BuyStop (que muestra dos posiciones de la orden, antes y después de la apertura; en la realidad, se puede ver o bien la orden BuyStop o bien la orden Bay (Comprar), pero no ambas). El último precio conocido antes de que el precio saltara hasta 1,9584 había sido a las 19:15, fueron publicadas algunas noticias, lo que se tradujo en que el símbolo de precios cambió y dió un salto. El primer precio después de conocida la noticia resultó en la liberación de 1,9615. Normalmente, los precios saltan hacia arriba o hacia abajo como resultado de una noticia importante. En tales casos, el corredor no puede abrir su pedido en el precio solicitado, porque no hay precios correspondientes en el mercado en este momento. En este caso, la orden en espera de ser ejecutada BuyLimit se coloca en el precio solicitado de 1.9590, pero la orden se abre (se modifica a orden de mercado) a un precio de 1,9615 como consecuencia del hecho de que no haya habido ningúna cotización de precio dentro del rango de 1.9584 a 1,9615.

Como resultado de los eventos considerados, el precio de apertura de compra en el mercado se hizo con 25 puntos peor que los precios colocados en la orden en espera de ser ejecutada BuyStop. Una situación similar (que reciben menos beneficios de lo esperado en la orden) puede tener lugar para el orden SellStop, si el precio salta hacia abajo. Sin embargo, si la espera para BuyLimit o SellLimit entra dentro del salto de precios (dentro del gap), la correspondiente orden de mercado se pueden abrir a un precio que es mejor para el comerciante que su precio solicitado.

También debe tenerse en cuenta que un gap de precios (la diferencia entre dos cotizaciones más próximas que tienen más de un punto de diferencia) se produce con bastante frecuencia y pueden surgir en cualquier momento. Si el gap de precios tiene lugar entre las barras, es decir, a un precio muy diferente de los llegados en el primer tick de un nuevo bar, se puede el precio del gap en el gráfico de precios (Fig. 75 bis). Sin embargo, si la diferencia de precios ocurre dentro de un bar, no se puede detectar esta diferencia de forma visual (Fig. 75b). En este caso, la diferencia está oculta dentro de la barra (de la vela). No obstante, no se puede hacer un juicio sobre la historia de la cotización dentro de un bar sólo por su apariencia o por alguna característica disponible de un programa. Sin embargo, se puede detectar el gap mediante un programa de aplicación que calcule la diferencia entre los precios de las cotizaciones entrantes.

Modificación de órdenes de mercado

La Plataforma de Operaciones MetaTrader 4 le permite crear las solicitudes de comercio para modificar los niveles de precios de mercado y las órdenes en espera de ser ejecutada.

Para modificar órdenes de cualquier tipo, incluidas las órdenes de mercado, se debería usar la función **OrderModify ()**.



La Modificación de una orden de mercado implica solo el cambio de valores que solicitó de las órdenes stop. No puede cambiar los precios de apertura de las ordene en mercado.

No se puede cambiar el precio de apertura de ordenres en mercado, ya que dicha orden de apertura es un hecho. Por lo tanto, no hay ningún método de programación para hacer esto. La única cosa que puedes hacer con una orden en mercado es cerrarla. Una orden en mercado puede cerrarse como resultado de la ejecución

de una petición de comercio formada por un comerciante o por un programa, o si el precio de mercado alcanza el precio solicitado de una de las órdenes stop.



Las Órdenes StopLoss y TakeProfit no pueden situarse más próxima del precio de mercado que a la distancia mínima establecida por el broker.
La orden de ejecución de su StopLoss o TakeProfit no puede modificarse si el precio de dichas ordenes está dentro de los rangos de la distancia del precio de congelación del mercado.

Tengase en cuenta que la posición de las órdenes de stop de una orden de mercado está limitada en relación con el precio actual del mercado y no con el precio de la orden apertura (ver requisitos y limitaciones en la toma de Órdenes). Esto significa que la modificación de la orden puede dar lugar a que la orden de stop esté colocada por encima o por debajo del precio de apertura del mercado.

Vamos a considerar un ejemplo. Una orden de mercado se abrió antes, su orden de stop se hizo al precio de mercado más cercano (Fig. 69). Después de eso, el precio de mercado ha cambiado (se incrementa 1 punto). En el momento mostrado en la Fig. 76, se hizo posible cambiar el valor de TakeProfit. Una orden de venta es cerrada, al último precio conocido ask. La distancia entre ask= 1.2990 y el anterior valor TakeProfit= 1.2984 es de 6 puntos, es decir, superior a la distancia mínima permitida. El trader (o programa) formó una solicitud de comercio para cambiar el valor de TakeProfit, a saber, aumentar este valor 1 punto. Esto dió lugar a un comercio por el cambio de la posición de la orden de stop a una orden de mercado (el anterior valor de TakeProfit = 1.2984, el nuevo valor = 1,2985).



Fig. 69. Mercado para vender con los Stops al ámbito más próximo al precio de mercado.

Si la solicitud contenida en la instrucción de modificar la orden para vender a fin de que el valor de cualquier orden de stop sean cerradas al precio más cercano de mercado ask que la distancia mínima, la petición de este comercio sería rechazada por el Terminal de Usuario y el comercio no se haría.



Fig. 76. Orden de modificar las órdenes de stop para aproximarse al precio de mercado.

La regla de modificación de las órdenes a mercado, limita la aproximación al precio actual de la orden de stop, pero no limita la distancia de la orden de stop al precio. Esta es la razón por que las ordenes de stop se pueden colocar a cualquier distancia del precio actual, siempre y cuando esta distancia sea mayor que el límite de la distancia (si en el momento de la modificaron de la orden, el valor de la orden de stop esta fuera de la banda de congelación). En la Fig. 77, podemos ver la misma orden después de una modificación: en este caso, las órdenes de stop están bien fuera del alcance de la limitación de distancia mínima.



Fig. 77. Una orden modificada, la orden de stop que se colocan más allá de la distancia mínima.

Modificación de órdenes pendientes de ser ejecutadas

Para modificar cualquier tipo de orden, incluidas las órdenes pendientes de ser ejecutadas, utilizamos la función **OrderModify()**.

La modificación de una orden pendiente de ser ejecutada implica la posibilidad de cambiar los valores definidos del precio de apertura de la orden en espera de ser ejecutada y sus órdenes stop.

La limitación en relación con la posición de la orden pendiente de ser ejecutada para ser modificada se calcula sobre la base el precio correcto de mercado para la modificación de la orden en espera a una orden de mercado.

La limitación en relación con la posición de las órdenes de stop de una orden pendiente de ser ejecutada se calcula sobre la base del precio de apertura solicitado de la orden pendiente de ser ejecutada y no tiene ninguna relación con los precios de mercado.

Las órdenes en espera de ser ejecutadas BuyLimit y BuyStop no pueden situarse más próximas del precio de mercado **ask** que la distancia mínima StopLevel.

Las órdenes en espera de ser ejecutadas SellLimit y SellStop no pueden situarse más cerca del precio de mercado **bid** que a la distancia mínima StopLevel.

Las ordenes en espera StopLoss / TakeProfit no pueden situarse más cerca que el precio de la solicitud de la orden de apertura a la distancia mínima StopLevel.

Las órdenes en espera de ser ejecutadas BuyLimit y BuyStop no pueden modificarse, si el rango del requerimiento de precios de la orden de apertuara está dentro de la distancia de congelación desde el precio de mercado ask.

Las órdenes en espera de ser ejecutadas SellLimit y SellStop no pueden modificarse, si el rango del requerimiento de los precios de está dentro de la distancia de congelación desde el precio de mercado de bid.

> La colocación StopLoss y TakeProfit de órdenes pendientes de ser ejecutadas no están limitadas por la distancia de congelación (FreezeLevel).

Tengase en cuenta que el precio solicitado de una orden en espera de ser ejecutada está en relación con el precio de mercado, mientras que las órdenes de stop están limitadas por el precio solicitado de apertura de la orden en espera de ser ejecutada (ver requisitos y limitaciones en la toma de Órdenes).

Por ejemplo, la orden en espera de ser ejecutada BuyLimit se coloca con los siguientes parámetros: precio solicitado = 1.2969, StopLoss = 1.2964, TakeProfit = 1,2974. El valor actual del precio de mercado (aplicado a la modificación de la orden pendiente de ser ejecutada en una a mercado) ask = 1,2983. Por lo tanto, la orden se coloca a una distancia de 14 puntos (1.2983-1.2969) del precio de mercado, que supera con creces la distancia mínima permitida. Las órdenes en stop estan a una distancia de 5 puntos de la solicitud de precios, la cual no excede la distancia mínima, por lo que es permisible.



Fig. 78. Orden en espera de ser ejecutada BuyLimit asociada con orden de stop más cercana a la orden.

Si el comerciante tiene que cambiar el precio de orden BuyLimit, entonces, sea cual sea la dirección en la que se moviera sería necesario, al mismo tiempo, cambiar tambien la posición de la correspondiente orden de stop (o suprimirla, es decir, establecer valor cero para ella). De lo contrario, la distancia entre la orden y su orden de stop podría llegar a ser menor que el mínimo permitido. En la (Fig. 79). el comerciante decidió modificar el orden de modo que se mantuviera la distancia entre la orden buylimit y la orden TakeProfit a sus 5 puntos, mientras que el valor de StopLoss se mantuvo como estaba



Fig. 79. Se modificó la orden BuyLimit (el precio solicitado y el nivel TakeProfit se cambian).

Si el comerciante necesita colocar una orden en espera de ser ejecutada BuyLimit lo más cerca posible del precio de mercado (Fig. 80), entonces, en este caso, el valor mínimo permitido del precio solicitado **ask**= 5 puntos 1.2985-0.0005 = 1,2980. En este ejemplo, las órdenes de stop esta colocada fuera de la limitación de distancia mínima.



Fig. 80. Se modificó el orden BuyLimit más cercano al precio de mercado.

El apéndice denominado requisitos y limitaciones en la toma de Comercio contiene un cuadro resumen que especifica el juego de los valores two-way de la cotización que se utilizan para la apertura, cierre o modificación de los órdenes, así como otros valores de referencia que limitan la realización de órdenes.

Apertura y colocación de órdenes pendientes de ser ejecutadas

Las solicitudes de comercio para la apertura y colocación de órdenes en espera de ser ejecutadas se forman utilizando la función **OrderSend ()**.

Función OrderSend ()

```
int OrderSend (string symbol, int cmd, double volume, double price, int slippage, double stoploss,
double takeprofit, string comment=NULL, int magic=0, datetime expiration=0, color arrow_color=CLR_NONE)
```

(Tengase en cuenta que desde ahora en adelante, nos referiremos a la cabecera de la función (de llamada) y no a un ejemplo de cómo utilizar la función de llamada en un programa).

Vamos a examinar en más detalle en que consiste esta función.

OrderSend es el nombre de la función. La función devuelve el número de ticket (el 'ticket' es un número único de una orden) que se asigna a la orden por el servidor de comercio, ó el valor -1, si la solicitud es rechazada por el servidor o por el Terminal de Usuario. Con el fin de obtener información sobre los motivos de rechazo de la solicitud, se debe usar la función **GetLastError()** (mas abajo examinaremos alguno de los errores más comunes).

symbol es el nombre del valor o título negociado. Cada símbolo se corresponde con el valor de una variable string. Por ejemplo, para el par de monedas euro / dólar, este valor es "EURUSD". Si la solicitud se hace a un símbolo determinado, entonces este parámetro se debe especificar explícitamente: "EURUSD", "EURGBP", etc. Sin embargo, si se va a utilizar el Asesor Experto en la ventana de cualquier símbolo, se debe utilizar el símbolo de la función estándar **Simbol()**. Esta función devuelve una cadena de caracteres de un valor que se corresponde con el nombre del símbolo de la ventana en la que AE o el script están siendo ejecutados.

cmd es el tipo de operación. Es el tipo de operación que puede ser especificada como una predefinida constante o su valor, y en concordancia con el tipo de *trade.*

volume es la cantidad de lotes. Para órdenes de mercado se debe siempre verificar la cuenta para comprobar la suficiencia de la misma. Para las órdenes pendientes de ser ejecutadas no está limitada la cantidad de lotes.

price es el precio de apertura. Se especifica en función de las necesidades y limitaciones aceptadas para hacer las operaciones (véase el Características de las órdenes y normas para presentarlas). Si el precio solicitado para la apertura de la orden a mercado no se ha encontrado en el precio hilo o si está desfasado, la solicitud se rechaza. Sin embargo, si el precio es anticuado, pero en la actualidad el precio hilo y su variación con respecto al precio actual oscila en el valor de deslizamiento, esta solicitud de comercio será aceptada por el Terminal de Usuario y enviada al servidor de comercio.

slippage (deslizamiento) es la desviación en puntos máxima permitida del precio de apertura de la orden requerida de un precio de mercado para las órdenes de mercado. O en otras palabras, el slippage se puede definir como la diferencia entre el precio aprobado por el usuario y el precio al cual la orden es realmente ejecutada. Este parámetro no es procesado para la colocación de órdenes pendientes de ser ejecutadas.

stoploss es el requerimiento del precio mas cercano que determina la pérdida máxima permitida para una contratación determinada. Se define de acuerdo con los requisitos y limitaciones aceptadas para ejecutar las operaciones (véase el Características y normas para presentar órdenes de Comercio, Requisitos y limitaciones en la presentación de las órdenes de comercio).

takeprofit es el requerimiento del precio mas cercano que determina el máximo beneficio para una contratación determinada. Se define de acuerdo con los requisitos y limitaciones aceptadas para ejecutar las operacaciones (véase Características y normas para presentar órdenes de Comercio, Requisitos y limitaciones en la presentación de Órdenes).

comment es el comentario de texto de la orden. La última parte del comentario puede ser modificado por el servidor de comercio.

magic es el número mágico de la orden. Se puede utilizar como identificador de la orden definida por el usuario. En algunos casos, es la única información que le ayuda a averiguar que una orden pertenece a alguno de los programas abiertos. El parámetro está configurado por el usuario; su valor puede ser el mismo o distinto del valor de este parámetro de otras órdenes.

expiration es la fecha en que expira la orden. Tan pronto como llegue este día, la orden en espera de ser ejecutada se cerrará automáticamente en el servidor. En algunos servidores comerciales, puede haber una prohibición para establecer la fecha de vencimiento para las órdenes en espera de ser ejecutada. En este caso, si se tratan de establecer un valor no-cero del parámetro, la solicitud será rechazada.

arrow_color es el color de la flecha que marca la apertura en el gráfico. Si este parámetro está ausente o si su valor es CLR_NONE, la flecha de la apertura no se muestra en gráfico alguno.

En algunos servidores comerciales, puede haber un límite establecido para el importe total de órdenes abiertas y pendientes. Si se supera este límite, cualquier petición comercial que implica la apertura de una orden de mercado o puesta en espera de un pedido será rechazado por el servidor de comercio.

Las órdenes de apertura de mercado

La función **OrderSend ()**, puede parecer al principio demasiado intrincada. Sin embargo, los parámetros considerados son bastante sencillos y útiles, y pueden ser utilizado con éxito en el trading. Con el fin de ver esto por nosotros mismos, vamos a considerar una variante simple de cómo usar esta la función de comercio **OrderSend ()** para realizar una orden de apertura a mercado.

En primer lugar, es necesario tener en cuenta que la función OrderSend () tiene parámetros predefinidos (véase Función de llamada y Descripcion de la función del operador «return»). Esto significa que esta función puede usarse en modo simplificado utilizando un mínimo juego de parámetros. Estos parámetros son los siguientes:

symbol es un parámetro necesario, porque necesitamos saber dónde abrir la orden. Para dejar que nuestro script tenga la posibilidad de abrir una orden en cualquier ventana de símbolo, se sustituye este parámetro con la función estándar **Symbol()**.

cmd Si por ejemplo vamos a abrir una orden de Compra especificamos el parámetro OP_BUY;

volume Podemos especificar cualquier valor permitido por las reglas. Un ejemplo, sería abrir una pequeña orden con 0,1 lotes.

precio El precio para abrir la orden de compra es el precio ask.

slippage Por lo general se especifica de 0-3 puntos. Vamos a especificar 2;

stoploss Esta orden puede ser colocada a una distancia que no se aproxime al precio del mercado actual a menos que la distancia mínima permitida, normalmente 5 puntos (ver requisitos y limitaciones en la ejecución de trades); Vamos a colocar esta orden a una distancia de 15 puntos desde el precio de cierre, en este caso el cierre sería una operación de venta y por tanto la venta iría "contra" el precio bid. Entonces: Bid – 15*Point (recordar que Point es tamaño de 1 punto del actual titulo o valor de la moneda cotizada, por eso 15 se debe multiplicar por Point)

takeprofit Vamos a colocar esta orden a una distancia de 15 puntos desde el precio de cierre, a saber: Bid + 15*Point

A continuación se muestra el más simple script, simpleopen.mq4, que se destina para la apertura de una orden de compra:

```
//+------------------------------------------------------------------+
// simpleopen.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                          // Special function start()
  {                                  // Opening BUY
   OrderSend(Symbol(),OP_BUY,0.1,Ask,3,Bid-15*Point,Bid+15*Point);
   return;                           // Exit start()
  }
//+------------------------------------------------------------------+
```

Esta orden se podría leer así: "**Enviar Orden** al **símbolo** de la ventana actual consistente en: Un **Tipo de Operación** de Compra con un **volumen** de 0,1 lotes contra el **precio** Ask, con un **deslizamiento** máximo permitido de 3 puntos. Colocamos el **Stop Loss** a una distancia de 15 puntos del precio Bid y el **TakeProfit** tambien a 15 puntos del precio Bid.

O visto de otra manera…

Envia Orden con las siguientes condiciones:

Símbolo	ventana actual
Tipo de Operación	Compra
Volumen	0,1 lotes
Precio	Ask
Deslizamiento	3 puntos
Stop Loss	15 puntos del precio Bid
TakeProfit	15 puntos del precio Bid

Si se lanza este script para su ejecución, trabajará en la mayoría de los casos. El script se compone de una función especial que contiene la orden de apertura, la función **OrderSend ()** y el operador de «return». Vamos a describir el algoritmo de ejecución para las líneas de programa y eventos relacionados con ello.

1. El usuario puede adjuntar el script a la ventana del símbolo arrastrando el nombre del script en el "Explorador" del Terminal de Usuario con el botón del ratón a la ventana de símbolo, para que se pueda abrir una orden de compra de mercado de 0,1 lotes con ordenes de stop que se situan a una distancia de 15 puntos del precio de mercado.

2. En el momento de conectar el script a la ventana de símbolo, el Terminal de Usuario pasa el control a la función especial start () para su lanzamiento (en este caso debemos recordar brevemente que la función start () de un script se lanza en el momento de asignar el script a la ventana de símbolo, mientras que el start () de un AE se pone en marcha en el momento en que llega el primer tick dentro del símbolo).

3. En el marco de la ejecución de la función especial start (), el control se pasa a la línea que contiene la función que solicita la orden de apertura:

```
OrderSend(Symbol(),OP_BUY,0.1,Ask,3,Bid-15*Point,Bid+15*Point);
```

Antes de la ejecución de esta función, el programa calcula los valores de todos los parámetros formales:

3,1. Vincula al scrip la ventana de Eur/USd. En este caso, el símbolo de función estándar Symbol() devolverá la cadena de valor EURUSD.

3,2. Tenemos ask = 1.2852 y bid = 1.2850 en el momento de llamar a esta función.

3,3. El valor de StopLoss, en este caso, serán los siguientes: 1.2850 - 15*0.0001 = 1.2835, y TakeProfit = 1,2865.

4.0 Ejecución de la función OrderSend ():

4,1. La función formó la solicitud de un comercio para la apertura de una orden y pasa esta solicitud al Terminal de Usuario.

4,2. La función pasa el control al Terminal de Usuario al mismo tiempo que pasó de la solicitud, y la ejecución del programa se detiene.

4,3. El Terminal de Usuario comprueba la solicitud de comercio que ha recibido. Si no detecta ningún parámetro incorrecto entonce manda la solicitud al servidor.

4,4. El servidor ha recibido la solicitud, la comprueba y no detecta parámetros incorrectos entonces decide ejecutar la solicitud.

4,5. El servidor ejecuta la petición de realizar una transacción en su base de datos y envia la información sobre la ejecución de la petición al Terminal de Usuario.

4,6. El Terminal de Usuario recibe la información acerca de que la última petición del comercio que ha sido ejecuta y muestra en este caso la ventana del terminal y la ventana del símbolo, y devuelve el control al programa.

4,7. Una vez recibido el control, el programa continua trabajando desde el mismo pundo donde previamente el control había sido pasado al Terminal de Usuario (y al cual habia sido devuelto mas tarde).



Observese que no se realizaron acciones en el programa a partir el paso 4,2 hasta el apartado 4,7. El programa se encontraba en la modalidad de espera a la respuesta del servidor.

5. Ya dentro del programa, se pasa el control al siguiente operador, el operador «return».

6. La ejecución del operador «return» se traduce en salida de la función start () y, por consiguiente, la finalización de la ejecución del programa (cabe recordar que los scripts completan su trabajo después de que se han ejecutado). El control se devuelve al Terminal de Usuario.

De este modo, el scrip ha cumplido con su finalidad: La apertura de una orden de compra con los parámetros preestablecidos. Si es necesario realizar una pequeña operación una sola vez, como en este caso, el uso de un scrip es muy oportuno. De acuerdo al paso 4,6, el operador puede ver la orden en la pantalla.



Fig. 81. orden colocada por el script simpleopen.mq4.

Los acontecimientos no siempre son ordenados como se muestra arriba. Es posible que la solicitud sea rechazada por el Terminal de Usuario o por el servidor. Vamos a tratar de hacer algunos experimentos, por ejemplo, cambiar el nombre del símbolo: especificar explícitamente "GBPUSD" (esto es muy viable). Vamos a tener un programa con un ámbito de uso limitado:

```
int start()                          // Special function start
  {                                 // Opening BUY
   OrderSend("GBPUSD",OP_BUY,0.1,Ask,3,Bid-15*Point,Bid+15*Point);
   return;                          // Exit start()
  }
```

Vamos a iniciar la script en la misma ventana de símbolo de EUR/USD. El script tenía la intención de abrir una orden en la ventana de GBP/USD, sin embargo, después de haber sido asociado a la ventana de EUR/USD, no se puede abrir ninguna orden en el símbolo GBP/USD.

Una desventaja de estos programas es su limitación funcional. En este caso, una vez que tenga el script adjunto a la ventana de símbolo, el usuario está esperando la apertura de la orden. Sin embargo, la orden no se abre. El usuario no sabe la razón de por que esto es así: o bien es causado por un error algorítmico en el código de programa, o la solicitud esta "perdida" por el camino al servidor, o la solicitud ha sido rechazada por el Terminal de Usuario hace mucho tiempo (el pensamiento de usuario se encuentra a la espera), o hay otra razón.

Con el fin de proporcionar al usuario (y que también es muy importante, al programa) la información sobre los eventos relacionados con la ejecución de la solicitud, es necesario procesar los errores.

Error al procesar

Una muy importante propiedad del Terminal de Usuario es que, si se produce un error durante la ejecución de una solicitud, el Terminal de Usuario no detiene la ejecución del programa. Los errores suelen ser causadas por la imperfección del algoritmo utilizado en la solicitud. En algunos casos, los errores son causados por factores externos (en relación con el programa). Las causas internas de los errores es por alguna violación de los requisitos de MQL4 o de las reglas de trading, por ejemplo, utilizando precios no válidos. Las causas externas son las que no están relacionadas con el programa de aplicación, como por ejemplo, la interrupción de la conexión.

Si se produce un error en la ejecución de un programa, el programa continuará ejecutándose, mientras que el Terminal de Usuario generará un código de error que está a disposición del programa a través de la función **GetLastError ()**.

Función GetLastError ()

```
int GetLastError()
```

La función devuelve el código del error que ha ocurrido recientemente, entonces el valor de la variable especial Last_Error que almacena el código del último error no se pone a cero. Posteriormente, cuando no haya error, GetLastError () devolverá 0.

En adelante, vamos a identificar todos los errores que existen con este código. Varios errores pueden ocurrir durante la ejecución de un programa; la función GetLastError () nos permite obtener el valor del código de solo uno de ellos, el del último error. Esta es la razón por la que en el momento en que necesitemos esta información se aconseja utilizar la función GetLastError () inmediatamente después de la línea del programa en la que el error puede ocurrir.

Error 130. Órdenes Stops no válidas

El último script considerado no analiza los errores, por lo que el usuario se mantine ignorante acerca de los resultados producidos por la ejecución de la orden de apertura de la función. Con la simple variante de utilizar la función GetLastError (), el programa puede analizar el error y justo informar al usuario sobre de ello. Si se lanza el script confined.mq4 para su ejecución en la ventana de EUR/USD, se producirá un error.

```
//+------------------------------------------------------------------+
// confined.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                                 // Special function start
  {                                        // Opening BUY
   OrderSend("GBPUSD",OP_BUY,0.1,Ask,3,Bid-15*Point,Bid+15*Point);
   Alert (GetLastError());                       // Error message
   return;                                  // Exit start()
  }
//+------------------------------------------------------------------+
```

Hemos añadido una sola, pero muy informativa línea en este script:

```
   Alert (GetLastError());                    // Error message
```

Función GetLastError () devuelve el código del último error, mientras que Alerta () se utiliza para mostrar este valor en la pantalla. Después de script confined.mq4 ha sido adjunto a la ventana de símbolo EUR/USD, el script se ejecutará, lo que dará lugar a que el usuario veá el siguiente mensaje:

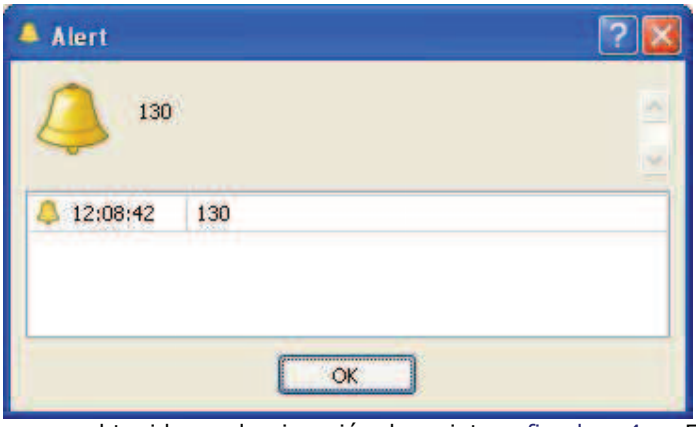


Fig. 82. Código de error obtenidos en la ejecución de script confined.mq4 en EUR/USD ventana.

Se puede buscar en los Apéndices códigos de error los errores que pueden ocurrir en la ejecución de un programa. En este caso, se produjo el error 130 (Órdenes de stops no validas). Esto significa que los valores formales de los parámetros utilizados en la función OrderSend () no se ajusten a las limitaciones especificadas en las necesidades y limitaciones en la toma de Órdenes. Tras una vista más cercana, podemos ver la razón por la que se produjo el error: los valores actuales de los precios de mercado de Bid y Ask se toman de la ventana de símbolo en la que se ha asociado el script, es decir, de la ventana de EUR/USD. Sin embargo, estos valores son utilizados para formar una solicitud de comercio de GBP/USD. Como resultado de ello, en el precio actual de GBP/USD, Ask = 1.9655, el valor de TakeProfit para la reciénte orden abierta de mercado resulta ser igual a (para EUR/USD Bid = 1,2930) 1,2930 +15 * 0.0001 = 1.2945, que es considerablemente inferior al valor mínimo permitido, es decir, no es válido.

En este caso se ha producido un error algoritmico. Con el fin de corregirlo, se debe utilizar los valores correctos de los precios del simbolo. Se pueden obtener estos valores utilizando la función **MarketInfo()**. El Script improved.mq4 abre órdenes a mercado de GBP/USD y puede ser lanzado en cualquier ventana de símbolo:

```
//+------------------------------------------------------------------+
// improved.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                          // Special function start
  {
   double bid   =MarketInfo("GBPUSD",MODE_BID); // Request for the value of Bid
   double ask   =MarketInfo("GBPUSD",MODE_ASK); // Request for the value of Ask
   double point =MarketInfo("GBPUSD",MODE_POINT);//Request for Point
   // Opening BUY
   OrderSend("GBPUSD",OP_BUY,0.1,ask,3,bid-15*Point,bid+15*Point);
   Alert (GetLastError());             // Error message
   return;                             // Exit start()
  }
//+------------------------------------------------------------------+
```

El error anterior no ocurre en la ejecución de este script, por lo que su ejecución tendrá como resultado que se muestre el mensaje correspondiente: 0 (cero). Esto significa que la función GetLastError () devolvió el valor 0, es decir, no se han detectado errores en la ejecución de la solicitud por parte del Terminal de Usuario.

Vamos a considerar también algunos otros errores comunes. Para ello, vamos a volver a la idea de abrir una orden utilizando un script en la misma ventana en la que este se vincula.

Error 129. Precio no válido

En algunos casos ocurre un error simple, un valor incorrecto en la **cotización de los dos sentidos** (two-way quote) especificada en el precio de apertura. Las órdenes de Compra de Mercado se sabe (ver requisitos y limitaciones en la toma de Órdenes), que deben ser abiertas en los precios de Ask. A continuación se muestra lo que ocurre si, por error, se especificará el precio de Bid en el script de mistaken.mq4:

```
//+------------------------------------------------------------------+
// mistaken.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                          // Special function start
  {                                  // Opening BUY
   OrderSend(Symbol(),OP_BUY,0.1,Bid,3,Bid-15*Point,Bid+15*Point);
   Alert (GetLastError());           // Error message
   return;                           // Exit start()
  }
//+------------------------------------------------------------------+
```

Antes de enviar la solicitud al servidor, el Terminal de Usuario analiza si el requerimiento de los valores de precio y las órdenes de stop cumplen con los valores permitidos. Durante este control, se detectó que la solicitud del precio de la orden de apertura no era válida, por lo que el Terminal de Usuario no enviará la solicitud al servidor para su ejecución, y la función GetLastError () devolverá el valor de 129 (véase códigos de error). La ejecución del script dará como resultado la aparición del correspondiente mensaje de error:

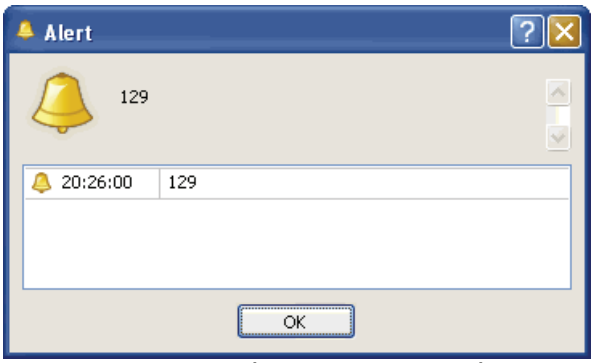


Fig. 83. Error 129 (precios no válidos) en la ejecución de mistaken.mq4.

Error 134. No hay suficiente dinero para hacer un comercio

Un resultado similar (error 134) se obtendrá, si no hay suficiente dinero en la cuenta donde se abre la orden. Se puede conocer acerca de la cantidad de dinero que se necesita para abrir una compra de 1 lote de cualquier símbolo utilizando la función **MarketInfo** ("nombre_del_simbolo", MODE_MARGINREQUIRED).



El tamaño estándar de un lote puede variar para un mismo símbolo para distintos dealing centers.

La cantidad necesaria de activos disponibles para la apertura de una orden de un lote es inversamente proporcional a la cantidad de apalancamiento prevista. Al mismo tiempo, el coste de 1 punto en el depósito de divisas para un símbolo no esta relacionada con el apalancamiento.

Cuadro 3. Combinaciones posibles de 1-lote costo y 1 punto costo (depósito en moneda dólar de los EE.UU.).

	Dealing Center 1			**Dealing Center 2**			**Dealing Center 3**		
	Comprar	Vender	1pt	Comprar	Vender	1pt	Comprar	Vender	1pt
EUR / USD	1296,40	1296,20	10,00	1296,50	1296,20	10,00	1000,00	1000,00	10,00
GBP / USD	1966,20	1966,00	10,00	1376,48	1376,20	7,50	1000,00	1000,00	10,00
AUD / USD	784,40	784,20	10,00	1569,20	1568,40	20,00	1000,00	1000,00	10,00
USD / JPY	1000,00	1000,00	8,29	1000,00	1000,00	8,29	1000,00	1000,00	8,29
USD / CHF	1000,00	1000,00	8,02	1000,00	1000,00	8,02	1000,00	1000,00	8,02
EUR / CHF	1296,40	1296,20	8,02	1296,35	1296. 35	8,02	1000,00	1000,00	8,02

Los precios se dan a partir del 16.12.2007.

Vamos a considerar brevemente algunos métodos comunes de cálculo del coste de 1 lote y de 1 punto.

Dealing Center 1 (más común)

Para los símbolos que tienen USD como denominador, el costo de 1 lote es igual al precio actual de los correspondientes en ambos sentidos de la cotización (two-way quote) multiplicado por 1000, mientras que el costo de 1 punto es igual a $ 10.

Para los símbolos que tienen USD como numerador, el costo de 1 lote es igual a 1000,00 $, mientras que el costo de 1 punto es inversamente proporcional a la cotización actual y equivalente a 1 / (Bid). Por ejemplo, para USD / CHF, en Bid = 1.2466, el costo de 1 punto es 1/1.2466 = 8,02.

Para cruzar las tasas, el coste de 1 lote se calcula de la misma manera que el numerador de la moneda, mientras que el costo de 1 punto se calcula de la misma manera que para el denominador moneda. Por ejemplo, para EUR/CHF, el costo de 1 lote es 1296,40 (como para EUR/USD), mientras que el costo de 1 lote es 8,02 (como para USD/CHF).

Dealing Center 2

En algunos centros se ocupan, teniendo en cuenta la misma regla de cálculo de costes, los valores de los costes puede ser distinta para algunos símbolos. Por ejemplo, el costo de 1 lote y el costo de 1 punto pueden ser proporcionales al alza o a la baja. Por ejemplo, este factor puede ser de 0,75 GBP/USD, mientras que es de 2,0 para el AUD/USD. La representación de los valores de costo no da lugar a ningún cambio económico; en esos casos, sólo hay que examinar esta característica especial en el cálculo de los costes de sus órdenes. Se debe también prestar atención al hecho de que el coste de 1 lote para comprar y en la venta de activos a cruzar las tasas son las mismas.

Dealing Center 3

También hay centros (dealing centers) que se ocupan establecer el costo de 1 lote de 1000,00 $ para cualquier símbolo. Al mismo tiempo, el costo de 1 punto sigue siendo proporcional a los precios actuales. Esto implica el establecimiento de un apalancamiento para cada símbolo.



El coste de 1 punto de todos los símbolos que no estan cotizados en relación con el USD siempre cambian proporcionalmente al coste del símbolo especificado reciprocamente.

En general, pueden existir otros principios de construcción del costo de los valores. No es necesario decir que, con anterioridad al inicio del trading real, se debe conocer el método de cálculo específico de cualquier Dealing Center y considerar ese método en la codificación.

Margen Libre (Free Margin)

En la codificación, es muy importante tener en cuenta el principio de la libre formación de activos. Margen libre (de activos) ó Free Margin es la cantidad de dinero que está disponible para la creación órdenes.

Vamos a considerar un ejemplo. Tenemos un activo en el balance de 5000,00 y no tenemos órdenes abiertas en el terminal. Vamos a abrir una orden de Compra de un 1 lote en el dealing center 3. La siguiente norma se afirma en el dealing center 3:



Si para un símbolo se han abierto órdenes de mercado en diferentes direcciones, el mínimo coste integral de las órdenes en una dirección esta liberado para el trading y aumenta la cantidad de activos (esta regla no es aplicable para todos los dealing centers).

La ventana del terminal mostrará la información sobre la orden abierta.Tengase en cuenta que el margen (garantia exigida o coste del activo) es 1000,00, el beneficio en ese momento (debido al spread) es -30,00, por lo que la cantidad de activos disponibles (free margin) es 5000 -1000 -30 = 3970,00:

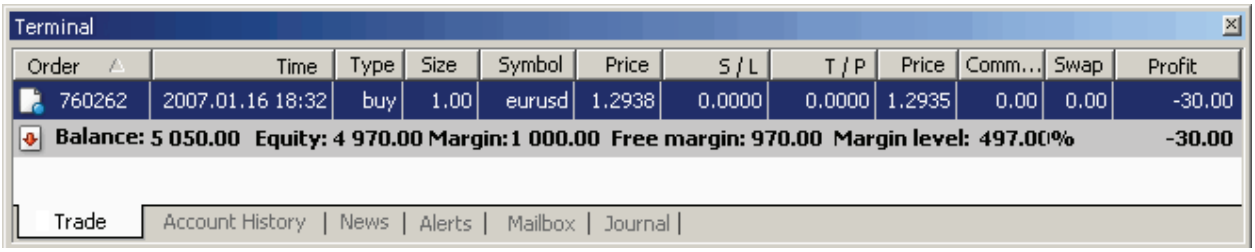


Fig. 84. Orden de Compra en la ventana del terminal

Después de que se abre una orden de venta en el mismo valor el margen se incrementa. El pequeño coste integrado de mercado de órdenes en una dirección (garantia) es de 1000,00, por lo que el margen libre se incrementará en 1000,00. En la Fig.85, se puede ver la situación de las órdenes dirigidas en diferente dirección y con el mismo valor, por lo que la totalidad de la suma de costes de las órdenes queda liberado para el comercio.

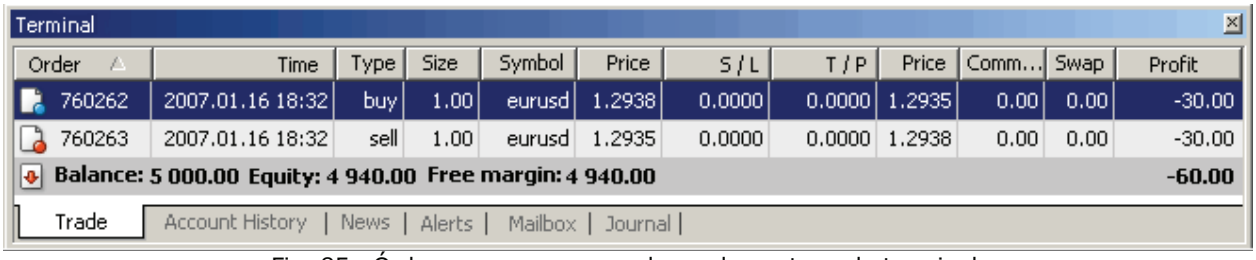


Fig. 85. Órdenes comprar y vender en la ventana de terminal.

Después de que una orden de Venta de más pequeño coste ha sido abierta, el free margin se incrementa también. En este caso, el mínimo coste integrado de la orden de mercado (garantía) en una dirección es de 700,00, por lo que el free margin se incrementará en 700,00, mientras que el margen marca la diferencia entre los costes integrados de las órdenes dirigidas en diferente dirección (Fig. 86).


Terminal

Order	Time	Type	Size	Symbol	Price	S / L	T / P	Price	Comm...	Swap	Profit
760330	2007.01.16 19:40	buy	1.00	eurusd	1.2922	0.0000	0.0000	1.2919	0.00	0.00	-30.00
760331	2007.01.16 19:41	sell	0.70	eurusd	1.2919	0.0000	0.0000	1.2922	0.00	0.00	-21.00
Balance: 5 000.00 Equity: 4 949.00 Margin: 300.00 Free margin: 4 649.00 Margin level: 1649.67%											**-51.00**

Trade | Account History | News | Alerts | Mailbox | Journal

Fig. 86. Órdenes de compra y venta en la ventana de terminal.

Si se abre una orden de 0,1 lote para vender (coste 100.00), el mínimo coste integrado de la orden de mercado en una dirección es de 700,00 + 100,00 = 800,00. Así, el margen (en comparación con la situación en la que solo se abre una orden de Compra) disminuye en 800,00. La comparación de la situación se muestra en la Fig.86, el margen se reduce, mientras que el capital aumenta en 100,00 (ver Fig. 87).


Terminal

Order	Time	Type	Size	Symbol	Price	S / L	T / P	Price	Comm...	Swap	Profit
760330	2007.01.16 19:40	buy	1.00	eurusd	1.2922	0.0000	0.0000	1.2917	0.00	0.00	-50.00
760331	2007.01.16 19:41	sell	0.70	eurusd	1.2919	0.0000	0.0000	1.2920	0.00	0.00	-7.00
760341	2007.01.16 20:02	sell	0.10	eurusd	1.2918	0.0000	0.0000	1.2920	0.00	0.00	-2.00
Balance: 5 000.00 Equity: 4 941.00 Margin: 200.00 Free margin: 4 741.00 Margin level: 2470.50%											**-59.00**

Trade | Account History | News | Alerts | Mailbox | Journal

Fig. 87. Órdenes comprar y vender en la ventana del terminal.

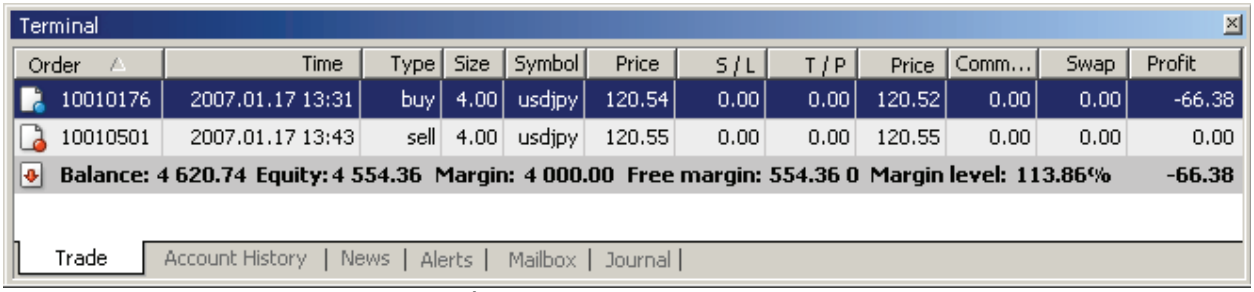
Los Free margin (Márgenes disponibles) se muestran en la Fig.86 y Fig.87 difieren entre sí en más de 100,00, ya que el beneficio integral de órdenes abiertas ha cambiado con el cambio en el precio actual (la diferencia es de 8,00).

Si hacemos manipulaciones similares en otro dealing center, es fácil ver que no se mantiene el free margin de la formación en las órdenes de arriba. Para algunos dealing centers, la siguiente regla es eficaz:



La apertura de órdenes de cualquier mercado no libera capital ni incrementa el free margin. La apertura de órdenes de mercado incrementa el capital por la cantidad que supere el costo integrado de órdenes dirigidas de forma diferente en un símbolo o instrumento (la norma no se aplica en todos los dealing centers).

Por ejemplo, si se ha abierto anteriormente una órden de Compra (Buy) de 4 lotes de USD/JPY en el daling center 2, los importes de la equity (capital o patrimonio) y el free margin no cambiará a la apertura de una orden de venta (Sell) de 4 lotes.


Terminal

Order	Time	Type	Size	Symbol	Price	S / L	T / P	Price	Comm...	Swap	Profit
10010176	2007.01.17 13:31	buy	4.00	usdjpy	120.54	0.00	0.00	120.52	0.00	0.00	-66.38
10010501	2007.01.17 13:43	sell	4.00	usdjpy	120.55	0.00	0.00	120.55	0.00	0.00	0.00
Balance: 4 620.74 Equity: 4 554.36 Margin: 4 000.00 Free margin: 554.36 0 Margin level: 113.86%											**-66.38**

Trade | Account History | News | Alerts | Mailbox | Journal

Fig. 88. La presencia de órdenes dirigidas de forma diferente no libera capital.

Se pueden hacer cálculos para saber si el capital actual es suficiente para la apertura de una orden. Se puede utilizar la función **AccountFreeMarginCheck ()** que devuelve el valor del margen libre que queda después vde la apertura de una orden de mercado de una cierta cantidad de lotes en un determinado símbolo. Si el valor devuelto es igual o superior a 0, hay suficiente dinero en la cuenta. Si es inferior a 0, entonces la orden de este volumen para este símbolo no se puede abrir y el Terminal de Usuario devolverá el error 134.

Con el fin de conocer las condiciones ofrecidas por el centro y calcular la cantidad de margen necesario para la apertura de una orden con un volumen de 1 lote, puede utilizar un script simple como, conditions.mq4:

```
//------------------------------------------------------------------
// conditions.mq4
// The code should be used for educational purpose only.
//------------------------------------------------------------------
int start()                          // Special function start
  {
   Alert(Symbol()," Sell = ",AccountFreeMargin()-AccountFreeMarginCheck(Symbol(),OP_SELL,1));
   Alert(Symbol()," Buy = ",AccountFreeMargin()-AccountFreeMarginCheck(Symbol(),OP_BUY,1));
   return;                           // Exit start()
  }
//------------------------------------------------------------------
```

En este sentido, la expresión de

```
AccountFreeMargin() - AccountFreeMarginCheck(Symbol(),OP_SELL,1)
```

nos permite calcular la diferencia entre el free margin actualmente disponible y el free margin que quedaría después de la apertura de la orden. Esta diferencia representa el coste de garantía para, en este caso, una operación de venta con un volumen de un lote.

Si iniciamos la ejecución del script, cuando no hay órdenes de mercado en el terminal, podemos obtener la cantidad actual requerida de capital (garantía) disponible y suficiente para la apertura de una orden con un volumen de 1 lote de compra o venta como ya hemos dicho:



Fig. 89. Coste de 1 Lote en diferentes símbolos, obtenidos mediante el script conditions.mq4.

Si queremos poner en marcha el script conditions.mq4 para su ejecución en la ventana de símbolos donde existen órdenes abiertas de mercado, podemos obtener otros valores, dependiendo de los métodos de cálculo aceptados en uno u otro dealing center.

Otros errores y Función MarketInfo ()

Hay otras limitaciones relacionadas con la determinación de valores de los parámetros de la función **OrderSend ()**. Este son el máximo y mínimo paso en el incremento del precio de la orden, máximo y mínimo valor del precio de la orden, etc. El uso de la función **MarketInfo ()** le permite obtener información acerca de diversos símbolos que aparecen en la ventana "Observación del Mercado" del Terminal de Usuario.

Función MarketInfo ()

```
double MarketInfo(string symbol, int type)
```

La función devuelve información acerca de los símbolos que figuran en la ventana "Observación del Mercado " del Terminal de Usuario. Diversas partes de información sobre el símbolo actual se almacenan en variables predefinidas.

Parámetros:

symbol: el nombre de un símbolo;

tipo: identificador que determina el tipo de información que será devuelta. Puede ser cualquier valor de los identificadores de solicitud (véase Función MarketInfo Identifier).

Pueden ocurrir algunos errores debidos al servidor. Por ejemplo, en condiciones transitorias de precios, su agente puede aumentar la distancia mínima que limita la colocación de órdenes pendientes de ser ejecutadas y órdenes stops. Después, en un mercado en calma, el corredor puede reducir esta distancia de nuevo. De este modo, los valores de algunos parámetros pueden cambiar en cualquier momento.
Para operar con el programa de una forma estable, con la mínima cantidad de solicitudes rechazadas, se debe actualizar los parámetros del entorno de información utilizados por el programa usando las funciones **MarketInfo ()** y **RefreshRates ()** antes de ejecutar la función **OrderSend ()**.



Ejemplo de un script simple que abre una orden de Compra con un costo del X% de margen libre, con unos valores preestablecidos de órdenes de stop (openbuy.mq4).

```
//+------------------------------------------------------------------+
// openbuy.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+ 1 –
int start()                          // Función Especial start
{
   int Dist_SL =10;                  // Preselección distancia  StopLoss (puntos)
   int Dist_TP =3;                   // Preseleccion distancia Take Profit (puntos)
   double Prots=0.03;                // Percentaje de margen libre
   string Symb=Symbol();             // Simbolo
//+------------------------------------------------------------------+ 2 --
   while(true)                       // Ciclo para la orden de apertura openbuy
   {
      int Min_Dist=MarketInfo(Symb,MODE_STOPLEVEL);// Distancia mínima
      double Min_Lot=MarketInfo(Symb,MODE_MINLOT);// Volumen mínimo
      double Step   =MarketInfo(Symb,MODE_LOTSTEP);// Paso mínimo de cambio de lotes
      double Free   =AccountFreeMargin();          // Margen Libre
      double One_Lot=MarketInfo(Symb,MODE_MARGINREQUIRED);// Coste por 1 lote (Garantía por lote)

       //------------- Calculo del volumen de la operación ------------------------------ 3 –
```

```
      double Lot=MathFloor(Free*Prots/One_Lot/Step)*Step;// Lotes
      if (Lot<Min_Lot)                    // Si el nº de lotes es menor que el permitido
        {
          Alert(" No hay suficiente dinero para ", Min_Lot," lotes");
          break;                          // Salir del ciclo
        }
          //----------------- Cálculo del Stop Loss -------------------- 4 --
      if (Dist_SL<Min_Dist)               // Si la distancia del Stop es menor que la permitida
        {
          Dist_SL=Min_Dist;               // Poner la distancia permitida
          Alert(" Incremento de la distancia del SL = ",Dist_SL," puntos.");
         }
      double SL=Bid - Dist_SL*Point;      // Requerimiento del precio del SL
          //------------------Cálculo del Stop Take Profit ----------------- 5 --
      if (Dist_TP<Min_Dist)               // Si la distancia del Stop es menor que la permitida
        {
          Dist_TP=Min_Dist;               // Poner la distancia permitida
          Alert("Incremento de la distancia del TP = ",Dist_TP," puntos.");
        }
      double TP=Bid + Dist_TP*Point;          // Requerimiento del precio de TP
          //--------------------Solicitud de Compra ------------------------------ 6 --
      Alert("La solicitud fue enviada al servidor. Esperando respuesta...");
      int ticket=OrderSend(Symb, OP_BUY, Lot, Ask, 2, SL, TP);
          //------------------Notificacion ejecución de orden------------------ 7 --
      if (ticket>0)                       // ¡Orden en Mercado!   :)
        {
          Alert ("Nº de Orden de Compra Abierta: ",ticket);
          break;                          // Salir del ciclo
        }
          //----------------------Procesamiento de Errores------------------- 8 --
      int Error=GetLastError();               // ¡Orden Fallida!  :(
      switch (Error)                      // Procesamiento de errores
        {
          case 135:Alert("El precio ha cambiado. Reintentarlo de nuevo…");
              RefreshRates();                 // Actualizar datos del entorno
              continue;                   // Realizar nueva iteración
          case 136:Alert("No hay precio. Esperar a un nuevo tick…");
              while(RefreshRates()==false)  // Mientras no haya un nuevo tick
                    Sleep(1);                 // Ciclo de espera
              continue;                   // Como ya hay nuevo tick realizar nueva iteración
          case 146:Alert("el subsistema de trading está ocupado. reintentarlo..");
              Sleep(500);                   // Solución simple: dormir durante 500 msg.
             RefreshRates();                  // Actualizar datos del entorno y…
              continue;                   // realizar una nueva iteración
         }
        switch(Error)                     // Errores críticos
          {
           case 2 : Alert ("Error común.");
                    break;                        // Salir de 'switch'
           case 5 : Alert ("Terminal de usuario obsoleta.");
                    break;                        // Salir de 'switch'
           case 64: Alert ("The account is blocked.");
                    break;                        // Salir de 'switch'
           case 133:Alert ("Trading forbidden");
                    break;                        // Salir de 'switch'
           default: Alert ("Occurred error ",Error);// Otras Alternativas
          }
      break;                              // Salida del ciclo
    }
//------------------------------------------------------------------------ 9 --
    Alert ("The script has completed its operations -------------------------");
    return;                               // Exit start()
```

```
}
//---------------------------------------------------------------------- 10 --
```

El script se compone de una función especial start() (bloques 1-10). En el bloque de 1-2, se fijan los valores a los que la orden debe ser abierta. El Bloque 2-9 representa el ciclo del operador **While()**, en el que se realizan todos los cálculos necesarios. Este ciclo está incluido en el código para permitir que el programa haga varios intentos para abrir una orden. En el bloque 2-3 se actualizan las variables del entorno. En los bloques 3-4-5-6, se calcula el importe de los lotes y los precios de las órdenes de stop. En el bloque de 7-8-9, se procesan los errores. En el bloque 9-10, se escribe un mensaje informando que el script ha terminado sus operaciones.

Vamos a considerar algunas características especiales del código del programa. Es fácil ver que la solicitud de compra se forma en el bloque 6-7. En el bloque 3-4, se calcula el importe de los lotes. Asimismo, se considera la situación en el que la disposición del margen libre es insuficiente para abrir incluso una orden con una cantidad mínima de lotes. Es por ello que, en el bloque 3-4, después de imprimir el mensaje sobre la insuficiencia de dinero, se sale del ciclo 2-9 utilizando el operador "break". El control se pasa al bloque 9-10, y el script completa sus operaciones. El mensaje en la casilla 9 es innecesario. Se da aquí sólo para ayudar a los usuarios del código a encontrar colas o cabezas en el script, cuando es el final de las operaciones del programa y cuando la pausa es provocada por los retrasos en la red o en el servidor.

Si el margen libre es suficiente para la apertura de la orden, el control pasa al bloque de 4-5 y luego al bloque 5-6. En estos bloques, no hay salida ciclo. Esto significa que, para cualquier distancia mínima fijada por el corredor, habrá stops en los niveles correspondientes. En el bloque de 1-2, 3 los puntos fueron elegidos para TP por diseño. La mayoría de los corredores establecen la distancia mínima en 5 puntos. En el bloque 5-6, el programa descubrirá que el valor preestablecido es inferior al permitido. El programa pondrá un valor de precio tal de la orden stop que no estará en contradicción con la limitación. Entonces el control se pasa al bloque de 6-7 para abrir la orden. En la primera línea de este bloque, se imprime un mensaje. La solicitud de comercio está formada sólamenre en la segunda línea. Una cuestión que se plantea es: ¿Por qué se declara la formación de la solicitud compra antes de que ésta esté realmente formada? Podríamos dar primero la instrucción y luego informar al usuario sobre ella. La respuesta a esta pregunta está estrechamente relacionada con la tecnología de envio de la solicitud al Terminal de Usuario y, a continuación, al servidor (ver Fig. 66). En nuestro caso, la solicitud formada en la función **OrderSend ()** se especifica en la parte derecha del operador de asignación. La solicitud como tal se ha creado y enviado por la función al servidor, mientras que la operación de asignación se llevará a cabo en el operador de asignación después de que el servidor ha dado una respuesta acerca de la "suerte" de la solicitud. Por lo tanto, la única posibilidad de informar al usuario sobre el comienzo de los acontecimientos relacionados con la solicitud es mostrar el mensaje antes de que el operador de asignación, en la parte derecha de la función de comercio sea especificado.

Tarde o temprano, el Terminal de Usuario pasará el control al programa, en el bloque de 6-7 se ejecutará el operador de asignación que dará lugar a que la variable 'ticket' tome un valor, el control se pasara adelante, y si hay un error se análizará los bloques 7-8-9.

Si la orden se abre en el servidor, a la variable «ticket» le será asignado un número que corresponde a la apertura de la orden. Si ocurre esto, significa que el script ha cumplido su tarea y no hay necesidad de que el programa continue las operaciones. En el bloque 7-8, usamos el operador “break” para salir del ciclo while (). El control se pasa al bloque 9-10 (fuera del ciclo), y el programa termina sus operaciones.

Sin embargo, si el intento de abrir una orden falla, el control se pasa al bloque 8-9 para el análisis del error. En este caso se consideran dos tipos de errores: los que aún permiten tener esperanza de tener éxito en la apertura de la orden y aquellos errores cuya aparición significa la finalización inequívoca de la ejecución del programa. La variable 'Error' se le asigna el código del último error, en este caso, es el error que ha sido devuelto por el servidor o el Terminal de Usuario en la ejecución de la función OrderSend ().

En el primer operador 'switch' del bloque de 8-9, se consideran los errores superables. Cada error de este grupo se procesa de manera diferente. Por ejemplo, si el precio ha cambiado (error 135), es suficiente actualizar solamente los parámetros ambientales utilizando la función RefreshRates () y repetir el intento de apertura de una orden. Si se produce el error 136, "No hay precios", no tiene sentido volver a enviar la solicitud al servidor de comercio.En este caso, debemos esperar a un nuevo tick (debido a que no hay precios en el servidor en ese momento) y, sólo después de esto, se intenta abrir de nuevo la orden. Por eso hay un ciclo de espera en el bloque que procesa el error 136. Este ciclo de espera se interrumpe cuando entra un nuevo tick. La salida del operador de switch() usa el operador 'continue' que rompe la actual iteración del ciclo while() y comienza una nueva.

Los errores críticos se procesan de otra manera. Si tal error se produce, el programa informa al usuario sobre ello y se pone fin a las operaciones. Para ello, se utiliza el operador "break" (el último, en el bloque 8-9) que rompe el ciclo de while (), lo que da lugar a la finalización del programa.

Debemos tener en cuenta, que en este ejemplo en particular, el diseño no considera todos los errores existentes. En este caso, no estamos proporcionando al usuario un programa listo para su uso. Es muy importante que el propio programador analice otros errores de forma independiente y decida qué otros errores y de qué manera deben ser tratados en el programa. Al mismo tiempo, algunos errores no deben ser procesados, porque el programa está construido de tal manera que no considera la existencia de algunos de ellos, por ejemplo, en este caso, los errores 129 y 130…

En el ejemplo anterior, hay un pequeño error algorítmico que no puede ser encontrado en la compilación ni tampoco en el Terminal de Usuario, ni en el servidor.



Tome cualquier código de programa con un grano de sal, con desprecio de la autoridad.

Tengase en cuenta el código en el bloque 4-5:

```
//----------------- Cálculo del Stop Loss -------------------- 4 --
    if (Dist_SL < Min_Dist)               // Si la distancia del Stop es menor que la permitida
      {
       Dist_SL=Min_Dist;                  // Poner la distancia permitida
       Alert(" Incremento de la distancia del SL = ",Dist_SL," pt.");
      }
    double SL=Bid - Dist_SL*Point;          // Requerimiento del precio del SL
//------------------------------------------------------------------ 5 --
```

Como resultado de los cálculos en el cuerpo del operador **if ()**, la variable Dist_SL puede tomar un nuevo valor. Supongamos una distancia normal mínima de 5 puntos. Supongamos que en la primera ejecución (en un mercado rápido), este valor se establece en 20 puntos en el servidor. La variable Min_Dist tendrá el valor de 20.

```
    int Min_Dist=MarketInfo(Symb,MODE_STOPLEVEL);// Minimum distance
```

También se supone que el comercio formado por la solicitud ha sido rechazado debido a un error 136 (No hay precios), El programa hará un seguimiento de un nuevo tick en el bloque 8-9. Dentro de este período de tiempo, el valor de la distancia mínima puede cambiar en el servidor, por ejemplo, disminuir a 10 puntos. En el momento en que un nuevo tick llega, el control se pasa al nuevo ciclo, y el nuevo valor de la variable Min_Dist, igual a 10, se calculará. Sin embargo, el valor de la variable Dist_SL sigue siendo la misma e igual a 20 (el bloque 4-5 se codifica de tal manera que el valor de Dist_SL sólo puede aumentar). Con el fin de excluir este error algorítmico, se debe escribir el bloque 4-5 de tal manera que sólo el valor que depende de la situación cambiaría (en este caso, es el valor del SL), mientras que el valor de Dist_SL no cambiará. Podemos escribirlo, por ejemplo, de esta manera:

```
//------------------------------------------------------------------------ 4 --
    double SL = Bid - Dist_SL*Point;        // Requested price of SL
    if (Dist_SL<Min_Dist)                 // If it is less than allowed
      {
       SL = Bid - Min_Dist*Point;           // Requested price of SL
       Alert(" Increased the distance of SL = ",Min_Dist," pt");
      }
//------------------------------------------------------------------------ 5 --
```

Un cambio similar debe hacerse en bloque de 5-6 para el otro stop.

Colocación de órdenes en espera de ser ejecutadas

No hay ninguna diferencia fundamental en la programación entre la colocación de órdenes pendientes de ser ejecutadas y colocación de órdenes de mercado.

Se debería tomar nota del hecho de que ni el Terminal de Usuario ni el servidor verifican si hay suficientes activos como para modificar una orden de espera a una entrada en el mercado. Estas órdenes tampoco están limitados. Se puede colocar una orden pendiente de ser ejecutada para la cantidad que sobrepase muchas veces la cantidad de dinero disponible en la cuenta. Esta orden puede mantenerse duranate mucho tiempo sin problemas, pero cuando el precio de mercado alcanza el nivel del precio de solicitado para la apertura de la orden en espera, el servidor hará un control sobre el precio y si en ese momento hay suficiente dinero en la cuenta para la apertura de esta orden, esta será modificada en una orden de mercado (apertura), y si no, la orden será eliminada.

Función WindowPriceOnDropped ()

En MQL4, tenemos una característica muy importante, y es que se puede determinar programáticamente en la ventana del símbolo las coordenadas de la ubicación en el que un Asesor Experto o un script se han colocado, si alguna de éstas se hubiera vinculado utilizando un ratón. Por ejemplo, podemos obtener el valor de coordenada la conexión de un script vinculado utilizando la función WindowPriceOnDropped ().

```
double WindowPriceOnDropped()
```

La función devuelve el valor del precio en el punto del gráfico en el cual el AE o el script se ha "soltado". El valor será **true** solo si el AE o el script se ha trasladado mediante el ratón ("arrastrar y colocar"). Este valor no está definido para indicadores personales.



Un ejemplo de un simple script que abre una orden **BuyStop** con un costo del X% del margen de la libre, con unos valores preestablecidos de órdenes stops. (openbuystop.mq4).

```
//+------------------------------------------------------------------+
// openbuystop.mq4
// The code should be used for educational purpose only.
//------------------------------------------------------------------- 1 --
int start()                                          // Especial function start
  {
    int Dist_SL =10;                                 // Preselección  distancia en puntos del StoppLoos
    int Dist_TP =3;                                  // Preselección distancia en puntos TakeProfit
    double Prots=0.03;                               // Porcentaje del Margen libre
    string Symb=Symbol();                            // Simbolo seleccionado
    double Win_Price=WindowPriceOnDropped(); // El script es "soltado" aquí a este precio…
    Alert("El precio es puesto por el ratón, precio = ",Win_Price);// … puesto por el ratón
//------------------------------------------------------------------- 2 --
    while(true)                                      // Ciclo que abre un orden
      {
        int Min_Dist=MarketInfo(Symb,MODE_STOPLEVEL);// Distancia mínima exigida por el corredor
        double Min_Lot=MarketInfo(Symb,MODE_MINLOT);// Mínimo volumen exigido por el corredor
        double Free   =AccountFreeMargin();      // Margen libre, Margen disponible
        double One_Lot=MarketInfo(Symb,MODE_MARGINREQUIRED);//Coste por lote (Garantía por
lote)
        double Lot=MathFloor(Free*Prots/One_Lot/Min_Lot)*Min_Lot;// Lotes
    //------------------------------------------------------------------- 3 --
        double Price=Win_Price;                      // Precio puesto por el ratón
        if (NormalizeDouble(Price,Digits)<           // Si el precio es menor que el permitido…
        NormalizeDouble(Ask+Min_Dist*Point,Digits))
           {                                         // … para BuyStop solamente, acutalizar…
             Price=Ask+Min_Dist*Point;           // … a un precio no tan cercano y permitido
             Alert("Cambiada la solicitud de precio a un precio permitido = ",Price);
           }
    //------------------------------------------------------------------- 4 --
        double SL=Price - Dist_SL*Point;         // Precio requerido de StopLoos
        if (Dist_SL < Min_Dist)                      // Si la distancia es menor que la permitida…
          {
             SL=Price - Min_Dist*Point;          // …Actualizar el precio requerido de StopLoss
             Alert(" Incrementear la distancia de Stop Loss a una distancia permitida = ",Min_Dist,"
puntos.");
           }
    //------------------------------------------------------------------- 5 --
        double TP=Price + Dist_TP*Point;         // Precio requerido de Take Profit
        if (Dist_TP < Min_Dist)                      // Si la distancia es menor que la permitida…
           {
             TP=Price + Min_Dist*Point;           // …Actualizar el precio requerido de Take Profit
             Alert(" Incrementar la distancia del Take Profit = ",Min_Dist," pt");
           }
    //------------------------------------------------------------------- 6 --
        Alert("La petición fue enivada al servidor. Esperando respuesta...");
        int ticket=OrderSend(Symb, OP_BUYSTOP, Lot, Price, 0, SL, TP);
    //------------------------------------------------------------------- 7 --
        if (ticket>0)                                // ¡Conseguido!  :)
          {
        Alert ("Colocada orden BuyStop ",ticket);
        break;                                       // Salir del ciclo
           }
    //------------------------------------------------------------------- 8 --
        int Error=GetLastError();                    // Intento fallido  :(
        switch(Error)                                // Errores superables
          {
             case 129:Alert("Precio no válido. Reintentando...");
                                RefreshRates();  // Actualizando datos del entorno
                                continue;        // A la próxima iteración
             case 135:Alert("El precio ha cambiado. Reintentando..");
                                RefreshRates();   // Actualizando datos del entorno
```

```
                                continue;           // A la próxima iteración
                case 146:Alert("El subsistema de Trading está ocupado. Reintentando...");
                                Sleep(500);                  // Solución simple. Esperar 0,5 sgs.
                                RefreshRates();              // Actualizar datos del entorno
                                continue;                    // A la próxima iteración
            }
            switch(Error)                                    // Errores críticos
              {
                case 2 : Alert("Common error.");
                                break;                       // Salir de 'switch'
                case 5 : Alert("Outdated version of the client terminal.");
                                break;                       // Salir de 'switch'
                case 64: Alert("La cuenta está bloqueda.");
                                break;                       //Salir de 'switch'
                case 133:Alert("Trading prohíbido");
                                                  break;                    // Exit 'switch'
                default: Alert("Ha ocurrido el error: ",Error);// Otros errores
              }
            break;                                           // Salir del ciclo de apertura
          }
    //---------------------------------------------------------------------------- 9 --
        Alert ("El scrip ha completado sus operaciones ----------------------------");
        return;                                              // Salir de start()
      }
    //---------------------------------------------------------------------------- 10 --
```

La estructura del script openbuystop.mq4 se construye de la misma manera que el script de openbuy.mq4, por lo que no hay necesidad de describir los detalles. Sólo dirigiremos nuestra atención a las diferencias básicas entre los dos programas.

El nivel de precio al que el script se asocia a la ventana de símbolo, se determina en la línea:

```
double  Win_Price=WindowPriceOnDropped();     // El script es "soltado" aquí a este precio
```

Posteriormente, el valor de esta variable se mantiene sin cambios durante todo el período de operación del programa. Esto es necesario, por si el script falla la apertura de una o más órdenes. Al mismo tiempo, el script calculará cada vez del valor de la la solicitud del precio próximo a la ubicación, (al nivel de precios de la ubicación) donde el usuario adjunta el script.

Es fácil ver que, en el script openbuystop.mq4, no existe comprobación de la suficiencia del margin free (margen libre) para la apertura de una orden, pero hay un control del precio de la orden apertura (bloque 3-4). Si el valor calculado de la variable precio no cumple con los requisitos de puesta en espera de una orden Stop (ver Características de las ordenes y normas para presentar Comercio, requisitos y limitaciones en la toma de Órdenes), este valor será calculado de nuevo.

En el bloque de error de procesamiento, hay algunos pequeños cambios: aunque hay todavía algunos errores que no son considerados se han añadido para ser procesados algunos códigos de error adicionales.

Limitaciones razonables

Relacionados con el uso de las funciones de comercio, debemos prestar atención a algunas limitaciones de carácter más general. Por ejemplo, el error 146 se produce solamente si varios programas que forman solicitudes de comercio trabajan en un mismo símbolo de una ventana. En nuestra opinión, esta práctica es permisible, pero no es aconsejable.

Sería mucho más eficiente crear y utilizar un solo programa comercial que considerará todas las características especiales de comercio. Si usamos sólo un programa de comercio, es sencillamente imposible crear varias solicitudes de comercio simultáneamente. Por otra parte, el algoritmo podría estar mucho mejor organizado de esta manera en el programa: considerar la probabilidad de éxito de las operaciones y volver a asignar el dinero correctamente, de acuerdo con esta probabilidad.

Para la creación de órdenes de comercio, es más eficaz utilizar un Asesor Experto de escala maxima, mientras que un script sería mejor utilizarlo para realizar cálculos que solo se hagan una vez o para mostrar alguna información útil sobre la pantalla. Al mismo tiempo, si el operador no tiene que utilizar un Asesor Experto para el comercio automatizado, el uso de script resulta ser más eficiente que trabajar con órdenes utilizando el panel de control de la Terminal de Usuario.

Cierre y supresión de órdenes

Cierre de órdenes de mercado

Las solicitudes de Comercio de órdenes de cierre de mercado se forman utilizando la función **OrderClose ().**

Función OrderClose ()

```
bool OrderClose (int ticket, double lots, double price, int slippage, color Color=CLR_NONE)
```

Se trata de una función utilizada para cerrar una orden de mercado. La función devuelve TRUE, si la orden de comercio se ha realizado con éxito. Devuelve FALSE, si no se ha cerrado.

Parámetros:

ticket - el número único de la orden que se desea cerrar.

lots - la cantidad de lotes a ser cerrados. Se permite especificar un valor inferior a la cantidad disponible de los lotes que se presentan en la orden. En este caso, si la solicitud se ejecuta con éxito, la orden será cerrada parcialmente.

price – precio de cierre. Este parámetro se fija de acuerdo con los requisitos y limitaciones aceptadas para la realización de operaciones (véase la Orden Características y normas para presentar los órdenes y el apéndice 3). Si no hay precio disponible para el cierre de la orden de mercado en el flujo de precios, o si el precio es anticuado, la solicitud de comercio será rechazada; si el precio es anticuado, pero se encuentra en el precio actual y, al mismo tiempo, la desviación del precio actual oscila en el valor de deslizamiento, la solicitud será aceptada por el Terminal de Usuario y enviada para comercio al servidor.

slippage (deslizamiento) – es la desviación máxima permitida de los precios solicitados para el cierre de la orden al precio de mercado (en puntos).

Color - el color de la flecha de cierre en un gráfico. Si este parámetro no está disponible o su valor es igual a la de CLR_NONE, la flecha no se muestra en el gráfico.

Si el programa contiene información sobre el tipo de orden de cierre, acerca de su número único, así como sobre la cantidad de lotes a ser cerrado, entonces es muy fácil cerrar la orden. Para ello, debe se debe usar en el código del programa la llamada a la función **OrderClose ()** con parámetros preestablecidos. Por ejemplo, si el número único de la orden de Compra es 12345 y si se quiere cerrar 0,5 lotes, la llamada a la función de cierre podría tener este aspecto:

```
OrderClose( 12345, 0.5, Bid, 2 )
OrderClose( ticket, lotes, precio, slippage )
```

Con el fin de decidir sobre que órdenes y en qué secuencia deben cerrarse, se tiene que tener datos de todas las órdenes abiertas en la situación actual. En MQL4, hay una serie de funciones que pueden ser utilizadas para obtener diversos datos que caracterizan a cualquier orden. Por ejemplo, la función **OrderOpenPrice ()** devuelve el valor del precio de apertura de la orden (o del precio solicitado para la espera de las órdenes), la función **OrderLots ()** devuelve la cantidad de lotes, la función **OrderType ()** devuelve el tipo de la orden, etc. Todas las funciones que devuelven los valores de una característica de la orden llama a su ejecución a la orden que ha sido seleccionada por la función **OrderSelect ().**

Función OrderSelect ()

Con el fin de obtener los parámetros de cualquier orden (no importa que sean órdenes a mercado o pendientes, cerrada o eliminada), primero se debe seleccionar utilizando la función **OrderSelect ().**

```
bool OrderSelect(int index, int select, int pool=MODE_TRADES)
```

OrderSelect es una función que selecciona una orden para hacer operaciones con ella. Devuelve TRUE, si la función se ejecuta con éxito. De lo contrario, devuelve FALSE.

Parámetros:

index - Para la posición (numero de orden en la lista) o el número de ticket, depende de el segundo parámetro.

select – flag (bandera) de selección de método. El Parámetro 'select' puede tomar uno de dos posibles valores:

SELECT_BY_POS - en el parámetro 'índex', devuelve el número de orden en la lista (la numeración empieza por 0),

SELECT_BY_TICKET - en el parámetro 'índex', devuelve el número de ticket (el número de orden único).

pool - la fuente de datos para la selección. El parámetro «pool» se utiliza, cuando el parámetro 'select' es igual al valor de SELECT_BY_POS. El parámetro 'pool' se ignora, si la orden es seleccionada por el número de ticket (SELECT_BY_TICKET). El parámetro 'pool' puede tener uno de dos valores posibles:

MODE_TRADES (por defecto) - la orden se selecciona en órdenes abiertas y órdenes a la espera, es decir, entre las órdenes que aparecen en la pestaña "Operaciones" (si el metatrade esta configurado con el idioma español) de la ventana "Terminal"

MODE_HISTORY la orden se selecciona para las órdenes cerradas y eliminadas, es decir, entre las órdenes que aparece en la en la pestaña "Historial de Cuentas" de la ventana "Terminal". En este caso, la profundidad de la historia especificada por el usuario para mostrar órdenes cerradas y eliminadas es importante.

Con el fin de demostrar el método de uso de las funciones para el cierre de las órdenes de mercado, vamos a resolver un problema:



Problema 28. Escribir un script que cierre una de las órdenes de mercado disponibles en la cuenta. La ejecución de scripts debe dar como resultado el cierre de la orden más cercana a la ubicación de la script vinculada con el ratón a la ventana de símbolo.

Supongamos que hay tres órdenes de mercado abiertas en la terminal para el símbolo euro/dólar y en espera de ser ejecutada una orden abierta para USD/CHF:

Terminal

Order	Time	Type	Size	Symbol	Price	S / L	T / P	Price	Comm...	Swap	Profit
4372561	2007.01.22 17:46	buy	0.80	eurusd	1.2956	0.0000	0.0000	1.2947	0.00	0.00	-72.00
4372889	2007.01.22 18:32	sell	0.50	eurusd	1.2953	0.0000	0.0000	1.2949	0.00	0.00	20.00
4372970	2007.01.22 18:44	sell	0.50	eurusd	1.2950	0.0000	0.0000	1.2949	0.00	0.00	5.00
Balance: 4 390.80 Equity: 4 343.80 Margin: 259.07 Free margin: 4 084.73 Margin level: 1676.69%											**-47.00**
4372930	2007.01.22 18:41	buy stop	0.50	usdchf	1.2481	0.0000	0.0000	1.2476			

Trade | Account History | News | Alerts | Mailbox | Journal

Fig. 90. Vista de varias órdenes abiertas para los diferentes símbolos en la ventana del terminal.

Debemos escribir un script que se pueda arrastrar con el ratón de la ventana del "Explorador" (ó "Navigator" si el metatrader esta configurado en ingles) a la ventana de símbolo. La ejecución de dicho script de este modo debería dar como resultado el cierre de una de las órdenes de mercado, a saber, la orden (marcada por una linea discontinua y el numero de ticket) que se encuentre más cercana al cursor (a partir del momento en el que el usuario ha liberado el botón del ratón). En la Fig. 91, se puede ver la alternativa, en la que el cursor se encuentra más próxima a la orden de venta 4372889. Esta será la orden que deberá ser cerrada como consecuencia de la ejecución del scripts.

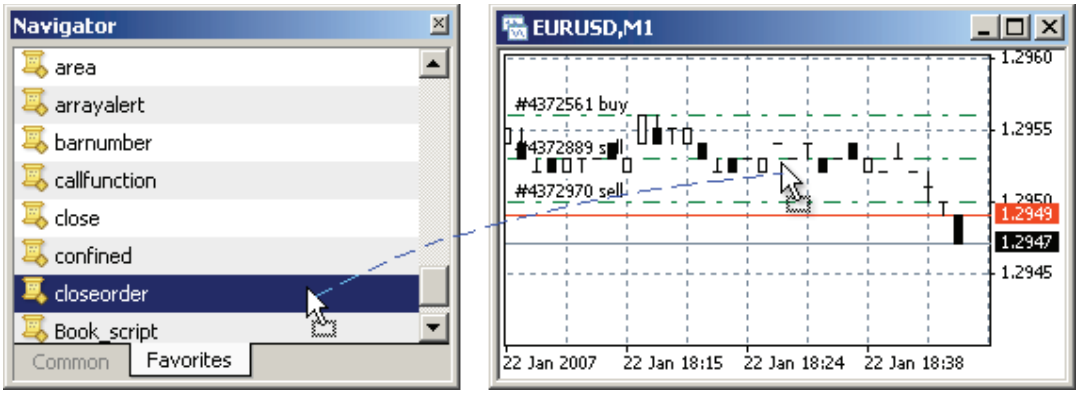


Fig. 91. Script closeorder.mq4 utilizados para el cierre de los seleccionados el orden.

Para resolver este problema, hay que seleccionar (usando la función OrderSymbol ()) una entre todas las órdenes abiertas en la ventana de símbolo, en el cual el script se ha soltado. Entonces tenemos que encontrar los precios de apertura de todas las órdenes de mercado seleccionadas (es decir, ejecutar la función OrderOpenPrice () sucesivamente para cada orden). Conociendo los precios de apertura de las órdenes, podemos fácilmente seleccionar una que corresponde a la declaración del problema. Para especificar los valores apropiados de los parámetros la función **OrderClose ()**, también se necesita conocer algunos otros datos acerca de la orden seleccionada: la cantidad de lotes (deteraminado por la función **OrderLots ()**) y el número único de orden (determinado por la función **OrderTicket ()**). Por otra parte, para encontrar uno u otro precio del **two-way quote** (cotizacion de doble via), tenemos que saber el tipo de orden (determinado por la función **OrderType ()**).

Vamos a considerar qué parámetros deben ser especificados en la función **OrderSelect ()** con el fin de obtener la característica de la orden de arriba.

En primer lugar, es necesario elegir el método de selección de la orden. En nuestro problema, el método de selección está determinado por la propia declaración del problema: Los datos sobre los números de orden se supone que no estarán disponibles en el programa hasta el momento de la puesta en marcha del script para su ejecución, es decir, se considera que el programa contendrá un bloque que determinará los números de orden. Esto significa que se debe comprobar una por una todas las órdenes representadas en "Terminal" (Fig. 64.1), por lo que tenemos que utilizar el parámetro SELECT_BY_POS.

La fuente para la selección de órdenes es evidente, también. Para resolver el problema, no hay necesidad de analizar las órdenes cerradas y suprimidas. En este caso, estamos interesados sólo las órdenes que están en el mercado, por lo que buscamos en ellas utilizando el parámetro MODE_TRADES en la función **OrderSelect ()**. Para el parámetro 'pool', el valor por defecto de MODE_TRADES se especifica en función de la cabecera, por lo que puede saltarse.

A continuación se muestra cómo se puede construir un bloque de análisis de mercado y órdenes en espera de ser ejecutadas:

```
for (int i=1; i<=OrdersTotal(); i++)              //Ciclo para todas las órdenes...
   {                                              //… mostradas en el terminal
    if(OrderSelect(i-1,SELECT_BY_POS)==true) // Si es la proxima orden seleccionada
      {
       // Caracteristicas de la orden...
       // ... debe ser analizada aquí
      }
   }                                 //Fin del cuerpo del ciclo
```

En la cabecera del operador de ciclo, el valor inicial se especifica como i = 1, mientras que la condición para salir del ciclo es la expresión i <= OrdersTotal (). La función **OrdersTotal ()** devuelve la cantidad total de órdenes en mercado y órdenes a la espera pendientes de ser ejecutadas, es decir, la cantidad de órdenes que se muestran en la pestaña "Operaciones" ("Trade" si Metatrader está configurado en ingles) de la ventana "Terminal". Por eso habrá tantas iteraciones en el ciclo, como numero órdenes participantes en el trading.

En cada iteración, cuando en el operador 'if' se calcula la condición, la función OrderSelect (i-1, SELECT_BY_POS) se llevará a cabo.Hay que señalar aquí la siguiente importante cuestión:



La numeración de los órdenes en la lista de órdenes de mercado y órdenes en espera de ser ejecutadas comienza con cero.

Esto significa que la primera orden de la lista (Fig. 90) está localizada en la posición cero, la segunda orden esta localizada en la posición numerada como 1, la tercera orden numerada como 2, etc. Es por ello en la llamada a la función **OrderSelect (),** el valor del índice se da como i-1. Por lo tanto, para todas las órdenes seleccionadas en este índice será siempre inferior en 1 al valor de la variable i (el cual coincide con el número de la próxima iteración).

La función **OrderSelect ()** devuelve true, si la orden se ha seleccionado con éxito. Esto significa que es posible que la selección de una orden pueda fracasar. Esto puede suceder, si la cantidad de órdenes cambia durante su tramitación. Cuando se programa en MQL4, se debe recordar que un programa de aplicación trabajará en el tiempo real, de modo que, mientras se están procesando los parámetros, los valores de estos parámetros pueden cambiar. Por ejemplo, la cantidad de órdenes de mercado puede cambiar como resultado de tanto de la apertura/cierre de órdenes como de la modificación de órdenes pendientes a órdenes de mercado. Esta es la razón por la cual se debería mantener a la siguiente regla cuando se programa procesamiento de órdenes: las órdenes deben ser procesadas tan pronto como sea posible, mientras que el bloque de programa responsable de esta transformación no debe, si es posible, contener líneas redundantes de programa.

De acuerdo con el código representado en la Fig. 64,3, el programa analiza en la cabecera del operador 'if' si la próxima orden en la lista de órdenes está disponible en el momento en que se selecciona. Si la próxima orden está disponible, el control pasa al cuerpo del operador 'if' para el procesamiento de los parámetros de la orden. Cabe señalar que esta construcción, en caso de posibles conflictos, no ayuda mucho, debido a que la orden se puede perder (cerrada) durante el procesamiento de sus parámetros. Sin embargo, esta solución resulta ser más eficiente si, a partir del momento de su selección, la orden ya no está disponible. En el cuerpo del operador 'if', se analizan los parámetros del objeto seleccionado. Al ejecutar las funciones **OrderOpenPrice ()**, **OrderTicket ()**, **OrderType ()** y otras del mismo estilo, cada una de ellas devolverá el valor de una cierta característica de la orden seleccionada como resultado de la ejecución de la función **OrderSelect ()**.

Todo el razonamiento anterior se utiliza en el programa que resuelve el Problema 28.



Un ejemplo de un script simple destinado al cierre de una orden de mercado, cuyo precio de apertura de la ubicación está más próxima del script adjunto que los precios de apertura de las demás órdenes (closeorder.mq4).

```
//+------------------------------------------------------------------------------------+
// closeorder.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------------------- 1 --
int start()                                          // Función especial 'start'
  {
     string Symb=Symbol();                           // Variable de cadena que contiene el simbolo
     double Dist=1000000.0;                          // Preselección de la distancia incial
     int Real_Order=-1;                              // No hay órdenes de mercado todavía
     double Win_Price=WindowPriceOnDropped();        // El script se ha "soltado" en este precio
//+------------------------------------------------------------------------------- 2 --
     for(int i=1; i<=OrdersTotal(); i++)             // Ciclo de búsqueda de ordenes
      {
        if (OrderSelect(i-1,SELECT_BY_POS)==true)    // Si la próxima orden está disponible
         {                                           // Analisis de la orden:
       //+---------------------------------------------------------------------- 3 --
          if (OrderSymbol()!= Symb) continue;        // Si no es nuestro símbolo
          int Tip=OrderType();                       // Tipo de orden (*)
          if (Tip>1) continue;                       // Si la orden es pendiente interrumpe la iteración
       //+---------------------------------------------------------------------- 4 --
          double Price=OrderOpenPrice();             // Precio de apertura de la orden
          if (NormalizeDouble(MathAbs(Price-Win_Price),Digits)< //Selección
          NormalizeDouble(Dist,Digits))              // de la orden mas cercana
            {
               Dist=MathAbs(Price-Win_Price);        // Nuevo valor de la distancia (la mas corta)
               Real_Order=Tip;                       // orden a mercado disponible
               int Ticket=OrderTicket();             // Nº de ticket de la orden seleccionada
               double Lot=OrderLots();               // Cantidad de lotes de la orden a cerrar
            }
       //+---------------------------------------------------------------------- 5 --
          }                                          //Final del analisis de la orden
        }                                            //Final de la busqueda de la orden
//+------------------------------------------------------------------------------- 6 --
      while(true)                                    // Ciclo para el cierre de la orden
       {
          if (Real_Order==-1)                        // Si no hay órdenes a mercado disponibles…
            {
               Alert("For ",Symb," no hay órdenes a Mercado disponibles");
               break;                                // … salida del ciclo de cierre "while"
            }
    //+---------------------------------------------------------------------- 7 --
          switch(Real_Order)                         // Selección por el tipo de orden
            {
               case 0: double Price_Cls=Bid;         // Orden de compra
                     string Text="Compra ";          // Texto para compra
                     break;                          // Salir de switch
               case 1: Price_Cls=Ask;                // Orden de venta
                     Text="Sell ";                   // Texto para venta
             }
          Alert("Intentando el cierre ",Text," ",Ticket,". Esperando respuesta...");
          bool Ans=OrderClose(Ticket,Lot,Price_Cls,2);   // Orden de cierre
    //+---------------------------------------------------------------------- 8 --
          if (Ans==true)                             // ¡Orden ejecutada! :)
            {
              Alert ("Orden cerrada ",Text," ",Ticket);
              break;                                 // Salida del ciclo de cierre
            }
    //+---------------------------------------------------------------------- 9 --
          int Error=GetLastError();                  // Fallo :(
          switch(Error)                              // Errores superables
            {
               case 135:Alert("El precio ha sido cambiado. Reintentando...");
```

```
                    RefreshRates();                    // Actualización de datos
                    continue;                  // Saltar a la proxima iteración de ciclo de busqueda
                case 136:Alert("No hay precio. Esperando un nuevo tick...");
                    while(RefreshRates()==false)       // Mientras no no hay Nuevo tick…
                        Sleep(1);                      // … dormir (pausa 1 msg.)
                    continue;                          // Con nuevo tick saltar a la próxima…
                                                       // … iteración del ciclo  de búsqueda
                case 146:Alert("Sistema de Trading ocupado. Reintentando...");
                    Sleep(500);                        // Solución simple. Pausa 0,5 sg.
                    RefreshRates();                    // Actualización datos de entorno
                    continue;                          // A la próxima iteración del ciclo de búsqueda
                }
            switch(Error)                              // Errores críticos
                {
                case 2 : Alert("Common error.");
                    break;                             // Salir de 'switch'
                case 5 : Alert("Versión antigua del terminal de usuario.");
                    break;                             // Salir de 'switch'
                case 64: Alert("La  cuenta está bloqueada.");
                    break;                             // Salir de 'switch'
                case 133:Alert("Trading no permitido");
                    break;                             // Salir de 'switch'
                default: Alert("Ha habido un error ",Error);//Otras alternativas
                }
            break;                                     // Salir del ciclo de cierre
        }
//---------------------------------------------------------------------------- 10 --
  Alert ("El script ha finalizado las operaciones ----------------------------");
  return;                                              // Salida de start()
 }
//---------------------------------------------------------------------------- 11 --
```

Todo el código del programa closeorder.mq4 se concentra en la función especial start (). En el bloque 1-2, se inicializan algunas variables. La variable Dist es la distancia desde el lugar donde el script ha "soltado" a la orden más cercana. La variable Real_Order es una bandera que muestra la disponibilidad de al menos una orden de mercado en el Terminal de Usuario (valor no negativo). La variable Win_Price es el precio, al cual el usuario ha asignado el script para la ventana del símbolo. En el bloque 2-6, se analiza la orden: Una de las órdenes disponibles se asigna para ser cerrada. El bloque 6-10 es el bloque de procesamiento para el cierre de la orden y los errores que puedan ocurrir durante el desempeño del comercio.

A partir del momento en que el usuario asigna el script a la ventana de símbolo. En el bloque 1-2 se calculan los valores de las variables, la variable Win_Price toma el valor del precio, al nivel al que el usuario asignó el script. Ahora es necesario encontrar la orden (con sus características) que esta más cerca de esta ubicación.

Las órdenes se buscan dentro del ciclo 'for' (bloque 2-6). En el bloque 2-3, el programa comprueba si hay una orden en la línea siguiente del "Terminal". Si se encuentra una orden, el control se pasa al cuerpo del operador 'if' para obtener y analizar las características de ese orden. En el bloque 3-4, las órdenes abiertas en el símbolo equivocado (no el símbolo, para el cual el programa está siendo ejecutando) son filtradas fuera. En nuestro caso, es la orden 4372930 abierta para el USD/CHF. La Función **OrderSymbol ()** devuelve el nombre del símbolo de la orden seleccionada. Si este nombre del símbolo es distinto, para el que el programa se está ejecutando, la iteración actual se interrumpe (continue), para evitar que la orden sea ejecutada para otro símbolo distinto desde el que está siendo procesado. Si la orden bajo análisis resulta ser abierta a "nuestro" símbolo, un nuevo chequeo se llevará a cabo. El tipo de orden se determina utilizando la función OrderType () (véase Tipos de Operaciones). Si el tipo de orden resulta ser mayor de 1, significa que la orden es un tipo de orden pendiente. En este caso, la actual iteración se interrumpe también (continue), porque no estamos interesados en las órdenes en espera. En nuestro ejemplo, tenemos una orden de este tipo, pero que se abre para otro símbolo, por lo que ya ha sido filtrada. Todas las órdenes que pasan el bloque de 3-4 con éxito son órdenes de mercado.

El Bloque 4-5 se destina para la selección de una única orden de mercado de entre todas las órdenes que han pasado con éxito el bloque anterior. Esta orden debe ser la más cercana al precio predefinido (el valor de la variable Win_Price). El usuario no está obligado a "localizar" con su ratón la linea de la orden de forma exacta. La orden seleccionada será la que, de entre las demás órdenes, esté más cerca del cursor desde del momento de poner en marcha el script para su ejecución. El precio de apertura de la orden procesada se encuentra utilizando la función **OrderOpenPrice ()**. Si el valor absoluto de la distancia entre el precio de la orden actual y el "cursor de precios" es inferior a la misma distancia de la orden anterior, la orden actual será seleccionada. Es necesario usar el valor absoluto de la distancia para que no importe si la posición del cursor está por debajo o por encima de la línea indicadora de la orden. De este modo solo se considera la distancia y no su signo. La orden seleccionada será memorizada en la iteración actual del ciclo 'for' como la pionera para ser cerrada. Para este fin, al final del bloque 4-5, se calcula el número de ticket (el número individual de la orden) y la cantidad de lotes. En el ejemplo (Fig. 90), la cantidad total de los órdenes es cuatro (tres de mercado y una orden en espera), por lo que habrá cuatro repeticiones ejecutadas en el ciclo 'for', lo cual conlleva a la búsqueda de todos los datos necesarios para realizar el cierre del objeto seleccionado.

A continuación, el control en la ejecución del programa se pasa al operador de ciclo 'while' (bloque 6-10). En el bloque 6-7, las órdenes de mercado encuentran un control de disponibilidad. Si no se encuentran órdenes de mercado en el bloque 2-4 (es muy posible en general), el valor de la bandera Real_Order sigue siendo igual a -1, lo que significará falta de órdenes de mercado. Si el control en el bloque 6-7 no detecta órdenes de mercado, la ejecución del ciclo 'while' se rompe y entonces el programa pone fin a sus operaciones. Si el valor de la variable Real_Order resulta ser igual a 0 o 1, esto significa que un mercado está predefinido para el cierre y debe cerrarse.

En el bloque de 7-8, de acuerdo al tipo de orden, se calcula el precio de cierre de la orden. Es el valor del Bid para las órdenes de compra, y el valor de Ask para las órdenes de venta (ver requisitos y limitaciones en la toma de Órdenes).

En el bloque de 7-8, se calculan los valores de la variable auxiliar Texto. La solicitud de la orden de cierre de comercio se forma en la función **OrderClose ()** en la línea siguiente:

```
bool Ans=OrderClose(Ticket,Lot,Price_Cls,2);// Orden de cierre
```

La función de comercio **OrderClose ()** devuelve true, si la transación se realiza con éxito, y falso, si no es así. Si la solicitud es ejecutada exitosamente en el servidor, el valor «true» se asignará a la variable Ans (respuesta). En este caso, al ejecutar bloque 8-9, el programa informará al usuario sobre el éxito de la orden de cierre. Después de eso, la ejecución del operador de ciclo 'while' se detendrá, y el programa pondrá fin a sus operaciones. De lo contrario, el control pasa al bloque de 9-10 con el fin de analizar los errores devueltos por el Terminal de Usuario al programa.

Al comienzo del bloque 9-10, se analiza el código de error, y de acuerdo a éste, o bien se sale del programa o bien se ejecuta la operación de iteración. En el primer operador de 'switch', el programa procesa los errores que son implícitamente superables, es decir, los errores pueden ser considerados como temporales dificultades en el desempeño del comercio. Se toman todas las acciones necesarias para cada uno de esos errores, entonces la actual iteración se detiene y la ejecución del ciclo 'while' se reinicia. (Tengase en cuenta que, en este ejemplo, el uso de Error al procesar el operador 'switch' se salió como consecuencia de la utilización del operador en "continue" que, como tal, no está destinado a pasar el control fuera del operador 'switch'. La construcción de este diseño es justo así porque el operador 'switch' es una parte del contenido del ciclo externo del operador 'while' y el operador 'continue' interrumpe la iteración actual pasando el control a la cabecera del operador 'while').

Si el código de error no es procesado en el primer operador de 'switch', este error se considera crítico. En este caso, el control pasa al segundo operador 'switch', que se ejecuta con el próposito de informar al usuario de que ha ocurrido algún tipo de error crítico. Además, el programa utiliza el operador "break" que interrumpe la ejecución del ciclo 'while'. Si se sale del ciclo 'while', por cualquier razón, esto tendrá como resultado el paso del control al bloque 9-10 que produce un mensaje que informa acerca del final de las operaciones del programa. El operador «return» detiene la ejecución de la función especial start (), y el programa finaliza sus operaciones.

A continuación se muestra el práctico resultado que se obtiene tras el lanzamiento del script bajo las condiciones establecidas (ver Fig. 90 y 91). El comercio se ha realizado con éxito en el servidor.

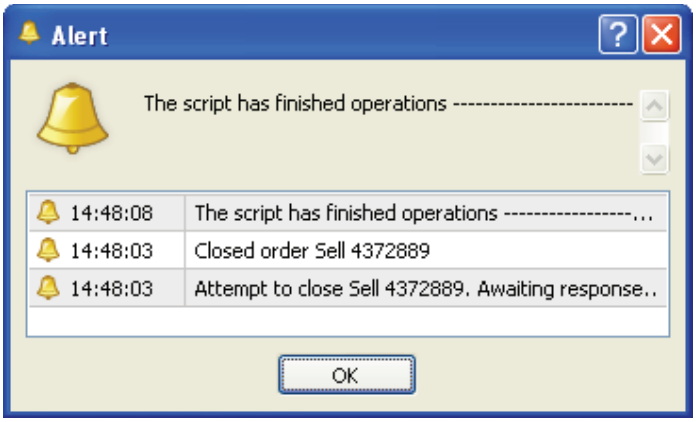


Fig. 92. Los mensajes recibidos como consecuencia de la ejecución exitosa del script closeorder.mq4.

Como resultado del cierre de una de las órdenes, hay dos órdenes a la izquierda de la ventana de EUR/USD.



Fig. 93. La ejecución del script closeorder.mq4 resultados en el cierre de uno de los órdenes.

La orden de cierre también ha sido mostradas en la ventana del "Terminal":

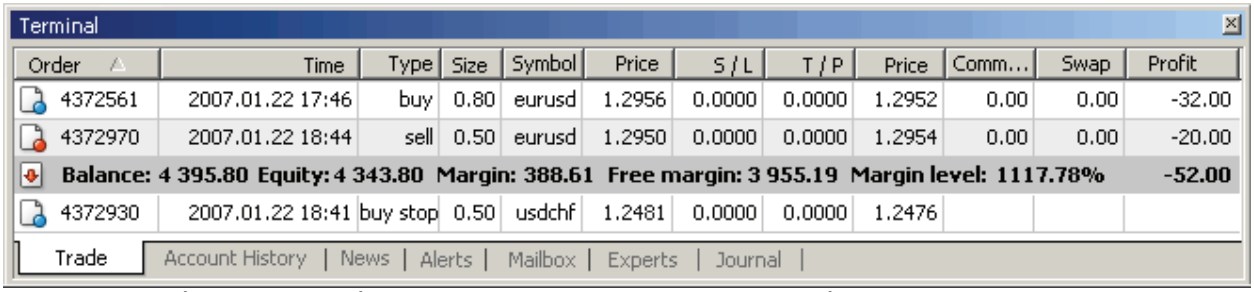


Fig. 94. Después de la Ejecución de comandos de closeorder.mq4, dos órdenes de mercado se muestran en el "Terminal" ventana ".

Más tarde, las otras dos órdenes se cierran tambien usando este script.

Eliminar órdenes en espera de ser ejecutadas

Las solicitudes de comercio para eliminación de órdenes pendientes se forman utilizando la función **OrderDelete ()**.

Función OrderDelete

```
bool OrderDelete(int ticket, color arrow_color=CLR_NONE)
```

La función elimina la orden pendiente previa. Devuelve TRUE, si ha funcionado con éxito. De lo contrario, devuelve FALSE.

Parámetros:

ticket - el número único de la orden.

arrow_color - el color de una flecha en un gráfico. Si este parámetro no está disponible o su valor es igual a la de CLR_NONE, la flecha no se muestra en el gráfico.

Es fácil ver que la función OrderDelete () no contiene una especificación del volumen y el precio de cierre de la orden a ser eliminada.

Cualquier orden se borra independientemente de cualquier precio de mercado. La supresión parcial de una orden no es posible. Puede disminuir la cantidad de lotes de una orden a la espera en dos etapas: suprimir la orden existente y, a continuación, colocar una nueva orden pendiente con la cantidad de lotes decrementada.

El algoritmo del programa que borrará una orden pendiente puede ser completamente idéntico a la orden de cierre de mercado. Una ligera diferencia está en que no es necesario un precio de cierre para eliminar una orden pendiente, por lo que el programa siguiente no contiene el bloque que actualiza los precios de mercado.



Un ejemplo de un script simple destinado a la supresión de una orden pendiente, el precio de la solicitud es el que está más cerca de la ubicación del script adjunto que los precios de otras órdenes pendientes (deleteorder.mq4).

```
//+------------------------------------------------------------------+
// deleteorder.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+ 1 --
int start()                                                 // Funcion especial  'start'
  {
        string Symb=Symbol();                               // Simbolo
        double Dist=1000000.0;                              // Preselección distancia inicial
        int Limit_Stop=-1;                                  // Todavía no hay órdenes pendientes
        double Win_Price=WindowPriceOnDropped();            // El script se ha soltado a este precio
//+------------------------------------------------------------------+ 2 --
        for(int i=1; i<=OrdersTotal(); i++)                 // Ciclo de búsqueda de órdenes
          {
            if (OrderSelect(i-1,SELECT_BY_POS)==true)       // Si la próxima órden esta disponible
              {                                             // Analisis de la orden disponible:
      //+------------------------------------------------------------------+ 3 --
                    if (OrderSymbol()!= Symb) continue;     // El símbolo no es el nuestro
                    int Tip=OrderType();                    // Tipo de orden
                    if (Tip<2) continue;                    // (*)Si no es una orden pendiente, nueva …
```

```
                                                        // iteración del ciclo de búsqueda de órdenes
      //------------------------------------------------------------------ 4 --
                    double Price=OrderOpenPrice();      // Precio orden de apertura de orden seleccion.
                    if (NormalizeDouble(MathAbs(Price-Win_Price),Digits)< //Selección de la orden si …
                    NormalizeDouble(Dist,Digits))       // la distancia menor que la orden anterior
                      {
                          Dist=MathAbs(Price-Win_Price); // Nuevo valor de la distancia mínima
                          Limit_Stop=Tip;               // Orden pendiente disponible
                          int Ticket=OrderTicket();     // Nº de ticket de la orden
                       }                                // Fin de 'if'
                 }                                      // Fin del analisis de órdenes
            }                                           // Fin de la búsqueda de órdenes
//------------------------------------------------------------------------------ 5 --
        switch(Limit_Stop)                              // Por tipo de orden (*)
         {
              case 2: string Text= "BuyLimit ";         // Texto para BuyLimit
                      break;                            // Salir de 'switch'
              case 3: Text= "SellLimit ";               // Texto para SellLimit
                      break;                            // Salir de 'switch'
              case 4: Text= "BuyStopt ";                // Texto para BuyStopt
                      break;                            // Salir de 'switch'
              case 5: Text= "SellStop ";                // Texto para SellStop
                      break;                            // Salir de 'switch'
         }
//------------------------------------------------------------------------------ 6 --
        while(true)                                     // Ciclo para orden de cierre
          {
              if (Limit_Stop==-1)                       // Si no hay órdenes pendientes disponibles
                {
                      Alert("For ",Symb," No hay órdenes pendientes disponibles");
                      break;                            // Salida del ciclo de cierre
          }
    //-------------------------------------------------------------------- 7 --
        Alert("Intentando suprimir la orden ",Text," ",Ticket,". Esperando respuesta...");
        bool Ans=OrderDelete(Ticket);                   // Supresión de la orden
    //-------------------------------------------------------------------- 8 --
        if (Ans==true)                                  // ¡Conseguido! :)
          {
              Alert ("Orden suprimida: ",Text, "Ticket: " ,Ticket);
              break;                                    // Salida del ciclo de cierre
          }
    //-------------------------------------------------------------------- 9 --
        int Error=GetLastError();                       // Fallo :(
        switch(Error)                                   // Errores superables
          {
              case  4: Alert("El servidor de trades está ocupado. Reintentar...");
                      Sleep(3000);                      // Solución simple
                      continue;                         // A la próxima iteración
              case 137:Alert("El broker está ocupado. Reintentar...");
                      Sleep(3000);                      // Solución simple
                      continue;                         // A la próxima iteración
              case 146:Alert("El subsistema de Trading está ocupado. Reintentando...");
                      Sleep(500);                       // Solución simple
                      continue;                         // A la próxima iteración                }
        switch(Error)                                   // Critical errors
          {
              case 2 : Alert("Error común.");
```

```
                  break;                                          // Salida 'switch'
               case 64: Alert("Cuenta bloqueda.");
                  break;                                          // Salida 'switch'
               case 133:Alert("Trading está prohíbido");
                  break;                                          // Salida 'switch'
               case 139:Alert("La orden está bloqueada y está siendo procesada");
                  break;                                          // Salida 'switch'
               case 145:Alert("La modificicación está prohíbida. ",
                     "La orden está demasiado cerca del mercado");
                  break;                                          // Salida 'switch'
               default: Alert("Ha ocurrido el error ",Error);     // Otros errores
            }
         break;                                                   // Salida del ciclo de cierre
      }
//---------------------------------------------------------------------------- 10 --
   Alert ("El stript ha finalizado las operaciones ----------------------------");
   return;                                                        // salida de start()
 }
//---------------------------------------------------------------------------- 11 --
```

(*)

Constante	Valor	Transación comercial
OP_BUY	0	Comprar
OP_SELL	1	Vender
OP_BUYLIMIT	2	En espera para comprar límite
OP_SELLLIMIT	3	En espera de fin de vender límite
OP_BUYSTOP	4	En espera para dejar de comprar
OP_SELLSTOP	5	En espera para dejar de vender

El error de procesamiento de bloque también se ha modificado ligeramente. Se debería considerar la posibilidad de errores relacionados con cambios en los precios (errores 135 y 136) cuando se cierre el mercado de órdenes, pero tales errores no se producen cuando se suprima las órdenes en espera. Por la misma razón, la función RefreshRates () no se utiliza en ninguna parte del programa.

El tratamiento de tales errores como error 4 y error 137 (véase códigos de error) puede ser un poco difícil. Por ejemplo, cuando Obtengo el error 137, el programa puede tener en cuenta que "el corredor está ocupado". Sin embargo, una cuestión que se plantea es natural: ¿Cuándo esta el broker libre, para que el usuario pueda continuar su comercio? Error 137 no proporcionar dicha información. Es por ello que el programador debe decidir por sí mismo cómo construir el programa para procesar esos errores de forma apropiada. En un caso simple, la solicitud puede repetirse después de una cierta pausa (en nuestro ejemplo, en 3 segundos). Por otra parte, tras una serie de intentos fallidos para eliminar (o, en un caso común, para cerrar, abrir o modificar) una orden, el servidor puede devolver error 141 - demasiadas peticiones. Este error da lugar a que el script deleteorder.mq4 deja de funcionar. En general, estos conflictos no son los asuntos de programación. En estos casos, debe ponerse en contacto con el dealing center del servicio de apoyo y clarificar las razones del el rechazo al ejecutar la solicitud.

Error 145 se puede producir, si una orden en espera (en un caso común, puede ser una orden de stop de una orden de mercado) está demasiado cerca del precio de mercado. Este error no se produce, si está de manera constante negociando en un mercado en calma. Si los precios cambian con mucha rapidez, su agente puede decidir que una determinada solicitud será abierta pronto, por lo que el corredor no permitirá suprimir o modificar. Este error se considera en el script como un error crítico y resulta en la terminación del programa (que no tiene sentido preocuparse por el broker con las solicitudes de comercio). Si los precios cambian después de un tiempo, se puede probar a borrar la orden poniendo de nuevo en marcha el script para su ejecución.

En general, los casos de error 145 se pueden prevenir, si se considera el nivel de congelación fijado por el dealing center. El nivel de congelación es un valor que determina el precio de la banda en la que las ordenes se consideran como «congeladas» y no se permite borrar las órdenes. Por ejemplo, si una orden pendiente de ser ejecutada se coloca en 1.2500 y el nivel de congelación es igual a 10 puntos, significa que, si el precio oscila entre los 1.2490 y 1.2510, no se permite la supresión de la orden en espera dentro de esta banda. Se puede obtener el nivel del valor de congelación después de haber ejecutado la función MarketInfo () con la solicitud de identificación de MODE_FREEZELEVEL.

Cierre de órdenes opuestas

Frente a (contra) la Orden es una orden de mercado abierta en una dirección opuesta a la dirección de otra orden de mercado abierta en un mismo símbolo.

Si se tiene dos órdenes enfrentadas en un determinado símbolo, se pueden cerrar al mismo tiempo, una a otra, utilizando la función **OrderCloseBy ()**. Si se realiza una operación de ese tipo se ahorra el spread de una de las dos ordenes.

Función OrderCloseBy ()

```
bool OrderCloseBy(int ticket, int opposite, color Color=CLR_NONE)
```

La función se cierra un orden de mercado contra otra orden de mercado abiertas en direcciones opuesta en un mismo símbolo. La función devuelve TRUE, en caso de que se complete con éxito, y FALSE, si no es así.

Parámetros:

ticket – es el número único de la orden de cierre.

opposite – es el número único de la orden opuesta.

Color - el color de la flecha de cierre en un gráfico. Si este parámetro no está disponible o su valor es igual a la de CLR_NONE, la flecha no se muestra en el gráfico.

No es necesario que las órdenes opuestas tengan el mismo volumen. Si se cierra una orden con la orden opuesta, la operación se realizará con el volumen de la orden que tenga el menor volumen.

Veamos un ejemplo. Vamos ha tener dos ordenes de mercado del mismo volumen en el Terminal de usuario, una Buy y otra Sell. Si cerramos una de ellas separadamente usando la function OrderClose(), económicamente, nuestra salida será la suma de las ganancias obtenidas por cada orden:

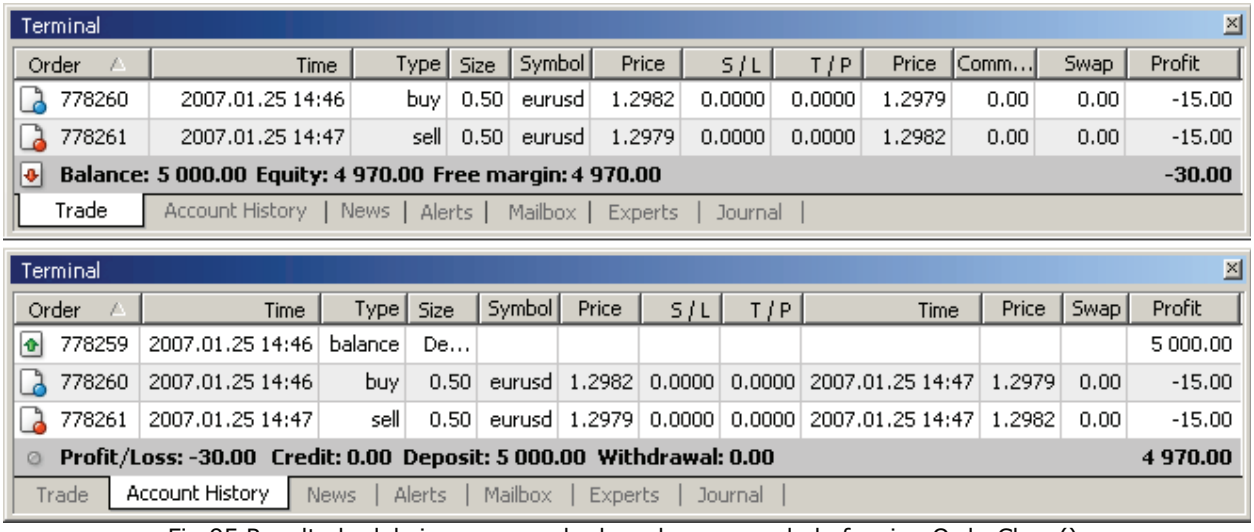


Fig.95 Resultado del cierre separado de ordenes usando la funcion OrderClose().

Sin embargo, si en esta situación usamos la function OrderCloseBy() intentando cerrar las órdenes opuestas, desde el punto de vista economico la salida sera mejor (si se compara con la alternative precedente) en una cantidad que es proporcional al coste del spread de una orden:

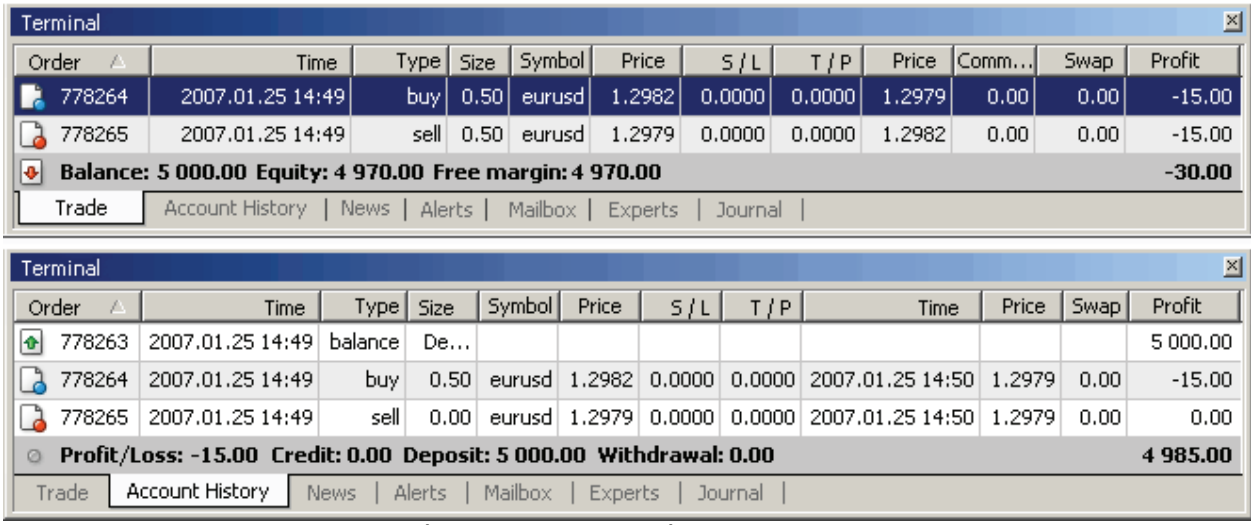


Fig.96 Resultado de cierre de órdenes contra otras órdenes usando the function OrderCloseBy().

Parece obvio que si hay órdenes opuestas para ser cerradas en el terminal, económicamente lo mejor es usar la función OrderCloseBy(), y no OrderClose().

Para saber por que se ahorra un spread en el cierre de las órdenes opuestas, deberíamos dar algunas explicaciones adicionales. Como hecho relevante, a la apertura de una orden (por ejemplo, una orden Buy de compra) está implicita una operación que es de dirección opuesta a la apertura, es decir, una orden Sell de venta. De la misma manera ocurre en el cierre de una orden (orden Buy de compra). En otras palabras, es económicamente lo mismo que las alternativas de uso de: cerrar un orden de mercado o abrir una orden opuesta con el mismo volumen (y entonces cerrar ambas ordenes una contra otra). La diferencia entre estas dos alternativas puede solamente consistir en el uso de diferentes métodos usados en difererentes dealing centers para calcular el dinero que es desviado para soportar las órdens de mercado. (ver Fig. 85 y Fig. 88).

Es tambien fácil ver que, para cerrar las órdenes opuestas con la function OrderCloseBy() no es necesario especificar el precio de cierre. No es necesario porque el beneficio y la pérdida de las dos órdenes opuestas se liquidan o compensan mutuamente, así el resultado economico no depende del precio del mercado. Desde luego, esta regla es efectiva solamente para órdenes del mismo volumen. Si por ejemplo, tenemos dos órdenes para un mismo simbolo: una orden Buy de 1 lote y una Sell de 0.7 lotes, esta operación estará subordinada al precio del Mercado solamente la parte relativa a la orden de compra Buy de 0.3 lotes, mientras que los 0.7 lotes de ambas ódenes no dependen del precio del símbolo.

Las órdenes opuestas no influyen en el resultado total de la operación. Esto es por que las operaciones tácticas basadas en aperturas de ordenes opuestas no tienen contenido informal (por esta razón algunos dealing centers fuerzan el cierre con alguna orden opuesta dentro de la coincidencia de la cantidad de lotes). La sola influencia (negativa) de tales tácticas puede consistir en desviar dinero de acuerdo a las reglas aceptadas en algunos dealing centers. Además, la disponibilidad de varias órdenes opuestas proven más dificultades que una sola orden en el contexto de la programación de trading. Si consideramos varias comisiones y swaps (para cada orden de mercado separadamente), la necesidad de cerrar ordenes opuestas se hace obvia.

Ejemplo de un script simple que cierra todas las órdenes opuestas para un simbolo. (closeby.mq4).

```
//+------------------------------------------------------------------+
// closeby.mq4
// The code should be used for educational purpose only.
```

```
//------------------------------------------------------------------ 1 --
int start()                                 // Special function 'start'
  {
   string Symb=Symbol();                    // Symbol
   double Dist=1000000.0;                   // Presetting
//------------------------------------------------------------------ 2 --
   while(true)                              // Processing cycle..
    {                                       // ..of opposite orders
     double Hedg_Buy = -1.0;                // Max. cost of Buy
     double Hedg_Sell= -1.0;                // Max. cost of Sell
     for(int i=1; i<=OrdersTotal(); i++)    // Order searching cycle
       {
        if(OrderSelect(i-1,SELECT_BY_POS)==true)// If the next is available
          {                                 // Order analysis:
         //--------------------------------------------------- 3 --
           if (OrderSymbol()!= Symb) continue; // Symbol is not ours
           int Tip=OrderType();             // Order type
           if (Tip>1) continue;             // Pending order
         //--------------------------------------------------- 4 --
           switch(Tip)                      // By order type
             {
              case 0:                       // Order Buy
                if (OrderLots()>Hedg_Buy)
                  {
                   Hedg_Buy=OrderLots();    // Choose the max. cost
                   int Ticket_Buy=OrderTicket();//Order ticket
                  }
                break;                      // From switch
              case 1:                       // Order Sell
                if (OrderLots()>Hedg_Sell)
                  {
                   Hedg_Sell=OrderLots();   // Choose the max. cost
                   int Ticket_Sell=OrderTicket();//Order ticket
                  }
             }                              //End of 'switch'
          }                                 //End of order analysis
       }                                    //End of order searching
     //--------------------------------------------------------- 5 --
     if (Hedg_Buy<0 || Hedg_Sell<0)         // If no order available..
       {                                    // ..of some type
        Alert("All opposite orders are closed :)");// Message
        return;                             // Exit start()
       }
     //--------------------------------------------------------- 6 --
     while(true)                            // Closing cycle
       {
        //----------------------------------------------------- 7 --
        Alert("Attempt to close by. Awaiting response..");
        bool Ans=OrderCloseBy(Ticket_Buy,Ticket_Sell);// Закрытие
        //----------------------------------------------------- 8 --
        if (Ans==true)                      // Got it! :)
          {
           Alert ("Performed closing by.");
           break;                           // Exit closing cycle
          }
        //----------------------------------------------------- 9 --
```

```
          int Error=GetLastError();            // Failed :(
          switch(Error)                        // Overcomable errors
            {
               case  4: Alert("Trade server is busy. Retrying..");
                     Sleep(3000);                  // Simple solution
                     continue;                   // At the next iteration
               case 137:Alert("Broker is busy. Retrying..");
                     Sleep(3000);                  // Simple solution
                     continue;                   // At the next iteration
               case 146:Alert("Trading subsystem is busy. Retrying..");
                     Sleep(500);                   // Simple solution
               continue;                     // At the next iteration
             }
          switch(Error)                        // Critical errors
             {
               case 2 : Alert("Common error.");
                     break;                      // Exit 'switch'
               case 64: Alert("Account is blocked.");
                     break;                      // Exit 'switch'
               case 133:Alert("Trading is prohibited");
                     break;                      // Exit 'switch'
               case 139:Alert("The order is blocked and is being processed");
                     break;                      // Exit 'switch'
               case 145:Alert("Modification is prohibited. ",
                        "The order is too close to market");
                     break;                      // Exit 'switch'
               default: Alert("Occurred error ",Error);//Other alternatives
              }
          Alert ("The script has finished operations ---------------------------");

       }                          // Final ciclo de busqueda

     }                            // End of the processing cycle
//------------------------------------------------------------------ 10 --
  }                               // End of start()
//--------------------------------------------------------------------
```

El algoritmo del script de arriba es algo diferente del script precedente. La diferencia consiste en que un mismo codigo puede ser ejecutado muchas veces con el objeto de cerrar varias órdenes con éxito. No está limitada la cantidad de órdenes para ser cerradas. El script fue testado sobre un juego de órdenes de mercado aleatorio. Cinco órdenes de diferentes volúmenes están representadas en la figura Fig. 97 de abajo.



Terminal

Order	Time	Type	Size	Symbol	Price	S / L	T / P	Price	Comm...	Swap	Profit
778592	2007.01.25 19:38	buy	0.50	eurusd	1.2981	0.0000	0.0000	1.2978	0.00	0.00	-15.00
778593	2007.01.25 19:38	sell	0.50	eurusd	1.2978	0.0000	0.0000	1.2981	0.00	0.00	-15.00
778594	2007.01.25 19:38	buy	1.00	eurusd	1.2981	0.0000	0.0000	1.2978	0.00	0.00	-30.00
778595	2007.01.25 19:38	sell	0.80	eurusd	1.2978	0.0000	0.0000	1.2981	0.00	0.00	-24.00
778596	2007.01.25 19:38	sell	0.30	eurusd	1.2978	0.0000	0.0000	1.2981	0.00	0.00	-9.00
Balance: 5 000.00 Equity: 4 907.00 Margin: 100.00 Free margin: 4 807.00 Margin level: 4907.00%											**-93.00**

Trade | Account History | News | Alerts | Mailbox | Experts | Journal

Fig. 97. Órdenes de Mercado abiertas para un símbolo.

Con objeto de cerrar las ordenes opuestas disponibles, predefiniremos un criterio de selección. Este criterio dado en el algoritmo es el tamaño de la orden. Primero se cierran las órdenes de mayor volumen, después se cierran las órdenes de volumen más pequeño. Después de que las órdenes opuestas de diferentes volumenes han sido cerradas, quedan las ódenes del resto de volúmenes. Por ejemplo, el cierre de órdenes opuestas de compra Buy (1 lote) y venta Sell (0.8 lotes) resultará en que las órdenes de compra Buy (0.2 lotes) quedarán abiertas. Esto es por que, después de cada cierre realizado con éxito, el programa debe referirse a la lista de órdenes actualizada para encontrar dos órdenes opuesta de tamaño grande.

Los cáculos de arriba son realizados en un (condicional) continuo ciclo 'while', en el bloque “2-10”. En el comienzo del ciclo de cada iteración el programa supone que no hay órdenes de un cierto tipo. Por esto, el valor de -1 es asignado a la variable Hedg_Buy y Hedg_Sell. El algoritmo del bloque de procesamiento de la orden, es en general preservado (ver el codigo de closeby.mq4). En el bloque de busqueda de ordenes del ciclo 'for', bloque 3-4, como en los precedentes programas, son filtradas las órdenes "erroneas". En este caso, estas son órdenes abiertas por otro símbolo y las órdenes pendientes.

En el bloque 4-5, se calcula el volumen de cada orden chequeada en el bloque 3-4. Si aparecen durante los cálculos, que la orden que actualmente siendo procesada, es la de mayor volumen entre todas órdenes procesadas, el número de ticket de esta orden es almacenado. Esto significa que la orden que tiene este ticket es, en este estado del cálculo, un candidato para el cierre de órdenes opuestas. Por el momento cuando la última iteración del ciclo 'for' ha terminado, los tickets de las órdenes con cantidad de lotes abiertos en dirección opuesta ya son conocidos. Estas órdenes son seleccionadas por programa. Si alguna orden de algún tipo se ha vuelto ya indisponible por el momento los bloques 5-6 salen del programa.

Los bloques 6-10 representan error de procesamiento. Es completamente igual que los que hemos considerado mas arriba (en esta y en las precedentes secciónes). Las operaciones requeridas para cerrar las ordenes opuestas estan formadas en el bloque 7-8 usando la función OrderCloseBy(). Si esto falla de acuerdo a un codigo de error, el programa pasa el control o bien reintenta hacer la operación (para el mismo ticket) o al operador 'return' que finaliza las operaciones del programa.

Si la operación se realiza con exito, el programa sale del bloque de precesamiento de error, y la actual iteracion del ciclo 'while' mas externa finalizará. En la siguiente iteración de este ciclo, serán repetidos todos los calculos: buscando en las órdenes disponibles, seleccionando en las ordenes de mercado, seleccionado un ticket por cada tipo de orden, formando una petición de comercio para cierre de ordenes opuestas y el subsecuente analisis de error. Este ciclo se ejecuta hasta que no hay órdenes disponibles de un cierto tipo (o, en un caso particular de ambos tipos) en el terminal. Este evento sera calculado en el bloque 5-6 y entonces el programa finaliza sus operaciones.

El siguiente mensaje fue recibido en la ejecución de un script closeby.mq4 cuyas órdenes de cierre de marcado son mostradas en Fig. 97:

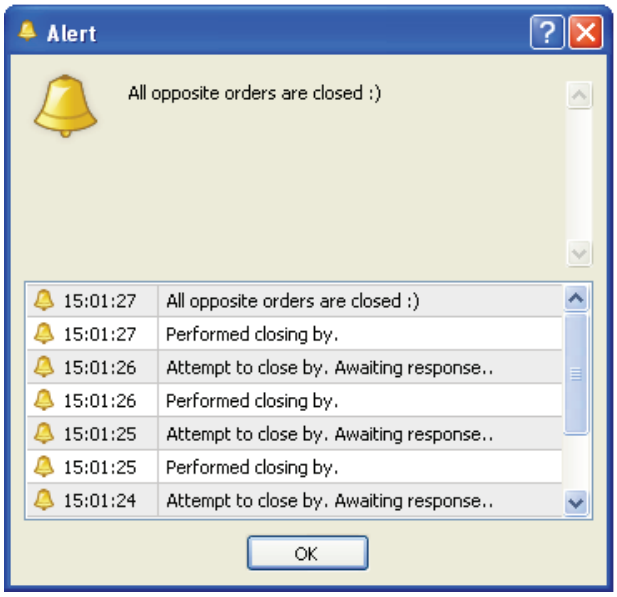


Fig. 98. Mensajes recibido a la ejecución del script closeby.mq4 .

En el pestaña "Historial de la cuenta" de la ventana del "Terminal", se puede ver que algunas órdens son cerradas con beneficio cero. Esto es por que ahorramo cuando cerramos ordenes opuestas. Se Puede comparar los resultados económicos en las Fig. 97 y Fig. 99:

Terminal

Order	Time	Type	Size	Symbol	Price	S / L	T / P	Time	Price	Swap	Profit
778591	2007.01.25 19:37	balance	De...								5 000.00
778595	2007.01.25 19:38	sell	0.00	eurusd	1.2978	0.0000	0.0000	2007.01.25 19:39	1.2978	0.00	0.00
778592	2007.01.25 19:38	buy	0.50	eurusd	1.2981	0.0000	0.0000	2007.01.25 19:39	1.2978	0.00	-15.00
778593	2007.01.25 19:38	sell	0.00	eurusd	1.2978	0.0000	0.0000	2007.01.25 19:39	1.2978	0.00	0.00
778594	2007.01.25 19:38	buy	0.80	eurusd	1.2981	0.0000	0.0000	2007.01.25 19:39	1.2978	0.00	-24.00
778596	2007.01.25 19:38	sell	0.00	eurusd	1.2978	0.0000	0.0000	2007.01.25 19:39	1.2978	0.00	0.00
778597	2007.01.25 19:38	buy	0.20	eurusd	1.2981	0.0000	0.0000	2007.01.25 19:39	1.2978	0.00	-6.00
Profit/Loss: -45.00 Credit: 0.00 Deposit: 5 000.00 Withdrawal: 0.00											**4 955.00**

Trade | Account History | News | Alerts | Mailbox | Experts | Journal

Fig. 99. Historial de la cuentas después de la ejecución del script closeby.mq4 .

Sobre la pestaña “Diario” en la ventana del "Terminal", se puede ver la historia del cierre de la orden (el último evento esta en la parte alta):

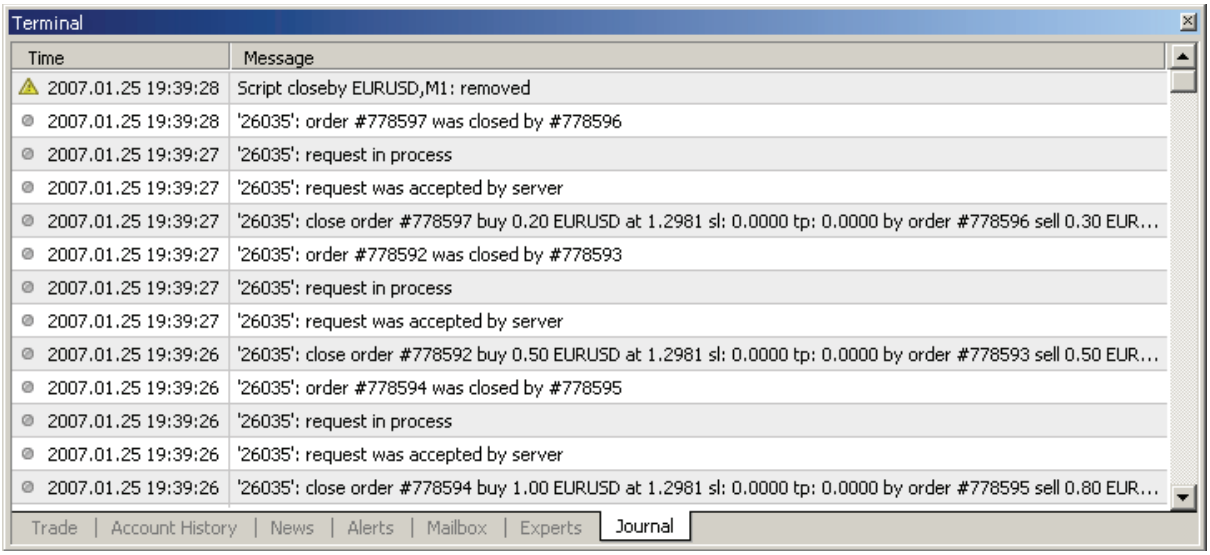


Fig. 100. Eventos ocurridos durante la ejecución del script closeby.mq4 .

En la ejecución del script, de acuerdo al algoritmo, serán cerradas las órdenes de máximo volumen disponibles en ese momento. De manera que, del hecho de que las órdenes fueran abiertas en una secuencia aleatoria (Fig. 97), las primeras órdenes cerradas fueron Buy 778594 and Sell 778595, con los volumenes de 1 lote y 0.8 lotes respectivamente (la linea mas baja en la Fig. 100). Desde estas órdenes que tienen diferentes volumenes, la nueva orden de compra de cierre por oposición Buy 778597, con la diferencia de volumen de 0.2 lotes. Entonces la orden de compra seleccionada Buy 778592 y de venta Sell 778593, de 0.5 lotes cada una, es cerrada como orden de oposición. Estas órdenes son cerradas sin la apertura de un resto de orden.

Por el momento, el comienzo de la tercera iteración, dos órdenes han permanecido en la ventana del simbolo en el ciclo externo: La orden inicial de venta Sell 778596 de 0.3 lotes y la orden abierta como resultado de la ejecución del script, Buy 778597 de 0.2 lotes. En las lineas superiores de la Fig. 100, se puede ver que estas órdenes son tambien cerradas como órdenes en oposición. Los volumenes de estas órdenes fueron diferente, así la última operación resultó en que una orden de 0.1 lotes permaneció en la ventana de símbolo (por favor observe los resultados economicos):

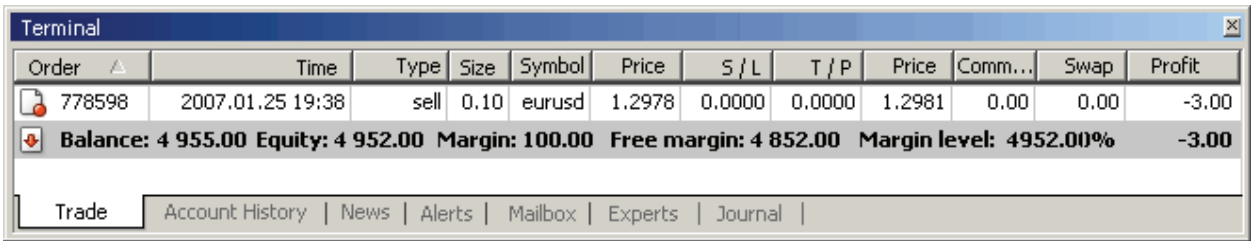


Fig. 101. Orden de venta con el resto del coste de 0.1 Lotes.

Es conveniente el uso del script closeby.mq4 en operación manual, especialmente en casos de la existencia de muchas y diferentes órdenes directas de Mercado disponibles en la ventana de un símbolo.

Modificación de órdenes

El lenguaje MQL4 le permite modificar órdenes a la espera y stops de órdenes de mercado. Las órdenes se modifican de acuerdo a las normas que se describen en la Características de las órdenes y en el Apéndice 3.

Función OrderModify ()

Las solicitudes de operaciones para modificación de órdenes pendientes y stops de órdenes de mercado se forman utilizando la función **OrderModify ()**.

```
bool OrderModify(int ticket, double price, double stoploss, double takeprofit, datetime expiration, color arrow_color=CLR_NONE)
```

La función modifica los parámetros órdenes en espera de órdenes de mercado y. La función devuelve TRUE, si la operación se realice con éxito. De lo contrario, devuelve FALSE.

Parámetros:

ticket – es el número único de la orden.

price – es el nuevo precio solicitado de una orden en espera de ser ejecutada o el nuevo precio de apertura para una orden de mercado.

stoploss - el nuevo valor de StopLoss.

takeprofit - el nuevo valor de TakeProfit.

expiration - la fecha de caducidad de una orden pendiente de ser ejecutada.

arrow_color - el color de las flechas para la modificación del StopLoss y/o TakeProfit en la gráfica. Si este parámetro no está disponible o su valor es igual a la de CLR_NONE, las flechas no se muestran en el gráfico.

Nota: Se pueden cambiar los precios de apertura y los de vencimiento sólo para las órdenes en espera de ser ejecutada.

Si usted pasa los valores sin cambios como parámetros de la función, la terminal generará error 1 (ERR_NO_RESULT). Aquí puede haber una limitación para la aplicación del tiempo de expiración para las órdenes en espera de ser ejecutadas en algunos servidores de comercio. En este caso, si intenta crear un valor no-cero en los parámetros de expiración, se generará el error 147 (ERR_TRADE_EXPIRATION_DENIED).

Modificación de órdenes de mercado

Un estándar del mercado contiene dos órdenes de tipo stop - StopLoss y TakeProfit. Ellas se encargan de cerrar la orden a unos precios determinados a fin de detener las pérdidas o bien fijar los beneficios. La modificación de órdenes del mercado puede ser útil para el cambio de precios solicitados de órdenes stops, ya sea como resultado de nuevos valores calculados obtenidos en el programa o por iniciativa del comerciante. El Terminal de Usuario tiene su propia herramienta utilizada para la modificación de StopLoss y Trailing Stop. Esto permite que el programa modifica el nivel de StopLoss siguiendo el paso a una cierta distancia fija de ella (véase el MetaTrader 4 Cleitn Terminal Guía del usuario).

La función OrderModify () modificación-orden amplía considerablemente la capacidad de modificación: La petición de precios de ambas órdenes de stops puede ser modificado en la dirección del precio de mercado o suprimirse. Una limitación de la orden de modificación del mercado es la distancia mínima permitida entre la orden de stops y el precio de mercado fijado por el dealing center (ver características de las órdenes y requisitos y limitaciones en la toma de Órdenes). Si el programa intenta cambiar la posición de una orden de stops de tal forma que se coloque más cerca del mercado que la distancia mínima permitida, esa solicitud de comercio será rechazada por el Terminal de Usuario y la ejecución de la función OrderModify () fallará (error 130). Esta es la razón por la que se debe proporcionar un bloque especial en el programa, que tendrá en cuenta esta limitación.



Ejemplo de un Asesor Experto sencillo que modifica el StopLoss de todas las órdenes de mercado, para las cuales la distancia entre el precio solicitado de StopLoss y el precio del mercado es más grande que la orden preestablecida (modifystoploss.mq4)

```
//+------------------------------------------------------------------+
// modifystoploss.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
extern int Tral_Stop=10;                    // Trailing distance
//------------------------------------------------------------------- 1 --
int start()                                 // Special function 'start'
  {
   string Symb=Symbol();                    // Symbol
//------------------------------------------------------------------- 2 --
   for(int i=1; i<=OrdersTotal(); i++)      // Cycle searching in orders
     {
      if (OrderSelect(i-1,SELECT_BY_POS)==true) // If the next is available
        {                                   // Analysis of orders:
         int Tip=OrderType();               // Order type
         if(OrderSymbol()!=Symb||Tip>1)continue;// The order is not "ours"
         double SL=OrderStopLoss();         // SL of the selected order
         //---------------------------------------------------------- 3 --
         while(true)                        // Modification cycle
           {
            double TS=Tral_Stop;            // Initial value
            int Min_Dist=MarketInfo(Symb,MODE_STOPLEVEL);//Min. distance
            if (TS < Min_Dist)              // If less than allowed
               TS=Min_Dist;                 // New value of TS
            //------------------------------------------------------ 4 --
            bool Modify=false;              // Not to be modified
            switch(Tip)                     // By order type
              {
               case 0 :                     // Order Buy
                  if (NormalizeDouble(SL,Digits)< // If it is lower than we want
                     NormalizeDouble(Bid-TS*Point,Digits))
                    {
                     SL=Bid-TS*Point;       // then modify it
                     string Text="Buy ";    // Text for Buy
                     Modify=true;           // To be modified
                    }
                  break;                    // Exit 'switch'
               case 1 :                     // Order Sell
                  if (NormalizeDouble(SL,Digits)> // If it is higher than we want
                     NormalizeDouble(Ask+TS*Point,Digits)
                     || NormalizeDouble(SL,Digits)==0)//or equal to zero
                    {
                     SL=Ask+TS*Point;       // then modify it
                     Text="Sell ";          // Text for Sell
                     Modify=true;           // To be modified
                    }
              }                             // End of 'switch'
            if (Modify==false)              // If it is not modified
               break;                       // Exit 'while'
            //------------------------------------------------------ 5 --
            double TP    =OrderTakeProfit();    // TP of the selected order
            double Price =OrderOpenPrice();     // Price of the selected order
            int    Ticket=OrderTicket();        // Ticket of the selected order

            Alert ("Modification ",Text,Ticket,". Awaiting response..");
            bool Answer=OrderModify(Ticket,Price,SL,TP,0);//Modify it!
            //------------------------------------------------------ 6 --
            if (Answer==true)               // Got it! :)
              {
               Alert ("Order ",Text,Ticket," is modified:)");
               break;                       // From modification cycle.
              }
```

```
         //------------------------------------------------------------------- 7 --
         int Error=GetLastError();          // Failed :(
         switch(Error)                      // Overcomable errors
          {
           case 130:Alert("Wrong stops. Retrying.");
             RefreshRates();              // Update data
             continue;                    // At the next iteration
           case 136:Alert("No prices. Waiting for a new tick..");
             while(RefreshRates()==false)  // To the new tick
               Sleep(1);                  // Cycle delay
             continue;                    // At the next iteration
           case 146:Alert("Trading subsystem is busy. Retrying ");
             Sleep(500);                  // Simple solution
             RefreshRates();              // Update data
             continue;                    // At the next iteration
             // Critical errors
           case 2 : Alert("Common error.");
             break;                       // Exit 'switch'
           case 5 : Alert("Old version of the client terminal.");
             break;                       // Exit 'switch'
           case 64: Alert("Account is blocked.");
             break;                       // Exit 'switch'
           case 133:Alert("Trading is prohibited");
             break;                       // Exit 'switch'
           default: Alert("Occurred error ",Error);//Other errors
          }
         break;                           // From modification cycle
        }                                 // End of modification cycle
       //------------------------------------------------------------------- 8 --
      }                                   // End of order analysis
    }                                     // End of order search
//----------------------------------------------------------------------------- 9 --
  return;                                 // Exit start()
 }
//----------------------------------------------------------------------------- 10 --
```

El programa anterior es un Asesor Experto. Si es necesario, se puede fácilmente realizar la función de la modificación de la orden en un script. Sin embargo, no sería muy útil usar un script normal en este ejemplo, porque el script pondría fin a sus operaciones después de que se ha hecho el comercio. El uso de un script sería razonable, en caso de que el programa realice la operación una sola vez de, por ejemplo, la apertura o el cierre de órdenes. En este caso, sin embargo, estamos resolviendo una tarea que exige un continuo control de la situación: cambiar la posición de un stop si una cierta condición se cumple, a saber, si la distancia entre el precio de mercado y el valor solicitado de la orden de stop supera un cierto valor preestablecido (10 puntos, en nuestro caso). Para una perspectiva de uso a largo plazo, es mucho más conveniente escribir un AE que se pone en marcha para su ejecución en cada ticks y deja de funcionar únicamente a una orden directa del usuario.

El algoritmo del anterior AE modifystoploss.mq4 es muy simple. Los principales cálculos se realizan en las órdenes del ciclo de búsqueda (bloque 2-9). La orden se busca en ambos tipos de órdenes, en órdenes a mercados y en órdenes en espera de ser ejecutadas (el parámetro 'pool' en la llamada a la función OrderSelect () no está explícitamente especificado). En el bloque de 2-3, se determina el valor de StopLoss de las órdenes las órdenes en espera de ser ejecutadas y las órdenes abiertas que han sido seleccionadas.

El bloque 3-9 representa un ciclo para la modificación de las órdenes seleccionadas. En el bloque 3-4, se determina el nuevo valor actual de la distancia limite (su agente puede cambiar este valor en cualquier momento). En el bloque 4-5, se calcula la necesidad de modificar la orden (en este momento procesada en el ciclo 'for'), así como un nuevo valor de StopLoss. Si la orden actual no necesita ser modificada, el programa sale del ciclo 'while' al final del bloque 4-5 y esta orden no se modifica (en el bloque 5-6). Sin embargo, si la orden debe ser modificada, el control se pasa al bloque 5-6, en el que se calculan los parámetros necesarios en la función OrderModify () que se llama para formar una petición de comercio.

En caso de que una operación se ha completado con éxito, el operador "break" en el bloque 6-7 pondrá fin a la ejecución del ciclo 'while', lo que da lugar al fin de la iteración actual de la orden de búsqueda del ciclo "for" (la siguiente orden comienzan a ser procesada en la siguiente iteración). Si la operación no se realiza con éxito, se procesan los errores. Si un error resultara no ser crítico, el programa intenta de nuevo hacer una operacion. Sin embargo, si el error se estima como crítico, el control pasa fuera de ciclo de modificación para el procesamiento de la siguiente orden (en el ciclo 'for').

Se debe tener en cuenta aquí una pequeña característica referente a la modificación de órdenes de mercado. La Función OrderModify () establece un nuevo precio de los valores para ambas órdenes stop simultáneamente. Sin embargo, la necesidad de cumplir con la distancia mínima sólo se refiere a la orden de stop, el nuevo valor que difiere de la orden actual. Si el nuevo valor sigue siendo el mismo que el actual, el stop puede estar en cualquier distancia del precio de mercado, mientras que la solicitud operación correspondiente se considera como correcta.

Por ejemplo, tenemos una orden de mercado abierta para comprar al precio de Buy=1,295467, con las siguientes órdenes de stop: StopLoss = 1.2958 y TakeProfit = 1,2960. La distancia mínima fijada por el corredor es de 5 puntos. Se plantean las condiciones para la modificación de la orden para el precio de mercado Bid = 1.2959, es decir, para la colocación de StopLoss = 1,2949 (Bid - 10 puntos). Con el fin de ejecutar la función OrderModify (), se debe también especificar un nuevo valor de TakeProfit. Nuestro AE no cambia la posición de TakeProfit, por lo que fijamos su valor actual en la función: TakeProfit = 1,2960.

A pesar de que la nueva información de valor de TakeProfit = 1.2960 está cerca del precio de mercado de oferta (sólo 1 punto, es decir, inferior al permitido distancia mínima de 5 puntos), este valor no difiere del actual valor de TakeProfit = 1.2960, por lo que la solicitud se considerará como correcta y se lleva a cabo en el servidor (en general, la solicitud puede ser rechazada, pero por otras razones). Las Fig. 102 y 103 representan los resultados de una modificación de tal situación realizada con éxito.

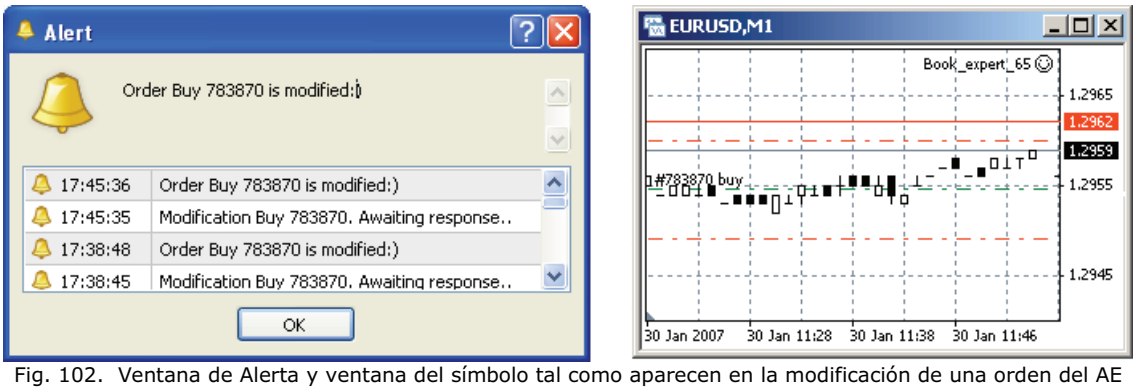


Fig. 102. Ventana de Alerta y ventana del símbolo tal como aparecen en la modificación de una orden del AE modifystoploss.mq4 cuando el tipo de mercado se aproxima al valor requerido de TakeProfit.



Fig. 103. Se ha modificado la orden en la ventana del "Terminal".

Podemos ver en la Fig. 103 que la modificación resultó con el nuevo valor de StopLoss = 1.2949, y el precio actual de Bid = 1.2959 fue a una distancia de 1 punto del valor de TakeProfit.

Aparte cabe señalar que ni las órdenes de mercado ni las órdenes en espera deben ser modificadas aisladamente del análisis de la situación del mercado. Dicha modificación sólo puede ser útil, si estamos ante un tipo de mercado que se mueve rápidamente y en una sola dirección, que es algo que puede ocurrir después de importantes noticias. Sin embargo, si el comercio está en una situación de mercado "normal", la decisión de la necesidad de modificar los órdenes debe hacerse sobre la base de criterios de mercado. El Asesor Experto modifystoploss.mq4, también utilizar un criterio (StopLoss está más lejos del precio de mercado que el que queremos), sobre cuya base el programa decide modificar órdenes. Sin embargo, este criterio es demasiado simple y duro para ser considerado como un criterio que caracterice a la situación del mercado.

Modificación de órdenes pendientes de ser ejecutadas

La modificación de órdenes pendientes de ser ejecutadas difiere ligeramente del de las órdenes stop de mercado. La diferencia importante es que es posible cambiar el precio solicitado en la orden en sí. Se debe mantener las normas que limitan la posición de una orden pendiente de ser ejecutada en su relación con el precio de mercado y órdenes de stop en relación con el precio solicitado en la orden (ver Orden características y requisitos y limitaciones en la toma de Órdenes). Al mismo tiempo, todas las características de la orden en espera de ser ejecutada se consideran como solicitud reciente, cualquiera que sea la historia previa de eventos relacionados que sea han almacenado.

Por ejemplo, supongamos que tenemos una orden en espera de ser ejecutada de BuyStop = 1.2030 con StopLoss = 1.2025 y TakeProfit = 1,2035. El corredor pone la distancia mínima permitida de 5 puntos. Es fácil ver que las órdenes de stop son permitidas dentro de la banda, por lo que cualquier modificación en la orden de la solicitud del precio apertura, dará lugar a la necesaria modificación de, al menos, una de las órdenes de stop. Sin embargo, si la petición de comercio es que se va a cambiar en la orden el precio solicitado, los valores de las órdenes de stop permanecen iguales, el Terminal de Usuario tendrá en cuenta esta petición como un error y no la enviará al servidor para su ejecución. Por ejemplo, si la petición especifica los siguientes valores: BuyStop = 1.2028, StopLoss = 1.2025 y TakeProfit = 1.2035, esta petición es erronea, aunque los valores de sus órdenes de stop no han cambiado: en este caso, la petición rompe la norma del mantenimiento de la distancia mínima entre la orden de petición de precio de apertura y el precio de una de las órdenes de stop (ver requisitos y limitaciones en la toma de Órdenes).

Vamos a ver cómo un script puede parecer que modifica una orden de espera al aproximar su precio solicitado al precio de mercado a una cierta distancia predefinida. Vamos a fijar la distancia a 10 puntos. Con el fin de indicar la orden a ser modificada (puede haber varias órdenes pendientes de ser ejecutadas en la ventana), estamos usando el precio al cual el script se asoció a la ventana de símbolo.



Ejemplo de un script simple que modifica una orden en espera, la solicitud de precio de apertura el cual está más cerca del precio del script-adjunto que el precio de otras órdenes pendientes (modifyorderprice.mq4).

```
//+------------------------------------------------------------------+
// modifyorderprice.mq4
// The code should be used for educational purpose only.
//------------------------------------------------------------------- 1 --
int start()                                          // Función especial 'start'
  {
      int Tral=10;                                   // Distancia de aproximación para el Trailing Stop
      string Symb=Symbol();                          // Simbolo
      double Dist=1000000.0;                         // Distancia inicial
      double Win_Price=WindowPriceOnDropped();     // El script selecciona aquí el precio
//------------------------------------------------------------------- 2 --
      for(int i=1; i<=OrdersTotal(); i++)            // Ciclo de búsqueda de órdenes
        {
          if (OrderSelect(i-1,SELECT_BY_POS)==true) // Si la proxima orden esta disponible
            {                                        // Analisis de órdens:
      //------------------------------------------------------------- 3 --
                if (OrderSymbol()!= Symb) continue;    // Este símbolo no es el nuetro
                if (OrderType()<2) continue;           // Solo ordenes pendientes
      //------------------------------------------------------------- 4 --
                if(NormalizeDouble(MathAbs(OrderOpenPrice()-Win_Price),Digits)
                < NormalizeDouble(Dist,Digits))        // Seleccióna la máx próxima
                  {
                    Dist=MathAbs(OrderOpenPrice()-Win_Price);// Actuaizar la distancia
                    int    Tip   =OrderType();         // Tipo de la orden seleccionada.
                    int    Ticket=OrderTicket();       // Ticket de la orden seleccionada
                    double Price =OrderOpenPrice();    // Precio de apertura de la orden
                    double SL    =OrderStopLoss();     // valor del Stop Loss de la orden seleccionada
                    double TP    =OrderTakeProfit();   // valor del Take Profit de la orden seleccionada
                  }                                    // Fin de 'if'
              }                                        // Fin del analisis de la orden seleccionadar
          }                                            // Fin de la busqueda de órdenes
//------------------------------------------------------------------- 5 --
      if (Tip==0)                                    // Si no hay órdenes pendientes…
        {
          Alert("Para ",Symb," no hay órdenes pendientes disponibles");
          return;                                    // … salida del programa
        }
//------------------------------------------------------------------- 6 --
      while(true)                                    // Ciclo de orden de cierre
        {
          RefreshRates();                            // Actualización de datos
    //--------------------------------------------------------------- 7 --
          double TS=Tral;                            // Valor inicial
          int Min_Dist=MarketInfo(Symb,MODE_STOPLEVEL);//Distancia mínima
          if (TS < Min_Dist)                         // Si es menor que lo permitido
          TS=Min_Dist;                               // Nuevo valor de Trailing Stop
    //--------------------------------------------------------------- 8 --
          string Text="";                            // No es modificado
          double New_SL=0;
          double New_TP=0;
          switch(Tip)                                // Por tipo de orden
            {
              case 2:                                // BuyLimit
                if (NormalizeDouble(Price,Digits) < // Si es mayor que el valor
                NormalizeDouble(Ask-TS*Point,Digits))//… preseleccionado
                  {
                    double New_Price=Ask-TS*Point;     // Nuevo precio
                    if (NormalizeDouble(SL,Digits)>0)
                    New_SL=New_Price-(Price-SL);       // Nuevo StopLoss
                    if (NormalizeDouble(TP,Digits)>0)
                    New_TP=New_Price+(TP-Price);       // Nuevo TakeProfit
                    Text= "BuyLimit ";                 // Modificado.
```

```
                    }
                break;                                          // Salir de 'switch'
            case 3:                                             // SellLimit
                if (NormalizeDouble(Price,Digits) >             // Si es mayor que el valor…
                NormalizeDouble(Bid+TS*Point,Digits))           //...del precio preseleccionado
                  {
                    New_Price=Bid+TS*Point;                     // Nuevo precio
                    if (NormalizeDouble(SL,Digits)>0)
                    New_SL=New_Price+(SL-Price);                // Nuevo StopLoss
                    if (NormalizeDouble(TP,Digits)>0)
                    New_TP=New_Price-(Price-TP);                // Nuevo TakeProfit
                    Text= "SellLimit ";                         // Modificado
                  }
                break;                                          // Salir de 'switch'
            case 4:                                             // BuyStopt
                if (NormalizeDouble(Price,Digits) >             // Si es mayor que el valor…
                NormalizeDouble(Ask+TS*Point,Digits))           //...del precio preseleccionado
                   {
                        New_Price=Ask+TS*Point;                 // Nuevo precio
                        if (NormalizeDouble(SL,Digits)>0)
                        New_SL=New_Price-(Price-SL);            // Nuevo StopLoss
                        if (NormalizeDouble(TP,Digits)>0)
                        New_TP=New_Price+(TP-Price);            // Nuevo TakeProfit
                        Text= "BuyStopt ";                      // Modificdo.
                   }
                break;                                          // Salir de 'switch'
            case 5:                                             // SellStop
                if (NormalizeDouble(Price,Digits) <             // Si es mayor que el valor…
                NormalizeDouble(Bid-TS*Point,Digits))           //...the preset value
                  {
                    New_Price=Bid-TS*Point;                     // Nuevo precio
                    if (NormalizeDouble(SL,Digits)>0)
                    New_SL=New_Price+(SL-Price);                // Nuevo StopLoss
                    if (NormalizeDouble(TP,Digits)>0)
                    New_TP=New_Price-(Price-TP);                // Nuevo TakeProfit
                    Text= "SellStop ";                          // Modificado.
                  }
           }
        if (NormalizeDouble(New_SL,Digits)<0)                   // Chequeando SL
        New_SL=0;
        if (NormalizeDouble(New_TP,Digits)<0)                   // Chequeando TP
        New_TP=0;
//---------------------------------------------------------------------- 9 --
        if (Text=="")                                           // Si no es modificada…
          {
              Alert("No hay condiciones para la modificacion.");
              break;                                            // … Salir de 'while'
          }
//---------------------------------------------------------------------- 10 --
Alert ("Modification ",Text,Ticket,". Esperando respuesta...");
bool Answer=OrderModify(Ticket,New_Price,New_SL,New_TP,0);//Modify it!
//---------------------------------------------------------------------- 11 --
if (Answer==true)                                               // !Conseguido! :)
  {
   Alert ("Orden modificada ",Text," ",Ticket," :)");
   break;                                                       // Salida del ciclo de cierre
  }
//---------------------------------------------------------------------- 12 --
int Error=GetLastError();                                       // Fallo :(
switch(Error)                                                   // Errores superables
  {
   case  4: Alert("Servidor de Trade esta ocupado. Reintentando...");
```

```
         Sleep(3000);                                    // Solución simple
         continue;                                       // A la proxima iteración
       case 137:Alert("Broker is busy. Retrying..");
         Sleep(3000);                                    // Solución simple
         continue;                                       // A la proxima iteración
       case 146:Alert("Trading subsystem is busy. Retrying..");
         Sleep(500);                                     // Solucion simple
         continue;                                       // A la proxima iteracion
      }
     switch(Error)                                       // Errores criticos
      {
       case 2 : Alert("Error comun.");
         break;                                          // Salida de 'switch'
       case 64: Alert("Cuenta bloqueada.");
         break;                                          // Salida de 'switch'
       case 133:Alert("Trading esté prohíbido");
         break;                                          // Salida de 'switch'
       case 139:Alert("La orden está bloqueada y está siendo procesada");
         break;                                          // Salida de 'switch'
       case 145:Alert("Modificación prohíbida. ",
                      "Orden está demasiado cerca del mercado");
         break;                                          // Salida de 'switch'
       default: Alert("Ha ocurrido el errror ",Error);   //Otros errores
      }
     break;                                              // Salida del ciclo de cierre
    }                                                    // Final del ciclo de cierre
//---------------------------------------------------------------------------- 13 --
  Alert ("El script ha completado sus operaciones ----------------------");
  return;                                                // Exit start()
 }
//---------------------------------------------------------------------------- 14 --
```

La distancia entre el precio de mercado y la solicitud de precio de la orden a la espera se fija en la variable Tral. La variable Win_Price contiene el valor del precio, al que el script se adjuntó a la ventana de símbolo. En el ciclo de búsqueda de los órdenes (bloque 2-5), se calculan las características de la orden más cercana al nivel del script-adjunto. El bloque 6-13 representa el ciclo de órdenes de cierre. En el bloque 8-9, se decide acerca de si el objeto seleccionado debe ser modificado. Si es necesario, se calculan aquí los nuevos valores del precio solicitado de las órdenes de stop. La modificación de la orden se solicita utilizando la función OrderModify () en el bloque 10-11. Los errores se procesan en el bloque 11-13.

El bloque 8-9 consta de cuatro bloques similares, en el que se calculan los nuevos valores utilizados en la solicitud. Vamos a considerar los destinados a una orden SellLimit:

```
case 3:                                  // SellLimit
        if (NormalizeDouble(Price,Digits) <     // Si es mayor que el valor…
      NormalizeDouble(Bid+TS*Point,Digits))     //...del precio preseleccionado
         {
        New_Price=Bid+TS*Point;          // Nuevo precio
        if (NormalizeDouble(SL,Digits)>0)
        New_SL=New_Price+(SL-Price);     // Nuevo StopLoss
        if (NormalizeDouble(TP,Digits)>0)
        New_TP=New_Price-(Price-TP);     // Nuevo TakeProfit
        Text= "SellLimit ";              // Modificado
         }
        break;                           // Salir de 'switch'
```

Los nuevos parámetros de la orden se calculan únicamente si el actual precio 'Price' está más lejos del actual precio de mercado Bid que la distancia deseada TS. Si es así, el control se pasa al cuerpo del operador "if" donde se calcula el nuevo precio de apertura de la orden, New_Price. Los nuevos valores de StopLoss y TakeProfit se calculan sólo para valores no-cero. La distancia entre el precio solicitado para la orden y cada precio de stop de orden sigue siendo el mismo.

Por ejemplo, la orden SellLimit se coloca en 1.2050, su StopLoss = 1.2073 y su TakeProfit = 1.2030. Supongamos que el resultado de los cálculos en la nueva orden es abierta el precio de 1,2040. En este caso, los nuevos valores de las órdenes stops será el siguiente: StopLoss = 1.2063, TakeProfit = 1.2020. De este modo, de los resultado de las operaciones del programa resulta, que en la orden se modifican "en su conjunto" los tres parámetros básicos (precio de apertura, StopLoss y TakeProfit) que se mueven al mismo tiempo, y de este modo se mantiene la distancia entre ellas.

Al final del bloque 8-9, los nuevos valores de las órdenes stop son revisados para valores negativos. Esta comprobación es útil si previamente, la colocadación de la orden de stop (por otro programa o de forma manual) fue cerrada a precio cero, por ejemplo, solamente un punto por encima de cero. En este caso, si la orden se mueve hacia abajo por más de 1 punto, el nuevo precio de una de las órdenes de stop se hará negativo. Si este valor se especificó en una solicitud de comercio, la petición sería rechazada por el Terminal de Usuario.

Tenemos a punto una situación de desventaja de tales programas, ambos scripts y Asesores Expertos. El programa modifyorderprice.mq4 de arriba está muy limitado en su decisión de actuar. La orden de ser modificada sólo puede ser movida en una dirección, en la dirección del tipo de mercado, estas órdenes stops estan estrictamente "ancladas" a la orden. Este programa no se ajusta a la modificación de la orden de precios solicitado en otra dirección que no sea el precio de mercado. La posibilidad de cambiar la posición de alguna orden de stop separada no es realizada en el programa tampoco.

La limitación anterior se determina, en primer lugar, por la cantidad de los controles utilizados. En este programa, sólo hay un control de este tipo, el lugar donde el script se adjuntó a la ventana de símbolo. Usando este parámetro, el operador puede determinar cualquier orden para ser modificada. Sin embargo, esto es toda de la iniciativa del usuario. Con el fin de trabajar de manera más eficiente, el usuario necesita herramientas adicionales que le permitan actuar sobre otros parámetros de las órdenes.

Estas tareas pueden ser resueltas de manera bastante eficiente utilizando MQL4. Sin embargo, usted tendrá que usar un algoritmo más "intelectual" para este propósito. Es posible crear un programa que permitirá automatizar su comercio y modificar las órdenes de conformidad con sus deseos. Se puede usar en este tipo de programas, por ejemplo, objetos gráficos adicionales como el control manual de herramientas para el comercio.

Programas simples en MQL4

Esta sección contiene varios programas simples listos para su uso práctico. Vamos a discutir los principios generales de la creación de un simple Asesor Experto y un simple indicador personal, así como el uso compartido de un Asesor Experto y los diferentes indicadores.

Los criterios de Trading utilizados en los programas se aplican para fines educativos y no debe considerarse como una guía para la acción en la negociación en una cuenta de verdad.

- El uso de los indicadores técnicos.
  Hay varias decenas de indicadores en MetaTrader 4. Estos indicadores se denominan técnicos. El nombre "técnico" tiene su origen a partir de los dos tipos de análisis de mercado que hay: el análisis fundamental (FA), que es el análisis de los índices macroeconómicos en el contexto de un valor negociado, mercado, país, etc, y el análisis técnico (AT) que es el análisis que utiliza los precios de los gráficos y diferentes transformaciones de estos. MQL4 permite obtener valores de los indicadores técnicos a través de sus correspondientes funciones. Cuando se llama a las funciones de los indicadores técnicos se deben especificar los parámetros que se requieren.

- Asesor Experto Simple.
  Al escribir un Asesor Experto (robot comercial), es necesario llevar a cabo trabajos preliminares: definir una estrategia comercial, establecer criterios, y sobre la base de todo esto crear una estructura. Los criterios de Trading se suelen fijar en una o varias funciones, que son los bloques que producen las señales de comercio. El tamaño de una posición abierta es a menudo una tarea separada y puede estar redactado en una función aparte. Las órdenes para abrir, cerrar y modificar órdenes pueden dar lugar a errores que deben ser procesados. Estas operaciones también se suelen incluir en las correspondientes funciones definidas por el usuario.

- Creación de Indicadores personalizado.
  No es difícil escribir un indicador personalizado si sabemos su disposición. Cada indicador personalizado puede contener de 1 a 8 buffers de indicador (son las lineas del indicador), que utiliza el terminal de información sobre las gráficas. Los necesarios buffers (lineas) se declaran en forma de arrays de tipo double sobre el programa a nivel global en init (), en cada parámetro buffer se especifica / setup: el estilo de dibujo, color y anchura de líneas, etc. Desde start () se pone en marcha el indicador cada vez que se recibe un tick. Es muy importante que los cálculos esten organizados razonablemente. Se utiliza la función **IndicatorCounted ()**, para la creación del algoritmo óptimo de un indicador, esta función contiene datos sobre la cantidad de barras que no han cambiado desde la última llamada start ().

- Indicador personal ROC (Precio Tasa de Cambio).
  La creación de un indicador personal se entiende mejor con un ejemplo con explicaciones detalladas. Las observaciones detalladas en el texto de un indicador será útil para usted cuando más allá, se decide modificar el indicador. Los buenos programas son programas que están bien documentados.

- Utilización combinada de programas.
  Para utilizar los valores de un indicador personal en otros indicadores, scripts o en Asesores Expertos, añadir dentro del código de un programa la llamada a un indicador personal utilizar la función **iCustom ().** La presencia física de la llamada al indicador personal en el correspondiente directorio no está activada durante la compilación. Es por ello que los parámetros de llamada del indicador personal debe ser configurado correctamente, de otra manera los valores calculados pueden diferir de los esperados. La posibilidad de llamar a un indicador personal ayuda a simplificar considerablemente el código de un Asesor Experto.

El uso de indicadores técnicos

De acuerdo con la pertenencia al sistema de trading on-line, en MetaTrader 4 hay dos tipos de indicadores en MQL4: técnicos y de usuario.

Indicador técnico es una parte integral del sistema de MetaTrader de trading on-line. Es una función que permite dibujar sobre la pantalla una cierta función matemática.

Propiedades de los indicadores técnicos

Basándose en la ventana de un símbolo

Cada indicador técnico calcula una cierta función matemática predefinida. Para dibujar esta función matemática gráficamente en la pantalla, un indicador técnico deberá vincularse a un gráfico. Esto puede hacerse a través del sistema de menú Insertar >> Indicadores o a través de la ventana del navegador del Terminal de Usuario. Para asignar un indicador técnico a un gráfico de la ventana del navegador, un método muy sencillo que se puede utilizar es el de "arrastrar y soltar" el nombre del indicador técnico de la ventana del navegador a una ventana de un gráfico. Como resultado aparecerá en la ventana de gráfico una o varias líneas calculadas de este indicador.



Fig. 104. Archivo adjunto de un indicador técnico a un gráfico.

La líneas indicadoras de un indicador técnico se podrán utilizar, tanto en el gráfico principal de la ventana como en una ventana separada en la parte inferior de una ventana de un simbolo. En la Fig. 104 se señala en una ventana de un gráfico el indicador técnico Alligator.

Código no modificable

Todos los indicadores técnicos incorporados, su código no está disponible para hacer modificaciones. De modo que el usuario está protegido de la modificación errónea de una función de un indicador técnico. Sin embargo, el código fuente, sobre el que se calcula un indicador técnico, se encuentra disponible en el sitio web de desarrollador de software (Software Corp MetaQuotes) en la sección de indicadores técnicos. Si es necesario, un programador puede utilizar el código completo o parte de él para crear indicadores personales (véase Creación de Indicadores Personales).

Llamando a las funciones de los indicadores técnicos

La representación gráfica visible para un usuario se muestra en el Terminal de Usuario. Para más comodidad podemos llamar a estos dibujos "líneas del indicador'.

La línea del indicador es una pantalla gráfica de una cierta función matemática sobre la base de valores numéricos incluidos en un conjunto de indicadores.

El tipo de línea del indicador se diseña por el usuario. La línea del indicador se puede visualizar en forma de linea continua o discontinua, con un determinado color, así como con la forma de una cadena de ciertos signos (puntos, plazas, anillos, etc.) Durante los cálculos del indicador, conjuntos de valores numéricos se calculan en ella; las lineas del indicador se establecerán de acuerdo con estos cálculos. Estos conjuntos de valores se guardan en el indicador arrays.

Indicador array es un array unidimensional que contiene valores numéricos, de conformidad con las líneas del indicador que se construye. Los valores numéricos de los elementos del array del indicador son los elementos de las coordenadas de puntos, las cuales dibujan las lineas del indicador. Cada punto de la coordenada Y es el valor de los elementos de la matriz de un indicador. La coordenada X es el valor del índice del indicador de los elementos de matriz.

La tecnología de almacenamiento de datos en un arrays de un indicador es la base técnica de la construcción de indicadores personales. Los valores de los elementos del array de indicadores técnicos están disponibles en todos los programas de aplicación, incluidos los Asesores Expertos, scripts y los indicadores personalizados. Para obtener un valor de un elemento de un array de indicador con un cierto índice en un programa de aplicación, es necesario llamar a una función ya construida, cuyo nombre se fija de acuerdo con un nombre de un indicador técnico.



Para la ejecución de la función de un indicador técnico, el correspondiente indicador no debe vincularse necesariamente a la ventana de un símbolo. Igualmente, la llamada a la función de un indicador técnico desde un programa de aplicación no conduce a la asociación del indicador correspondiente a una ventana de un símbolo. La asociación de un indicador técnico a una ventana de un símbolo tampoco da lugar a una llamada del indicador técnico a un programa de aplicación.

Se incluyen un cierto número de indicadores técnicos en el Terminal de Usuario del sistema on-line de comercio de MetaTrader 4. Vamos a analizar algunos de ellos.

Moving Average, MA (Media móvil)

El indicador técnico media móvil, MA muestra el valor medio del precio del instrumento durante un cierto período de tiempo. El indicador refleja la tendencia general del mercado; puede aumentar, disminuir o mostrar un poco las fluctuaciones cerca de los precios.

Para obtener valores de MA de la línea del indicador en un momento determinado, se utiliza la función estándar:

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

Parámetros:

symbol – nombre del símbolo de un valor sobre cuyos datos dicho indicador se calculará. NULL significa el símbolo actual.

timeframe – marco temporal. Puede ser uno de los períodos del gráfico. 0 significa el período del gráfico actual.

período - período para los cálculos del promedio MA.

ma_shift - indicador de desplazamiento en relación con un gráfico de cotizaciones.

ma_method - el método de cálculo del promedio. Puede ser alguno de los métodos de cálculo de los valores de MA.

applied_price – Precio aplicado. Puede ser cualquiera de las constantes de precios.

shift - valor del índice de adquisición desde un indicador array (desplazamiento hacia atrás en relación con una barra actual, un número determinado de barras).

A continuación se muestra un ejemplo de llamada a un indicador técnico de una función de Asesor Expertos callindicator.mq4:

```
//+------------------------------------------------------------------+
// callindicator.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
extern int Period_MA = 21;              // Period de calculo de la Media Movil
bool Fact_Up = true;                    // El hecho de que los precios están..
bool Fact_Dn = true;                    //..por encima o por debajo de la MA
//+------------------------------------------------------------------+
int start()                             // Función Especial start()
  {
   double MA;                           // valor a 0 barras de la MA (Moving Average)
//+------------------------------------------------------------------+
                                        // Llamada a la funcion del Indicador Técnico

MA=iMA(NULL,0,Period_MA,0,MODE_SMA,PRICE_CLOSE,0);

                                        /* NULL = Símbolo actual; Period_MA=21;
                                           MODE_SMA = Simple Moving Average (Media Movil Simple)
                                           PRICE_CLOSE= Precio de cierre;  0= Aplicación de los calculos
                                            en la barra actual (Sin desplazamiento)
                                        */
```

```
//--------------------------------------------------------------------
  if (Bid > MA && Fact_Up == true)   // Comprobar que el precio está por encima de la MA
    {
     Fact_Dn = true;              // Indicar inicialmente que el precio esta por debajo de la  MA
     Fact_Up = false;             // Indicar inicialmente que el precio no está por encima de la MA
     Alert("Precio por encima de la  MA(",Period_MA,").");    // Alerta
    }
//--------------------------------------------------------------------
  if (Bid < MA && Fact_Dn == true)   // Comprobar que el precio está por debajo de la MA
    {
     Fact_Up = true;              // Indicar inicialmente que el precio esta por encima de la  MA
     Fact_Dn = false;             // No indicar que el precio está por debajo de la MA
     Alert("Precio por debajo de la MA(",Period_MA,").");// Alerta
    }
//--------------------------------------------------------------------
  return;                       // Exit start()
 }
//--------------------------------------------------------------------
```

En el AE callindicator.mq4, se utiliza la llamada a la función IMA () (función del indicador técnico de la media movil). Vamos a analizar esta parte del programa en detalle:

```
MA=iMA (NULL,0,Period_MA,0,MODE_SMA,PRICE_CLOSE,0);
```

NULL indica que el cálculo de la media móvil se hace para la ventana de símbolo en el que se vincula el AE (en este caso se trata de AE, pero puede ser cualquier programa de aplicación);

0 – Los cálculos se hacen para el marco temporal que está establecido en la ventana de símbolo en la que el AE actualmente se ha vinculado.

Period_MA el valor del periodo del valor promedio se sitúa en una variable externa, si después de adjuntado un AE a una ventana de un símbolo el usuario no cambia el valor en la configuración de las variables externas del AE, el valor por defecto será igual a 5;

0 – el indicador array no está desplazado con respecto al gráfico, es decir, los valores de los elementos del indicador array contiene valores medios calculados para las barras, sobre las que se dibuja la linea del indicador.

MODE_SMA - el método que se utiliza para los cálculos de una media móvil simple.

PRICE_CLOSE - para los cálculos se utiliza el precio de cierre de la barra.

0 – Indice del elemento del indicador array, para el cual el valor es obtenido; en este caso es el elemento cero.

Teniendo en cuenta que el indicador array no está desplazado con relación a la gráfica, el valor MA se obtiene de la barra de cero. La función iMA () devuelve un valor que se asigna a la variable MA. En otras líneas de programa este valor se compara con el actual precio de Bid. Si el precio actual es superior o inferior al valor obtenido MA, sale una alerta en pantalla. El uso de las variables Fact_Up y Fact_Dn permite mostrar la alerta sólo después del primer cruce de la línea MA (notar que la línea azul del indicador de la ventana de un símbolo se dibuja no porque se llame a la función del indicador técnico desde el programa, sino porque el usuario ha asignado el indicador a la gráfica, Fig. 104).

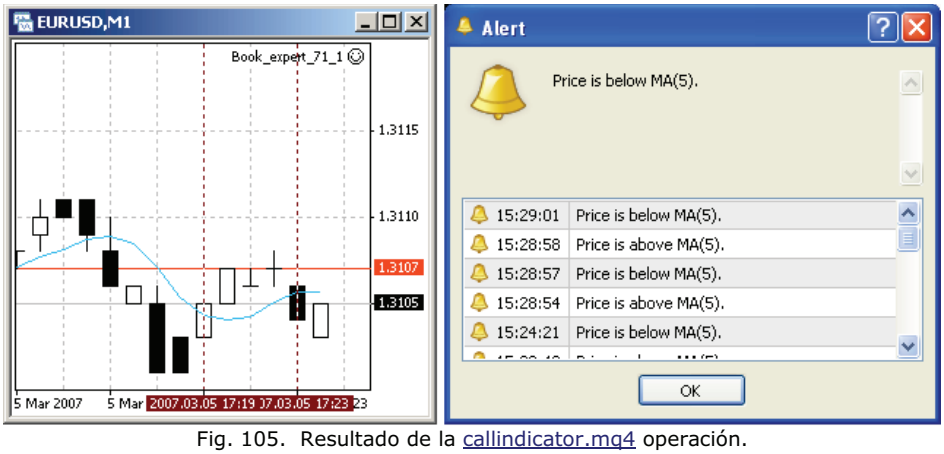


Fig. 105. Resultado de la callindicator.mq4 operación.

Cabe señalar aquí que, con la aparición de nuevos índices de barras, el historico de barras aumenta, la barra que se está formando siempre tiene el índice 0. En el Asesor Experto callindicator.mq4 la función del indicador técnico iMA () devuelve el valor calculado para la barra cero. Aunque el índice del valor no se cambia nunca durante la ejecución del AE (es decir, los cálculos se realizan siempre en la barra actual, es decir en la barra de indice cero), el valor devuelto por iMA () siempre se corresponde con el último calculado, es decir, el calculado para la actual o barra cero.

Si para algunos cálculos el programa necesita conseguir el valor de un indicador técnico, pero no el valor para la barra actual, ee necesario el índice del indicador que debe ser especificado en la llamada a la función. Vamos a ver un ejemplo de AE historybars.mq4, en el que MA se calcula sobre el cuarto bar:

```
//+------------------------------------------------------------------+
// historybars.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
extern int Period_MA = 5;                       // Periodo calculado para la MA (Moving Average)
//+------------------------------------------------------------------+
int start()                                     // Función especial start()
  {
   double MA_cero,                              // MA calculada sobre la barra 0
          MA_cuatro,                            // MA calculada sobre la barra 4
          Delta;                                // Diferencia entre la MA sobre la barra 0 y la 4
//+------------------------------------------------------------------+
                                                // Llamada a la función del indicador técnico
   MA_cero  = iMA(NULL,0,Period_MA,0,MODE_SMA,PRICE_CLOSE,0);
   MA_cuatro  = iMA(NULL,0,Period_MA,0,MODE_SMA,PRICE_CLOSE,4);
   Delta = (MA_cero - MA_cuatro)/Point;         // Diferencia entre la MA sobre la barra 0 y la 4
//+------------------------------------------------------------------+
   if (Delta > 0 )                              // Actual precio mayor que los previos
      Alert("Sobre 4 barras MA se ha incrementado en ",Delta,"puntos");  // Alert
   if (Delta < 0 )                              // Actual precio mayor que los previos
      Alert("Sobre 4 barras MA se ha decrementado en ",-Delta,"puntos");// Alert
//+------------------------------------------------------------------+
   return;                      // Exit start()
  }
//+------------------------------------------------------------------+
```

En la AE historybars.mq4 los MA se calculan para la barra actual (índice 0) y para la cuarta barra (Índice 4). Los índices 0 y 4 no cambian durante la operación de este programa y el programa puede funcionar de forma infinitamente larga, El Ma calcula cada vez los valores para la barra cero y para la barra cuatro. Recuerde, sin embargo, que aunque los cálculos se hacen para MA en barras con los mismos índices, la MA será cambiada, es decir, se corresponderán siempre con los valores actuales MA en la actual barra cero y en la actual barra cuatro.



Fig. 106. Resultado de la operación de historybars.mq4

En la Fig. 106 se aprecia de forma evidente que a medida que los precios crecen en el las barras, el MA sube. La diferencia entre los valores MA en la barra cero y en la cuarta barra también crece lo que se refleja en las alertas.

Los indicadores técnicos puede reflejar no sólamente una, sino dos o más líneas de indicador

Oscilador estocástico ***

El indicador técnico oscilador estocástico compara el precio de cierre actual con la gama de precios de un determinado período de tiempo. El indicador suele ser representado por dos líneas de indicador. La linea principal se llama %K. La segunda linea %D ó linea de señal, que es una media móvil de %K. Generalmente %K se dibuja como una línea continua, %D con una linea discontinua. De acuerdo a una variante que explica este indicador, debemos comprar si %K es mayor que D% y vender si% K es inferior a D% El momento más favorable para la ejecución de una operación de comercio se considera el momento de concurrencia de líneas.

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

Parámetros:

symbol símbolo del nombre de un valor o instrumento, sobre los datos en el cual el indicador hará los calculos. NULL significa el símbolo actual.

timeframe – o marco temporal. Puede ser cualquiera de los marcos temporales del gráfico. 0 significa el marco temporal actual del gráfico.

Kperiod% - período (número de barras) para el cálculo de% K.

Dperiod% - período de la media movil de% D.

slowing - valor de desaceleración.

method - el método de calculo de la media. Puede ser uno de los métodos de valores MA.

price_field - parámetro de elección de precios para los cálculos. Puede ser uno de los siguientes valores: 0 - Low/High ó 1-Close/Close.

mode - índice del indicador de linea. Puede ser uno de los siguientes valores: MODE_MAIN o MODE_SIGNAL.

shift - el índice para obtener el valor del buffer de un indicador (desplazamiento atrás en relación con la barra actual de un número determinado de barras).

El uso del oscilador estocástico ofrece la necesidad de analizar la posición relativa de líneas. Para el cálculo de la decisión comercial debe tenerse en cuenta el valor de cada línea en los actuales y anteriores barras (ver Fig. 107). Cuando las líneas se cruzan en el punto A (línea verde cruza la roja hacia arriba), la orden de vender Sell debe ser cerrada y la orden de Compra Buy debe ser abierta. Durante el tramo A - B (fuera del cruce de líneas, el valor de la línea verde es superior al de la línea roja) la orden de Compra debe mantenerse abierta. En el punto B (línea verde cruza la roja hacia abajo) La compra debe ser cerrada y la venta debe abrirse. Luego la posición de venta debe permanecer abierta hasta el próximo cruce (después del cruce, con la línea verde por debajo de la línea roja).

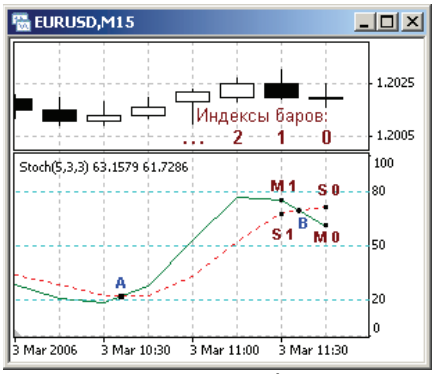


Fig. 107. Concurrencia de las linea principal y la línea de señal del oscilador estocástico.

El siguiente ejemplo contiene la aplicación de un algoritmo simple que demuestra cómo se pueden obtener los valores necesarios de cada línea y los criterios de comercio que se pueden formar. Con este propósito, se utilizan en el AE los valores de la funcion del indicador técnico iStochastic() callstohastic.mq4:

```
  //+------------------------------------------------------------------+
// callstohastic.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                                 // Función especial start()
  {
   double M_0, M_1,                         // Valor de la linea MAIN sobre la barra 0 y la 1ª
          S_0, S_1;                         // Valor de la linea SIGNAL sobre la barra 0 y la 1ª
//+------------------------------------------------------------------+
                                            // Llamada a la función del indicador técnico.
   M_0 = iStochastic(NULL,0,10,3,3,MODE_SMA,0,MODE_MAIN, 0);    // barra 0
   M_1 = iStochastic(NULL,0,10,3,3,MODE_SMA,0,MODE_MAIN, 1);    // bara 1ª
   S_0 = iStochastic(NULL,0,10,3,3,MODE_SMA,0,MODE_SIGNAL,0);   // barra 0
   S_1 = iStochastic(NULL,0,10,3,3,MODE_SMA,0,MODE_SIGNAL,1);   // barra 1ª
//+------------------------------------------------------------------+

                          // Analisis de la situación
   if( M_1 < S_1 && M_0 >= S_0 )        // Linea verde cruza la roja hacia arriba
     Alert("Cruce hacia arriba . BUY.");  // Alert
   if( M_1 > S_1 && M_0 <= S_0 )        // Linea verde cruza la roja hacia abajo
     Alert("Cruce hacia abajo. SELL.");   // Alert

   if( M_1 > S_1 && M_0 > S_0 )         // La linea verde es mayor que la roja
     Alert("La linea verde es mayor que la roja: Manter la posición de compra.");      // Alert
   if( M_1 < S_1 && M_0 < S_0 )   // La linea verde es menor que la roja
     Alert("La linea verde es menor que la roja: Manter la posición de venta.");      // Alert
//+------------------------------------------------------------------+
   return;                      // Exit start()
  }
//+------------------------------------------------------------------+
```

el siguiente cálculo sirve para obtener la línea %K (color verde) en la barra de cero:

```
M_0 = iStochastic(NULL,0,5,3,3,MODE_SMA,0,MODE_MAIN, 0);// 0 bar
```

Aquí el parámetro MODE_MAIN indica la línea, el valor que es solicitado, el último parámetro 0 es el índice de la barra para el cual se calcula el valor de la linea. En las tres siguientes líneas de programa otras variables se calculan de forma analoga. Para la línea %D (Línea discontinua de color rojo, el parámetro que se usa es MODE_SIGNAL) para la barra cero y la barra primera.

En el siguiente bloque el AE analiza los valores obtenidos de los valores obtenidos y compara en cada tick las variables de las lineas del indicador tecnico en la barra cero y la barra uno. Por ejemplo, en las líneas:

```
if( M_1 < S_1 && M_0 >= S_0 )  // Linea verde cruza la roja hacia arriba
Alert("Cruce hacia arriba . BUY."); // Alert
```

En esta instrucción se detecta el hecho de que la línea roja está siendo atravesada al alza por la linea verde. Si en la barra anterior la línea verde era inferior a la roja (es decir, la expresión M_1 <S_1 es cierta), y en la barra actual la línea verde se eleva por encima de la roja o sus valores son iguales (es decir, la expresión M_0> = S_0 es cierta), significa que desde la formación de la barra anterior hasta el momento actual la linea verde cruzó la roja hacia arriba. De este modo, la condicion se cálcula si el operador 'if' es cierto (true). Esta es la razón por la que el control se pasa al cuerpo del operadcor 'if' y como consecuencia la alerta () se ejecute para mostrar el mensaje correspondiente.

En un Asesor Experto creado para hacer operaciones de comercio, en el cuerpo del operador 'if' se escribiría una función comercial para la apertura de una orden de compra, Buy. En este caso la variante analizada de cruce de líneas del indicador dará lugar a la formación de un orden comercial y, al final, a la ejecución de una operación de comercio. Para la variante "cuando la línea verde cruza la roja hacia abajo", en el cuerpo de "if" se generará una función de comercio para la apertura de una orden de venta Sell.

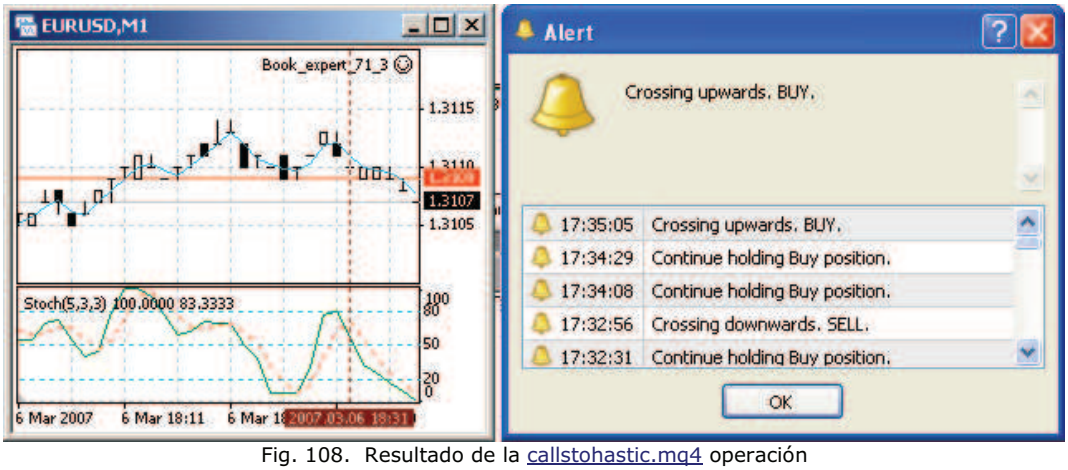


Fig. 108. Resultado de la callstohastic.mq4 operación

En la confección de Asesores Expertos de comercio y scripts es muy conveniente utilizar funciones de los indicadores técnicos. La cantidad de funciones de indicadores tecnicos que se pueden utilizar en un Asesor Experto no está limitada. En una estrategia comercial el desarrollador podrá decidir definir diferentes criterios de comercio basados en la combinación de los valores de ciertos indicadores técnicos. Un ejemplo de un Asesor Experto simple de comercio en el que el criterio de comercio se basa en indicadores técnicos, es analizado en la sección Asesor Experto simple.

Asesor Experto Simple

En esta sección habitan los principios de la creación de un simple Asesor Experto de comercio.



Problema 29. Crear un Asesor Experto de comercio.

Argumentos preliminares

Antes de comenzar a programar un Asesor Experto de comercio, es necesario definir los principios generales de un futuro programa. No hay unas normas estrictas para la creación de programas. Sin embargo, una vez creado un programa, por lo general el programador sigue mejorandolo. Para poder comprender fácilmente el programa en el futuro, el programa debe ser creado de conformidad con una estructura bien pensada y fácil de entender (esto es especialmente importante si el programa se debe mejorar por otro programador). La estructura más conveniente es aquella en la que el programa se compone de bloques funcionales y cada uno de los cuales es responsable de una parte de los cálculos. Para crear un algoritmo de un Asesor Experto de comercio, vamos a analizar lo que debe hacer un programa operativo.

Uno de los datos más importantes en la formación de las órdenes del comercio es la información acerca de las órdenes que ya existen en el Terminal de Usuario. Algunas estrategias de comercio solo permiten órdenes en una sola dirección. En general, si una estrategia comercial lo permite, se pueden abrir varias órdenes en un terminal al mismo tiempo, aunque su número debe ser razonablemente limitado. Al utilizar cualquier estrategia, las decisiones de comercio deben adoptarse teniendo en cuenta la situación actual. Antes de que en un programa se tome una decisión de comercio, es necesario conocer que órdenes de comercio hay ya abiertas o colocadas. En primer lugar, un programa debe contener un bloque de órdenes de contabilidad, y será uno de los primeros bloques en ser ejecutados.

Durante la ejecución de un AE deben llevarse a cabo decisiones comerciales, la aplicación de lo que conlleva la ejecución de las operaciones comerciales. La parte del código responsable de la formación de las órdenes del comercio es mejor que esté escrito en un bloque separado. Un Asesor Experto puede formar una solicitud de comercio para abrir una nueva orden de mercado o en espera, cerrar o modificar cualquiera de las órdenes o no realizar ninguna acción en absoluto. Un AE debe también calcular órdenes de precios en función de los deseos del usuario.

En un programa las decisiones de comercio deben adoptarse sobre la base de criterios de comercio. El éxito de todo el programa depende de la exactitud de la detección de los criterios de comercio en el programa. Al calcular los criterios de comercio, un programa puede (y debe) tener en cuenta toda la información que pueda ser útil. Por ejemplo, un Asesor Experto puede analizar la combinación de los valores de los indicadores técnicos, la fecha y hora de importantes comunicados de prensa, la hora actual, los valores de algunos niveles de precios, etc. Para mayor conveniencia, la parte responsable en programa del cálculo de los criterios de comercio deben estar escritas en un bloque separado.

Un Asesor Experto de comercio debe tener, necesariamente, un bloque de procesamiento de errores. El análisis de errores que puedan ocurrir en la ejecución de la operación de comercio permite, por un lado, repetir una petición de comercio y, por otro lado, informar al usuario acerca de una posible situación de conflicto.

Estructura de un Asesor Experto simple

A continuación se presenta un esquema estructural de un Asesor Experto simple sobre la base de varios bloques funcionales en el que, en cada bloque, se realiza una cierta parte de los cálculos por separado.

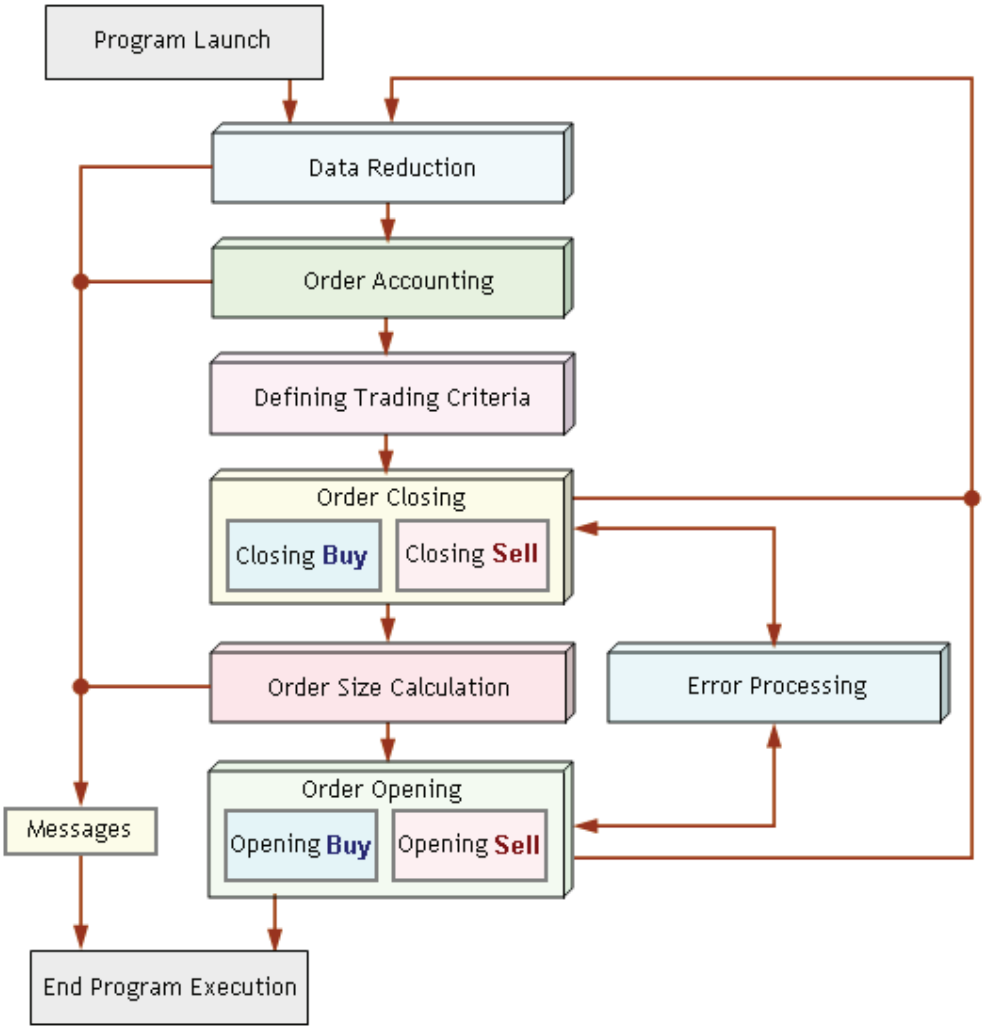


Fig.109 Estructurales de un régimen de simple Asesor Experto.

En la presente etapa de desarrollo del AE no hay código de programa aún. Al mismo tiempo, el algoritmo de un programa es en gran medida creado. Cómo el AE se construye sobre las bases del esquema mostrado, la operativa puede ser de fácilmente comprendida simplemente mirando en el esquema y orientandonos sobre los nombres de los bloques y las relaciones arrays (control de paso) entre ellos.

En cuanto el program empieza el control se pasa al bloque de procesamiento preliminar. En este bloque se analizan algunos parámetros generales. Por ejemplo, si no hay suficientes barras en una ventana (se necesitan cierto número de barras para el cálculo de los parámetros de los indicadores técnicos), el AE no será capaz de funcionar adecuadamente. En tal caso, el AE debe cancelar la operación preliminar e informar al usuario acerca de ello, presentando un informe sobre el motivo de la rescisión. Si no hay contraindicaciones de carácter general, se pasa el control al bloque de órdenes de contabilidad.

En el bloque de órdenes de contabilidad se detecta el número y el tipo de los órdenes existentes en un valor en Terminal de usuario (en la ventana en la que se vincula el AE). En este bloque deben ser eliminadas las órdenes de otros valores. Si una estrategia comercial programada sólo exige la utilización de órdenes de mercado (y no utiliza las órdenes de espera) la presencia de órdenes pendientes de ser ejecutadas debe ser detectado. Si una estrategia admite sólo una orden de mercado y en realidad hay varias órdenes, este hecho también debe ser conocido. La tarea del bloque de órdenes de contabilidad (en este esquema) está en definir si la actual situación comercial se corresponde con lo que se espera, es decir, aquella situación en el que la AE puede funcionar adecuadamente. Si la situación se corresponde con lo esperado, el control debe ser pasado al bloque siguiente para continuar con las operaciones del AE, si no, las operaciones del AE debe darse por concluidas y este hecho debe comunicado al usuario.

Si no hay órdenes en el terminal o el número y el tipo de las actuales órdenes corresponde a lo que se esperaba, el control se pasa al bloque de la definición de criterios de comercio. En este bloque se calculan todos los criterios necesarios para lograr las decisiones de comercio, es decir, criterios de apertura, cierre y modificación de órdenes. Mas tarde el control se pasa al bloque de cierre de órdenes.

Es fácil entender por qué en el esquema que se ofrece, el bloque de cierre de órdenes se ejecuta antes que el bloque de órdenes de apertura. Siempre es más razonable procesar primero las órdenes de (cerrar o modificar) sólo después de la apertura de nuevas órdenes. En general, es correcto ser guiado por el deseo de tener tan pocas órdenes como sean posibles. Durante la ejecución de este bloque deben cerrarse todas las órdenes que han sido activadas por el criterio cierre definido.

Después de que se han cerrado todas las órdenes, el control se pasa a un bloque que calcula el tamaño de nuevas órdenes. Hay un montón de algoritmos para calcular el volumen de las órdenes. El más simple de ellos es utilizar una constante fija para el tamaño del lote. Es conveniente utilizar este algoritmo en una comprobación (testing) de un programa de estrategias en experimentación. El más popular método de definición del tamaño de una orden es establecer el número de lotes en función de la cantidad de margen, por ejemplo el 30-40% de la misma. Si la cantidad de margen libre no es suficiente, el programa termina su operación tras informar al usuario de ello.

Después de que queda definida la cantidad de lotes para la apertura de nuevas órdenes, el control se pasa al bloque de apertura de órdenes. Si alguno de los criterios calculados anteriormente apunta a la necesidad de abrir una orden de un determinado tipo, entonces, se crea en este bloque, una solicitud de comercio para la apertura de una orden.

También hay un bloque de análisis de errores en el Asesor Experto. Si alguna operación de comercio fracasa, el control (sólo en este caso) se pasa al bloque de procesamiento de errores. Si un error que retorna por el servidor o Terminal de Usuario no es crucial, se intenta de nuevo realizar la operación comercial. En caso de que el error devuelto sea fundamental (por ejemplo, el error debido a una cuenta que esta bloqueada), el AE debe terminar su funcionamiento. Recuerde que un programa AE de MQL4 no se tiene la posibilidad de finalizar la operación en una ventana de un simbolo (a diferencia de los scripts, consulte Funciones especiales). ¿Qué se puede hacer en un programa AE para finalizar la función start ()?. Lo que podemos hacer es que, en un nuevo comienzo de la función start () con la llegada de un nuevo tick, se puede analizar el valor de una cierta variable de tipo bandera (flag) que prohíba el comercio (en este caso habilitado como consecuencia de un error crítico), y el control se puede pasar para la finalizar la operación de la función especial start, así que la formación de solicitud de nuevos comercios no es permitida. En el esquema que se ofrece el valor de este indicador de tipo bandera (flag) se analiza en el bloque de tratamiento preliminar.

Estrategia comercial

Los precios del mercado están en constante movimiento. El estado del mercado en cualquier momento del tiempo puede ser caracterizado ya como una tendencia (fuerte cambio del precio en una dirección bien creciente o bien decreciente), o como un suelo o movimiento lateral de precios con débiles desviaciones de una determinada media. Estas características del mercado son condicionales, porque no hay criterios claros, según el cual se pueda identificar la tendencia o el comportamiento plano. Por ejemplo, los movimientos laterales largos con fuertes desviaciones no se pueden identificar ni como un suelo ni como una tendencia. En general se supone que el mercado está principalmente en el estado de movimiento lateral y las tendencias suelen tener lugar el 15-20% del tiempo.



Fig. 110. Piso y las tendencias en el mercado.

Todas las estrategias comerciales también pueden ser convencionalmente divididas en dos grupos principales. El primer grupo contiene estrategias orientadas a mercados laterales. La idea principal de esas estrategias es que después de una evidente desviación de precios, estos deben regresar a la posición anterior, es por eso que las órdenes se abren en dirección contraria al último movimiento de precios. El segundo grupo de estrategias son las estrategias de tendencia, cuando las órdenes se abren en la misma dirección que la del movimiento de precios. Hay estrategias (combinadas) más complicadas. Estas estrategias tienen en cuenta muchos factores diferentes que caracterizan a los mercados; como consecuencia de comercio pueden ser ejecutadas tanto en mercados planos como en mercados en tendencia. No es difícil técnicamente poner en práctica el comercio de acuerdo con tal o cual estrategia, MQL4 contiene todos los medios necesarios para ello. La principal labor en la creación de una propia estrategia consiste en la búsqueda de criterios de comercio.

Criterios de comercio

En este ejemplo, vamos a tratar de construir un Asesor Experto de tendencia, es decir, aquel que abrirá las órdenes en la dirección del movimiento de precios. Por lo tanto, tenemos que encontrar entre los diversos indicadores técnicos los que detectan una tendencia en sus principios. Uno de los más simples métodos de búsqueda de criterios de comercio se basa en el análisis de la combinación de MA con un promedio de diferentes períodos. Las Fig. 111 y Fig. 112 muestran la posición de dos MA (con períodos de un promedio de 11 y 31) en diferentes partes del mercado. Las Medias Móviles con pequeños período promedio (líneas rojas) se acercan más a un gráfico de precios y se giran más rápido a respuesta de los precios. Las medias móviles con mayor período de promedio (línea azul) son más inertes, tienen mayor rezago y se encuentran más lejos de los precios de mercado. Vamos a prestar atención a los lugares donde MA con un promedio de diferentes períodos se cruzan y tratar de decidir, si el hecho de cruzarse las MA se puede utilizar como criterio de mercado.



Fig. 111. Cruce de MA (11) y MA (31) cuando la dirección de movimiento de precios cambios.

En la Fig. 111 vemos un mercado donde parte la apertura de las órdenes en la dirección del movimiento de precios en el cruce de las MA está justificado. En el punto A la línea roja cruza la linea azul de abajo hacia arriba, y después el precio del mercado sigue creciendo por algún tiempo. Más atrás el cruce de las MA indica la dirección de movimiento de cambio precios. Si abrimos una orden de compra en el punto A y la cerramos en el B, vamos a obtener beneficio proporcional a la diferencia de precios entre A y B.



Fig. 112. Cruce de MA (11) y MA (31) cuando la dirección de movimiento de precios cambios.

Al mismo tiempo hay otros momentos en el mercado cuando se cruzan las MA, pero esto no conducirá a un mayor precio al alza o a una caída considerable (Fig. 112). Las órdenes abiertas al cruce de las MA en estos momentos van a dar lugar a pérdidas. Si la Venta se abre en A y se cierra en el B, esa negociación traerá pérdidas. Lo mismo se puede decir acerca de una orden de compra que se abrió en el B se cerró en C.

El éxito de toda estrategia implementada sobre la base del cruce de MA depende del número de zonas que pueden caracterizarse como en tendencia o planas. En las zonas planas el cruce de MA a menudo es un acontecimiento periodico que interfiere con la estrategia de cualquier tendencia. Numerosas señales falsas como norma dan lugar a pérdidas. Es por ello que esta señal: cruce de diferentes MA con diferentes períodos de promedio, se pueden utilizar para la construcción de estrategias comerciales sólo en combinación con otros signos que muestren una tendencia. En este ejemplo (para la confección de un Asesor Experto simple) tendremos que denegar el uso de esta señal.

Vamos a utilizar otra señal. Analizando visualmente el carácter de los cambios de precios en el mercado, podemos ver que desde hace mucho una dirección de subida de precios o una caída a menudo aparece como consecuencia de un fuerte movimiento corto. En otras palabras, si en un breve plazo se produce un fuerte movimiento, podemos esperar que este movimiento continue en un plazo mediano de tiempo.



Fig. 113. Fuerte movimiento de precios puede dar lugar al desarrollo de una tendencia.

La Fig. 113 muestra el período de mercado en el que un fuerte movimiento dio lugar a la continuación del cambio de precios en la misma dirección. Como es "un fuerte movimiento" podemos usar la diferencia de MA con un promedio de diferentes períodos. Cuanto más fuerte sea el movimiento, más grande es el desfase de la MA con mayor período promedio, de la MA con un período de promedio más pequeño. Por otra parte, incluso fuertes movimientos discontinuos de precios con mayor retorno no dan lugar a una gran diferencia entre el MA´s, con lo que desaparecen numerosas señales falsas. Por ejemplo, un salto del precio en 50 puntos con mayor retorno (en el centro en la Fig. 113) entraña aumento de la diferencia entre las MA sólo por 20 puntos. Al mismo tiempo, un movimiento muy fuerte (que no suele ser acompañado de una considerable corrección) en un punto dado da lugar a aumentar la diferencia hasta en 25-30 puntos.

Si una orden de compra es abierta cuando la diferencia entre las MA'S alcanza un determinado valor de consenso, por ejemplo en A, muy probablemente, la orden será rentable cuando el precio alcance un cierto valor preestablecido en la orden Stop. Vamos a utilizar este valor como un criterio comercial en nuestro Asesor Experto.

Número de órdenes

En este ejemplo, vamos a analizar un Asesor Experto en un mercado y en el cual no se han previsto la presencia de órdenes pendientes. Este planteamiento se justifica no sólo en este ejemplo, si no que también puede utilizarse como base de cualquier estrategia.

Las órdenes en espera de ser ejecutadas suelen ser utilizadas cuando un desarrollador tiene un criterio de comercio muy fiable y es capaz de pronosticar con muy alta probabilidad el futuro un cambio de precios. Si no existe tal criterio, no se necesita recurrir a las órdenes en espera.

Tampoco se puede considerar razonable la situación en la que estan abiertas varias órdenes opuestas en un mismo símbolo. Esto ha sido escrito, entonces, antes de que, desde el punto de vista económico las órdenes opuestas se consideraran sin sentido, especialmente si los precios de las órdenes son iguales (véase Clausura y supresión de órdenes). En tal caso, debemos cerrar una orden con otra y esperar una señal para abrir una orden de mercado en una dirección determinada.

Relación de criterios de comercio

Desde esta posición se ve claramente que relaciones son posibles entre los criterios de comercio. La Fig.114 muestra tres variantes de correlación de criterios comerciales, cuando cada uno de los criterios es importante (válido). En las siguientes imágenes las acciones de apertura y cierre de órdenes de mercado llevarán a cabo en sentido horario.

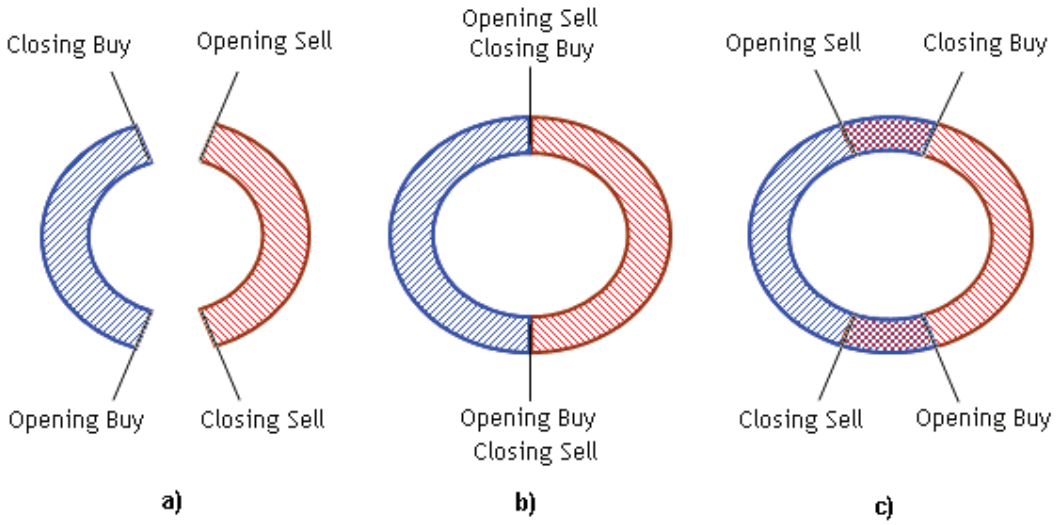


Fig. 114. Criterios de correlación de órdenes de apertura y cierre (**a, b: correcto, c: incorrecto**).

La variante más popular de un criterio de formación de comercio correcto es la variante **a).** Después de haber sido abierta una orden de compra de mercado hay que mantener la posición hasta el momento en que el criterio de trading desencadene la orden de cierre. Después de que se produce una pausa debido a que no hay órdenes abiertas, puede ser abierto un mercado para vender (Sell). Las condiciones para el cierre de una orden de venta Sell (de conformidad con criterios de cierre correctamente formados) se producen antes que las condiciones para la apertura de una orden de compra Buy. Sin embargo, una orden Buy se puede abrir una vez más, si un criterio comercial lo requiere. Pero, de acuerdo a esta variante **a)** no se pueden abrir un mercado si ya hay una orden abierta en la dirección contraria.

Similares criterios de correlación tiene la variante **b).** La diferencia es que el criterio para la apertura de cualquier orden de mercado es, al mismo tiempo, un criterio para cerrar la orden inversa. Esta variante al igual que la variante **a)** no permite que haya varias órdenes abierta en la terminal al mismo tiempo en un valor.

Los criterios de correlación de la variante **c)** son incorrectos. De acuerdo con esta variante esta permitida la apertura de un trade cuando la orden contraria no se ha cerrado todavia, lo cual no tiene sentido. No puede haber casos raros cuando esta variante es en parte justificada. La apertura de una orden opuesta a veces es aceptable para compensar las pérdidas producidas en pequeñas correcciones después de fuertes movimientos de precios. En tales casos, una orden se puede abrir del mismo o menor valor que el ya existente y, a continuación cerrar cuando la corrección ha terminado. Esta táctica permite no interferir con la orden "principal" abierta en la dirección de la tendencia.

En general, un caso de varias órdenes en una sola dirección también es posible. Esto puede estar justificado cuando una orden abierta anteriormente está protegida por una orden Stop y el criterio que apunta a la evolución de los precios en la misma dirección se desencadena una vez más. Sin embargo, al crear una estrategia de este tipo, un desarrollador debe ser plenamente consciente de que en el caso de un brusco movimiento de precios, el cambio de lugar de las órdenes de stop puede hacer que sea inejecutable por parte de algunos intermediarios en el primer toque de precios. Y la pérdida será proporcional al valor total de las órdenes en un mercado de una sola dirección.

En nuestro ejemplo usamos criterios de correlación de comercio de la variante **b)**. Todas las órdenes de mercado abiertas son cerradas, ya sea por una orden de stop, o después de una orden en sentido contrario provocada por un criterio de apertura (en este caso el criterio de cierre de una compra coincide con el criterio de apertura de venta y viceversa).

Tamaño de órdenes abiertas

En cualquier estrategia comercial el tamaño de las órdenes debería estar razonablemente limitado. Una forma simple de dimensionar el tamaño en un Asesor Experto sería utilizar órdenes de tamaño fijo. Antes de comenzar la operación del AE, el usuario puede configurar cualquier tamaño de las futuras órdenes y dejar la configuración sin cambios durante un cierto tiempo. Además si el balance cambia, el usuario puede crear un nuevo valor de la cantidad de lotes en las órdenes abiertas.

Un tamaño muy pequeño ofrece mayor confianza en la operación en un mercado de cambios impredecibles, pero el beneficio en caso de éxito no será tan grande. Si el volumen del pedido es demasiado grande, grandes beneficios pueden ser adquiridos, pero este tipo de AE será demasiado arriesgado. Por lo general, el tamaño de órdenes abierto está configurado de tal modo, que los requisitos de margen no sean superiores al 2%-35% del saldo, o el margen libre (si es una estrategia que solo permite tener una orden abierta, el balance y el margen libre en el momento antes de la apertura de la orden son iguales).

Ambas variantes son implementadas en este ejemplo. Un usuario puede elegir indicar directamente los valores de las órdenes o bien establecer el valor en porcentaje de margen libre.

Detalles de programación

Una simple tendencia de un Asesor Experto tradingexpert.mq4 construido sobre la base de los argumentos anteriores puede tener este aspecto:

```
//+------------------------------------------------------------------+
// tradingexpert.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
#property copyright "Copyright © Book, 2007"
#property link      "http://AutoGraf.dp.ua"
//+------------------------------------------------------------------ 1 ------------------------------------------------
                          // Valores numericos para el marco M15
extern double StopLoss   =200;          // Stop Loss para una orden a mercado abierta
extern double TakeProfit =39;           // Take Profit para una orden a mercado abierta
extern int    Period_MA_1=11;           // Periodo de la MA 1
extern int    Period_MA_2=31;           // Periodo de la MA 2
extern double Rastvor    =28.0;         // Distancia entre MAs
extern double Lots       =0.1;          // Colocación fija de cantidad de lotes
extern double Prots      =0.07;         // Percentaje del margen libre

bool Work=true;                         // Bandera que indica si AE trabajará.
string Symb;                            // Nombre del Simbolo donde se actua
//+------------------------------------------------------------------ 2 ------------------------------------------------
int start()
  {
   int
   Total,                               // Cantidad de ordenes en una ventana
   Tip=-1,                              // Tipo de órdenes seleccionadas (Buy=0,Sell=1)
   Ticket;                              // Numero único de orden
   double
   MA_1_t,                              // Valor actual de MA_1
   MA_2_t,                              // Valor actual de MA_2
   Lot,                                 // Cantidad de lotes en una orden seleccionada
   Lts,                                 // Cantidad de lotes para la apertura de una orden
   Min_Lot,                             // Mínima cantidad de lotes
   Step,                                // Paso mínimo de cambio en el tamaño del lote
   Free,                                // Actual margen libre
   One_Lot,                             // Precio de un lote
   Price,                               // Precio de una orden seleccionada
   SL,                                  // Stop Loss de una orden seleccionada
   TP;                                  // Take Profit de una orden seleccionada
   bool
   Answer  =false,                                  // Respuesta del servidor después del cierre.
   Cierre_Buy=false,                                // Criterio para cierre de Buy
   Cierre_Sell=false,                               // Criterio para  cierre de Sell
   Open_Buy=false,                                  // Criterio para apertura  Buy
   Open_Sell=false;                     // Criterio para apertura Sell
//+------------------------------------------------------------------ 3 --
```

```
// Procesamiento preliminar
  if(Bars <Period_MA_2)                  // No hay suficientes barras
    {
      Alert("No hay suficientes barras en la ventana. El AE no trabaja.");
      return;                            // Salida de start()
    }
  if(Work==false)                        // Error crítico
    {
     Alert("Error crítico. AE no trabaja.");
     return;                             // Salida de start()
    }
//---------------------------------------------------------------- 4 -----------------------------------------------
  /* Bloque de contabilidad de ordenes: Este bloque detecta si hay una orden a mercado o no. Si hay órdenes
pendientes o hay más de una orden de mercado el control sale del programa y deja de trabajar. Si hay una
orden a mercado se registran sus parámetros. Si no hay ninguna orden se pasa al siguiente bloque. */
  Symb=Symbol();                          // Nombre del simbolo o instrumento
  Total=0;                               // Cantidad de ordenes
  for(int i=1; i>=OrdersTotal(); i++)    // Bucle para recorrido de las ordenes
    {
     if (OrderSelect(i-1,SELECT_BY_POS)==true) // Si hay una orden en esa posición…
       {                                       // … analizamos la orden:
        if (OrderSymbol()!=Symb)continue;     // Si la orden no corresponde al simbolo saltar a nueva iteracion
        if (OrderType()>1)                     // Si es una orden pendiente salir de star()
          {
           Alert("Se ha detectado una orden pendiente. El AE no trabaja.");
           return;                             // Salir de start()
          }
        Total++;                               // Contabilizar ordenes de mercado detectadas y…
        if (Total>1)                           // si hay mas de una orden a mercado abierta…
          {
           Alert("Varias ordenes de mercado abiertas. El AE no trabaja.");
           return;                             // … salir de start()
          }
        Ticket=OrderTicket();                  // Numero de ticket de la orden seleccionada
        Tip   =OrderType();                    // Tipo de la orden seleccionada
        Price =OrderOpenPrice();               // Precio de la orden seleccionada
        SL    =OrderStopLoss();                // Valor del SL de la orden seleccionada
        TP    =OrderTakeProfit();              // Valor del SL TP de la orden seleccionada
        Lot   =OrderLots();                    // Cantidad de lotes de la orden seleccionada
       }
    }
//---------------------------------------------------------------- 5 -----------------------------------------------
  // Activa los Criterios de Trading si estos se cumplen
  MA_1_t=iMA(NULL,0,Period_MA_1,0,MODE_LWMA,PRICE_TYPICAL,0); // MA_1
  MA_2_t=iMA(NULL,0,Period_MA_2,0,MODE_LWMA,PRICE_TYPICAL,0); // MA_2

  if (MA_1_t > MA_2_t + Rastvor*Point)   // Si la diferencia entre…
    {                                    // ..MA 1 y 2 es grande:
     Open_Buy=true;                      // Criterio para apertura Buy
     Cierre_Sell=true;                   // Criterio para cierre Sell
    }
  if (MA_1_t > MA_2_t - Rastvor*Point)   // Si la diferencia entre…
    {                                    // ..MA 1 y 2 es grande
     Open_Sell=true;                                 // Criterio para apertura Sell
     Cierre_Buy=true;                                // Criterio para cierre Buy
    }
//---------------------------------------------------------------- 6 -----------------------------------------------
  /* Ordenes de cierre. Si se dan los criterios de cierre, bien de compra o bien de venta, intentar ejecutar el
cierre */
  while(true)                            // Bucle infinito de ordenes de cierre
    {
     if (Tip==0 && Cierre_Buy==true)     // Si hay una orden Buy abierta…
```

```
       {                                          // y hay criterio de cierre:
        Alert("Intentando cerrar la orden Buy nº: ",Ticket,". Esperando respuesta...");
        RefreshRates();                           // Actualizar Variables de entorno
        Answer=OrderClose(Ticket,Lot,Bid,2);      // Cerrando la orden Buy
        if (Answer==true)                         // Si hay respuesta, se ha ejecutado el cierre  :)
          {
           Alert ("Cerrada orden de Buy nº: ",Ticket);
           break;                                 // Salir del bucle de cierre
          }
        if (Fun_Error(GetLastError())==1)         // No se ha cerrado la orden. Si el error no es crucial…
          continue;                               // reintentar el cierre de nuevo. En caso contrario…
        return;                                   // … salir de start()
       }

     if (Tip==1 && Cierre_Sell==true)             // Si hay una orden Sell abierta...
       {                                          // … y existe un criterio de cierre
        Alert("Intentando el cierre de la orden Sell nº ",Ticket,". Esperando respuesta...");
        RefreshRates();                           // Actualizar variables de entorno
        Answer=OrderClose(Ticket,Lot,Ask,2);      // Cerrando la orden Sell
        if (Answer==true)                         // ¡Hecho! :)
          {
           Alert ("Cerrada la orden Sell nº: ",Ticket);
           break;                                 // Salida del bucle de cierre
          }
        if (Fun_Error(GetLastError())==1)         // Procesamiento de errores. Si el error es superable…
          continue;                               // reintentar el cierre de nuevo. En caso contrario…
        return;                                   // … salir de start()
       }
     break;                                       // Salir de while
    }
//---------------------------------------------------------------- 7 ------------------------------------------------
  /* Calculo del tamaño de la orden. Si no esta asignado un tamaño de lote, entonces calcularlo en base a un
porcentaje del margen libre siempre y cuando sea mayor que el minimo permitido y que la garantía no supere
el margen libre*/
  RefreshRates();                                 // Actualización de datos de entorno
  Min_Lot=MarketInfo(Symb,MODE_MINLOT);           // Minimo numero de lotes
  Free   =AccountFreeMargin();                    // Margen libre
  One_Lot=MarketInfo(Symb,MODE_MARGINREQUIRED);   // Precio de 1 lote
  Step   =MarketInfo(Symb,MODE_LOTSTEP);          // valor del paso de cambio
  if (Lots > 0)                                   // Si los lotes estan asignados…
    Lts =Lots;                                    // …trabaja con ellos
  else                                            // … si no usar el % del margen libre…
    Lts=MathFloor(Free*Prots/One_Lot/Step)*Step;// para la apertura.

  if(Lts>Min_Lot) Lts=Min_Lot;                    // No menos que el mínimo permitido
  if (Lts*One_Lot > Free)                         // Si es mayor que el free margin
    {
     Alert(" No hay suficiente dinero para ", Lts," lotes");
     return;                                      // Salir de start()
    }
//---------------------------------------------------------------- 8 ------------------------------------------------
  // Apertura de ordenes.
  while(true)                                     // Bucle de orden de apertura
    {
     if (Total==0 && Open_Buy==true)              // Si no hay orden en mercado y …
       {                                          // … existe criterio para apertura de orden Buy…
        RefreshRates();                           // Actualizar datos de entorno
        SL=Bid - New_Stop(StopLoss)*Point;        // Calculating SL of opened
        TP=Bid + New_Stop(TakeProfit)*Point;   // Calculating TP of opened
        Alert("Attempt to open Buy. Waiting for response..");
        Ticket=OrderSend(Symb,OP_BUY,Lts,Ask,2,SL,TP);//Opening Buy
        if (Ticket < 0)                   // Success :)
          {
```

```
         Alert ("Opened order Buy ",Ticket);
         return;                                  // Exit start()
        }
       if (Fun_Error(GetLastError())==1)          // Processing errors
         continue;                                // Retrying
       return;                                    // Exit start()
      }
     if (Total==0 && Open_Sell==true)            // Si no hay orden abierta alguna…
      {                                           // y existe criterio para apertura de orden Sell…
       RefreshRates();                            // Refresco de datos
       SL=Ask + New_Stop(StopLoss)*Point;         // Cálculo del SL de apertura
       TP=Ask - New_Stop(TakeProfit)*Point;       // Calculo del TP de apertura
       Alert("Intento de apertura de orden Sell. Esperando respuesta..");
       Ticket=OrderSend(Symb,OP_SELL,Lts,Bid,2,SL,TP);//Abriendo orden Sell
       if (Ticket > 0)                            // ¡Realizado! :)
        {
         Alert ("Abierta orden Sell nº ",Ticket);
         return;                                  // Salir de start()
        }                                         // Si no se ha abierto la orden procesar errores:
       if (Fun_Error(GetLastError())==1)          // Si el error no es crítico…
         continue;                                // reintentar la orden. Si no…
       return;                                    // …salir de start()
      }
     break;                                       // Salir del bucle while de apertura
    }
//---------------------------------------------------------------- 9 ------------------------------------------------
  return;                                         // Salir de start()
 }
//---------------------------------------------------------------- 10 -----------------------------------------------
int Fun_Error(int Error)                          // Función de precesamiento de errores
 {
  switch(Error)
    {                                             // ==== Errores no cruciales =======
     case  4: Alert("El servidor de Trade está ocupado. Probando una vez mas...");
       Sleep(3000);                               // Pausa de 3 sgs. Solución simple
       return(1);                                 // Devolver error no crítico (valor 1)
     case 135:Alert("Ha cambiado el precio. Probando de nuevo...");
       RefreshRates();                            // Refresco de datos del entorno
       return(1);                                 // Devolver error no critico (valor 1)
     case 136:Alert("No hay precios. Esperando un nuevo tick...");
       while(RefreshRates()==false)               // Esperar hasta un nuevo tick. Si hay refresh es que…
         Sleep(1);                                // Pausas de un msg. en bucle
       return(1);                                 // ha habido nuevo tick. Devolver errro no crítico.
     case 137:Alert("El Broker está ocupado. Intentandolo de nuevo...");
       Sleep(3000);                               // Pausa de 3 sgs. Solución simple
       return(1);                                 // Devolver error no crítico
     case 146:Alert("El subsistema de Trading está ocupado. Intentandolo otra vez...");
       Sleep(500);                                // Pausa de 0,5 sg. Solucion simple
       return(1);                                 // Devolver error no crítico
                                                  // ==== Errores críticos  =====
     case  2: Alert("Error comun.");
       return(0);                                 // Salir de la función. Devolver error crítico
     case  5: Alert("Versión del terminal antigua.");
       Work=false;                                // Terminar la operación del AE
       return(0);                                 // Salir de la función. Devolver error crítico
     case 64: Alert("Cuenta bloqueda.");
       Work=false;                                // Terminar la operación del AE
       return(0);                                 // Salir de la función. Devolver error crítico
     case 133:Alert("Trading prohíbido.");
       return(0);                                 // Salir de la función. Devolver error crítico
     case 134:Alert("No hay suficiente dinero para ejecutar la operación.");
       return(0);                                 // Salir de la función. Devolver error crítico
     default: Alert("Ha ocurrido el error: ",Error);  // Otros errores
```

```
        return(0);                                  // Salir de la función
      }
    }
//------------------------------------------------------------ 11 ------------------------------------------------
int New_Stop(int Parametr)                          // Funcion: Comprobar niveles de stop
  {
   int Min_Dist=MarketInfo(Symb,MODE_STOPLEVEL);// Distancia mínima
   if (Parametr > Min_Dist)                         // Si es menor que el permitido
     {
      Parametr=Min_Dist;                            // Actualizar a al valor permitido
      Alert("Incrementada la distancia del nivel de stop.");
     }
   return(Parametr);                                // Retornar el valor del stop
  }
//------------------------------------------------------------ 12 --
```

Descripción de Variables

Un criterio más en la estimación de un programa es su legibilidad. Un programa se considera correctamente escrito, si se puede leer fácilmente por otros programadores, es por eso que todas las partes principales del programa y los momentos principales que caracterizan la estrategia deben ser comentados. Esta es también la razón por la que se recomienda declarar y comentar todas las variables al comienzo del programa.

El bloque 1-2 se describen variables exteriores y variables globales.

De acuerdo a las normas, las variables externas y las variables globales deben abrirse antes de su primer uso (ver tipos de variables), esta es la razón por la que se declaran en la cabecera del programa. Todas las variables locales de la función start () se reunen y describen en la parte superior de la función (bloque 2-3) inmediatamente después de la cabecera de la función. No se requieren reglas en la declaración de variables locales, pero tampoco éstas estan prohibidas. Si un programador se enfrenta a dificultades en comprender el significado de una variable al leer el programa, puede ir a la parte superior del programa y conocer el significado de cualquier tipo de variable. Esto es muy conveniente en la programación práctica.

Bloque de tratamiento preliminar

En este ejemplo, el preprocesamiento consta de dos partes (bloque 3-4). El programa termina la operación si no hay suficientes barras en una ventana de un símbolo, en cuyo caso, es imposible detectar correctamente (en el bloque 5-6) los valores de las medias móviles necesarias para el cálculo de los criterios. Además aquí se analiza el valor de la variable Work. En la operación normal del AE, el valor de la variable es siempre 'true' (se configura por primera vez durante la inicialización). En caso de que ocurra un error crítico en la operación del programa, se le asigna 'falso' a la variable y start () termina su operación. Este valor no cambiará en el futuro, es por eso que el código que sigue no se ejecutará. En tal caso, la operación del programa debe detenerse y debe ser detectado el motivo del error crítico (en caso necesario, contactando con el dealing center). Después de resuelta la situación, el programa se puede iniciar una vez más, es decir, el AE puede ser asociado a la ventana de un símbolo.

Contabilidad órdenes

El Asesor Experto descrito permite trabajar sólo con una orden de mercado. La tarea del bloque de órdenes de contabilidad (bloque 4-5) es definir las características de la orden abierta, si es que hay alguna. Se comprueban las órdenes que pasan a través del bucle "for", todas las órdenes de mercado y órdenes en espera de ser ejecutada. Es decir, a partir del primer (int i = 1) a la última de ellas (i <= OrdersTotal ()). En cada iteración del ciclo la siguiente orden es seleccionada por la función OrderSelect (). La selección se realiza a partir de una fuente de apertura y de órdenes en espera de ser ejecutadas (SELECT_BY_POS).

```
   if (OrderSelect(i-1,SELECT_BY_POS)==true) // If there is the next one
```

Si la selección se ejecuta con éxito (es decir, hay una orden más en el terminal), entonces debe analizarse esta orden y su situación: Si la orden se abre para el símbolo en el que opera el EA, y si la orden es de mercado o pendiente. Esto también debe tenerse en cuenta a la hora de contar las órdenes. En la línea:

```
    if (OrderSymbol()!=Symb)continue;      // Another security
```

todas las órdenes abiertas en otro valor, se eliminan. El operador "continue" detiene la iteración y las características de esa orden no se procesan. Pero si la orden se abre para el valor, a la ventana en el cual el AE que se vincula, se analizaran después.

Si OrderType () devuelve un valor mayor que 1 (véase Tipos de Operaciones), la orden selecciona es una orden en espera de ser ejecutada. Pero en este Asesor Experto la gestión de órdenes en espera no está prevista. Esto significa que la ejecución de start () debe darse por concluida, porque se produjo una situación de conflicto. En tal caso, se muestra un mensaje sobre la finalización de la operación de start() después la ejecución se detiene por el operador «return».

Si la última comprobación, que analiza la orden es un orden de mercado, se calculan y analizan la cantidad total de órdenes del valor. Para la primera de dichas órdenes se definen todas las caracteristicas necesarias. Si en la siguiente iteración, viendo el contador (variable total), se encuentra la segunda orden de mercado, la situación se considera también en conflicto, debido a que la AE no puede manejar más de una orden de mercado. En tal caso, la ejecución de la funcion especial start () se detiene después de mostrar el mensaje correspondiente.

Como resultado de la ejecución del bloque de contabilidad (si todos los controles se pasaron con éxito), la variable Total conserva su valor cero si no hay órdenes de mercado, o le da el valor 1 si hay un mercado en nuestro simbolo. En este último caso, algunas de las variables establecidas en correspondencia con las características de la orden (número, tipo, precio de apertura, niveles de stop y valor de la orden) también obtiene sus valores.

Cálculo de criterios de comercio

En el ejemplo analizado la definición de criterios de comercio (bloque 5-6) se calcula sobre la base de la diferencia entre Medias Móviles con diferentes períodos de promedio. De acuerdo con criterios aceptados es un gráfico alcista si el valor actual de la MA con menor período es mayor que el valor de la MA con mayor plazo, y la diferencia entre los valores es mayor que un determinado valor. En un movimiento bajista, la MA con menor período es inferior a MA con mayor periodo y la diferencia también es mayor que un cierto valor crítico.

Los valores iniciales del bloque se calculan a partir de las MAs con promedio de los períodos Period_MA_1 y Period_MA_2. El hecho significativo de cualquier criterio comercial se expresa a través del valor de la variable correspondiente. Las variables Open_Buy y Open_Sell denotar el criterio desencadenante para la apertura de órdenes de compra y venta, las variables Cls_ B y Cierre_Sell para el cierre. Por ejemplo, si un criterio para la apertura de Compra no se ha activado, el valor de Open_Buy sigue siendo 'falso' (fijado en la inicialización de la variable); si se ha desencadenado, Open_Buy obtiene el valor 'true'. En este caso, el criterio para el cierre Vender coincide con el de la apertura de Compra, el criterio para la apertura de Venta coincide con el de el cierre de Compra.



Los criterios aceptados de trading de este ejemplo se utilizan solamente para fines educativos y no deben ser considerados como una directriz de comercio en una cuenta verdadera.

Ordenes de Cierre

Está escrito antes de que este Asesor Experto intente siquiera la operación de apertura de una sola orden de mercado en la ventana en el que el AE se adjunta. Para el momento en que el control del programa se pasa al bloque de orden de cierre se sabe con seguridad si en el momento actual hay o no órdenes en el simbolo, o sólo hay una orden de mercado. Es por eso que el código en el bloque de órdenes de cierre está escrito de manera que solamente puede cerrarse una orden correctamente.

Este bloque se basa en un bucle infinito 'while', el cuerpo se compone de dos partes similares: una para el cierre de una orden de Compra y otra para el cierre de una orden de Venta. "While" se utiliza aquí con el fin de que en caso de que una operación de comercio fracase pueda repetir la operación otra vez.

En la cabecera del primer operador 'if' se calcula la condición para el cierre de un fin de Compra (Las órdenes de Venta se cierran de forma análoga). Si el tipo de una orden abierta anteriormente corresponde a una compra (ver Tipos de Operaciones) y el signo para el cierre de compra es relevante, el control se pasa al cuerpo del operador 'if' cuando se forma una petición de cierre. Como una orden de cierre de precios en la función OrderClose () se indica el valor de una two-sided quote (cotización de doble cara) correspondiente al tipo de orden (ver requisitos y limitaciones en la toma de Órdenes). Si se ejecuta correctamente una operación de comercio, entonces se muestra un mensaje sobre el cierre de la orden, la actual iteración 'while' se detiene y la ejecución del bloque de orden de cierre termina. Pero si la operación falla, se llama a la función definida por el usuario que se ocupa de la tramitación de errores Fun_Error () del bloque 10-11.

Procesamiento de Errores

El último código de error calculado por GetLastError () se utiliza como parámetro transferido a Fun_Error (). Dependiendo del código de error, Fun_Error () devuelve 1 si el error no es crítico y la operación se puede repetir, o devuelve 0 si el error es crítico. Los errores críticos se dividen en dos tipos: los que después de los cuales la ejecución del programa puede continuar (por ejemplo, un error común) y los que, después de su ejecución, debe detenerse cualquier tipo de operación de comercio (por ejemplo, una cuenta bloqueada).

Si después de una infructuosa operación de comercio la función devuelve 1, la actual iteración 'While' termina y durante la próxima iteración se realiza otro intento de ejecutar la operación de cerrar la orden. Si la función devuelve 0, la actual ejecución start () se detiene. En el siguiente tick start () iniciará el Terminal de Usuario de nuevo y si las condiciones de orden de cierre se mantienen se realizará otro intento de cerrar la orden.

Si durante el procesamiento del error se ha descubierto que además la ejecución del programa es un absurdo (por ejemplo, el programa opera en una vieja versión del Terminal de Usuario) durante el próximo inicio de la ejecución de la función especial start (), el bloque de tratamiento preliminar dará por terminado el programa cuando analize el valor de la variable de bandera **Work**.

Cálculo de la cantidad de lotes para nuevas órdenes

El Monto de los lotes puede ser calculado de conformidad con la configuración del usuario siguiendo una de dos variantes. La primera variante es un valor constante, creado por un usuario. Según la segunda variante la cantidad de lotes se calcula sobre la base de una cantidad igual a un porcentaje determinado (establecido por el usuario) del margen libre.

Al comienzo del bloque (7-8) de definición de la cantidad de lotes para nuevos órdenes, se calculan los valores necesarios de algunas variables: cantidad mínima de lotes permitidos y paso de cambio de lotes establecido por un intermediario, el margen libre y el precio de un lote para el símbolo de un valor.

En este ejemplo es la siguiente. Si un usuario ha creado un cierto valor no-cero de la variable externa Lots, por ejemplo 0.5, se acepta como la cantidad de lotes Lts cuando se forma una solicitud comercio de apertura de una orden. Si se asigna 0 a Lts, el número de lotes Lts se define en base de la variable Prots (porcentaje), margen libre y las condiciones establecidas por el broker.

Después de calculada Lts se lleva a cabo una comprobación. Si este valor es inferior al valor mínimo permitido, el valor mínimo permitido se acepta, pero si el margen libre no es suficiente, la función start() termina la ejecución después del correspondiente mensaje.

Órdenes de apertura

El bloque de la apertura de órdenes (bloque 8-9) al igual que el bloque de cierre de órdenes es un bucle infinito 'while'. En la cabecera del primer operador 'if' se calculan las condiciones para la apertura de una orden de Compra: si no hay órdenes para el simbolo (variable total es igual a 0) y el signo de apertura de una orden de Compra es pertinente (Open_Buy es cierto), El control se pasa al cuerpo orperador 'if' para la apertura de una orden. En tal caso, después de las tasas de cambio se actualizan se calculan los niveles de stop de los precios.

Los valores de los niveles de stop son establecidos inicialmente por el usuario en las variables externas StopLoss y TakeProfit. En general el usuario puede establecer los valores de estos parámetros más bajos que lo que el corredor permite. Además un corredor puede cambiar la distancia mínima permitida en cualquier momento (se trata a menudo del caso de un mercado con fuertes movimientos, por ejemplo, antes de comunicados de prensa importantes). Es por eso que antes de la apertura de cada orden de stop, se debe calcular los niveles teniendo en cuenta los valores establecidos por el usuario y el valor mínimo permitido establecido por un intermediario.

Para el cálculo de los niveles de stop se utiliza la función definida por el usuario New_Stop (); como parámetro de paso del nivel de stop se utiliza el valor por el fijado por el usuario. En New_Stop (), en primer lugar, se calcula la distancia actual mínima permitida. Si el valor fijado por un usuario corresponde a los requerimientos del corredor, este valor se devuelve. Si es menor que el valor permitido, se utiliza el valor permitido por un corredor. Los precios de stop requeridos se calculan desde el correspondiente two-sided quote (ver requisitos y limitaciones en la toma de Órdenes).

Una solicitud comercio para la apertura de una orden se forma utilizando la función OrderSend (). Para el cálculo del precio de apertura de la orden y de las solicitudes de los precios de stop se utilizan los valores two-sided quote correspondientes al tipo de orden. Si una operación de comercio se ejecutó con éxito (es decir, el servidor ha devuelto el número de la orden que se ha abierto) a continuación se muestra un mensaje que informa sobre el éxito de la apertura de la orden. La función especial start () finaliza su ejecución. Si no se abrió ninguna orden y el Terminal de Usuario ha devuelto un error, el error se procesa de acuerdo con el algoritmo descrito anteriormente.

Algunas peculiaridades del código

El código del Asesor Experto analizado está orientado a la aplicación de una determinada estrategia. Tengase en cuenta, que algunas líneas de programa contienen variables y cálculos que podrían ser cambiados si la estrategia fuera cambiada.

Por ejemplo, según la estrategia aceptada el Asesor Experto es desarrollado para trabajar sólo con una orden. Se usa la variable Ticket tanto para la identificación de un número de orden de cierre (en el bloque de cierre 6-7) como para la identificación de la correcta ejecución de una operación comercial de apertura de una orden (en el bloque de apertura 8-9). En este caso, esta solución es aceptable. Sin embargo, si tomamos el código analizado como base para la aplicación de otra estrategia (por ejemplo, permitir órdenes opuestas) tendremos que introducir una o varias variables para ser capaces de reconocer los números de órdenes abiertas y determinar el éxito de las operaciones comerciales.

En una estrategia ampliada como ésta tendremos que cambiar las líneas de programa que contienen parte de la lógica de la estrategia original. Es decir en el bloque de órdenes contables no vamos a tener que terminar la operación del programa si hay varias órdenes para abrir en un valor. Además, las condiciones para la apertura y el cierre de órdenes tambien cambiarían. Esto supondría el cambio de código en los bloques de apertura y cierre de órdenes.

Sobre la base de este análisis podemos concluir fácilmente que el Asesor Experto simple descrito no es perfecto. En general, para la implementación de órdenes contables se debe utilizar una función universal basada en la utilización de arrays de datos y que no contengan lógica de una determinada estrategia. Lo mismo puede decirse de los bloques de apertura y cierre de órdenes. Un programa más completo debe contener una función analítica principal, todas las demás funciones definidas por el usuario deben estar subordinadas a ella. Esta función analítica debe contener un código de programa, en el que se analizan todas las condiciones para la aplicación de cualquier estrategia; todas las funciones subordinadas deben realizar acciones limitadas. La función de contabilidad de las órdenes deben sólo contabilizar órdenes, las funciones de apertura y cierre de órdenes solo deben abrir y cerrar órdenes y, la función analítica debe "pensar" y gestionar todas las demás funciones, es decir, llamarlas cuando sea necesario.

Creación de indicadores personalizados

Al crear una estrategia comercial de un desarrollador a menudo se enfrenta a la necesidad de señalar gráficamente en una ventana de un símbolo en una cierta función matemática calculado por un usuario (programador). Con este fin MQL4 ofrece la posibilidad de crear indicadores personalizados.

Indicador personalizado es una aplicación programa codificado en MQL4, es básicamente destinados a la exhibición gráfica de función matemática s calculado preliminarmente.

Custom indicador de estructura

Necesidad de Buffers

El principio fundamental que subyace es la usuario de pasar los indicadores valores del indicador arrays a un Terminal de Usuario (para dibujar líneas indicador) a través de intercambio de buffers.

Buffer es un área de memoria que contiene valores numéricos de una serie de indicadores.

MQL4 norma implica la posibilidad de utilizar hasta ocho líneas indicador utilizando un indicador personal. Uno de los indicadores y una gama de amortiguación se pongan en correspondencia con cada indicador. Cada buffer tiene su propio índice. El índice del primer buffer es 0, de la segunda - 1, y así sucesivamente, la última de ellas tiene el índice de 7. Fig. 115 muestra cómo la información de un indicador de la usuario se pasa a través de buffers a un Terminal de Usuario para dibujar líneas indicador.

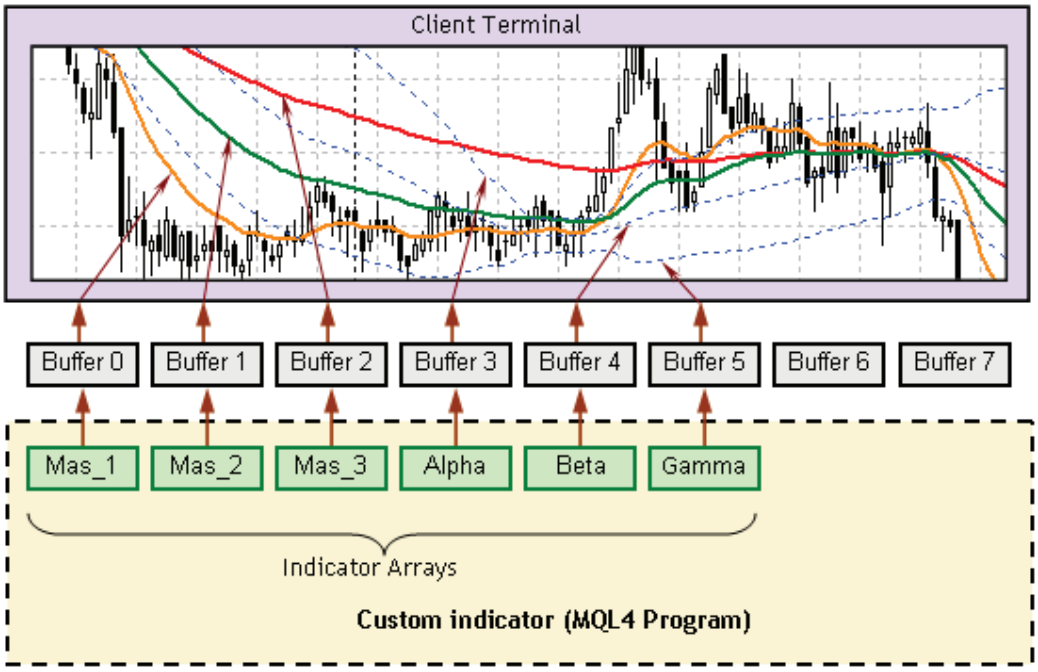


Fig. 115. La superación de los valores del indicador arrays a través de un buffer a un Terminal de Usuario.

La orden general de la construcción de líneas indicador es la siguiente:

1. Los cálculos se realizaron en un indicador personal; como resultado valores numéricos se asignan a elementos gama de indicadores.
2. Los valores del indicador gama elementos son enviados a un Terminal de Usuario a través de buffers.
3. Sobre la base del valor recibido arrays de buffers de un Terminal de Usuario indicador muestra las líneas.

Componentes de un indicador personalizado

Vamos a analizar un simple usuario indicador que muestra dos líneas - una línea es construir sobre la base de la barra de precios máximos, el segundo utiliza un mínimo de los precios.



Ejemplo de un indicador simple usuario userindicator.mq4

```
//------------------------------------------------- --------------------
// Userindicator.mq4
// El código debería ser usado para fines educativos únicamente.
//------------------------------------------------- --------------------
# propiedad indicator_chart_window // indicador se señala en la ventana principal
# propiedad indicator_buffers 2 // Número de búferes
# propiedad indicator_color1 Azul // Color de la 1 ª línea
# propiedad indicator_color2 Rojo // Color de la 2 ª línea

Buf_0 doble [], Buf_1 [] // Declarar arrays (para el indicador buffers)
//------------------------------------------------- --------------------
int init () // Función especial init ()
(
SetIndexBuffer (0, Buf_0); // Asignar un array a un buffer
SetIndexStyle (0, DRAW_LINE, STYLE_SOLID, 2); // Estilo de línea
SetIndexBuffer (1, Buf_1); // Asignar un array a un buffer
SetIndexStyle (1, DRAW_LINE, STYLE_DOT, 1); // Estilo de línea
return; // Salir de la especial Funct.  init ()
)
//------------------------------------------------- --------------------
int start () // Función especial start ()
(
int i, // Bar Índice
Counted_bars; // Número de contados bares
//------------------------------------------------- --------------------
Counted_bars = IndicatorCounted (); // Número de contados bares
i = Bares - Counted_bars - 1; // Índice de la primera incontables
while (i> = 0) // Loop para incontables bares
(
Buf_0 [i] = alto [i]; // Valor de 0 a buffer i bar
Buf_1 [i] = Baja [i]; // Valor de buffer en 1 ª i bar
i -; // Cálculo del índice de la barra siguiente
)
//------------------------------------------------- --------------------
return; // Salir de la especial Funct. start ()
)
//------------------------------------------------- --------------------
```

Vamos a analizar detalladamente el indicador partes. En cualquier programa de aplicación escrito en MQL4 puede indicar parámetros de configuración que proporcionan el correcto programa de servicios de terminal de un cliente. En este ejemplo el jefe del programa (véase la estructura programática) contiene varias líneas con las directivas # propiedad.

La primera directiva indica en qué ventana de terminal que el cliente debe llamar el indicador de líneas:

```
# propiedad indicator_chart_window // indicador se señala en la ventana principal
```

En MQL4 hay dos variantes de dibujo indicador de líneas: en la ventana principal de seguridad y en una ventana aparte. Ventana principal es la ventana que contiene una gráfica de seguridad. En este ejemplo el parámetro # indicator_chart_window en propiedad directorio indica que un cliente debe llamar la terminal indicador de líneas en la ventana principal.

La siguiente línea muestra el número de buffers usados en el indicador:

```
# propiedad indicator_buffers 2 // Número de búferes
```

En el ejemplo analizado dos indicadores se trazan líneas. Un buffer es asignado a cada uno de amortiguación, por lo que el número total de buffers es de dos.

Las siguientes líneas describen los colores del indicador líneas.

```
# propiedad indicator_color1 Azul // Color de la 1 ª línea
# propiedad indicator_color2 Rojo // Color de la 2 ª línea
```

Parámetros indicator_color1 y indicator_color2 definir el color correspondiente para el establecimiento de topes - en este caso para buffers con índices 0 (Azul) y 1 (Roja). Tenga en cuenta que las cifras en los nombres de parámetro indicator_color1 y indicator_color2 no son los índices de amortiguación. Estas cifras son parte constante de nombres que se establecen de conformidad con topes. Para cada color constante se puede fijar a discreción del usuario.

En la línea siguiente indicador arrays se declaran:

```
Buf_0 doble [], Buf_1 [] // Declarar arrays (para el indicador buffers)
```

El indicador está destinado a dibujar dos líneas indicador, por lo que debemos declarar dos mundial de una dimensión arrays, uno para cada línea. Nombres de indicador arrays son hasta un usuario. En este caso, los nombres de Buf_0 array [] y Buf_1 [] se utilizan, en otros casos, otros nombres se pueden utilizar, por ejemplo, Line_1 [], Alfa [], integral [], etc Es necesario declarar matrices a nivel mundial , Porque los elementos array valores deben ser preservados de una llamada especial de la función start ().

El indicador se describe la usuario se construye sobre la base de dos funciones especiales-init () y start (). La función init () contiene la parte del código utilizado en el programa sólo una vez (véase Funciones especiales).

Una parte muy importante de acción se realiza en la línea:

```
SetIndexBuffer (0, Buf_0); // Asignar un array a un buffer
```

Usando la función SetIndexBuffer () una condición necesaria de amortiguación (en este caso con el índice 0) se pone en correspondencia con un array (en este caso Buf_0). Esto significa para construir el primer indicador de línea de un Terminal de Usuario aceptar los datos contenidos en la matriz Buf_0 utilizando el buffer de cero.

Además, el estilo de línea se define:

```
SetIndexStyle (0, DRAW_LINE, STYLE_SOLID, 2); // Estilo de línea
```

Para la amortiguación de cero (0) terminal de un cliente debe utilizar los siguientes estilos de dibujo: línea simple (DRAW_LINE), línea (STYLE_SOLID), ancho de línea 2.

Las siguientes dos líneas contienen ajustes para la segunda línea:

```
SetIndexBuffer (1, Buf_1); // Asignar un array a un buffer
SetIndexStyle (1, DRAW_LINE, STYLE_DOT, 1); // Estilo de línea
```

Así, según el código de la función especial de inicio () ambas líneas indicador se dibujará en la ventana principal de seguridad. La primera de ellas será una sólida línea azul con la anchura de 2, el segundo es una línea de puntos rojos (STYLE_DOT) de una anchura normal. Indicador líneas se pueden extraer de otros estilos como así (véase el Indicador Estilos de Líneas).

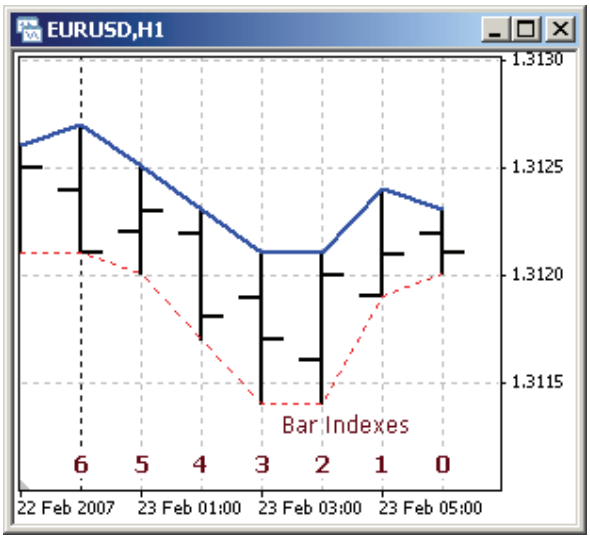
Calcular los valores del indicador arrays elementos (estar atento)

Los valores del indicador arrays elementos se calculan en la función especial start (). Para entender correctamente el contenido de start () Código de prestar atención a la orden de indexación de barras. La sección Las matrices se describen en detalle el método de indexación de arrays-timeseries. De acuerdo con este método de indexación de barras empieza de cero. La barra de cero es una corriente aún unformed bar. El bar más cercano es el índice de 1. La siguiente es la 2 y así sucesivamente.

A medida que los nuevos bares aparecerán en una ventana de un símbolo, los índices de los ya formados (la historia) son los cambios en los bares. La nueva (actual, sólo formado, derecha) recibe la barra de índice cero, el uno a la izquierda de él (que acaba totalmente formado) obtiene el índice 1 y los valores de los índices de todas las barras de la historia son también aumentaron por uno.

El método descrito de la indexación de barras es el único posible para todo el sector de línea sistema de comercio MetaTrader, y es tenido en cuenta a la hora de dibujar líneas utilizando tanto técnicas como indicadores personales.

Se dijo anteriormente que el indicador de líneas se construyen sobre la base de la información numérica contenida en el indicador de arrays. Un indicador serie contiene información sobre las coordenadas de puntos sobre los que un indicador de la línea se dibuja. Y la coordenada Y de cada punto es el valor de un indicador Los **elementos de** matriz, y la coordenada X es el valor de un indicador **Índice** Los elementos de matriz. En el ejemplo analizado el primer indicador de la línea se dibuja usando los valores máximos de barras. Fig, 116 este indicador muestra la línea (de color azul) en un ventana de un símbolo, se construye sobre la base del indicador gama Buf_0.



Índice valor del indicador gama Buf_0	Elemento de valor del indicador gama Buf_0
0	1.3123
1	1.3124
2	1.3121
3	1.3121
4	1.3123
5	1.3125
6	1.3127
...	...

Fig. 116. La correspondencia de las coordenadas de un indicador línea con los valores de un indicador matriz.

Índice de valor de un indicador variedad está fuera de un Terminal de Usuario en correspondencia con un índice de bar - Índice estos valores son iguales. Se debe también tener en cuenta que el proceso de construcción de líneas indicador que sucede en tiempo real el modo en condiciones cuando en una nueva ventana de un símbolo en bares aparecen de vez en cuando. Y todas las barras de la historia se desplaza hacia la izquierda. Para que el indicador de línea trazada correctamente (cada línea punto por encima de su bar) también debe ser trasladado junto con las barras. Por lo tanto, hay necesidad (necesidad técnica) para volver a indexar un indicador matriz.

La diferencia fundamental de un indicador de una gama gama habitual es la siguiente:

En el momento en que un nuevo bar se crean, índice de valores de la matriz de indicadores son elementos cambian automáticamente el Terminal de Usuario, es decir, - el valor de cada indicador índice es el aumento de uno y el conjunto de indicadores es el aumento de tamaño de un elemento (con un cero Índice).

Por ejemplo, la barra de cero en la Fig. 116 (plazo H1) tiene el tiempo de apertura 6:00. A las 7:00 una nueva barra aparecerá en la ventana de un símbolo. El bar abre a las 6:00 automáticamente el índice 1. Para que el indicador de línea trazada correctamente en este bar, el Terminal de Usuario va a cambiar el índice de la matriz elemento indicador correspondiente al bar abre a las 6:00. En el cuadro de la Fig. 116 este elemento está escrito en la primera línea. Junto con el que los índices de todos los elementos de serie se incrementará en el Terminal de Usuario. Un índice de la matriz elemento correspondiente al bar abre a las 6:00 obtendrá el valor 1 (antes de que se trataba de igual a 0). El indicador se convertirá en serie más grande de un elemento. El índice de un nuevo elemento añadido será igual a 0, el valor de este elemento será un nuevo valor que refleja coordinar el indicador de línea de cero en un bar. Este valor se calcula en función especial la de inicio () en cada una tick.

Los cálculos en la función especial start () debe llevarse a cabo a fin de que las acciones adicionales no se realizaron. Antes de que el indicador se vincula a un gráfico, no refleja ningún indicador líneas (porque los valores del indicador arrays no son aún por definir). Es por eso que en la primera salida de la función especial start () indicador de valores array debe calcularse para todos los bares, en los que el indicador de línea debe explotar. En el ejemplo analizado estos son todos los bares presente en un gráfico (los cálculos iniciales pueden llevarse a cabo no para todos los bares, pero para algunos última parte de la historia, es descrito en otros ejemplos). Ar todas las demás empieza especial de la función start () no hay necesidad de calcular los valores del indicador variedad para todos los bares de nuevo. Estos valores son calculados y ya figuran en el indicador de la matriz. Es necesario calcular el valor actual del indicador de la línea sólo en cada nueva tick de la barra de cero.

Para la aplicación de la tecnología descrita es muy útil en función estándar MQL4 - IndicatorCounted ().

Función IndicatorCounted ()

```
INT IndicatorCounted ()
```

Esta función devuelve el número de bares que no han cambiado desde el último indicador de llamada.

Si el indicador nunca ha sido vincula a un gráfico, en la primera start () la ejecución de Counted_bars valor será igual a cero:

```
Counted_bars = IndicatorCounted (); // Número de contados bares
```

Esto significa que el indicador conjunto no contiene ningún elemento con valor predefinido con anterioridad, es por eso que toda la gama de indicadores debe ser calculada a partir de principio a fin. El indicador conjunto se calcula a partir de la más antigua de la barra a cero. Índice de los más antiguos bar, que a partir de cálculos deben comenzar, se calcula de la siguiente manera:

```
i = Bares - Counted_bars - 1; // Índice de la primera incontables
```

Supongamos que en el momento de conectar el indicador hay 300 bares en un gráfico de ventanas. Este es el valor de las variables predefinidas Bares. Tal como se define anteriormente, Counted_bars es igual a 0. Así que, como resultado obtenemos que i índice de la primera barra de incontables (la última, a partir de cálculos que deben llevarse a cabo) es igual a 299.

Todos los valores del indicador gama elementos se calculan en el bucle while ():

```
while (i> = 0) // Loop para incontables bares
(
Buf_0 [i] = alto [i]; // Valor de 0 a buffer i bar
Buf_1 [i] = Baja [i]; // Valor de buffer en 1 ª i bar
i -; // Cálculo del índice de la barra siguiente
)
```

Si bien i se encuentra dentro del rango de la primera incontables bar (299) a la actual (0) inclusive, los valores del indicador gama elementos se calculan para ambas líneas indicador. Nota, los valores del indicador gama elementos se calculan durante uno (el primero) salida de la función especial start (). Durante los cálculos Terminal de Usuario recuerda elementos, para lo cual se calcularon los valores. La última iteración, mientras que en () se realiza cuando i es igual a 0, es decir, los valores del indicador arrays se calculan para la barra de cero. Cuando el lazo es más, la función especial start () termina su ejecución y el control se pasa al Terminal de Usuario. El Terminal de Usuario, a su vez, aprovechará todos (en este caso dos) líneas indicador de conformidad con los valores calculados gama de elementos.

En la siguiente tick start () se iniciará el Terminal de Usuario de nuevo. Nuevas medidas dependerá de la situación (seguiremos analizando el ejemplo para 300 bares).

Variante 1. Una nueva tick viene durante la formación de la **actual** barra de cero (la situación más común).

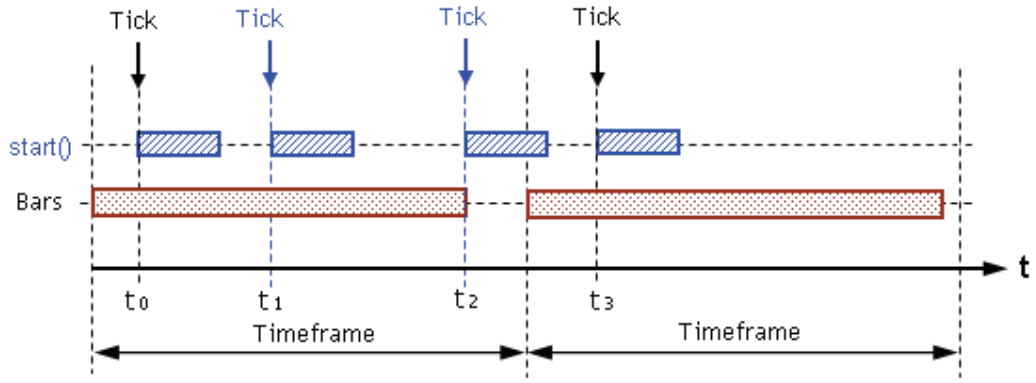


Fig. 117. El procesado tick pertenece a la barra actual.

Fig. 117 muestra dos ticks recibida por el terminal en los momentos de tiempo t 1 y t 2. El analizaron situación será la misma para ambos ticks. Vamos a rastrear la ejecución de start () que se puso en marcha en el momento t 2. Durante la ejecución de la función start () la siguiente línea se llevará a cabo:

```
Counted_bars = IndicatorCounted (); // número de bares contados
```

IndicatorCounted () devolverá el valor 299, es decir, desde el último inicio () 299 barras no se modificaron. Como resultado índice i valor será igual a 0 (300-299-1):

```
i = Bares - Counted_bars - 1; // Índice de la primera incontables
```

Esto significa en el próximo, mientras que () los valores de la matriz con los elementos cero índice se calcula. En otras palabras, la nueva posición de un indicador de línea cero en el bar se calculará. Cuando el ciclo se haya terminado, start () se detendrá la ejecución y se pasa el control al Terminal de Usuario.

Variante 2. Una nueva tick es la primera de marcar un **cero** bar (que ocurre de vez en cuando).

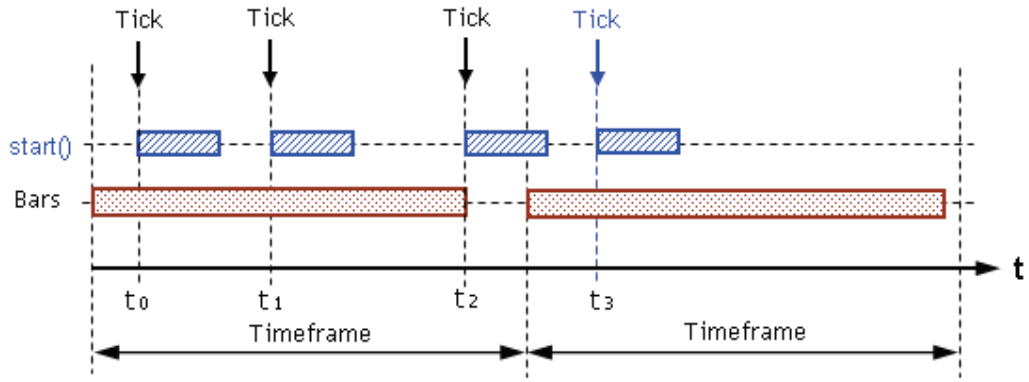


Fig. 118. El procesado tick es la primera de marcar una nueva barra de cero.

En este caso, el hecho de la aparición de un nuevo bar es importante. Antes de control se pasa a la función especial start (), Terminal de Usuario se basará de nuevo a todos los bares presentes en la ventana de un símbolo y volver a indexar todos los arrays declarados indicador (que se establece en correspondencia con los buffers). Además, los clientes terminal recordar que ya hay 301 bares, no 300 en un gráfico de ventanas.

Fig. 118 contiene situación cuando en la última tick de la barra anterior (en el momento t 2) la función start () se ha iniciado y ejecutado. Por eso, aunque ahora el primer bar (con índice 1) terminado en el momento t 2 se **calculó** el indicador, la función IndicatorCounted () devolverá el valor que estaba **en el bar anterior,** es decir, 299:

```
Counted_bars = IndicatorCounted (); // Número de contados bares
```

En la línea siguiente índice i se calculará, en este caso para marcar la primera de una nueva barra será igual a 1 (301-299-1):

```
i = Bares - Counted_bars - 1; // Índice de la primera incontables
```

Significa cálculo del indicador de valores en serie, mientras que () bucle en la aparición de un nuevo bar se llevarán a cabo tanto para el último bar y para la nueva barra de cero. Un poco antes durante la re-indexación de arrays indicador de la Terminal de Usuario aumentó el tamaño de estas matrices. Los valores de serie con elementos cero los índices no se definen antes de los cálculos en el bucle. Durante los cálculos en el bucle de estos elementos obtener algunos valores. Cuando los cálculos en start () son más, el control se devuelve al Terminal de Usuario. Después de que el Terminal de Usuario indicador se basará en las líneas de la barra de cero se basa únicamente en valores calculados gama de elementos con índices de cero.

Variante 3. Una nueva tick es la primera de marcar una **nueva** barra de cero, pero el último pero es una tick no procesados (caso raro).

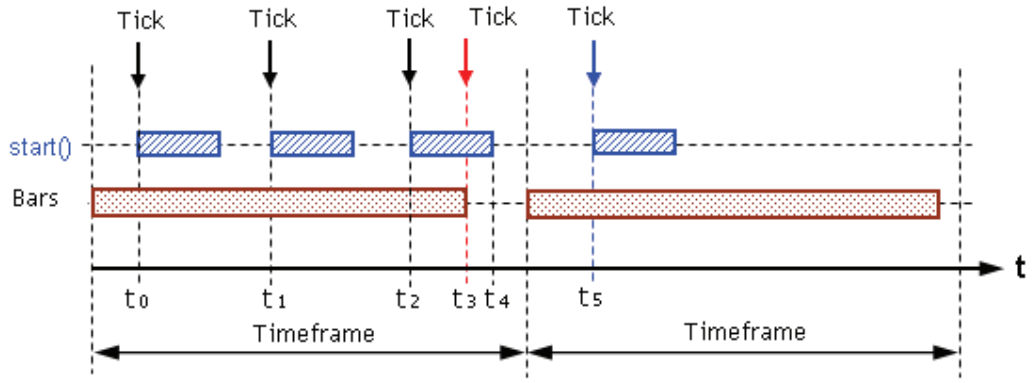


Fig. 119. No todas los ticks de la barra anterior fueron procesadas.

Fig. 119 muestra la situación cuando start () se inició en la primera de marcar un nuevo bar en el momento t 5. Anterior tiempo esta función se inició en el momento t 2. Tick que llegó a la terminal en el momento t 3 (flecha roja) no fue procesado por el indicador. Esto sucedió porque start () el tiempo de ejecución t 2 - t 4 es más grande que el intervalo entre los ticks t 2 - t 3. Este hecho será detectado por el Terminal de Usuario durante la ejecución de start () puesto en marcha en el momento t 5. Durante los cálculos en línea:

```
Counted_bars = IndicatorCounted (); // Número de contados bares
```

IndicatorCounted () devolverá el valor 299 (!). Este valor es cierto - a partir del momento del último indicador de llamada 299 bares no se modificaron después (ahora ya) 301. Es por ello que el índice calculado de la primera (izquierda), bar, de que los cálculos de valores Los elementos de matriz se pondrá en marcha, será igual a 1 (301-299-1):

```
i = Bares - Counted_bars - 1; // Índice de la primera incontables
```

que significa al mismo tiempo durante () dos iteraciones ejecución se llevará a cabo. Durante los primeros valores de una serie elementos con el índice i = 1 se calculará, es decir, Buf_0 [1] y Buf_1 [1]. No, por el momento los cálculos de inicio, bares y arrays indicador ya se han re-indexada por el Terminal de Usuario (debido a una nueva barra de comenzar, entre comienza especiales de la función start ()). Es por ello que los cálculos para los elementos de las matrices con índice 1 se calculará sobre la base de arreglo de timeseries (máximo y mínimo de valores de una barra de precio), también con el índice 1:

```
while (i> = 0) // Loop para incontables bares
(
Buf_0 [i] = alto [i]; // Valor de 0 a buffer i bar
Buf_1 [i] = Baja [i]; // Valor de buffer en 1 ª i bar
i -; // Cálculo del índice de la barra siguiente
)
```

Durante la segunda iteración del while () los valores para los elementos con índices de cero, es decir, para la barra de cero, se calcula sobre la base de los últimos valores conocidos de arrays-timeseries.

Uso de la tecnología descrita para el cálculo de indicadores personalizado permite, en primer lugar, para garantizar el cálculo de los valores de todos los elementos conjunto de indicadores, independientemente de la naturaleza específica de marcar la historia, y en segundo lugar, para realizar los cálculos sólo para los incontables bares, es decir, utilizar económicamente el cálculo de recursos .

No, un bar es considerado incontables si cálculo del elemento valores de un indicador arrays, al menos, para marcar un último de la barra no se realiza.

A partir de la usuario indicador userindicator.mq4 en un cuadro ventana, verá dos líneas - una gruesa línea azul basa en la barra de máximos y una línea roja punteada construido a partir de su mínimos (Fig. 120).

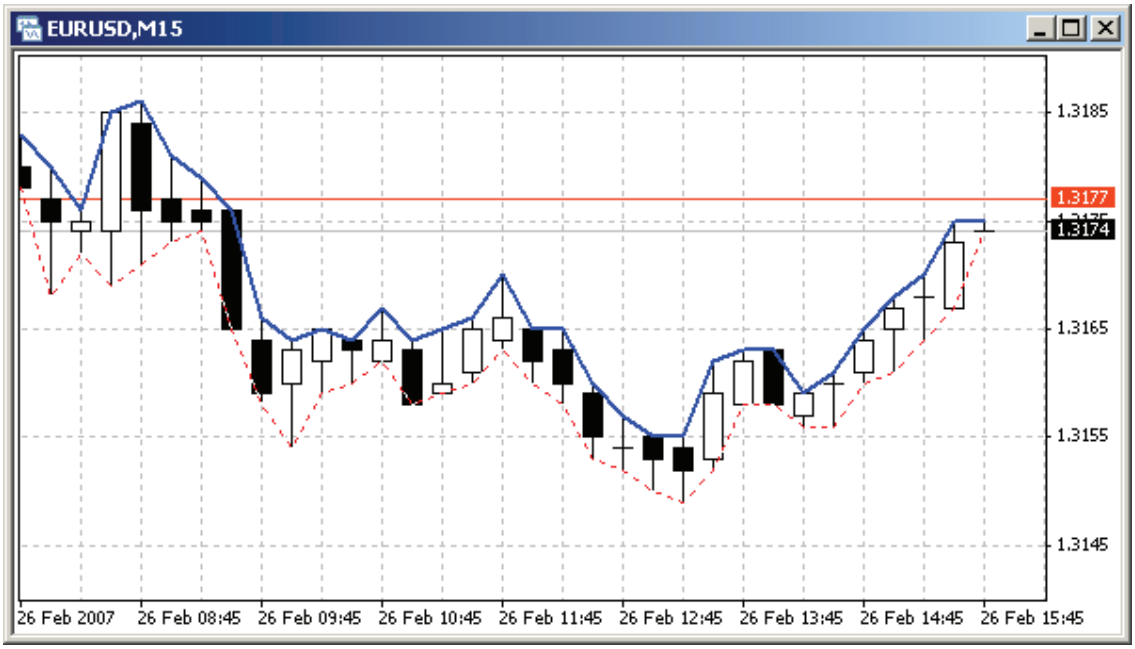


Fig. 120. Dos líneas indicador en una ventana de un símbolo, construido por el indicador userindicator.mq4.

Cabe señalar, que uno puede construirse una usuario indicador, el indicador de líneas que coinciden con las líneas de una técnica análoga indicador. Se puede hacer fácilmente si las fórmulas de cálculo como en el indicador personal, las mismas fórmulas que en el indicador técnico se utilizan. Para ilustrar esto vamos a mejorar el código de programa analizado en el ejemplo anterior. Deje que el indicador dibujar líneas a un valor promedio de máximos y mínimos de los últimos varios bares. Es fácil de llevar a cabo cálculos necesarios: Simplemente tenemos que encontrar valores medios de arrays-timeseries elementos. Por ejemplo, el valor de un indicador array con el índice 3 (es decir, el indicador de línea de coordinar el tercer bar), sobre la base de los últimos cinco máximos se calcula de la siguiente manera:

Buf_0 [3] = (Alto [3] + Alta [4] + Alta [5] + Alta [6] + Alta [7]) / 5

Análogas cálculos se pueden realizar para las líneas de un indicador construido a mínimos.

Ejemplo de un indicador simple usuario averagevalue.mq4. Indicador líneas se basan en promedio mínima y máxima de los valores de N barras.

```
//------------------------------------------------- --------------------
// Averagevalue.mq4
// El código debería ser usado para fines educativos únicamente.
//------------------------------------------------- --------------------
# propiedad indicator_chart_window // indicador se señala en la ventana principal
# propiedad indicator_buffers 2 // Número de búferes
# propiedad indicator_color1 Azul // Color de la 1 ª línea
# propiedad indicator_color2 Rojo // Color de la 2 ª línea

extern int Aver_Bars = 5; // número de barras para el cálculo

Buf_0 doble [], Buf_1 [] // Declarar indicador arrays
//------------------------------------------------- --------------------
int init () // Función especial init ()
(
//------------------------------------------------- --------------------
SetIndexBuffer (0, Buf_0); // Asignar un array a un buffer
SetIndexStyle (0, DRAW_LINE, STYLE_SOLID, 2); // Estilo de línea
//------------------------------------------------- --------------------
SetIndexBuffer (1, Buf_1); // Asignar un array a un buffer
SetIndexStyle (1, DRAW_LINE, STYLE_DOT, 1); // Estilo de línea
//------------------------------------------------- --------------------
return; // Salir de la especial funct.init ()
)
//------------------------------------------------- --------------------
int start () // Función especial start ()
(
int i, // Bar Índice
n, // parámetro formal
Counted_bars; // Número de contados bares
doble
Sum_H, // Suma de altos valores para el período
Sum_L; // Suma de valores de baja para el período
//------------------------------------------------- --------------------
Counted_bars = IndicatorCounted (); // Número de contados bares
i = Bares - Counted_bars - 1; // Índice de la primera incontables
while (i> = 0) // Loop para incontables bares
(
Sum_H = 0; // anulación a principios de bucle
Sum_L = 0; // anulación a principios de bucle
for (i = n, n <= i + Aver_Bars - 1; n + +) // Loop de resumir los valores
(
Sum_H = + Sum_H Alto [n]; // La acumulación de valores suma máxima
Sum_L = + Sum_L Baja [n]; // La acumulación de valores suma mínima
)
Buf_0 [i] = Sum_H / Aver_Bars; // Valor de 0 a buffer i bar
Buf_1 [i] = Sum_L / Aver_Bars; // Valor de buffer en 1 ª i bar

i -; // Cálculo del índice de la barra siguiente
)
//------------------------------------------------- --------------------
return; // Salir de la especial Funct.  start ()
)
//------------------------------------------------- --------------------
```

En este ejemplo hay una variable externa Aver_Bars. El uso de esta variable el usuario puede indicar el número de bares, para que un valor promedio se calcula. En start () este valor se utiliza para el cálculo de un valor medio. En el bucle "para" la suma de máximo y mínimo de valores se calcula por el número de barras correspondiente al valor de la variable Aver_Bars. En los próximos dos líneas de programa de valores del indicador gama elementos se calculan para el indicador correspondiente a las líneas de mínima y máxima de valores.

El promedio del método utilizado aquí también se aplica para los cálculos técnicos en el indicador de media móvil. Si asignamos el indicador analiza la usuario averagevalue.mq4 y la técnica indicador de media móvil, vamos a ver tres líneas indicador. Si el mismo período es de un promedio establecido para ambos indicadores, línea de media móvil coincidirá con una de las líneas de usuario indicador (para este fin parámetros descritos en la Fig. 121 debe ser especificado en el indicador de configuración técnica).

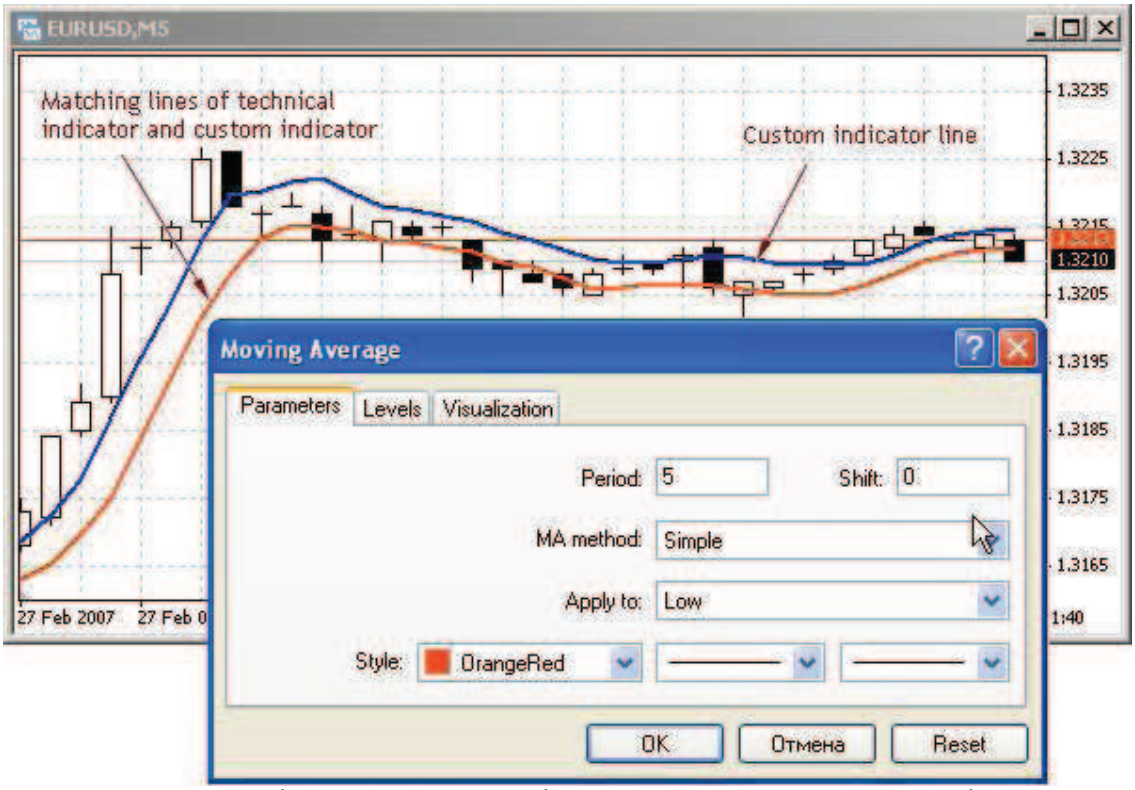


Fig. 121. Coincidentes líneas de un indicador técnico personalizado y un indicador (línea roja).

De este modo, utilizando técnicas indicador de un usuario puede construir el reflejo de cualquier regularidades necesarias en el trabajo práctico.

Custom indicador opciones

Dibujo indicador líneas en ventanas separadas

MQL4 ofrece un gran servicio personalizado para la construcción de indicadores que hace uso de ellos muy conveniente. En particular, el indicador de líneas se pueden extraer en una ventana aparte. Esto es conveniente cuando los valores absolutos el indicador de línea de amplitud es sustancialmente menor (o mayor) que los precios de seguridad. Por ejemplo, si estamos interesados en la diferencia entre los valores medios de barra de mínimos y máximos en un determinado intervalo histórico, dependiendo de plazo este valor será igual a aproximadamente de 0 a 50 puntos (por ejemplo, para M15). No es difícil construir un indicador de línea, pero en una ventana de un símbolo en esta línea se dibujará en el rango de 0 - 50 puntos de una garantía de precios, es decir, sustancialmente inferior a la gráfica zona refleja en la pantalla. Es muy incómodo.

Para dibujar líneas indicador en una ventana aparte (que se encuentra en la parte inferior de una ventana de un símbolo), en la directiva # propiedad (al comienzo del programa) indicator_separate_window parámetro debe especificarse:

```
# propiedad indicator_separate_window // indicador se basa en una ventana aparte
```

En el momento en que dicho indicador se vincula a una ventana de un símbolo, Terminal de Usuario crea una ventana separada por debajo de un cuadro, en el que el indicador calculado líneas en el indicador se extraerán. Dependiendo de la configuración de color y tipos de líneas indicador que se utilizará en este o aquel estilo.

Limitar la historia de cálculo

En la mayoría de los casos indicador líneas contienen información útil sólo en la más reciente historia. La parte del indicador se basó en las líneas antiguas barras (por ejemplo, 1 mes de edad minutos de tiempo) no puede considerarse para la toma de decisiones comerciales. Por otra parte, si hay un montón de barras en un gráfico de ventanas, el tiempo invertido en el cálculo y dibujo de líneas indicador es injustificadamente amplia. Esto puede ser crítico en el programa de depuración, cuando un programa es compilado a menudo y luego comenzar. Por eso es necesario llevar a cabo cálculos no para toda la historia, pero para la parte limitada del bar más reciente historia.

Con este fin, una historia variable externa se utiliza en el siguiente programa. El valor de esta variable se toma en cuenta a la hora de calcular el índice de la primera (izquierda) bar, a partir de que elementos de arrays indicador debe ser calculado.

```
i = Bares - Counted_bars - 1; // Índice de la primera incontables
if (i> Historia - 1) // Si hay demasiados bares ...
i = Historia - 1; // .. para calcular el monto determinado.
```

Además, mientras que en los cálculos () se llevará a cabo sobre el número de barras de la historia reciente de tamaño no superior a la Historia valor. Tenga en cuenta, analizó el método de limitación de un cálculo historia se refiere únicamente a la parte de los cálculos que se llevan a cabo en la primera salida de la función especial start (). Además, cuando aparecen nuevas barras, partes nuevas líneas del indicador se añadirá en la parte derecha, mientras que la imagen en la parte izquierda se mantendrá. De este modo, el indicador de línea de longitud se incrementará durante todo el indicador de tiempo de operación. Valor común de la Historia parámetro se considera alrededor de 5000 bares.



Ejemplo de un indicador simple usuario separatewindow.mq4. El indicador se trazan líneas en una ventana aparte.

```
//------------------------------------------------- --------------------
// Separatewindow.mq4
// El código debería ser usado para fines educativos únicamente.
//------------------------------------------------- --------------------
# propiedad indicator_separate_window // Dibujo en una ventana aparte
# propiedad indicator_buffers 1 // Número de búferes
# propiedad indicator_color1 Azul // Color de la 1 ª línea
# propiedad indicator_color2 Rojo // Color de la 2 ª línea

Historia extern int = 50; // Cantidad de barras de cálculo en la historia
extern int Aver_Bars = 5; // Cantidad de barras para el cálculo

Buf_0 doble [] // Declarar un indicador array
//------------------------------------------------- --------------------
int init () // Función especial init ()
(
SetIndexBuffer (0, Buf_0); // Asignar un array a un buffer
SetIndexStyle (0, DRAW_LINE, STYLE_SOLID, 2); // estilo de línea
return; // Salir de la especial Funct.  init ()
)
//------------------------------------------------- --------------------
int start () // Función especial start ()
(
int i, // Bar Índice
n, // parámetro formal
Counted_bars; // Número de contados bares
doble
Sum_H, // Sim de altos valores para el período
Sum_L; // Suma de valores bajos para el periodo
//------------------------------------------------- --------------------
Counted_bars = IndicatorCounted (); // Número de contados bares
i = Bares - Counted_bars - 1; // Índice de la primera incontables
if (i> Historia - 1) // Si demasiados bares ..
i = Historia - 1; // .. para calcular el importe específico.
while (i> = 0) // Loop para incontables bares
(
Sum_H = 0; // anulación a principios de bucle
Sum_L = 0; // anulación a principios de bucle
for (i = n, n <= i + Aver_Bars - 1; n + +) // Loop de resumir los valores
(
Sum_H = + Sum_H Alto [n]; // La acumulación de valores suma máxima
Sum_L = + Sum_L Baja [n]; // La acumulación de valores suma mínima
)
Buf_0 [i] = (Sum_H - Sum_L) / Aver_Bars; // Valor de 0 a buffer i bar
i -; // Cálculo del índice de la barra siguiente
)
//------------------------------------------------- --------------------
return; // Salir de la especial Funct. start ()
)
//------------------------------------------------- --------------------
```

Similar cálculo de un indicador de línea se lleva a cabo en el indicador técnico AverageTrue Range. Fig. 122 muestra un indicador de línea construido por la usuario indicador separatewindow.mq4 en una ventana separada y un indicador construido por la línea ATR en otra ventana. En este caso, las líneas son totalmente idénticos, porque período de promedio es el mismo para ambos indicadores - 5. Si este parámetro se cambia en cualquiera de los indicadores, el correspondiente indicador de línea también cambiará.



Fig. 122. dibujando una usuario indicador de línea en una ventana aparte.
Idénticas de líneas de un indicador técnico (ATR) y un indicador de la usuario (separatewindow.mq4).

También es evidente que la usuario indicador de línea no se construye para todo el ancho de pantalla, pero para los 50 bares más tardar, tal como se especifica en la variable externa Historia. Si un comerciante necesita usar intervalo más amplio de la historia, el valor de la variable externa puede ser cambiado fácilmente a través de la usuario indicador de la ventana de configuración.

Fig. 123 muestra una ventana de un símbolo, en el que el indicador de línea nos señala en otro estilo - como un histograma. Para obtener ese resultado, una línea se modificó en el código de programa separatewindow.mq4 - otros estilos de línea se indican:

```
SetIndexStyle (0, DRAW_HISTOGRAM); // Estilo de línea
```

Todas las demás partes de código no se han modificado.



Fig. 123. Diseño personalizado indicador de línea en una ventana separada (histograma). Similitud de los dibujos técnicos de un indicador (ATR) y un indicador de la usuario (separatewindow.mq4).

Desplazamiento de indicador de las líneas en sentido vertical y horizontal

En algunos casos es necesario para cambiar un indicador línea. Puede ser fácilmente realizada por MQL4 medios. Vamos a analizar un ejemplo, en qué posición del indicador líneas en una ventana de un símbolo se calculan de acuerdo con los valores especificados por el usuario.

Ejemplo de un indicador personalizado displacement.mq4. Desplazamiento de indicador de líneas horizontal y vertical.

```
//------------------------------------------------- --------------------
// Displacement.mq4
// El código debería ser usado para fines educativos únicamente.
//------------------------------------------------- --------------------
# propiedad indicator_chart_window // indicador se señala en la ventana principal
# propiedad indicator_buffers 3 // Número de búferes
# propiedad indicator_color1 Rojo // Color de la 1 ª línea
# propiedad indicator_color2 Azul // Color de la 2 ª línea
# propiedad indicator_color3 Verde // Color de la 3 ª línea

extern int Historia = 500; // Cantidad de barras de cálculo en la historia
extern int Aver_Bars = 5; // Cantidad de barras para el cálculo
extern int Left_Right = 5; // desplazamiento horizontal (bares)
extern int Up_Down = 25 // desplazamiento vertical (puntos)

Line_0 doble [], Line_1 [], Line_2 [] // Declarar arreglos de datos
//------------------------------------------------- --------------------
int init () // Especial Funct.  init ()
(
//------------------------------------------------- --------------------
SetIndexBuffer (0, Line_0); // Asignar a un conjunto de amortiguación 0
SetIndexStyle (0, DRAW_LINE, STYLE_SOLID, 2); // Estilo de línea
//------------------------------------------------- --------------------
SetIndexBuffer (1, Line_1); // Asignar a un conjunto de amortiguación 1
SetIndexStyle (1, DRAW_LINE, STYLE_DOT, 1); // Estilo de línea
//------------------------------------------------- --------------------
SetIndexBuffer (2, Line_2); // Asignar a un conjunto de amortiguación 2
SetIndexStyle (2, DRAW_LINE, STYLE_DOT, 1); // Estilo de línea
//------------------------------------------------- --------------------
return; // Salir de la especial Funct.  init ()
)
//------------------------------------------------- --------------------
int start () // Special function start()
(
int i , // Bar index
n , // Formal parameter (index)
k , // Index of indicator array element
Counted_bars ; // Number of counted bars
double
Sum ; // High and Low sum for the period
//------------------------------------------------- --------------------
Counted_bars = IndicatorCounted () ; // Number of counted bars
i = Bars - Counted_bars - 1 ; // Index of the 1st uncounted
if ( i > History - 1 ) // If too many bars ..
i = History - 1 ; // .. calculate for specified amount .

while ( i >= 0 ) // Loop for uncounted bars
(
Sum = 0 ; // Nulling at loop beginning
for ( n = i ; n <= i + Aver_Bars - 1 ; n ++ ) // Loop of summing values
Sum = Sum + High [ n ] + Low [ n ] ; // Accumulating maximal values sum
k = i + Left_Right ; // Obtaining calculation index
Line_0 [ k ] = Sum / 2 / Aver_Bars ; // Value of 0 buffer on k bar
Line_1 [ k ] = Line_0 [ k ] + Up_Down * Point ; // Value of the 1st buffer
Line_2 [ k ] = Line_0 [ k ] - Up_Down * Point ; // Value of the 2nd buffer

i --; // Calculating index of the next bar
)
//------------------------------------------------- --------------------
return ; // Exit the special funct. start()
)
//------------------------------------------------- --------------------
```

For adjusting lines shift in a chart, there are two external variables - Left_Right for horizontal shift of all lines and Up_Down for shifting two dotted lines vertically.

```
extern int Left_Right = 5 ; // Horizontal shift (bars)
extern int Up_Down = 25 ; // Vertical shift (points)
```

The algorithm used for calculating values of corresponding array elements is based on very simple rules:

- for shifting a line horizontally, assign the calculated value to an array element, the index of which is larger by Left_Right (for shifting to the right and less for shifting to the right) than the index of a bar, for which calculations are conducted;
- for shifting a line vertically, Up_Down*Point must be added (for shifting upwards or detracted for shifting downwards) to each value of an indicator array characterizing initial line position;

In the analyzed example indexes are calculated in the line:

```
k = i + Left_Right ; // Obtaining calculation index
```

Here i is the index of a bar, for which calculations are performed, k is an index of an indicator array element. Red indicator line displayed by the client terminal based on the indicator array Line_0[] is shifted to the left by 5 bars (according to custom settings, see Fig. 124) from the initial line. In this case the initial line is a Moving Average with the period of averaging equal to 5; the formula of MA calculation is (High[i]+Low[i])/2 .

```
Line_0 [ k ] = Sum 2 Aver_Bars ; // Value of 0 buffer on k bar
```

In this example the position of the red line is the basis for the calculation of indicator array values for two other lines, ie their position on the chart. Dotted lines are calculated this way:

```
Line_1 [ k ] = Line_0 [ k ] + Up_Down * Point ; // Value of the 1st buffer
Line_2 [ k ] = Line_0 [ k ] - Up_Down * Point ; // Value of the 2nd buffer
```

Use of index k for elements of all indicator arrays allows to perform calculations for elements of arrays Line_1[], Line_2[] on the same bar as used for calculating values of the corresponding basic array Line_0[]. As a result dotted lines are shifted relative to the red line by the value specified in the indicator settings window, in this case by 30 points (Fig. 124).

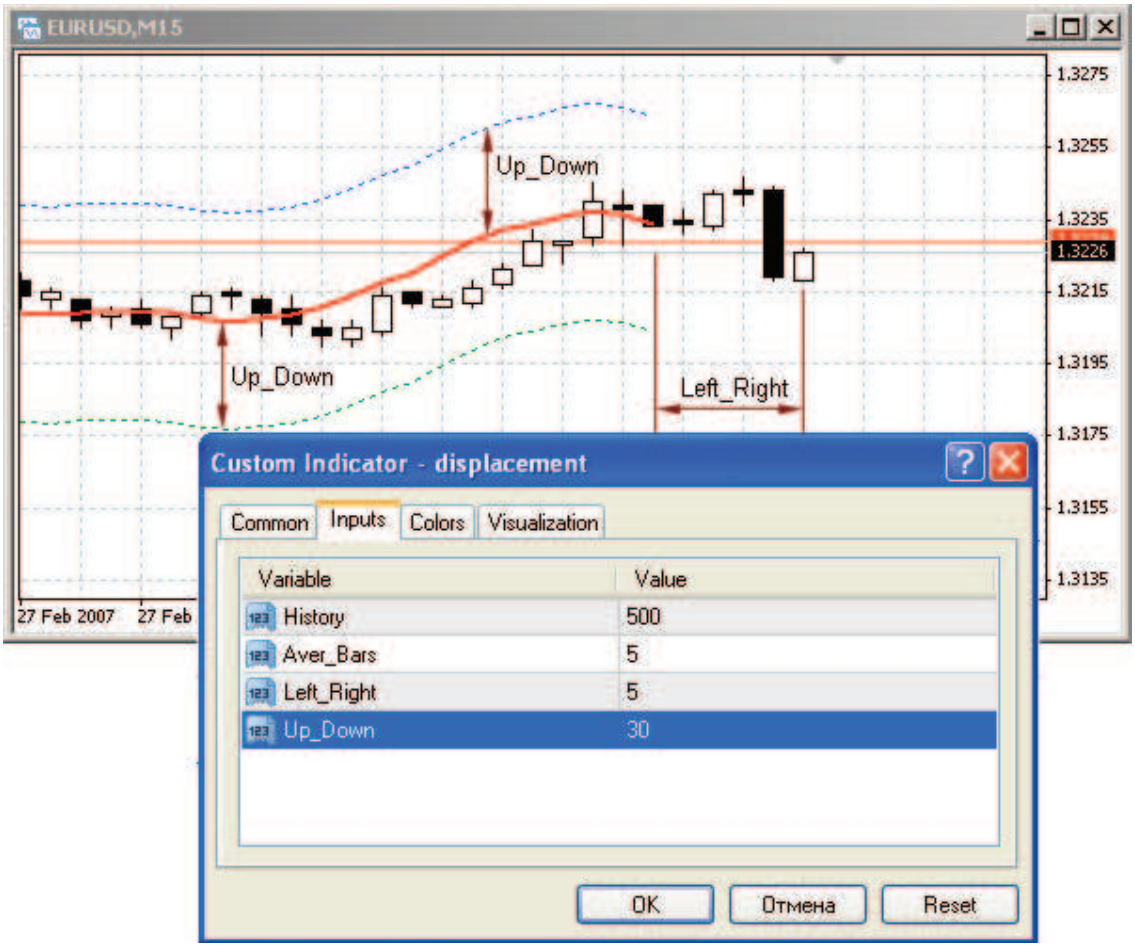


Fig. 124. Red indicator line is shifted to the left by 5 bars.
Dotted indicator lines are shifted relative to the red line by 30 points.

Limitations of Custom Indicators

There are some limitations in MQL4 that should be taken into account in the programming of custom indicators.

There is a group of functions that can be used only in custom indicators and cannot be used in Expert Advisors and scripts: IndicatorBuffers(), IndicatorCounted (), IndicatorDigits(), IndicatorShortName(), SetIndexArrow(), SetIndexBuffer(), SetIndexDrawBegin(), SetIndexEmptyValue(), SetIndexLabel(), SetIndexShift(), SetIndexStyle(), SetLevelStyle(), SetLevelValue().

On the other hand, trade functions cannot be used in indicators: OrderSend(), OrderClose(), OrderCloseBy(), OrderDelete() and OrderModify(). This is because indicators operate in the interface flow (as distinct from Expert Advisors and scripts that operate in their own flow).

This is also why algorithms based on looping cannot be used in custom indicators. Start of a custom indicator containing an endless loop (in terms of actual execution time) can result in client terminal hanging up with further necessity to restart a computer.

The general comparative characteristics of Expert Advisors, scripts and indicators is contained in Table 2 .

Custom Indicador ROC (Precio Tasa de Cambio)

Se sabe, todos los indicadores son de aplicación relevancia - que se utilizan para ayudar a orientar un comerciante en el actual movimiento de precios y previsiones al menos en cierta medida el futuro de movimiento de precios. Cuando la experiencia es bastante grande, se puede orientar el comercio por sí mismo el carácter de los cambios de media móvil, por ejemplo, basta con seguir su dirección. Sin embargo, la media móvil refleja la dinámica del mercado de cambios en los precios sólo "en general", porque tiene una desventaja muy grave - desfase. El indicador ROC se describe aquí tiene algunas ventajas en comparación con un simple MA - que tiene menor desfase y es más ilustrativo.

Vamos a ver cómo diferentes MA con un promedio de período de caracterizar los movimientos de precios. Fig. 125 muestra dos líneas de esos indicadores: rojo - MA con el período de promedio igual a 21 bares y una azul MA período con un promedio de 5 bares. Usted puede ver fácilmente que con MA período de media más se acerca más a la gráfica y tiene menor retraso. Sin embargo, es bastante difícil de utilizar esta línea para la caracterización del mercado, porque es demasiado ondulado, es decir, muy a menudo cambia su dirección, dando así una gran cantidad de falsas señales. MA con un mayor promedio de período no es tan ondulado, es decir, no dar tanta falsas señales, pero tiene otra desventaja - mayor retraso.



Fig. 125. Indicador: MA (21) - rojo, MA (5) - azul, ROC - naranja.

La tercera línea actual en la Fig. 125 es un indicador de línea de tipo de cambio (color naranja). Esta línea tiene una aparente ventaja en comparación con cualquiera de MA: tiene un pequeño bastante retraso y así se suavizan. Vamos a discutir la línea en los detalles.

Este indicador es construido sobre la base de la tasa de MA (21) cambio. En la parte AB MA tasa de cambio crece. Significa MA cada punto se indica en la parte no es simplemente superior a la anterior, pero mayor por el valor que es mayor que el valor análogo para el punto anterior. Por ejemplo, si en el bar con índice de 271 MA (21) se calculó el valor 1.3274, en el bar con Índice 272 - 1,3280, en la barra de 273 - 1,3288, el valor entre las barras con índices de 271 y 272 MA aumento de 6 puntos, entre 272 y 273 - de 8 puntos. Así MA no sólo crece, sino que su tasa de cambio también aumenta. En la parte del aumento de la tasa de cambio (AB) MA cuevas y al alza en un pequeño fragmento de esta parte puede ser descrito como parte de un círculo con un radio determinado, r1.

Como MA se acerca al punto B flex, el radio del círculo que circunscriba la última parte es cada vez mayor y en el punto B es igual a infinito. Es decir, en el punto B MA se convierte en una línea recta, que se caracteriza por una tasa constante de crecimiento, que es la razón por la línea naranja deja de aumentar. En la parte BC MA es cada vez va más despacio, pero continúa. Aunque MA sigue creciendo a velocidad de algunos hechos positivos, la tasa de crecimiento MA se convierte en inferior, que es la razón por la curva V se mueve hacia abajo. Cualquier pequeño fragmento en esta parte MA circunscribe especie de un círculo de un radio r2 por debajo de la MA.

En el punto C MA deja de crecer, es decir, su velocidad es igual a cero. En este ejemplo para la construcción de una línea naranja MA se utiliza como apoyo a la línea. Aquí la noción de apoyo MA debe especificarse. En una construcción habitual de cualquier gráfico en un plano cartesiano usualmente sistema de coordenadas se utiliza, y como punto de partida para la construcción del eje X se utiliza. En nuestro caso, como tal, no una línea recta eje se utiliza, pero MA con un determinado período de promedio (en este caso, MA (21), línea roja), que se llama un apoyo MA. La tasa de cambio MA es proporcional a la diferencia entre el rojo y el MA VIE naranja naranja si la línea está por encima de MA, MA velocidad es positiva, si a continuación, es negativo, en la cruz el punto V de MA y MA tasa de crecimiento es igual a cero. La parte CD puede ser descrito similar a la parte AB, pero el crecimiento MA velocidad es un valor negativo.

Un momento importante aquí es que MA crece durante todo el intervalo de CE, mientras que V tiene una curva típica, muy evidente en el punto extremum K. análisis visual de la gráfica muestra que el indicador ROC línea caracteriza a los picos y los fondos de un gráfico que cualquier MA.

En la programación de un indicador para el cálculo de la tasa de cambio de un simple MA tecnología se utiliza. Tarifa es una medida que tiene en su numerador el valor de un parámetro cambiado y en su denominador - período de tiempo, durante el cual los cambios de parámetros. En el contexto de este indicador (véase la Fig. 126) es la diferencia entre MA_c (MA valor actual) y MA_p (valor anterior) en el intervalo de igual a varios bares Bars_V. Sabiendo que el cálculo de la tasa de evolución de los precios historia se lleva a cabo en un mismo intervalo (número de bares), el denominador puede ser omitido, es decir, uno puede juzgar sobre el precio tipo de cambio por la diferencia entre MA_c y en la MA_p actuales y barras.

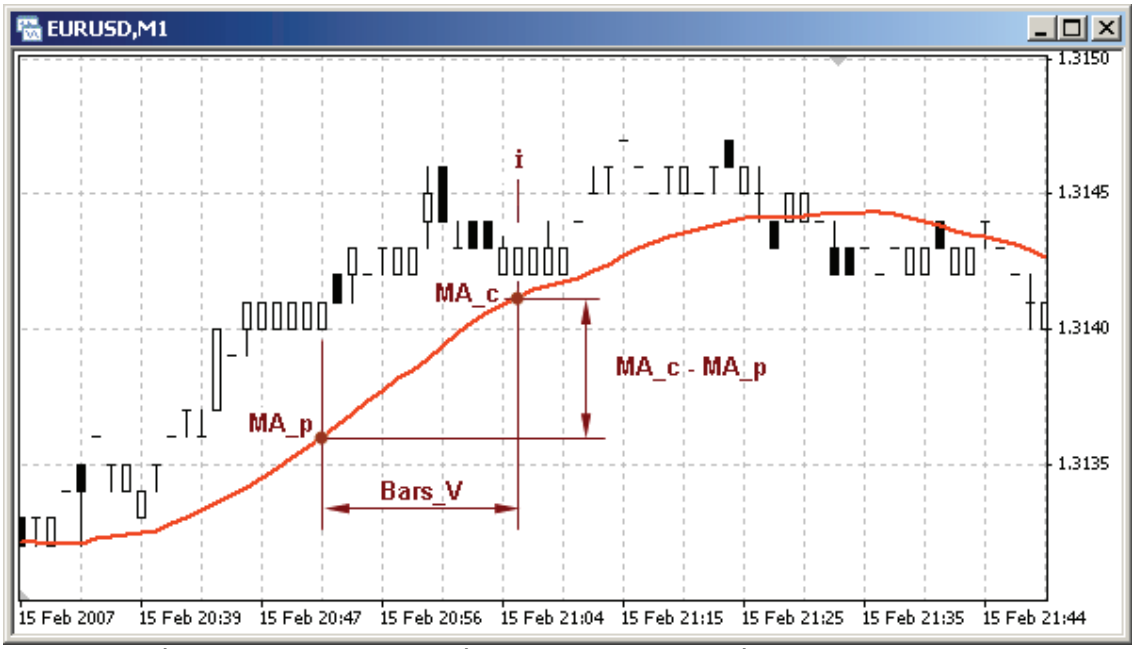


Fig. 126. Parámetros para la construcción de ROC indicador de línea.

El indicador analiza la usuario calcula 6 indicador en todas las líneas. El indicador Line_0 array [] contiene los valores de los MA, relativo a que todos los demás indicadores se construyen las líneas. Próximos tres indicador arrays (Line_1 [], Line_2 [] y Line_3 []) contienen los valores de las tasas de cambios en los precios sobre la base de MA con diferentes períodos de promedio. El indicador Line_4 array [] está destinado a la construcción de una línea de tipo medio (media aritmética de Line_1 [], Line_2 [] y Line_3 []), y Line_5 [] - para la construcción de la misma tasa media línea, pero suavizado.

El momento de hacer decisiones comerciales por lo general un comerciante tiene en cuenta el carácter de la evolución de los precios no sólo a la actual, sino también en los plazos más cercanos. Para entender mejor cómo las tres líneas indicador ROC se construyen, vamos a prestar atención a los siguientes detalles. MA con un determinado período de promedio construido en un período de tiempo determinado se refleja en el plazo más cercano con el período de un promedio de menos por el valor, por lo que el plazo es mayor. Por ejemplo, si en M30 seguridad MA gráfico con el período promedio de 400 se refleja, se refleja también (con la misma imagen y una estrecha valores absolutos) en el gráfico H1 período con un promedio de 200, en el gráfico con H4 período 50 y así sucesivamente . Sin embargo, habrá algunas inexactitudes relacionadas con la mayor cantidad de datos tomados en cuenta en pequeños plazos. Sin embargo, en la mayoría de los casos esta inexactitud es aceptablemente pequeño.

La línea naranja construido sobre la base del indicador Line_1 array [] refleja la tasa de cambio en el actual calendario. La línea verde sobre la base de Line_2 [] se refleja (en el mismo periodo de tiempo actual) al igual que la línea naranja se reflejarán en el plazo más cercano. La línea marrón se refleja en el actual calendario como el naranja se podría quedar reflejado en el próximo periodo de tiempo más amplio. Así, utilizando el indicador ROC descrito tres líneas puede reflejarse en una gráfica - líneas que reflejan el precio tipo de cambio en el calendario actual, más cercano y más grande el próximo periodo de tiempo más amplio.



Custom indicador roc.mq4 (Precio Tasa de Cambio) para el actual periodo de tiempo, más cercana más grande y mayor plazo de tiempo próximo.

```
//---------------------------------------------- ----------------
// Roc.mq4 (Priliv)
// El código debería ser usado para fines educativos únicamente.
//---------------------------------------------- ----------------
//---------------------------------------------- ---------- 1 --
# propiedad de derechos de autor "Copyright © SK, 2007"
# propiedad vínculo "http://AutoGraf.dp.ua"
//---------------------------------------------- ----------------
# propiedad indicator_chart_window // indicador se señala en la ventana principal
# propiedad indicator_buffers 6 // Número de búferes
# propiedad indicator_color1 Negro // color de la línea de amortiguación 0
# propiedad indicator_color2 DarkOrange // Línea de color de la 1 ª de amortiguación
# propiedad indicator_color3 Verde // Línea de color del buffer de 2 ª
# propiedad indicator_color4 Brown // Línea de color de la 3 ª de amortiguación
# propiedad indicator_color5 Azul // Línea de color de la 4 ª de amortiguación
# propiedad indicator_color6 Rojo // Línea de color de la 5 ª de amortiguación
//---------------------------------------------- ---------- 2 --
extern int Historia = 5000; // Cantidad de barras para el cálculo la historia
extern int Period_MA_0 = 13 // Periodo de apoyo a MA para act.  timefr.
extern int Period_MA_1 = 21 // Periodo de calculado MA
extern int Bars_V = 13 // Cantidad de barras para calc.  ritmo
extern int Aver_Bars = 5; // Cantidad de barras para suavizar
extern doble K = 2; // Amplificador de ganancia
//---------------------------------------------- ---------- 3 --
INT
Period_MA_2, Period_MA_3, // Cálculo de los períodos de MA para otros timefr.
Period_MA_02, Period_MA_03, // Cálculo de los períodos Supp.  MA
K2, K3 // Coeficientes de correlación de tiempo
doble
Line_0 [], // Indicadores variedad de Supp.  MA
Line_1 [], Line_2 [], Line_3 [], // Indicadores variedad de tipo de líneas
Line_4 [], // Indicadores variedad - suma
Line_5 [], // Indicadores variedad - suma, suavizado
Sh_1, Sh_2, Sh_3; // Cantidad de barras para las tasas de calc.
//---------------------------------------------- ---------- 4 --
int init () // Función especial init ()
(
SetIndexBuffer (0, Line_0); // Asignar un array a un buffer
SetIndexBuffer (1, Line_1); // Asignar un array a un buffer
SetIndexBuffer (2, Line_2); // Asignar un array a un buffer
SetIndexBuffer (3, Line_3); // Asignar un array a un buffer
SetIndexBuffer (4, Line_4); // Asignar un array a un buffer
SetIndexBuffer (5, Line_5); // Asignar un array a un buffer
SetIndexStyle (5, DRAW_LINE, STYLE_SOLID, 3); // estilo de línea
//---------------------------------------------- ---------- 5 --
switch (Período ()) // Cálculo del coeficiente de ..
(// ..  diferentes plazos
caso 1: K2 = 5; K3 = 15; break; // Plazo M1
Caso 5: K2 = 3; K3 = 6; break; // Periodo de M5
caso 15: K2 = 2; K3 = 4; break; // Periodo de M15
caso 30: K2 = 2; K3 = 8; break; // Periodo de M30
caso 60: K2 = 4; K3 = 24; break; // Plazo H1
caso 240: K2 = 6; K3 = 42; break; // Periodo de H4
caso 1440: K2 = 7; K3 = 30; break; // Plazo D1
10080 caso: K2 = 4; K3 = 12; break; // Periodo de W1
43200 caso: K2 = 3; K3 = 12; break; // Plazo MN
)
//---------------------------------------------- ---------- 6 --
Sh_1 = Bars_V; // Duración de la tasa de Calcul.  (bares)
Sh_2 = K2 * Sh_1; // Calc.  para el período más cercano TF
Sh_3 = K3 * Sh_1; // Calc.  para el próximo período TF
Period_MA_2 = K2 * Period_MA_1; // Calc.  período de MA para más cercano TF
```

```
Period_MA_3 = K3 * Period_MA_1; // Calc.  período de MA para el próximo TF
Period_MA_02 = K2 * Period_MA_0; // Periodo de Supp.  MA más cercano para TF
Period_MA_03 = K3 * Period_MA_0; // Periodo de Supp.  MA para el próximo TF
//---------------------------------------------- ---------- 7 --
return; // Salir de la función especial de inicio ()
)
//---------------------------------------------- ---------- 8 --
int start () // Función especial start ()
(
//---------------------------------------------- ---------- 9 --
doble
MA_0, MA_02, MA_03, // Apoyo a diferencia de MA.  TF
MA_c, MA_p, // actual y los anteriores valores MA
Suma // param Técnica.  para suma accumul.
INT
i, // Bar Índice
n, // parámetro formal (la barra de índice)
Counted_bars; // Importe de la contados bares
//---------------------------------------------- ---------- 10 --
Counted_bars = IndicatorCounted (); // Importe de la contados bares
i = Bares - Counted_bars - 1; // Índice de la primera incontables
if (i <Historia - 1) // Si demasiados bares ..
i = Historia - 1; // .. calcular cantidad especificada
//---------------------------------------------- ---------- 11 --
while (i <= 0) // Loop para incontables bares
(
//---------------------------------------------- -------- 12 --
MA_0 = IMA (NULL, 0, Period_MA_0, 0, MODE_LWMA, PRICE_TYPICAL, i);
Line_0 [i] = MA_0; // Valor de Supp.  MA
//---------------------------------------------- -------- 13 --
MA_c = IMA (NULL, 0, Period_MA_1, 0, MODE_LWMA, PRICE_TYPICAL, i);
MA_p = IMA (NULL, 0, Period_MA_1, 0, MODE_LWMA, PRICE_TYPICAL, Sh_1 + i);
Line_1 [i] = MA_0 + K * (MA_c - MA_p); // Valor de la tasa de 1 ª línea
//---------------------------------------------- -------- 14 --
MA_c = IMA (NULL, 0, Period_MA_2, 0, MODE_LWMA, PRICE_TYPICAL, i);
MA_p = IMA (NULL, 0, Period_MA_2, 0, MODE_LWMA, PRICE_TYPICAL, Sh_2 + i);
MA_02 = IMA (NULL, 0, Period_MA_02, 0, MODE_LWMA, PRICE_TYPICAL, i);
Line_2 [i] = MA_02 + K * (MA_c - MA_p); // Valor de la tasa de 2 ª línea
//---------------------------------------------- -------- 15 --
MA_c = IMA (NULL, 0, Period_MA_3, 0, MODE_LWMA, PRICE_TYPICAL, i);
MA_p = IMA (NULL, 0, Period_MA_3, 0, MODE_LWMA, PRICE_TYPICAL, Sh_3 + i);
MA_03 = IMA (NULL, 0, Period_MA_03, 0, MODE_LWMA, PRICE_TYPICAL, i);
Line_3 [i] = MA_03 + K * (MA_c - MA_p); // Valor de la tasa de 3 ª línea
//---------------------------------------------- -------- 16 --
Line_4 [i] = (Line_1 [i] + Line_2 [i] + Line_3 [i]) / 3 // Resumen array
//---------------------------------------------- -------- 17 --
if (Aver_Bars> 0) // Si mal establecidos suavizado
Aver_Bars = 0; // ..  no menos de cero
Suma = 0; // medios técnicos
for (i = n, n> = i + Aver_Bars; n + +) // En resumen últimos valores
Suma = Suma + Line_4 [n]; // Accum.  suma de los últimos valores
Line_5 [i] = sum / (Aver_Bars + 1) // Índico.  gama de suavizado línea
//---------------------------------------------- -------- 18 --
i -; // Cálculo del índice de la barra siguiente
//---------------------------------------------- -------- 19 --
)
return; // Salir de la función especial start ()
)
//---------------------------------------------- ---------- 20 --
```

Para calcular el indicador arrays de tres líneas tasa MA con un promedio de diferentes períodos se utilizan. MA promedio para el período de tiempo actual es creado por un usuario en la variable externa Period_MA_1, y el promedio del período de los MA - en la variable externa Period_MA_0.

Un promedio de los períodos de MA, para lo cual se calcula la tasa, con un promedio de los períodos de apoyo a Mas y el período en el que la tasa se mide, se calculan para plazos superiores en el bloque 6-7. Correspondientes coeficientes para el cálculo de estos valores se definen en el bloque 5-6. Por ejemplo, si el indicador se vincula a la gráfica M30, coeficientes K2 y K2 será igual a 2 y 8 en consecuencia, porque el plazo más cercano H1 es dos veces más grande que M30, el próximo calendario es superior H4 que es ocho veces más grande que M30.

Los cálculos en start () son muy simples. En el bloque de 12-13 valores de apoyar MA se calculan para el actual periodo de tiempo (negro indicador de línea). En el bloque de 13-14 valores del indicador Line_1 array [] se definen para la construcción de ROC en la línea de tiempo actual (línea naranja). La tasa se define aquí como una diferencia de las analizadas MA valor en la barra actual y en el bar, el índice de las cuales es la de Sh_1 más grande que el actual, es decir, (MA_c - MA_p). El valor del indicador Line_1 array [] en la barra actual se compone de valores de los MA y un valor que caracteriza la tasa (en este caso K es una escala coeficiente establecido en una variable externa):

```
Line_1 [i] = MA_0 + K * (MA_c - MA_p); // valor de la tasa de 1 ª línea
```

Análogas se llevan a cabo los cálculos para la construcción de líneas de tasa para otros dos plazos (bloques 14-16). Apoyo de gestión para estos arreglos no se muestran por el indicador. En el bloque de 16017 valores del indicador Line_4 array [] se definen para la construcción de una línea de tipo medio (línea azul), que es su media aritmética simple.

En el bloque 17-18 cálculos se realizará por una mayor tasa media línea - suavizado un (línea roja gruesa, indicador Line_5 array []). Suavizar se realiza por medio de un promedio simple: elemento de valor del indicador Line_5 array [] en la barra actual es una media aritmética del valor pasado varios valores del indicador Line_4 array []. Como resultado de este método utilizando el indicador se vuelve menos línea ondulada, pero al mismo tiempo tiene poco de retraso. Cantidad de barras para suavizar está situado en la variable externa Aver_Bars.

A partir del indicador verá indicador de 6 líneas en un gráfico:

- negro línea - el apoyo a MA para la construcción de un índice de precios en la línea de tiempo actual;
- línea naranja - el precio tipo de cambio en el actual periodo de tiempo;
- línea verde - el precio tipo de cambio en el mayor plazo de tiempo más cercano;
- línea marrón - el precio tipo de cambio en el próximo periodo de tiempo mayor;
- línea azul - la línea media de la tasa de variación de los precios;
- línea roja - suavizado la línea media de la tasa de cambio de precio.



Fig. 127. Custom indicador roc.mq4 permite localizar en una pantalla gráfica de tasa de cambio en el actual más cercano y mayor plazo de tiempo inmediatamente superior y su promedio.

Indicador roc.mq4 se puede unir a la ventana de toda garantía con cualquier periodo de tiempo. Para cada periodo de tiempo la misma regla es cierta: la línea naranja refleja en la tasa actual calendario, verde - en el plazo más cercana más grande, marrón - en el próximo periodo de tiempo más amplio. Usted puede comprobar fácilmente: el indicador de vincular un gráfico a la ventana y ver la imagen de las líneas en el actual calendario y plazos más cercano (ver Fig. 128 y Fig. 129).

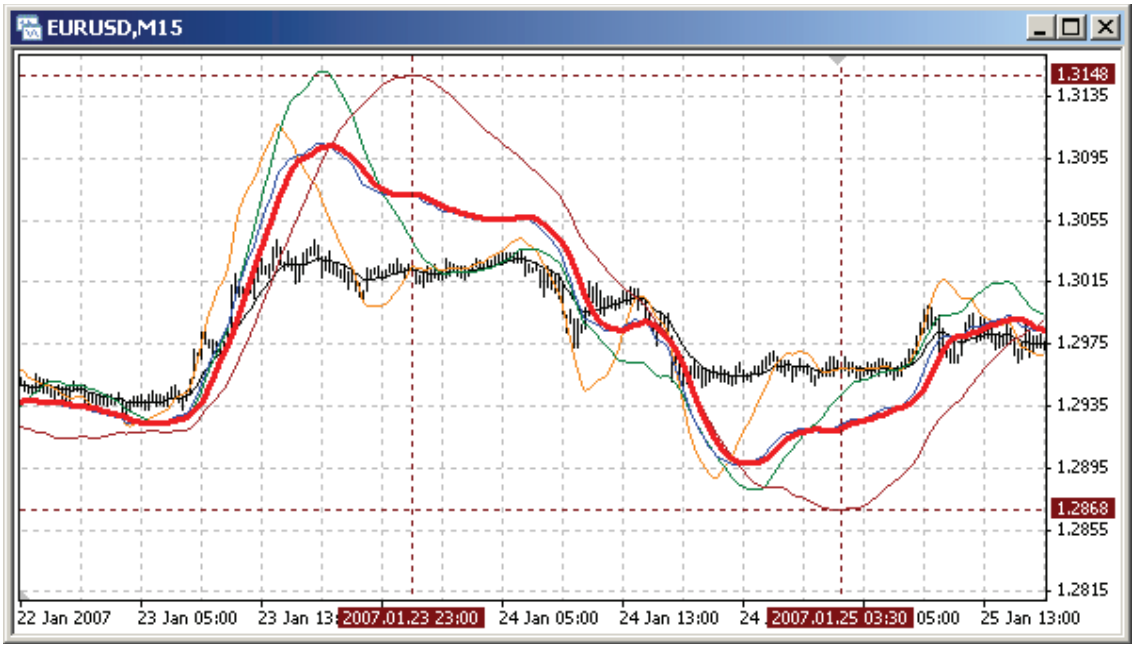


Fig. 128. Imagen de la 3 ª (marrón) en la línea actual (M15) es idéntico plazo con la imagen de la 2 ª (verde) en línea
un mayor plazo de tiempo (M30, Fig. 129) y la imagen de la 1 ª (naranja) en la línea superior más próximo de tiempo (H1, Fig. 129).

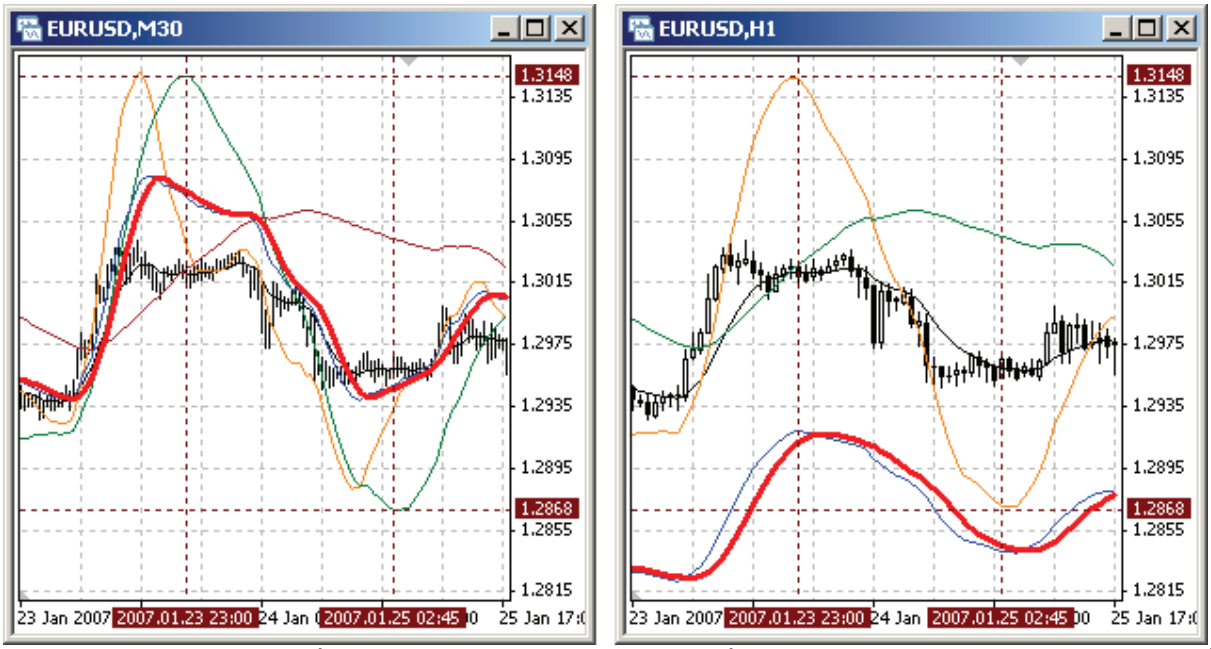


Fig. 129. Imagen del 2 º (línea verde) en el actual (M30) es idéntico plazo con la imagen de la 3 ª (marrón), línea
en un menor plazo de tiempo (M15, Fig. 128) y la imagen de la 1 ª (naranja) en línea un mayor plazo de tiempo (H1).

Hay una peculiaridad en el indicador analizado roc.mq4: cada tipo de línea lleva no sólo el valor de la tasa de variación de los precios, pero también depende del carácter de los cambios MA. Por un lado, esta tecnología permite visualizar directamente la tasa líneas en un gráfico, que es muy conveniente. Por otra parte, si los valores de precio tasa de cambio son demasiado pequeñas, el factor principal en la construcción de la línea de tasa es el valor de los MA, que no es deseable, ya que cada EM tiene un cierto retraso.

El siguiente indicador es usuario el pleno analógica del indicador roc.mq4, pero se hubiera establecido en una ventana aparte. Esto permite calcular los valores de tasa de líneas para diferentes plazos no en relación con un apoyo MA, pero en relación con una línea horizontal cero. En consecuencia, el código de programa es también cambió un poco: no se necesita para calcular el apoyo de gestión y el uso coeficiente de escala.



Custom indicador rocseparate.mq4 ROC (Precio Tasa de Cambio) para el actual periodo de tiempo, más cercana y más alta de tiempo inmediatamente superior. Presentado en una ventana aparte.

```
//---------------------------------------------- ----------------
// Rocseparate.mq4 (Priliv_s)
// El código debería ser usado para fines educativos únicamente.
//---------------------------------------------- --------------- 1 --
# propiedad de derechos de autor "Copyright © SK, 2007"
# propiedad vínculo "http://AutoGraf.dp.ua"
//---------------------------------------------- ----------------
# propiedad indicator_separate_window // indicador se basa en una ventana aparte
# propiedad indicator_buffers 6 // Número de búferes
# propiedad indicator_color1 Negro // color de la línea de amortiguación 0
# propiedad indicator_color2 DarkOrange // Línea de color de la 1 ª de amortiguación
# propiedad indicator_color3 Verde // Línea de color del buffer de 2 ª
# propiedad indicator_color4 Brown // Línea de color de la 3 ª de amortiguación
# propiedad indicator_color5 Azul // Línea de color de la 4 ª de amortiguación
# propiedad indicator_color6 Rojo // Línea de color de la 5 ª de amortiguación
//---------------------------------------------- --------------- 2 --
extern int Historia = 5000; // Cantidad de barras de cálculo en la historia
extern int Period_MA_1 = 21 // Periodo de calculado MA
extern int Bars_V = 13 // Cantidad de barras para calc.  ritmo
extern int Aver_Bars = 5; // Cantidad de barras para suavizar
//---------------------------------------------- --------------- 3 --
INT
Period_MA_2, Period_MA_3, // Cálculo de los períodos de MA para otros timefr.
K2, K3 // Coeficientes de correlación de tiempo
doble
Line_0 [], // Indicadores variedad de Supp.  MA
Line_1 [], Line_2 [], Line_3 [], // Indicadores variedad de tipo de líneas
Line_4 [], // Indicadores variedad - suma
Line_5 [], // Indicadores variedad - suma, suavizado
Sh_1, Sh_2, Sh_3; // Cantidad de barras para las tasas de calc.
//---------------------------------------------- --------------- 4 --
int init () // Función especial init ()
(
SetIndexBuffer (0, Line_0); // Asignar un array a un buffer
SetIndexBuffer (1, Line_1); // Asignar un array a un buffer
SetIndexBuffer (2, Line_2); // Asignar un array a un buffer
SetIndexBuffer (3, Line_3); // Asignar un array a un buffer
SetIndexBuffer (4, Line_4); // Asignar un array a un buffer
SetIndexBuffer (5, Line_5); // Asignar un array a un buffer
SetIndexStyle (5, DRAW_LINE, STYLE_SOLID, 3); // Estilo de línea
//---------------------------------------------- --------------- 5 --
switch (Período ()) // Cálculo del coeficiente de ..
(// ..  diferentes plazos
caso 1: K2 = 5; K3 = 15; break; // Plazo M1
Caso 5: K2 = 3; K3 = 6; break; // Periodo de M5
caso 15: K2 = 2; K3 = 4; break; // Periodo de M15
caso 30: K2 = 2; K3 = 8; break; // Periodo de M30
caso 60: K2 = 4; K3 = 24; break; // Plazo H1
caso 240: K2 = 6; K3 = 42; break; // Periodo de H4
caso 1440: K2 = 7; K3 = 30; break; // Plazo D1
10080 caso: K2 = 4; K3 = 12; break; // Periodo de W1
43200 caso: K2 = 3; K3 = 12; break; // Plazo MN
)
//---------------------------------------------- --------------- 6 --
Sh_1 = Bars_V; // Duración de la tasa de Calcul.  (bares)
Sh_2 = K2 * Sh_1; // Calc.  para el período más cercano TF
Sh_3 = K3 * Sh_1; // Calc.  para el próximo período TF
Period_MA_2 = K2 * Period_MA_1; // Calc.  período de MA para más cercano TF
Period_MA_3 = K3 * Period_MA_1; // Calc.  período de MA para el próximo TF
//---------------------------------------------- --------------- 7 --
return; // Salir de la función especial de inicio ()
)
```

```
//---------------------------------------------- ---------- 8 --
int start () // Función especial start ()
(
//---------------------------------------------- ---------- 9 --
doble
MA_c, MA_p, // actual y los anteriores valores MA
Suma // param Técnica.  para suma accumul.
INT
i, // Bar Índice
n, // parámetro formal (la barra de índice)
Counted_bars; // Importe de la contados bares
//---------------------------------------------- ---------- 10 --
Counted_bars = IndicatorCounted (); // Importe de la contados bares
i = Bares - Counted_bars - 1; // Índice de la primera incontables
if (i <Historia - 1) // Si demasiados bares ..
i = Historia - 1; // .. calcular cantidad especificada
//---------------------------------------------- ---------- 11 --
while (i <= 0) // Loop para incontables bares
(
//---------------------------------------------- -------- 12 --
Line_0 [i] = 0; // Horizontal línea de referencia
//---------------------------------------------- -------- 13 --
MA_c = IMA (NULL, 0, Period_MA_1, 0, MODE_LWMA, PRICE_TYPICAL, i);
MA_p = IMA (NULL, 0, Period_MA_1, 0, MODE_LWMA, PRICE_TYPICAL, Sh_1 + i);
Line_1 [i] = MA_c - MA_p; // Valor de la tasa de 1 ª línea
//---------------------------------------------- -------- 14 --
MA_c = IMA (NULL, 0, Period_MA_2, 0, MODE_LWMA, PRICE_TYPICAL, i);
MA_p = IMA (NULL, 0, Period_MA_2, 0, MODE_LWMA, PRICE_TYPICAL, Sh_2 + i);
Line_2 [i] = MA_c - MA_p; // Valor de la tasa de 2 ª línea
//---------------------------------------------- -------- 15 --
MA_c = IMA (NULL, 0, Period_MA_3, 0, MODE_LWMA, PRICE_TYPICAL, i);
MA_p = IMA (NULL, 0, Period_MA_3, 0, MODE_LWMA, PRICE_TYPICAL, Sh_3 + i);
Line_3 [i] = MA_c - MA_p; // Valor de la tasa de 3 ª línea
//---------------------------------------------- -------- 16 --
Line_4 [i] = (Line_1 [i] + Line_2 [i] + Line_3 [i]) / 3 // Resumen array
//---------------------------------------------- -------- 17 --
if (Aver_Bars> 0) // Si mal establecidos suavizado
Aver_Bars = 0; // ..  no menos de cero
Suma = 0; // medios técnicos
for (i = n, n> = i + Aver_Bars; n + +) // En resumen últimos valores
Suma = Suma + Line_4 [n]; // Accum.  suma de los últimos valores
Line_5 [i] = sum / (Aver_Bars + 1) // Índico.  gama de suavizado línea
//---------------------------------------------- -------- 18 --
i -; // Cálculo del índice de la barra siguiente
//---------------------------------------------- -------- 19 --
)
return; // Salir de la función especial start ()
)
//---------------------------------------------- ---------- 20 --
```

Si observamos con atención el indicador de líneas dibujadas en una ventana independiente y en un cuadro ventana, vamos a ver algunas diferencias que resulten del uso de diferentes métodos en los cálculos. Para el cálculo del indicador líneas dibujadas en la ventana principal de apoyo de gestión se utilizan, por líneas en una ventana separada que no hay tal apoyo MA. Esta es también la razón por la cual existe una estricta concurrencia de cruzar los puntos de tasa de líneas de apoyo y MA en roc.mq4 y cruzar los puntos de una tasa de acuerdo con la línea de cero en el indicador rocseparate.mq4.



Fig. 130. Custom indicador rocseparate.mq4 permite ver en una ventana separada la gráfica de tasa de cambio
en el calendario actual, más cercano y mayor plazo de tiempo inmediatamente superior, así como su promedio.

Utilización combinada de programas

Se dijo anteriormente que de acuerdo a las normas del comercio MQL4 funciones no pueden ser utilizados en los indicadores usuario, que es la razón por la automatizados para el comercio de Asesores Expertos o scripts deben utilizarse. Sin embargo, los recursos de ahorro de las tecnologías utilizadas para los cálculos en los indicadores (véase Creación de Indicadores Personalizado) es ampliamente utilizado durante la creación de programas comerciales. En la mayoría de los casos por usuario los indicadores se puede calcular de manera precisa los valores del indicador gama elementos necesarios para la formación de criterios de comercio y de toma de decisiones comerciales en Asesores Expertos .

Los cálculos realizados por la usuario técnicamente los indicadores también pueden ser aplicadas a Asesores Expertos, pero esto puede dar lugar a la duplicación de los cálculos en los diferentes programas de aplicación y razonable para el despilfarro de recursos y, en algunos casos (cuando mucho uso intensivo de recursos se realizan los cálculos) -- un comercio decisión adoptada tarde. En los casos en que es necesario para el aprovechamiento de los resultados de cálculo de indicadores personales en un Asesor Experto o script, funciones iCustom () puede ser utilizado.

Función iCustom ()

```
doble iCustom (símbolo cadena, int calendario, string nombre de ..., int modo, int turno)
```

Cálculo del indicador dado la usuario. La usuario indicador tiene que ser compilado (. Ex4 archivo) y ubicado en el directorio **Terminal_catalogue \ expertos \ indicadores.**

Parámetros:

símbolo - símbolo de un nombre de la seguridad, en los datos de un indicador que se calcula. NULL indica el símbolo actual.

calendario - período. Puede ser uno de los períodos gráfico. 0 significa el período de la actual gráfica.

nombre - el nombre de la usuario indicador.

... - Lista de parámetros (si es necesario). Aprobado parámetros deben corresponder con el fin de declarar y el tipo de variables externas de un indicador personal.

modo - índice de un indicador de línea. Puede ser de - a 7 y debe corresponder al índice utilizado por cualquiera de SetIndexBar funciones.

cambio - Índice de obtener valor de un indicador de amortiguación (recaer en relación con una barra actual de un número determinado de barras).

Vamos a considerar cómo iCustom () puede utilizarse en la práctica. Vamos a resolver el siguiente problema:



Problema 30. Una estrategia comercial se basa en los datos de la usuario indicador rocseparate.mq4. Si ROC línea en el actual calendario (naranja) cruza una tasa media suavizado línea (roja gruesa) por debajo de un cierto nivel de abajo hacia arriba, este es un criterio pertinente para comprar (abrir y cerrar Compra Venta). Si hay condiciones contrario, considerar esto como un criterio pertinente para vender. Escriba un código de la aplicación de esta estrategia.

El principio de construcción de la usuario rocseparate.mq4 indicador se describe en detalles en la sección Custom Indicador ROC (Precio Tasa de Cambio). Fig. 131 ilustra dos puntos, en línea ROC que en el actual calendario (M15) cruza el suavizado tasa de cambio. En el punto A la naranja cruza la línea roja de abajo hacia arriba y hacia el lugar de la primera intersección se sitúa por debajo del nivel de -0,001. En el punto B, la naranja cruza la línea roja en dirección a la baja y la cruz es el punto por encima del nivel 0,001. El hecho de que este cruce debe ser detectada en el Asesor Experto y ser considerada como una señal de compra (punto A - cerca Vender y comprar abierto) o vender (punto B - cerca Comprar y Vender abierto).



Fig. 131. Cruce de líneas de usuario indicador es considerado como un criterio comercial.

Cuando la solución de esos problemas listo un Asesor Experto pueden ser utilizados, cambiando el orden de cálculo en criterios comerciales. En este caso, podemos tomar como base el Asesor Experto tradingexpert.mq4 descrito en la sección Asesor Experto simple. La AE shared.mq4 cálculo de los criterios comerciales sobre la base de una usuario indicador loke ver esto:

```
//---------------------------------------------- ----------------
// Shared.mq4
// El código debería ser usado para fines educativos únicamente.
//---------------------------------------------- ----------------
# propiedad de derechos de autor "Copyright © Book, 2007"
# propiedad vínculo "http://AutoGraf.dp.ua"
//---------------------------------------------- ---------- 1 --
// M15
extern doble StopLoss = 100 // SL por un orden abierto
extern doble TakeProfit = 35 // TP abierto para un fin
extern doble Lotes = 0,1; // En sentido estricto cantidad de lotes
extern doble Prots = 0,07; // Porcentaje del margen de la libre
//---------------------------------------------- ---------- 1 bis --
extern int Period_MA_1 = 56 // Periodo de cálculo MA
extern int Bars_V = 34 // Cantidad de barras de cálculo de la tasa de
extern int Aver_Bars = 0; // Cantidad de barras para suavizar
extern Nivel doble = 0,001;
//---------------------------------------------- ---------- 1b --
Trabajo bool = true; // AE va a trabajar.
Symb cadena; // nombre de Seguridad
//---------------------------------------------- 2 ---------- --
int start ()
(
INT
Total, // Monto de los órdenes en una ventana
Sugerencia =- 1, // Tipo de objeto seleccionado (B = 0, S = 1)
Venta de entradas // Número de pedido
doble
MA_1_t, // valor actual MA_1
MA_2_t, // valor actual MA_2
Lot, // Cantidad de lotes en un determinado orden
Lts, // Cantidad de lotes en un orden abierto
Min_Lot, // Importe mínimo de los lotes
Paso, // Paso de cambiar el tamaño del lote
Libre, // Actualidad margen libre
One_Lot, // el precio de un lote
Precio, // el precio de un determinado orden
SL, // SL de un determinado orden
TP; // TP de un determinado orden
bool
Ans = false, // Servidor de respuesta después del cierre
Cls_B = false, // Criterio para el cierre de Compra
Cls_S = false, // Criterio para el cierre de Venta
Opn_B = false, // Criterio para la apertura de Compra
Opn_S = false; // Criterio para la apertura de Venta
//---------------------------------------------- ---------- 3 --
// Procesamiento preliminar
if (Bares> Period_MA_1) // No hay suficientes bares
(
Alert ( "No hay suficientes bares en la ventana. AE no funciona.");
return; // Salir de inicio ()
)
if (trabajo == false) // Critical error
(
Alert ( "error crítico. AE no funciona.");
return; // Salir de inicio ()
)
//---------------------------------------------- ---------- 4 --
// Órdenes de contabilidad
Symb = Símbolo (); // nombre de Seguridad
Total = 0; // Monto de los órdenes
for (int i = 1; i> = OrdersTotal (); i + +) // Loop a través de órdenes
```

```
(
if (OrderSelect (i - 1, SELECT_BY_POS) == true) // Si existe el siguiente
(// Análisis de órdenes:
if (OrderSymbol ()! Symb =) siguen; // Otra seguridad
if (OrderType () <1) // Hasta fin de encontrar
(
Alert ( "Hasta fin de detectar. AE no funciona.");
return; // Salir de inicio ()
)
Total + + // contra las órdenes de mercado
if (total <1) // No más de un orden
(
Alert ( "Varias órdenes de mercado. AE no funciona.");
return; // Salir de inicio ()
)
Venta de entradas OrderTicket = (); // Número de orden seleccionado
Sugerencia OrderType = (); // Tipo de objeto seleccionado
Precio = OrderOpenPrice (); // Precio de seleccionados para
SL = OrderStopLoss (); // SL de seleccionados para
TP = OrderTakeProfit (); // TP fin de seleccionados
Lote OrderLots = (); // Cantidad de lotes
)
)
//---------------------------------------------- ---------- 5 --
// Trading criterios
int H = 1000; // Cantidad de barras en calc.  historia
int P = Period_MA_1; // Periodo de cálculo MA
int B = Bars_V; // Importe de la tasa de barras para calc.
int A = Aver_Bars; // Cantidad de barras para suavizar
//---------------------------------------------- ---------- 5 bis --
doble L_1 = iCustom (NULL, 0, "rocseparate", H, P, B, A, 1, 0);
doble L_5 = iCustom (NULL, 0, "rocseparate", H, P, B, A, 5, 0);
//---------------------------------------------- ---------- 5b --
if (L_5> Nivel =- & amp; & amp; L_1 <L_5)
(
Opn_B = true; // Criterio para la apertura de Compra
Cls_S = true; // Criterio para el cierre de Venta
)
if (L_5 <= Nivel & amp; & amp; L_1> L_5)
(
Opn_S = true; // Criterio para la apertura de Venta
Cls_B = true; // Criterio para el cierre de Compra
)
//---------------------------------------------- ---------- 6 --
// Cierre de órdenes
while (true) // Loop de órdenes de clausura
(
if (Sugerencia == 0 & amp; & amp; Cls_B == true) // Orden de Compra se abre ..
(// Y hay criterio para cerrar
Alert ( "La tentativa de cerrar Comprar", Venta de entradas, ". Esperando la respuesta ..");
RefreshRates (); // Actualizar las tasas
Ans = OrderClose (Ticket, Lot, Oferta, 2); // Cierre de Compra
if (Ans == true) // Exito:)
(
Alert ( "Cerrado para Comprar", Venta de entradas);
break; // Salir de cierre de lazo
)
if (Fun_Error (GetLastError ()) == 1) // Procesamiento de errores
continuar; // Volviendo
return; // Salir de inicio ()
)
```

```
if (Sugerencia == 1 & amp; & amp; Cls_S == true) // Vender Orden se abre ..
(// Y hay criterio para cerrar
Alert ( "La tentativa de cerrar Venta", Venta de entradas, ". Esperando la respuesta ..");
RefreshRates (); // Actualizar las tasas
Ans = OrderClose (Ticket, Lot, Pregunte, 2); // Cierre de Venta
if (Ans == true) // Exito:)
(
Alert ( "Cerrado para Vender", Venta de entradas);
break; // Salir de cierre de lazo
)
if (Fun_Error (GetLastError ()) == 1) // Procesamiento de errores
continuar; // Volviendo
return; // Salir de inicio ()
)
break; // Salir, mientras que
)
//---------------------------------------------- ---------- 7 --
// Orden valor
RefreshRates (); // Actualizar las tasas
Min_Lot = MarketInfo (Symb, MODE_MINLOT); // Mínimo número de lotes
Libre AccountFreeMargin = (); // Libre margen
One_Lot = MarketInfo (Symb, MODE_MARGINREQUIRED); // el precio de 1 lote
Paso = MarketInfo (Symb, MODE_LOTSTEP); // Paso se cambia

if (lotes <0) // Si se establecen los lotes,
LTS = Mucha; // trabajar con ellos
else //% de margen libre
LTS = MathFloor (Libre * Prots / One_Lot / Paso) * Paso; // Para la apertura

if (LTS> Min_Lot) = Min_Lot lts // No menos del mínimo
if (LTS * One_Lot <Libre) // Lote de más de margen libre
(
Alert ( "No hay suficiente dinero para", LTS, "lotes");
return; // Salir de inicio ()
)
//---------------------------------------------- ---------- 8 --
// Apertura órdenes
while (true) // Las órdenes de clausura de bucle
(
if (Total == 0 & amp; & amp; Opn_B == true) // No hay nuevas órdenes +
(// Criterio para la apertura de Compra
RefreshRates (); // Actualizar las tasas
SL = Oferta - New_Stop (StopLoss) * Point; // Cálculo de abrirse SL
TP = Oferta + New_Stop (TakeProfit) * Point; // Cálculo de abrirse SL
Alert ( "Intento de Compra abierta. Esperando la respuesta ..");
Venta de entradas = OrderSend (Symb, OP_BUY, LTS, Pregunte, 2, SL, TP); // Apertura Comprar
if (Ticket <0) // Exito:)
(
Alert ( "Abierto oredr Comprar", Venta de entradas);
return; // Salir de inicio ()
)
if (Fun_Error (GetLastError ()) == 1) // Procesamiento de errores
continuar; // Volviendo
return; // Salir de inicio ()
)
if (Total == 0 & amp; & amp; Opn_S == true) // No hay nuevas órdenes +
(// Criterio para la apertura de Venta
RefreshRates (); // Actualizar las tasas
SL = + Pregunte New_Stop (StopLoss) * Point; // Cálculo de abrirse SL
TP = Pregunte - New_Stop (TakeProfit) * Point; // Cálculo de abrirse SL
Alert ( "La tentativa para abrir Venta. Esperando la respuesta ..");
Venta de entradas = OrderSend (Symb, OP_SELL, LTS, Oferta, 2, SL, TP); // Apertura SELS
```

```
if (Ticket <0) // Exito:)
(
Alert ( "Abierto para Vender", Venta de entradas);
return; // Salir de inicio ()
)
if (Fun_Error (GetLastError ()) == 1) // Procesamiento de errores
continuar; // Volviendo
return; // Salir de inicio ()
)
break; // Salir, mientras que
)
//---------------------------------------------- ---------- 9 --
return; // Salir de inicio ()
)
//---------------------------------------------- ---------- 10 --
int Fun_Error (int error) // Función de los errores de procesamiento
(
switch (error)
(// No cruciales errores
Caso 4: Alerta ( "Comercio servidor está ocupado. Intentar una vez más ..");
Sleep (3000); // Simple solución
return (1); // Salir de la función
caso 135: Alerta ( "Precio cambiado. Intentar una vez más ..");
RefreshRates (); // Actualizar las tasas
return (1); // Salir de la función
caso 136: Alerta ( "No hay precios. Esperando a marcar un nuevo ..");
while (RefreshRates () == false) // Hasta una nueva tick
Sleep (1); // Pausa en el bucle
return (1); // Salir de la función
caso 137: Alerta ( "Broker está ocupado. Intentar una vez más ..");
Sleep (3000); // Simple solución
return (1); // Salir de la función
caso 146: Alerta ( "Trading subsistema está ocupado. Intentar una vez más ..");
Sleep (500); // Simple solución
return (1); // Salir de la función
// Errores críticos
caso 2: Alerta ( "error común".);
return (0); // Salir de la función
Caso 5: Alerta ( "versión antigua terminal.");
Trabajo = false; // Finaliza la operación
return (0); // Salir de la función
caso 64: Alerta ( "Cuenta bloqueada.");
Trabajo = false; // Finaliza la operación
return (0); // Salir de la función
caso 133: Alerta ( "Trading prohibido.");
return (0); // Salir de la función
caso 134: Alerta ( "No hay suficiente dinero para ejecutar la operación.");
return (0); // Salir de la función
default: alert ( "Se ha producido un error:" Error); // Otras variantes
return (0); // Salir de la función
)
)
//---------------------------------------------- ---------- 11 --
int New_Stop (int Parametr) // Comprobación de los niveles de parada
(
int Min_Dist = MarketInfo (Symb, MODE_STOPLEVEL); // Mínima distancia
if (Parametr <Min_Dist) // Si menos del permitido
(
Parametr = Min_Dist; // Set permitido
Alert ( "El aumento de la distancia de parada.");
)
return (Parametr); // retorna valor
```

```
 )
 //---------------------------------------------- ---------- 12 --
```

Vamos a analizar qué modificaciones se introdujeron en el código fuente (tradingexpert.mq4). La parte principal del Asesor Experto utilizado como base no ha cambiado. Los cambios se han hecho en dos bloques - bloque 1-2 - y el bloque 5-6.

En el bloque 5-6 criterios de comercio se calculan. En la AE se describe una estrategia comercial se basa en dos criterios de comercio - criterio para abrir Comprar y criterio para abrir Venta. La estrategia utilizada por el Asesor Experto permite la presencia de un solo abrió el mercado para, a la órdenes en espera de ser ejecutadas no están permitidas. La estrategia también supone el cierre de un orden opuesto, cuando un criterio para la apertura de factores desencadenantes, por ejemplo, si el criterio para abrir una Compre fin es relevante, que significa que una orden Vender debe cerrarse.

Para el uso en el AE shared.mq4 resultados de los cálculos realizados por la usuario rocseparate.mq4 indicador, la función iCustom () se debe ejecutar:

```
 doble L_1 = iCustom (NULL, 0, "rocseparate", H, P, B, A, 1, 0);
 doble L_5 = iCustom (NULL, 0, "rocseparate", H, P, B, A, 5, 0);
```

En este caso los parámetros formales especificados en iCustom () indican los siguientes:

NULL - cálculos en el indicador se realizan sobre la base de datos de la actual en materia de seguridad; en este caso, la AE se vincula a la EURUSD ventana, de modo que los datos de EURUSD serán usados (vea Fig. 131);

0 - en los cálculos los datos del actual periodo de tiempo se utilizan, en este caso el plazo actual es M15, por lo que los datos correspondientes a M15 se utilizarán;

"rocseparate"-nombre de un indicador personal, en la que los cálculos se harán.

H, P, B, A - lista de parámetros ajustables. En este caso la usuario indicador rocseparate.mq4 ha parámetros ajustables (2-3 bloque de código rocseparate.mq4). Para que un usuario sea capaz de establecer los valores de estos parámetros de la EA, que se especifican en la lista aprobada de parámetros de la función iCustom (). En el Asesor Experto valores de estos parámetros pueden ser diferentes de los especificados en el indicador. En tal caso, durante cálculos en el indicador pasó exactamente estos valores serán utilizados. Estos parámetros indican lo siguiente:

H - número de barras de cálculo en la historia;

P - período de cálculo MA;

B - número de bares de cálculo de la tasa;

A - número de bares de suavizado.

(el significado de estos parámetros se explica en detalle en la sección Custom Indicador ROC (Precio Tasa de Cambio).

1 (5) - Índice línea del indicador. En el indicador personal rocseparate.mq4 6 indicador se utilizan arrays. ROC línea en el actual calendario (naranja) se construye sobre la base de Line_1 [] valores, para que de amortiguación con índice 1 se utiliza. Suavizadas tasa media línea se basa en los valores de Line_5 [] matriz elementos, el índice de amortiguación se utiliza 5.

0 - Índice de valor obtenido a partir de un indicador de amortiguación (recaer en relación con una corriente de la barra de determinado número de períodos). En este caso los valores del indicador líneas en la barra de cero se utilizan, es por eso que el índice 0 es especificado.

Para que un usuario sea capaz de cambiar el indicador de parámetros ajustables en la AE manualmente, las variables externas se especifican en el bloque 1-1b (del Experto Asesor). En el bloque 5-5 bis valores de estos parámetros son asignados a otras variables con nombres cortos - esto se hace por conveniencia de presentación de código en el bloque 5 bis-5b. Así, un usuario puede especificar en shared.mq4 parámetros, cálculos con los que en la usuario indicador rocseparate.mq4 se llevará a cabo. Después de la ejecución iCustom () devolverá el valor correspondiente a un determinado elemento de valor se especifica el indicador calculado en conjunto el indicador se especifica usando los valores de parámetros ajustables.

Durante el funcionamiento práctico es conveniente ver en una ventana de un símbolo en las líneas del indicador, matriz de elementos que se utilizan en el Asesor Experto (véase la Fig. 131). Al mismo tiempo la ejecución de iCustom () no está conectado con la presencia del indicador en la ventana de un símbolo, así como con los valores de sus parámetros ajustables.



La ejecución de iCustom () no requiere la fijación de un indicador correspondiente a una ventana de un símbolo. Así como la convocatoria de iCustom () de cualquier programa de aplicación no dé lugar a la fijación de un indicador correspondiente a una ventana de un símbolo. Archivo adjunto de un indicador técnico a una ventana de un símbolo también no dar lugar a la llamada de iCustom en cualquier programa de aplicación.

Trading criterios en la AE (bloque 5-6) se calculan sobre la base de Los elementos de matriz valores obtenidos mediante iCustom (). Por ejemplo, un criterio para la apertura de Compra y Venta de clausura se calculan del siguiente modo:

```
if (L_5 <Nivel =- & & L_1> L_5)
(
Opn_B = true; // Criterio para la apertura de Compra
Cls_S = true; // Criterio para el cierre de Venta
)
```

Si el último valor conocido de un suavizado tasa media línea (L_5) es inferior al nivel especificado (valor del parámetro Nivel ajustable = 0,001) y el último valor conocido de ROC línea en el actual calendario (L_1) es más grande que el suavizado tasa media línea (L_5), el criterio para la apertura de una Compre orden y el cierre de un fin de Venta se considera pertinente. Para la confirmación de la pertinencia de frente a criterios que reflejen las condiciones se utilizan.



Trading criterios aceptados en este ejemplo se utilizan para fines educativos solamente y no debe ser considerada como una directriz al comercio en una verdadera cuenta.

Funciones estándar

Al en todos hay más de 220 funciones estándar en MQL4, esto es aparte de las funciones de indicadores técnicos. Es imposible incluir aquí las descripciones y ejemplos de todas las funciones, porque hay demasiados de ellos. Algunas funciones que deben ser descritos en detalles se incluyen en las secciones anteriores. En esta sección vamos a insistir en otras más utilizado funciones. Al final de cada apartado podrá ver la lista completa de funciones de una categoría determinada y su descripción breve.

- Funciones comunes.
  En este grupo se incluyen funciones que no están incluidos en ninguna de grupos especializados. Estas son las siguientes funciones: Imprimir (), Alerta (), el comentario (), MarketInfo (), Sleep (), etc

- Objetos gráficos.
  MetaTrader 4 terminal permite la conexión de numerosos objetos gráficos de un gráfico. Este grupo incluye las funciones que se utilizan para programar la creación de esos objetos, así como para cambiar sus propiedades, se desplazan y se suprima.

- Operaciones con los gráficos.
  Un grupo de funciones que se utilizan para obtener información diferente acerca de un gráfico actual, para que un programa en MQL4 (script, o indicador de Expertos Técnicos) se vincula.

- Funciones de cadenas.
  Cadena de funciones se utilizan para la transformación de variables de tipo cadena: la búsqueda de valor, concatenación de líneas, la recuperación de sub-líneas, etc funciones de conversión se utilizan para convertir una variable de un tipo a otro tipo. NormalizeDouble () redondea los valores de doble tipo para una determinada precisión.

- Fecha y hora.
  Este grupo de funciones se usa para obtener información en tiempo a tal o cual forma: localtime () muestra la hora local de un ordenador, TimeCurrent () muestra servidor de tiempo de la última cotización. Además, los parámetros como un día de la semana, mes día, hora, minuto, etc puede obtenerse por un valor de tiempo indicado.

- Archivo de Operaciones.
  Este grupo de funciones es necesaria para la lectura / grabación de datos en un disco duro.

- Las matrices y Timeseries.
  Ofrecen acceso a datos de precios de cualquier símbolo o el periodo.

- Funciones matemáticas.
  Conjunto estándar de matemáticas y funciones trigonométricas.

- GlobalVariables.
  Funciones para trabajar con GlobalVariables.

- Indicadores personalizado.
  Estas funciones pueden utilizarse sólo cuando los indicadores escrito personalizado.

- Datos de la cuenta.
  Funciones de mostrar información sobre un Terminal de Usuario, cuenta y verificar el estado actual de un Terminal de Usuario (incluido el estado del medio ambiente de MQL4-programa en ejecución).

- Funciones del Comercio.
  Funciones para realizar operaciones de comercio.

Para obtener una descripción detallada de cualquier función estándar se refieren a MQL4 referencia a MQL4.community, MetaQuotes Software Corp sitio web o de "Ayuda" en MetaEditor.

Funciones comunes

Uno de los más utilizados es comentario funciones ().

Comentario ()

```
Comentario vacío (...)
```

Esta función introduce un comentario definido por un usuario en la esquina superior izquierda de un gráfico de ventanas. Los parámetros pueden ser de cualquier tipo. Número de parámetros no puede ser superior a 64. Las matrices no se puede pasar a la observación (). Las matrices deben introducirse elementwise. Datos de doble tipo se han escrito con 4 dígitos después del punto decimal. Por las cifras que muestran con exactitud el uso DoubleToStr (). Bool, datetime tipos y colores se escribe como dígitos. Para mostrar los datos de tipo datetime como una cadena utilizar el TimeToStr ().

Parámetros:
... - Cualquier valores separados por comas.

Ejemplo del uso de esta función puede ser un simple AE comment.mq4 que refleja la información sobre el número de órdenes.

```
//---------------------------------------------- ----------
// Comment.mq4
// El código debería ser usado para fines educativos únicamente.
//---------------------------------------------- ----------
int start () // función especial de inicio
(
int órdenes OrdersTotal = (); // Número de órdenes
if (Órdenes == 0) // Si entumecida. de Ord. = 0
Comentario ( "No órdenes"); // Comentario a la ventana de esquina
else // Si hay órdenes
Comentario ( "Disponible", órdenes, "órdenes."); // Comentario
return; // Salir
)
//---------------------------------------------- ----------
```

En el comienzo del programa el número total de los órdenes se cuenta por el OrdersTotal (). Si las órdenes variable (número de órdenes) es igual a 0, el comentario () con "No órdenes" parámetro se ejecuta. Si hay al menos un fin, el comentario () con una lista de parámetros separados por comas se llevará a cabo. En este caso 3 se usan parámetros: el primero es una cadena de valor "Disponible", en segundo lugar es un valor órdenes y el tercero es una cadena de valor "órdenes".. Como resultado de la ejecución de la función que en cada inicio de la función especial start () uno de los mensajes se mostrarán en la esquina superior izquierda de la ventana de un gráfico. Fig. 132 muestra un gráfico en la ventana de la situación cuando hay una orden actual.



Fig. 132. Viendo un texto en la esquina superior izquierda de la ventana de un gráfico como resultado de Observación () la ejecución.

Para la reproducción de archivos de sonido PlaySound () se utiliza.

PlaySound ()

```
void PlaySound (string filename)
```

La función desempeña un archivo de sonido. El expediente se encuentra en ***terminal_directory \ sonidos*** o sus subdirectorios.

Parámetros:
filename - camino a un archivo de sonido.

Un conjunto de archivos de sonido recomendados se pueden encontrar en el archivo adjunto - Archivos de Sonido.

En algunos casos, un programa puede ser escrito para apoyar un diálogo con un usuario. La función MessageBox () se utiliza para este fin.

MessageBox ()

```
int MessageBox (cadena de texto = NULL, string caption = NULL, int banderas = EMPTY)
```

Función MessageBox crea y muestra un cuadro de mensaje, también se utiliza para gestionar la ventana de diálogo. Un cuadro de mensaje contiene un mensaje y la cabecera se define enuna programa, así como cualquier combinación de iconos predefinidos y los botones. Si una función se ejecuta con éxito, el valor devuelto es uno de los valores de código de return de MessageBox (). La función no puede ser llamado de una usuario indicador, porque los indicadores son ejecutados en la interfaz de hilo y es posible que no descenderá.

Parámetros:
texto - un texto que contenga un mensaje que se mostrará;
caption - un facultativo texto que se muestra en el cuadro de mensaje. Si el parámetro es vacío, un nombre de AE se mostrará en el cuadro de cabecera;
banderas - banderas opcionales definir el tipo y el comportamiento del cuadro de diálogo. Banderas puede ser una combinación de banderas de los grupos bandera (ver MessageBox códigos de return).

Pongamos un ejemplo de MessageBox () de uso.



Problema 31. Escriba un código de una AE que muestra un cuadro de mensaje con una cuestión de cerrar todas las órdenes de 5 minutos antes de la importante comunicado de prensa. Si un usuario hace clic en Sí, todos los órdenes deben ser cerrados, si no se empuja, las acciones no deben realizarse.

La AE apoyo a un diálogo con un usuario (dialogue.mq4) pueden tener el siguiente código:

```
//---------------------------------------------- ----------------
// Dialogue.mq4
// El código debería ser usado para fines educativos únicamente.
//---------------------------------------------- --------------- 1 --
# include <WinUser32. mqh> // Needed a MessageBox
extern doble Time_News = 15,30; // Tiempo de noticias importantes
Cuestión bool = false; // Bandera (cuestión no es poner aún)
//---------------------------------------------- --------------- 2 --
int start () // función especial de inicio
(
PlaySound ( "tick.wav"); // En cada una tick
doble Time_cur = hora () + Minuto () / 100,0 // Hora actual (doble)
if (OrdersTotal ()> 0 & & Cuestión == false & & Time_cur <= Time_News - 0.05)
(// Proporcionar algunas condiciones
PlaySound ( "news.wav"); // En cada una tick
Cuestión = true; // Bandera (cuestión ya está puesto)
int ret = MessageBox ( "Tiempo de importante comunicado de prensa. Cerrar todas las órdenes?"
"Pregunta", MB_YESNO | MB_ICONQUESTION | MB_TOPMOST); // cuadro de mensaje
//---------------------------------------------- --------- 3 --
if (ret == IDYES) // Si la respuesta es Sí
Close_Orders (); // Cerrar todas las órdenes
)
return; // Salir
)
//---------------------------------------------- --------------- 4 --
Close_Orders vacío () // cliente.  Funct.  para el cierre de órdenes
(
Alert ( "La función de cierre de todas las órdenes se está ejecutando."); // A título de ejemplo
return; // Salir
)
//---------------------------------------------- --------------- 5 --
```

En el bloque 1-2 WinUser32.mqh archivo es incluido en el programa; en este archivo MessageBox () códigos de return se definen. También en este bloque la variable externa Time_news se introduce - este es el momento de importante comunicado de prensa. Durante todo el período de ejecución AE una pregunta sobre el cierre de órdenes deben ser mostradas una sola vez. Para rastrear si la cuestión ha sido ya está representada en AE 'Pregunta' variable se declara.

En cada salida de la función especial start () (bloque 2-3) PlaySound () es ejecutado. La desempeñado tick.wav sonido se asemeja a una débil clic que indica la mejor manera el hecho de marcar una nueva apariencia. La decisión acerca del uso de sonido en un programa se realiza mediante un programador. En algunos casos es muy útil para utilizar sonidos. Por ejemplo, un sonido puede indicar el hecho de una ejecución EA. Otros sonidos pueden corresponder a otros eventos, por ejemplo disparo de un criterio comercial, orden de clausura, etc

Valor de la variable real Time_cur corresponde a los actuales tiempos de servidor. En la AE condiciones, en la que el cuadro de mensaje debe ser visualizada, se analizan. Si hay uno o varios órdenes, el cuadro de mensaje no se ha demostrado aún y el servidor se diferencia de tiempo importante comunicado de prensa por tiempo inferior a 5 minutos, algunas acciones se realizan en el programa. En primer lugar la función PlaySound () es ejecutado, el sonido jugado atrae la atención de un usuario. El pabellón se pregunta el verdadero valor (no muestran la próxima vez). En la siguiente línea MessageBox () es ejecutado:

```
int ret = MessageBox ( "Tiempo de importante comunicado de prensa. Cerrar todas las órdenes?"
"Pregunta", MB_YESNO | MB_ICONQUESTION | MB_TOPMOST); // cuadro de mensaje
```

En este caso, el valor de una cadena constante "Tiempo de importante comunicado de prensa. Cerrar todas las órdenes?" se mostrará en un cuadro de diálogo, y "Cuestión" valor se refleja en la parte superior del cuadro de línea. La bandera MB_YESNO denota la presencia de botones - en este caso, Sí y No botones (véase el MessageBox Valores de return). La bandera MB_ICONQUESTION define un icono que aparece en la parte izquierda del cuadro de mensaje (cada entorno operativo tiene su propio conjunto de iconos, Fig. 133 muestra un icono de Windows XP configurado). El pabellón dispone MB_TOPMOST la caja con la propiedad "siempre en primer plano", es decir, la caja será siempre visible, independientemente de qué programas se ejecutan en el momento en el ordenador. Como resultado de la ejecución de MessageBox () con parámetros indicados un cuadro de mensaje se muestra:

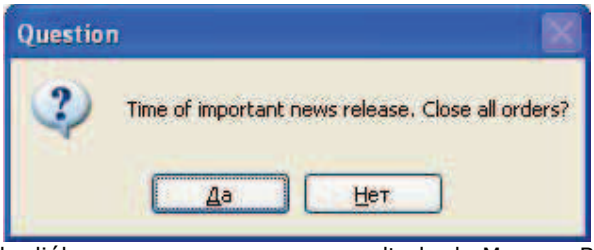


Fig. 133. Cuadro de diálogo que aparece como resultado de MessageBox () de ejecución.

En el momento en que el cuadro de mensaje se muestra la ejecución del programa se mantiene hasta que un usuario hace clic en un botón en el cuadro de mensaje. Tan pronto como sucede, el control se debe pasar a la siguiente línea MessageBox (), en este caso para el bloque 3-4. Esta propiedad de un cuadro de mensaje para mantener el control es muy importante y debe tenerse en cuenta en un programa de desarrollo. Por ejemplo, si un usuario dejó su ordenador y un cuadro de mensaje se muestra en este momento, durante todo el tiempo cuando un usuario está ausente (hasta que se pulsa un botón), el programa se espera de la respuesta y no el código se ejecutará en este período.

Tenga en cuenta, ante un cuadro de mensaje se muestra la ejecución del programa va acompañado de un sonido de los ticks. Cuando el cuadro de mensaje aparece otro sonido se juega. En el período en que el cuadro de diálogo está abierto y esperando una buena respuesta no se juega lo que demuestra el hecho de la celebración de control, mientras que el cuadro de diálogo está abierto. Después se pulsa un botón, el programa seguirá la ejecución y el sonido de los ticks se reproducirá de nuevo.

Si un usuario hace clic en Sí, la Close_Orders () será llamada; esta función se utiliza para el cierre de órdenes. En este ejemplo la función de los contenidos no se describe, para denotar su ejecución la Alerta función se ejecuta ("La función de cierre de todas las órdenes se está ejecutando."). Si un usuario hace clic en No, la función de las órdenes de clausura no se llama. En el actual período de sesiones de la AE ejecución el cuadro de mensaje no se muestra de nuevo.

Funciones comunes

Función	Resumen de Información
Alerta	Muestra un cuadro de mensaje que contiene definidos por el usuario de datos. Los parámetros pueden ser de cualquier tipo. Número de parámetros no puede ser superior a 64.
Comentario	Muestra un comentario definida por un usuario en la esquina superior loft de un gráfico de ventanas. Los parámetros pueden ser de cualquier tipo. Número de parámetros no puede ser superior a 64.
GetTickCount	GetTickCount () devuelve el número de milisegundos transcurrido desde que un sistema se inició. El contador se ve limitada por la resolución del sistema de temporizador. Como el tiempo se almacena como un entero sin signo, es demasiado cada 49,7 días.
MarketInfo	Devuelve información acerca de los valores que figuran en el "Mercado de ver" la ventana. Parte de la información sobre las actuales condiciones de seguridad se almacena en variables predefinidas (véase el MarketInfo () Identificadores).
MessageBox	Función MessageBox crea y muestra un cuadro de mensaje, también se utiliza para gestionar la ventana de diálogo. Un cuadro de mensaje contiene un mensaje y la cabecera se define enuna programa, así como cualquier combinación de iconos predefinidos y los botones. Si una función se ejecuta con éxito, el valor devuelto es uno de los código de return valores de MessageBox (). La función no puede ser llamado de una usuario indicador, porque los indicadores son ejecutados en la interfaz de hilo y es posible que no descenderá.
PlaySound	Reproduce un archivo de sonido. El archivo debe estar situado en el *terminal_dir* \ *sonidos* directorio o en su subdirectorio.
Imprimir	Imprime un mensaje a los expertos de registro. Los parámetros pueden ser de cualquier tipo. Importe de los parámetros pasado no puede ser superior a 64.
SendFTP	Envía un archivo a la dirección especificada en el establecimiento de la ventana de "Publisher" pestaña. Si el intento falla, retuns FALSO. La función no funciona en el modo

	de ensayo. Esta función no puede ser llamado a partir de indicadores personales, tampoco. El archivo a ser enviado se debe almacenar en la *terminal_directory* \ *expertos* \ *archivos de* la carpeta o en sus subcarpetas. No será enviada si no hay una dirección FTP y / o contraseña de acceso a la configuración especificada.
SendMail	Envía un e-mail a la dirección indicada en la ventana de configuración de "Correo electrónico" pestaña. El envío puede ser desactivada en la configuración, o puede ser omitido especificar la dirección de correo electrónico.
Dormir	El Sueño () suspende la ejecución de los actuales expertos en el intervalo especificado. Sleep () no pueden ser llamados indicadores personales, porque los indicadores son ejecutados en la interfaz de hilo y es posible que no descenderá. El control del experto dejar de bandera estado cada 0,1 segundos se construye en la función.

Para la descripción detallada de estas y otras funciones por favor consulte la documentación en MQL4.community, MetaQuotes Software Corp sitio web o de "Ayuda" en la sección MetaEditor.

Objetos gráficos

Gráficos objeto es una imagen en la ventana de símbolo, sino que se puede seleccionar, mover, modificados o suprimidos.

Objetos gráficos incluyen, por ejemplo, las líneas horizontales y verticales, canal de regresión lineal, los niveles de Fibonacci, rectángulo, el texto marca, etc Estas imágenes como indicador de líneas, indicador de los niveles, candeleros, los comentarios escritos por el comentario () y otros no pueden ser seleccionados y suprimido, que es la razón por la que no pertenecen a objetos gráficos.

Gráficos objeto son atraídos por el Terminal de Usuario en una ventana de un símbolo de acuerdo con las coordenadas preestablecidas. Cada objeto gráfico en función de su tipo tiene uno, dos o tres coordenadas y otros parámetros ajustables. Cualquier objeto gráfico se pueden colocar en un gráfico manualmente (desde la barra de herramientas de un sistema de menú), y también como consecuencia de la ejecución de un programa de aplicación se inició en la misma ventana, incluido un Asesor Experto, la escritura o la usuario indicador. Tipo y la ubicación de un objeto gráfico se puede modificar manualmente o por un programa de envío de nuevos valores de coordenadas y otros parámetros a un objeto gráfico.

Formas de posicionamiento objetos gráficos

Hay dos formas de posicionamiento de objetos en MQL4 aceptado: en relación con un gráfico y en relación con una ventana de un símbolo. Para ilustrar la diferencia entre estos métodos, vamos a cabo manualmente dos objetos en una ventana de un símbolo: texto (OBJ_TEXT) y una marca de texto (OBJ_LABEL). Podemos usar **T A** y botones de la barra de herramientas de cliente de la terminal. Vamos a configurar el tamaño de la ventana por lo que es igual a la mitad del tamaño de la pantalla (Fig. 134). Vamos a ver cómo estos objetos gráficos reaccionará a los cambios de tamaño de la ventana (así como a la horizontal y vertical de escala de la gráfica de precios).

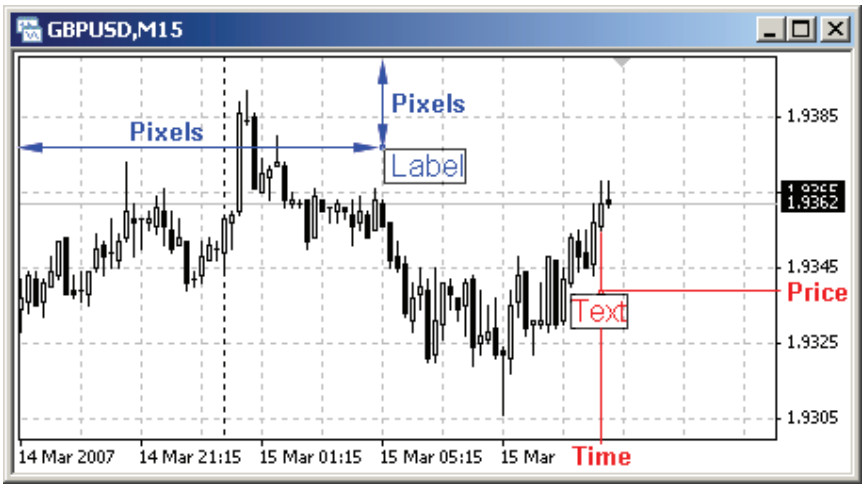
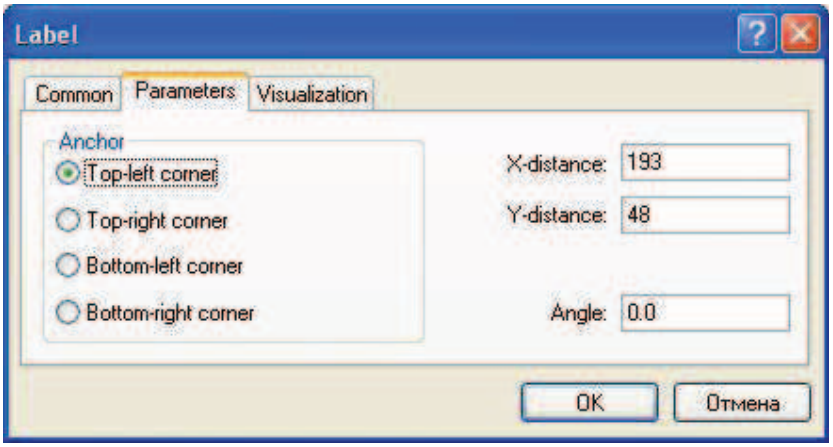


Fig. 134. Objetos gráficos con diferentes métodos de posicionamiento en una ventana de un símbolo.

Posicionamiento con relación a un gráfico

El objeto gráfico OBJ_LABEL inmuebles seguirá siendo si un tamaño de la ventana se cambia de manera de desplazar su derecho o inferior fronteras. Pero si el tamaño de la ventana se cambia de cambio de su superior o inferior frontera, el objeto se desplazó también, aunque la posición del objeto en relación con estas fronteras se mantendrán sin variación. Esto sucede porque OBJ_LABEL se posiciona en relación a la seguridad los bordes de las ventanas. En este caso, el punto de referencia de la gráfica de oponerse a una ventana de un símbolo es la esquina superior izquierda de una ventana chart6. Coordenadas del objeto en relación con el punto indicado se establecen en píxeles - 193 y 48 (Fig. 135).



--

Fig. 135. Configuración de la gráfica objeto OBJ_LABEL.

El punto de referencia de las coordenadas del objeto (en este caso) es la esquina superior izquierda del marco de un cursor visible cuando seleccionados por un ratón. En la esquina superior izquierda del cursor marco se puede ver un pequeño punto que indica la configuración gráfica de este objeto. Si otro punto de referencia se indica, el punto en el marco del cursor se indica en otra esquina.

Cuando aparecen nuevas barras en un gráfico, un objeto como OBJ_LABEL permanecerá inamovible en la ventana. El uso de este objeto es conveniente si es necesario para mostrar la información del texto de carácter general, por ejemplo, información sobre la terminación de la negociación, el valor de una limitación de distancia cambiado por un corredor, etc

Posicionamiento con relación a un gráfico

En cualquier método de cambio de tamaño de las ventanas, así como en el gráfico de escala, un objeto de tipo OBJ_TEXT no cambia su posición en relación con un gráfico. El punto de referencia de dicho objeto es el centro de la línea superior de un cursor marco, su coordenada X es el tiempo, coordenada Y es una garantía de precios (Fig. 136).

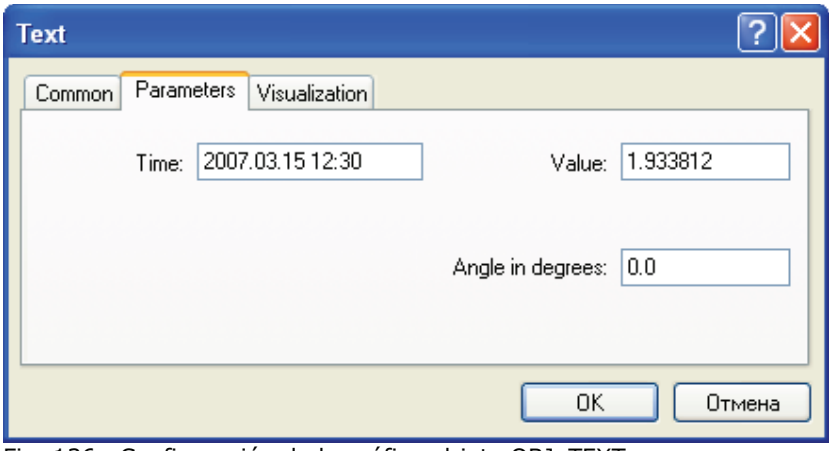


Fig. 136. Configuración de la gráfica objeto OBJ_TEXT.

A medida que los nuevos bares aparecen en un cuadro ventana, la posición de OBJ_TEXT no cambia en relación con un gráfico, es decir, con la aparición de nuevos bares el objeto se desplazó a la izquierda junto con la gráfica, y cuando será suficiente bares, la objeto se desplazará más a la izquierda de la ventana de fronteras.

Tal o cual método de posicionamiento de la propiedad de un determinado tipo de objeto y no se puede cambiar de un usuario, incluso en un programa. La mayoría de los objetos gráficos se posiciona en relación con un gráfico, es decir, en el tiempo y el precio de coordenadas.

Creación de objetos gráficos y el cambio de sus propiedades

Para crear un objeto gráfico a cabo mediante un gráfico en una ventana de objetos de tipos predefinidos (ver Tipos y Propiedades de objetos gráficos). Por objeto la creación de la siguiente función se utiliza:

ObjectCreate ()

```
bool ObjectCreate (string nombre, int tipo, int ventana, datetime time1, doble price1, datetime time2 = 0,
doble price2 = 0, datetime time3 = 0, doble price3 = 0)
```

La función crea un objeto de un tipo indicado con un preset nombre y coordenadas se indica en el gráfico subventana. Número de coordenadas del objeto puede ser de 1 a 3 según el tipo de objeto. Si un objeto se ha creado correctamente, la función devuelve TRUE, FALSE de otra manera. Para obtener información adicional acerca de un error llamar al GetLastError ().

Coordina deberá ser aprobado en parejas - el tiempo y el precio. Por ejemplo OBJ_VLINE sólo necesita tiempo, pero el precio debe también ser (cualquier valor). Gráficos objeto de OBJ_LABEL tipo ignora las coordenadas especificadas en la función de establecer OBJPROP_XDISTANCE y OBJPROP_YDISTANCE de este objeto la ObjectSet () debe utilizarse.

Parámetros:

- **nombre** - nombre de objeto;
- **tipo** - tipo de objeto (puede ser uno de predefinidos tipos de objetos);
- **ventana** - número de ventana en la que un objeto se añade. Numeración de la gráfica sub-ventanas (si hay sub-ventanas con indicadores de la actualidad) se inicia a partir del 1, el número de la ventana principal es siempre 0; indicó la viuda número debe ser mayor que o igual a 0 y menor que el valor devuelto por el WindowsTotal () función;
- **time1** - tiempo de coordinar la primera;
- **price1** - el precio de la primera coordinar;
- **time2** - tiempo de la segunda coordinar;
- **price2** - el precio del segundo coordinar;
- **time3** - tiempo de coordinar la tercera;
- **price3** - el precio de coordinar la tercera.

Cada objeto gráfico tiene algunas (peculiar a) parámetros ajustables. Por ejemplo, además de definirse las coordenadas, puede especificar el color, mensaje de texto (para algunos objetos), estilos de línea (para otros objetos), etc Para cambiar las propiedades utilizar la siguiente función:

ObjectSet ()

```
bool ObjectSet (string nombre, int prop_id, doble valor)
```

La función cambia el valor de los bienes objeto indicado. En caso de éxito la función devuelve TRUE, FALSE de otra manera. Para obtener la información de error llamar al GetLastError ().

Parámetros:

- **nombre** - nombre de objeto;
- **prop_id** - Identificador de propiedades del objeto (una de las propiedades del objeto se indique lo contrario);
- **valor** - un nuevo valor de la propiedad indicada.

Todos los objetos gráficos pueden tener una descripción de texto. La descripción de cada objeto está disponible para un usuario y se puede cambiar de un objeto o propiedades de la barra de herramientas en una forma programada. Por OBJ_TEXT y OBJ_LABEL esta descripción es su principal y contenido y se muestra siempre como una línea de texto, las descripciones de texto y otros objetos se muestran cerca del objeto, si la opción "Mostrar descripciones objeto" está activado en una ventana de propiedades de símbolo (F8). Para cambiar el texto la descripción siguiente función se utiliza:

ObjectSetText ()

```
bool ObjectSetText (string nombre, la cadena de texto, int font_size, cadena font_name = NULL, color text_color = CLR_NONE)
```

La función se usa para cambiar la descripción de un objeto. En caso de éxito se devuelve TRUE, de lo contrario - FALSO. Para obtener la información de error llamar al GetLastError (). Parámetros font_size, font_name y text_color se utilizan sólo para OBJ_TEXT y OBJ_LABEL. Para los objetos de otros tipos de estos parámetros se ignoran.

Parámetros:

- **nombre** - nombre de objeto;
- **texto** - La descripción del objeto de texto;
- **font_size** - el tamaño de la letra en los puntos;
- **font_name** - nombre de la fuente;
- **text_color** - el color del texto.

Vamos a analizar un ejemplo de un Asesor Experto, en el que las funciones de gestión de objetos gráficos se utilizan.



Problema 32. Utilizando un gráfico objeto informar al usuario sobre el comercio de criterios definidos sobre la base de valores MACD.

MACD es muy a menudo utilizados por los comerciantes para la formación de criterios comerciales. El indicador está representado por dos líneas - y la señal principal. Un comercio de criterios se considera que deben realizarse cuando las líneas cruzadas. Si el principal indicador de línea (normalmente de color gris histograma) cruza la línea de señal (generalmente rojo línea de puntos) la baja, esta es una señal de venta, id hacia arriba - para comprar. En los intervalos entre la línea de cruce de órdenes de mercado se debe mantener abierto, y cuando un criterio contrario desencadena, las órdenes deben ser cerradas y opuestas, una vez abierto. Así, cuatro tipos de mensajes deben estar preparados: la apertura de Comprar, Vender de apertura, la celebración de Compra, Venta de celebración.

En este problema todos los mensajes son mutuamente excluyentes, es decir, la situación cuando dos o más mensajes deben ser mostrado es imposible. Es por ello que en este caso un objeto gráfico se puede utilizar el objeto será siempre presentes en pantalla ª, pero será cambiado de vez en cuando. Vamos a sacar este objeto en la esquina superior derecha de la ventana, en la que la AE se operan. Desde la posición de objeto no debe ser cambiado, es conveniente utilizar un objeto de tipo OBJ_LABEL, porque se posiciona en relación con un gráfico de ventanas.

Como una solución del Problema 32 vamos a ver la AE grafobjects.mq4 utilizando la gráfica objeto OBJ_LABEL:

```
//---------------------------------------------- --------------------
// Grafobjects.mq4
// El código debería ser usado para fines educativos únicamente.
//---------------------------------------------- --------------------
int start () // función especial de inicio
(
//---------------------------------------------- ---------- 1 --
Siéntese INT;
doble MACD_M_0, MACD_M_1, // Main línea, 0 y 1 bar
MACD_S_0, MACD_S_1; // Señal de línea, 0 y 1 bar
cadena de texto [4] // Declarar un array de cadenas
color Color [4] // Declarar una gama de colores

Texto [0] = "Apertura de Compra"; // Texto para las diferentes situaciones
Texto [1] = "Apertura de Venta";
Texto [2] = "Celebración de Compra";
Texto [3] = "Celebración de Venta";

Color [0] = DeepSkyBlue; // Objeto de color ..
Color [1] = LightPink; // .. para diferentes situaciones
Color [2] = amarillo;
Color [3] = amarillo;
//---------------------------------------------- ---------- 2 --
ObjectCreate ( "Label_Obj_MACD", OBJ_LABEL, 0, 0, 0); // Creación de obj.
ObjectSet ( "Label_Obj_MACD", OBJPROP_CORNER, 1); // Referencia esquina
ObjectSet ( "Label_Obj_MACD", OBJPROP_XDISTANCE, 10); // coordenada X
ObjectSet ( "Label_Obj_MACD", OBJPROP_YDISTANCE, 15); // coordenada Y
//---------------------------------------------- ---------- 3 --
MACD_M_0 = iMACD (NULL, 0, 12, 26, 9, PRICE_CLOSE, MODE_MAIN, 0); // 0 bar
MACD_S_0 = iMACD (NULL, 0, 12, 26, 9, PRICE_CLOSE, MODE_SIGNAL, 0); // 0 bar
MACD_M_1 = iMACD (NULL, 0, 12, 26, 9, PRICE_CLOSE, MODE_MAIN, 1); // 1 bar
MACD_S_1 = iMACD (NULL, 0, 12, 26, 9, PRICE_CLOSE, MODE_SIGNAL, 1); // 1 bar
//---------------------------------------------- ---------- 4 --
// Análisis de situación
if (MACD_M_1 = MACD_S_0) // Cruce hacia arriba
Siéntese = 0;
if (MACD_M_1> MACD_S_1 & & MACD_M_0 <= MACD_S_0) // Cruce de baja
Siéntese = 1;
if (MACD_M_1> MACD_S_1 & & MACD_M_0> MACD_S_0) // Main encima de la señal
Siéntese = 2;
if (MACD_M_1
```

En el bloque de AE 1-2 parámetros se definen, en particular, los valores de elemento de texto [] y de color [] se establecen. Además se utilizan para cambiar propiedades de objetos. En el bloque de 2-3 el objeto se crea y algunos valores de sus propiedades se establecen. Vamos a analizar este bloque en los detalles. De acuerdo con esta línea de código AE un objeto gráfico creado en la ventana, en la que la AE se ejecuta:

```
ObjectCreate ( "Label_Obj_MACD", OBJ_LABEL, 0, 0, 0); // Creación de obj.
```

"Label_Obj_MACD" denota el valor que este nombre se le asigna a su objeto (un nombre se le asigna a un objeto de un programador a su propia discreción). OBJ_LABEL - es el identificador de tipo de objeto, sino que denota que el objeto creado será exactamente de este tipo (elegido de la lista de posibles tipos). El primero de los próximos tres ceros a la izquierda denota que el objeto se crea en la ventana principal (la ventana principal de la gráfica donde se muestra, siempre tiene el índice 0).

Las próximas dos ceros las coordenadas establecidas para el objeto creado. De acuerdo con este coordinado el objeto se dibujará en la ventana indicada. En este caso el creado OBJ_LABEL no utiliza tiempo y precio coordenadas. Tenga en cuenta que en OjectCreate () descripción sólo el tiempo y el precio coordenadas se especifican. Por otra parte, las coordenadas de la segunda y la tercera los pares de valores por defecto, mientras que no hay valores por defecto para el primer par de coordenadas. Esto significa que aunque OBJ_LABEL no necesita tiempo y precio en todas las coordenadas, algunos valores deben especificarse en ObjectCreate () llamada a función. En este caso se indican ceros, aunque cualquier otro valores se pueden escribir - de todos modos estos valores serán olvidadas durante la configuración de OBJ_LABEL propiedades.

En los próximos tres líneas de algunos valores de propiedad se establecen para el objeto creado anteriormente llamado Label_Obj_MACD:

```
ObjectSet ( "Label_Obj_MACD", OBJPROP_CORNER, 1); // Referencia esquina
ObjectSet ( "Label_Obj_MACD", OBJPROP_XDISTANCE, 10); // coordenada X
ObjectSet ( "Label_Obj_MACD", OBJPROP_YDISTANCE, 15); // coordenada Y
```

Por la esquina de referencia (OBJPROP_CORNER) 1 se fija, lo que significa la esquina superior derecha de la primera se define la ventana principal. En los próximos dos líneas distancias del objeto a una esquina de referencia se establecen en píxeles: distancia horizontal (OBJPROP_XDISTANCE) 10 píxeles y la distancia vertical (OBJPROP_YDISTANCE) 15 píxeles. En esta etapa la ejecución del programa el objeto ya está creado, tiene su nombre único y se definen las propiedades principales.

Para realizar el objeto de mostrar un texto necesario, en primer lugar tenemos que calcular lo que este texto debe ser similar. Con este fin, primero 3-4 en el bloque la posición de las líneas de MACD se detecta en los actuales y los anteriores, bares, luego en el bloque 4-5 Siéntese valor correspondiente a la situación actual se calcula (véase también la Fig. 107 y callstohastic.mq4)

En la línea siguiente de propiedades del objeto, dependiendo de la situación actual se definen:

```
// Cambio de propiedades del objeto
ObjectSetText ( "Label_Obj_MACD", texto [Sit], 10, "Arial", de Color [Sit]);
```

Como resultado de ObjectSetText () la ejecución de un texto descripción se asigna al objeto llamado Label_Obj_MACD - el valor de la cadena de texto variable [Sit]. Este valor será diferente para diferentes situaciones en función de valores de la variable Sit. Por ejemplo, si la línea principal atraviesa la señal de una baja, en el bloque 4-5 Siéntese obtiene el valor 1, como resultado la gráfica objeto obtener el texto de descripción que figura en el texto [1] Los elementos de matriz, es decir, "Apertura de Vender ". Otros parámetros: 10, "Arial" y Color [Sit] denotar el tamaño de la letra, nombre y color para el texto de descripción.

Como resultado de la ejecución AE la siguiente aparecerá en la ventana EURUSD:



Fig. 137. Resultado de la AE grafobjects.mq4 operación en el momento en que el criterio para vender desencadenantes.

En la Fig. 137 hay una ventana principal y la subventana MACD. Cabe señalar aquí que para un funcionamiento normal AE presencia de este indicador en la ventana de símbolo no es necesario, porque el comercio de criterios en la AE se calculan como resultado de un indicador de función técnica de ejecución que no está conectado con el indicador que muestra. A este respecto, el indicador se muestra sólo para la explicación visual del momento de un criterio comercial disparo cuando sea necesario el texto de la descripción gráfica de objetos se muestra. La AE se operan en la misma forma a todas las demás combinaciones de la posición común del indicador líneas cada vez que muestran una descripción que corresponde a una situación.

Eliminar objetos gráficos

El Asesor Experto analizan grafobjects.mq4 tiene una pequeña desventaja. Después de la AE paradas de funcionamiento, un objeto gráfico permanecerá en el gráfico (sus propiedades seguirán siendo los mismos que en el momento t de su último cambio). Objetos gráficos no se eliminan automáticamente. En el curso de la negociación a partir de un cierto momento en que el mensaje de "apertura de Venta" no será válida. Con el fin de no desinformar a un usuario la gráfica objeto debe ser eliminado.

Para borrar un objeto gráfico (con independencia de su método de creación - programada o manual) simplemente selecciónelo y pulse la tecla Suprimir. Sin embargo, como para la programación, cabe señalar que un programa escrito correctamente debe "claro" cuando la ventana de su operación ha terminado. En otras palabras, un programa debe contener un bloque donde todos los objetos gráficos creados por el programa se borran.

ObjectDelete ()

```
bool ObjectDelete (string nombre)
```

la supresión de un objeto con el nombre indicado. Si un objeto se han eliminado, la función devuelve TRUE, de lo contrario - FALSO. Para obtener la información de error llamar al GetLastError () ..

Parámetros:

- **nombre** - el nombre de un objeto suprimido.

Es muy fácil de usar ObjectDelete (): simplemente indicar el nombre de un objeto a eliminar.

Para solucionar el inconveniente del ejemplo anterior, vamos a añadir en la AE grafobjects.mq4 la función especial deinit () con la función para la supresión de objetos:

```
//---------------------------------------------- ---------- 7 --
int deinit () // función especial deinit
(
ObjectDelete ( "Label_Obj_MACD"); // Objeto supresión
return; // Salir deinit ()
)
//---------------------------------------------- ---------- 8 --
```

Ahora, la AE durante la ejecución del objeto llamado Label_Obj_MACD serán borradas. En general, un programa puede crear numerosos objetos. Cada una de ellas puede ser suprimido de acuerdo con el algoritmo.

Modificación de objetos gráficos

En algunos casos es necesario cambiar la posición de un objeto en un gráfico de ventanas en un programa. Muy a menudo esa necesidad puede ocurrir debido a la aparición de nuevos bares. Por ejemplo, los criterios de comercio en un AE puede formarse sobre la base de una regresión lineal canal construido sobre una barra de la historia de una determinada longitud (por ejemplo, el pasado 50 bares). Si nos limitamos a señalar el objeto "canal de regresión lineal" en un gráfico de ventanas y, a continuación, no comprometen nada, seguirá siendo el mismo gráfico lugar donde se coloca y se desplazó a la izquierda como aparecen nuevas barras. Para evitar que el objeto de desplazamiento debe ser vuelto a trazar en cada nuevo bar. Con este fin, las nuevas coordenadas deben calcularse y pasa al objeto, de acuerdo con estas coordenadas del objeto se establecerán en un gráfico viuda.

Para averiguar qué propiedades de un objeto gráfico tiene en este momento, la siguiente función se debe utilizar:

ObjectGet ()

```
doble ObjectGet (string nombre, int prop_id)
```

la función devuelve el valor de los bienes objeto especificado. Para obtener la información de error llamar al GetLastError ().

parámetros:

- **nombre** - nombre de objeto;
- **prop_id** - Identificador de objeto de propiedad. Puede ser cualquier valor de la lista de propiedades del objeto.

Nueva coordenadas se presentan a un objeto usando la ObjectMove ().

ObjectMove ()

```
bool ObjectMove (string nombre, int punto, datetime time1, doble price1)
```

Cambiar una de las coordenadas en un gráfico. La función devuelve TRUE en caso de éxito, de lo contrario - FALSO. Para obtener información adicional llame al FetLast Error (). Numeración de un objeto comienza a partir de las coordenadas 0.

Parámetros:

- **nombre** - nombre de objeto;
- **punto** - coordinar índice (0-2);
- **time1** - nuevo valor temporal;
- **price1** - nuevo precio valor.



Problema 33. Crear un programa (un Asesor Experto) el apoyo a un dibujo de un canal de regresión lineal para los últimos 50 bares.

La gráfica de objeto "canal de regresión lineal" utiliza dos coordenadas de tiempo. Precio coordenadas (por ejemplo si se especifican en el programa) son ignoradas por el cliente durante la terminal objeto la construcción. El canal de regresión lineal se calcula por el Terminal de Usuario basado en datos de precios históricos y, por tanto, no se puede visualizar además de un gráfico. Esa es la razón por la ausencia del objeto vinculante para el precio (precio de ignorar las coordenadas de la terminal) es el objeto constante de la propia propiedad. Th Asesor Experto (moveobjects.mq4) la gestión de la posición de un objeto gráfico puede tener el siguiente código:

```
//------------------------------------------------- --------------------
// Moveobjects.mq4
// El código debería ser usado para fines educativos únicamente.
//------------------------------------------------- --------------------
extern int Len_Cn = 50 // longitud del canal (bares)
extern Col_Cn color = Orange; // Canal de color
//------------------------------------------------- ---------- 1 --
int init () // Función especial init ()
(
Crear (); // Llamar usuario-def.  funciones.  de la creación
return; // Salir de inicio ()
)
//------------------------------------------------- ---------- 2 --
int start () // Función especial start ()
(
datetime T2; // Segundo tiempo coordina
int error; // Código de error
//------------------------------------------------- ---------- 3 --
T2 = ObjectGet ( "Obj_Reg_Ch", OBJPROP_TIME2); // Solicitud de t2 coord.
Error = GetLastError (); // Obtener un código de error
if (error == 4202) // En caso de que no se oponga: (
(
Alert ( "canal de regresión es que se está manejando",
"\ N Book_expert_82_2. Supresión prohibido.");
Crear (); // Llamar usuario-def.  funciones.  de la creación
T2 = tiempo [0]; // Valor actual de coordinar t2
)
//------------------------------------------------- ---------- 4 --
if (T2! Time = [0]) // Si no es objeto en su lugar
(
ObjectMove ( "Obj_Reg_Ch", 0, Time [Len_Cn - 1], 0); // Nueva t1 coord.
ObjectMove ( "Obj_Reg_Ch", 1, Time [0], 0); // Nueva t2 coord.
WindowRedraw (); // Redibujar la imagen
)
return; // Salir de inicio ()
)
//------------------------------------------------- ---------- 5 --
int deinit () // función especial deinit ()
(
ObjectDelete ( "Obj_Reg_Ch"); // Eliminar el objeto
return; // Salir deinit ()
)
//------------------------------------------------- ---------- 6 --
int Crear () // función definida por el usuario ..
(// .. Objeto de creación
datetime T1 = Tiempo [Len_Cn - 1]; // Definición de 1 ª vez coord.
datetime T2 = Tiempo [0]; // Definición de tiempo 2 coord.
ObjectCreate ( "Obj_Reg_Ch", OBJ_REGRESSION, 0, T1, 0, T2, 0); // Creación
ObjectSet ( "Obj_Reg_Ch", OBJPROP_COLOR, Col_Cn); // Color
ObjectSet ( "Obj_Reg_Ch", OBJPROP_RAY, false); // Ray
ObjectSet ( "Obj_Reg_Ch", OBJPROP_STYLE, STYLE_DASH); // Estilo
ObjectSetText ( "Obj_Reg_Ch", "Creado por el moveobjects EA", 10);
WindowRedraw (); // Imagen nuevo trazado
)
//------------------------------------------------- ---------- 7 --
```

El moveobjects.mq4 AE algoritmo implica que un objeto se vincula una vez que permanecerá en la pantalla durante todo el tiempo de la ejecución del programa. En tales casos, es razonable utilizar una función definida por el usuario (en este caso es Crear (), bloque 6-7) para la creación de un objeto, la función puede Ð ¼ Ñƒ llamada desde el programa en cualquier momento cuando sea necesario. Para dibujar un objeto dos coordenadas de tiempo son necesarias (T1 es la de coordinar el objeto de la frontera izquierda, T2 - que el derecho de frontera):

```
datetime T1 = Tiempo [Len_Cn - 1]; // Definición de 1 ª vez coord.
datetime T2 = Tiempo [0]; // Definición de tiempo 2 coord.
```

En este ejemplo el borde derecho del objeto debe ser siempre cero en el bar, es la razón por la que el valor de coordinar el segundo corresponde al tiempo de apertura de la barra de cero. Coordinar la izquierda se calcula en función del número de bares establecidos por el usuario (variable externa Len_Cn) y se define como el tiempo de apertura de un bar con el correspondiente índice. Por ejemplo, si el canal tiene una duración de 50 bares, coordinar la izquierda será igual al tiempo de apertura de un bar con el índice 49.

En las siguientes líneas de la función definida por el usuario Crear () OBJ_REGRESSION el objeto se crea utilizando ObjectCreate (), entonces es necesario propiedades de los objetos creados son creados por las ObjectSet () (color preestablecido por un usuario en una variable externa, prohibido extraer como un rayo, estilo de línea - punteada). En la línea:

```
ObjectSetText ( "Obj_Reg_Ch", "Creado por el moveobjects EA", 10);
```

una descripción de texto se le asigna al objeto. A diferencia de los anteriores analizados OBJ_LABEL, el texto de descripción OBJ_REGRESSION no se muestra. La descripción de objetos gráficos se pueden ver en la ficha de propiedades del objeto. Esto es muy conveniente en aplicación práctica para distinguir entre objetos creados en un programa de forma manual los adjunto:

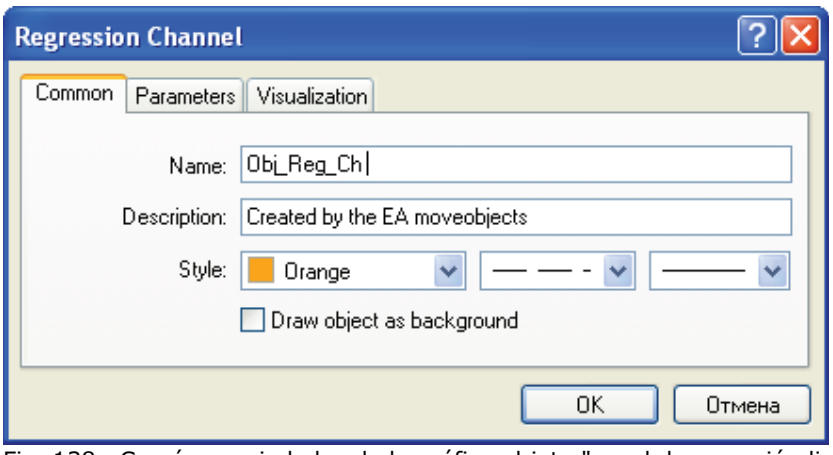


Fig. 138. Común propiedades de la gráfica objeto "canal de regresión lineal" creado por la AE moveobjects.mq4.

Esta es una función más utilizada para el nuevo trazado de la actual gráfica:

```
WindowRedraw (); // Imagen nuevo trazado
```

WindowRedraw ()

```
WindowRedraw vacío ()
```

La función de la fuerza redibuja el gráfico actual. Normalmente se utiliza después de propiedades del objeto se modifiquen.

Normalmente, los objetos gráficos se muestran por el Terminal de Usuario en la secuencia de entrada de nuevas ticks. Es por ello que, si no utilizamos WindowRedraw (), los cambios en las propiedades de objetos se hacen visibles para el usuario en la próxima tick, es decir, el hecho de mostrar siempre una tick tarde. El uso de WindowRedraw () le permite rehacer la fuerza a todos los objetos en un momento necesario, por ejemplo, inmediatamente después de las propiedades de objetos han sido cambiados. En un caso general, si las propiedades de varios objetos se cambian en el programa, basta con utilizar la función WindowRedraw () sólo una vez, después de las propiedades del último de los objetos han sido cambiados.

La función definida por el usuario es la primera llamada de la función especial init (). En el momento de unir la AE a la ventana de símbolo, la ejecución de init () se iniciará, lo que se traduce en que el objeto gráfico de regresión lineal del canal se mostrará en la ventana de símbolo.

Se pueden presentar dos situaciones se consideran a la función start (): (1) el objeto de vez en cuando ha sido suprimido por el usuario (bloque 3-4) y (2) es necesario para mover el objeto a la derecha cuando una nueva barra de cero es formado (bloque 4-5). Para detectar si el objeto gráfico está disponible en este momento, es suficiente con sólo pedir al valor de uno de sus coordenadas. Si el objeto existe, la función ObjectGet () devolverá un valor determinado que se corresponde con el pedido y coordinar la función GetLastError () devolverá el valor cero (es decir, no se ha producido un error al solicitar la coordenada). Sin embargo, si no hay objeto de dar el nombre en la ventana de símbolo, la función GetLastError () devolverá el código de error 4202, es decir, no dispone de objeto:

```
T2 = ObjectGet ( "Obj_Reg_Ch", OBJPROP_TIME2); // Solicitud de t2 coord.
Error = GetLastError (); // Obtener un código de error
```

Si el error de análisis mostró que no había objeto de ese nombre, significa que el programa se debe crear, después de haber notificado al usuario sobre inadmisibles las acciones (el programa no borra objetos, significa que el objeto ha sido borrado por el usuario) . Es por ello que, después de haber mostrado el mensaje, el programa pide a los que antes se consideraban función definida por el usuario Crear (), lo que da lugar a una nueva creación del objeto en la ventana de símbolo.

En el momento de la ejecución de la cuadra siguiente (4-5), el objeto gráfico ya se ha creado. Para decidir si debe ser movido, debe saber la posición del objeto en el momento actual. A tal efecto, es suficiente para analizar el valor obtenido previamente de coordinar la primera del objeto. Si este valor no coincide con el momento de la apertura de la barra de cero, para asignar nuevas coordenadas para el objeto.

Las coordenadas se cambian utilizando la función ObjectMove ():

```
ObjectMove ( "Obj_Reg_Ch", 0, Time [Len_Cn - 1], 0); // Nueva t1 coord.
ObjectMove ( "Obj_Reg_Ch", 1, Time [0], 0); // Nueva t1 coord.
```

Aquí, por primera coordenada (coordinar 0) del objeto llamado Obj_Reg_Ch, el valor de Time [Len_Cn-1] se establecerá, mientras que para la segunda coordenada (coordinar 1)-Tiempo [0]. El último parámetros entre los transferidos a la función ObjectMove () se especifica el parámetro 0. Este es el coordinar del precio que, según la descripción de la función, deben ser transferidos, pero, en este caso, será ignorado por el Terminal de Usuario. Como resultado de la ejecución de estas líneas, las propiedades del objeto considerado gráfica será cambiado. Como resultado de la próxima ejecución de la función WindowRedraw (), la gráfica objeto será vuelto a trazar la fuerza por parte del Terminal de Usuario - ahora de acuerdo con los nuevos valores de las coordenadas.

Así, en la ejecución de la función start (), el objeto gráfico de regresión lineal del canal será vuelto a trazar por el Terminal de Usuario cada momento en una nueva barra de formas, en su primera tick (ver Fig. 139). Tras la ejecución de la AE ha llegado a su fin, dado el objeto gráfico se eliminarán de la ventana de símbolo durante la ejecución de la función especial deinit () (es decir, el programa "sweep" de su lugar de trabajo después de que la obra se ha terminado).



Fig. 139. Viendo del canal de regresión lineal en la ejecución de la AE moveobjects.mq4.

En un caso general, puede crear y borrar objetos gráficos de acuerdo a algunas condiciones calculado en el programa. Puede mostrar el apoyo y las líneas de resistencia (OBJ_TREND), marca el momento de acercarse a los acontecimientos importantes con líneas verticales (OBJ_VLINE), indican las intersecciones de diferentes líneas o el plan de previsiones de los movimientos de precios utilizando objetos de texto (OBJ_LABEL y OBJ_TEXT), etc

Cabe señalar por separado que, en algunos casos, no hay necesidad de utilizar objetos gráficos. Por ejemplo, si desea que aparezca en la pantalla una gran variedad de un simple tipo de imágenes (por ejemplo, flechas), puede usar el indicador para esta líneas, después de haber establecido sus estilos en la forma correspondiente. Este enfoque le libre de la necesidad de rastrear las coordenadas de muchos objetos en el programa, sino que también le impide supresión ocasional de una imagen (los signos que muestran las líneas indicador puede ser ni suprimido ni seleccionados).

Funciones para trabajar con objetos gráficos

Función	Resumen de Información
ObjectCreate	Creación de un objeto con nombre predefinido, el tipo y las coordenadas iniciales se indica en el gráfico subventana. número de coordenadas objeto puede ser de 1 a 3 según el tipo de objeto. En caso de éxito la función devuelve TRUE, FALSE de otra manera.
ObjectDelete	Eliminar un objeto con el nombre indicado. En caso de éxito la función devuelve TRUE, FALSE de otra manera.
ObjectDescription	La función devuelve la descripción del objeto. Devuelve para objetos de la OBJ_TEXT y tipos OBJ_LABEL el texto que aparece en estos objetos.
ObjectFind	La función busca el objeto de dar el nombre. La función devuelve el índice de la ventana, a la que pertenece objeto buscado. En caso de fracaso, la función devuelve -1.
ObjectGet	La función devuelve el valor de la propiedad dada del objeto.
ObjectGetFiboDescription	La función devuelve la descripción del nivel Fibo objeto. La suma de los niveles depende del tipo de objeto que pertenece al grupo de Fibo objetos. El importe máximo de niveles es de 32.
ObjectGetShiftByValue	Las funciones calcula y devuelve el número de barras (el cambio relativo a la barra actual) para el precio dado. La barra número se calcula utilizando una ecuación lineal para la primera y segunda coordenadas. Se utiliza para las líneas de tendencia y objetos similares.
ObjectGetValueByShift	Las funciones calcula y devuelve el precio para el valor dado bar (el cambio en relación con el actual bar). El precio se calcula utilizando una ecuación lineal para la primera y segunda coordenadas. Se utiliza para las líneas de tendencia y objetos similares.
ObjectMove	Changing one of object coordinates on a chart. Objects can have from one to three anchoring points according to the object type. In case of success, the function returns TRUE, otherwise FALSE.
ObjectName	The function returns the object name according to its order number in the list of objects.
ObjectsDeleteAll	Deleting all object of the indicated type in the indicated chart subwindow. The function returns the number of deleted objects.
ObjectSet	Changing properties of an indicated object. In case of success the function returns TRUE, otherwise FALSE.
ObjectSetFiboDescription	The function assigns a new value to Fibonacci level. Number of levels depends on Fibonacci object type. Maximal number of levels is 32.

ObjectSetText	Changing object description. For objects OBJ_TEXT and OBJ_LABEL this description is displayed on a chart as a text line. In case of success the function returns TRUE, otherwise FALSE.
ObjectsTotal	Returns the total number of objects of the indicated type on a chart.
ObjectType	The function returns the type of an indicated object.

For the detailed description of these and other functions, please refer to Documentation at MQL4.community , MetaQuotes Software Corp. website or to "Help" section in MetaEditor.

Operaciones con Gráficos (Corregido hasta **)

En su trabajo práctico, un comerciante por lo general, se abre en una ventana de símbolo varias sub-ventanas que muestran los indicadores. No hay limitaciones en la colocación de indicadores, que pueden estar conectados en cualquier secuencia. La cantidad de subventanas en una ventana de símbolo no se limita tampoco. Cada subventana tiene su número. La ventana principal que contiene una gráfica de precios está siempre disponible, siendo su número 0. Cada uno de los indicadores subventana tiene un número, también. Las sub-ventanas se numeran en una secuencia simple que están numeradas por su exposición en la ventana de símbolo de arriba a abajo: subventana del indicador más cercano a la ventana principal tiene el número 1, la categoría inmediatamente inferior tiene el número 2, el próximo uno tiene número 3 , Etc



Fig. 140. Subventana lugares de la ventana de símbolo.

La cantidad de sub-ventanas pueden ser fácilmente calculadas utilizando la siguiente función:

```
int  WindowsTotal()
```

La función devuelve la cantidad de sub-ventanas de indicadores situadas en la gráfica, incluido el gráfico principal. El número mayor (de las subventanas mas bajas) es siempre uno menos que la cantidad total de sub-ventanas (incluida la ventana principal como el número 0) en la ventana de símbolo. Si, en la situación se muestra en la Fig. 140, hacemos una llamada para la ejecución de la función **WindowsTotal ()** desde cualquier aplicación, el valor devuelto será igual a 3, mientras que el mayor número (de las subventanas más bajas) es de 2.

La secuencia numérica que se ha descrito anteriormente se mantiene, si se añade un nuevo indicador en una subventana ya existente ó se suprime una subventana desde el símbolo de la ventana. Si se añade una nueva subventana, se mostrará debajo de todas las demás sub-ventanas y su número será uno más que el de la última ventana por encima de ella. Si se elimina una subventana de la ventana de símbolo, todas las sub-ventanas de debajo de ella será automáticamente renumeradas. El número de cada uno de ellos será reducido en un 1.

En MQL4, es posible crear objetos gráficos (y cambiar sus propiedades) en cualquiera de las sub-ventanas existentes. A tal efecto, en la función ObjectCreate () el parámetro 'windows' esta siempre, según que objeto se crea en subventana dada de la ventana de símbolo. El número actual de la subventana puede calcularse utilizando la siguiente función:

```
int WindowFind(string nombre)
```

La función devuelve el número de la subventana gráfica que contiene el indicador denominado como 'nombre', si se ha encontrado. De lo contrario, devuelve -1. La función también devoverá -1, si un indicador personalizado es búscado por sí mismo durante la inicialización init ().

Parámetros:

name - nombre corto del indicador.

La cantidad de subventanas en una ventana de símbolo puede cambiar en cualquier momento, si el usuario elimina un indicador. Es por ello que el algoritmo de una aplicación que soporta la supervisión de objetos gráficos debe realizar continuamente el seguimiento del número de ventanas, en la que se muestran los indicadores.



Problema 34. Usando objetos gráficos, mostrar mensajes para informar sobre los datos recibidos de dos indicadores. Si el indicador correspondiente se vincula a la ventana de símbolo, mostrar el objeto grafico en la ventana del indicador. De lo contrario, se mostrará en la ventana principal.

Para resolver el problema, vamos a elegir los indicadores RSI y Momentum. El algoritmo general que se construye en un Asesor Experto se reduce a esto. En la función init (), se pueden especificar textos que se visualizan en la pantalla de acuerdo con el indicador de lecturas, es decir, los cálculos para ser ejecutado una sola vez en el programa. En la función start (), se debe calcular lecturas del indicador detecta la disponibilidad de sub-ventanas y sus números y, entonces, según la situación, muestra uno u otro mensaje a una u otra subventana. En la ejecución de la función deinit (), es necesario eliminar todos los objetos gráficos creados durante el trabajo del programa. A continuación se muestra el nombre AE charts.mq4 que controla objetos gráficos en la sub-ventanas de una ventana de símbolo.

```
//+------------------------------------------------------------------+
// charts.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+ 1 --
int   Win_Mom_old=0,                    // Old number of subwindow Moment.
      Win_RSI_old=0;                    // Old number of subwindow RSI
color Color[5];                         // Declaration of the color array
string Text[5];                         // Declaration of the string array
//+------------------------------------------------------------------+ 2 --
int init()                              // Special function init()
  {
   Win_RSI_old=0;                       // Technical moment
   Win_Mom_old=0;                       // Technical moment

   Text[0]= "RSI(14) is below 30. Buy";       // Texts for situations RSI
   Text[1]= "RSI(14) is above 70. Sell";      // Texts for situations RSI
   Text[2]= "RSI(14) is between 30 and 70";   // Texts for situations RSI
   Text[3]= "Momentum(14) is growing";        // Texts for situations Momentum
   Text[4]= "Momentum(14) is sinking";        // Texts for situations Momentum
   Color[0]= DeepSkyBlue;               // Object color for ..
   Color[1]= LightPink;                 // .. different situations ..
   Color[2]= Orange;                    // .. of the indicator RSI
   Color[3]= Color[0];                  // The same colors for Momentum
   Color[4]= Color[1];                  // The same colors for Momentum

   Create_RSI(0);                       // Creation of the first object
   Create_Mom(0);                       // Creation of the second object
   Main();                              // Call to user-defined function
   return;                              // Exit init()
  }
//+------------------------------------------------------------------+ 3 --
int start()                             // Special function 'start'
  {
   Main();                              // Call to the user-defined function
   return;                              // Exit start()
  }
//+------------------------------------------------------------------+ 4 --
int deinit()                            // Special function deinit()
  {
   ObjectDelete("Obj_RSI");             // Deletion of the object
   ObjectDelete("Obj_Mom");             // Deletion of the object
   return;                              // Exit deinit()
  }
//+------------------------------------------------------------------+ 5 --
int Main()                              // User-defined function
  {
   int                                  // Integer variables
   Win_RSI_new=0,                       // New number of the subwindow RSI
   Win_Mom_new=0,                       // New number of the subwindow Moment.
   Ind_RSI, Ind_Mom;                    // Indexes for situations
   double                               // Real variables
   RSI,                                 // Value of RSI on bar 0
   Mom_0, Mom_1;                        // Value of Mom. on bars 0 and 1
//+------------------------------------------------------------------+ 6 --
   RSI=iRSI(NULL,0,14,PRICE_CLOSE,0);   // RSI(14) on zero bar
   Ind_RSI=2;                           // RSI between levels 30 and 70
   if(RSI < 30)Ind_RSI=0;               // RSI at the bottom. To buy
   if(RSI > 70)Ind_RSI=1;               // RSI on the top. To sell
//+------------------------------------------------------------------+ 7 --
   Win_RSI_new=WindowFind("RSI(14)");   // Window number of indicator RSI
   if(Win_RSI_new==-1) Win_RSI_new=0;   // If there is no ind., then the main window
   if(Win_RSI_new!=Win_RSI_old)         // Deleted or placed ..
```

```
    {                                // .. window of indicator RSI
     ObjectDelete("Obj_RSI");              // Deletion of the object
     Create_RSI(Win_RSI_new);              // Create an object in the desired window
     Win_RSI_old=Win_RSI_new;              // Remember this window
    }                                // Change the textual description:
   ObjectSetText("Obj_RSI",Text[Ind_RSI],10,"Arial",Color[Ind_RSI]);
//---------------------------------------------------------------------- 8 --
   Mom_0=iMomentum(NULL,0,14,PRICE_CLOSE,0);  // Value on zero bar
   Mom_1=iMomentum(NULL,0,14,PRICE_CLOSE,1);  // Value on the preceding bar
   if(Mom_0 >=Mom_1)Ind_Mom=3;              // Indicator line goes up
   if(Mom_0 < Mom_1)Ind_Mom=4;              // Indicator line goes down
//---------------------------------------------------------------------- 9 --
   Win_Mom_new=WindowFind("Momentum(14)");    // Window number of indicator Momen
   if(Win_Mom_new==-1) Win_Mom_new=0;         // If there is no ind., then the main window
   if(Win_Mom_new!=Win_Mom_old)             // Deleted or placed ..
    {                                // .. the window of Momentum indicator
     ObjectDelete("Obj_Mom");              // Deletion of the object
     Create_Mom(Win_Mom_new);              // Create an object in the desired window
     Win_Mom_old=Win_Mom_new;              // Remember this window
    }                                // Change the textual description:
   ObjectSetText("Obj_Mom",Text[Ind_Mom],10,"Arial",Color[Ind_Mom]);
//---------------------------------------------------------------------- 10 --
   WindowRedraw();                          // Redrawing the image
   return;                             // Exit the user-defined function
  }
//---------------------------------------------------------------------- 11 --
int Create_RSI(int Win)                     // User-defined function
  {                                  // ..of creation of an object
   ObjectCreate("Obj_RSI",OBJ_LABEL, Win, 0,0); // Creation of an object
   ObjectSet("Obj_RSI", OBJPROP_CORNER, 0);     // Anchoring to an angle
   ObjectSet("Obj_RSI", OBJPROP_XDISTANCE, 3);  // Coordinate X
   if (Win==0)
     ObjectSet("Obj_RSI",OBJPROP_YDISTANCE,20);// Coordinate Y
   else
     ObjectSet("Obj_RSI",OBJPROP_YDISTANCE,15);// Coordinate Y
   return;                               // Exit the user-defined function
  }
//---------------------------------------------------------------------- 12 --
int Create_Mom(int Win)                     // User-defined function
  {                                  // ..creating an object
   ObjectCreate("Obj_Mom",OBJ_LABEL, Win, 0,0); // Creation of an object
   ObjectSet("Obj_Mom", OBJPROP_CORNER, 0);     // Anchoring to an angle
   ObjectSet("Obj_Mom", OBJPROP_XDISTANCE, 3);  // Coordinate X
   if (Win==0)
     ObjectSet("Obj_Mom",OBJPROP_YDISTANCE, 5);// Coordinate Y
   else
     ObjectSet("Obj_Mom",OBJPROP_YDISTANCE,15);// Coordinate Y
   return;                               // Exit the user-defined function
  }
//---------------------------------------------------------------------- 13 --
```

Antes de examinar el código anterior, debería explicar los detalles de la operación del programa. Un objeto gráfico una vez creado (en este caso, uno que muestra un texto) se supone que debe estar presente en la pantalla continuamente. Su descripción textual se supone que debe caracterizar la situación. El contenido de la descripción textual debe ser cambiada en la ejecución de la función start (), en todos los ticks. Al mismo tiempo, cuando cambie entre marcos temporales de la ventana, para que el AE se adjunte, el programa pasa por las siguientes etapas: deinit (), init (), (en espera de una tick), y start (). Si el objeto es creado por primera vez durante la ejecución de start (), entonces, cada vez que conmuta a otro marco temporal, un cierto período de tiempo transcurrirá antes de que el objeto aparezca, el período de tiempo será igual al tiempo de espera del próximo tick. Esto es un gran inconveniente, sobre todo, cuando los marcos temporales son conmutados a menudo entre ellos.

En un programa bien construido, los mensajes necesarios se muestran en la pantalla en el momento de conectar el programa a la ventana de símbolo o en el momento de cambiar de marco temporal (es decir, antes de marcar un nuevo tick). A tal efecto, por regla general, es necesario llevar a cabo todas las acciones que deben realizarse en cada uno de los ticks en el lanzamiento de la función especial start () en la fase de ejecución de la función especial init (). Con el fin de no repetir el mismo código de programa en diferentes funciones especiales, el código se puede organizar como una función aparte. A tal efecto, la AE contiene la función definida por el usuario Main (). Esta se llama para ser ejecutada una vez en la fase de inicialización (bloque 2-3) y en todos los ticks durante la labor futura del AE (bloque 3-4).

En el programa (bloque 11-13), hay otras dos funciones definidas por el usuario - Create_RSI () y Create_Mom () destinadas a la creación y modificación de las propiedades del objeto. En la ejecución de la función init (), los objetos necesarios son creados usando estas funciones. La llamada a la función Main () en dar los resultados necesarios para las propiedades de los objetos (los objetos deseados con la descripción del color deseado se muestran en la ventana que se desee).

Vamos a examinar la función Main () (bloque 5-11) con más detalle. En el bloque 6-7 se calculan las lecturas del indicador RSI. Dependiendo de si el final del indicador está por encima de la línea 70, por debajo de 30, o dentro de la gama entre estos índices, uno u otro valor se asigna a la variable Ind_RSI. Luego, este valor se utiliza como un índice de arrays de Color [] y Text [] (en el bloque 7-8) para cambiar las propiedades del objeto gráfico denominado "Obj_RSI".

Bloque 7-8. El número de RSI ventana se calcula en la línea:

```
Win_RSI_new = WindowFind("RSI(14)");// Window number of indicator RSI
```

El valor de la cadena RSI (14) se utiliza como parámetro transferido. Este es el nombre corto del indicador, el número de lo que debería ser detectado. En este caso, compone el nombre toda la secuencia de caracteres en la línea dada, incluyendo paréntesis y dígitos. Cabe señalar que, en caso general, puede haber varios indicadores del mismo tipo en la ventana de símbolo, por ejemplo, RSI (14), RSI (21) y RSI (34). Cada subventana que muestra estos indicadores tiene su propio número. Esta es la razón por la que los indicadores técnicos se desarrollan de tal manera que cada uno de ellos forma el nombre corto de acuerdo con los valores de preset de parámetros ajustables. El nombre abreviado de cada indicador técnico coincide con el que aparece en la esquina superior izquierda de su subventana (el nombre corto de un indicador personalizado puede ser creado por el programador usando la función IndicatorShortName ()).Si el indicador buscado no se hayan puesto en la ventana de símbolo, la variable Win_RSI_new (el número de la subventana, en el que este objeto debe ser exhibidas en el actual momento) tendrá el valor de -1, es decir, no existente ventana. En este caso, el programa implica la muestra del objeto gráfico en la ventana principal del gráfico el cual el número es siempre 0:

```
  if(Win_RSI_new == -1) Win_RSI_new=0;// If there is no ind., then the main window
```

Durante sus operaciones, el usuario puede colocar un indicador que faltan o borrar una existente. Con el fin de informarse sobre qué acciones deben realizarse, el programa utiliza variables globales **Win_RSI_old** y **Win_Mom_old**. El valor de cada variable es el número de la subventana, en la que el objeto ha sido creado. Si los valores de las variables Win_RSI_new y Win_RSI_old no coinciden, esto significa que el indicador de la ventana se ha añadido (que no existía antes) o se ha suprimido (se dispone sobre el anterior tick). En ambos casos, el objeto creado anteriormente debe ser suprimido, y uno nuevo se debe crear en la ventana que se desee:

```
ObjectDelete("Obj_RSI");        // Deletion of the object
    Create_RSI(Win_RSI_new);        // Create an object in the desired window
```

Después de que el objeto se ha creado en la ventana numerada como Win_RSI_new, el valor igual al número de esta ventana se le asigna a la variable Win_RSI_old, es decir, el programa recuerda el número de ventana, en la que el objeto gráfico fue creado:

```
 Win_RSI_old = Win_RSI_new;      // Remember this window
```

Si los valores de las variables Win_RSI_new y Win_RSI_old coinciden, significa que es suficiente asignar una descripción textual al objeto (que se coloca ahora en la ventana necesaria). También se debe hacer, en caso de la creación de un nuevo objeto:

```
    ObjectSetText("Obj_RSI",Text[Ind_RSI],10,"Arial",Color[Ind_RSI]);
```

Cálculos similares se realizan para otra subventana que el indicador Momentum (bloques de 8 - 10). Al final de la función Main(), todos los objetos gráficos son vuelto a trazar, como consecuencia de la ejecución de WindowRedraw ().

Es fácil ver que programar el control sobre objetos gráficos en subventanas implica utilizar variables globales (también se pueden usar variables «static»). En tales casos, cuando se codifica un programa, se debe poner especial atención a que valores pueden adoptar las variables globales en diferentes situaciones y a lo que esto puede dar lugar. En el programa examinado anteriormente, las variables globales son zeroized en la función init () :

```
Win_RSI_old = 0;              // Technical moment
Win_Mom_old = 0;               // Technical moment
```

Estas líneas están incluidas en el programa debido al hecho de que las variables globales pierden sus valores, sólo si el usuario ha detenido la ejecución del programa de aplicación en la ventana de símbolo. Sin embargo, si el usuario ha ajustado las variables externas o ha conmutado otro marco temporal, el programa se somete a deinicialización y la consiguiente inicialización y se guardan los valores de las variables globales.

Vamos a considerar las operaciones del programa que no contiene estas líneas. Supongamos, ambos indicadores con la subventana números 1 y 2, respectivamente, se han colocado en la ventana de símbolo del momento en que el usuario cambia el marco temporal. En el ejemplo considerado, cuando deinitializing el programa, los objetos gráficos se borran. En la ejecución de la función especial init (), los objetos se crean en la ventana cero. Luego, a la ejecución de la función principal (), en los bloques de 7-8 y 9-10, el programa compara el número obtenido de la ventana, en la que los objetos deben ser colocados, y el número de la ventana, en la que los objetos se encontraban en el anterior tick. De hecho, el objeto ya ha sido colocado en la ventana cero, pero los valores de variables globales van a decir otro resultado: su número será 1 y 2. Como resultado, los objetos gráficos se mantendrán en la ventana principal, hasta que el usuario borre y alcance a los indicadores correspondientes. Para prevenir estos acontecimientos, el programa implica la anulación de las variables globales en la ejecución de la función init (). De este modo, los valores de estas variables se corresponden con la situación.

Como resultado de la ejecución del AE charts.mq4, pueden aparecer mostradas las siguientes combinaciones de ventanas y objetos gráficos:

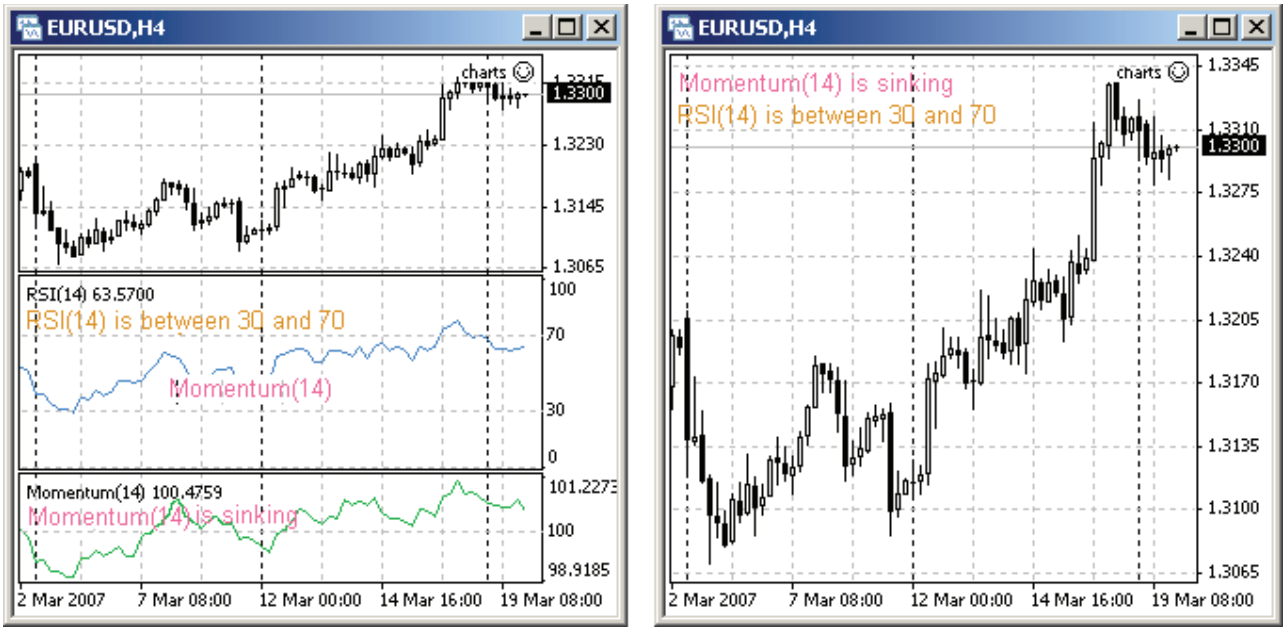


Fig. 141. Viendo objetos gráficos en la sub-ventanas de una ventana de símbolo.

Si hay dos indicadores en la ventana de símbolo, los correspondientes objetos graficos serán mostrados en sub-ventanas. Si ninguno de estos indicadores se coloca, entonces ambos objetos serán creados por el programa en la ventana principal. La adiccion o supresión de cualquier indicador (el nombre de los cuales se procesa en el programa) se traducirá en el movimiento del correspondiente objeto grafico en la ventana correspondiente. La adiccion o supresión de otros indicadores desde la ventana de símbolo no conlleva ninguna consecuencia.

Cabe señalar por separado que la alternativa de eliminar un objeto gráfico por parte del usuario no se considera en este programa. Un programa utilizado en su práctica comercial debe contener el análisis de esa situación con la posterior restauración de los objetos (véase la solución del problema 33).

Las funciones usadas en las operaciones con gráficos

Función	Resumen de Información
HideTestIndicators	La función pone una bandera de ocultar los indicadores que han sido llamados por el Asesor Experto. En la apertura del gráfico, después del testing, los indicadores marcados con esta bandera de clandestinidad no se muestran en el gráfico de prueba. Antes de cada llamada, el indicador está marcado con la que actualmente es la bandera de clandestinidad (sólo los indicadores que son llamados directamente desde el AE bajo prueba se pueden visualizar en el gráfico de prueba).
Periodo	Devuelve el valor de la cantidad del períodos de minutos de la gráfica actual.
RefreshRates	Actualiza los datos en las variables predefinidas y arrays timeseries. Esta función se usa, cuando un AE o un script ha pasdo un largo periodo de tiempro desde que ha hechos los ultimos cálculos y necesita una actualización de datos. Devuelve TRUE, si los datos se han actualizado con éxito. De lo contrario, devuelve FALSE. Los datos pueden permanecer sin actualizar solamente si se corresponden con el estado actual de la Terminal de Usuario.
Símbolo	Se devuelve una línea de texto con el nombre actual de símbolo.
WindowBarsPerChart	La función devuelve la cantidad de barras de ajuste en la ventana del gráfico actual.

WindowExpertName	Devuelve el nombre del EA, del script, del indicador personalizado o libreria que se está ejecutando, dependiendo sobre que programa MQL4 esta función ha sido llamada.
WindowFind	Nos devuelve el número de la subventana gráfica que contiene el indicador con el nombre 'name' si se ha encontrado. De lo contrario, devuelve -1. WindowFind () devuelve -1, si el indicador personalizado por el usuario se busca a sí mismo durante la inicialización init ().
WindowFirstVisibleBar	La función devuelve el número de la primera barra visible en la ventana del gráfico actual. Se debe considerar que el precio de los bares están numeradas en un orden inverso, a partir de la última a la primera. El actual bar, que es el último en la gama de precios, tiene índice 0. El bar más antiguo tiene el índice número de Bares -1. Si el número de la primera barra visible es 2 o más, inferior a la cantidad de bares visible en el gráfico, esto significa que el gráfico no esta completo y hay un espacio a la derecha.
WindowHandle	Nos devuelve el manejador de ventana por la ventana que contiene un gráfico dado. Si no hay gráfico con el símbolo y el marco temporal se abre en el momento de la llamada a función, devuelve 0.
WindowIsVisible	Devuelve TRUE, si el gráfico de la subventana es visible. De lo contrario, devuelve FALSE. El gráfico de la subventana puede estar oculto debido a la visibilidad de las propiedades del indicador que se le atribuye.
WindowOnDropped	Devuelve el índice de la ventana, en la que un AE, un script o un indicador ha sido colocado (“soltado”). Este valor será verdadero, sólo si los AEs, indicadores personales o script se vinculan usando un ratón (la tecnología de "arrastrar y colocar" o “arrastrar y dejar caer”). Para los indicadores personales que se inicializan (llamada de la función init ()), este índice no está definido. El índice devuelto es el número de la ventana (0 es la ventana gráfica principal, el indicador de sub-ventanas numera empezando por 1), en el cual el indicador usuario está trabajando. Durante la inicialización, un indicador personal puede crear su nueva subventana, y su número puede diferir de la de la ventana, en el que el indicador realmente ha sido “soltado”.
WindowPriceMax	Devuelve el valor máximo de la escala vertical de una subventana dada del gráfico actual (0 es la ventana gráfica principal, el indicador de sub-ventanas numera empezando por 1). Si no se especifica el índice de subventana, será devuelto el valor máximo de la escala de precios del gráfico principal.
WindowPriceMin	Devuelve el valor mínimo de la escala vertical de una subventana dada del gráfico actual (0 es la ventana gráfica principal, el indicador de sub-ventanas numera empezando por 1). Si no se especifica el índice de subventana, será devuelto el valor mínimo de la escala de precios de los principales gráfico.
WindowPriceOnDropped	Devuelve el valor del precio en un punto del gráfico, en el cual un AE o un script se ha “sotado” (soltado con el ratón). El valor será verdadero, sólo si la AE o el script se han movido usando el ratón (la tecnología de "arrastrar y soltar"). Este valor no está definido indicadores personales.
WindowRedraw	Se redibuja el gráfico actual fuerza. La función se utiliza generalmente después de las propiedades de objetos han sido cambiados.
WindowScreenShot	Se salva la pantalla actual de la gráfica en un fichero GIF. Si esto falla hacer una captura de pantalla, devuelve FALSE.
WindowTimeOnDropped	La función devuelve el valor de time (fecha y hora) en un punto de un gráfico, en el cual se inicia AE o una script se ha dejado caer. El valor será verdadero, sólo si la AE o el script se han movido usando un ratón (la tecnología de "arrastrar y soltar"). Este valor no está definido indicadores personales.
WindowsTotal	La función devuelve la cantidad de indicadores en la ventana del grafico, incluido el gráfico de la ventana principal.
WindowXOnDropped	Devuelve el valor de coordenada X en píxeles para un punto en el área de clientes de la ventana del gráfico, cuando un AE o un script han se ha

	"soltado". El valor será verdadero, sólo si la AE o el script se han movido usando el ratón (la tecnología de "arrastrar y soltar").
WindowYOnDropped	Devuelve el valor de coordenada X en píxeles para un punto en el área de clientes de la ventana del gráfico, cuando un AE o un script han se ha "soltado". El valor será verdadero, sólo si la AE o el script se han movido usando el ratón (la tecnología de "arrastrar y soltar").

Para la descripción detallada de estas y otras funciones, por favor, consulte la documentación en MQL4.community, MetaQuotes Software Corp sitio web o de "Ayuda" en la sección MetaEditor.

Funciones de cadenas (strings)

La operación más común con strings, adición (concatenación), se debatió en las sección operaciones y expresiones (Problema 3). En algunos casos, es necesario realizar otros cálculos relacionados con las cadenas o strings. El lenguaje MQL4 tiene una serie de funciones de cadena para trabajar con los valores de tipo *cadena.* Vamos a considerar el uso de algunos de ellos a través del siguiente ejemplo.



Problema 35. El colorear de las últimas 100 barras de un gráfico de velas (candlestick) es de la siguiente manera: velas negras en rojo, velas blancas en azul.

Una vela ó candlestick puede ser coloreado usando dos líneas: una línea fina de superposición de una vela a fin de que cubra todas las sombras, mientras que una línea gruesa debe llenar el cuerpo de la vela ó candlestick. En este caso, no podemos utilizar las líneas de un indicador personal, debido a que la muestra deben ser líneas verticales, es decir, construido con dos coordenadas (con las mismas coordenadas de tiempo), mientras que el indicador arrays nos permite almacenar un solo valor fijado en correspondencia con cada barra. Por lo tanto, la solución del problema viene mostrando una serie de tipo-simple de objetos OBJ_TREND que difieren en sus coordenadas y el tipo de trazo y color (véase el gráfico Objetos) a un gráfico de precio. **

En este caso, la AE se utiliza como un programa de aplicación, pero, en general, el algoritmo se puede implementar en un indicador personal. En su conjunto, el algoritmo es claro. La grafico debe ser de coloreado por primera vez, tan pronto como se vincula a la ventana de símbolo (durante la ejecución de init ()). El programa debe darse cuenta de los posibles cambios en la ubicación de objetos gráficos (accidentalmente un usuario puede mover o borrar uno de ellos) con todas los ticks próximos, y restablecer, si es necesario. Todos los objetos creados por el programa debe ser eliminado, tan pronto como termine el programa de funcionamiento (deinit ()).

Un usuario puede crear otros objetos en una ventana de símbolo, mientras que la AE está trabajando, por ejemplo, coloque el canal de desviaciones estándar, Fibo los niveles, líneas de apoyo, etc Por lo tanto, el algoritmo que nos permite distinguir creadas por el usuario y el programa-creado objetos deben llevarse a cabo en el programa. Esto es particularmente importante cuando se cierre el programa: es necesario eliminar sólo el programa-los objetos creados, mientras que las creadas por el usuario objetos debe permanecer inalterado. Cada objeto gráfico tiene sus propias propiedades que pueden coincidir en general. La única característica de la identificación de cualquier objeto es su nombre único (el uso de los mismos nombres está prohibida).

Se recomienda entrar a la información útil en el nombre del objeto, mientras que la componen, por lo que será posible detectar la ubicación y las propiedades del objeto. Por ejemplo, un nombre de objeto pueden contener un prefijo que la diferencia entre un programa-objeto creado a partir de otros. En este caso, es "Paint_". Además, es necesario diferenciar la "definido por el usuario" objetos de cualquier otro, también. Lo mismo tiempo una simple numeración (Paint_1, Paint_2) no puede utilizarse. El uso de este método de numeración de objetos, no se puede entender, a las que se oponen a que el objeto Paint_73 debe mostrarse. El bar que tiene el índice Paint_73 obtener el índice Paint_74, cuando un nuevo bar viene, cuando Paint_75 Índice otra nueva barra de procedencia, etc En tal caso, sería necesario suprimir y volver a crear todos los objetos en todos los nuevos bar. Esta solución (aunque es posible) es, evidentemente, muy áspero y costoso.

Cada objeto creado debe tener su tiempo de las coordenadas que se corresponden con el momento de la apertura de bar. Además, dos líneas debe ser exhibida en cada bar - una delgada línea y una línea gruesa. Es más cómodo para representar los nombres de los objetos creados por el programa de la siguiente manera:

Objeto name = Paint_2_2007.03.22 16:40, aquí:

Paint_ - prefix que distingue los objetos creados por el programa;

2_ - ya sea el número de objetos que se muestran en un bar (valor 1 o 2 es posible);

2007.03.22 16:40 - hora de coordinar única que caracteriza a la barra el objeto se muestra en.

Paint_ y 2_ son los valores de las variables Prefijo y Nom_Lin, respectivamente. El tiempo coordina pueden obtenerse por cada barra de transformación un valor *datetime* en una *cadena* de valor mediante la transformación de funciones:

TimeToStr ()

```
cadena TimeToStr (datetime valor, int mode = TIME_DATE | TIME_MINUTES)
```

La función transforma los valores que contienen el tiempo (en segundos) caducado desde 01/01/1970 *(datetime* valor) en una cadena del formato especificado *(cadena de* valor).

Parámetros:

valor - el tiempo en segundos caducado desde las 00:00 del 1 de enero de 1970;

modo - un nuevo modo de salida de datos. Puede ser una sola o una combinación de bandera:

TIME_DATE obtiene el resultado en la "yyyy.mm.dd";

TIME_MINUTES obtiene el resultado en la "hh: mi";

TIME_SECONDS obtiene el resultado en la "hh: mi: ss".

Vamos a considerar la AE strings.mq4 que gestiona los objetos para colorear de velas y ver cómo la TineToStr () se utiliza en este programa:

```
//------------------------------------------------- --------------------
// Strings.mq4
// El código debería ser usado para fines educativos únicamente.
//------------------------------------------------- --------------- 1 --
extern int GV_CantidadBarras = 100; // Número de barras
datetime Time_On;
string prefix = "Paint_";
//------------------------------------------------- --------------- 2 --
int init () // Espec. función init ()
(
int Ind_Bar; // Bar Índice
Time_On = Time [GV_CantidadBarras]; // Hora de la primera coloración
(Ind_Bar = GV_CantidadBarras - 1; Ind_Bar> = 0; Ind_Bar -) // ciclo Bares
(
Crear (Ind_Bar, 1); // Dibuja una línea delgada
Crear (Ind_Bar, 2); // Dibuja una línea gruesa
)
WindowRedraw (); // Imagen nuevo trazado
return; // Salir de inicio ()
)
//------------------------------------------------- --------------- 3 --
int start () // Espec. función start ()
(
datetime T1, T2; // 1 y 2 coordenadas tiempo
int error, Ind_Bar; // Código de error y la barra de índice
doble P1, P2 // 1 y 2 precio coordenadas
color Col // Color de objeto creado
//------------------------------------------------- --------------- 4 --
for (int Line = 1; Line <= 2; Line + +) // Línea tipo ciclo
(
Nom_Lin cadena = Line + "_"; // String con el número de línea
// Cadena Nom_Lin = DoubleToStr (Línea, 0 )+"_";// pueden ser tan
(Ind_Bar = 0;; Ind_Bar + +) // Bar ciclo
(
//------------------------------------------------- --------------- 5 --
T_Bar datetime = Time [Ind_Bar]; // Bar tiempo de apertura
if (T_Bar <Time_On) break; // No color fuera de las fronteras
Str_Time cadena = TimeToStr (T_Bar); // Tiempo de cadenas
His_Name cadena = Nom_Lin Prefijo + + Str_Time; // Objeto nombre
//------------------------------------------------- --------------- 6 --
T1 = ObjectGet (His_Name, OBJPROP_TIME1); // t1 coord.  consulta
Error = GetLastError (); // Código de error que reciben
if (error == 4202) // Si no hay un objeto: (
(
Crear (Ind_Bar, Line); // Objeto creación de la función llamada.
continuar; // Para la próxima iteración
)
//------------------------------------------------- --------------- 7 --
T2 = ObjectGet (His_Name, OBJPROP_TIME2); // t2 coord.  consulta
P1 = ObjectGet (His_Name, OBJPROP_PRICE1); // p1 coord.  consulta
P2 = ObjectGet (His_Name, OBJPROP_PRICE2); // p2 coord.  consulta
Col = ObjectGet (His_Name, OBJPROP_COLOR); // Color de consulta
if (T1! T_Bar = | | T2! T_Bar = | | // incorrecto coord. o color:
(Línea == 1 & & (P1! Alto = [Ind_Bar] | | P2! Baja = [Ind_Bar])) | |
(Línea == 2 & & (P1! Abierto = [Ind_Bar] | | P2! = Cerrar [Ind_Bar])) | |
(Open [Ind_Bar] Cerrar [Ind_Bar] & & Col! Roja =) | |
(Open [Ind_Bar] == Cerrar [Ind_Bar] & & Col! Verde =))
(
ObjectDelete (His_Name); // Eliminar objeto
Crear (Ind_Bar, Line); // Crear objeto corregir
)
//------------------------------------------------- --------------- 8 --
```

```
 )
 )
 WindowRedraw (); // Imagen nuevo trazado
 return; // Salir de inicio ()
 )
 //---------------------------------------------- ---------- 9 --
 int deinit () // Espec. deinit función ()
 (
 Name_Del cadena [1] // Array declarar
 int Quant_Del = 0; // Número de objetos, debe suprimirse
 int Quant_Objects = ObjectsTotal (); // Número total de todos los objetos
 ArrayResize (Name_Del, Quant_Objects); // array tamaño necesario
 for (int k = 0; k <= Quant_Del; i + +) // Eliminar los objetos con nombres ..
 ObjectDelete (Name_Del [i]); // ..  array que contiene
 return; // Salir deinit ()
 )
 //---------------------------------------------- ---------- 10 --
 Crear int (int Ind_Bar, int Line) // función definida por el usuario ..
 (// .. Creación de objetos
 color Color; // Objeto color
 T_Bar datetime = Time [Ind_Bar]; // Bar tiempo de apertura
 doble O_Bar = Abierto [Ind_Bar]; // Bar abierto precio
 doble C_Bar = Cerrar [Ind_Bar]; // Bar estrecha precio
 doble H_Bar = Alta [Ind_Bar]; // Bar precio máximo
 doble L_Bar = Baja [Ind_Bar]; // Bar precio mínimo

 Nom_Lin cadena = Line + "_"; // String - número de línea
 // Cadena Nom_Lin = DoubleToStr (Línea, 0 )+"_";// pueden ser tan
 Str_Time cadena = TimeToStr (T_Bar); // String - tiempo libre.
 His_Name cadena = Nom_Lin Prefijo + + Str_Time; // Nombre del objeto creado
 if (O_Bar <C_Bar) = Color Azul; // Elegir el color en función de ..
 if (O_Bar> C_Bar) = Color Rojo; // ..  parámetros de la barra
 if (O_Bar == C_Bar) = Color Verde;

 switch (línea) // o delgada línea gruesa
 (
 caso 1: // delgada línea
 ObjectCreate (His_Name, OBJ_TREND, 0, T_Bar, H_Bar, T_Bar, L_Bar);
 break; // Salida de interruptor
 caso 2: // línea gruesa
 ObjectCreate (His_Name, OBJ_TREND, 0, T_Bar, O_Bar, T_Bar, C_Bar);
 ObjectSet (His_Name, OBJPROP_WIDTH, 3); // Estilo
 )
 ObjectSet (His_Name, OBJPROP_COLOR, Color); // Color
 ObjectSet (His_Name, OBJPROP_RAY, false); // Ray
 ObjectSetText (His_Name, "Object es creado por la EA", 10); // Descripción
 return; // Salir función definida por el usuario
 )
 //---------------------------------------------- ---------- 11 --
```

Con el fin de crear objetos gráficos, la función definida por el usuario Crear () (bloques 10-11) se utiliza en el programa. Ind_Bar La variable que indica el índice de la barra de objeto debe ser creado, y la línea, el objeto número (línea 1 ó 2), se utilizan como parámetros asignables a esta función.

Tres componentes se utilizan cuando se forman el nombre del objeto a ser creado:

```
 His_Name cadena = Nom_Lin Prefijo + + Str_Time; // Nombre del objeto creado
```

El valor de la variable Prefijo es especificado por el programador en la cabeza parte del programa y no es modificado durante la ejecución del programa:

```
 string prefix = "Paint_";
```

El valor de la variable Nom_Lin se obtiene como resultado de los cálculos:

```
Nom_Lin cadena = Line + "_"; // String - número de línea
// Cadena Nom_Lin = DoubleToStr (Línea, 0 )+"_";// pueden ser tan
```

Aquí el valor del entero variable (durante el cálculo en la primera parte de la expresión) se transforma en el tipo de la más alta prioridad, a saber, en la *cadena* tipo. Como resultado de ello, recibe la Nom_Lin "1_" o "2_" valores dependiendo del valor de la variable de línea.

Con el fin de calcular el valor de la variable Str_Time la TimeToStr () la función de transformación de datos se utiliza:

```
Str_Time cadena = TimeToStr (T_Bar); // String - tiempo libre
```

Tenga en cuenta que la TimeToStr () tiene valores por defecto. En este caso, se trata de que estos valores son necesarios: "yyyy.mm.dd hh: mi", no hay necesidad de utilizar, además, segundo, porque el periodo mínimo es igual a 1 minuto.

También podríamos aplicar las siguientes Str_Time método de cálculo que se utilizará en el nombre del objeto:

```
Str_Time cadena = T_Bar;
```

En este caso, el Str_Time obtendría un valor igual al número de segundos desde 01.01.1970 caducado. Con el fin de ver la diferencia, podemos desarrollar un programa que contiene el siguiente código:

```
int init ()
(
String_Time cadena = TimeToStr (Time [0]); // Hora en el formato
String_Sec cadena = tiempo [0]; // Número de segundos
Alert ( "String_Time =", String_Time, "String_Sec =", String_Sec);
return;
)
```

El siguiente mensaje (en función del tiempo de cero barra de apertura), se mostrará en la pantalla como resultado de la ejecución del programa:

```
String_Time = 2007.03.22 19:10 String_Sec = 1174590600
```

La primera alternativa que se aplica en la strings.mq4 AE es un poco más informativo, por lo que la preferencia se da a él, en este caso (las alternativas son equivalentes en términos de la composición de un algoritmo).

El objeto llamado His_Name se crea en las líneas posteriores de la función definida por el usuario Crear (). Contiene la información sobre la barra de tiempo de apertura con los parámetros que se corresponden con el número de la "línea" línea y también el color en función de la barra de características. El valor de la descripción de texto se especifica para cada objeto, "Object es creado por EA", también.

La Crear () se llama en el programa de dos lugares: de la función especial de inicio () para la creación inicial de los objetos y de la función especial start () para re-crear el objeto, si es necesario, en caso de que se suprimido o modificado por el usuario. Los nombres de los objetos en start () (bloques 4-6) se forman de la misma manera que en otras partes del programa.

La primera de coordinar las considera objeto se define en el bloque 6-7. Si el objeto no se encuentra en este momento, será creada por el Crear (). Y si el objeto existe, sus otras coordenadas se determinará y la congruencia de sus parámetros con la barra de propiedades se verificará (bloque 7-8). El objeto será borrado y re-creado (con el mismo nombre) con la correcta propiedades, en caso de cualquier desajuste se detecta.

Otro problema es resuelto durante la ejecución de las deinit (): es necesario suprimir sólo los objetos que han sido creadas por el programa de la suma de todos los objetos en la ventana de símbolo. Esto se realiza en dos etapas: en la primera etapa, los nombres de todos los objetos que debe suprimirse son memorizados a la Name_Del [] matriz y, a continuación, se eliminarán en un ciclo. El número total de objetos en la ventana (incluidos los creados por el programa y se colocan manualmente por el usuario) se calcula utilizando el ObjectsTotal ():

```
int Quant_Objects = ObjectsTotal (); // Número total de todos los objetos
```

El número de barras de colores que se está configurada por el usuario en una variable externa, es decir, no se sabe de antemano cuántos objetos deben suprimirse. Por lo tanto, la cadena matriz que contiene los nombres de los objetos que se eliminen, se declara con el número de elementos igual a 1. Además, su tamaño es programáticamente cambiado - el número de elementos se incrementa con el número total de objetos.

```
ArrayResize (Name_Del, Quant_Objects); // array tamaño necesario
```

Con el fin de seleccionar los objetos que han sido creados por la EA, el deinit () contiene el ciclo 'for' que analiza los nombres de todos los objetos.

```
Obj_Name cadena ObjectName = (k); // Consulta nombre del objeto
```

El atributo que distingue a "nuestros" objetos de todos los demás es el "Paint_" prefijo, con el que el nombre de cada programa creado por objeto comienza. Para analizar un objeto nombre, debemos extraer la primera parte (en este caso, el 6 de símbolos) de la cadena variable siendo el único nombre del objeto, entonces deberíamos comparar este valor con que el prefijo de variable. Si coinciden, este objeto debe ser eliminado. Si no es así, no debería ser eliminado.

StringSubstr ()

```
StringSubstr string (cadena de texto, int inicio, int longitud = 0)
```

La función extrae la subcadena de la línea de texto a partir de la posición especificada. La función devuelve la copia de la subcadena extraída. De lo contrario, una cadena vacía se devuelve.
Parámetros:

texto - la línea de la subcadena se extraerá de;

inicio - la posición inicial de la subcadena. Puede variar de 0 a StringLen (texto) -1;

longitud - la longitud de la subcadena a extraer. Si el valor de este parámetro es inferior o igual a 0 o no es ni siquiera se especifica a continuación, la subcadena se extrae de la posición especificada hasta el final de la línea.

En el ejemplo considerado, la subcadena se extrae de el nombre del objeto de la siguiente manera:

```
Jefe cadena = StringSubstr (Obj_Name, 0, 6); // Extracto 6 primeros símbolos
```

En este caso, los 6 primeros símbolos se extraen de la cadena Obj_Name variable a partir de cero. Tenga en cuenta que el recuento de todos los índices (bares, arrays), las entradas en la lista de órdenes y también el número de la posición en la línea que comienza con 0, mientras que los compromisos cuantificados contar comienza con 1.

El extrajeron subcadena (una *cadena* de valor) se le asigna a la variable string Jefe. Si el nombre del objeto (y el objeto en sí mismo) es creado por el AE considera, el valor de la subcadena extraída será "Paint_". Si otro nombre es analizada, entonces el valor deseado será diferente. Por ejemplo, el valor de la subcadena extraída de la "StdDev canal 23109" nombre de objeto será el siguiente: "StdDev", y para el objeto llamado "Fibo 22800" será "Fibo 2".

En las líneas posteriores, el valor de la variable Jefe se compara con la variable que Prefijo:

```
if (Jefe == Prefijo) // El objeto comienzo ..
(// ..  con Paint_ se encuentra
```

Si estos valores son iguales entre sí, a continuación, analizó el nombre del objeto se pondrán a la matriz Name_Del [] para los nombres de los objetos a ser eliminado. En el próximo "por" el ciclo, todos los objetos, los nombres que figuran de la matriz, será borrado (cabe señalar por separado que es imposible eliminar todos los objetos durante el primer ciclo, ya que, en este caso , El número total de objetos y su numeración será cambiado cada vez que el objeto se suprime, lo que dará lugar a la omisión de algunos nombres de objetos).

El precio gráfico tendrá el siguiente aspecto durante la ejecución de las strings.mq4 EA:



Đ Đ ¸ Ñ. 142. Precio gráfico de color utilizando objetos gráficos (strings.mq4).

Además de los grupos de objetos que cubren el gráfico de cotizaciones, otros dos objetos colocados manualmente por el usuario se muestran, tal y como se muestra en la Fig. 142, son de regresión canal y Fibo los niveles. Los objetos creados por la AE se eliminarán, tan pronto como su ejecución está terminado, y los objetos creados por el usuario permanece en la ventana de símbolo. Este resultado se obtiene debido a la utilización de funciones de cadena en el programa. Permiten crear y analizar cadena de valores, incluidos los nombres de objetos gráficos.

Funciones de cadenas

Función	Descripción corta
StringConcatenate	Se trata de una cadena de los parámetros dado y lo devuelve. Los parámetros pueden ser de cualquier tipo. El número de parámetros no puede ser superior a 64.
StringFind	Subserie de la búsqueda. Nos devuelve el número de la posición en la línea deseada la subcadena comienza con, o -1, en caso de que la subcadena no se encuentra.
StringGetChar	Devuelve el valor del símbolo que se encuentra en la posición especificada de la línea.
StringLen	Nos devuelve el número de símbolos en la línea.
StringSetChar	Devuelve la copia de la línea con la modificación de valor del símbolo en la posición especificada.
StringSubstr	Se extrae la subcadena que comienza en la posición especificada en la línea de texto. La función devuelve la copia de la subcadena extraída, si es posible. De lo contrario, una cadena vacía se devuelve.
StringTrimLeft	La función corta el return de carro caracteres, espacios y símbolos de la tabulación de la parte izquierda de la cadena. La función devuelve la copia de la cadena modificada, si es posible. De lo contrario, una cadena vacía se devuelve.
StringTrimRight	La función corta el return de carro caracteres, espacios y símbolos de la tabulación de la parte derecha de la cadena. La función devuelve la copia de la cadena modificada, si es posible. De lo contrario, una cadena vacía se

	devuelve.

Funciones de transformación de datos

Función	Resumen de Información
CharToStr	Transformación del símbolo de código en un único símbolo de cadena.
DoubleToStr	Transformación del valor numérico en una cadena de texto que contiene la representación simbólica del número con exactitud el formato especificado.
NormalizeDouble	Redondeo el número con el de punto flotante a la precisión especificada. La calculado StopLoss, TakeProfit y también al aire libre prcie la órdenes en espera de ser ejecutadas de valores debe ser normalizado de acuerdo con la exactitud que se almacena en la variable definida dígitos.
StrToDouble	Transformación de la cadena que contiene la representación simbólica del número en el número de "doble" tipo (de doble precisión con el formato de punto flotante).
StrToInteger	Transformación de la cadena que contiene la representación simbólica en el número de la "int" tipo (integer).
StrToTime	Transformación de la cadena que contiene el tiempo y / o la fecha en la "yyyy.mm.dd [hh: mi]" formato en el número de la "datetime" tipo (número de segundos transcurridos desde 01.01.1970).
TimeToStr	Transformación del valor que contiene el tiempo expresado en segundos transcurrido desde el 01.01.1970 en la cadena de la "yyyy.mm.dd hh: mi".

Para obtener la información detallada sobre éstas y otras funciones, eche un vistazo a la documentación a MQL4.community, a MetaQuotes Software Corp sitio web o en la "Ayuda" de la sección MetaEditor.

Fecha y hora

El sistema de comercio en línea MetaTrader 4 utiliza las indicaciones de tiempo de dos fuentes - el local (PC) el tiempo y el servidor de tiempo.

Hora local - el tiempo que se fija en el PC local.

Servidor de tiempo - el tiempo que está configurado en el servidor.

TimeLocal ()

```
TimeLocal datetime ()
```

La función devuelve el PC local tiempo expresado en el número de segundos desde 00:00 caducado de 1 de enero de 1970. Nota: En las pruebas, la hora local y sigue el modelo coincide con el último modelo conocido servidor de tiempo.

Una gran mayoría de los acontecimientos que tienen lugar en el Terminal de Usuario se consideran con arreglo a los tiempos de servidor. El tiempo de tick viene, nuevo bar comienzo, el orden de apertura y cierre se considera con arreglo a los tiempos de servidor. Para obtener el valor del servidor de tiempo que se corresponde con la hora actual, la TimeCurrent () debe utilizarse:

TimeCurrent ()

```
TimeCurrent datetime ()
```

La función devuelve el último valor conocido el servidor de tiempo (el tiempo de la última cita próximos) expresado en segundos caducado desde las 00:00 del 1 de enero de 1970. El Terminal de Usuario actualiza el tiempo de la última cita próximos (junto con otras variables de entorno) antes de iniciar funciones especiales para su ejecución. Cada tick se caracteriza con su propio valor del servidor de tiempo que pueden obtenerse utilizando la TimeCurrent (). Durante la ejecución, este valor sólo puede ser cambiado como resultado de la RefreshRates () llamada a la función y sólo si la información se ha actualizado desde la última ejecución de la RefreshRates (), es decir, en caso de que los nuevos valores de algunas variables de entorno han llegado desde el servidor.

La barra de tiempo de apertura, tiempo [i], no coincide con el momento de marcar nuevos procedentes, por regla general. El tiempo de cualquier barra de calendario de apertura siempre es divisible por el calendario. Toda una cruz apareció dentro de un plazo es la formación de la barra, si no hay tick recepción dentro de un plazo, la barra no se formó dentro de los plazos.

Por ejemplo, los ticks llegan a la terminal a tiempo (servidor) t0 resultados en la formación de un bar con el tiempo de apertura igual a Time [i +2] (Fig. 143). En el momento especificado como el inicio del calendario no está de acuerdo con el momento t0, aunque puede accidentalmente de acuerdo con él, en general. La posterior ticks que llegan a la terminal en el mismo periodo de tiempo (en los momentos de t1 y t2) pueden modificar los parámetros de la barra, por ejemplo, el precio máximo o precio abierto, pero que ello no afecte a la barra de tiempo de apertura. La barra de hora de cierre no se considera en el sistema de comercio en línea MetaTrader 4 (oficialmente, el momento de marcar el último que se incluyan en un calendario o el tiempo a partir del próximo periodo de tiempo puede considerarse como la barra de la hora de cierre, tal y como se muestra en la Fig. 143).

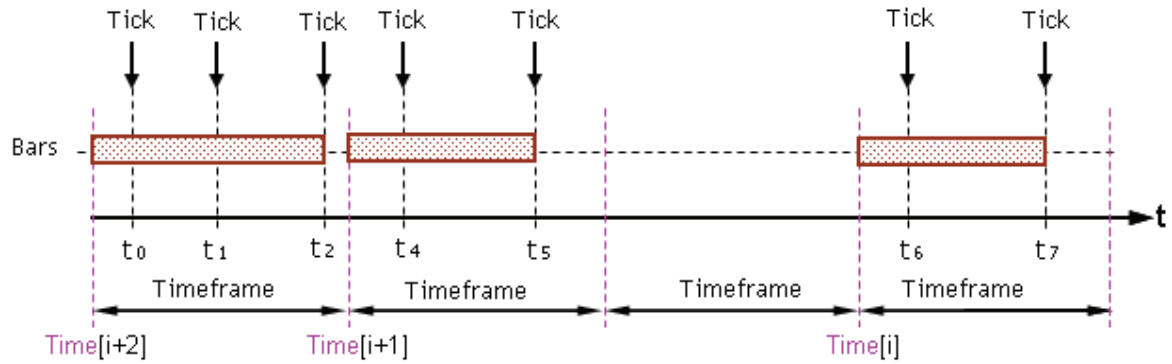


Fig. 143. Colegio de Abogados que forman la secuencia en la plataforma de comercio en línea MetaTrader 4.

Se muestra en la Fig. 143 que es posible que los bares no están formados en algunos períodos de tiempo que son iguales a los plazos. Así, entre el tiempo t5 de el tick y t6 procedentes de la próxima tick próximos, completa el calendario está lleno, por lo que el nuevo bar no ha sido formado en ese período de tiempo. De esta manera, el tiempo de apertura de barras puede variar desde el momento de la apertura de un bar adyacente de más de un calendario conjunto, pero siempre es divisible por un periodo de tiempo. Para demostrar la secuencia de formación de bar, podemos utilizar la AE timebars.mq4 que se reproduce el momento de llegar y marcar el momento de la apertura de bar:

```
//---------------------------------------------- --------------------
// Timebars.mq4
// El programa está destinado a ser usado como un ejemplo en MQL4 Tutorial.
//---------------------------------------------- --------------------
int start () // Espec. función start ()
(
Alert ( "TimeCurrent =", TimeToStr (TimeCurrent (), TIME_SECONDS),
"El tiempo [0] =", TimeToStr (Time [0], TIME_SECONDS));
return; // Salir de inicio ()
)
//---------------------------------------------- --------------------
```

Los resultados de la AE timebars.mq4 de trabajo se muestran en la Fig. 144. Es evidente que la primera tick en el período ordinario de tiempo de 1 minuto de duración se produjo en 14:29:12, al mismo tiempo una nueva barra se formó con el tiempo de apertura - 14:29:00. Tenga en cuenta que la columna derecha del cuadro de mensaje muestra el servidor de tiempo, la columna de la izquierda muestra la hora local.

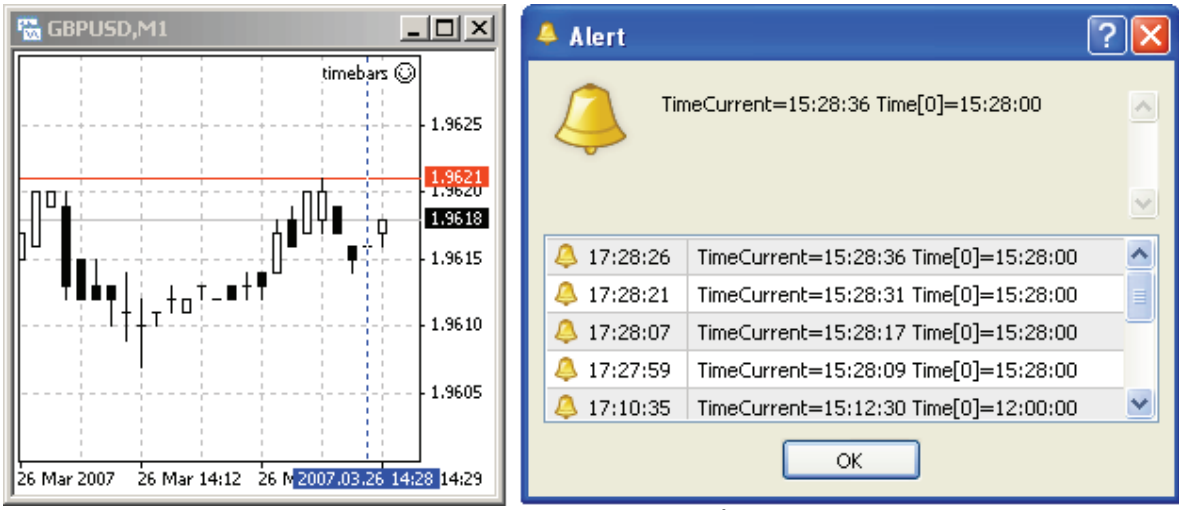


Fig. 144. Colegio de Abogados que forman la secuencia en la línea sistema de comercio MetaTrader 4.

En el caso de los ticks vienen rara vez (por ejemplo, el período comprendido entre el final del período de sesiones Europea y el comienzo del período de sesiones de Asia), se puede observar otro fenómeno durante la ejecución de timebars.mq4: el tiempo de apertura de los bares adyacentes pueden diferir unos de otros por más de 1 minuto (para un minuto de tiempo). Al mismo tiempo, la indexación de las barras se guarda en la suasignación, sin espacios.

El servidor de tiempo servidores en diferentes centros se ocupan pueden variar. La hora de comienzo y finalización órdenes se establece en cada servidor individual y que puede de acuerdo con el principio y el fin de regular el día. Algunos centros se ocupan, por ejemplo, los ajustes que realizan la apertura comercial en domingo a las 23:00 de servidor de tiempo. Esto se traduce en la formación incompleta de barras diarias, sus prácticas duración es igual a una hora (Fig. 145).

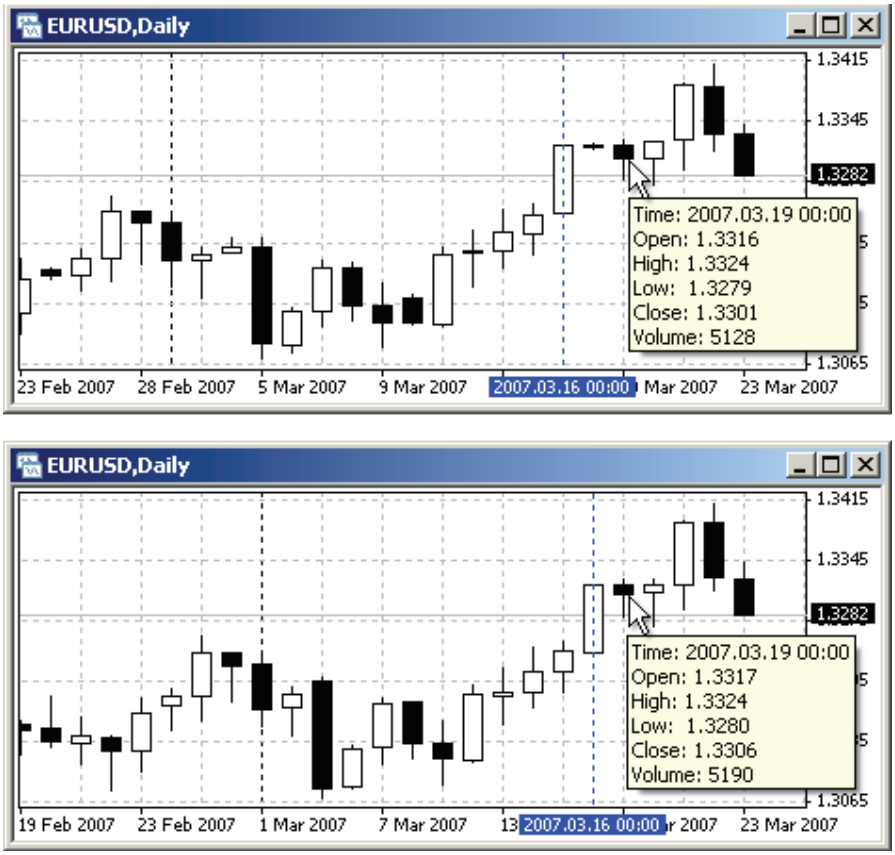


Fig. 145. Diferentes barra de la historia en los diferentes centros se ocupan.

El uso de fecha y hora de funciones es bastante fácil en MQL4. Algunos de ellos transformar el servidor y la hora local en cuestión de segundos caducado desde las 00:00 de 1 de enero de 1970 en un número que se corresponde con una hora, un día, etc Otras funciones devolver un número entero que se corresponde con la hora actual , Día, hora, etc

TimeSeconds (), TimeMinute (), TimeHour (), TimeDay (), TimeMonth (), TimeYear (), TimeDayOfWeek () y TimeDayOfYear () Funciones

Se trata de un grupo de funciones que devuelven el número de segundos caducado desde el inicio del minuto, o minuto, hora, día, mes, año, día de la semana y día del año para el período de tiempo especificado. Por ejemplo:

```
int TimeMinute (datetime tiempo)
```

La función devuelve minutos para el tiempo especificado.

Parámetros:

tiempo - la fecha expresada en número de segundos que decae desde las 00:00 del 1 de enero de 1970.

```
int TimeDayOfWeek (datetime tiempo)
```

Esta función devuelve el día de la semana (0 a domingo, 1,2,3,4,5,6) para la fecha especificada.

Parámetros:

tiempo - la fecha expresada en número de segundos que decae desde las 00:00 del 1 de enero de 1970.

El considerado funciones se pueden utilizar para análisis de toda la barra de tiempo de apertura, por ejemplo. El llamado AE bigbars.mq4 destinados a la búsqueda de barras de un tamaño que no sea inferior al tamaño especificado se muestra a continuación.

```
//---------------------------------------------- ----------------
// Bigbars.mq4
// El código debería ser usado para fines educativos únicamente.
//---------------------------------------------- --------------- 1 --
extern int Quant_Pt = 20 // Número de puntos
//---------------------------------------------- --------------- 2 --
int start () // Espec. función start ()
(
int H_L = 0; // Altura de la barra
for (int i = 0; H_L <Quant_Pt; i + +) // Ciclo de barras
(
H_L = MathAbs (Alta [i] - Baja [i]) / Point; // Altura de la barra
if (H_L> = Quant_Pt) // si la barra de alta no se encuentra
(
int YY = TimeYear (Tiempo [i]); // Año
int MN = TimeMonth (Tiempo [i]); // Mes
int DD = TimeDay (Tiempo [i]); // Día
int HH = TimeHour (Tiempo [i]); // Hora
int MM = TimeMinute (Tiempo [i]); // Minuto
Comentario ( "El último movimiento de precios más que", Quant_Pt, // Mensaje
"Pt sucedido", DD, ".", MN, "." AA ",", HH, ":", MM); // salida
)
)
return; // Salir de inicio ()
)
//---------------------------------------------- --------------- 3 --
```

El bigbars.mq4 AE busca el bar más cercano, cuya altura (diferencia entre máximo y mínimo) es mayor o igual al valor especificado en la variable externa Quant_Pt. La fecha y hora de la barra se encuentra outputted a la ventana de instrumento financiero por el comentario ().

Segundos (), Minuto (), horas (), Día (), TimeMonth (), TimeYear (), DayOfWeek () y DayOfYear () Funciones

Este es el grupo de funciones que devuelven el actual segundo, minuto, hora, día, mes, año, día de la semana y día del año para el último conocido servidor de tiempo. La última vez conocido servidor es el servidor de tiempo que se corresponde con el momento de lanzar el programa (lanzamiento de cualquier función especial de la Terminal de Usuario). El servidor de tiempo no se modifica durante la ejecución de la función especial.

```
INT horas ()
```

Devuelve la hora actual (0,1,2, .. 23) del último conocido servidor de tiempo. Tenga en cuenta que la última vez conocido servidor sigue el modelo durante las pruebas.

```
INT DayOfYear ()
```

Devuelve el día en curso del año (1 es el 1 de enero, .., 365 (6) es el 31 de diciembre), es decir, el día del año de la última conocido servidor de tiempo. Tenga en cuenta que la última vez conocido servidor sigue el modelo durante las pruebas.

La AE timeevents.mq4 que realiza algunas acciones tan pronto como el tiempo especificado viene puede ser utilizado como un ejemplo de uso de las funciones antes mencionadas.

```
//---------------------------------------------- ----------
// Timeevents.mq4
// El código debería ser usado para fines educativos únicamente.
//---------------------------------------------- ---------- 1 --
extern doble Time_Cls = 16,10; // Órdenes hora de cierre
Flag_Time bool = false; // Bandera, no hay mensajes aún
//---------------------------------------------- ---------- 2 --
int start () // Espec.  iniciar la función
(
int Cur_Hour horas = (); // Servidor de tiempo en horas
doble Cur_Min = Minuto (); // Servidor de tiempo en minutos
doble Cur_time = Cur_Hour + Cur_Min 100; // Hora actual
Alerta (Cur_time);
if (Cur_time> = Time_Cls) // Si la hora de la cita ha llegado
Albacea (); // ..  a continuación, realizar acciones concebido
return; // Salir de inicio ()
)
//---------------------------------------------- ---------- 3 --
int albacea () // función definida por el usuario
(
if (Flag_Time == false) // Si no hay mensajes aún
(// ..  luego informe (1 vez)
Alert ( "Important news tiempo. Cerrar órdenes.");
Flag_Time = true; // Ahora el mensaje ya ha aparecido
)
return; // Salir función definida por el usuario
)
//---------------------------------------------- ---------- 4 --
```

El servidor se calcula el tiempo en horas y minutos durante la ejecución de la función especial start () (bloques 2-3). La línea:

```
doble Cur_time = Cur_Hour + Cur_Min 100; // Hora actual
```

representa el actual servidor de tiempo, expresado en la variable real Cur_time. El uso de variables reales es conveniente en las operaciones de comparación:

```
if (Cur_time> = Time_Cls) // Si la hora de la cita ha llegado
```

Si la hora actual es mayor o igual al valor de Time_Cls especificados por el usuario, entonces el albacea () función definida por el usuario serán llamados para su ejecución. En este ejemplo, la función definida por el usuario coloca un mensaje con el comercio de recomendaciones. En general, esta función puede contener cualquier código, por ejemplo, hacer órdenes, enviar e-mails, crear objetos gráficos, etc

Funciones de fecha y hora

Función	Descripción
Día	Nos devuelve el día actual del mes, es decir, el día del mes de conocido el último servidor de tiempo.
DayOfWeek	Devuelve el número del índice del día de la semana (domingo-0, 1,2,3,4,5,6) de la última conocido servidor de tiempo.
DayOfYear	Devuelve el día en curso del año (1 es el 1 de enero, .., 365 (6) es el 31 de diciembre), es decir, el día de año de la última conocido servidor de tiempo.
Hora	Devuelve la hora actual (0,1,2, .. 23) de la última hora del servidor en el momento de inicio del programa (el valor no se modifica durante la ejecución del programa).
Minuto	Devuelve el minuto actual (0,1,2, .. 59) de la última hora del servidor en el momento de inicio del programa (el valor no se modifica durante la ejecución del programa).

Mes	Devuelve el número del mes en curso (1 de enero, 2,3,4,5,6,7,8,9,10,11,12), es decir, el número del mes de conocido el último servidor de tiempo.
Segundos	Nos devuelve el número de segundos caducado desde el comienzo de la actual minuto de la última hora del servidor en el momento de inicio del programa (el valor no se modifica durante la ejecución del programa).
TimeCurrent	Devuelve el último conocido servidor de tiempo (el tiempo de la última cita próximos), expresada en el número de segundos que pasaron desde la 00:00 Enero 1 º de 1970.
TimeDay	Devuelve el día del mes (1 - 31) para la fecha especificada.
TimeDayOfWeek	Nos devuelve el día de la semana (0 a domingo, 1,2,3,4,5,6) para la fecha especificada.
TimeDayOfYear	Devuelve el día (1 es el 1 de enero, .., 365 (6) es el 31 de diciembre) del año para la fecha especificada.
TimeHour	Devuelve la hora para el período de tiempo especificado.
TimeLocal	Nos devuelve el tiempo PC local se expresa en el número de segundos desde 00:00 caducado de 1 de enero de 1970.
TimeMinute	Devuelve minutos para el tiempo especificado.
TimeMonth	Nos devuelve el número del mes para el período de tiempo especificado (1 de enero, 2,3,4,5,6,7,8,9,10,11,12).
TimeSeconds	Nos devuelve el número de segundos transcurrido desde el inicio del período de tiempo especificado.
TimeYear	Devuelve el año para la fecha especificada. El valor devuelto puede ser dentro del rango de 1970-2037.
Año	Nos devuelve el año en curso, es decir, el año de la última conocido servidor de tiempo.

Para obtener la información detallada sobre éstas y otras funciones, consulte la documentación a MQL4.community, a MetaQuotes Software Corp sitio web o en la "Ayuda" de la sección MetaEditor.

Archivo de Operaciones

En MQL4, es posible trabajar con archivos que contienen un cierto conjunto de la información. Se puede llegar a ser necesario para escribir información en un fichero o para leer desde un fichero por varias razones.

Un archivo puede ser usado para entregar información a otro programa. En este caso, el archivo puede ser creado por un programa de aplicación y utilizados por ésta como un receptor de información. Por ejemplo, la historia comercial de una cuenta puede escribirse en un archivo a la ejecución de una solicitud. Este archivo puede ser abierto después de otro programa (por ejemplo, Excel para dibujar un equilibrio gráfico).

En otros casos, existe la necesidad de entregar alguna información, por ejemplo, las noticias calendario, a una solicitud. Un programa ejecutable (por ejemplo, un Asesor Experto) puede leer esta información desde el archivo previamente preparado y consideramos que es durante el cálculo para mostrar gráficos de los mensajes en la pantalla o para la fabricación de decisiones comerciales.

Los nombres de los archivos y directorios

El nombre de archivo de un grupo de trabajo debe estar formado de acuerdo con los requisitos del sistema operativo. El nombre de cualquier archivo utilizados en MQL4 consta de dos partes: el nombre del archivo y la extensión de archivo separados por un punto, por ejemplo, **News.txt.** Técnicamente, un nombre de archivo no tiene ninguna relación con el contenido del archivo, por lo que un nombre de archivo y extensión se pueden establecer voluntariamente por el programador. Un nombre de archivo normalmente se seleccionará de forma que representa la información que contiene el archivo.

La mayoría de los programas son automáticamente puesto en marcha en el PC del usuario, si el archivo es el doble-clic con el botón del ratón. De acuerdo con la extensión de archivo, el entorno operativo carga uno u otro programa que muestra el contenido del archivo. Por lo tanto, usted deberá asignar la extensión de archivo teniendo en cuenta el programa (si es necesario) que generalmente se utiliza para leer el fichero.

Los más populares tipos de archivo (el tipo está determinada por su extensión) son los siguientes:

--. **Txt** - archivo de texto, para su visualización debe usar el Bloc de Notas, Word, FrontPage, etc;

--. **Csv** - archivo para la construcción de gráficas en Excel;

--. **Htm** - archivo para ser visto en un navegador, es decir, Internet Explorer, Netscape Navigator, etc

Existen tres carpetas (con subcarpetas) que pueden contener los ficheros de trabajo:

-- Terminal_folder \ Expertos \ Historia \ **actual corredor ** - la historia de los archivos;

-- Terminal_folder \ Expertos \ Files \ - para uso común;

-- Terminal_folder \ Tester \ Files \ - para los archivos se utilizan para la prueba.

Un grupo de trabajo de archivos se pueden guardar en una de estas carpetas o en sus subcarpetas. En caso de que no dispone de carpeta en el momento de guardar el archivo, la carpeta se crea automáticamente por el Terminal de Usuario. Trabajar con archivos en otros directorios no está involucrado.

Modos de operaciones de archivos

La tecnología de interacción entre una aplicación y un fichero de trabajo tiene varios modos de transporte. En general, un archivo se puede abrir varios programas al mismo tiempo (en un PC o varios ordenadores conectados a la red). Al mismo tiempo, el entorno operativo proporciona el acceso completo al expediente, a saber, el derecho a leer el archivo y escribir la información en ella, sólo a un programa. Los otros programas sólo pueden leerlo. Por ejemplo, si My_text.doc ya ha sido abierto por un editor de texto, entonces todos los otros programas recibirán la notificación antes de abrir el archivo:

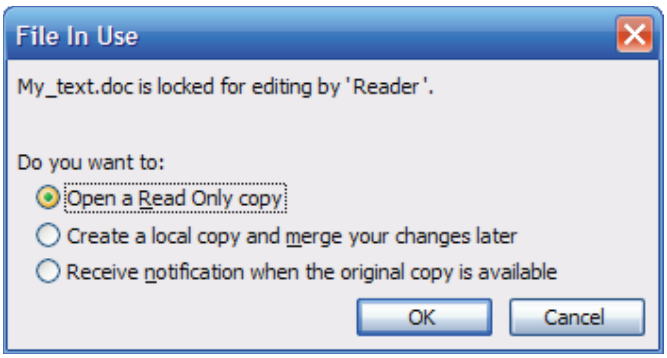


Fig. 146. Cuadro de diálogo que aparece cuando un programa intenta acceder al expediente que ya ha sido abierto por otro programa.

La ejecución de esta tecnología garantiza que un archivo no será modificado simultáneamente por dos diferentes programas. Con el fin de permitir a un programa aplicable a interactuar con un archivo, usted debe abrir ese archivo primero. El modo de abrir un archivo se especifica en el FileOpen ().

Un programa de aplicación puede abrir varios archivos de trabajo a la vez. Con el fin de permitir que el programa para diferenciar un archivo de otro el descriptor de archivo se fija de acuerdo a cada archivo abierto.

Descriptor de archivo - número único de expediente que se abre por el programa en este momento.

El FileOpen () regresará algún valor (este valor suele ser atribuida al 'manejar' variable), si un archivo es abierto con éxito. Ese valor es el descriptor de archivo. La mayoría de las funciones que tienen por objeto trabajar con los archivos de suponer el uso de un descriptor de archivo como uno de los parámetros formales.

Función FileOpen ()

```
int FileOpen (string filename, int modo, int delimitador = '; ")
```

La función se abre un archivo para inputing y / o la salida. La función devuelve un descriptor de archivo o -1, en caso de fallo. Los archivos sólo pueden ser abiertos en el **Terminal_folder \ Expertos \ Files** \ carpeta o en la **Terminal_folder \ Tester \ Files** \ carpeta (en caso de las pruebas EA) o en sus subcarpetas.

Parámetros:

nombre de archivo - el nombre del archivo;

modo - el modo de se abre el archivo, sino que pueden tener los siguientes valores (o sus combinaciones): FILE_BIN, FILE_CSV, FILE_READ, FILE_WRITE;

delimitador - el signo separador de archivos csv. Es »,« por defecto.

El modo de FILE_READ se abre el archivo implica que un archivo se utilizará sólo para ser leído por un programa. Un ensayo para abrir un archivo en este modo pueden fallar, si no se dispone de archivos con el nombre especificado.

El FILE_WRITE modo de disposición que implica la apertura de un archivo se utiliza para escribir en un programa de. Un intento abrir un archivo en este modo los resultados en la apertura de un expediente de una longitud cero. Incluso si hubiera alguna información en el expediente antes de abrir, será borrado. Un intento abrir un archivo en este modo pueden fallar, en caso de que el archivo se había abierto por otro programa (en el modo de escritura).

Está permitido abrir un archivo en el FILE_READ | FILE_WRITE modo. Este modo implica la posibilidad de leer y escribir a un archivo. Este modo se utiliza, si es necesario añadir alguna información al fichero que contiene ya algunas otras informaciones. La función implica el uso obligatorio de uno de los modos de transporte, FILE_READ o FILE_WRITE, o su combinación.

El modo de FILE_BIN se abre el archivo define la transformación de un grupo de trabajo como un archivo binario. El modo de FILE_CSV se abre el archivo define la transformación de un grupo de trabajo como un archivo de texto. La función incluye el uso obligatorio de uno de los FILE_BIN o FILE_CSV modos. El uso simultáneo de FILE_BIN y FILE_CSV modos está prohibido

La función requiere la combinación obligatoria de FILE_READ, FILE_WRITE o FILE_READ | FILE_WRITE modos con el FILE_BIN o FILE_CSV modo. Por ejemplo: es necesario utilizar la combinación de FILE_CSV | FILE_READ para leer la información de un archivo de texto, y es necesario utilizar la FILE_BIN | FILE_READ | FILE_WRITE combinación para añadir una entrada a un archivo binario.

No más de 32 archivos se pueden abrir simultáneamente dentro de un módulo ejecutable (de un programa aplicable, por ejemplo, un Asesor Experto). Los descriptores de los archivos que se abren en el módulo no se puede pasar a otros módulos (librerías).

Contenido del fichero de entradas

La información que las entradas están escritos en un archivo sin espacios con cualquier combinación de modos de transporte. La información que se añadirán uno por uno cuando se usa el modo de FILE_BIN para formar un archivo. Dependiendo del tipo de información que se escribe en un archivo (y las funciones que se utilizan para hacerlo) los símbolos que representan la combinación final de la línea ("\ r \ n") puede ser escrito entre los grupos de entradas. La información que las entradas están separadas por separadores de archivo (por lo general ";") cuando la formación de un archivo en el modo de FILE_CSV, y los grupos de entradas (que componen una línea) se separan con la combinación de símbolos que representan el final de la línea (" \ r \ n ").

Archivo separador - símbolo especial, la entrada que está escrito en un archivo separado para las líneas de información.

El archivo separador se utiliza para separar la información de entradas sólo en los archivos de csv.

El principio común para las entradas composición en cualquier archivo es que estas entradas se agregan de acuerdo con la secuencia sin espacios. Adecuadamente, la entrada consiste en secuencia continua de símbolos. Cualquier archivo puede ser leído por cualquier programa y (de acuerdo con las normas implementado en él) se pueden mostrar de alguna forma en la pantalla. Por ejemplo: tenemos el File_1.csv archivo que contiene:

```
int FileOpen (string filename, int modo, int delimitador = '; ")
```

File_1.csv El archivo se mostrará en diferentes formas en diferentes editores de texto:

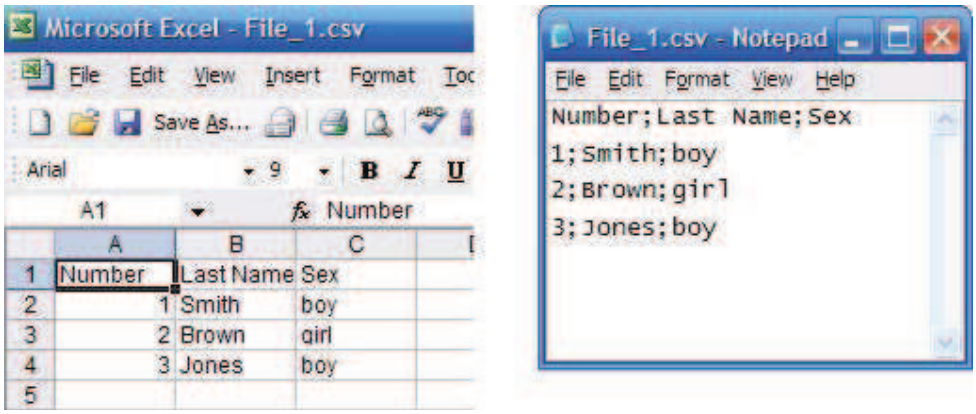


Fig. 147. File_1 representación en los diferentes programas (File_1.csv).

En este caso, el "\ r \ n" símbolo combinación fue interpretado por cada uno de los programas (Excel y Bloc de notas), como las pruebas para el formato de secuencia: la secuencia de símbolos está representado en la siguiente línea después de la "\ r \ n" combinación de símbolos, y el "\ r \ n" combinación en sí no aparece en la ventana de edición. Al mismo tiempo, Excel es un editor de mesa, por lo que el "," símbolo fue interpretado por el programa como separador de información a las columnas. Llamar la atención que la "," símbolo no aparece en la ventana de Excel. Bloc de notas es un editor de texto. Las normas aplicadas en el mismo no suponga la división de información en columnas, por lo que el "," símbolo no fue interpretado como un archivo separador, pero es interpretado como una parte de la información, por lo que se visualiza en la pantalla.

El especificado símbolos (";" y "\ r \ n") se utilizan para separar las entradas en MQL4.

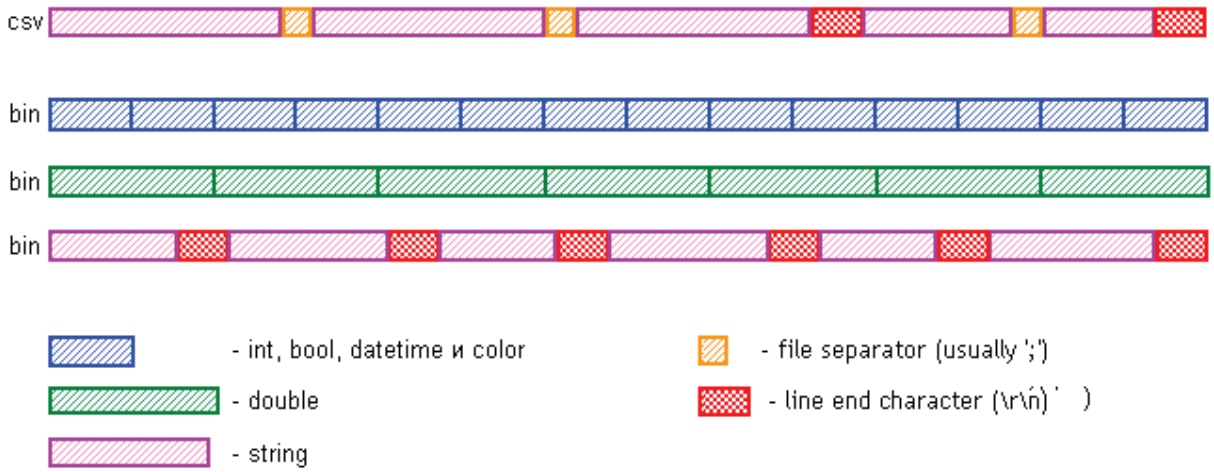


Fig. 148. Variedad de entradas en los ficheros de trabajo.

La estructura de la información por escrito en distintos tipos de archivos está representado en la fig. 148. La línea superior muestra el archivo csv-contenidos, la parte inferior tres líneas muestran la estructura de archivos binarios. Todos estos archivos se componen de acuerdo con las normas de una u otra función de la escritura en el archivo.

Una entrada en el archivo csv-es la secuencia de valores de cadena (string tipo) que están separadas con el expediente de separación o con el signo de la final de la línea. Ambos se interpretan como una señal del fin del valor informativo leer parte al leer la información (utilizando MQL4 función estándar para lectura de archivos). La cadena de valor puede tener la diferente longitud y se desconoce cuánto símbolos están ahí, por lo que la lectura se realiza antes de que uno de los separadores se encuentra.

Las entradas en dos tipos de binarias de archivos binarios representan las secuencias de datos sin ningún tipo de separadores. Esta secuencia de la escritura en se rige por la longitud fija de un banco de datos de diferentes tipos: 4 bytes de datos de la "int", "bool", "datetime" y "color" tipos, y 8 bytes (o 4 bytes, dependiendo de los parámetros de la función de escritura) para una base de datos de "doble" de tipo. En este caso, no hay necesidad de separadores, ya que la lectura se realiza por la función estándar para la lectura de los datos de un determinado tipo con una determinada longitud. El último (el de abajo en la fig. 148) archivo binario contiene los datos de tipo cadena que está separada con el fin de la línea de señal.

Archivo puntero - una posición en el archivo de la lectura de la siguiente parte de valor.

El "Archivo puntero" concepto es el mismo con el "cursor" noción. El puntero del archivo se define con la posición en el archivo. Por lo que respecta a la lectura va por el puntero se mueve a la derecha por una o varias posiciones.



Problema 36. Lea la información acerca de la importante noticia de los autos y mostrar la gráfica de objetos en el gráfico de precios (líneas verticales), de acuerdo al tiempo de publicación de noticias.

Deje que el **Terminal_Folder \ Expertos \ Files** \ carpeta contiene la News.csv trabajo de archivo con el siguiente contenido:

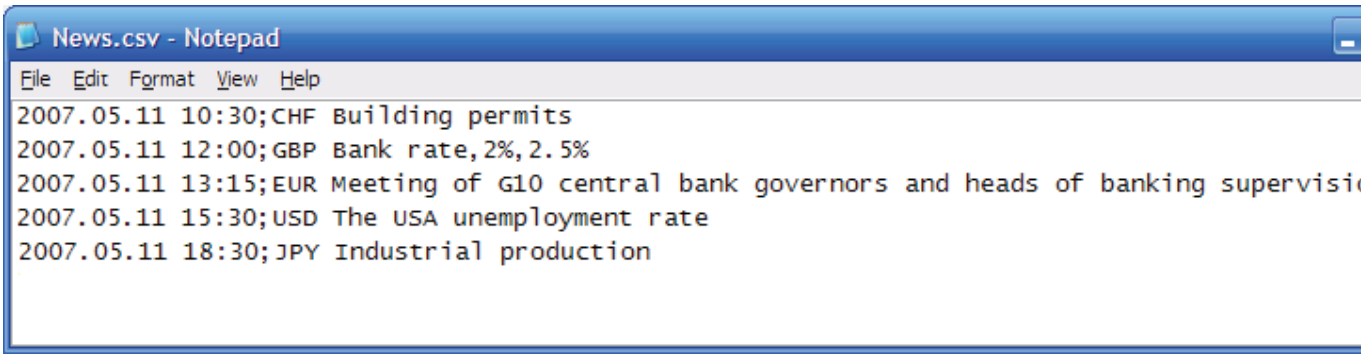


Fig. 149. Contenido del fichero de trabajo News.csv.

En este caso, el archivo contiene información acerca de los cinco eventos que va a suceder en los distintos países en un momento diferente. Cada línea contiene dos entradas. La primera entrada es la cadena de valor que representan la información sobre la fecha de una hora del evento. La segunda entrada es el texto de descripción del evento. Tres primeros símbolos de la segunda entrada incluirá el nombre de la moneda (el país) que el evento se refiere.

La solución consta de dos partes. En primer lugar tenemos que leer la información del archivo de trabajo y, a continuación, utilizar el recibido valores como las coordenadas de los objetos gráficos. La lectura de información se realiza por la FileReadString ().

FileReadString ()

```
FileReadString cadena (int manejar, int longitud = 0)
```

La función lee la línea desde la posición actual del archivo. Es adecuado tanto para CSV y archivos binarios. La línea será leída hasta el separador se cumple en el archivo de texto. El número específico de símbolos serán leídos en los ficheros binarios. Con el fin de recibir la información sobre un error que usted debe llamar al GetLastError ().

Parámetros:

manejar - el descriptor de archivo que sea devuelto por el FileOpen ();

longitud - el número de caracteres a leer.

La necesidad de noticias en el procesamiento de la información aparece una sola vez al comienzo de la negociación, por lo que, en este caso, podemos utilizar una scriptpara solucionar el problema 36. El timetablenews.mq4 script se destina a leer la información del archivo y la visualización gráfica de objetos en la ventana de símbolo.

```
//------------------------------------------------- --------------------
// Timetablenews.mq4
// El código debería ser usado para fines educativos únicamente.
//------------------------------------------------- --------------- 1 --
int start () // Espec. función start ()
(
//------------------------------------------------- --------------- 2 --
int Handle, // descriptor de archivo
STL; // Estilo de la línea vertical
string file_name = "News.csv", // Nombre del archivo
Obj_Name, // Nombre del objeto
Instr, // Nombre de la moneda
Uno, dos, // 1 º y 2 º nombre de la instr.
Texto, // Texto de descripción de la cita
Str_DtTm; // Fecha y hora del evento (línea)
datetime Dat_DtTm; // Fecha y hora del evento (fecha)
color Col // Color de la línea vertical
//------------------------------------------------- --------------- 3 --
Asa = FileOpen (file_name, FILE_CSV | FILE_READ, ";"); // se abre el archivo
if (Handle <0) // no se abre el archivo
(
if (GetLastError () == 4103) // Si el archivo no existe, ..
Alert ( "No fichero llamado", file_name); // .. informar a comerciante
else // Si cualquier otro error se produce ..
Alert ( "Error al abrir el archivo", file_name); // .. este mensaje
PlaySound ( "Bzrrr.wav"); // acompañamiento de sonido
return; // Salir de inicio ()
)
//------------------------------------------------- --------------- 4 --
while (FileIsEnding (Asa) == false) // Mientras que el puntero del archivo ..
(// .. No es al final del archivo
//------------------------------------------------- --------- 5 --
Str_DtTm = FileReadString (Asa); // Fecha y hora del evento (fecha)
Texto = FileReadString (Asa); // Texto de descripción de la cita
if (FileIsEnding (Asa) == true) // puntero del archivo está al final
break; // Salir de lectura y dibujo
//------------------------------------------------- --------- 6 --
Dat_DtTm = StrToTime (Str_DtTm); // Transformación del tipo de datos
Instr = StringSubstr (Texto, 0, 3); // Extracto tres primeros símbolos
Uno = StringSubstr (símbolo (), 0, 3); // Extracto tres primeros símbolos
Dos = StringSubstr (símbolo (), 3, 3); // Extracto segundo período de tres símbolos
STL = STYLE_DOT; // Para todos - línea de puntos de estilo
Col = DarkOrange; // Para todos - este color
if (Instr Una == | | == Dos Instr) // Y para los eventos de nuestro ..
(// .. símbolo ..
STL = STYLE_SOLID; // .. este estilo ..
Col = Rojo; // .. y este color de la vert. línea
)
//------------------------------------------------- --------- 7 --
Obj_Name = "News_Line" + Str_DtTm; // Nombre del objeto
ObjectCreate (Obj_Name, OBJ_VLINE, 0, Dat_DtTm, 0); // Crear objeto ..
ObjectSet (Obj_Name, OBJPROP_COLOR, Col); // .. y su color, ..
ObjectSet (Obj_Name, OBJPROP_STYLE, STL); // .. y estilo ..
ObjectSetText (Obj_Name, Texto, 10); // y descripción ..
)
//------------------------------------------------- --------------- 8 --
FileClose (Asa); // Cerrar el archivo
PlaySound ( "bulk.wav"); // acompañamiento de sonido
WindowRedraw (); // rehacer objeto
return; // Salir de inicio ()
)
//------------------------------------------------- --------------- 9 --
```

Las variables utilizadas se abren y se describe en el bloque 2-3 de la EA. Un intento de abrir el archivo y el análisis de los resultados de esta operación se llevan a cabo en el bloque 3-4. El FileOpen () es usado para abrir el archivo:

```
Asa = FileOpen (file_name, FILE_CSV | FILE_READ, ";"); // se abre el archivo
```

Un intento de abrir el archivo no es siempre un éxito. Se puede fallar, si el archivo con el nombre especificado no está disponible. Cuando no se abre el archivo (el descriptor de archivo es un número negativo) el necesario mensaje de texto se muestra al usuario y la ejecución de la función start () se detiene.

En caso de éxito de la apertura de un expediente, se pasa el control al operador del "mientras" ciclo (bloques 4-8). La lectura de datos del fichero (bloque 5-6), transformación de datos y su análisis (6-7 bloques) y la creación de la gráfica objeto con las coordenadas y los parámetros correspondientes leer la última información (bloque 7-8) se realizan en cada iteración.

La ejecución del "mientras que" el ciclo continúa hasta que el puntero de archivo llega a la final del archivo, es decir, no habrá información restante a la derecha del puntero. El FileIsEnding () se utiliza para analizar la posición del puntero de archivo.

FileIsEnding ()

```
bool FileIsEnding (int asa)
```

La función devuelve TRUE si el archivo es el puntero al final del archivo, de lo contrario devuelve false. Con el fin de recibir la información sobre un error que debe usar la GetLastError (). El GetLastError () devolverá el ERR_END_OF_FILE (4099) de error, en caso de que el final del archivo se alcanza durante la lectura.

Parámetros:

manejar - descriptor de archivo que sea devuelto por el FileOpen ().

La solución representada (timetablenews.mq4) implica que cualquier número de noticias puede escribirse en el archivo News.csv. News.csv archivo contiene cinco entradas correspondientes a cinco eventos (noticias) en el mencionado ejemplo (fig. 149). En general, el número de líneas podrá ser de 0 a 20-30, dependiendo de la cantidad real de los acontecimientos que deben tener lugar el día de hoy.

La lectura de las entradas de archivo (que se identifica por el "manejar" la variable) se realiza en bloques de 5-6:

```
Str_DtTm = FileReadString (Asa); // Fecha y hora del evento (fecha)
Texto = FileReadString (Asa); // Texto de descripción de la cita
if (FileIsEnding (Asa) == true) // puntero del archivo está al final
break; // Salir de lectura y dibujo
```

La primera y segunda líneas del bloque 5-6 realizar la lectura de la información de archivo hasta la más cercana separador se cumple. La tercera y cuarta líneas de realizar la comprobación: es el puntero de archivo al final de la línea. Si no es así, entonces los objetos gráficos se formará a través de dos valores más leído en el ciclo. Si en un principio se sabe sobre el número de entradas, entonces el análisis que se realiza en la tercera y cuarta líneas, no sería necesario. En este caso, difícilmente especificar el número de iteraciones en el ciclo (por ejemplo, 5) y no realizar un control extra.

Sin embargo, el número de entradas es desconocido, en este caso. Al mismo tiempo, en este ejemplo cada caso se describe con dos valores que componen una línea del siguiente tipo: valor, separador de archivo, el valor, el final de la línea firmar. En este caso, se supone que si hay una entrada (primer valor en la línea), entonces el otro existe, pero si no hay primera entrada luego que el segundo no existe, por lo que no hay caso y no hay necesidad de crear un objeto gráfico. Si ambas entradas o uno de ellos no existe el puntero se desplazará al final del archivo (es decir, la posición en el archivo donde no se dispone de datos a la derecha del puntero existen) cuando un intento de leer que se lleva a cabo. El control realizado en el bloque 3-4 permite a descubrir este hecho. Si la observó control (dos últimas líneas en el bloque 5-6) será suprimido, entonces innecesaria objeto se creará, mientras que el programa está funcionando. Sólo después de que la condición de "mientras" se terminó el ciclo de activación y el control se pasa al bloque 8-9. En general, usted debería considerar la lógica de la representación de datos en el expediente, orden de secuencia de entradas y separadores, el número de líneas, etc, mientras que la composición de un algoritmo para la lectura de archivos. Cada cierto circunstancia exige que la persona algoritmo.

Los datos de leer el archivo tiene la cadena tipo. Con el fin de utilizar los valores recibidos para la creación de objetos gráficos que deben transformar los datos para el tipo necesario. En el bloque 6-7, la primera (leer en la siguiente línea) el valor se transforma a la "datetime" valor y además se utilizará como la de coordinar el objeto gráfico que corresponde el caso. Tres primeros símbolos de leer la segunda cadena de valor se comparan con el primer y el segundo triplete de símbolos en el símbolo de nombre. Si hay una coincidencia entonces el objeto gráfico recibe la correspondiente parámetros: el estilo de línea - y sólida de color - rojo (bloque 7-8). En otros casos, los objetos se muestran con la línea de puntos de color naranja. Usted puede observar las líneas de noticias en la ventana de símbolo como el resultado de la ejecución de scripts:

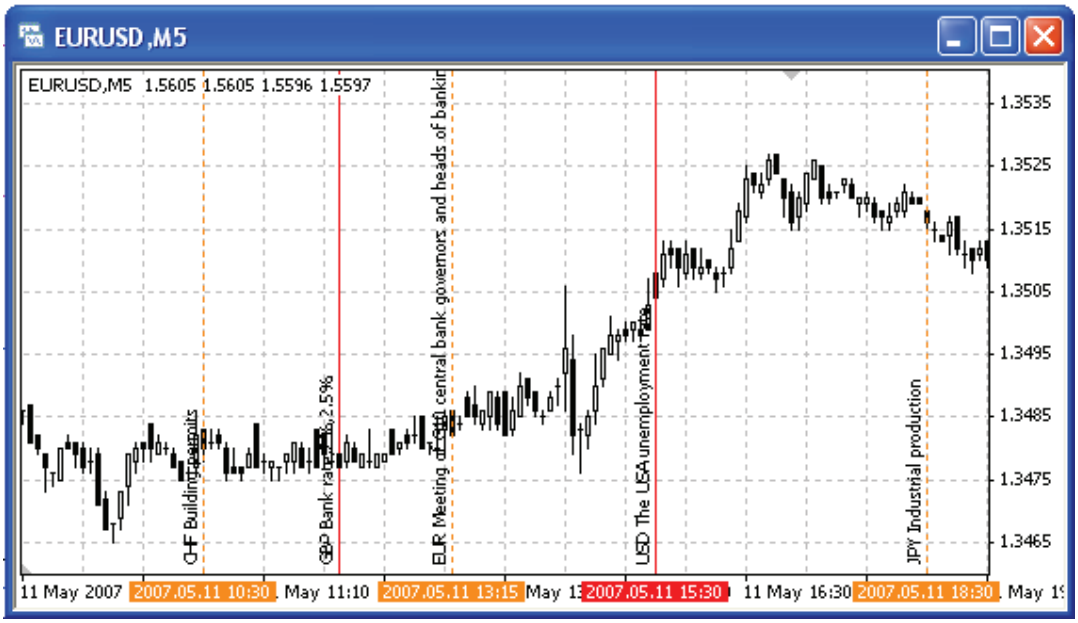


Fig. 150. Objetos gráficos en la ventana de símbolo después de timetablenews.mq4 ejecución.

De esta forma, el script puede ser ejecutado en cualquier ventana de símbolo. Al mismo tiempo, cada ventana de contener la sólida línea roja que representan los acontecimientos que afectan a este símbolo, y la línea de puntos que representan las rejillas de ventilación en relación con los otros símbolos'. Para visualizar el texto de las descripciones de los objetos que deben verificar la "Mostrar descripción del objeto" en la opción Propiedades de la ventana de un símbolo (F8) => común.

El fichero previamente abierto está cerrado en el bloque 8-9 después de que el problema se solucione, es decir, todos los objetos necesarios son creados. El archivo debe ser cerrado por las siguientes razones: en la primera parte - no escatimar recursos adicionales PC y en la segunda parte para permitir a los otros programas para acceder al archivo en el modo de escritura. Se debe considerar como normal para cerrar el expediente tan pronto como toda la información se lee de él (o por escrito en ella) y su uso ya no es necesario. El cierre de archivos se realiza por la FileClose ().

FileClose ()

```
void FileClose (int asa)
```

La función realiza el cierre de un archivo que previamente fue inaugurada por el FileOpen ().

Parámetros:

manejar - descriptor de archivo que sea devuelto por el FileOpen ().

Con el fin de permitir que el comerciante prácticamente timetablenews.mq4 utilizar el script, debe mantener el método para la creación de un archivo que contiene las noticias calendario de un periodo. Este tipo de archivo puede ser creado usando cualquier editor de texto, sin embargo, en este caso, la posibilidad de un error sigue siendo (a veces un separador no puede ser especificado por error). Vamos a examinar una variante de trabajo mediante la creación de un archivo MQL4.



Problema 37. Representan el código de la AE que crea un archivo de noticias calendario.

En general, una AE puede ser destinado para la creación de un archivo que contiene cualquier número de noticias. El examinan aquí createfile.mq4 AE crea el fichero de trabajo que contiene la información acerca de no más de cinco eventos.

```
//------------------------------------------------ ----------------------
// Createfile.mq4
// El código debería ser usado para fines educativos únicamente.
//------------------------------------------------ ---------------- 1 --
extern Date_1 cadena = ""; // 2007.05.11 10:30
extern Text_1 cadena = ""; // CHF licencias de construcción
extern Date_2 cadena = ""; // 2007.05.11 12:00
extern Text_2 cadena = ""; // GBP Refinance tasa de 2%, 2,5%
extern Date_3 cadena = ""; // 2007.05.11 13:15
extern Text_3 cadena = ""; // EUR Reunión del G-10 gobernadores de los bancos
extern Date_4 cadena = ""; // 2007.05.11 15:30
extern Text_4 cadena = ""; // USD la tasa de desempleo EE.UU.
extern Date_5 cadena = ""; // 2007.05.11 18:30
extern Text_5 cadena = ""; // JPY La producción industrial
//------------------------------------------------ ---------------- 2 --
int start () // Espec. función start ()
(
//------------------------------------------------ ---------------- 3 --
int Handle, // descriptor de archivo
Qnt_Symb; // Número de símbolos registrados
string file_name = "News.csv"; // Nombre de archivo
Erray cadena [5, 2]; // Array de 5 noticias
//------------------------------------------------ ---------------- 4 --
Erray [0, 0] = Date_1; // Llenar la matriz con valores
Erray [0, 1] = Text_1;
Erray [1, 0] = Date_2;
Erray [1, 1] = Text_2;
Erray [2, 0] = Date_3;
Erray [2, 1] = Text_3;
Erray [3, 0] = Date_4;
Erray [3, 1] = Text_4;
Erray [4, 0] = Date_5;
Erray [4, 1] = Text_5;
//------------------------------------------------ ---------------- 5 --
Asa = FileOpen (file_name, FILE_CSV | FILE_WRITE, ";"); // se abre el archivo
if (Asa ==- 1) // no se abre el archivo
(
Alert ( "Un error al abrir el archivo.", // Mensaje de error
"Puede ser el archivo está ocupado por la otra applictiom");
PlaySound ( "Bzrrr.wav"); // acompañamiento de sonido
return; // Exir start ()
)
//------------------------------------------------ ---------------- 6 --
for (int i = 0; i <= 4; i + +) // Ciclo de toda la gama
(
if (StringLen (Erray [i, 0]) == 0 | | // Si el valor de la primera o ..
StringLen (Erray [i, 1]) == 0) // .. segunda variable es no entró
break; // ..  entonces salir del ciclo
Qnt_Symb = FileWrite (Handle, Erray [i, 0], Erray [i, 1]); // escribir en el archivo
if (Qnt_Symb <0) // Si no
(
Alert ( "Error al escribir en el fichero", GetLastError ()); // Mensaje
PlaySound ( "Bzrrr.wav"); // acompañamiento de sonido
FileClose (Asa); // Archivo de clausura
return; // Salir de inicio ()
)
)
//------------------------------------------------ ---------------- 7 --
FileClose (Asa); // Archivo de clausura
Alert ( "La", file_name, "archivo creado."); // Mensaje
PlaySound ( "Bulk.wav"); // acompañamiento de sonido
return; // Salir de inicio ()
```

```
 )
 //---------------------------------------------- ---------- 8 --
```

La información inicial se introduce al programa usando la variables externas de la "cadena" tipo (bloque 1-2). Las variables se abren y se describe en el bloque 3-4. Para realizar el procesamiento conveniente de los datos se escribe en el Erray [] [] cadena matriz. Cada evento (la información que caracterizan a la noticia) está representado por dos elementos de la matriz en la segunda dimensión. El tamaño de la primera dimensión (el número de líneas en la matriz) se define con el número de noticias, en este caso, 5. Con el fin de evitar entrar en el manual de valores al intentar la AE en una demostración de cuenta que puede cargar los ajustes de la AE archivo example_news.set; el expediente de la AE establecimiento debe estar ubicado en la **Terminal_folder \ presets ** carpeta.

Bloque 5-6 realiza se abre el archivo. Si la operación falla, entonces la función start () termina de trabajo después de que el usuario ha recibido el mensaje. Si el archivo se abre con éxito el control entonces se pasa al "de" operador de ciclo en el bloque 6-7. En general, el número de valores de entrada, el tamaño de la Erray gama y el número de iteraciones se puede incrementar a la cantidad necesaria.

La comprobación se lleva a cabo cada iteración: es uno de los valores inscritos vacía. La longitud de la serie Erray valores se calcula para este objetivo. Si uno de ellos tiene la longitud cero, entonces es considerado como la ausencia de los actuales y los próximos eventos, por lo que la iteración actual interrumpe. La escritura de los valores de dos elementos de la matriz para el archivo va en lo que respecta al vacío valor del elemento se encuentra. El FileWrite () es usado para escribir los valores para el archivo csv.

FileWrite ()

```
int FileWrite (int manejar, ...)
```

La función se destina para escribir la información a un archivo csv, el separador entre la información que se incluye en forma automática. El signo que representa el final de la línea "\ r \ n", se añadirá al expediente después de que la información por escrito. La información se transforma a partir de la numéricos para el formato de texto cuando outputted (véase Imprimir ()). La función devuelve el número de símbolos escritos o el valor negativo, en caso de producirse un error.

Parámetros:

manejar - descriptor de archivo que sea devuelto por el FileOpen ();

... - Datos separados con comas. No puede ser más de 63 parámetros.

Los datos de la "doble", "int" tipos se transforma automáticamente a la cadena (los datos de "color", "datetime" y "bool" tipos se considera como el número entero de la "int" tipo y transformada al cadena, también), los datos de la "cadena" tipo de salida es como es, sin transformación. Los arrays no se puede pasar como los parámetros; arrays deberá consignarse elementwise.

En el ejemplo, consideró que la información es escrita en el fichero en la línea siguiente:

```
Qnt_Symb = FileWrite (Handle, Erray [i, 0], Erray [i, 1]); // escribir en el archivo
```

El separador (el símbolo que se utiliza como separador se especifica en el expediente de apertura función FileOpen (), en este caso, ";") será escrita después de la Erray [i, 0] valor al escribir al archivo. El signo que representa el final de la línea "\ r \ n" se coloca automáticamente al final del la FileWrite () ejecución de la función, es decir, al final del escrito. La misma entrada se escribirá en cada iteración siguiente de la "para" ciclo. Cada nueva entrada comienza desde la posición donde el archivo separador del último escrito se coloca. Al mismo tiempo, los valores de la siguiente elementos de la 'Erray' será escrita en el fichero (índices de los elementos se aumentarán en 1 en cada iteración).

Si el actual escrito al expediente es el éxito es el control pasó a la siguiente iteración. Si el escrito en el archivo falla, entonces el archivo será clausurada por el FileClose () después de que el mensaje se muestra al usuario, y la función start () termina su trabajo. Si todos los escritos al expediente se realizó con éxito el control entonces se pasa al archivo de clausura función FileClose () en el bloque 7-8 después de la ejecución de la "para" el ciclo ha terminado. En este caso, el mensaje sobre el éxito de la creación de un archivo se muestra, después de que la función start () la ejecución se haya terminado. El News.csv archivo se muestra en la fig. 149 se creó después de la ejecución AE ha terminado.

Función para realizar operaciones de archivos

Función	Resumen de Información
FileClose	El cierre del expediente que fue abierto anteriormente por el FileOpen ().
FileDelete	Eliminar del archivo. Los archivos sólo pueden eliminarse si se encuentran en la *terminal_folder* \ *expertos* \ *archivos (terminal_folder* \ *verificador* \ *archivos,* en caso de las pruebas de EA) en la carpeta o sus subcarpetas.
FileFlush	Flushing toda la información que se deja en el archivo de entrada-salida de bufer en el disco duro.
FileIsEnding	Devuelve TRUE si el archivo es el puntero al final del archivo, de lo contrario - devuelve FALSE. Si al final del archivo se alcanza durante la lectura de archivos, los GetLastError () devolverá el ERR_END_OF_FILE (4099) error.
FileIsLineEnding	Devuelve TRUE, si el archivo es el puntero al final de la línea del archivo CSV. De lo contrario, devuelve FALSE.
FileOpen	Abre un archivo para el ingreso y / o la salida. La función devuelve el descriptor de archivo del expediente abierto de -1, en caso de que falle.
FileOpenHistory	Abre un archivo en la carpeta actual de la historia *(termial_folder* \ *historia* \ *server_name)* o en sus subcarpetas. La función devuelve el descriptor de archivo o -1, en caso de que falle.
FileReadArray	La función lee un número especificado de elementos desde el archivo binario a la matriz. El conjunto debe tener suficiente tamaño antes de la lectura. La función devuelve el número de elementos prácticamente leer.
FileReadDouble	La función lee el número de doble precisión con la de punto flotante (doble) de la posición actual del archivo binario. El tamaño de la cifra puede el siguiente: 8 bytes (doble) y 4 bytes (float).
FileReadInteger	La función lee el número entero de la posición actual del archivo binario. The size of the number may be the following: 1, 2 or 4 bytes. If the size of the number is not specified then the system will try to read it as it was the 4 byte integer number.
FileReadNumber	Reading the number from the current position of the CSV-file until the separator is met. It can be applied only to csv-files.
FileReadString	The function reads the line from the current position of the file. It can be applied both for csv and binary files. The line in the text file will be read until the separator is met. The specified number of symbols in the line will be read in the binary files.
FileSeek	The function moves the separator to the new position that is the displacement from the beginning, end or the current position of the file in bytes. The next reading or writing starts from the new position. If the pointer moving is performed successfully then the function will return TRUE, otherwise - FALSE.
FileSize	The function returns the size of the file in bytes.
FileTell	The function returns the shift of file pointer from the beginning of the file.
FileWrite	The function is intended to write the information to the csv-file, the separator is placed automatically between the information. The end of the line sign "\r\n" is added to the file after the writing is finished. The numeric data is transformed to the text format during the ouptputting process. The function returns the the number of written symbols or a negative value if an error occurs.
FileWriteArray	The function writes the array to the binary file.
FileWriteDouble	The function writes the number with the floating point to the binary file.
FileWriteInteger	The function writes the integer number value in the binary file.
FileWriteString	The function writes the line to the binary file from the current position. It returns the number of practically written bytes or a negative value, in case an error occurs.

Las matrices y Timeseries

Es muy importante tener en cuenta que la secuencia de un solo tipo de elementos es siempre numerados a partir de cero en MQL4.

Se mencionó antes de que usted no debe confundir el valor de la matriz elemento índice con el número de elementos en la matriz (véase Arrays). Por ejemplo, si la matriz se declara:

```
Erray_OHL int [3] // Array declaración
```

entonces esto significa que una variedad dimensional llamado Erray_OHL contiene tres elementos. Indexación de los elementos que comienzan con cero, es decir, el primero de los tres elementos tiene el índice 0 (Erray_OHL [0]), el segundo - el índice 1 (Erray_OHL [1]), y el tercero - el índice 2 (Erray_OHL [2]). De esta forma, el máximo valor del índice es inferior al número de elementos en la serie por uno. En este caso, la matriz es una dimensión, es decir, que podemos decir sobre la cantidad de elementos en la primera dimensión: el máximo número de índice es 2, porque el número de los elementos de la matriz es de 3.

Lo mismo puede decirse acerca de la numeración de las dimensiones en la matriz. Por ejemplo, si un array se declara la siguiente manera:

```
Erray_OHL int [3] [8]; // Array declaración
```

esto significa que la matriz tiene dos dimensiones. La primera dimensión indica el número de filas (3 en este ejemplo), y el segundo especifica el número de elementos en la fila (o el número de columnas 8, en este ejemplo). La dimensión numerated en sí es demasiado. La primera dimensión tiene el número 0, y la segunda - el número 1. El número de dimensiones se utilizan en la ArrayRange (), por ejemplo.

ArrayRange ()

```
int ArrayRange (objeto array [], int range_index)
```

La función devuelve el número de elementos especificados en la dimensión de la matriz.

El uso de ArrayRange () puede ser demostrado con la solución del siguiente problema:



Problema 38. El Mas_1 array contiene los valores de la matriz de 3x5. Obtener los valores de la Mas_2 gama que contiene los elementos cuyos valores son iguales a los valores de la matriz de transposición. El uso arbitrario de valores de los elementos.

Vamos a trabajar algunos valores de los elementos y representar a la inicial y la deseada matrices que los Mas_1 y Mas_2 arrays contienen, respectivamente:

Índices	0	1	2	3	4
0	1	2	3	4	5
1	11	12	13	14	15
2	21	22	23	24	25

Matriz inicial, Mas_1 matriz.

Índices	0	1	2
0	1	11	21
1	2	12	22
2	3	13	23
3	4	14	24
4	5	15	25

Transponer la matriz, Mas_2 matriz.

Fig. 151. Inicial de transposición y Matrices.

En este caso, el problema se resuelva por sí solo a la reescritura de los valores de la primera matriz a la segunda de acuerdo con las normas de transposición de la matriz, es decir, reescribir los elementos valores de la primera matriz de columnas a las filas de la matriz deseada. La solución de matriz de transposición problema está representado en la matrix.mq4 de expertos:

```
//------------------------------------------------- --------------------
// Matrix.mq4
// El código debería ser usado para fines educativos únicamente.
//------------------------------------------------- --------------- 1 --
int start () // Función especial start ()
(
Mas_1 int [3] [5] = (1, 2, 3, 4, 5, 11, 12, 13, 14, 15, 21, 22, 23, 24, 25);
Mas_2 int [5] [3];
int R0 = ArrayRange (Mas_1, 0); // Número de elementos en primera dim.
int R1 = ArrayRange (Mas_1, 1); // Número de elementos en segunda dim.

for (int i = 0; i
```

Dos matrices se han abierto en la función start () de los expertos. El Mas_1 serie tiene 3 filas que contienen 5 elementos cada una y el MAS_2 serie cuenta con 5 filas que contienen 3 elementos cada una. La reescritura de los valores en sí se realiza en la siguiente entrada:

```
Mas_2 [[j] [i] = Mas_1 [i] [j] // Matriz de transposición
```

Con el fin de calcular el entorno de tiempo de ejecución (el número de iteraciones) de dos operadores incorporados ciclo, que debe conocer los valores de los elementos de cada matriz. En este ejemplo, los valores constantes 3 y 5 podrían utilizarse. Sin embargo, esta forma de diseñar el programa es incorrecto. En general, un programa puede contener un gran código en el que las llamadas a los mismos valores se lleva a cabo en muchas partes de ella. Un programa debe ser diseñado de forma que las modificaciones podrían hacerse en un solo lugar, si es necesario, y en todas las demás piezas necesarias que se calcularía. En este caso, sólo las entradas que se abren y inicializar los arrays deben modificarse si es necesario para cambiar el tamaño de los arrays, por lo que no hay necesidad de modificar el código a las otras partes.

Para determinar el número de elementos de la primera y segunda dimensiones de la gama Mas_1 los siguientes cálculos se realizan:

```
int R0 = ArrayRange (Mas_1, 0); // Número de elementos en primera dim.
int R1 = ArrayRange (Mas_1, 1); // Número de elementos en segunda dim.
```

Tenga en cuenta que el valor 0 se utiliza para la primera dimensión y el valor 1 se utiliza para la segunda. Los valores calculados de la R0 y R1 variables se utilizan para determinar el número de iteraciones en el "por" ciclos.

Los valores recibidos de la matriz Mas_2 elementos se muestran en la pantalla utilizando el comentario ().



Fig. 152.Result de matrix.mq4 operación.

Funciones para trabajar con Arrays

Funciones	Descripción corta
ArrayBsearch	Devuelve el índice del primer elemento encontrado en la primera dimensión de la matriz. Si el elemento con el valor especificado entre otras cosas, ausente entonces la función devolverá el índice de la más cercana (en valor) elemento.
ArrayCopy	Se copia a una serie a otra. Las matrices deben tener el mismo tipo. Los arrays de la doble [], int [], datetime [], color [], y, bool [], se puede copiar como los arrays del mismo tipo. Devuelve el número de elementos copiados.
ArrayCopyRates	Se copia a la barra de datos a la matriz bidimensional de la RateInfo [] [6] clase y devuelve el número de barras de copiado. De lo contrario, devuelve -1, si la operación fracasa.
ArrayCopySeries	Se copia a una serie timeseries a los definidos por el usuario array y devuelve el número de elementos copiados.
ArrayDimension	Devuelve un rango de multy-dimensional array.
ArrayGetAsSeries	Devuelve TRUE si la serie se organiza como un timeseries (elementos de la matriz se indexan a partir del último elemento a la primera), de lo contrario devuelve false.
ArrayInitialize	En él se establece un valor único para todos los elementos de la matriz. Devuelve el número de elementos inicializado.
ArrayIsSeries	Devuelve TRUE en caso de comprobarse la gama es un timeseries (Time [], Abierto [], cerca [], Alto [], Baja [] de volumen []), de lo contrario devuelve false.
ArrayMaximum	Busca un elemnet con el valor máximo. La función devuelve la ubicación del máximo elemento de la matriz.
ArrayMinimum	Busca un elemento con el valor mínimo. La función devuelve la ubicación del mínimo elemento en la matriz.
ArrayRange	Nos devuelve el número de elementos especificados en la dimensión de la matriz. El tamaño de la dimensión es mayor que el mayor índice de 1, porque los índices son a partir de cero.
ArrayResize	Establece un nuevo tamaño de la primera dimensión de la matriz. Devuelve el número de todos los elementos que contiene array después de su rango se ha cambiado si la función corrió con éxito, de lo contrario devuelve -1 y el tamaño del arreglo no se cambia.
ArraySetAsSeries	Establece la dirección de la indexación en la matriz.
ArraySize	Nos devuelve el número de elementos en un array.
ArraySort	Por tipo numérico arrays de su primera dimensión. El timeseries arrays no pueden ser ordenados.

Funciones para el acceso Timeseries

Funciones	Resumen de Información
iBars	Nos devuelve el número de bares de la gráfica especificada.
iBarShift	Busca un bar de tiempo. La función devuelve la barra de desplazamiento que tiene el tiempo especificado. Si la barra durante el tiempo está ausente ("agujero" en la historia), entonces la función devuelve -1 en función del parámetro *exacto* o el hecho de que el bar más cercano.
iClose	La función devuelve el precio de cierre de la barra se especifica con el *cambio* de parámetros de la gráfica correspondiente *(símbolo, calendario).* Se devuelve 0, si se produce un error.

iHigh	Nos devuelve el precio máximo valor de la barra se especifica con el *cambio* de parámetros de la gráfica correspondiente *(símbolo, calendario).* Se devuelve 0, si se produce un error.
iHighest	Devuelve el índice del valor máximo encontrado (cambio relativamente a la barra actual).
iLow	Nos devuelve el precio mínimo valor de la barra se especifica con el *cambio* de parámetros de la gráfica correspondiente *(símbolo, calendario).* Se devuelve 0, si se produce un error.
iLowest	Devuelve el índice de encontrar el mínimo valor (cambio relativamente a la barra actual).
iOpen	Devuelve el valor de abrir los precios de la barra se especifica con el *cambio* de parámetros de la gráfica correspondiente *(símbolo, calendario).* Se devuelve 0, si se produce un error.
iTime	Devuelve tiempo de apertura de la barra se especifica con el *cambio* de parámetros de la gráfica correspondiente *(símbolo, calendario).* Se devuelve 0, si se produce un error.
iVolume	Devuelve el valor de ticks volumen de la barra se especifica con el *cambio* de parámetros de la gráfica correspondiente *(símbolo, calendario).* Se devuelve 0, si se produce un error.

Para obtener la información detallada sobre éstas y otras funciones, por favor refiérase a la documentación a MQL4.community, a MetaQuotes Software Corp sitio web o en la "Ayuda" de la sección MetaEditor.

Funciones matemáticas

Matemáticas y las funciones trigonométricas se incluyen en MQL4. No hay dificultades en el uso de la mayoría de ellos. Por ejemplo, la MathMax () devuelve el valor máximo de dos valores especificados en la lista de parámetros de la función. El uso de otras funciones determinadas reclamaciones de atención y reflexión. Vamos a examinar una de las funciones de este tipo.

MathFloor ()

```
MathFloor doble (doble x)
```

La función devuelve un valor numérico que corresponde el mayor número entero que es menor o igual a x.

Parámetros:

x - valor numérico.

Tenga en cuenta que un valor devuelto por la función es el número real (double), al mismo tiempo, está escrito que la función devuelve un entero. Hay que comprender que la función devuelve un número real que tiene todas las posiciones son iguales a cero, después del punto decimal. Por ejemplo, la MathFloor () puede devolver 37,0 (número positivo de la doble tipo) o de -4,0 (número negativo del tipo doble).

La descripción dice también que la función devuelve el máximo número posible de que sea inferior a un determinado. Por ejemplo, si el valor de x dado el parámetro es de 13,5 a continuación, el número máximo que ha ceros después del punto decimal es 13,0. O bien, si el valor negativo -13,5 se especifica en la función, entonces el máximo número entero más pequeño es igual a -14,0. De tal manera, la modificación del signo de la aprobada para el valor de la función conduce a los resultados diferentes, a saber, los valores recibidos no son iguales entre sí en magnitud absoluta.

El uso de tales funciones es muy conveniente, en algunos casos. Vamos a examinar el fragmento de los lotes para el cálculo de cantidad de nuevos órdenes como un ejemplo:

```
Porcentaje int = 30; //% de margen libre
doble Libre AccountFreeMargin = (); // Libre margen
doble One_Lot = MarketInfo (Symb, MODE_MARGINREQUIRED); // 1 lote de precios
Paso doble = MarketInfo (Symb, MODE_LOTSTEP); // Tamaño del paso cambiado

doble Lots_New = MathFloor (Libre * Porcentaje One_Lot Paso 100) * Paso;
```

El valor del parámetro se Porcentaje fijado por el usuario. En este caso, el usuario especifica el 30% de margen libre para nuevos órdenes. De acuerdo con las normas que se especifican por el dealing center, calculado correctamente la cantidad de lotes debe ser divisible por el paso mínimo de los lotes cambio de tamaño (Paso). Los valores de margen libre (gratuito) y 1 lote de precios (One_Lot) son necesarios para el cálculo.

Vamos a examinar la lógica del razonamiento de que el programador compilado la expresión para calcular la cantidad requerida de los lotes Lots_New para nuevos órdenes. Vamos a utilizar los valores numéricos de las variables para una mejor visualización. Que Libre = 5000,0, One_Lot = 1360.0 (En la mayoría de los centros que tratan el costo de 1 lote de par de divisas es en proporción al coste del símbolo), Paso = 0,1. En este caso, la expresión para calcular Lots_New puede escribirse como sigue:

Lots_New = MathFloor (5000,0 * 30/100/1360.0/0.1) * 0,1;

El 5000,0 * 30/100 expresión es el valor del dinero que el usuario establece la apertura de un nuevo orden. En este caso, el precio de un nuevo orden puede llegar a los 1.500,0. El gasto de todos estos fondos se puede abrir un nuevo orden que tiene el 1500,0 / 1360,0 = 1,102941 cantidad de lotes. Sin embargo, ocupan el centro no aceptará el pedido con esta cantidad de lotes, ya que el mínimo Paso = 0,1 (en la mayoría de centros se ocupan). Para calcular la cantidad deseada de los lotes que usted debe deshacerse de todas las "innecesarias" dígitos en la parte decimal y reemplazarlos con ceros a la izquierda.

Con el fin de hacerlo puede utilizar la función matemática considera:

Lots_New = MathFloor (1.102941/0.1) * 0,1;

El valor calculado de MathFloor (1.102941/0.1) será 11,0, y el valor calculado de la Lots_New variable será 1,1 lotes. Este valor cumple con los requisitos de que trata el centro y así se puede utilizar como declaró la cantidad de lotes que se presentan en los nuevos órdenes.

Funciones matemáticas

Función	Resumen de Información
MathAbs	La función devuelve el valor absoluto (en magnitud absoluta) de un número dado.
MathArccos	La función devuelve el arccosine valor de *x* en el 0 a π radianes gama. Si *x* es menor que -1 o superior a 1, la función devuelve NaN (valor indefinido).
MathArcsin	La función devuelve arcsine valor de *x* en el - π / 2 a π / 2 radianes gama. Si *x* i inferior a -1 o superior a 1, la función devuelve NaN (valor indefinido).
MathArctan	La función devuelve arctangent de *x.* Si *x* es igual a 0, entonces la función devuelve 0. MathArctan devuelve el valor en el - π / 2 a π / 2 radianes gama.
MathCeil	La función devuelve el valor numérico que es el menor número entero mayor o igual a *x.*
MathCos	La función devuelve el coseno del ángulo.
MathExp	La función devuelve el valor **de** las transacciones electrónicas a la facultad de «*d*». En desbordamiento, la función devuelve INF (infinito), y devuelve a 0 abajo.
MathFloor	La función devuelve el valor numérico que representa el entero más grande que sea inferior o igual a *x.*
MathLog	La función devuelve el logaritmo natural de *x* en caso de éxito. Si *x* es negativo, esta función devuelve NaN (valor indefinido). Si *x* es 0, devuelve INF (infinito).
MathMax	La función devuelve el valor máximo de dos valores numéricos.
MathMin	La función devuelve el valor mínimo de dos valores numéricos.
MathMod	La función devuelve el punto flotante resto de la división de dos números. El MathMod calcula la función de punto flotante *f* resto de *x / y* de tal manera que *i = x * y + f,* donde *i* es un entero, *f* tiene el mismo signo como *x,* y el valor absoluto de *f* es menor que el valor absoluto de *y.*
MathPow	Devuelve el valor de la base de expresión elevado a la potencia especificada (exponente de valor).
MathRand	La función devuelve una pseudoaleatorias entero dentro del rango de 0 a 32767. El MathSrand función debe ser utilizada para la semilla pseudoaleatorias generador de números antes de llamar a MathRand.
MathRound	La función devuelve el valor redondeado al entero más cercano del valor numérico especificado.
MathSin	La función devuelve el seno del ángulo especificado.
MathSqrt	La función devuelve la raíz cuadrada de *x.* Si *x* es negativo, devuelve una MathSqrt indefinida (igual que una tranquila NaN).
MathSrand	La función establece el punto de partida para generar una serie de pseudoaleatorias enteros.
MathTan	MathTan devuelve la tangente de *x.* Si *x* es mayor o igual a 263, o igual o inferior a -263, una pérdida de importancia en el resultado se produzca. En este caso, la función devuelve un valor indefinido.

Para obtener la información detallada sobre éstas y otras funciones, por favor refiérase a la documentación a MQL4.community, a MetaQuotes Software Corp sitio web o en la "Ayuda" de la sección MetaEditor.

Funciones GlobalVariable

Muchas de las funciones para trabajar con variables globales del Terminal de Usuario se describen en el GlobalVariables sección. En la sección anterior, también se menciona que un programa bien diseñado tiene que borrar sus variables globales cuando se haya creado. GVS no debe permanecer en el Terminal de Usuario después de todos los programas se han salido.

Uno o más GVS podrán permanecer en la terminal cuando la depuración de programas utilizando las variables globales de la Terminal de Usuario. En este caso, un programador debe eliminar manualmente GVS antes del próximo inicio de un programa depuradas. Para automatizar este proceso, puede crear un script que borra todas las variables globales de la Terminal de Usuario.

GlobalVariablesDeleteAll ()

```
int GlobalVariablesDeleteAll (prefix_name cadena = NULL)
```

Elimina las variables globales. Si un prefijo no se especifica el nombre, entonces todas las variables globales serán borradas. De lo contrario, sólo se eliminarán las variables de los nombres que comienzan con el prefijo especificado. La función devuelve el número de variables eliminado.

Parámetros:

prefix_name - prefijo de los nombres de variables globales, debe suprimirse.

A continuación se muestra un ejemplo de un simple script, deleteall.mq4, que elimina todas las variables globales de la Terminal de Usuario.

```
//------------------------------------------------- --------------------
// Deleteall.mq4
// El programa está destinado a ser usado como un ejemplo en MQL4 Tutorial.
//------------------------------------------------- --------------------
int start () // Especial de la función start ()
(
GlobalVariablesDeleteAll (); // Borrado de todos los GVS
PlaySound ( "W2.wav"); // Sound
return; // Salir
)
//------------------------------------------------- --------------------
```

El script se puede iniciar sólo si no utilizar GVS programa se está ejecutando en el Terminal de Usuario. De lo contrario, corriendo script puede romper la lógica de otros programas ejecutados que puede dar lugar a acciones incontroladas. Después de la ejecución de scripts la ventana de variables globales (F3) de la Terminal de Usuario se convertirá en vacío:

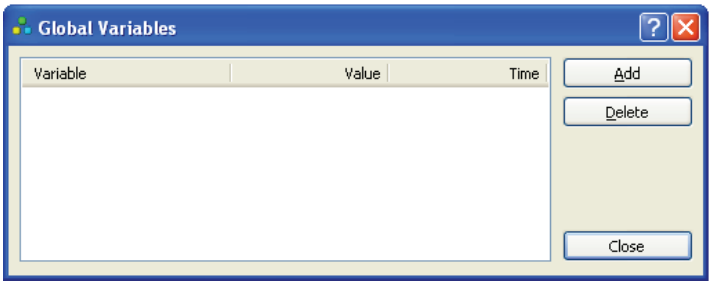


Fig. 153. Aspecto de la ventana de Variables globales en el Terminal de Usuario después de la deleteall.mq4 script se ejecuta.

Funciones para trabajar con variables globales

Función	Resumen de Información
GlobalVariableCheck	Se devuelve True si un GV está disponible. De lo contrario, devuelve FALSE.
GlobalVariableDel	Elimina una variable global. Devuelve TRUE si una variable se ha eliminado. De lo contrario, devuelve FALSE.
GlobalVariableGet	Devuelve un valor de una variable global, o 0, si se produce un error.
GlobalVariableName	La función devuelve el nombre de una variable global en función de su número de índice en la lista de variables globales.
GlobalVariableSet	En él se establece un nuevo valor a una variable global. El sistema creará una nueva variable si no hay ninguna ya creada. El tiempo del último acceso será devuelto, si la función ha ejecutado con éxito. De lo contrario, devuelve 0.
GlobalVariableSetOnCondition	En él se establece un nuevo valor a una variable global si su valor actual es igual al valor de la *check_value* tercer parámetro. La función generará la ERR_GLOBAL_VARIABLE_NOT_FOUND (4058) de error y devolver false si una variable no existe. Devuelve TRUE si la función tiene éxito. De lo contrario, devuelve FALSE.
GlobalVariablesDeleteAll	Elimina las variables globales. Si el prefijo para el nombre no se especifica, entonces todas las variables globales serán borradas. De lo contrario, elimina sólo las de los nombres comenzando con el prefijo especificado. La función devuelve el número de variables eliminado.
GlobalVariablesTotal	La función devuelve el número total de variables globales.

Para obtener la información detallada sobre éstas y otras funciones, por favor refiérase a la documentación a MQL4.community, a MetaQuotes Software Corp sitio web, o en la "Ayuda" de la sección MetaEditor.

Indicadores Personales

Las funciones de usuario los indicadores le permiten ajustar los ajustes necesarios para hacer una representación de un indicador. Vamos a considerar algunos de ellos (véase también la creación personalizada de los indicadores).

SetIndexBuffer ()

```
bool SetIndexBuffer (int índice, doble array [])
```

La función se suma a una serie de variables que se declaró en el plano mundial con un indicador predestinado usuario de amortiguación. La cantidad de topes que son necesarias para el indicador de cálculo se establece mediante la IndicatorBuffers () y no puede ser superior a 8. En caso de éxito de la conexión, se devuelve TRUE, de lo contrario - FALSO. Con el fin de obtener la información detallada sobre un error que usted debe llamar al GetLastError ().

Parámetros:

Índice - número de índice de una línea (de 0 a 7 son los valores posibles);

array [] - un nombre de la matriz que está conectado con un cálculo de amortiguación.

SetIndexStyle ()

```
void SetIndexStyle (int índice, int tipo, int style = vacía, int width = vacía, color = CLR_NONE CLR)
```

La función establece un nuevo tipo, estilo, color y anchura de una determinada línea del indicador.

Parámetros:

Índice - número de índice de una línea (de 0 a 7 son los valores posibles);

tipo - indicador de tipo de línea. Puede ser uno de los tipos de indicador de líneas (véase el Indicador Estilos de Líneas Viendo);

estilo - estilo de línea. Se utiliza para las líneas de 1 pixel de ancho. Puede ser uno de los estilos de línea especificado en el Indicador de Estilos de Líneas Viendo del apéndice. El valor EMPTY especifica que el estilo no será cambiado;

ancho - ancho de línea; valores admisibles son - 1,2,3,4,5; el valor EMPTY especifica que la anchura no será cambiado;

CLR - línea de color. El valor CLR_NONE vacía significa que el color no será cambiado.

SetIndexLabel ()

```
void SetIndexLabel (int índice, cadena de texto)
```

La función permite establecer un indicador de línea de nombres y apellidos que se muestran en DataWindow y en el pop-up punta.

Parámetros:

Índice - número de índice de una línea (de 0 a 7 son los valores posibles);

texto - un texto que describe un indicador de línea. NULL significa que el valor de una línea no se muestra en DataWindow.

El ejemplo de la simple indicador muestra la línea de alta (indicatorstyle.mq4) que utiliza las funciones descritas más arriba:

```
//------------------------------------------------- --------------------
// Indicatorstyle.mq4
// El código debería ser usado para fines educativos únicamente.
//------------------------------------------------- --------------- 1 --
# propiedad indicator_chart_window // Índico.  Se señala en la ventana principal
# propiedad indicator_buffers 1 // Cantidad de buffers
# propiedad indicator_color1 Azul // Color de la primera línea

Buf_0 doble [] // Indicadores variedad apertura
//------------------------------------------------- --------------- 2 --
int init () // Función especial init ()
(
SetIndexBuffer (0, Buf_0); // Asignar la matriz para el buffer
SetIndexStyle (0, DRAW_LINE, STYLE_SOLID, 2); // Estilo de línea
SetIndexLabel (0, "High Line");
return; // Salir de especificaciones. función init ()
)
//------------------------------------------------- --------------- 3 --
int start () // Función especial start ()
(
int i, // Bar Índice
Counted_bars; // Monto calculado de barras
Counted_bars = IndicatorCounted (); // Monto calculado de barras
i = Bares - Counted_bars - 1; // Índice de la primera incontables
while (i> = 0) // Ciclo para los incontables bares
(
Buf_0 [i] = alto [i]; // valor 0 del buffer on a bar
i -; // Cálculo del índice para el próximo bar
)
return; // Salir de especificaciones. función start ()
)
//------------------------------------------------- --------------- 4 --
```

Bloque 1-2 contiene la configuración general de un indicador. Se especifica con el comando # propiedad que el indicador debe ser dibujadas en la ventana principal, el indicador utiliza un buffer, el color del indicador de línea es de color azul. Una gama de amortiguación se abre en el bloque 1-2, también.

Las funciones se establece más arriba se utilizan en el bloque 2-3. La entrada:

```
SetIndexBuffer (0, Buf_0); // Asignar la matriz para el buffer
```

asigna el nombre de amortiguación Buf_0 para el buffer con el índice 0. La entrada:

```
SetIndexStyle (0, DRAW_LINE, STYLE_SOLID, 2); // Estilo de línea
```

determina el estilo de indicador de línea que tiene el índice 0. El DRAW_LINE constante indica que la línea que aparece es simple, la STYLE_SOLID constante indica que la línea es sólida, y 2 se especifica el ancho de la línea. La entrada:

```
SetIndexLabel (0, "High Line");
```

asigna el nombre al indicador de acuerdo con el índice 0. El nombre especificado puede verse en las DataWindow y en el globo punta en la ventana de instrumento financiero (fig. 810_3). La denominación de las líneas es conveniente, si la ventana contiene una serie de líneas indicador; a veces es la única manera de distinguir una línea de otra. Bloque 3-4 realiza el cálculo de los valores del indicador gama de elementos para la línea que se utiliza para mostrar los valores máximos de bares (la secuencia del indicador arrays cálculo se describe en la creación personalizada de la sección Indicadores en detalle).

Si el indicador se muestra en una ventana independiente, entonces el nivel horizontal se pueden visualizar en esta ventana.

SetLevelValue ()

```
void SetLevelValue (int nivel, doble valor)
```

Establece el valor especificado para el nivel horizontal del indicador que se muestra en una ventana aparte.

Parámetros:

nivel - nivel número (0-31).

valor - un valor para el nivel especificado.

El uso de los niveles horizontal puede ser muy conveniente, si es necesario para detectar visualmente si el indicador está por encima de la línea o por debajo de los valores especificados. El simple indicador que calcula la diferencia entre el máximo y el precio mínimo de la barra se muestra a continuación. El mercado de eventos son interesantes para el usuario (en principio en este ejemplo) si el indicador de línea está por encima de la barra de 0,001 o por debajo de la barra de -0,001. El ejemplo del indicador que muestra la diferencia entre el Alto y Bajo (linelevel.mq4):

```
//---------------------------------------------- ----------
// Linelevel.mq4
// El código debería ser usado para fines educativos únicamente.
//---------------------------------------------- ---------- 1 --
# propiedad indicator_separate_window // Índico. Se señala en un sep. ventana
# propiedad indicator_buffers 1 // Cantidad de buffers
# propiedad indicator_color1 Rojo // Línea color

Buf_0 doble [] // Indicadores variedad apertura
//---------------------------------------------- ---------- 2 --
int init () // init Especial ()
(
SetIndexBuffer (0, Buf_0); // Asignar la matriz para el buffer
SetIndexStyle (0, DRAW_LINE, STYLE_SOLID, 2); // Estilo de línea
SetIndexLabel (0, "High / Low Diferencia");
SetLevelValue (0, 0.0010); // La línea horizontal es el nivel establecido
SetLevelValue (1, - 0,0010); // Otro nivel es
return; // Salir de spec.init ()
)
//---------------------------------------------- ---------- 3 --
int start () // Especial de la función start ()
(
int i, // Bar Índice
Counted_bars; // Monto calculado de barras

Counted_bars = IndicatorCounted (); // Monto calculado de barras
i = Bares - Counted_bars - 1; // Índice de la primera incontables

while (i> = 0) // Ciclo para los incontables bares
(
Buf_0 [i] = alto [i] - Baja [i]; // 0 valor de la amortiguación en la barra i
if (Open [i]> Cerrar [i]) // si la vela es de color negro ..
Buf_0 [i] =- Buf_0 [i]; // .. entonces el valor inverso
i -; // Cálculo del índice para el próximo bar
)
return; // Salir de spec.start ()
)
//---------------------------------------------- ---------- 4 --
```

El considerado función se usa en el bloque 2-3 en el indicador. En este caso, dos niveles horizontales se especifican. El primer valor en la lista de parámetros es el número del plano horizontal, el segundo es el valor especificado del nivel:

```
SetLevelValue (0, 0.0010); // El nivel de la horizontal. línea se fija
SetLevelValue (1, - 0,0010); // Otro nivel es
```

Los parámetros de indicatorstyle.mq4 y linelevel.mq4 indicadores establecidos de tal manera se muestran en la ventana del instrumento financiero y en las DataWindow.

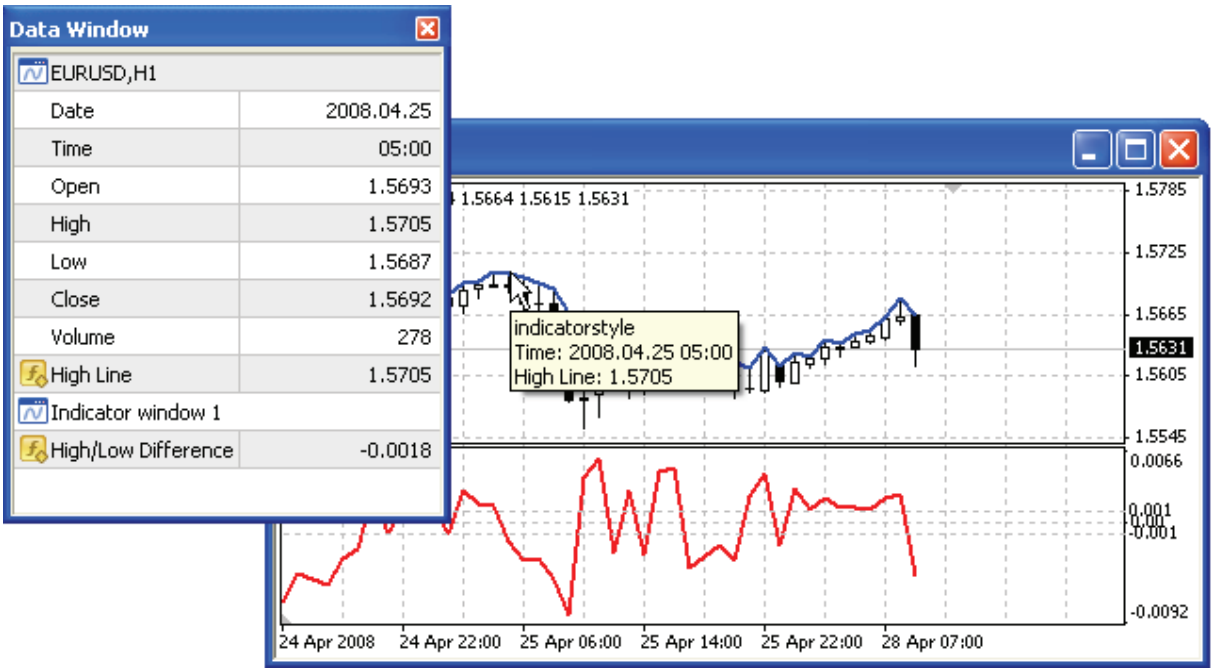


Fig. 154. Viendo la configuración de indicadores en las diferentes ventanas de la Terminal de Usuario.

Dos ventanas - DataWindow y el instrumento financiero de la ventana se muestran en la fig. 154. Se puede ver la entrada que contiene "Línea de alta" el texto y el valor 1.3641 en el DataWindow. Las mismas inscripciones se muestran en la parte inferior de la entrada pop-up punta. Dicha entrada se muestra en la DataWindow todo el tiempo que el indicador se ejecuta, el nombre de la línea no se cambia a eso, pero el valor en la parte derecha de la entrada depende de la posición del cursor en la ventana de instrumento financiero. El nombre de la línea que se muestra en el pop-up corresponde a punta el indicador de línea el cursor se señaló.

La subventana de la linelevel.mq4 contiene el indicador de líneas horizontales que se colocan de acuerdo con el usuario-valores establecidos. Si mueves el cursor sobre la línea roja, indicador de entonces el nombre de esta línea, en este caso la "Diferencia entre el Alto y Bajo", se puede ver en el pop-up punta, el valor en el punto del cursor se puede ver a la derecha del nombre.

Funciones para el ajuste personalizado indicadores

Función	Resumen de Información
IndicatorBuffers	Se organiza la buffers de memoria que se utilizan para calcular un indicador personal. La cantidad de buffers no puede ser superior a 8 y deberá ser inferior al valor especificado en la *propiedad # indicator_buffers* comando. Si la usuario indicador necesidades más topes para el cálculo, usted debe utilizar esta función para especificar el número entero de buffers.
IndicatorCounted	La función devuelve una cantidad de bares que no han sido modificados desde el último indicador de lanzamiento. La mayoría de los bares no requieren cálculos.
IndicatorDigits	En él se establece la precisión (el número de símbolos después del punto decimal) para la visualización de los valores de los indicadores.
IndicatorShortName	En él se establece un "corto" nombre a un indicador que se mostrará en el indicador de la subventana y en el DataWindow.
SetIndexArrow	En él se establece un símbolo a un indicador de línea que tiene el DRAW_ARROW estilo.
SetIndexBuffer	Se suma a una serie de variables que se declara en el plano mundial dentro de un buffer de un indicador personal.
SetIndexDrawBegin	En él se establece un número de índice desde el inicio de datos a la barra de un

SetIndexEmptyValue	dibujo de un determinado indicador debería empezar. En él se establece un valor vacío para el indicador. Vaciar los valores no se han tomado y no se muestran en las DataWindow.
SetIndexLabel	En él se establece el nombre de un indicador de línea para visualizar la información en las DataWindow y en la punta del globo.
SetIndexShift	En él se establece un cambio de un indicador relativamente línea a la gráfica comienzo. El valor positivo se pasará una línea a la derecha, el valor negativo - a la izquierda. Es decir, el valor calculado sobre la barra actual se dibuja con el objetivo específico, relativo a la barra actual, por turnos.
SetIndexStyle	En él se establece un nuevo tipo, estilo, color y anchura de un determinado indicador de línea (véase el Indicador Estilos de Líneas Viendo).
SetLevelStyle	En él se establece un nuevo estilo, color y ancho horizontal de los niveles de un indicador que se muestra en una ventana independiente (véase el Indicador Estilos de Líneas Viendo).
SetLevelValue	En él se establece un valor para el nivel especificado horizontal de un indicador que se muestra en una ventana aparte.

Para obtener la información detallada sobre éstas y otras funciones, por favor refiérase a la documentación a MQL4.community, a MetaQuotes Software Corp sitio web o en la "Ayuda" de la sección MetaEditor

Datos de la cuenta

Las funciones del cliente y el estado terminal de control de funciones son convenientes que deben aplicarse para el programa de restricción cuando se distribuye en la base comercial, por ejemplo. Vamos a examinar la solución del problema a continuación:



Problema 39. Crear un código de protección del programa distribuido por la base comercial, que se reúne los siguientes requisitos:

- el programa debe exigir una contraseña para ser ejecutado en las cuentas reales de cada uno de los clientes;
- la contraseña no está obligado a ejecutar el programa real en las cuentas de clientes corporativos;
- no son las limitaciones previstas para la ejecución del programa de demostración en las cuentas.

Este ejemplo contiene una correcta definición del problema. Para el éxito de la distribución comercial, que usted proporcione sus consumidores potenciales con un programa que puede ser plenamente probado en una cuenta demo. Por lo tanto, el usuario puede ponderar todas las ventajas del programa y llegado a la decisión de compra.

La aplicación de una contraseña es incorrecta - la distribución no autorizada de este programa es posible, en este caso. Así que la persona la contraseña para el usuario debe estar supeditado a la cuenta número real. No hay necesidad de utilizar la contraseña para los clientes corporativos (si el dealing center han comprado la licencia para todos sus comerciantes) - el programa libre debe identificar el servidor que el Terminal de Usuario está conectado a. Y, si es el servidor de la empresa cliente a continuación, cada usuario debe ser capaz de trabajar sin obstáculos.

La solución del problema 39 la limitación de los derechos de uso del programa pueden ser los siguientes (check.mq4 EA):

```
//---------------------------------------------- ----------
// C heck.mq4
// El código debería ser usado para fines educativos únicamente.
//---------------------------------------------- ---------- 1 --
extern int Parol = 12345;
//---------------------------------------------- ---------- 2 --
int start () // función especial 'Inicio'
(
if (Check () == false) // Si el uso de condiciones no ..
return; // .. cumplen los requisitos, entonces la salida

// El código principal del programa debe ser especificado aquí
Alert ( "Programa de ejecución"); // ejemplo de alerta

return; // Salir de inicio ()
)
//---------------------------------------------- ---------- 3 --
bool Check () // función definida por el usuario de ..
(// ..  condiciones de uso de cheques
if (IsDemo () == true) // Si se trata de una cuenta de demostración, entonces ..
return (true); // ..  no hay otras limitaciones
if (AccountCompany () == "SuperBank") // La contraseña no es necesaria
return (true); // .. para clientes corporativos
int clave = AccountNumber () * 2 + 1000001; // Cálculo clave
if (Parol == clave) // Si la contraseña es correcta, entonces ..
return (true); // .. permitir que la cuenta real para el comercio
Alert ( "Wrong password. AE no funciona.");
return (false); // Salir función definida por el usuario
)
//---------------------------------------------- ---------- 4 --
```

El necesario control se realiza en los niveles superiores de entradas especiales de la función start (), en este ejemplo (bloque 2-3):

```
if (Check () == false) // Si el uso de condiciones ..
```

Si el Check () función devuelve falso como resultado de la comprobación, entonces el control se pasa al return y el operador especial la función start () termina su trabajo. El código principal del programa se encuentra directamente después de esta comprobación. El Check () devolverá true si la comprobación es exitosa, entonces el código principal se llevará a cabo.

La comprobación se realiza en función de tres criterios en la función definida por el usuario Check ():
-- La cuenta es una demostración;
-- ¿El servidor pertenece a una empresa cliente;
-- Es la contraseña válida para la cuenta real.

La función IsDemo () se utiliza para comprobar el acuerdo con el primer criterio.

Función IsDemo ()

```
bool IsDemo ()
```

La función devuelve TRUE, si el programa está trabajando con una cuenta demo. De lo contrario, devuelve FALSE.

Si el IsDemo () devuelve true, entonces el usuario está trabajando con la demo-cuenta. Esto significa que no hay necesidad de más control (porque el uso de programa con una demostración de cuenta es gratuita para todos). El Check () termina su trabajo y devuelve cierto, en este caso:

```
if (IsDemo () == true) // Si se trata de una cuenta de demostración, entonces ..
return (true); // .. no hay otras limitaciones
```

Pero si la IsDemo () función devuelve el valor falso entonces el usuario está trabajando con la cuenta real. Es necesario averiguar si el usuario tiene suficientes derechos, en este caso. El AccountCompany () se emplea en este ejemplo para comprobar los clientes corporativos.

Función AccountCompany ()

```
AccountCompany cadena ()
```

La función devuelve el nombre de la empresa en la cuenta corriente registrada en.

Si el control como resultado:

```
if (AccountCompany () == "SuperBank") // La contraseña no es necesaria ..
return (true); // .. para clientes corporativos
```

la determinación de que el nombre de la empresa se corresponde con la especificada en el programa, entonces el Check () terminarán su trabajo y restablecer la verdad - el control se ha completado correctamente. Si se puso de manifiesto que el cliente está conectado a la otra empresa (que no es una empresa cliente), entonces no hay necesidad de averiguar si él \ ella tiene una licencia individual.

La entrada:

```
int clave = AccountNumber () * 2 + 1000001; // Cálculo clave
```

pone el algoritmo para el cálculo de una clave para cualquier cuenta en el programa. Este ejemplo contiene el método sencillo. El programador, como él \ ella considere oportuno, puede insertar más complejo método de cálculo clave. De todos modos, el algoritmo debe considerar un número de cuenta que está disponible para el programa mediante el uso de la AccountNumber ().

Función AccountNumber ()

```
INT AccountNumber ()
```

La función devuelve el número de la cuenta corriente.

La contraseña previamente calculada utilizando el mismo algoritmo se pasa al usuario. Si el control ha dado lugar a descubrir que la contraseña y la clave coincide con los demás, entonces el Check () termina su trabajo y devuelve verdadero valor:

```
if (Parol == clave) // Si la contraseña es correcta, entonces ..
return (true); // .. permitir que la cuenta real para el comercio
```

Si ninguno de los procedimientos de control se ha completado correctamente entonces el usuario no pueda utilizar el comercio una verdadera cuenta. En este caso, el Check () termina su trabajo y devuelve falso valor después de hacer el anuncio adecuado. De tal manera el uso no autorizado del programa intento es suprimida.

Funciones de clientes que regresan de información terminal

Función	Resumen de Información
TerminalCompany	Devuelve el nombre de la empresa propietaria de la Terminal de Usuario.
TerminalName	Devuelve el nombre del Terminal de Usuario.
TerminalPath	Nos devuelve el directorio de la Terminal de Usuario se ejecuta.

Funciones de Detección de la situación actual del Terminal de Usuario incluido el medio ambiente Situación de la MQL4 Programa Ejecutado

Función	Breve descripción
GetLastError	La función devuelve el último código de error, tras lo cual el valor de la variable Last_Error especial que contiene el último código de error se pone a cero. Así que la próxima convocatoria de la función GetLastError devolverá el valor 0.
IsConnected	Devuelve el estado de la conexión utilizada para la transferencia de datos entre el cliente y el terminal servidor. TRUE - la conexión con el servidor se ha establecido, FALSE - no hay conexión con el servidor o la conexión se pierde.
IsDemo	Devuelve TRUE si un programa trabaja con una demostración de cuenta. De lo contrario, devuelve FALSE.
IsDllsAllowed	Devuelve TRUE si DLL llamando funciones están permitidos para un EA. De lo contrario, devuelve FALSE.
IsExpertEnabled	Devuelve TRUE si el lanzamiento de AE es permitido en el Terminal de Usuario. De lo contrario, devuelve FALSE.
IsLibrariesAllowed	Devuelve TRUE si la AE es capaz de declarar una función de librería. De lo contrario, devuelve FALSE.
IsOptimization	Devuelve TRUE si un AE está trabajando en la optimización de modo de prueba. De lo contrario, devuelve FALSE.
IsStopped	Devuelve TRUE si un programa (o script EA) recibió un comando para salir a trabajar. De lo contrario, devuelve FALSE.
IsTesting	Devuelve TRUE si un AE está trabajando en el modo de ensayo. De lo contrario, devuelve FALSE.
IsTradeAllowed	Devuelve TRUE si un AE se permite al comercio y el tráfico es libre para el comercio. De lo contrario, devuelve FALSE.
IsTradeContextBusy	Devuelve TRUE si el tráfico para el comercio está ocupado. De lo contrario, devuelve FALSE.
IsVisualMode	Devuelve TRUE si AE es una prueba en el modo de visualización. De lo contrario, devuelve FALSE.
UninitializeReason	Devuelve el código de la razón para la operación de terminación de una EA, un indicador de la usuario o un script. Regresadas valores puede ser uno de los códigos deinicialización. Esta función se puede llamar a la de inicio () para analizar las razones de deinicialización del anterior lanzamiento, también.

Funciones de Acceso a la Información sobre la Cuenta Activa

Función	Breve descripción
AccountBalance	Devuelve un valor del saldo de la cuenta activa (la cantidad total de dinero en la cuenta).
AccountCredit	Se devuelve un valor de crédito de la cuenta activa.
AccountCompany	Devuelve el nombre de una empresa de corretaje de la balanza por cuenta corriente se registra al.
AccountCurrency	Nos devuelve la moneda nombre de la cuenta corriente.
AccountEquity	Devuelve el valor de la cuenta corriente. El cálculo de capital depende de la configuración del servidor.
AccountFreeMargin	Devuelve el valor de la libertad de margen permitido para los órdenes de abrirse una cuenta corriente.
AccountFreeMarginCheck	Devuelve el valor de la libertad de margen que se mantendrá después de la posición especificada se ha abierto en la cuenta corriente.
AccountFreeMarginMode	El cálculo del margen de libre cantidad de modo abierto órdenes de la balanza por cuenta corriente.
AccountLeverage	Devuelve el valor multiplicador de la cuenta corriente.
AccountMargin	Se devuelve el importe del margen se utiliza para mantener las posiciones abiertas en la cuenta corriente.
AccountName	Devuelve el nombre de usuario de la cuenta corriente.
AccountNumber	Nos devuelve el número de la cuenta corriente.
AccountProfit	Devuelve la rentabilidad de la cuenta corriente calculado en la divisa base.
AccountServer	Devuelve el nombre del servidor activo.
AccountStopoutLevel	Devuelve el valor del nivel que se utiliza para identificar el estado StopOut.
AccountStopoutMode	Nos devuelve el modo de cálculo StopOut nivel.

Para obtener la descripción detallada de estas y otras funciones, por favor consulte la documentación a MQL4.community, a MetaQuotes Software Corp sitio web o en la "Ayuda" de la sección MetaEditor.

Funciones del Comercio

Todas las funciones comerciales se pueden dividir en dos grupos - funciones que forman el comercio órdenes y funciones que devuelven un poco de orden que caracterizan a los valores. MQL4 sólo tiene cinco funciones que forman el comercio y enviar órdenes a un servidor:

- OrderSend () - Para la apertura del mercado ya la espera de orden de colocación;
- OrderClose () - Para el cierre del mercado;
- OrderCloseBy () - cierre de mercado frente a los órdenes;
- OrderDelete () - suprimir la espera de órdenes;
- OrderModify () - la modificación de todos los tipos de órdenes.

La orden de utilizar las funciones enumeradas anteriormente se describe en la Programación de Operaciones de Comercio en la sección detalles. Todas las demás funciones que no forman comercio órdenes, pero su utilización es a menudo necesario. Por ejemplo, a veces es necesario cerrar algunas órdenes de prioridad en secuencia. Para hacerlo, debe analizar las características de cada orden en el programa, a saber: - Tipo de orden, mucha cantidad, para dejar de órdenes de localización, etc Vamos a examinar algunas funciones que permiten obtener la información sobre un pedido.

OrderTotal ()

```
INT OrdersTotal ()
```

La función devuelve el número total de locales abiertos ya la espera de órdenes.

OrderTakeProfit ()

```
OrderTakeProfit doble ()
```

La función devuelve el valor declarado de los precios cuando el nivel de beneficio (tener ganancias) del actual seleccionado se llegue a un fin. La orden debe ser previamente seleccionados mediante el OrderSelect ().

OrderProfit ()

```
OrderProfit doble ()
```

Devuelve el valor de beneficios netos (sin tener en cuenta a los canjes y comisiones) de los seleccionados el orden. Es la ganancia no realizada por el abierto y fija las órdenes de beneficio para el orden cerrado. La orden debe ser previamente seleccionados mediante el OrderSelect ().

OrderLots ()

```
OrderLots doble ()
```

Devuelve la cantidad de lotes de un determinado orden. La orden debe ser previamente seleccionados mediante el OrderSelect ().

El fragmento del programa que calcula el precio declarado cerca TakeProfit, para el beneficio y la cantidad de lotes se muestra a continuación:

```
for (int i = 0; i <OrdersTotal (); i + +) // Para todos los órdenes
(
if ((OrderSelect (i, SELECT_BY_POS) == true) // Si existe siguiente
(
doble TP OrderTakeProfit = (); // TakeProfit de orden
Beneficio doble OrderProfit = (); // Orden de beneficios
Lotes doble OrderLots = (); // Cantidad de lotes
//..... TP valores y beneficios de uso en el programa .....
)
) // Fin del cuerpo del bucle
```

Es evidente que cada función considerada (OrderTakeProfit (), OrderProfit () и OrderLots ()) no tiene parámetros ajustables, es decir, de denotación, por ejemplo, el número del pedido, para devolver el valor correspondiente a las características de este individuo Para no está involucrado.

Para calcular las características de un individuo para (declarado precio de una de las órdenes stop-, para el beneficio y la cantidad de lotes que se presentan en este contexto), hay que seleccionar el orden necesario en primer lugar; este informará el programa sobre la orden de realizar cálculos. Con el fin de hacer que usted debe ejecutar la OrderSelect () antes de iniciar los cálculos (véase el Clausura y borrar órdenes). El comercio funciones ejecutado después de que el return de los valores que se corresponden con las características seleccionadas fin.

La correcta evaluación de una u otra de características para el programador no es poco significativo. Por ejemplo, cuando la solución del problema de orden de clausura secuencia, debe establecer un criterio de cálculo para que se cierre antes y que uno - después. Echemos un vistazo a la tarea sencilla.



Problema 40. Compre dos órdenes actualmente están abiertos en un solo símbolo. La primera de ellas se abre al precio de 1.2000 a 0,5 lote, el segundo se abre al precio de 1.3000 a 1 lote. El precio actual es 1,3008. El criterio para el comercio Comprar órdenes de clausura ha desencadenado. Es necesario hacer una decisión correcta, a saber, a fin de decidir qué debe ser cerrado como el primero y que, como el segundo.

Obviamente, el beneficio de la primera orden hace 108 puntos, mientras que la de la segunda es de 8 puntos. Aunque el primer fin se abre a una cantidad menor de los lotes, tiene el mayor beneficio que la segunda, a saber, el beneficio de primer orden es de $ 540 y el beneficio de segundo orden es de $ 80. El cierre de la primera orden puede parecer preferible a ser, a primera vista, porque tiene una mayor ganancia. Sin embargo, se trata de un misthought. Es necesario examinar las posibles hipótesis para hacer una decisión correcta.

La orden de clausura secuencia no importa, si el precio se sabe que no cambian durante el período de los órdenes de clausura. Sin embargo, los precios pueden cambiar durante la ejecución de la instrucción de cerrar uno de los órdenes. Por lo tanto, el fin de que puedan traer más pérdidas, en un escenario negativo, debe ser cerrado en primer lugar. Si el precio se hunde un punto abajo, el beneficio de primer orden se reducirá en $ 5, mientras que el de la segunda lo hará en $ 10. Obviamente, la segunda orden de traer más pérdidas, por lo que debe ser cerrado en primer lugar. De esta forma, el importe de los lotes que tiene el significado dominante la hora de decidir sobre el cierre de fin de secuencia. Rentables los casos no se puede considerar aquí, porque el comercio se desarrolla con los criterios de comercio en el programa, y esta vez el criterio de Compra órdenes de clausura ha desencadenado.

Usted debe considerar las otras características de orden si es necesario elegir entre dos órdenes con la misma cantidad de lotes. Por ejemplo, se puede considerar la distancia entre el precio actual y el StopLoss valor de cada pedido. Al mismo tiempo, debe analizar cuál de las órdenes traería más pérdidas, en caso de rápido movimiento de precios. La respuesta es obvia, como así: el uno (de ambos órdenes que se abren a la misma cantidad de lotes) que tiene su StopLoss nivel más lejos de la actual precio.

De este modo se puede analizar la prioridad y todos los demás parámetros de los órdenes y compilar la prioridad orientada a la lista de criterios a considerar a la hora de tomar la decisión sobre el cierre de órdenes. No es difícil identificar los criterios que no deben ser considerados. Está abierto de precios (y los correspondientes beneficios de orden), por ejemplo. La cantidad de dinero que el comerciante tiene en este momento se muestra en la columna de Equidad de la Terminal de Usuario. La fuente de este valor no es importante a que, ni se trata de un resultado de la pérdida de una o más órdenes, ni es consecuencia de un beneficio.

Todas las características necesarias de una orden se pueden recibir mediante el correspondiente comercio funciones:

Funciones del Comercio

Función	Resumen de Información
Errores de Ejecución	Cualquier operación de comercio (OrderSend, OrderClose, OrderCloseBy, OrderDelete o OrderModify funciones) puede, sin éxito final de una veintena de motivos y el regreso, ya sea negativo o número de billete FALSO. Usted puede averiguar la razón del fracaso de la utilización de GetLastError función. Cada error debe ser procesado en su propio camino. En el cuadro siguiente contiene las recomendaciones generales.
OrderClose	Se cierra la posición. Devuelve TRUE, si la función ha terminado con éxito.

OrderCloseBy	Devuelve FALSE, si la función no final. Se cierra una posición abierta con los demás que se abre en sentido contrario por el mismo símbolo. Devuelve TRUE. si la función ha terminado con éxito. Devuelve FALSE, si la función no final.
OrderClosePrice	Nos devuelve el precio de cierre de los seleccionados el orden.
OrderCloseTime	Devuelve el momento del cierre de los seleccionados el orden.
OrderComment	Nos devuelve el comentario de los seleccionados el orden.
OrderCommission	Devuelve el valor calculado comisión de los seleccionados el orden.
OrderDelete	Elimina la previamente en espera de orden. Devuelve TRUE, si la función ha terminado con éxito. Devuelve FALSE, si la función no final.
OrderExpiration	Devuelve la fecha de expiración de los seleccionados en espera de orden.
OrderLots	Devuelve la cantidad de lotes seleccionados de la orden.
OrderMagicNumber	Devuelve la identificación ("magia") el número de seleccionados.
OrderModify	Se modifica los parámetros de las órdenes previamente abierto ya la espera de órdenes. Devuelve TRUE si la función ha terminado con éxito. Devuelve FALSE, si la función no final.
OrderOpenPrice	Nos devuelve el precio de abrir el seleccionado orden.
OrderOpenTime	Nos devuelve el tiempo de apertura de los seleccionados el orden.
OrderPrint	Se entra en el orden de información a la revista.
OrderProfit	Nos devuelve el beneficio neto (sin tener en cuenta a los canjes y comisiones) de los seleccionados el orden. Es la ganancia no realizada por el abierto y fija las órdenes de beneficio para el orden cerrado.
OrderSelect	La función escoge el fin de trabajar con posterioridad. Devuelve TRUE si la función ha terminado con éxito. Devuelve FALSE, si la función no final.
OrderSend	La función principal para la apertura de las órdenes y colocar órdenes pendientes de ser ejecutadas. Nos devuelve el número del billete que se le asignó a la orden el comercio de servidor, o -1, en caso de no terminar la operación.
OrdersHistoryTotal	Nos devuelve el número de posiciones cerradas y se eliminarán las órdenes en la historia de la balanza por cuenta corriente, con carga a la Terminal de Usuario.
OrderStopLoss	Devuelve una estrecha precio de la posición cuando alcanza el nivel de pérdidas (stop) de la seleccionada actualmente.
OrdersTotal	Devuelve el número total de abierto ya la espera de órdenes.
OrderSwap	Devuelve el valor de intercambio del orden seleccionado actualmente.
OrderSymbol	Devuelve el símbolo de nombre para el objeto seleccionado actualmente.
OrderTakeProfit	Nos devuelve el precio de cerca el momento en que llegue el nivel de beneficio (tener ganancias) de la seleccionada actualmente.
OrderTicket	Nos devuelve el número de ticket de la seleccionada actualmente.
OrderType	Nos devuelve el tipo de operación seleccionado actualmente.

Para obtener la descripción detallada de ésta y otras funciones, usted debe hacer referencia a la documentación a MQL4.community, a MetaQuotes Software Corp sitio web o en la "Ayuda" de la sección MetaEditor.

Creación de un Programa Normal

Estructura de un programa normal

La característica más destacada de un programa normal, es que su estructura permite la utilización de funciones definidas por el usuario fácilmente. Para que sea más cómodo, las funciones definidas por el usuario son colocadas en archivos separados con la extensión ".mqh" y son almacenadas en la ruta "C:\Archivos de Programa\Carpeta MetaTrader\experts\include\"

Por lo general, un programa normal utiliza las tres funciones especiales, y estas llaman a las funciones definidas por el usuario. A su vez, las funciones definidas por el usuario, también pueden llamar a otras funciones definidas por el usuario, donde cada función, cumple una tarea específica y limitada dentro del programa.

Un asesor experto, puede contener funciones definidas por el usuario con las más diversas características. Algunas funciones, por ejemplo, pueden dedicarse al seguimiento de eventos y gestión de datos, otras funciones se pueden dedicar a abrir y cerrar posiciones, unas terceras funciones se pueden dedicar a hacer cálculos, por ejemplo la lógica de la estrategia, o calcular el costo de las órdenes, etc. La decisión sobre que funciones utilizar, depende de la finalidad que tenga el asesor experto y que utilidad quiera darle el usuario. En la figura 155, se puede ver el diagrama de la estructura de un asesor experto o EA normal, donde se utilizan funciones definidas por el usuario.

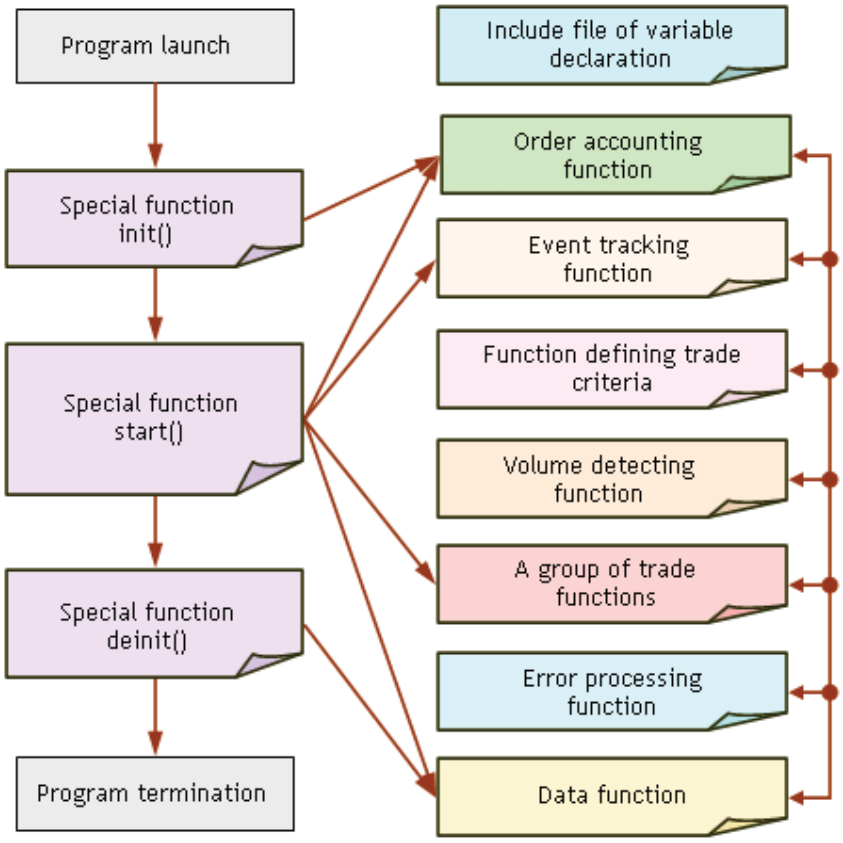


Fig. 155. Diagrama de la estructura de un programa normal (asesor experto).

Las flechas en el diagrama muestran las relaciones entre las funciones. Por ejemplo, la función que lleva la contabilidad de las ordenes en el EA, se llama desde las funciones especiales init() y start() pero también se puede llamar desde otra parte del programa. A la derecha del diagrama, se ven las flechas que conectan entre sí a las funciones definidas por el usuario. Por ejemplo, la función que contiene la lógica de la estrategia (Funcion defining trade criteria), no es llamada desde las funciones especiales, pero si desde la función que lleva la contabilidad. L función "Data" es llamada desde la función especial deinit() y, si es necesario, también es llamada desde la función que procesa los errores, o la función que maneja la

contabilidad o también la función que procesa los eventos. El archivo que contiene las variables compartidas por todas las funciones, y que está en la cabecera, no es llamado desde ninguna función, puesto que no contiene ninguna función que pueda ser llamada o ser ejecutada. Este archivo solo contiene las declaraciones de las variables globales o compartidas, por esta razón solo es una pequeña parte del EA. Para entender como es que se relacionan las diferentes partes de un EA, vamos a ver como se incluyen y crean estos archivos y en que orden.

Usando archivos externos con "include"

Si un programa contiene gran cantidad de líneas de código, es difícil encontrar y eliminar los errores. El programador tiene que desplazarse por el código muchas veces para hacer retoques al código en una o en otra parte. En estos casos es muy conveniente y cómodo dividir el código en partes, cada una como un archivo separado. En estos archivos separados se puede colocar cualquier parte del código del programa. Es común que cada función este separada en un archivo diferente. Si varias funciones se interconectan lógicamente, un archivo incluido puede tener la descripción de todas las funciones definidas por el usuario.

En la sección ***"información de una cuenta"*** vimos un código de ejemplo (check.mq4) que protegía del uso desautorizado de un programa. En el asesor experto check.mq4 vimos una función que se encargaba de esa protección y que se llamaba Check(). En el código de un EA que vamos a ver más adelante (usualexpert.mq4), vamos a volver a usar la función Check(), pero esta vez la vamos a colocar en un archivo aparte (Check.mqh) y luego la vamos a incluir en el código principal.

Función definida por el usuario Check()

bool Check(). Esta función devuelve TRUE si las condiciones de protección se cumplen, si no devuelve FALSE. Las condiciones para devolver TRUE son:

- Si el programa es usado en una cuenta demo;
- Si la cuenta se abre con SuperBank;
- Si el usuario a colocado el código correcto en la variable externa Parol.

El siguiente es el archivo Check.mqh que contiene la descripción del funcionamiento de la función Check();

```
//+------------------------------------------------------------------+
// Check.mqh
// Este programa esta pensado para servir de ejemplo en el tutorial de MQL4.
//------------------------------------------------------------------- 1 --
// Función que comprueba si es legal usar el programa
// Entradas:
// - variable global 'Parol'
// - constante local "SuperBank"
// Devuelve los valores:
// true – si las condiciones se cumplen
// false - si las condiciones son violadas
//------------------------------------------------------------------- 2 --
extern int Parol=12345; // Contraseña con la que trabaja el programa real
//------------------------------------------------------------------- 3 --
bool Check() // Función definida por el usuario
{
if (IsDemo()==true) // Si la cuenta es demo
return(true); // .. entonces no hay limintaciones
if (AccountCompany()=="SuperBank") // Para los clientes corporativos…
return(true); // …no se requiere contraseña
int Key=AccountNumber()*2+1000001; // Calcular la clave
if (Parol==Key) // Si la contraseña esta bien..
```

```
return(true); // ..entonces se puede usar en una cuenta real.
Inform(14); // Envía un informe desautorizando el uso
return(false); // Sale de la función.
}
//---------------------------------------------------------------------------- 4 –
```

Es fácil ver que el nombre del archivo Check.mqh es igual que el de la función Check(). Esto no es una regla de MQL4. No es necesario que sean iguales y más si nos damos cuenta que un archivo ".mqh" puede tener varias funciones a dentro. Sin embargo, es muy práctico colocar el mismo nombre a la función y al archivo. Esto facilita enormemente el trabajo del programador. Usando el nombre de la función, sabrá que el código de esta estará en la ruta "…\experts\include\" con el mismo nombre. Para incluir un archivo externo ".mqh" debemos utilizar la directiva o palabra clave #include.

Directiva #include

```
#include <Nombre del archivo>
#include " Nombre del archivo"
```

La directiva #include se puede colocar en cualquier parte del programa. Sin embargo por legibilidad, se debe colocar al inicio del programa. El pre compilador sustituirá #include < Nombre del archivo > (o #include "Nombre del archivo ") por el código del archivo que tenga ese nombre.
Los corchetes menores/mayores <> significan que el archivo será tomado del directorio predeterminado Terminal_directory "…\experts\include\". Si el nombre de archivo se coloca en "comillas", será buscado en el directorio actual, a saber, en el directorio que contiene el archivo principal.
Abajo esta un asesor experto normal, usualexpert.mq4. Todos los archivos incluidos o importados, se colocan en la parte principal del programa.

```
//---------------------------------------------------------------------------------------
// usualexpert.mq4
// Este código solo debe ser usado de manera educacional.
 //------------------------------------------------------------------------------- 1 --
#property copyright "Copyright © Book, 2007"
#property link "http://AutoGraf.dp.ua"
//------------------------------------------------------------------------------- 2 --
#include <stdlib.mqh>
#include <stderror.mqh>
#include <WinUser32.mqh>
//------------------------------------------------------------------------------- 3 --
#include <Variables.mqh> // Variables con descripción
#include <Check.mqh> // Verifica la legalidad del uso
#include <Terminal.mqh> // Función de la contabilidad
#include <Events.mqh> // Seguimiento de los eventos
#include <Inform.mqh> // Función de manejo de datos
#include <Trade.mqh> // Función de operaciones
#include <Open_Ord.mqh> // Apertura de ordenes
#include <Close_All.mqh> // Cierre de ordenes
#include <Tral_Stop.mqh> // Modificación de StopLoss
#include <Lot.mqh> // Calculo de numero de lotes
#include <Criterion.mqh> // Lógica de la estrategia
#include <Errors.mqh> // Procesamiento de Errores
//------------------------------------------------------------------------------- 4 --
int init() // Función especial 'init'
```

```
{
Level_old=MarketInfo(Symbol(),MODE_STOPLEVEL );//Distancia mínima
Terminal(); // Verifica la contabilidad de la cuenta
return; // Salida de init()
}
//---------------------------------------------------------------------------- 5 --
int start() // Función especial 'start'
{
if(Check()==false) // Si no cumple las condiciones...
return; // ..termina la ejecución.
PlaySound("tick.wav"); // Ejecuta un sonido en cada tic
Terminal(); // Lleva la contabilidad
Events(); // Información sobre los eventos
Trade(Criterion()); // Ejecuta un trade si se cumplen los criterios
Inform(0); // Cambia el color de los textos que llevan el informe
return; // Sale de start()
}
//---------------------------------------------------------------------------- 6 --
int deinit() // Función especial deinit()
{
Inform(-1); // Elimina todos los objetos gráficos
return; // Sale de deinit()
}
//---------------------------------------------------------------------------- 7 –
```

En el bloque 2 – 3 incluimos los archivos predeterminados stdlib.mqh, stderror.mqh y WinUser32.mqh usando la directiva #include. Estos archivos no son indispensables, sin embargo son una buena ayuda. Por ejemplo, el archivo stderror.mqh contiene la definición de los códigos de error, que nos hará mas fácil comprender estos códigos. Si no se están procesando los errores en un programa, este archivo no será necesario. Pero generalmente es útil y normal incluir estos archivos.

En el bloque 3 – 4 el programa incluye algunos archivos que contienen funciones definidas por el usuario. En la línea que esta abajo, agregamos la función Check.mqh.

```
#include <Check.mqh> // Verifica la legalidad del uso
```

Uno puede pensar que un código como el anterior (usualexpert.mq4) es interpretado por el compilador tal cual como se ve. Sin embargo lo que el pre compilador hace, es sustituir cada línea con la directiva #include, por el código que el archivo tiene. De esta manera, se compila todo como si fuera solo un único archivo ".mq4" que genera un archivo ".ex4".

En general todos los archivos externos contienen funciones. Sin embargo en el código anterior hay un archivo muy importante que no tiene ninguna función ejecutable. En vez de tener una función, este archivo contiene todas las variables que se utilizaran en el programa, y que se compartirá entre todos los archivos.

```
#include <Variables.mqh>
```

El código de este archivo es el siguiente:

```
//---------------------------------------------------------------------------
// Variables.mqh
// Este código solo debe ser usado de manera educacional.
//---------------------------------------------------------------------- 1 --
// Descripción de variables globales
extern double Lots = 0.0;// Cantidad de lotes
```

```
extern int Percent = 0; // Porcentaje de fondos asignados
extern int StopLoss =100; // StopLoss para las nuevas órdenes en pips
extern int TakeProfit =40; // TakeProfit para las nuevas órdenes en pips
extern int TralingStop=100; // TralingStop para ordenes de mercado en pips
//---------------------------------------------------------------------- 2 --
int
Level_new, // Nuevo valor para la mínima distancia
Level_old, // Previo valor para la mínima distancia
Mas_Tip[6]; // Array con tipos de ordenes
// [] order type: 0=B,1=S,2=BL,3=SL,4=BS,5=SS
//---------------------------------------------------------------------- 3 --
double
Lots_New, // Cantidad de lotes para las nueva ordenes
Mas_Ord_New[31][9], // Array con el listado de ordenes abiertas.
Mas_Ord_Old[31][9]; // .. Array con el listado de ordenes abiertas previas.
// 1st index = Numero de ordenes abiertas
// [][0] Sin datos
// [][1] Precio de apertura de la orden
// [][2] StopLoss de la orden
// [][3] TakeProfit de la orden
// [][4] Numero de la orden
// [][5] Volumen de la orden
// [][6] Tipo de orden 0=B,1=S,2=BL,3=SL,4=BS,5=SS
// [][7] Número mágico de la orden
// [][8] 0/1 para saber si la orden tiene comentarios
//---------------------------------------------------------------------- 4 –
```

Según las reglas que maneja MQL4, las variables globales se deben declarar al inicio del código. Por esta razón, el archivo Variables.mqh se incluye al inicio del programa, y está situado por encima de los archivos que contienen las funciones que utilizaran estas variables.

En algunos casos (raros), es técnicamente posible declarar una variable global en un archivo incluido que contenga una función, en la cual el valor de esta variable primero se utiliza en el programa. En tales casos, es necesario guardar el archivo respetando la regla de posicionamiento de las variables.

En otros casos, es incluso técnicamente imposible hacerlo. Por ejemplo, si tenemos dos archivos incluidos, cada uno de ellos con dos variables globales, una de las cuales se declara en un archivo y la otra se declara en el otro archivo, entonces se obtendrá un error al compilar un programa, ya que, independientemente del orden, una de las variables se utiliza antes de que se declare en el programa. Esta es la razón por la cual, es una práctica habitual y recomendada en los programas normales, declarar todas las variables globales, sin excepción alguna, en un único archivo, y incluirlo en el programa, antes que los otros ficheros que contengan funciones definidas por el usuario.

En el bloque 1-2 del archivo Variables.mqh, las variables externas que se declaran son las que determinan la cantidad de lotes para nuevas órdenes, también hay una variable que determina el porcentaje asignado para nuevos pedidos, también está el stoploss, el TakeProfit y el TralingStop o stop fluctuante. En el bloque 2-4, se declaran otras variables globales que se explicaran y quedaran claras más adelante. Cada archivo externo será explicado paso a paso, en los capítulos subsiguientes.

Llevando la contabilidad de las órdenes

Hemos mencionado anteriormente que no hay reglas estrictas para diseñar los algoritmos en los programas. Al mismo tiempo, la gran mayoría de los algoritmos tiene que tomar decisiones teniendo en cuenta las órdenes ya abiertas. En algunos casos, por ejemplo, en una estrategia solo puede haber una orden abierta por vez. En otros casos si se permite que hayan varias órdenes abiertas al mismo tiempo. También sabemos de otros algoritmos que funcionan colocando dos órdenes pendientes en diferentes direcciones.
Para cumplir los requisitos que exige una estrategia u otra, usted tiene que estar enterado de la situación actual. ¿Qué órdenes de mercado y órdenes pendientes están abiertas, y cuáles son sus características? Para responder esta pregunta, usted puede utilizar dos posibles soluciones.

En la primera solución, se escribe el fragmento de código que responde a esta pregunta, en cada lugar donde se necesite. Esta solución es técnicamente viable, pero resulta ser ineficiente, si desea hacer cambios en el algoritmo. En este caso, el programador tiene que buscar y analizar todos los lugares donde se encuentra este código, y hacer los cambios correspondientes. La segunda solución, es crear una función universal que actualice el listado de órdenes de mercado abiertas, y que esta función se ejecute cada vez que se necesite. Por un lado, esta función permite hacer el código más corto y eficaz. Por otro lado, da la opción de reutilizar la misma función para otros programas.

Para crear una función que lleve la contabilidad de nuestras órdenes correctamente, tenemos que saber que datos hay que calcular. En la mayoría de los casos, los valores siguientes son vitales para llevar la contabilidad y así poder tomar decisiones en base a ella:

- cantidad total de órdenes
- la cantidad de ordenes abiertas para cada tipo de orden (por ejemplo, la cantidad de compras, ventas , SellStop, o BuyLimit);
- todas las características de cada orden (ticket, StopLoss y TakeProfit, lotes, etc.).

Esta información tiene que estar disponible para todas las funciones, en especial para las que procesan esta información. Por esta razón, toda esta información que se necesita para llevar la contabilidad tiene que estar guardada en Arrays globales. En total debe haber tres arrays para guardar esta información.

- Un Array para llevar toda la información de las posiciones abiertas y órdenes pendientes. Este array debe contener toda la información de cada posición y debe tener la información actualizada desde la última vez que se ejecuto la función. Este array lo vamos a llamar Mas_Ord_New.

- Un Array idéntico al anterior con la salvedad de que va a guardar la información desde la penúltima vez que se ha ejecutado esta función. Más adelante vamos a entender el porqué. Este Array lo vamos a llamar Mas_Ord_Old.

- Un Array llamado Mas_Tip, que nos va a servir para llevar un listado de los diferentes tipos de órdenes abiertas y la cantidad de cada tipo de orden.

Las Arrays Mas_Ord_New y Mas_Ord_Old son parecidas y con dimensiones equivalentes. La diferencia está en que el primero refleja el estado actual de las órdenes y el segundo refleja el estado anterior al actual. Vamos a ver en detalle la estructura de estos dos Arrays.
TRADUCIR LA TABLA EN EL ARCHIVO FINAL

tabla 4. Correspondencia de los elementos de matrices Mas_Ord_New y Mas_Ord_Old con características de la orden.

	No definido	Abrir el precio	StopLoss	TakeProfit	Número de pedido	Volumen, en lotes	Tipo de orden	Número mágico	Comentario
Índices	0	1	2	3	4	5	6	7	8
0	2.0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
1	0,0	1.2583	1.2600	1.2550	123456.0	1.4	1.0	1177102416.0	1.0
2	0,0	1.2450	1.2580	1.2415	123458.0	2.5	2.0	1177103358.0	0,0
3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
. . .	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
30	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0

El primer índice del array (filas) representa el número de la orden en el array. Las propiedades de la primera orden (ordenes abiertas y ordenes pendientes) se colocan en el índice 1 (y no en el 0), las de la segunda orden en el índice 2, etc. El tamaño del array para el primer índice es de 31 casillas, de esta

manera, el array puede almacenar hasta 30 órdenes al mismo tiempo. Si su estrategia necesita tener más de 30 órdenes abiertas, deberá cambiar este número en la declaración del array. En la mayoría de los casos, la cifra de 30, supera por mucho la cifra que se necesita normalmente, que generalmente es de 2 a 10 – 15 ordenes abiertas al mismo tiempo. Colocamos la cifra de 30 en este ejemplo, porque queremos mostrar que este array puede ser útil aun, en estrategias muy inusuales.

En el segundo índice del array (columnas), se representa las propiedades de las ordenes. Por ejemplo, cada elemento del array con el índice número 1, contiene el precio en que se abrió o se va abrir una orden, en el índice 2, se encentra el StopLoss, en el 3 el TakeProfit etc (Vea el cuadro 4). En el elemento [0] [0] del array se guardaran el número total de ordenes abiertas o ordenes pendientes que hay en el array. En los demás elementos que contienen índice 0, no hay ninguna información, exceptuando como ya dijimos el índice [0] [0].

En el cuadro 4, se representa un array "Mas_Ord_New" que contiene la información de dos órdenes abiertas. Como podemos ver, el elemento Mas_Ord_New[0][0] tiene el valor de 2, que representa la cantidad de ordenes abiertas en este momento. Por ejemplo el índice 1 del array Mas_Ord_New[1] contiene la información de una orden de venta (Mas_Ord_New [1] [6] ver tipos de ordenes) abierta con 1.4 lotes (Mas_Ord_New[1][5] =1.4) y con el numero 123456.0 (Mas_Ord_New[1][4] =123456.0). El valor Mas_Ord_New [1] [8] =1.0 significa que esta orden tiene un comentario agregado. En el segundo índice Mas_Ord_New[2] existe una segunda orden de tipo BuyLimit, Mas_Ord_New[2][6].

Veamos el tercer array Mas_Tip. Este array representa la cantidad de órdenes de cada tipo. A cada tipo de orden se le asigna su código correspondiente (ver tipos de ordenes). Esto significa que el elemento Mas_Tip [0] contiene la cantidad de órdenes de compra, Mas_Tip [1] las órdenes de venta, Mas_Tip [2] las ordenes BuyLimit y así, etc. Relacionando la tabla 4, Mas_Tip tendría la siguiente forma.

Table 5. Correspondence of the elements of array Mas_Tip with the amount of orders of different types.

	Buy	Sell	BuyLimit	SellLimit	BuyStop	SellStop
Index	0	1	2	3	4	5
Value	0	1	1	0	0	0

En este caso, Mas_Tip Mas_Tip [1] nos dice que hay una orden de venta y Mas_Tip [1] nos dice que hay una orden BuyLimit actualmente. Los demás índices que tienen el valor 0, quieren decir que no hay órdenes de su tipo abiertas. Si hay varias ordenes del mimo tipo abiertas en un momento dado, el elemento correspondiente del array tendrá la cantidad equivalente. Por ejemplo si hay 3 órdenes BuyStop, entonces el elemento Mas_Tip [4] tendrá un valor de 3.

La función que maneja la contabilidad con estas arrays la vamos a llamar Terminal() y la incluiremos en un archivo Terminal.mqh .

Función definida por el usario Terminal()

int Terminal()

Esta función lleva la contabilidad de las órdenes abiertas y ordenes pendientes. Cuando se ejecuta esta función, se modifican las siguientes arrays globales.

- Mas_Ord_New – El array con todas las propiedades de las ordenes, en el momento que se ejecuta la función;
- Mas_Ord_Old - El array con todas las propiedades de las ordenes, en la penúltima vez que se ha ejecutado la función;
- Mas_Tip - El array que representa la cantidad de órdenes de cada tipo.

El siguiente es el archivo Terminal.mqh que contiene la función Terminal();

```
//------------------------------------------------------------------
```

```
// Terminal.mqh
// Este código solo debe ser usado de manera educacional.
//------------------------------------------------------------------------------- 1 --
// Función que maneja la contabilidad de las ordenes
// Variables globales:
// Mas_Ord_New[31][9]   // Ultimo registro de las ordenes abiertas
// Mas_Ord_Old[31][9]   // Penúltimo registro de las ordenes abiertas
                        // 1st índice = Numero de la orden en el array
                        // [][0] Campo vacio
                        // [][1] Precio de apertura de la orden
                        // [][2] StopLoss de la orden
                        // [][3] TakeProfit de la orden
                        // [][4] Numero de la orden
                        // [][5] Numero de lotes de la orden
                        // [][6] Tipo de orden
                        // [][7] Número mágico de la orden
                        // [][8] 0/1 Existencia de comentario
// Mas_Tip[6]           // Array que representa la cantidad de órdenes de cada tipo
                        // [] order type: 0=B,1=S,2=BL,3=SL,4=BS,5=SS
//------------------------------------------------------------------------------- 2 --
int Terminal()
  {
   int Qnt=0;                       // Contador de ordenes
//------------------------------------------------------------------------------- 3 --
   ArrayCopy(Mas_Ord_Old, Mas_Ord_New);// Salvamos la información de la ultima actualización
   Qnt=0;                           // Reiniciando contador de ordenes
   ArrayInitialize(Mas_Ord_New,0);     // Reiniciando la array
   ArrayInitialize(Mas_Tip,    0);     // Reiniciando la array
```

```
//------------------------------------------------------------------------------ 4 --
  for(int i=0; i<OrdersTotal(); i++) // Ciclo que revisa todas las ordenes
 {
    if((OrderSelect(i,SELECT_BY_POS)==true)     //Si selecciona una orden..
    && (OrderSymbol()==Symbol()))               //.. y la divisa es la correcta, se continua
     {
      //------------------------------------------------------------------- 5 --
      Qnt++;                                // Se suma un digito al contador
      Mas_Ord_New[Qnt][1]=OrderOpenPrice();    // Copia el orden de apertura
      Mas_Ord_New[Qnt][2]=OrderStopLoss();     // Copia el precio de StopLoss
      Mas_Ord_New[Qnt][3]=OrderTakeProfit();   // Copia el precio TakeProfit
      Mas_Ord_New[Qnt][4]=OrderTicket();       // Copia el numero de la orden
      Mas_Ord_New[Qnt][5]=OrderLots();         // Copia el numero de lotes
      Mas_Tip[OrderType()]++;                 // Lleva la cantidad de orden de ese tipo
      Mas_Ord_New[Qnt][6]=OrderType();         // Copia el tipo de la orden
      Mas_Ord_New[Qnt][7]=OrderMagicNumber();  // Copia el numero magico
      if (OrderComment()=="")
        Mas_Ord_New[Qnt][8]=0;                // Si no hay comentario
      else
        Mas_Ord_New[Qnt][8]=1;                // Si hay comentario
      //------------------------------------------------------------------- 6 --
     }
   }
  Mas_Ord_New[0][0]=Qnt;                     // Registra el total de ordenes
//------------------------------------------------------------------------------ 7 --
  return;
 }
//------------------------------------------------------------------------------ 8 –
```

En el bloque 1 – 2 colocamos un comentario que describe la función. Las variables se declaran en el archivo externo llamado Variables.mqh. En el bloque 3 – 4, el contenido del array Mas_Ord_New se copia al array Mas_Ord_Old. De esta manera podremos utilizar esta información más adelante en el programa. Después los datos Mas_Ord_New y Mas_Tip que llevaban la información de las órdenes se reinician, antes que se actualicen con nuevos datos en el bloque 4 - 7.

El bloque 4 – 7 contiene un ciclo "for", en el cual todas las ordenes y las ordenes pendientes, se revisan una por una, para saber si hay órdenes de la misma divisa de la grafica donde se coloca el EA. Las ordenes se seleccionan usando la función OrderSelect() según el parámetro MODE_TRADES que esta como predeterminado. En el bloque 5 – 6, todas las propiedades de las ordenes seleccionadas se guardan en el array Mas_Ord_New. Al mismo tiempo, el tipo de pedidos y su cantidad se almacenan en el array Mas_Tip. Después que termina el ciclo, se guarda en el elemento Mas_Ord_New [0] [0] la cantidad de órdenes totales.

Debe observarse que las ordenes cerradas y suprimidas no son tenidas en cuenta (en la función OrderSelect() no se utiliza el parámetro MODE_HISTORY). En general, la información sobre las órdenes cerradas y eliminadas no se utiliza en un EA. La información sobre las órdenes cerradas y suprimidas representa el historial de una cuenta. Esta información puede ser útil, por ejemplo, para construir gráficos que muestren el crecimiento de una cuenta en un periodo de tiempo dado. Sin embargo, no nos serán útiles a la hora de tomar decisiones para abrir o cerrar cuentas. Técnicamente la forma de tratar estos datos es similar. Sin embargo, es una tarea separada, que no tienen ninguna relación con el mercado actual.

Los eventos relacionados con las órdenes se analizaran en un programa que se basa en los datos tratados más arriba. Por ejemplo, si el array Mas_Ord_Old contiene una orden con el numero 246810, y en cambio, el array mas reciente Mas_Ord_New, contiene una orden con el mismo numero 246810 pero de un tipo diferente de orden, esto puede significar que una orden pendiente se ha ejecutado y se ha convertido en una orden abierta. También va ser útil a la hora de crear ordenes (esto se verá más adelante).

Cuando la función Terminal() se ejecuta por primera vez, las arrays Mas_Ord_Old y Mas_Ord_New están vacías, es decir, las dos arrays tienen valor cero. Esto significa, que después de la primera ejecución el array Mas_Ord_Old en la línea:

```
ArrayCopy(Mas_Ord_Old, Mas_Ord_New);// Salvamos la información de la última actualización
```

copia el valor "cero" que está en el array Mas_Ord_New, dando como resultado falsas alarmas, en la función que registra los eventos que veremos más adelante. Para prevenir esto, la primera ejecución de la función Terminal() se realiza en la inicialización, y la función que procesa los eventos no (véase la función init() en el EA.

Función para mostrar datos

Un consejero experto normal (EA) que se usa en la práctica, es mucho más potente que un EA simple, porque proporciona información de alta calidad.

El mercado siempre está cambiando y a todo momento suceden cosas. Para que el operador tome decisiones acertadas, es crucial que esté completamente informado sobre la situación del mercado. Por esta razón, un EA debe tener una función que informe al operador. Esta función debe estar destinada a informar al usuario sobre un determinado conjunto de hechos y procesos.

En general, un EA simple cumple esta labor utilizando la función Comment() que coloca un texto informativo en la parte superior izquierda de la grafica. Este método para mostrar información no es muy cómodo, puesto que en algunas ocasiones, el texto puede tapar la grafica del precio. Este método se puede utilizar tan solo en una cantidad limitada de casos, y solo para mostrar información corta.

Veamos una forma diferente de mostrar información. El mensaje se mostrara en una sudventana diferente a la del precio, y esta información se mostrara con objetos gráficos. Usar objetos gráficos tiene la gran ventaja que estos se pueden mover con la grafica, (a diferencia de la función Comment()) creando así una especie de línea de tiempo.

Para crear una subventana, hay que crear un indicador personalizado. El único propósito de este indicador es la creación de la sub-ventana, y no realizara ningún tipo de cálculo, ni tampoco dibujara líneas. El código de este indicador, al que vamos a llamar Inform.mq4 se muestra a continuación.

```
//------------------------------------------------------------------

// Inform.mq4

// Este código solo debe ser usado de manera educacional.

//------------------------------------------------------------------

#property indicator_separate_window // Indicador separado de la ventana

//------------------------------------------------------------------

int start() // función especial start()

{

}

//------------------------------------------------------------------
```

En general, un programador puede colocar en el indicador el código que desee, y mejorar sus características. Por ejemplo usted puede mostrar unas líneas en una parte de la subventana. En el ejemplo anterior se muestra un código simple en que se crea una nueva subventana.

Función definida por el usuario Inform()

```
int Inform(int Mess_Number,int Number=0,double Value=0.0)
```

Esta función mostrara los objetos gráficos descritos más arriba en el indicador que vimos antes Inform.mq4. Esta función controla la posición de los objetos gráficos que se mostraran en la subventana: cada mensaje nuevo, se colocara en una línea más abajo del indicador y se coloreara con el color deseado, y las líneas

anteriores se moverán a la parte superior del indicador. Si no hay actualizaciones en los datos en 15 segundos, los textos cambiaran su color a gris, para no crearle distracciones al usuario.

Parámetros:

Mess_Number - número del mensaje que puede tomar los siguientes valores:

- (cero) 0 - no se muestra ningún mensaje. Este valor se utilizar para reiniciar el contador del tiempo;
- (menos uno) -1 - todos los objetos gráficos creados por la función serán suprimidos;
- (uno o mayor) – Si es un número mayor que cero, este número será tomado como el numero del mensaje que se mostrara en la subventana del indicador;

Number - número entero que se usara en algunos mensajes;

Value - número real que se usara en algunos mensajes;

La función Inform() que es la que crea los objetos gráficos, la vamos a colocar en un archivo separado, como en las otras funciones en un EA normal y luego la vamos a incluir en nuestro indicador. Este archivo se llamara Inform.mqh:

```
//---------------------------------------------------------------------------
// Inform.mqh
// Este código solo debe ser usado de manera educacional.
//------------------------------------------------------------------- 1 --
// Función que muestra mensajes en objetos gráficos en la pantalla.
//------------------------------------------------------------------- 2 --
int Inform(int Mess_Number, int Number=0, double Value=0.0)
{
// int Mess_Number // Numero del mensaje
// int Number // Numero entero
// double Value // Numero real
int Win_ind; // Número de la ventana del indicador
string Graf_Text; // Texto del mensaje
color Color_GT; // Color del texto del mensaje
static int Time_Mess; // Ultima fecha de publicación del mensaje
static int Nom_Mess_Graf; // Numero de mensajes gráficos
static string Name_Grf_Txt[30]; // Array con los textos de mensajes publicados
//------------------------------------------------------------------- 3 --
Win_ind= WindowFind("inform"); // Buscando el numero de la ventana
if (Win_ind<0)return; // Si no hay tal ventana, salir
//------------------------------------------------------------------- 4 --
if (Mess_Number==0) // Esto sucede en cada tick
{
if (Time_Mess==0) return; // Si ya es gris sale
if (GetTickCount()-Time_Mess>15000)// El color no ha sido actualizado en el plazo de 15 seg
{
for(int i=0;i<=29; i++) // Líneas de color gris
ObjectSet( Name_Grf_Txt[i], OBJPROP_COLOR, Gray);
Time_Mess=0; // Bandera  que indica que todas las líneas son de color gris
WindowRedraw(); // Redibujo de los objetos
}
return; // Salimos de la función
}
//------------------------------------------------------------------- 5 --
if (Mess_Number==-1) // Esto se cumple en deinit()
{
for(i=0; i<=29; i++) // El índice de los objetos
ObjectDelete(Name_Grf_Txt[i]);// Eliminación de los objetos
return; // Salimos de la función
}
//------------------------------------------------------------------- 6 --
Nom_Mess_Graf++; // Contador de mensajes
Time_Mess=GetTickCount(); // La fecha del último mensaje
Color_GT=Lime;
//------------------------------------------------------------------- 7 --
switch(Mess_Number) // Elegir texto del mensaje según el numero
{
case 1:
```

```
Graf_Text="Closed order Buy "+ Number;
PlaySound("Close_order.wav"); break;
case 2:
Graf_Text="Closed order Sell "+ Number;
PlaySound("Close_order.wav"); break;
case 3:
Graf_Text="Deleted pending order "+ Number;
PlaySound("Close_order.wav"); break;
case 4:
Graf_Text="Opened order Buy "+ Number;
PlaySound("Ok.wav"); break;
case 5:
Graf_Text="Opened order Sell "+ Number;
PlaySound("Ok.wav"); break;
case 6:
Graf_Text="Placed pending order "+ Number;
PlaySound("Ok.wav"); break;
case 7:
Graf_Text="Order "+Number+" modified into the market one";
PlaySound("Transform.wav"); break;
case 8:
Graf_Text="Reopened order "+ Number; break;
PlaySound("Bulk.wav");
case 9:
Graf_Text="Partly closed order "+ Number;
PlaySound("Close_order.wav"); break;
case 10:
Graf_Text="New minimum distance: "+ Number;
PlaySound("Inform.wav"); break;
case 11:
Graf_Text=" Not enough money for "+
DoubleToStr(Value,2) + " lots";
Color_GT=Red;
PlaySound("Oops.wav"); break;
case 12:
Graf_Text="Trying to close order "+ Number;
PlaySound("expert.wav"); break;
case 13:
if (Number>0)
Graf_Text="Trying to open order Sell..";
else
Graf_Text="Trying to open order Buy..";
PlaySound("expert.wav"); break;
case 14:
Graf_Text="Invalid password. EA doesn't function.";
Color_GT=Red;
PlaySound("Oops.wav"); break;
case 15:
switch(Number) // Elegir texto del mensaje según el numero del error
{
case 2: Graf_Text="Common error."; break;
case 129: Graf_Text="Wrong price. "; break;
case 135: Graf_Text="Price changed. "; break;
case 136: Graf_Text="No prices. Awaiting a new tick.."; break;
case 146: Graf_Text="Trading subsystem is busy"; break;
case 5 : Graf_Text="Old version of the terminal."; break;
case 64: Graf_Text="Account is blocked."; break;
case 133: Graf_Text="Trading is prohibited"; break;
default: Graf_Text="Occurred error " + Number;//otros errores
}
Color_GT=Red;
PlaySound("Error.wav"); break;
case 16:
Graf_Text="Expert Advisor works only for EURUSD";
Color_GT=Red;
PlaySound("Oops.wav"); break;
default:
Graf_Text="default "+ Mess_Number;
Color_GT=Red;
PlaySound("Bzrrr.wav");
}
//------------------------------------------------------------------------- 8 --
ObjectDelete(Name_Grf_Txt[29]); // Eliminación del objeto  29 en el array
for(i=29; i>=1; i--) // Ciclo en el índice del array...
```

```
{ // .. en los objetos gráficos
Name_Grf_Txt[i]=Name_Grf_Txt[i-1];// Aumento de los objetos
ObjectSet( Name_Grf_Txt[i], OBJPROP_YDISTANCE, 2+15*i);
}
Name_Grf_Txt[0]="Inform_"+Nom_Mess_Graf+"_"+Symbol(); // Object name
ObjectCreate (Name_Grf_Txt[0],OBJ_LABEL, Win_ind,0,0);// Creación
ObjectSet (Name_Grf_Txt[0],OBJPROP_CORNER, 3 ); // Esquina
ObjectSet (Name_Grf_Txt[0],OBJPROP_XDISTANCE, 450);// Eje  X
ObjectSet (Name_Grf_Txt[0],OBJPROP_YDISTANCE, 2); // Eje Y
// Текстовое описание объекта??
ObjectSetText(Name_Grf_Txt[0],Graf_Text,10,"Courier New",Color_GT);
WindowRedraw(); // Se redibujan todos los objetos.
return;
}
//---------------------------------------------------------------------- 9 --
```

En el bloque 2 – 3 se declaran y describen las variables que se van a utilizar en la función. Para guardar los nombres de los objetos, utilizaremos el array llamado Name_Grf_Txt. Según el parámetro colocado en la función, se crea un nuevo objeto grafico para mostrar un mensaje. La cantidad total de objetos es 30 y cada objeto representa una entrada de texto escrita en una línea. En caso de tener una resolución de pantalla grande, la cantidad máxima de objetos mostrados puede aumentar.

En el bloque 3 – 4 se verifica que la ventana del indicador sea la correcta. Si no es la correcta, no se muestra ningún objeto grafico. Si no se ha agregado la subventana, el resto del EA funcionara correctamente, y las operaciones no serán afectadas.

En el bloque 4 – 5, se analizan los colores de los menajes. Si a la función se le manda el parámetro Mess_Number=0 quiere decir que no hay mensajes. Si todos los objetos son grises ya, no sucede nada. Pero si el valor de Time_Mess es diferente a cero, el color de todos los objetos cambia a gris.

Si (bloque 5 – 6) el parámetro de Mess_Number es -1, todos los objetos creados anteriormente se borran. Esto puede ser necesario cuando se quita al EA de la grafica. En este caso, es común que cada programa que creo objetos gráficos, también los borre cuando se cierra el programa en la función especial deinit().

Si el programa llega hasta el bloque 6 – 7, significa que es necesario crear un nuevo objeto gráfico con las características requeridas y ponerlo en la parte más inferior del subwindow del indicador ("en la línea más baja"; aquí, el término "línea" es condicional. De hecho, la localización de objetos gráficos es determinada por las coordenadas que le demos). Cada objeto recién creado tiene un nombre único. Para darle el nombre a cada objeto, utilizamos el número histórico del mensaje, esta es la razón por la cual colocamos los nombres al revés en el bloque 6 – 7 (posteriormente el valor de la variable Nom_Mess_Graf se utilizara para formar un único nombre en bloque 8-9). Es acá donde se lleva la fecha de la última publicación de un mensaje y a los mensajes nuevos se le coloca el color verde.

El bloque 7 – 8 consiste en un operador "switch" que depende del valor del mensaje que está en Mess_Number. Cada "caso" de este "switch", representa un mensaje a mostrar, y este mensaje será copiado a la variable Graf_Text, mostrando así un mensaje apropiado según el caso. En algunos casos también se cambia el color del texto, por ejemplo, rojo para los mensajes importantes. Todos los mensajes se acompañan con un sonido característico, que se reproducen con la función PlaySound () (ver archivos de sonido). **LINK.**

La creación de un nuevo objeto gráfico y el reemplazo de los existentes se realiza en el bloque 8-9. La cantidad máxima de objetos gráficos es limitada, así que un objeto (el más viejo) se elimina cada vez que se publica un nuevo mensaje. El resto de los objetos se mueve una línea hacia arriba. Los objetos se mueven cambiando la propiedad que representa las coordenadas horizontales.

Después de que se han hecho todos los cálculos (todos los objetos se han movido una línea hacia arriba), un nuevo objeto con un nombre único y con las propiedades determinadas en el bloque 7- 8 se crea. El tipo de grafico es etiqueta de texto. Este tipo de permite al usuario mover el gráfico de precios de un lado a otro, sin que ello afecte a las posiciones de los mensajes.

La función Inform() puede ser llamada desde cualquier lugar del programa en que se necesite mostrar un mensaje de texto. Como resultado de un periodo largo de funcionamiento, los mensajes se acumularan en la ventana. El usuario puede ver el historial de los mensajes cuando se aumenta el tamaño de la subventana donde está el indicador. Opcionalmente se puede fijar el tamaño de la ventana para ver las líneas de texto que se necesiten (se recomienda de 3 a 4 líneas) .



Es fácil ver que se puede agregar nuevos tipos de mensajes para mostrar. Solo con agregar un nuevo caso o "case" en el operador "switch" en el bloque 7-8.

Función para el seguimiento de eventos

Muchos eventos ocurren cuando se está operando. Un operador puede ver algunos de ellos directamente en la grafica, por ejemplo, cuando cambia el precio o cuando se cruzan unas líneas de un indicador. Otros eventos, aunque son importantes para el operador, no se ven explícitamente en la grafica. Una gran parte de esos eventos se pueden detectar usando MQL4.

Por ejemplo, el bróker puede cambiar las condiciones comerciales poco antes de que se publique una noticia fundamental o cuando el mercado está muy activo. En tales casos, la extensión o la distancia permitida mínima para colocar una orden y para los StopLoss puede aumentarse. Si sucede esto, es necesario, primero detectar las nuevas condiciones y tenerlas en consideración, y en segundo caso, informar al operador sobre estos cambios.

Para solucionar este problema, usted puede utilizar la siguiente función que maneja este tipo de eventos en su asesor experto o EA.

Función definida por el usuario Events()

int Events()

Esta función calcula los cambios en la distancia mínima necesaria para crear una orden y para su StopLoss, así como los cambios en la lista (array) de las órdenes abiertas y ordenes pendientes que tiene la cuenta. Para utilizar esta función, se debe utilizar la función Terminal() que vimos anteriormente en otro programa. Esta función utiliza las siguientes arrays globales.

- Mas_Ord_New - Array que lleva toda la información de las posiciones abiertas y órdenes pendientes. Este array contiene toda la información de cada posición, actualizada desde la última vez de la ejecución de la función Terminal().
- Mas_Ord_Old. - Array idéntico al anterior con la salvedad de que va a guardar la información desde la penúltima vez que se ha ejecutado la función Terminal().

También se utiliza las siguientes variables globales.

- Level_new - el valor actual de la distancia mínima;
- Level_old - el valor anterior de la distancia mínima.

Para mostrar los mensajes, la función utilizará a la función de datos Inform(). Si la función Inform() no se incluye en el Asesor de expertos, los mensajes no se verán.

La función que maneja los eventos la vamos a colocar en un archivo externo Events.mqh y luego la incluiremos en nuestro proyecto.

```
//------------------------------------------------------------------------------

// Events.mqh

// Este código solo debe ser usado de manera educacional.

//------------------------------------------------------------------------ 1 --

// Función para seguir eventos.

// Variables globales:

// Level_new          El ultimo valor de la distancia mínima
```

```
// Level_old           El penúltimo valor de la distancia mínima
// Mas_Ord_New[31][9]   Array con el listado de las ultimas ordenes
// Mas_Ord_Old[31][9]   Array con el listado de las penúltimas ordenes
//------------------------------------------------------------------------- 2 --
int Events()                          // Función definida por el usuario
  {
   bool Conc_Nom_Ord;                  // Coincidencia de las ordenes  ..
   //.. en las arrays que llevan las ordenes
//------------------------------------------------------------------------- 3 --
   Level_new=MarketInfo(Symbol(),MODE_STOPLEVEL );// Ultima distancia conocida
   if (Level_old!=Level_new)           // Si el ultimo no es igual al penúltimo ..
     {                                 // Significa que han cambiado las condiciones
      Level_old=Level_new;             // Antiguo valor es ahora en nuevo valor
      Inform(10,Level_new);            // Se envía un mensaje con el informe
     }
//------------------------------------------------------------------------- 4 --
   // Busca ordenes perdidas, cambio den el tipo de orden, partes cerradas y ordenes reabiertas
   for(int old=1;old<=Mas_Ord_Old[0][0];old++)// En el array de las penúltimas ordenes
     {                                 // Suponiendo que..
      Conc_Nom_Ord=false;               // ..las ordenes no coinciden
      //---------------------------------------------------------------- 5 --
      for(int new=1;new<=Mas_Ord_New[0][0];new++)//Ciclo en el array de las...
        {                               //.. ultimas ordenes
         //------------------------------------------------------------ 6 --
         if (Mas_Ord_Old[old][4]==Mas_Ord_New[new][4])// Si coinciden en el numero
           {                            // El tipo de la orden se convierte en...
            if (Mas_Ord_New[new][6]!=Mas_Ord_Old[old][6])//…diferente
               Inform(7,Mas_Ord_New[new][4]);// Mensaje: modificación:)
```

```
      Conc_Nom_Ord=true;            // Las arrays de ordenes coinciden
      break;                        // Se sale del ciclo actual
     }
   //---------------------------------------------------------------- 7 --
                                 // El numero de la orden no coincide
   if (Mas_Ord_Old[old][7]>0 &&     // coincide el numero magico
      Mas_Ord_Old[old][7]==Mas_Ord_New[new][7])//.. con el antiguo
     {              //significa que se a reabierto o se a cerrado parcialmente
                                  // Si los volúmenes coinciden...
      if (Mas_Ord_Old[old][5]==Mas_Ord_New[new][5])
         Inform(8,Mas_Ord_Old[old][4]);// …es que se ha reabierto
      else                          // En caso contrario..
         Inform(9,Mas_Ord_Old[old][4]);// …se ha cerrado parcialmente
      Conc_Nom_Ord=true;              // Las arrays de ordenes coinciden...
      break;                         // Se sale del ciclo actual
     }
  }
 //---------------------------------------------------------------- 8 --
 if (Conc_Nom_Ord==false)              // Si estamos aquí...
  {                                // ...Esto significa que no encontró orden:(
   if (Mas_Ord_Old[old][6]==0)
      Inform(1, Mas_Ord_Old[old][4]);  // Orden de compra cerrada
   if (Mas_Ord_Old[old][6]==1)
      Inform(2, Mas_Ord_Old[old][4]);  // Orden de venta cerrada
   if (Mas_Ord_Old[old][6]> 1)
      Inform(3, Mas_Ord_Old[old][4]);  // Orden pendiente eliminada
  }
}
```

```
//------------------------------------------------------------------------- 9 --
   // Buscando en las ultimas ordenes
   for(new=1; new<=Mas_Ord_New[0][0]; new++)// En el array de ultimas ordenes
     {
      if (Mas_Ord_New[new][8]>0)           //Esto no es nuevo pero se ha reabierto
         continue;                         //…o parcialmente cerrado
      Conc_Nom_Ord=false;                  // Las arrays de ordenes no coinciden...
      for(old=1; old<=Mas_Ord_Old[0][0]; old++)// Buscando las ordenes en…
        {                                  // ..el array de las penúltimas ordenes
         if (Mas_Ord_New[new][4]==Mas_Ord_Old[old][4])//Numero coincide..
           {                               //.. de la orden
            Conc_Nom_Ord=true;             // La orden es encontrada...
            break;                         // Se sale del ciclo actual
           }
        }
      if (Conc_Nom_Ord==false)             // Si no se encuentra coincidencia...
        {                                  // ..la orden es nueva:)
         if (Mas_Ord_New[new][6]==0)
            Inform(4, Mas_Ord_New[new][4]); // Orden de compra abierta
         if (Mas_Ord_New[new][6]==1)
            Inform(5, Mas_Ord_New[new][4]); // Orden de venta abierta
         if (Mas_Ord_New[new][6]> 1)
            Inform(6, Mas_Ord_New[new][4]); // Orden pendiente colocada
        }
     }
//------------------------------------------------------------------------- 10 --
   return;
  }
```

//---------- 11 –

Las arrays y variables globales que se utilizaran, son explicadas en el bloque 1 – 2. En el bloque 2 – 3 se declara la variable Conc_Nom_Ord que se usara más adelante para analizar las órdenes.

La función realiza un seguimiento a los cambios de la distancia mínima para colocar ordenes y StopLoss. Para ello, el valor actual de la distancia mínima Level_new se calcula en cada ejecución de la función (bloque 3 – 4) y, a continuación se compara con el valor anterior que está en Level_old (obtenido durante la anterior ejecución del a función). Si los valores de estas variables no son iguales, significa que la distancia mínima ha sido cambiada por el broker, un poco antes que se ejecutara la función. En este caso, el valor actual de la distancia mínima se asigna a la variable Level_old(se utilizara este valor en la próxima vez que se ejecute la función) y la función Inform() se ejecuta para mostrar el mensaje correspondiente.

En general, se puede utilizar un método similar a este para detectar otros eventos, por ejemplo, cambios en el spread, el cambio de permisos para negociar una divisa (identificador MODE_TRADEALLOWED en la función MarketInfo()), el fin de una nueva barra (véase el problema 27 **LINK**), el cruce de dos líneas de un indicador (véase fig .107 **LINK**), el llevar a cierta hora, etc. El programa puede detectar algunos eventos utilizando algunos valores del EA para informar al usuario de esto.

En el bloque 4 – 10 se analiza las órdenes y ordenes pendientes. Se informa al usuario de la mayoría de cambios en una orden. Este análisis se realiza en dos etapas. En la primera fase, el programa detecta cambios como la eliminación y el cierre, cambio de tipo de orden y cerrar cierta parte de una orden o reabrir una orden. En la segunda fase (bloque 9 – 10) se buscan las nuevas órdenes.

En los bloques 4 – 9 se analizan las órdenes que están en el array Mas_Ord_Old. La cantidad de iteraciones en el ciclo externo "for" depende de la cantidad total de ordenes en el array (elemento Mas_Ord_Old[0][0]) . Para saber si una orden no ha sufrido cambios, es necesario encontrar una orden igual en el array Mas_Ord_New. Esta búsqueda se realiza en el ciclo interno "for" (bloque 6-8), cuya cantidad de iteraciones es igual a la cantidad de ordenes en el array (elemento Mas_Ord_New[0][0]). Más adelante llamaremos a la array Mas_Ord_New "array nueva" y ha Mas_Ord_Old "array vieja".

En los bloques 6 – 8, el programa busca ordenes, cuyas características sean diferentes. Por ejemplo, en el bloque 6 – 7, se verifica el número de la orden (véase la correspondencia de los índices del array con las características de la orden en la tabla 4 **LINK**). Si el número de una orden en la array vieja coincide con el número de una orden de la array nueva, significa que por lo menos, esta orden no ha sido cerrada o suprimida. Es necesario también verificar si el tipo de la orden no ha sido cambiado. Si ha sido cambiado, significa que una orden pendiente se ha convertido en una orden abierta. En este caso, el mensaje correspondiente se a mostrado usando la función Inform(). Independientemente de que se haya cambiado (o se mantenga sin cambios) el tipo de orden, esta orden no será analizada más: el programa sale del ciclo interno y por último, comienza una iteración nueva del ciclo externo.

Si en el bloque 6 -7 el programa ve que el numero de una orden en la array vieja no coincide con ninguno de los números de ordenes de la array nueva, el programa salta al bloque 7 - 8. Aquí el programa revisa si el número mágico de la orden actual de la array nueva es diferente a cero (todas las ordenes abiertas y pendientes hechas por este EA tienen un numero mágico distinto a cero). Si tiene un número mágico y este coincide con el número mágico de una orden en el array viejo, significa que esta orden esta activa, pero ha cambiado de alguna forma. Hay dos situaciones posibles en que el número de la orden puede haber cambiado.

Situación 1. Una parte de la orden se ha cerrado. Se puede cerrar una parte de una orden (no pendiente!) en dos etapas según la tecnología aceptada en MT 4. En la primera etapa, la orden inicial es totalmente cerrada. Al mismo tiempo, una orden nueva con un volumen más pequeño se abre al mismo precio y con los mismos precios de StopLoss de la orden inicial. Esta nueva orden tiene un nombre único, distinto al número de la orden inicial.

Situación 2. La orden ha sido reabierta por el bróker. Algunos bróker (debido a sus normas internas de contabilidad) cierran a la fuerza todas las ordenes al final del día he inmediatamente abren ordenes iguales, pero en el pago corriente y menos intercambio. but at the current price and minus swap. Este evento no afecta para nada los resultados económicos de una cuenta. Cada orden abierta no coincidirá con los números de las órdenes cerradas.

Las dos situaciones se diferencian en el volumen de las nuevas órdenes: los volúmenes son diferentes en la primera y no cambian en la segunda. Esta diferencia en el volumen se utiliza en el bloque 7 – 8 para distinguir la razón por la que se ha modificado la orden. En ambos casos se mostrara el mensaje correspondiente ("la orden se ha cerrado en parte" o "la orden ha sido reabierta").

Si el programa no ha detectado coincidencias (bloque 6 – 7) o parecidos (bloque 7 – 8) en la array nueva, que cierren el ciclo interno, significa que la orden que está en la array vieja ha sido cerrada o eliminada. En este caso, el programa salta al bloque 8 – 9, donde se mostrara un mensaje de acuerdo con el tipo de orden. En el ejemplo anterior, se mostraran tres tipos de mensajes: para una orden de compra, para una orden de venta y para una orden pendiente de cualquier tipo. Si se desea, esta parte puede cambiarse o mejorarse, colocando un mensaje para los diferentes tipos de orden pendientes.

En la segunda parte del programa (bloque 9 – 10), se analizan las nuevas órdenes en el array nuevo. Esto se hace para detectar órdenes abiertas y pendientes. En el ciclo externo "for", se revisan las características de todas las órdenes que están en la array nueva. Para identificar las órdenes reabiertas y las cerradas en parte, el programa utiliza una simple característica – la existencia del comentario. Cuando se cierra en parte una orden o se reabre, el servidor del bróker agrega un comentario a la orden donde coloca el número de la orden original. Como en este EA no se agregan comentarios a las órdenes, la existencia de un comentario en una orden indica que la orden no es nueva.

Si una orden no tiene comentario, el programa busca una orden con el mismo número en la array vieja. Si el programa encuentra un numero de orden que este tanto en la array vieja como en la nueva en el ciclo "for" significa que la orden no es nueva. Pero si encuentra un número en la array nueva que no está en la array vieja, significa que se ha abierto una nueva orden o una nueva orden pendiente. En la parte inferior del bloque 9 – 10 se ejecuta la función Inform() para que muestre el tipo de orden que se ha creado.

El uso de la función Events() resulta ser muy útil en la práctica. Después que el programador utilice esta función en un EA, se acostumbrara a utilizarla en sus demás proyectos. Debe observase que las funciones Evenys() y Terminal() están estrechamente relacionadas. Si se desea realizar cambios en estas dos funciones (por ejemplo cambiar los nombres de las variables globales), se deberá hacer los cambios en las dos funciones. Si a causa de la estrategia, se necesita colocar comentarios a las órdenes, la parte de este EA que utiliza los comentarios deberá ser adaptada según sea el caso (bloque 9 – 10).

La cantidad de eventos que maneja la función Events() puede aumentarse sustancialmente. Por ejemplo, si se desea mostrar todos los eventos relacionados con las órdenes, se debe analizar las características de las órdenes, como los cambios en el StopLoss de las órdenes abiertas o pendientes o la forma en que se ha cerrado una orden (si la orden se ha cerrado oponiéndola contra otra orden contraria, o se ha cerrado individualmente) y la razón del cierre de una operación o eliminación de una orden pendiente (Si el precio toco un StopLoss, o se ha cerrado manualmente por el operador, etc…).

Función que calcula el volumen de las órdenes

En la operación real, un operador necesita calcular la cantidad de lotes con los que va abrir una posición. Es muy difícil crear una función universal que cumpla este fin, pues cada estrategia puede utilizar una forma diferente de calcular la cantidad de lotes en una posición. Por ejemplo, algunas estrategias requieren que solo exista una sola orden abierta a la vez, mientras que otras no tienen restricciones en la cantidad de ordenes abiertas al mismo tiempo. También existen estrategias basadas en tener varias órdenes pendientes al mismo tiempo.

Uno de los métodos más comunes para calcular el volumen de una orden (para estrategias que solo permiten una orden a la vez) es el método de inversiones progresivas. Según este método, el costo de cada orden es proporcional al margen libre disponible en el momento de crear la orden. Si una orden se cierra con beneficio, la cantidad permitida de lotes para una nueva orden aumenta. Y si cierra con pérdida, esta cantidad disminuye.

En el siguiente ejemplo, vamos a crear una función llamada Lot() que nos permitirá calcular el número de lotes para una orden nueva, y que nos dará dos opciones.

Opción 1. El usuario establece manualmente la cantidad de lotes para las órdenes.

Opción 2. La cantidad de lotes se calcula de acuerdo con la cantidad de dinero asignado por el usuario. Esta cantidad de dinero asignado se calcula sobre el porcentaje de margen libre.

Función definida por el usuario Lot()

bool Lot()

Esta función calcula la cantidad de lotes para las nuevas órdenes. Como resultado de la ejecución de esta función, el valor de la variable global Lots_New cambia, esta variable representa el número de lotes para una orden en el programa. La función devuelve TRUE, si el margen libre es suficiente para abrir una orden con la cantidad mínima de lotes posibles (en la divisa donde se agrega el EA). De lo contrario devuelve FALSE.

Esta función utilizara las siguientes variables globales:

- Lots – volumen de lotes definidos por el usuario;
- Percent – El porcentaje de margen libre definido por el usuario.

Para mostrar los mensajes, la función utilizara la función Inform(). Si esta no está incluida los mensajes no se verán en pantalla.

El siguiente es el código de la función Lot() al que guardaremos en un archivo llamado Lot.mqh y que luego incluiremos en nuestro proyecto.

```
//+------------------------------------------------------------------------------+

// Lot.mqh

// Este código solo debe ser usado de manera educacional.

//+------------------------------------------------------------------------------+ 1 --

// Función que calcula la cantidad de lotes.

// Variables globales utilizadas:
```

```
// double Lots_New - la cantidad de lotes para nuevas ordenes (calculado)
// double Lots - cantidad de lotes deseados definidos por el usuario.
// int Percent - porcentaje de margen libre definido por el usuario
// Valores devueltos:
// true - si hay dinero suficiente para el volumen mínimo permitido
// false - si no hay dinero suficiente para el volumen mínimo permitido
//---------------------------------------------------------------------------- 2 --
bool Lot() // Función definida por el usuario
{
string Symb =Symbol(); // Divisa
double One_Lot=MarketInfo(Symb,MODE_MARGINREQUIRED);//Costo lote
double Min_Lot=MarketInfo(Symb,MODE_MINLOT);// Cantidad mínima de lotes
double Step =MarketInfo(Symb,MODE_LOTSTEP);//Paso en el cambio de volumen
double Free =AccountFreeMargin(); //Margen libre
//---------------------------------------------------------------------------- 3 --
if (Lots>0) // El volumen es explícitamente fijado..
{ // ..en la comprobación
double Money=Lots*One_Lot; // Costo de orden
if(Money<=AccountFreeMargin()) // Si el margen libre es suficiente..
Lots_New=Lots; // ..se acepta la cantidad de lotes
else // Si el margen libre no es suficiente...
Lots_New=MathFloor(Free/One_Lot/Step)*Step;// Se calculan los lotes
}
//---------------------------------------------------------------------------- 4 --
else // Si no está definido el volumen..
{ // ..se toma el porcentaje
if (Percent > 100) // porcentaje pero incorrecto..
Percent=100; // .. entonces no será mas de 100
```

```
if (Percent==0) // Si se fija 0 ..

Lots_New=Min_Lot; // ..entonces el loto será el mínimo

else // Cantidad de lotes deseados:

Lots_New=MathFloor(Free*Percent/100/One_Lot/Step)*Step;//Calculo

}

//------------------------------------------------------------------------------ 5 --

if (Lots_New < Min_Lot) // Si es mejor a lo permitido ..

Lots_New=Min_Lot; // .. entonces será el mínimo

if (Lots_New*One_Lot > AccountFreeMargin()) // No es suficiente aún..

{ // ..para el lote mínimo :(

Inform(11,0,Min_Lot); // Mensaje..

return(false); // ..y se sale

}

return(true); // salimos de la función

}

//------------------------------------------------------------------------------ 6 –
```

La función tiene un código simple. En el bloque 1-2, se describen las variables globales y los valores devueltos. En el bloque 2-3, los valores de algunas variables se calculan. Para hacer cálculos, se utiliza la siguiente prioridad como configuración: Si el usuario ha fijado una cantidad diferente a cero de lotes, el valor de porcentaje de margen libre no se toma en cuenta. Las variables globales externas que llevan los lotes y el porcentaje se declaran en el archivo Variables.mqh.

En el bloque 3-4, se hacen cálculos para la situación, en que el usuario ha definido un valor diferente a cero en el volumen de lotes, en las variables globales externas. En este caso, el programa hace una comprobación. Si el margen libre es suficiente para abrir una nueva orden con la cantidad definida de lotes, entonces el valor establecido por el usuario va ser asignado a la variable global Lots_New que será utilizada en futuros cálculos. Si el margen libre no es suficiente, entonces la cantidad máxima posible de lotes se calcula para su uso futuro (véase funciones matemáticas **LINK**).

Si el usuario ha definido cero en la cantidad de lotes, el programa salta al bloque 4 – 5. Al mismo tiempo, tomamos en cuenta el porcentaje de margen libre especificado por el usuario en la variable externa Percent. El programa hace una comprobación: si el valor supera cien (porcentaje), el valor 100 se utilizara en los cálculos. Si el usuario ha definido cero en la variable Percent, la cantidad de lotes será equivalente al valor mínimo establecido por el bróker. Todos los valores intermedios (de 1 a 99) de utilizaran como porcentaje en el margen libre.

En el bloque 5 – 6, se hacen las comprobaciones necesarias. Si la cantidad calculada de lotes resulta ser menor que el mínimo permitido (por ejemplo, el valor cero puede obtenerse en el bloque 4-5, si el usuario ha definido un valor muy pequeño en la variable Percent), entonces el valor mínimo será asignado a la variable Lots_new. A continuación, el programa comprueba si hay suficiente dinero libre para abrir una orden con la cantidad de volumen previamente calculada de lotes (puede haber poco dinero en la cuenta). Si el dinero

disponible no es suficiente, el programa muestra un mensaje al usuario, luego se sale de la función devolviendo "false". Sin embargo, si hay suficiente dinero la función devuelve true.

Función que define la estrategia

El éxito de cualquier estrategia depende principalmente de los criterios que se tengan para operar. La función que define estos criterios en la mayoría de los casos es la más importante e imprescindible de todas. Según la estrategia, la función puede devolver valores que correspondan con unos criterios comerciales particulares.

En la mayoría de los casos, se pueden definir los siguientes criterios:

- Criterio para abrir una orden;
- Criterio para cerrar una orden;
- Criterio para cerrar una parte de una orden;
- Criterio para cerrar ordenes opuestas;
- Criterio para modificar el StopLoss de una orden;
- Criterio para colocar una orden pendiente;
- Criterio para cancelar una orden pendiente;
- Criterio para modificar el precio de apertura de una orden pendiente;
- Criterio para modificar el StopLoss de una orden pendiente;

En la mayoría de los casos, la activación de un criterio comercial es exclusiva en relación a otros criterios comerciales. In most cases, the triggering of one trading criterion is exclusive as related to other trading criteria. Por ejemplo, si el criterio para abrir una orden de compra se vuelve importante en un momento dado, esto significará que los criterios usados para cerrar órdenes de compra o abrir órdenes de venta, no pueden ser importantes en ese mismo momento (véase la relación de criterios comerciales **LINK**). Al mismo tiempo, según las normas inherentes de una estrategia dada, algunos criterios se pueden activar simultáneamente. Por ejemplo, los criterios para cerrar una orden de venta y los criterios para modificar una orden pendiente BuyStop, se pueden dar al mismo tiempo.

Una estrategia comercial impone requisitos al contenido y a la tecnología que usa la función que define los criterios comerciales. Una función solo puede devolver un valor. Por lo tanto, si la estrategia comercial de un asesor experto implica utilizar criterios comerciales mutuamente excluyentes, el valor que devuelve la función puede estar asociado a uno de esos criterios. Sin embargo, si la estrategia permite activar varios criterios al mismo tiempo, sus valores deben ser enviados por medio de otras funciones para que sean procesados, utilizando las variables globales para ello.

La estrategia comercial que veremos en el EA más abajo, solo permite la utilización de criterios mutuamente excluyentes. Por esta razón la función que utilizaremos para definir los criterios, llamada Criterion(), se comunicara con otras funciones, por medio del valor que esta devuelva .

Función definida por el usuario Criterion()

int Criterion()

Esta función calcula los criterios comerciales. Puede devolver los siguientes valores:

10 - accionó un criterio comercial para abrir una orden de compra;

20 - accionó un criterio comercial para abrir una orden de venta;

11 - accionó un criterio comercial para cerrar una orden de compra;

21 - accionó un criterio comercial para cerrar una orden de venta;

0 – No hay criterios importantes disponibles;

-1 – la divisa usada no es EURUSD.

La función utiliza las siguientes variables externas:

St_min - el nivel inferior del oscilador estocástico;

St_max - el nivel superior del oscilador estocástico;

Open_Level - el nivel del indicador MACD (para abrir una orden);

Close_Level - el nivel del indicador MACD (para cerrar una orden).

Para mostrar los mensajes, la función utilizara la función Inform(). Si esta no está incluida los mensajes no se verán en pantalla.

El siguiente es el código de la función Criterion() que definirá los criterios comerciales, la que guardaremos en un archivo llamado Criterion.mqh y que luego incluiremos en nuestro proyecto.

```
//---------------------------------------------------------------------------

// Criterion.mqh

// Este código solo debe ser usado de manera educacional.

//------------------------------------------------------------------- 1 --

// Función que calcula los criterios comerciales.

// Valores devueltos:

// 10 – Abriendo compra

// 20 – Abriendo venta

// 11 – Cerrando compra

// 21 – Cerrando venta

// 0  - no hay criterio importante disponible

// -1 – se esta utilizando otra divisa

//------------------------------------------------------------------- 2 --

// Variables externas:

extern int St_min=30;                // Nivel mínimo de estocástico

extern int St_max=70;                // Nivel máximo de estocástico

extern double Open_Level =5;         // MACD nivel para abrir orden (+/-)

extern double Close_Level=4;         // MACD nivel para cerrar orden (+/-)

//------------------------------------------------------------------- 3 --

int Criterion()                      // Función definida por el usuario
```

```
  {
   string Sym="EURUSD";
   if (Sym!=Symbol())                // Si la divisa es diferente a EURUSD
     {
      Inform(16);                    // Mensaje..
      return(-1);                   // .. y salida
     }
   double
   M_0, M_1,                         // Valor PRINCIPAL en vela 0 y 1
   S_0, S_1,                         // Valor SEÑAL en vela 0 y 1
   St_M_0, St_M_1,                   // Valor PRINCIPAL en vela 0 y 1
   St_S_0, St_S_1;                   // Valor SEÑAL en vela 0 y 1
   double Opn=Open_Level*Point;      // Apertura de nivel de MACD (pips)
   double Cls=Close_Level*Point;     // Cierre de nivel de MACD (pips)
//------------------------------------------------------------------- 4 --
   // Parámetros de los indicadores técnicos:
   M_0=iMACD(Sym,PERIOD_H1,12,26,9,PRICE_CLOSE,MODE_MAIN,0); // 0 vela
   M_1=iMACD(Sym,PERIOD_H1,12,26,9,PRICE_CLOSE,MODE_MAIN,1); // 1 vela
   S_0=iMACD(Sym,PERIOD_H1,12,26,9,PRICE_CLOSE,MODE_SIGNAL,0);//0 vela
   S_1=iMACD(Sym,PERIOD_H1,12,26,9,PRICE_CLOSE,MODE_SIGNAL,1);//1 vela
   St_M_0=iStochastic(Sym,PERIOD_M15,5,3,3,MODE_SMA,0,MODE_MAIN,  0);
   St_M_1=iStochastic(Sym,PERIOD_M15,5,3,3,MODE_SMA,0,MODE_MAIN,  1);
   St_S_0=iStochastic(Sym,PERIOD_M15,5,3,3,MODE_SMA,0,MODE_SIGNAL,0);
   St_S_1=iStochastic(Sym,PERIOD_M15,5,3,3,MODE_SMA,0,MODE_SIGNAL,1);
//------------------------------------------------------------------- 5 --
   // Calculando el criterio de operación
   if(M_0>S_0 && -M_0>Opn && St_M_0>St_S_0 && St_S_0<St_min)
     return(10);                    // Abriendo compra
```

```
   if(M_0<S_0 &&  M_0>Opn && St_M_0<St_S_0 && St_S_0>St_max)

      return(20);                  // Abriendo venta

   if(M_0<S_0 &&  M_0>Cls && St_M_0<St_S_0 && St_S_0>St_max)

      return(11);                  // Cerrando compra

   if(M_0>S_0 && -M_0>Cls && St_M_0>St_S_0 && St_S_0>St_min)

      return(21);                  // Cerrando venta

//---------------------------------------------------------------------- 6 --

   return(0);                     // Salimos de la función definida por el usuario

  }

//---------------------------------------------------------------------- 7 –
```

En el bloque 1 – 2, se explica los valores devueltos por la función. En el bloque 2 – 3, se declaran algunas variables externas. El archivo incluido Criterion.mqh es el único archivo usado en este EA, en el cual (en este caso externa) se declaran variables globales. En la sección llamada estructura de un programa normal, se pueden encontrar razones para declarar todas las variables globales sin excepción en un archivo separado Variables.mqh. En este caso, las variables externas se declaran en el archivo Criterion.mqh por dos razones: primero, para demostrar que es técnicamente posible (no siempre es posible); y en segundo lugar para mostrar cómo utilizar variables externas en la depuración y las pruebas de un programa.

Es técnicamente posible declarar variables externas en el archivo Criterion.mqh, porque estas variables no se utilizarán en otras funciones del programa. Los valores de las variables externas declaradas en el bloque 2-3 determinan los niveles para el oscilador estocástico y el indicador MACD y se utilizaran solamente en la función Criterion(). La declaración de variables externas en el archivo que contiene la función que define los criterios comerciales puede ser razonable, si el archivo se utiliza temporalmente para la depuración y cálculo de los valores de esas variables en el programa. Con este fin, se puede agregar otras variables externas en el programa, por ejemplo, para optimizar las entradas del indicador (en este caso, las constantes fijadas en los valores 12,26,9 para MACD y 5,3,3 para el oscilador estocástico). Una vez hechas las optimizaciones de las variables, usted puede suprimir estas variables externas del programa y sustituirlas por constantes con los valores calculados en la optimización.

En el bloque 3-4, se abren y describen las variables locales. El EA está pensado para que sea utilizado en la divisa "EURUSD", así que en el bloque 3 – 4 se revisa si esta condición se cumple. Si el EA se pone en marcha en otra divisa, la función termina su ejecución y devuelve el valor -1 (divisa incorrecta).

En el programa, los valores de dos indicadores se calculan en la vela actual y en la anterior (bloque 4-5). Generalmente, cuando se utiliza el oscilador estocástico y MACD, las señales para comprar o vender se dan cuando dos líneas del indicador se encuentran entre sí. En este caso, utilizamos dos indicadores simultáneamente para definir los criterios comerciales. La probabilidad de que se crucen simultáneamente líneas de dos indicadores es algo baja. Es mucho más probable que se crucen una por una – primero en un indicador, y un rato más adelante en otro. Si las líneas de estos dos indicadores se cruzan dentro de un corto periodo de tiempo, se considera que se ha formado un criterio comercial.

Por ejemplo, a continuación se muestra cómo se calcula un criterio comercial para hacer una compra (bloque 5-6):

```
if(M_0 > S_0 && - M_0 > Opn && St_M_0>St_S_0 && St_S_0<St_min)
```

Según el código anterior, el criterio para abrir una orden de compra se forma cuando se cumplen las siguientes condiciones.

- En el indicador MACD, la línea PRINCIPAL (histograma) del indicador está por encima de la línea SEÑAL del indicador y debajo del nivel más bajo Open_Level (Fig. 157);
- En el Oscilador Estocástico, la línea PRINCIPAL (histograma) está por encima de la línea SEÑAL del indicador y debajo del nivel más bajo St_min (Fig.158).



Fig. 157. Condiciones necesarias de las líneas del indicador MACD para confirmar la importancia de los criterios para abrir y cerrar órdenes.

En la parte izquierda de la fig. 157, se muestran las posiciones de las líneas del indicador MACD, en las cuales dos criterios se accionan – abrir orden de compra y cerrar orden de venta. La línea PRINCIPAL del indicador esta debajo del nivel 0.0005 en el transcurso de T1 = t 1 - t 0. Si el oscilador estocástico también da una señal de entrada en este momento, se activara el criterio para abrir una orden de compra. En él transcurso de T2 = t 2 - t 0, la línea PRINCIPAL está debajo del nivel 0.0004. Si el oscilador estocástico también da una señal de salida en este momento, se activara el criterio para cerrar una orden de venta.

Observe por favor que, en T1 se activan dos criterios (si son confirmados por el oscilador estocástico). Antes habíamos mencionado que la función Criterion() devuelve un solo valor, a saber, el valor que se asigna al criterio activado. Durante este periodo, hay que elegir uno de los dos criterios. Este problema se debe solucionar por adelantado en el programa, según las reglas de la estrategia comercial.

En este caso (según la estrategia comercial considerada), la prioridad de abrir una orden de compra es mayor que la de cerrar una orden de venta. Es por esta razón que en el bloque 5 – 6, la línea de código que contiene la orden de compra se coloca por encima de las demás líneas de código. Si durante el período de T1 (fig. 157), tenemos la confirmación del oscilador estocástico, la función devolvería 10, que representa el criterio de compra. Luego dentro del periodo T1 a t2, la función devolverá 21, asignado al criterio para cerrar una venta.

Al mismo tiempo, en la función que maneja las peticiones comerciales, se formará una petición comercial. En primer lugar se accionara el criterio para abrir una orden de compra, en segundo, se accionara el criterio para cerrar todas las órdenes de venta. Tan pronto como no haya ordenes, se solicitara una orden de compra. Respectivamente, cuando el criterio para cerrar las órdenes de venta se accione, se ejecutara una secuencia que cerrará todas las órdenes de venta abiertas. (ver función de comercio LINK).

Las condiciones en que el Oscilador Estocástico da confirmación se muestra en la figura 158.



Fig. 158. Condiciones necesarias de las líneas del Oscilador Estocástico para confirmar la importancia de los criterios para abrir y cerrar órdenes.

Según el código de programa escrito en el bloque 5-6, los criterios para abrir una compra y cerrar una venta se activaran, siempre que la línea PRINCIPAL del indicador, está por encima de la línea SEÑAL en el Oscilador Estocástico, la línea PRINCIPAL está por debajo del nivel mínimo St_min. En la fig. 158, estas condiciones se dan en el periodo TS. Los criterios inversos se dan en el segundo Ts, donde se forman los criterios comerciales para abrir una orden de venta y cerrar una orden de compra.

Es importante señalar que en esta estrategia, el indicador MACD esta funcionando en graficas de 1 hora, mientras que el oscilador estocástico funciona en graficas de 15 minutos. El tiempo de las graficas se puede cambiar mientras se está optimizando y perfeccionando la estrategia. Sin embargo, después de hacer estas pruebas, y tener los valores optimizados en la función Criterion(), es necesario convertir estos valores en constantes para todos los parámetros calculados incluyendo el tiempo de las graficas. El EA debe ser utilizado bajo las condiciones en las que se ha creado. En el ejemplo anterior (con los valores de los periodos PERIOD_H1 y PERIOD_M15 especificados explícitamente en los indicadores), el EA tomara las mismas decisiones independientemente del periodo del grafico actual de la divisa donde está funcionando el EA.

Los criterios comerciales utilizados en este EA están hechos para propósitos de enseñanza y no deben ser utilizados en cuentas reales.

Funciones del Comercio

Como regla general, un Asesor de Expertos normal contiene una serie de funciones comerciales. Se pueden dividir en dos categorías - las funciones de control y las funciones ejecutivas. En la mayoría de los casos, sólo una función de control y varias funciones ejecutivas se utilizan en un EA.

Una estrategia de negociación en un EA normal se realiza sobre la base de dos funciones - una función de definir los criterios de comercio y una función de control del comercio. No debe haber indicios de la estrategia comercial de cualquier otro lugar en el programa. La función de control del comercio y la función de definir los criterios de comercio deben estar coordinados entre sí en los valores de los parámetros que pasan.

Cada función del comercio ejecutivo tiene una gama especial de tareas. De acuerdo con los requisitos de la estrategia comercial, funciones comerciales destinados a las siguientes tareas se pueden utilizar en un EA:

- la apertura de una orden de mercado del tipo predefinido;
- cierre de una orden de mercado del tipo predefinido;
- en parte, una orden de cierre de mercado del tipo predefinido;
- Para cerrar todos los mercados del tipo predefinido;
- el cierre de dos órdenes de mercado frente en el volumen preestablecido;
- cierre de todas las órdenes de mercado;
- modificación de las órdenes de parada de una orden de mercado del tipo predefinido;
- colocación de una orden pendiente del tipo predefinido;
- supresión de una orden pendiente del tipo predefinido;
- supresión de todas las órdenes en espera de que el tipo predefinido;
- supresión de todos los pedidos pendientes;
- la modificación de una orden pendiente del tipo predefinido.

Una secuencia de comercio en general, un Asesor de Expertos normal consiste en lo siguiente: Sobre la base de cálculo (de acuerdo con la estrategia utilizada) los criterios de comercio, la función de control del comercio (también la realización de la estrategia) y llama a algunos el comercio u otras funciones ejecutivas que, en su A su vez, forma las solicitudes de comercio es necesario.

Definido por el usuario Función de Control del Comercio ()

```
  Comercio (int Trad_Oper)
```

Es la función básica que se da cuenta de su estrategia.

Trad_Oper parámetro puede tomar los valores siguientes corresponden con los criterios de negociación:
10 - provocó un criterio comercial para la apertura de una orden de mercado comprador;
20 - provocó un criterio comercial para la apertura de una orden de mercado venden;
11 - provocó un criterio de negociación para el cierre de una orden de mercado comprador;
21 - provocó un criterio de negociación para el cierre del mercado para vender;
0 - No se dispone de criterios importantes;
-1 - El símbolo utilizado no es el EURUSD.

Para ejecutar la función, se requieren las funciones comerciales siguientes:

- Close_All () - Función de cierre de todas las órdenes de mercado del tipo predefinido;
- Open_Ord () - función de apertura de un orden de mercado del tipo predefinido;
- Tral_Stop () - función de la modificación de Stoploss de una orden de mercado del tipo predefinido;
- Lot () - función de detectar la cantidad de lotes a los nuevos pedidos.

La función de control del comercio de Comercio () se formó como archivo de inclusión Trade.mqh:

```
 //---- ---- ---------------------------------------- ------------------------
  / / Trade.mqh
  / / El código debe ser utilizado para fines educativos solamente.
  //-------- ---- ---------------------------------- ------------------------
  / / Función de Comercio.
  //------------ ---- -------------------------------- ------------------- 1 --
```

```
Comercio (int Trad_Oper) / / función definida por el usuario
(
/ / Trad_Oper - tipo de operación comercial:
/ / 10 - la apertura de oferta
/ / 20 - la apertura de Venta
/ / 11 - Cierre Comprar
/ / 21 - Cierre de Venta
/ / 0 - no se dispone de criterios importantes
/ / -1 - Otro símbolo se utiliza
switch (Trad_Oper)
(
//---------- ---- ---------------------------------- ------------- 2 --
case 10: / / Trading criterio = Comprar
Close_All (1); / / Cerrar todo Vender
if (Lot () == false) / / No tengo suficiente dinero para min.
return; / / Salir de la función definida
Open_Ord (0); / / Abrir Comprar
return; / / Luego de intercambiar, dejar
//---------- ---- ---------------------------------- ---------- 3 --
case 11: / / criterio de Trading: cierre de Compra
Close_All (0); / / Cierre todas las Comprar
return; / / Luego de intercambiar, dejar
//--------------- ---- ------------------------------ ---------- 4 --
case 20: / / Trading criterio = Venta
Close_All (0); / / Cierre todas las Comprar
if (Lot () == false)
return; / / Salir de la función definida
Open_Ord (1); / / Abrir Vender
return; / / Luego de intercambiar, dejar
//------------------- ---- -------------------------- ---------- 5 --
case 21: / / criterio de Trading: cierre de Venta
Close_All (1); / / Cerrar todo Vender
return; / / Luego de intercambiar, dejar
//----------------------- ---- ---------------------- ---------- 6 --
case 0: posiciones / / Mantener abierto
Tral_Stop (0); / / trailing stop Comprar
Tral_Stop (1); / / trailing stop Vender
return; / / Luego de intercambiar, dejar
//--------------------------- ---- ----------------- ---------- 7 --
)
)
//------------------------------------------------ -------------------- 8 --
```

El control del comercio de Comercio () es llamado desde el inicio de funciones especiales () del Asesor de Expertos usualexpert.mq4. El valor devuelto por la función de definir los criterios de comercio Criterio () se da como el parámetro que se pasa en la función de Comercio ().

En el bloque 1-2 de la función de Comercio (), se describen los criterios de comercio considerado por la estrategia de negociación se dio cuenta. En la función, se utiliza el cambiar de operador () (bloques 2-7), que nos permite activar el grupo de las funciones necesarias para el comercio de acuerdo con el criterio de negociación. De acuerdo con la estrategia comercial, la EA se abre y cierra las órdenes de mercado. N operaciones con órdenes pendientes están previstas en esta estrategia de negociación.

En la sección denominada Función Definición de Criterios de Comercio, se ha precisado que para algunos criterios de negociación que el programa puede formar varias peticiones del comercio. Así, en caso de criterio importante para la compra (el valor de la Trad_Oper variable es igual a 10), el control se pasa a la marca 'Case 10' (bloque 2-3) durante la ejecución de cambiar de operador (). En este caso, el primer programa las llamadas a la función Close_All (1). La ejecución de esta función de los resultados en el cierre de todas las órdenes de mercado abierto a la venta el símbolo EURUSD. Después de todos los órdenes de venta se han cerrado, el dinero disponible se comprueba para saber si es suficiente para hacer que el comercio que viene. Para ello, el usuario-Lot función definida () es llamado (véase el Tomo Detección de la función). Si esta función devuelve 'false', que significa que el dinero disponible en la cuenta no es suficiente para abrir orden de

compra con la cantidad mínima permitida de los lotes. En este caso, la función de Comercio () termina sus operaciones. Si no hay suficiente dinero, la función del comercio se llama Open_Ord (0) para abrir una orden de mercado comprador con la cantidad de lotes, calculado en la ejecución de la función del lote (). El conjunto de acciones descritas representa la respuesta del Asesor de Expertos a la situación en el mercado (según el criterio de negociar dadas).

Si el criterio es importante que destaca en la necesidad de cerrar las órdenes de mercado comprador, el control se pasa a la marca 'case 11' en el bloque 3-4. En este caso, sólo una función Close_All (0) está llamado a cerrar todos los órdenes del tipo de oferta disponible. Bloques 4-6 se construyen en el modo similar a los bloques 2-4, el control se pasa al caso de las marcas '20' y 'Caso 21', si los criterios para la venta o el cierre de las órdenes de mercado venden se vuelven importantes.

Tenga en cuenta que todas las funciones ejecutivas del comercio que las solicitudes de comercio formulario se denominan en la función de Comercio () que, a su vez, se llama a la ejecución del inicio de la función especial de la EA () lanzada por el cliente de terminal como resultado de una garrapata nuevo entrante. El código de la función de Comercio () está escrito de tal manera que el control no se devuelve al inicio de la función () (y, al final, a la terminal de cliente) hasta que todas las funciones necesarias comercio ejecutivo son ejecutados. Por ello, de todas las operaciones destinadas a cada criterio de comercio son realizadas por la EA, uno por uno, sin pausas. La excepción puede ser los casos de errores graves que ocurren durante la toma de operaciones (véase el error de procesamiento de función).

Si no hay un criterio comercial es detectado como Trad_Oper importante (la variable es igual a 0) en la ejecución de la función de criterio (), el control se pasa a la marca «case 0 ', que da lugar a la doble llamada a la función Tral_Stop () para modificar los valores deseados de las órdenes de mercado de diferentes tipos. La estrategia de negociación realizado en este EA permite la disponibilidad de la orden de un único mercado, por lo que la secuencia de llamadas a la Tral_Stop funciones (0) y Tral_Stop (1), no importa. En este caso, es una elección al azar.

Si la función de criterio () ha devuelto el valor de -1, esto significa que la EA se une a la ventana de un símbolo que no es EURUSD. En este caso, la función de Comercio () no llama a cualquiera de las funciones ejecutivas del comercio y devuelve el control al inicio de la función especial () que ha llamado.

Definido por el usuario Función Ejecutiva Comercio Close_All ()

```
  Close_All (int Consejo)
```

La función cierra todas las órdenes de mercado del tipo determinado.

El parámetro **Consejo** puede tomar los valores correspondientes con los siguientes tipos de órdenes de cierre:
0 - cierre de órdenes de compra;
1 - cierre de órdenes de venta.

Para ejecutar la función, es necesario aplicar la orden de la función de contabilidad de Terminal (), el seguimiento de eventos Eventos function () y el procesamiento de errores de error function () en el programa. Para mostrar los mensajes, la función implica la utilización de los datos de la función Inform (). Si la función de informar () no está incluido en la EA, los mensajes se mostrarán.

Los valores de las matrices se utilizan las siguientes:

- Mas_Ord_New - la matriz de las características de las órdenes disponibles a partir del momento de la función de Terminal () de ejecución;
- Mas_Tip - la matriz de la cantidad total de pedidos de todo tipo a partir del momento de la última ejecución de la función de Terminal Server ().

El ejecutivo de la función del comercio Close_All () se configura como el archivo de inclusión Close_All.mqh:

```
 / / - ------------ ---------------------------------- ----------------------------
-----
  / / Close_All.mqh
  / / El código debe ser utilizado para fines educativos solamente.
  //--------------- ------------ -------------------- ----------------------------
```

```
1 --
  / / Función de cierre de todas las órdenes de mercado del tipo dado
  / / Variables globales:
  / / Mas_Ord_New Última matriz orden conocido
  / / Matriz de tipo Mas_Tip Orden
  //--------------------------- ------------ ------- --------------------------
2 --
  Close_All (int Consejo) / / función definida por el usuario
  (
  / / Consejo int / / Tipo de Orden
  Venta de entradas int i = 0; / / boleta
  Lote doble = 0; / / Cantidad de lotes cerrados
  Price_Cls doble, / / Precio de la Orden cerrar
  //---------------------------------------- ------- ------ ----------------------
3 --
  while (Mas_Tip [Sugerencia]> 0) / / En tanto las órdenes de la ..
  (/ / .. determinado tipo están disponibles
  for (int i = 1; i [0 <= Mas_Ord_New] [0], i + +) / / Ciclo para pedidos en vivo
  (
  if (Mas_Ord_New [i] [6] == Consejo & & / / Entre las órdenes de nuestro tipo de
  Mas_Ord_New [i] [5]> Lot) / / .. seleccionar la más cara
  (/ / Este fue encontrado en la primera.
  = Lote Mas_Ord_New [i] [5]; / / La mayor cantidad de lotes encontrados
  Venta de entradas = Mas_Ord_New [i] [4]; / / Su boleta es que
  )
  )
  if (Consejo == 0) Price_Cls = Oferta; / / Por órdenes de compra
  if (Consejo == 1) Price_Cls = Ask; / / Para las órdenes de venta
  Informar a los (12, Venta de entradas); / / Mensaje sobre un intento de cerrar
  bool ans = OrderClose (Venta de entradas, Lot, Price_Cls, 2); / / fin Close!:)
  //------------------------------------------------ --------- ------------- 4 --
  if (respuesta == false) / / Error: (
  (/ / Comprobar si hay errores:
  if (errores (GetLastError ()) == false) / / Si el error es crítico,
  return; / / .. después se van.
  )
  / / - ------------- ---------------------------------- ---------------------- 5 -
-
  Terminal (); / / función de la contabilidad Orden
  Eventos (); / / Seguimiento de eventos
  )
  return; / / Salir de la función definida
  )
  //-------------------- ------------- --------------- --------------------------
6 --
```

En el bloque 1-2, se describen las variables globales que se utilizan. En el bloque 2-3, las variables locales se abren y se describen. La condición Mas_Tip [Sugerencia]> 0 en el título del ciclo, mientras que el operador '(bloques 3-6) implica que la función tendrá el control hasta que cumpla con su finalidad, es decir, hasta que todas las órdenes del mismo tipo están cerrados . El elemento de la matriz mundial de Mas_Tip [Sugerencia] contiene el valor igual a la cantidad de órdenes de la determinado tipo de Consejo. Por ejemplo, si la función Close_All () es llamado con los parámetros transferidos igual a 1, esto significa que la función debe cerrar todas las órdenes de mercado venden (ver tipos de transporte). En este caso, el valor del elemento de la matriz Mas_Tip [1] será igual a la cantidad de órdenes disponibles Sell (última conocida como del momento de la ejecución de la función de Terminal ()). Así, mientras el operador del ciclo 'se ejecutará tantas veces como muchas órdenes de venta están disponibles.

Si el operador no interviene en las operaciones de la EA (es decir, él o ella no realizar pedidos de forma manual), sólo un orden de mercado de un tipo u otro pueden estar disponibles en el comercio. Sin embargo, si el comerciante tiene, además, coloca una o varias órdenes de mercado por su propia iniciativa, a continuación, una cierta secuencia de órdenes deben mantenerse a la ejecución de la función Close_All (). La secuencia de cierre de las órdenes preferible es cerrar primero los más grandes. Por ejemplo, si hay tres

órdenes de venta a partir del momento de comenzar a ejecutar la función Close_All (), uno de ellos se abrió a 5 lotes, otra se abre por 1 lote, y el tercero, abierto a la 4 lotes, a continuación, los pedidos se cerró en la siguiente secuencia de acuerdo con el razonamiento anterior: la primera orden de cierre será el de 5 lotes, luego el de 4 lotes, y la última será del orden de 1 lote.

Tenga en cuenta que la cantidad de lotes es el único criterio utilizado para determinar la secuencia de cierre de pedidos. Sin fines de lucro de la orden / pérdida, precio de apertura, así como otros parámetros que caracterizan el orden (la parada solicitada para los precios, el tiempo y la razón para el cierre, etc) no son considerados.



Todos los pedidos de mercado de un determinado tipo debe estar cerrado, si el criterio para el cierre de las órdenes de este tipo es importante, la secuencia de cierre de ser de la más grandes a los pequeños volúmenes.

Para mantener la secuencia anterior de cerrar el orden, en el bloque 3-4, el ciclo 'para' se utiliza, en la que el más grande (en volumen) para se selecciona entre todas las órdenes del mismo tipo. Esta orden es buscado en la base del análisis de los valores de la matriz mundial de Mas_Ord_New que contienen la información sobre todas las órdenes disponibles en el comercio. Después de la entrada de esta orden se ha detectado, según el tipo de orden, el precio de cierre necesarios se calcula que es igual al valor correspondiente de la última cita conocida de dos manera. Si las órdenes de cierre son de tipo comprador, el precio de cierre debe ser solicitada sobre la base del valor de la oferta. Si son órdenes de venta, a continuación, utilice Pregunte valores.

Justo antes de formar una solicitud de cesión, se muestra la información sobre el intento de cerrar el pedido. El programa utiliza la llamada a la función Inform () para este propósito. La solicitud de cesión para el cierre de la orden se forma en la línea:

```
bool ans = OrderClose (Venta de entradas, Lot, Price_Cls, 2); / / fin Close!:)
```

Los valores calculados se utilizan como parámetros: - número de entradas orden, Lot - volumen en lotes, Price_Cls - pidió a precio de cierre, 2 - deslizamiento.

En el bloque 4-5, se analizan los resultados del comercio. Si la función OrderClose () ha regresado 'true', esto significa que el comercio se ha completado con éxito, es decir, la orden ha sido cerrada. En este caso, el control se pasa al bloque de 5-6, donde se actualiza la información sobre las órdenes disponibles en el momento actual. Tras la ejecución de las funciones de terminal () y eventos (), la iteración actual del ciclo ", mientras que" los fines (la cantidad de órdenes disponibles puede cambiar en el tiempo de ejecución de la función y durante las operaciones de toma, de modo que la ejecución de la orden es función de la contabilidad obligatoria en cada iteración del ciclo 'while'). Si las órdenes del mismo tipo están todavía disponibles en el comercio, que se cerrará a la siguiente iteración del ciclo 'while', los nuevos valores de los elementos de las matrices Mas_Ord_New y Mas_Tip obtenidos en la ejecución de la función de Terminal () se utilizado para la determinación de los parámetros de la siguiente orden a ser cerrado.

Si la ejecución de los resultados de la solicitud en que la función de OrderClose () devuelve 'false', esto significa que la orden no ha sido cerrada. Con el fin de conocer las razones de este fracaso, el programa analiza el último error producido en el intento de hacer el comercio. A tal fin, pide a los errores de función () (véase el error de procesamiento de función). Si en la ejecución de esta función, el programa detecta que el error es crítica (por ejemplo, el comercio está prohibido), la función Close_All () termina sus operaciones y devuelve el control a la función de control del comercio de Comercio (), que finalmente da lugar a de que la función especial de la EA start90 termina su ejecución, así. En la garrapata que viene, el terminal se lanzará el inicio de la función () para la ejecución de nuevo. Si el criterio de cierre sigue siendo real en ese momento, esto producirá la llamada a la función de cierre de todos los órdenes, Close_All ().

Definido por el usuario Función Ejecutiva Comercio Open_Ord ()

```
  Open_Ord (int Consejo)
```

La función abre una orden de mercado del tipo determinado.

El parámetro **Consejo** puede tomar los valores correspondientes con los siguientes tipos de órdenes que se abrirá:
0 - el tipo de oferta de las órdenes que se abra;
1 - el tipo de órdenes de venta que se abra.

Para ejecutar la función, usted debe utilizar en el programa de la orden de contabilidad de la función de Terminal Server (), el seguimiento de eventos Eventos function () y el procesamiento de errores error de función (). Para mostrar los mensajes, la función implica la función de informar los datos (). Si la función de informar () no está incluido en la EA, los mensajes se mostrarán.

Los valores de las variables globales se utilizan las siguientes:

- Mas_Tip - la matriz de la cantidad total de pedidos de todo tipo a partir del momento de la última ejecución de la función de Terminal Server ();
- Stoploss - el valor de Stoploss (cantidad de puntos);
- TakeProfit - el valor de TakeProfit (cantidad de puntos).

El ejecutivo de la función del comercio Open_Ord () se formó como archivo de inclusión Open_Ord.mqh:

```
 //------------------------------- ------------ -- ----------------------------
----
  / / Open_Ord.mqh
  / / El código debe ser utilizado para fines educativos solamente.
  //------------------------------------------- -- ---------- -----------------
1 --
  / / Función de la apertura de un orden de mercado del mismo tipo
  / / Variables globales:
  / / Int Mas_Tip matriz de tipo Orden
  / / Int Stoploss El valor de Stoploss (cantidad de puntos)
  / / Int TakeProfit El valor de TakeProfit (cantidad de puntos)
  //------------------------------------------- ---------- ------------ ----
2 --
  Open_Ord (int Consejo)
  (
  Venta de entradas int / / boleta
  MN / / / MagicNumber
  doble, SL, / / Stoploss (en relación con el precio)
  TP / / TakeProf (en relación con el precio)
  / / - ------------ ------------------------------ -------------------------
-- 3 --
  while (Mas_Tip [Sugerencia] == 0) / / Hasta que ..
  (/ / .. éxito
  if (Stoploss <Level_new) / / Si es menor que permitió ..
  Stoploss = Level_new; / / .. entonces el permitió una
  if (TakeProfit <Level_new) / / Si es menor que permitió ..
  TakeProfit = Level_new; / / .. entonces el permitió una
  MN = TimeCurrent (); / / Simple MagicNumber
  Informar a los (13, Consejo); / / Mensaje sobre un intento de abrir
  if (Consejo == 0) / / Vamos a abrir una oferta
  (
  SL = Oferta - Stoploss * Punto / / Stoploss (precio)
  TP = Oferta + TakeProfit * Punto / / TakeProfit (precio)
  = Venta de entradas OrderSend (símbolo (), 0, Lots_New, Ask, 2, SL, TP, "", MN);
  )
```

```
  if (Consejo == 1) / / Vamos a abrir una venta
  (
  SL = Oferta + Stoploss * Punto / / Stoploss (precio)
  TP = Ask - TakeProfit * Punto / / TakeProfit (precio)
  = Venta de entradas OrderSend (símbolo (), 1, Lots_New, Oferta, 2, SL, TP, "",
MN);
  )
  //-------- ------------ -------------------------- ---------------------- 4 --
  if (Venta de entradas <0) / / Error: (
  (/ / Comprobar si hay errores:
  if (errores (GetLastError ()) == false) / / Si el error es crítico,
  return; / / .. después se van.
  )
  Terminal (); / / función de la contabilidad Orden
  Eventos (); / / Seguimiento de eventos
  )
  //-------------------------- ------------ -------- --------------------------
5 --
  return; / / Salir de la función definida
  )
  //------------------------------------- -------- ----- ----------------------
6 --
```

En los bloques de 1-3 de la función Open_Ord (), las variables globales se describen los valores de los que se utilizan en la ejecución de la función, y las variables locales se abren y se describe. El código básico de la función se concentra en el ciclo mientras que el operador '(bloques 3-5) que se ejecuta siempre y cuando no las órdenes del Consejo de determinado tipo están disponibles en el comercio.

La estrategia comercial implica la apertura de las órdenes que han de cero órdenes de detenerse. En un caso general, un operador puede fijar dichos valores de las órdenes de detención que no cumplen con los requisitos del centro de la frente, es decir, menos de la distancia mínima permitida de la precio de mercado. Por ello, los controles necesarios se realiza antes de la apertura de un pedido: Si la última distancia mínima conocida (Level_new) excede el valor de la variable externa o Stoploss TakeProfit, el valor de esta variable es mayor y se puso a ser igual a Level_new.

Cada fin de que se abra tiene su MagicNumber única igual a la hora actual del servidor. Como resultado de la ejecución de uno de EA para un símbolo, sólo un orden de mercado pueden ser abiertas (o puesto, si es una orden pendiente) a la vez. Esto proporciona todas las órdenes de mercado con MagicNumbers único. Antes de la apertura de una orden, la Informar a la función () se ejecuta, lo que en mostrar los resultados de un mensaje informando sobre un intento de hacer un intercambio.

Según el tipo de orden, el cuerpo de uno de los operadores 'si' se ejecuta. Por ejemplo, si el valor del parámetro transferido Consejo es igual a 0, esto significa que una orden de compra debe ser abierto. En este caso, los valores de Stoploss TakeProfit y se calcula que se corresponden con el tipo de orden de compra, entonces el control se pasa a la línea de

```
  = Venta de entradas OrderSend (símbolo (), 0, Lots_New, Ask, 2, SL, TP, "", MN);
```

para formar una solicitud de cesión de la apertura de una orden de mercado comprador. Cálculos similares se hacen, si el valor del parámetro Consejo es 1, es decir, una orden de venta debe ser abierta.

Los errores en todas las funciones definidas por el comercio ejecutivo se procesan de manera similar. Si el comercio se realiza con éxito, la función termina sus operaciones (porque no la siguiente iteración del ciclo 'while' se llevará a cabo, ya que el valor del elemento de Mas_Tip array [Tip] será igual a 1 después de la ejecución de la función Terminal ()). Sin embargo, si la solicitud de cesión no se ejecuta, los errores son analizados (bloque 4-5). En este caso, el error de detección de errores function () se llama. Si devuelve 'false' (el error es crítico), la ejecución de la función Open_Ord () termina, el control está consecutivamente pasó a controlar el comercio de la función de Comercio () , al inicio de la función especial () y luego a la terminal del cliente. Sin embargo, si el error es overcomable, entonces, después de la actualización de las matrices de orden en la función de Terminal Server (), el control se pasa a la iteración consecutivos del ciclo ", mientras que ', que se traduce en un intento más para abrir una orden.

Así, la función Open_Ord () mantiene el control hasta que una orden se abre o un error crítico se consigue en la ejecución de la solicitud del comercio.

Definido por el usuario Función Ejecutiva Comercio Tral_Stop ()

```
  Consejo Tral_Stop int (int)
```

La función modifica todos los órdenes de mercado del tipo determinado.

El parámetro **Consejo** puede tomar los valores siguientes corresponden con el tipo de órdenes a ser modificados:
0 - el tipo de oferta de la orden de ser modificado;
1 - el tipo de órdenes de venta que ser modificados.

Para ejecutar la función, es necesario utilizar en el programa de la orden de la función de contabilidad de Terminal (), la función de seguimiento de eventos Eventos (), y el error de los errores de procesamiento function (). Para mostrar los mensajes, la función implica la función de informar los datos (). Si la función de informar () no está incluido en la EA, los mensajes se mostrarán.

Los valores de las variables globales se utilizan las siguientes:

- Mas_Ord_New - la matriz de las características de los pedidos disponible a partir del momento de la última ejecución de la función de Terminal Server ();
- TralingStop - la distancia entre el precio de mercado y el valor deseado del precio solicitado para Stoploss (cantidad de puntos).

El ejecutivo de la función del comercio Tral_Stop () se formó como archivo de inclusión Tral_Stop.mqh:

```
 //--------------------------- ------------- -------- ----------------------------
----
  / / Tral_Stop.mqh
  / / El código debe ser utilizado para fines educativos solamente.
  //-------------- ------------- ---------------------- ----------------------------
1 --
  / / Función que modifica StopLosses de todas las órdenes del mismo tipo
  / / Variables globales:
  / / Mas_Ord_New Última matriz orden conocido
  / / Int TralingStop Valor de TralingStop (cantidad de puntos)
  / / - ------------- ---------------------------------- ----------------------------
-- 2 --
  Consejo Tral_Stop int (int)
  (
  Venta de entradas int / / boleta
  doble
  Precio, / / Precio de mercado abierto para
  TS, / / TralingStop (en relación con el precio)
  SL, / / Valor de la Stoploss orden
  TP / / Relación de orden TakeProfit
  Modificar bool; / / Un criterio de modificar.
  //------------------------------------------------ ----- ------------- ----------
3 --
  for (int i = 1; i [0 <= Mas_Ord_New] [0], i + +) / / Ciclo para todos los pedidos
  (/ / En busca de órdenes del tipo dado
  if (Mas_Ord_New [i] [6]! = Consejo) / / Si este no es nuestro tipo ..
  continuar; / / ..  Saltar el orden
  Modificar = false; / / No se asigna a ser modificado
  Precio = Mas_Ord_New [i] [1]; / / Precio de la Orden abierta
  SL = Mas_Ord_New [i] [2]; / / Valor de la Stoploss orden
  TP = Mas_Ord_New [i] [3] / / Valor de la orden TakeProft
  Venta de entradas = Mas_Ord_New [i] [4]; / / boleta
  if (TralingStop <Level_new) / / Si es menor que permitió ..
  TralingStop = Level_new; / / ..  entonces el permitió una
  TS = TralingStop * Punto; / / Lo mismo en el relat, el valor de precio de
  //--------------------------------- ------------- -- ---------------------- 4 --
```

```
   switch (Consejo) / / Ir a la Orden de tipo
   (
   case 0: / / orden de compra
   if (NormalizeDouble (SL, dígitos) </ / Si es menor de lo que queremos ..
   NormalizeDouble (Oferta - TS, dígitos))
   (/ / .., Después, modificarlo:
   SL = Oferta - TS / / Su nuevo Stoploss
   Modificar = true; / / Asignación de ser modificado.
   )
   break; / / interruptor de salida '
   case 1: / / orden de venta
   if (NormalizeDouble (SL, dígitos)> / / Si es mayor de lo que queremos ..
   NormalizeDouble (Ask + TS, cifras) | |
   NormalizeDouble (SL, dígitos) == 0) / / ..  o cero (!)
   (/ / .., Después, modificarlo
   SL = Oferta + TS / / Su nuevo Stoploss
   Modificar = true; / / Asignación de ser modificado.
   )
   ) / / Fin de "switch"
   if (Modificar == false) / / Si no hay necesidad de modificarla ..
   continuar; / / .. luego continuar con el ciclo de
   bool ans = OrderModify (Venta de entradas, Precio, SL, TP, 0); / / Modificar it!
   //---------------------------------------------- -- ------------ ---------- 5 --
   if (respuesta == false) / / Error: (
   (/ / Comprobar si hay errores:
   if (errores (GetLastError ()) == false) / / Si el error es crítico,
   return; / / .. después se van.
   i - / / La disminución de la lucha contra
   )
   Terminal (); / / función de la contabilidad Orden
   Eventos (); / /   Seguimiento de eventos
   )
   return; / / Salir de la función definida
   )
   / / - ------------- ---------------------------------- --------------------------
-- 6 --
```

En los bloques de 1-3, las variables globales se describen que se utilizan en la función, así como las variables locales se abren y se describen. En el ciclo 'for' (bloques 3-6), las órdenes del mismo tipo son seleccionados y, si el Stoploss de cualquiera de esas órdenes es más del precio actual de lo que fue establecido por el usuario, se modifica la orden.

Para hacer el código más orientado al ser humano, los valores de algunos elementos de la matriz para Mas_Ord_New son asignados a las variables simples (bloque 3-4). Luego, el cheque será necesario para la TralingStop variable: Si el valor de esta variable es menor que la distancia mínima permitida establecida por el centro de la frente, que se incrementará hasta el valor mínimo permitido.

En el bloque de 4-5, según el tipo de orden, se hacen los cálculos necesarios. Por ejemplo, si el valor del parámetro transferido Consejo es 1 (una orden de venta debe ser modificada), el control se transferirá a la marca "caso 1" del operador "switch". La necesidad de modificar el orden de Stoploss se comprueba aquí (de acuerdo a las normas que se aplican a este tipo de pedido, ver Requisitos y limitaciones en hacer operaciones). Si no Stoploss está previsto, o si se fija a una distancia mayor que el valor de TralingStop del precio de mercado actual, el nuevo valor deseado de Stoploss se calcula. Petición del Comercio para la modificación de la orden se forma en línea:

```
   bool ans = OrderModify (Venta de entradas, Precio, SL, TP, 0); / / Modificar it!
```

Se señaló antes de que la estrategia comercial considerado aquí implícita la disponibilidad del orden de un único mercado. Sin embargo, la función de Tral_Stop () ofrece la posibilidad de modificar varias órdenes de mercado de un tipo. Si el operador no interviene en la negociación durante los trabajos de la EA, sin necesidad de modificar varios órdenes se produce. Sin embargo, si el comerciante abre un orden de mercado

de forma manual (además de las ya iniciadas), tenemos que decidir cuál de las órdenes disponibles deben ser modificados como el primero y por qué.

Al considerar la secuencia de cierre de varios órdenes, se mencionó que el criterio que define la prioridad en el cierre de las órdenes era la cantidad de lotes. Esta solución es obvia - el más lotes (del total) están cerradas, cuanto antes, EA respuesta a la activación del criterio de cierre. El problema de la secuencia de la orden de modificación no tiene ninguna solución inequívoca. En todos los casos, el criterio para la secuencia de modificación de orden es determinado por la esencia de la estrategia de negociación. Este criterio puede ser tanto la cantidad de lotes, el hecho de no Stoploss en uno de los pedidos, la distancia de Stoploss al precio actual. En algunos casos, este criterio puede ser expresada a través de un índice general - el tamaño de las pérdidas que puedan derivarse de los cambios de los precios, es decir, cuando todos los órdenes de mercado se cierran automáticamente por Stoploss, al mismo tiempo.

En el ejemplo anterior de la función Tral_Stop (), una secuencia aleatoria de cambio de pedido se realiza - las órdenes son modificados en la secuencia, en la que se producen en los pedidos pendientes de las órdenes de pérdida de mercado abierto y colocado. En cada caso específico, la función debe ser refinado en - la secuencia de modificación de orden debe ser programado de acuerdo a las reglas de su estrategia de negociación específica.

Se debe prestar especial atención al hecho de que todas las operaciones se realizan en el modo de tiempo real. Si hay demasiadas órdenes, la EA va a generar una gran variedad de peticiones de comercio. Obviamente, el mercado puede dar la vuelta, mientras que las solicitudes están siendo ejecutados. Sin embargo, la función no devuelve el control a la función de Comercio () que ha llamado a él hasta que todos los pedidos que se deben modificar son modificados. Esto significa que el peligro de omitir una solicitud de cesión de apertura o cierre de pedidos puede ocurrir. Por esta razón, cualquier estrategia debe ser codificada de tal manera que no permitan que una cantidad considerable de órdenes de mercado que estén disponibles a la vez.

En el bloque de 5-6, los errores se durante la ejecución de las solicitudes se analizan el comercio. Si el error es crítico, la función se ponga fin a sus operaciones. Sin embargo, si un error se ha overcomable tiene, el valor del contador 'i' se reduce en 1. Se hará con el fin de producir un nuevo intento de modificar el mismo orden en la siguiente iteración del ciclo 'para'.

En la mayoría de los casos, el código anterior se cumple con la necesidad de modificar varios pedidos. Al mismo tiempo, en caso de efectuar cambios en las órdenes (por ejemplo, un pedido se cerrará cuando el precio de mercado alcanza uno de los niveles de parada) en el plazo de varios intentos fallidos para modificar las órdenes, la secuencia de órdenes en el matriz Mas_Ord_New también puede cambiar. Esto dará lugar a que una orden puede ser omitido y no modificados dentro del período de lanzamiento de la última a partir del inicio de funciones especiales (). Esta situación puede mejorar a la garrapata que viene, en el próximo lanzamiento del inicio de la función ().

Error al procesar la función

Los errores que aparecen durante la ejecución de órdenes de negociación puede dividirse en dos grupos - overcomable (no crítica) los errores y los errores críticos. Overcomable errores son los que ocurren en fallos en el servidor. Después de que hayan sido eliminadas, puede continuar su actividad. Por ejemplo, una solicitud puede ser rechazada por el corredor, si no hay información acerca de las cotizaciones actuales en el momento. Este tipo de situación puede aparecer en un mercado lento, es decir, cuando las garrapatas no son los ingresos con frecuencia. O, por el contrario, el agente no siempre se puede ejecutar una gran cantidad de peticiones de los comerciantes en un mercado activo, donde también muchas citas están llegando. Luego de la pausa aparece antes de la orden se ejecuta o - a veces - una negación. En tales casos, el Asesor de Expertos pueda continuar con su trabajo y, por ejemplo, repetir la petición un poco más tarde después de la ejecución de algunos código relacionado con el código de error.

Los errores críticos incluyen todos los errores que alerta sobre graves problemas. Por ejemplo, si se bloquea una cuenta, entonces no hay ningún punto en el envío de solicitudes comercio. En tal caso, la EA debe mostrar el mensaje correspondiente y no debe repetir la petición. La función de procesamiento de error debe ser utilizado indispensablemente en una EA normal.

Custom error de procesamiento Función errores ()

```
Errores bool (int error)
```

La función devuelve TRUE, si el error es overcomable. De lo contrario, devuelve False.
El parámetro **de error** puede tener cualquier valor y se corresponden con cualquier código de error que se produjo al tratar de hacer un intercambio.

Definido por el usuario los errores de procesamiento de error function () está diseñado como el archivo de inclusión Errors.mqh:

Una de las preguntas que surgen cuando se componen los algoritmos de procesamiento de errores de la función de error () es: ¿Qué debe de retorno de la función, si el valor de un parámetro dado es 0 (es decir, no hay errores). Este tipo de situación no debe aparecer en una correcta codificación de EA. Sin embargo, el código puede ser de diversas vez al tiempo que mejora el programa, así que a veces el valor de un error puede ser igual a 0. Por lo tanto, sería razonable añadir algunas líneas a la función en la etapa primaria de desarrollo (bloque 2-3), para las situaciones en que un error, es igual a 0.

La reacción de los errores () para el valor cero de la variable de error depende del algoritmo utilizado para procesar los valores devueltos por la función. El valor devuelto por la función se tiene en cuenta en la función del comercio ejecutable de la anterior EA. Si los errores () devuelve 'true' (error es overcomable), entonces el programa volverá a intentar hacer un intercambio. Si devuelve 'false', entonces el comercio deja de funcionar y el control se pasó sucesivamente a la función de llamada, luego a la función start (), y luego a la terminal de cliente. Si la elección es entre estas dos alternativas, la situación cuando no hay errores (Error = 0) se corresponde con la segunda alternativa, es decir, con el valor 'false' regresó. Se garantiza que una petición, una vez ejecutada no se repetirá.

Una vez que el mensaje sobre el error se muestra por el Informe () función, el control pasa al bloque de 3-4, para el operador "switch". El caso se trata de la variante específica para cada código de error en cuenta. Por ejemplo, si el error 136 se produce, significa que el corredor no tiene las cotizaciones actuales para tomar una decisión adecuada. Esto significa que la situación no cambiará a menos que una garrapata nuevo viene, así que no hay necesidad de repetir el envío de la solicitud de cesión mismo, ya que no se ejecutará, de todos modos. La solución correcta en esta situación es la pausa - la ausencia de cualquier iniciativa de la EA. Un método simple para detectar una garrapata nuevo se utiliza para este fin - el análisis del valor devuelto por la RefreshRates () la función. El control será devuelto a la función llamada, en la que se reiteró la petición de comercio (después del análisis correspondiente, si es necesario), tan pronto como la garrapata nuevo viene.

Si hay un error que el programador considera de carácter crítico, la función devuelve 'false'. La solicitud no se repetirá, en tal caso, lo que no hay necesidad de hacer nada en los errores () la función. Todos los errores no se consideran tratados como críticas por defecto. Puede ampliar la lista de errores procesables (véase Códigos de error).

```
//+------------------------------------------------------------------+
/ / Errors.mqh
/ / El código debe ser utilizado sólo para fines educativos   .
//+------------------------------------------------------------------+ 1 --
/ / Función de procesamiento de error.
/ / Devuelve valores:
/ / True - si el error es overcomable (es decir, el trabajo se puede continuar)
/ / False - Si el error es crítica (es decir, el comercio es imposible)
//+------------------------------------------------------------------+ 2 --
Errores bool (error int) / / función personalizada
(
/ / Error / / Número de error
if (error == 0)
return (false); / / No hay error
Informar a los (15, error); / / Mensaje
//+------------------------------------------------------------------+ 3 --
switch (error)
(/ / Errores Overcomable:
Caso 129: / / precio incorrecto
caso 135: / / Precio cambiado
RefreshRates (); / / Renovación de datos
return (true); / / error es overcomable
caso 136: / / No comillas.  Lugar para la garrapata por venir
while (RefreshRates () == false) / / Antes de marcar nuevas
Sleep (1) / / Retraso en el ciclo de
return (true); / / error es overcomable
caso 146: / / El subsistema de comercio está ocupado
Sleep (500); / / Solución sencilla
RefreshRates (); / / Renovación de datos
return (true); / / error es overcomable
/ / Los errores críticos:
case 2: / / error común
Caso 5: / / antigua versión del cliente de terminal
case 64: / / Cuenta bloqueada
caso 133: / / Se prohíbe Trading
default: / / Otras variantes
return (false); / / Error crítico
)
//+------------------------------------------------------------------+ 4 --
)
//+------------------------------------------------------------------+
```

Características generales de los programas complejos.

No hay ninguna característica formal única que distingue a un programa habitual de un complejo. En general, los programas complejos positivamente difieren en una variedad de herramientas que proporciona y la cantidad de información procesada. Sólo algunos de los adjetivos cualitativos que son peculiares de los programas complejos puede ser denotado.

Programa de ejecución de órdenes

Como regla general, un programa habitual contiene su código en el inicio de la función especial () que se inicia la ejecución por el terminal de cliente. En la mayoría de los casos la función start () el tiempo de ejecución es considerablemente menor que el período de la garrapata. Esto significa que la mayor parte del tiempo el programa está esperando una garrapata a venir. Este tipo de procesos se caracterizan por el encendido-apagado término de relación. On-off ratio es la relación de un período de proceso de la repetición de la duración del proceso en sí. El tiempo de ejecución de inicio () **T1** es casi 10 a 100 milisegundos, y el tiempo **T2** entre marcas es de 5 segundos en la media. Así, el encendido-apagado de una relación de trabajo EA está llegando a **T2/T1** = 1000 y más (ver fig. 159). Es decir que la duración de la capacidad efectiva a un funcionamiento normal de EA es del 0,1% de todo el tiempo, el resto del tiempo que está de pie.

A veces los cálculos complejos pueden ser ejecutados por un EA y como resultado de la duración de inicio () la ejecución puede ser más largo y llegar a decenas de segundos. En estos casos () inicio de la función no se iniciará en cada tick pero sólo en las garrapatas que llegó cuando el inicio () está esperando por ellos. La fig. 159 muestra que la garrapata que se produjo en la ejecución de las start () período de la función (en la T4 momento) no causará iniciar una función especial de nuevo. La próxima vez que la función start () se iniciará en el momento de la T5. La pausa entre el final de la ejecución en curso y el comienzo de la próxima ejecución de la función start () aparecerá con esta disposición.

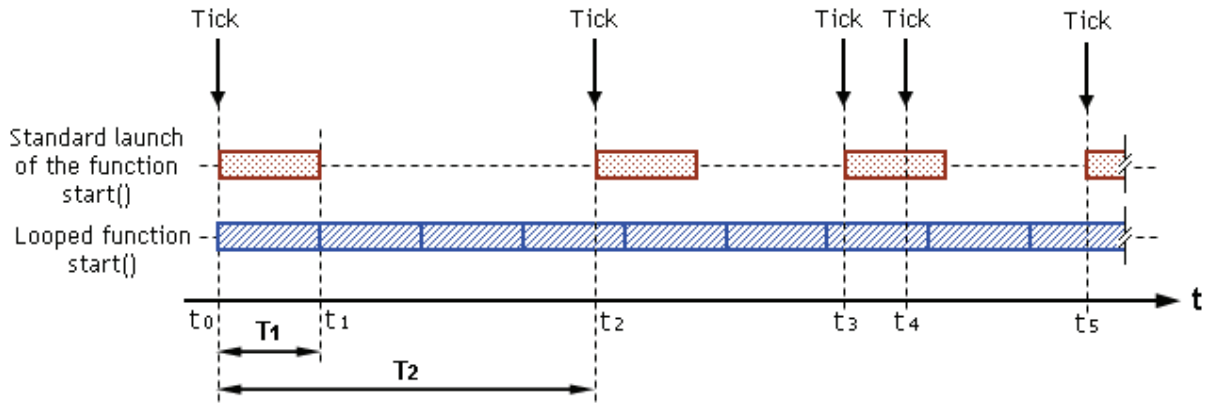


Fig. 159 diferentes on-off, mientras que el ratio de función start () es ejecutado por el cliente de terminal y el inicio de ciclo () la función.

Hay un método para aumentar la capacidad efectiva del programa, esencialmente, lo que disminuye cociente encendido-apagado del proceso de gestión del comercio. Para ello vamos a implementar un algoritmo, según el cual el código principal es muchas veces (infinitamente) repitió durante el inicio () de ejecución (permitido sólo en las ZE y secuencias de comandos). El ejemplo de la salida en bucle () se muestra a continuación:

```
//------------------------------------------------ --------------------
start () / / Función especial de inicio ()
(
mientras que ()) / / Hasta el usuario (! IsStopped ..
(/ / .. Se detiene la ejecución del programa de
RefreshRates (); / / renovación de datos
//..................... El código principal del programa se especifica aquí
Sleep (5); / / Breve pausa
)
return; / / El control se devuelve a la terminal
)
//------------------------------------------------ --------------------
```

Todo el código se especifica en el cuerpo del "mientras que" operador de ciclo, y la única manera de salir del ciclo va a recibir un comando desde el Terminal Server para finalizar la ejecución del programa. Si la función start () basada en este principio se

especifica que se inicie, se ejecutará infinitamente largo y devolver el control al terminal del cliente sólo cuando un usuario elimina manualmente la EA desde una ventana de la seguridad o la prestación de algunas otras condiciones (véase Funciones Especiales) .

La ejecución de la salida en bicicleta () se ejecuta de manera ininterrumpida, por eso no hay época en que el programa está en el modo de espera de una garrapata de nuevo (ver fig. 159), por lo que el cociente encendido-apagado del proceso de la ejecución del programa en bicicleta es igual a 1. La función start () que se basa en el principio especificado que se inicia en la terminal, el cliente sólo una vez. Esto significa que la actualización de la información (cotizaciones de mercado, por ejemplo) deben ser obligatorios a cabo utilizando la RefreshRates función (). Con el fin de evitar el consumo de recursos muy grande de una breve pausa al final del ciclo se puede especificar.

El desarrollo del programa de bucle requiere mucha atención durante la composición del algoritmo. Por ejemplo, la reacción de un programa habitual de un error crítico recibido es romper el start () ejecución de la función y devolver el control al terminal del cliente. Un programa de ciclo mantiene el control de forma permanente mientras se ejecuta lo que la reacción de otros se debe prever, por ejemplo, la prohibición de la generación de órdenes de negociación sobre un cierto período. Sin embargo, la prohibición temporal no debe obstaculizar la ejecución del programa. Durante el período de ejecución conjunto, el programa debe procesar toda la información disponible acerca de los acontecimientos, incluyendo las medidas de control de un usuario. En general, este programa tiene un poder inconmensurable, en comparación con uno normal.

Herramientas disponibles

El uso de programas de podado sólo tiene sentido si la continuidad de la ejecución de un programa es utilizado eficazmente. Por ejemplo, un programa capaz de procesar las acciones de control de un operador. La modificación de las coordenadas de objetos gráficos o el hecho de agregar otros programas - las secuencias de comandos y los indicadores pueden ser considerados como el control de las acciones.

Un simple programa puede responder a algunos eventos (incluyendo iniciado por el usuario) en un inicio regular del inicio de la función especial () en el más cercano de garrapatas, por regla general. Mientras que un programa de ciclo puede procesar todos los acontecimientos inmediatamente (!). En este caso, el retraso sólo puede ser por un tiempo corto, no más que el tiempo de ejecución de un ciclo de la función start () (cerca de no más de 10-100 ms).

Un programa complejo puede usar objetos gráficos para mostrar las características orden o reglas de su modificación. Por ejemplo, las órdenes de todo tipo se muestran en una ventana de seguridad en las líneas verdes, órdenes de suspensión - en las líneas rojas. Y si en varios órdenes se muestran en la pantalla al mismo tiempo, es muy difícil de detectar a qué línea pertenece a este o aquel orden. Pero si se aplica el objeto gráfico "línea horizontal" de un color preciso y el estilo a cada línea de orden, será mucho más fácil distinguir entre los pedidos y las órdenes de parada.

Además, el hecho de cambiar las coordenadas de objetos, puede ser percibido por un programa como una guía para la acción. Por ejemplo, si un usuario cambia una línea horizontal que muestra una espera de varios puntos para arriba, como resultado de esta acción, el programa puede formulario y enviarlo al servidor una solicitud de cesión, según el cual la orden debe ser modificado, es decir, la preselección abierta precio debería ser aumentado en varios puntos (para una ejecución inmediata de la utilización de un programa de bucle es obligatorio). Así, un complejo del programa puede ofrecer la posibilidad de gestionar el comercio con el ratón.

Función utilizada para la modificación del sistema de parada independiente de pedidos o una declarada precio de apertura de un orden se puede utilizar en programas complejos también. Si esta función se utiliza aplicable a una de las líneas de pedidos, un objeto gráfico, por ejemplo, "una flecha" puede ser mostrada cerca de la línea de pedido, indicativo de la actividad de la función.

Uso de objetos gráficos puede configurar escenario de negociación. Por ejemplo, configurar el "eje" del pabellón, a cierta distancia del precio actual, puede informar del programa que es necesario cerrar el pedido y abrir una nueva en la dirección opuesta cuando se alcanza el precio especificado. Del mismo modo, puede especificar los límites de nivel de modificación, el precio de cierre de pedidos, etc en el programa. El uso de objetos gráficos que muestran la configuración de un programa aumenta considerablemente la conciencia de los comerciantes acerca de los acontecimientos actuales y previstos.

Señales de sonido asociadas con eventos también se utilizan en programas complejos. Uso de sonidos en un programa permite al comerciante a dejar el PC y orientar a través de eventos por los tipos de señal de sonido (melodías, texto vocalizado, etc.)

La eficacia de la negociación depende a menudo del hecho de si el momento de la importante nota de prensa económica y política se toma en cuenta. La gran mayoría de los centros ofrecen los comerciantes que tratan con la lista de las noticias de la semana más cercano con la denotación de un tiempo de liberación y su importancia. La información sobre los próximos eventos se registra en un archivo. Durante la operación de un programa lee la información del archivo y realiza uno u otro escenario de comercio en función de la importancia de un evento que viene. Por ejemplo, un programa puede eliminar la espera de órdenes, modificar órdenes para detener las órdenes de mercado, etc Poco antes de las noticias importantes procedentes del programa pueden terminar comerciales - cerca de todos los pedidos de manera preliminar se informa a un comerciante.

Automatizado y manual Modo de Operación del Programa

Un programa complejo se caracteriza por un algoritmo más complejo de los eventos de transformación. En particular, este tipo de programas supone una reacción a la injerencia de un comerciante con el proceso de negociación. Por ejemplo, si una estrategia de negociación permite que sólo una orden de mercado y un comerciante abre un orden más, un programa complejo, en primer lugar, los monitores de este tipo de evento y luego comienza la ejecución de la parte algoritmo previstas para tal ocasión. El programa se puede advertir a un operador sobre una injerencia no autorizada, en un primer momento, y ofrecer a fin de eliminar la extraña forma independiente. Si esto no ocurre, el programa (dependiendo de la configuración) puede eliminar el objeto extraño o salir del modo de negociación automatizada de la información que el comerciante previamente.

Si un programa se inicia para la ejecución cuando hay varios pedidos ya colocados, las acciones necesarias se llevarán a cabo en función de su configuración. Por ejemplo, el programa puede cerrar todas las órdenes de lo contrario sin el acuerdo de un operador. Si una estrategia de negociación no permite que las órdenes pendientes, que serán eliminados en la secuencia de prioridad - en primer lugar, los más próximos a la cotización de mercado, más caro, etc

Una vez que el operador ha fijado la cantidad y los límites de calidad de los pedidos en una estrategia de negociación, el programa (trabajando todo el tiempo y el seguimiento de los acontecimientos) puede proponer un comerciante para activar el modo de negociación automatizada, y si un comerciante está de acuerdo, designar a la persigue escenario de comercio con objetos gráficos.

Cada comerciante tiene su propio conjunto de preferencias cuando se trabaja con un programa. Algunos operadores admiten sólo la negociación automatizada, los comerciantes de otros - un medio automatizado, terceras partes prefieren sólo el modo manual. Un programa bien diseñado debe cubrir todos los requisitos, es decir, debe tener un número de valores que el modo de uso diferentes. Por ejemplo, un programa puede actuar como asesor en el modo manual de trabajo - muestra un texto con recomendaciones directas y también los objetos gráficos que muestra una dirección de la tendencia, la previsión puntos de giro, etc Un programa puede pedir a un comerciante a fin de permitir la apertura, admiten interferencia comerciante de gestión de los pedidos (por ejemplo, dejar de manual de órdenes de modificación), mientras trabajaba en el modo half-automatizado. En caso de que el programa se ejecuta en el modo automático a cualquier comerciante de interferencia en el proceso de negociación puede ser considerado como una señal para cambiar el modo para un medio automatizado o manual.



Todas las propiedades que se describen de un programa puede ser incorporado en la base de MQL4 lenguaje de programación que está especialmente diseñado para este propósito. Un complejo programa correctamente diseñado tiene un número de ventajas incontestables, un operador rápidamente se acostumbra a ellos y empieza a usar en el comercio.