# MQL4 **Reference**

# **MQL4** 命令手册

（本手册采用 Office2007 编写）

2010 年 2 月

# 目录

MetaQuotes Language 4 (MQL4) 是一种新的内置型程序用来编写交易策略。 这种语言可以创建你自己的智能交易，使自己的交易策略能够完全自动地执行。而且，MQL4 还能自定义客户指标，脚本和数据库。

内包含了大量可以分析当前及历史报价所必须的函数，以及一些基本的运算和逻辑操作。并内置了一些基本的指标和操作命令。

MetaEditor 4 集合了编写 MQL4 程序代码的各种语句，它能帮助使用者方便地写出规范的代码。 MetaQuotes Language Dictionary 是 MQL4 语言的帮助工具，它包含了我们在使用工程中所有可能用到的函数。

MetaQuotes Language 4 可以编写不同作用的程序代码:

■**智能交易** 是一种连接到特定图表的自动交易系统。它能够根据设置的节点自动启动 ，当它开始运行后，它不会同时去处理另一个新的指令(也就是说必须等到当前程序完成)。 这种交易系统能够在提醒用户可以交易的同时，将交易定单自动送到交易服务器。与大多数交易系统一样， 它也能够用历史数据测试交易策略，并在图表上显示出来。

智能交易存储在 terminal_directory\experts。

■**自定义指标** 可用来编写新的技术指标，和内置的指标一样，它不能用来进行自动交易，只能作为分析数据的工具。

自定义指标储存在 terminal_directory\experts\indicators。

■**脚本** 是执行单一功能的一段程序，和 智能交易不同，脚本不能单独执行，只能被调用。

脚本存储在 terminal_dictionary\experts\scripts。

■**数据库** 常被使用的自定义函数的集合。数据库不能单独运行。

数据库建议存储在 terminal_directory\experts\libraries。

■**包含文件** 包含文件常被使用的程序块源代码，这些文件能够被包含在智能交易，脚本，客户指标和数据库 的源代码中。 使用包含文件比调用资料库更灵活快捷。

包含文件交易储存在 terminal_directory\experts\include。

# Basics 基础

MetaQuotes Language 4 (MQL4)是一种新型的交易策略内置语言。用来编写交易策略的程序语言。这种语言可以创建你自己的智能交易， 使自己的交易策略能够完全地自动执行。程序内包含了分析历史报价的必备函数， 以及一些基本的运算法和逻辑操作和一些基本的指标和操作命令。 而且，MQL4 还能自定义自己的客户指标，脚本和数据库。

## Syntax 语法

MQL4 的语法类似于 C 语言，除了以下这些特点:

- 没有运算地址；
- 没有 do ... while 语句；
- 没有 goto ... 语句;
- 没有 [条件][表达式 1]:[表达式 2] 语句；
- 没有复合数据类型 (结构);
- 复合负值是不允许的，例如：val1=val2=0; arr[i++]=val; cond=(cnt=OrdersTotal)>0; 等等;
- 逻辑表达式的计算完成前不可以提前终止。

## Comments 注释

多行注释使用 /* 作为开始到 */ 结束，在这之间不能够嵌套。单行注释使用 // 作为开始到新的一行结束，可以被嵌套到多行注释之中。

示例:

```
// 单独注解
/*   multi-
     line          // 嵌入单独注解
     comment
*/
```

## Identifiers 标识符

标识符用来给变量、函数和数据类型进行命名，长度不能超过 31 个字节，你可以使用数字 0-9、拉丁字母大写 A-Z 和小写 a-z(大小写有区分的)还有下划线(_)。此外首字母不可以是数字，标识符不能和保留字冲突.

示例:

NAME1 namel Total_5 Paper

## Reserved words 保留字

下面列出的是固定的保留字。不能使用以下任何保留字进行命名。

| 数据类型 | 储存类型 | 操作符 | 其他 |
|---|---|---|---|
| bool | extern | break | false |
| color | static | case | true |
| datetime | | continue | |
| double | | default | |
| int | | else | |
| string | | for | |
| void | | if | |
| | | return | |
| | | switch | |
| | | while | |

# Data types 数据类型

所有的程序都依靠数据来运作，数据因目的不同可以有不同的类型 。比如，访问数组可以用整型数据，价格可以用双精度的浮点型数据。在 MQL 4 中没有专门用来标记货币值的数据类型。

不同的数据类型有不同的处理速度，整型数据是最快的。 双精度的数据处理需要额外的处理器，所以处理浮点型数据比较复杂， 比处理整型数据慢一些。字符串是处理速度最慢的，因为它要存取动态内存。

主要的数据类型如下:

- 整型数据 (int)
- 布尔数据 (bool)
- 字符数据 (char)
- 字符串数据 (string)
- 浮点型数据 (double)
- 颜色数据 (color)
- 日期时间数据 (datetime)

color 和 datetime 可以使我们更清楚的区分图表中的内容，在 expert advisor 和 indicator 中经常使用这些数据类型。颜色和日期时间数据用整数来表示。int 和 double 都属于数值（数字）型。

在表达式运算中使用强制的类型转换。

## Type casting 类型转换

表达式中使用强制的数据转换，转换时类型的优先级如下:
int (bool,color,datetime);
double;
string;
在运算完成之前(除了数据已被定义的)，数据会根据优先级被转换。当定义数据的操作完成前 ，数据会转换成被定义的数据类型。

示例:

```
int       i = 1 / 2;        // 没有类型转换，结果为 0
int       i = 1 / 2.0;      // 表达式中有浮点型数据，但会转换成整型数据，结果为 0
double d = 1.0 / 2.0; // 没有类型转换，结果为 0.5
double d = 1 / 2.0;     // 表达式计算的结果是浮点型数据，和定义的类型一样，结果为 0.5
double d = 1 / 2;         // 表达式是整型数据的计算，然后被定义为浮点型数据，结果为 0.0
```

类型转换不但运用在常量中，还被运用在相应的变量中。

## Integer constants 整数常量

十进制: 数字 0 - 9 ，包括负数。

示例:

12, 111, -956 1007
十六进制: 数字 0 - 9 , 字面 a - f 或者 A - F 代表 10 - 15; 以 0x 或者 0X 开头。

示例:

0x0A, 0x12, 0X12, 0x2f, 0xA3, 0Xa3, 0X7C7
整型数据占用 4 字节的空间，其数值范围介于 -2147483648 ～ 2147483647 之间。如果超出这个范围，则视为无效。

## Literal constants 字面常量

任何带单引号的单一字符或者十六进制的 ASCII 码如 '\x10' 都是字符数据。一些特殊的字符如单引号(')、双引号(")、问号(?)、反斜线(\)和控制符必须以反斜线开头(\)， 组合表达原来的意思，如下表所示：

| | | |
|---|---|---|
| 换行 | NL (LF) | \n |

| | | |
|---|---|---|
| 制表符 | HT | \t |
| 回车 | CR | \r |
| 反斜线 | \ | \\ |
| 单引号 | ' | \' |
| 双引号 | " | \" |
| 十六进制 ASCII | hh | \xhh |

如果上述字符不使用反斜线，结果将不被定义:

```
int a = 'A';
int b = '$';
int c = '©';          // 代码 0xA9
int d = '\xAE';     //货币对代码 ®
```

字符数据占用 4 字节的空间。其数值范围介于 0 ～ 255 之间。如果超出这个范围，则视为无效。

## Boolean constants 布尔常量

Boolean 用来表示 是 和 否, 还可以用数字 1 和 0 进行表示。True 和 Flase 可以忽略大小写。

示例:

```
bool a = true;
bool b = false;
bool c = 1;
```

它的十进制表示一个长度为 4-byte 的整数值。Boolean 常数可以表示 0 或 1 值。

## Floating-point number constants (double)浮点数常量（双精度）

浮点型数据由整数部分、小数点(.)和小数部分组成，其中整数部分和小数部分为一系列十进制数字。

示例:

```
double a = 12.111;
double b = -956.1007;
double c = 0.0001;
double d = 16;
```

浮点型数据 (双精度)占用 4 字节的空间。其数值范围介于 -1.7 * e-308 ～ 1.7 * e308 之间。如果超出这个范围，则视为无效。

## String constants 字符串常量

字符串数据是带有双引号的一连串 ASCII 字符 ，如: "Character constant"。

字符串数据是引号里的一组字符，如果字符串中需要插入一个双引号(") 必须在它前面使用反斜线(\)。任何特殊字符都必须有前置的反斜线(\)才能在字符串中使用。字符串可以容纳 0 到 255 个字符，如果超过这个长度，右边多余的字符将被忽略，编译器也会有相应的警示。

示例 s:

```
"This is a character string"
"Copyright symbol \t\xA9"
"this line contains a line feed symbol \n"
"C:\\Program Files\\MetaTrader 4"
"A" "1234567890" "0" "$"
```

字符串数据占用 8 个字节的空间。其中第一部分为长的整型存储字符串缓冲区分布的长度。第二部分是 32 位的存储字符串缓冲区的地址。

## Color constants 颜色常数

颜色数据可以用三种方法表示： 字符数据、整型数据或者是颜色名(只能是 Web colors 中已命名的).

字符数据的表达方法是用三个数字来表示三种主要颜色：红、绿、蓝的比例。 以 C 开头，用单引号括住。数字的值在 0 ～ 255 之间按比例选取。

整数数据的表达方法使用十六进制或十进制数字。十六进制数字如 0x00BBGGRR， 其中 RR 是红色的比例，GG 是绿色的比例，BB 是蓝色的比例。十进制数不能直接体现红绿蓝的比例 ，而是十六进制数字的十进制表示方式。

特殊的颜色名可以参考 Web colors set 表。

示例:

```
// 字符数据
C'128,128,128'       // 灰色
C'0x00,0x00,0xFF' // 蓝色
// 颜色名
Red
Yellow
Black
// 整型数据
```

```
0xFFFFFF          // 白色
16777215          // 白色
0x008000          // 绿色
32768             // 绿色
```

颜色数据占用 4 字节的空间。第一个字节一般被忽略，后三个字节包含了红绿蓝的组成信息。

## Datetime constants 日期时间常数

日期时间数据由 6 个部分的字符组成：年、月、日、时、分、秒，以 D 开头， 用单引号括起。日期(年、月、日)或者时间(时、分、秒)甚至两者一起都可以不用填写。 日期时间数据开始于 1. 1. 1970 截止到 12.31. 2037

示例:

```
D'2004.01.01 00:00'       // 新年
D'1980.07.19 12:30:27'
D'19.07.1980 12:30:27'
D'19.07.1980 12'          //等于  D'1980.07.19 12:00:00'
D'01.01.2004'             //等于  D'01.01.2004 00:00:00'
D'12:30:27'               //等于  D'[编译日期] 12:30:27'
D''                       //等于  D'[编译日期] 00:00:00'
```

日期时间数据占用 4 字节空间长度的整型数值。其值从 1970 年 1 月 00:00 开始以秒的形式显示总数。

# Operations & Expressions 操作表达式

一些数字和字符的组合是特别重要的，它们被称为运算符，例如:

```
+ - * / %       算术运算符
&& ||           逻辑运算符
= += *=         负值运算符
```

运算符应用在表达式中实现特定的作用。
需要特别注意标点符号如圆括号、方括号、逗号、冒号、分号。
运算符、标点符号、空格用来分割语句的不同部分。

## Expressions 表达式

一个表达式可以拥有多个字符和操作符，一个表达式可以写在几行里面。

示例:

```
a++; b = 10;
x = (y * z) /
      (w + 2) + 127;
```
一个表达式的最后一个分号(;) 操作符。

## Arithmetical operations 算术运算

算术运算符包括加法和乘法运算:

| | |
|---|---|
| 求和 | i = j + 2; |
| 求差 | i = j - 3; |
| 改变运算符 | x = - x; |
| 求积 | z = 3 * x; |
| 求商 | i = j / 5; |
| 求模 | minutes = time % 60; |
| 自加 1 | i++; |
| 自减 1 | k--; |

添加 1 的运算符不能使用在表达式中。

示例:

```
int a=3;
a++;                    // 有效表达式
int b=(a++)*3;      // 无效表达式
```

## Assignment operation 赋值操作

表达式的值包括左边值给出的赋值运算符。

| | |
|---|---|
| 把变量 x 的值赋予变量 y | y = x; |

下列表达式中赋值运算符结合了算术运算符或位运算符:

| | |
|---|---|
| 在 y 值上加上 x | y += x; |
| 在 y 值上减去 x | y -= x; |
| 在 y 值上乘以 x | y *= x; |
| 在 y 值上除以 x | y /= x; |
| 在 y 值上求 x 的模 | y %= x; |
| 把 y 值向右做 x 位逻辑移位 | y >>= x; |
| 把 y 值向左做 x 位逻辑移位 | y <<= x; |
| AND 位运算符 | y &= x; |
| OR 位运算符 | y \|= x; |
| 把 x 和 y 按做逻辑异或的操作 | y ^= x; |

表达式中可以只能有一个赋值运算符。 位运算符只能用于整型数据。逻辑移位运算符中 x 值只能是小于 5 位的二进制数，过大的数值将会被拒绝。所以移动范围只能是 0 到 31 。用 %= 运算符 ( 用 x 的模板求 y 值)，其结果等于余数。

## Operations of relation 操作关系

逻辑值 FALSE 代表整数零值，逻辑值 TRUE 代表不同于零的任何值。
用返回 0(False)或 1(True)来表示两个量之间的关系。

| | |
|---|---|
| 等于 b | a = b; |
| 不等于 b | a != b; |
| 小于 b | a< b; |
| 大于 b | a >b; |
| 小于等于 b | a <=b; |
| 大于等于 b | a >= b; |

2 个不规范的浮点型数据不能用 = 或 != 运算符比较，但是我们可以把 2 者相减， 正常化后和 null 进行比较。

## Boolean operations 布尔运算

否定运算符(!),用来表示真假的反面的结果。如果运算值是 FALSE (0) 结果为 TRUE (1)；如果运算不同于 FALSE (0)等于 FALSE (0) 。
if(!a) Print("不是 'a'");
x 和 y 值的逻辑运算符或 OR (||)用来表示两个表达式只要有一个成立即可。如果 x 和 y 值为真的，表达式值为 TRUE (1)。否则，值为 FALSE (0)。逻辑表达式被完全计算。
if(x<0 || x>=max_bars) Print("超出范围");
x 和 y 值的逻辑运算符 AND (&&)。如果 x 和值都是真实的，表达式值为 TRUE (1)。 Otherwise, it is FALSE (0).
if(p!=x && p>y) Print("TRUE");

## Bitwise operations 位运算

运算符对操作数执行按位求补操作。表达式的数字值中包含 1，其中 n 包含 0 和数字值中包含 0 ，其中 n 包含 1。
b = ~n;
运算符 x 向右移动到数字 y 代表二进制代码。向右移动是逻辑运算，即左侧将被零填满。
x = x >> y;
运算符 x 向右移动到数字 y 代表二进制代码。左侧将被零填满
x = x << y;
二进制的 x 和 y 代表位逻辑运算符 AND 。在所有数组中 x 和 y 的值都不含有零表达式的值包含 1 (TRUE)；在所有其他数字中包含 0 (FALSE)。

b = ((x & y) != 0);
二进制的 x 和 y 代表位逻辑运算符 OR。在所有数字中 x 和 y 的值都不等于零表达值包含 1 并且在所有其他数字中包含 0。
b = x | y;
二进制的 x 和 y 代表位逻辑运算符 EXCLUSIVE 。在所有数字中 x 和 y 的值都不同于二进制值表达值包含 1 并且在所有其他数字中包含 0 。
b = x ^ y;
位逻辑运算符只作用于 Integers 类型。

## Other operations 其他运算

指数
在数组第一元素的位置，表达式值为 i 的系列数变量值。
示例:
array[i] = 3; //数组的 3 的计算值到第 i 个元素。
只有整数能够成为数组指数。四维以下的数组是禁止的。每组的检测是从 0 到 测量大小-1。特定情况下，对于维数组由 50 个元素组成，参照的第一个数组将为[0]，这样最后一个数组将是[49]。
获取超出数组，将会发生常规错误 ERR_ARRAY_INDEX_OUT_OF_RANGE， 可以调用 GetLastError() 函数。
调用 x1,x2,...,xn 自变数函数
每一个自变数可以显示一个常数，一个变量和相应类型表达式。自变数的通过必须根据通道命令。
用此函数返回表达式值。如果返回的表达式值为空，一些函数不能进行中转。请确认表达式 x1,x2,...,xn 是按照命令执行的。
示例:

```
double SL=Bid-25*Point;
int      ticket=OrderSend(Symbol(),OP_BUY,1,Ask,3,SL,Ask+25*Point,"My comment",123,0,Red);
```

标点操作符
从左到右的表达式用标点分开。所有表达式的计算是从左至右的。结果类型和值相互吻合，说明表达式是正确的。参量列表可以作为范例被通过。

示例:

```
for(i=0,j=99; i<100; i++,j--) Print(数组[i][j]);
```

## Precedence rules 优先规则

下面是从上到下的运算优先规则，优先级高的将先被运算。

| | | |
|---|---|---|
| () | 函数调用 | 从左到右 |

[]　　数组元素参考
!　　真假运算符　　从右到左
-　　改变运算符
++　　增量
--　　减量
~　　位逻辑运算符
&　　位逻辑运算符 AND　　从左到右
|　　位逻辑运算符 OR
^　　位逻辑运算符 OR
<<　　左移
>>　　右移
*　　乘法　　从左到右
/　　除法
%　　百分比
+　　加法　　从左到右
-　　减法
<　　小于　　从左到右
<=　　小于等于
>　　大于
>=　　大于等于
==　　等于
!=　　不等于
||　　逻辑 OR　　从左到右
&&　　逻辑 AND　　从左到右
=　　值　　从右到左
+=　　加法值
-=　　减法值
*=　　乘法值
/=　　除法值
%=　　百分比值
>>=　　右移值
<<=　　左移值
&=　　位逻辑运算符 AND 值
|=　　位逻辑运算符 OR 值
^=　　位逻辑运算符 OR 值
,　　逗号　　从左到右

插入语会显示所优先执行的运算

注意:在 MQL4 程序中执行优先运算不同于在 C 语言范围内的运算。

# Operators 操作符

语言操作符必须对执行完成任务的一些运算法操作进行描述。程序本身是这样的序列语句 。语句逐个随后以分号分离。

一个语句能占领一条或几条线。二个或更多语句可能位于同样线。单独执行命令的语句(if, if-else, switch, while and for) 可以相互插入

示例:

```
if(Month() == 12)
   if(Day() == 31) Print("新年快乐!");
```

## Compound operator 复合操作符

一个复合操作符有一个(一个区段)和由一个或多个任何类型的操作符组成的的附件{}. 每个表达式使用分号作为结束(;)。
示例:

```
if(x==0)
   {
    Print("无效位置  x=",x);
    return;
   }
```

## Expression operator 表达式操作符

任何以分号(;)结束的表达式都被视为是一个操作符。这里是一些表达式操作符得范例:
**称号运算符:**

```
Identifier=expression;
x=3;
y=x=3; //  错误
```

称号运算符在表达式操作符中只限一次使用。
**函数调用运算符:**

```
Function_name(argument1,..., argumentN);
   FileClose(file);
```

**空运算符:**
它是由分号(;)组成并且使用在一个检测运算符中。

## Break operator 终止操作符

一个嵌入 操作符终止最近外部操作符 switch, while 或 for 的执行。在终止操作符之后给出检测操作符。这个操作符的目的之一：当中心值指定为变量时，操作符完成循环执行。
示例:

```
//  搜索第一个零元素
for(i=0;i<array_size;i++)
   if((array[i]==0)
```

```
    break;
```

## Continue operator 继续操作符

一个 继续操作符。 我们将其放在嵌套内的指定位置,用来在指定情况下跳过接下来的运算，直接跳入下一次的循环 while 或 for 操作符。操作符 嵌入 位置与此操作符相反。

示例:

```
//   总结数组非零元素
int func(int array[])
  {
    int array_size=ArraySize(array);
    int sum=0;
    for(int i=0;i<array_size; i++)
       {
        if(a[i]==0) continue;
        sum+=a[i];
       }
    return(sum);
  }
```

## Return operator 返回操作符

一个 return 将需要返回的结果放在 return 后面的()中。
示例:

```
int CalcSum(int x, int y)
  {
    return(x+y);
  }
```

在函数中省缺值类型被返回，此 return 操作符必须使用:

```
void SomeFunction()
  {
    Print("Hello!");
    return;       // 这个操作符被删除
  }
```

函数正确的震动表示 return 操作符执行。

## Conditional operator if-else 条件操作符

如果表达式为 true，操作符执行并按照操作符 1 给出的检测。如果表达式为 false，操作符 2 执行。

```
if (expression)
        operator1
else
        operator2
```
if 操作符 else 部分可能被忽略。 if 操作符忽略 else 部分，显示分歧可能会嵌入。这种情况下， else 位置在先前 if 操作符的最近部位，这样不会出现 else 部分。
示例:
```
//    else 部分提及到第二个 if 操作符:
   if(x>1)
   if(y==2) z=5;
   else         z=6;
//    else 部分提及到第一个 if 操作符:
if(x>l)
   {
     if(y==2) z=5;
   }
else              z=6;
//   嵌入操作符
if(x=='a')
   {
     y=1;
   }
else if(x=='b')
   {
     y=2;
     z=3;
   }
else if(x=='c')
   {
     y = 4;
   }
else Print("ERROR");
```
## Switch operator 跳转操作符

在 case 全部变量和相应表达式值检测的操作符之内比较常数表达式值。每一个 case 变量会在整数或常数表达式内标注。常数表达式不包含函数变量调用。switch 表达式操作符必须是整数类型。
```
switch(expression)
   {
     case constant: operators
     case constant: operators
        ...
     default: operators
```

```
  }
```
如果在 case 操作符等于表达式值，操作符 default 标签连接将会执行。此 default 变量无需在最后。如果相应表达式值和 default 变量没有获取，不会有任何执行。关键词 case 和常数被标注，并且 if 操作符执行 case 变量，程序将执行以下所有操作符直至 break 操作符生成。一个常数表达式的计算是在编译期间。在一个 switch 操作符内部存在两个相同值的常数。
示例:
```
switch(x)
  {
   case 'A':
        Print("CASE A");
        break;
   case 'B':
   case 'C':
        Print("CASE B or C");
        break;
   default:
        Print("NOT A, B or C");
        break;
  }
```

## Cycle operator while 循环操作符 while

如果表达式为 true， 操作符执行直至表达式变成 false。如果表达式为 false，将检测最近操作符。
```
while(expression)
  operator;
```
在操作符执行前，一个表达式值已经被指定。不过，如果开始表达式为 false，操作符根本不会执行。
示例:
```
while(k<n)
  {
   y=y*x;
   k++;
  }
```

## Cycle operator for 循环操作符 for

用表达式 1Expression 来定义初始变量,当表达式 2Expression2 为真的时候执行操作运算符,在每次循环结束后执行表达式 3 Expression3。如果 true, 运算符 for 将被执行。循环重复直至 Expression2 变为 false。如果 false，循环将会被中断并且检测运算符文本。稍候执行。
```
for (Expression1; Expression2; Expression3)
    operator;
```
此 for 运算符下列运算符成功：

```
Expression1;
while(Expression2)
  {
    operator;
    Expression3;
  };
```
使用 for(;)可以造成一个死循环如同 while(1)一样. 表达式 1 和表达式 3 都可以内嵌多个用逗号(,)分割的表达式。 <
示例:
```
for(x=1;x<=7;x++) Print(MathPower(x,2));
for(;;)
  {
    Print(MathPower(x,2));
    x++;
    if(x>10) break;
  }
for(i=0,j=n-l;i<n;i++,j--) a[i]=a[j];
```

# Functions 函数

函数是部分程序的一个名称，它可以在需要时从任何一个部分调用。 它是由定义分类返回值，名称，形式参量和合成运算符组成并执行的。通过的总数被限定在 64 个字符之内。
示例:
```
double                                  // 被返回值的类型
linfunc (double x, double a, double b) // 函数名称和参量列表
{
                                        // 合成运算符
    return (a + b);                 // 返回值
 }
```
"返回"运算符可以返回在这个运算符内表达式的值。如果需要，此表达式值可以转换为函数结果类型。函数没有返回的值必定是“省缺” 类型。
示例:
```
void errmesg(string s)
  {
    Print("错误: "+s);
  }
```
通过函数的参量可能存在由特定类型常数指定的默认值。
示例:
```
int somefunc(double a, double d=0.0001, int n=5, bool b=true, string s="passed string")
  {
    Print("需求参量  a=",a);
    Print("下列参量被传送: d=",d," n=",n," b=",b," s=",s);
    return (0);
  }
```

如果此默认值指定一个参量，那么所有的参量也必须存在默认值。
错误范例:

```
int somefunc(double a, double d=0.0001, int n, bool b, string s="passed string")
  {
  }
```

## Function call 函数调用

如果显示的文件没有描述， 它将考虑上下文的联系作为函数名称。
函数名称 (x1, x2,..., xn)
自变数(形式参量) 以值的形式通过。 计算每一个表达式 xl, . . . , xn 并将其值发送到函数。表达式计算命令值是被保证的。在执行系统测试数字和自变数类型期间会给出函数。 这种形式的函数调用被称作调用值。调用函数是一个通过函数返回的表达式的值。 描述函数类型必须相应类型返回的值。全球范围内程序的任何一个部分函数是被公布或描述的，即其他函数之外。在另外函数里，函数不能被公布或描述。
例如:

```
int start()
  {
   double some_array[4]={0.3, 1.4, 2.5, 3.6};
   double a=linfunc(some_array, 10.5, 8);
   //...
  }
double linfunc(double x[], double a, double b)
  {
   return (a*x[0] + b);
  }
```

函数的调用是默认参量，通过参量的列表是被限定的，但不是之前的第一默认参量。
例如:

```
void somefunc(double init,double sec=0.0001,int level=10);   // function prototype
somefunc();                              // 错误调用, 第一请求参量必须存在。
somefunc(3.14);                          // 正确调用
somefunc(3.14, 0.0002);                  // 正确调用
somefunc(3.14, 0.0002, 10);              // 正确调用
```

当我们调用一个函数时，不可以忽略参量，存在默认值：

```
somefunc(3.14, , 10);                    // 错误调用。第二参量被忽略。
```

## Special functions 特殊函数

在 MQL4 中存在三种预定义名称函数:
init() 在载入时调用，可以用此函数在开始自定义指标或者自动交易之前做初始化操作。
start() 是基本函数。对于智能交易，在下一个替克进入之后被调用。对于客户指标，在指标添加到图表之后，客户端开始(如果指标添加到图表)并且下一个替克进入之后，函数被调用。对于脚本，在脚本被添加到图表之后立即执行并初始化。如果在模板中不存在 start()函数，

模板 (智能交易，脚本或客户指标) 不能开启。
deinit() 当数据变动时触发，对于自定义指标或者自动交易的编程主要依靠此函数进行
预定义函数需要一些参量。不过，当这些参量被客户端调用时，外部没有参量提供。 start(), init()和 deinit()函数从模板的任何一点按照常规调用，等于其他函数。
不建议从 init()函数调用 start()函数或是执行交易业务，作为图表数据，市场开价格。模板的初始化会出现残缺。这时，init() 和 deinit() 函数必须尽可能结束运行。在调用 start()函数之前，尝试重新全面开启运行。

# Variables 变量

可变量必须在公开之前使用。 可变量必须拥有特殊的辨认名。相关可变量的定义描述会显示。
基本类型如下：
布尔数据 -布尔值的 true 和 false;
字串符数据 - 特殊字符串;
双精度数字 - 带有浮点双精度数字。
示例:

```
string MessageBox;
int       Orders;
double SymbolPrice;
bool      bLog;
```

附加类型:
颜色 为整数代表 RGB 颜色;
日期时间 为日期和时间， 起始时间从 1979 年 1 月上午 0.00 开始以秒数计算。
添加数据类型在输入参量的属性窗口方便查看。
示例:

```
datetime tBegin_Data     = D'2004.01.01 00:00';
color       cModify_Color = C'0x44,0xB9,0xE6';
```

数组
相同数列数据被标注序列。

```
int       a[50];            // 50 整数的一维数组
double m[7][50];       // 7 个数组的二维数组
                              //每一个由 50 个整数组成。
```

唯一整数可以是 数组指数。 不允许四唯数列。数组元素开始编号为 0 。 一个一维列阵的最后元素是 1 的数字比列阵大小。这就意味着, 请求数列的最后元素包括 50 个整数将出现作为 a[49 ] 。 维度被标注从 0 到维度大小-1. 一个二维数组的最后元素从示例将出现作为 m[6][49 ] 。
如果访问超出数列范围，执行系统将发生错误可能生成错误 ERR_ARRAY_INDEX_OUT_OF_RANGE，在 GetLastError()函数中可以得到。

# Local variables 局部变量

在任意的地方内可变量的公开是局部的。局部变量在公开的部分里是被限定的。 局部变量

可以由任意一个表示结果初始化。 每次函数的运行只可以初始化一个局部变量。局部变量储存在相应的存储器上。
示例:

```
int somefunc()
  {
   int ret_code=0;
   ....
   return(ret_code);
  }
```

## Formal parameters 形式变量

通过函数的变量 是局部的。范围是在作用块内。在作用之内正式变量的名称必须不同于其他外部定义变量和函数变量。 作用块内的正式变量值已经被赋予。
示例:

```
void func(int x[], double y, bool z)
  {
   if(y>0.0 && !z)
       Print(x[0]);
   ...
  }
```

正式参量可能由常数初始化。在这种情况下，初始化的值作为缺省值被考虑。参量，在旁边（intialized） ，必须初始化。
示例:

```
void func(int x, double y = 0.0, bool z = true)
  {
   ...
  }
```

这样作用显现时，初始化的参量可能被省去，缺省值会代替它们。
示例:

```
func(123, 0.5);
```

MQL4 资料库功能在外部函数变量 模块之内无法有默认值初始化。
参量值通过。在任何情况下，变量里布局的修改将不会显示在功能板块内。 它是可以通过数列作为参量。但是，为了数列可以作为变量，需要改变它的数列元素。
它是还可能通过参量参考。在这种情况下，修改的这样参量将被显示在对应的变量。 数组元素无法参考通过。参量可能只在一个模块参考通过，数据库不提供。
示例:

```
void func(int& x, double& y, double& z[])
  {
   double calculated_tp;
   ...
   for(int i=0; i<OrdersTotal(); i++)
     {
      if(i==ArraySize(z))            break;
```

```
        if(OrderSelect(i)==false) break;
        z[i]=OrderOpenPrice();
       }
    x=i;
    y=calculated_tp;
  }
```

数组可以通过参考通过， 全部改变会在数列来源内显示。 不同于简单的参量, 数组可以进入数据库...
以缺省值参量通过无法初始化。
最大参量不可以超过 64 个。

## Static variables 静态变量

"静止" 记忆被称作静态变量。在数据类型之前指定成分"静止" 被公开。
示例:

```
int somefunc()
  {
   static int flag=10;
   ....
   return(flag);
  }
```

静态变量被存放在永久记忆里,在函数退出后静态变量不会丢失。所有在同一板块内（除正式变量作用外），可能作为静止变量定义。 静态变量可以由相对应的类型常数初始化。 与局部变量不同。如果没有明确地初始化, 静态变量初始化以零。 静态变量在"init()" 函数之前只可应用一次。

## Global variables 全局变量

整体变量作为函数被定义在相同水平，即，不可以局部使用。
示例:

```
int GlobalFlag=10;     // 整体变量
int start()
  {
   ...
  }
```

整体变量的范围是整个程序。整体变量在所有程序内是被定义的。如果它的值没有被定义，初始化值为零。 整体变量只对于相应的常数初始化。 整体变量只可以在 init()函数操作之前一次性初始化。
注解: 变量在整体变量的水平位上不能够与客户端 GlobalVariable...() 函数混淆。

## Defining extern variables 外部定义变量

外部定义的可变量。 在数据类型公布之前指定外部变量。
示例:

```
extern double InputParameter1 = 1.0;
extern color    InputParameter2 = red;
int init()
  {
    ...
  }
```

确定从外部程序输入的变量， 会直接显现输入数据窗口。数列本身不能作为外部变量。

## Initialization of variables 初始化变量

任何情况定义可变物可以初始化。如果它的原始值未被限定，任何可变物初始化为零(0)。 整体变量和静态变量的初始化由相应的常数进行。
整体变量和静态变量只能一次性初始化。局部变量的初始化与相应的调动进行。
示例:

```
int       n            = 1;
double p               = MarketInfo(Symbol(),MODE_POINT);
string s             = "hello";
double f[]          = { 0.0, 0.236, 0.382, 0.5, 0.618, 1.0 };
int       a[4][4] = { 1, 1, 1, 1,    2, 2, 2, 2,    3, 3, 3, 3,    4, 4, 4, 4 };
```

数组元素值列表必须被附寄在括号内。初始化省去的值被考虑为零。 如果初始化的数组大小不被定义，它将由编译器定义 。多维数组由一个一维序列，即序列初始化没有另外的括号。所有数列，只能以常数初始化。

## External functions definition 外部函数的定义

类型外在作用被定义在程序的其它组分必须明确地被描述。 缺乏这样定义也许导致错误在程序期间的编辑、联结，或施行。当描述一个外在对象，主题词进口必须被使用以在模块的参考。
示例:

```
#import "user32.dll"
   int         MessageBoxA(int hWnd ,string szText,string szCaption,int nType);
   int         SendMessageA(int hWnd,int Msg,int wParam,int lParam);
#import "lib.ex4"
   double    round(double value);
#import
```

进口可能被使用容易地描述作用叫从外在 DLLs 或编写 EX4 图书馆。
尖对可变物可能通过对进口的 dll 作用。串类型的数据 被通过作为尖对对应的记忆块(你应该记住串数据的内部表示法包括二份: 记忆块长度和记忆阻拦尖) 。如果有需要通过数据内

部或双重型，那么对应的型的一维一些应该参考通过作为参量。
示例:

```
#import "some_lib.dll"
   void       PassIntegerByref(int& OneInt[]);
#import
int start()
   {
    int array[1];
//...
    PassIntegerByref(array);
    Print(array[0]);
//...
   }
```

# Preprocessor 预处理

预处理程序是一个特殊 MQL4 的子程序，在程序执行之前预先准备的程序源代码。
预处理程序会尽可能地读取源代码。代码的结构可能包括 MQL4 程序源代码的特殊文件。
对于读取的代码尽可能地按照具体常数分配储存。
预处理程序允许 MQL4 程序参量指定。
如果# 标志被使用在程序的第一线，这条线是预处理程序方针。预处理程序方向末端以换行字符结尾。

# Constant declaration 常量声明

使用#define 定义常数可以在程序中指定货币对字串符并且定义货币对名称或货币对常数。稍候，编辑器会按照相应的字串符名称还原所有显示。事实上，这些名称可以由任意数组文本替换:

```
#define identifier value
```

此常数识别符符合变量名称的规则 值可以是以下任意类型:

```
#define ABC                 100
#define PI                  0.314
#define COMPANY_NAME "MetaQuotes Software Corp."
...
void ShowCopyright()
   {
    Print("版权所有 © 2001-2007, ",COMPANY_NAME);
    Print("http://www.metaquotes.net");
   }
```

# Controlling compilation 编译控制

每个 MQL4 程序允添加以#property 名称特殊的参量来帮助客户端服务。这是一个内设指标。
#property 识别值

| 常数 | 类型 | 描述 |
|---|---|---|
| link | string | 公司网站的相关连接 |
| copyright | string | 公司名称 |
| stacksize | int | 栈式储存器大小 |
| library | | 资料库;查看任何可出现的功能错误 |
| indicator_chart_window | void | 在图表窗口显示指标 |
| indicator_separate_window | void | 在指定窗口显示指标 |
| indicator_buffers | int | 对于指标计算的数字，最大为 8 |
| indicator_minimum | double | 在指标窗口下端 |
| indicator_maximum | double | 在指标窗口的上端 |
| indicator_colorN | color | 在 1 和 8 之间显示线的颜色 |
| indicator_widthN | int | 在 1 和 8 之间显示线的宽度 |
| indicator_styleN | int | 在 1 和 8 之间显示线的风格 |
| indicator_levelN | double | 在客户指标窗口 1 和 8 之间 N 的水平 |
| indicator_levelcolor | color | 水平线颜色 |
| indicator_levelwidth | int | 水平线宽度 |
| indicator_levelstyle | int | 水平线风格 |
| show_confirm | void | 在脚本运行之前显示确认 |
| show_inputs | void | 在脚本运行之前显示它的属性和确认 |

示例:

```
#property link          "http://www.metaquotes.net"
#property copyright     "MetaQuotes Software Corp."
#property library
#property stacksize     1024
```

在执行模板设定时，编译器将会写入值。

# Including of files 包含文件

#include 命令可以放置到程序的任意部分，但是通常所有文件的源代码被统一放置。调用格式;

```
#include <file_name>
```

#include "file_name";
示例:
#include <WinUser32.mqh>
#include "mylib.mqh"
对于 WinUser32.mqh.文件内容预处理程序还原线。三角括号表示 WinUser32.mqh 文件将会从默认目录调用(通常默认目录 terminal_directory\experts\include)。不需要搜索当前目录。如果载开盘价栏内文件名称未锁，搜索将在当前目录中执行(加载的源代码主文件)。 不需要搜索标准目录。

## Importing of functions 导入功能

函数从 MQL4 编译模板 (*.ex4 文件) 和执行系统文件模板(*.dll 文件)通过。模板名称被指定在#import 指令中。来自输入函数和通过参量的兵役数据需要带有完整的描述部分。函数描述会立即按照#import "模板"名称执行。新的#import 命令完成引入输入函数描述部分。

```
#import "file_name"
      func1 define;
      func2 define;
      ...
      funcN define;
#import
```

输入函数必须有自己的名称。相同名称的函数无法从不同的模块同时引入。引入的函数名不能与那些内部函数融合。
因为引入函数是在模块外面被编写，编译器无法检查通过参量的正确性。这就是为什么，避免运行错误，它是必要精确地公开命令的原因。在参量引入函数(从 EX4 和从 DLL 模块)后，没有任何值。
示例:

```
#import "user32.dll"
    int       MessageBoxA(int hWnd, string lpText, string lpCaption, int uType);
#import "stdlib.ex4"
    string ErrorDescription(int error_code);
    int       RGB(int red_value, int green_value, int blue_value);
    bool      CompareDoubles(double number1, double number2);
    string DoubleToStrMorePrecision(double number, int precision);
    string IntegerToHexString(int integer_number);
#import "Expert 示例.dll"
    int       GetIntValue(int);
    double GetDoubleValue(double);
    string GetStringValue(string);
    double GetArrayItemValue(double arr[], int, int);
    bool      SetArrayItemValue(double& arr[], int,int, double);
    double GetRatesItemValue(double rates[][6], int, int, int);
    int       SortStringArray(string& arr[], int);
    int       ProcessStringArray(string& arr[], int);
#import
```

在 mql4 程序执行期间引入输入函数，需要使用稍后安装。这就意味着直到调用输入函数，相应模板(ex4 或 dll)将不会进行加载。

不建议使用 Drive:\Directory\FileName.Ext 为文件名安装。MQL4 资料库会从 terminal_dir\experts\libraries 文件夹中卸下。如果没有发现资料库，则没有可能从 terminal_dir\experts 文件夹中卸下。

# Standard constants 标准常数

为了简化编写程序并使其程序文本更加方便，在 MQL4 中预定义了标准变量。
int 类型的标准变量与 macro substitutions 类似 。
此变量是按照用途分组的。

## Series arrays 系列数组

系列数组识别符在 ArrayCopySeries(), iHighest() 和 iLowest()函数中使用。
可以是以下任意值:

| 常数 | 值 | 描述 |
|---|---|---|
| MODE_OPEN | 0 | 开价 |
| MODE_LOW | 1 | 最低价 |
| MODE_HIGH | 2 | 最高价 |
| MODE_CLOSE | 3 | 关单价 |
| MODE_VOLUME | 4 | 应用在 iLowest()和 iHighest()函数中的成交量 |
| MODE_TIME | 5 | 应用在 ArrayCopySeries()函数中的开柱时间 |

## Timeframes 图表周期时间

图表的时间周期。可以是以下任意值:

| 常数 | 值 | 描述 |
|---|---|---|
| PERIOD_M1 | 1 | 1 分钟 |
| PERIOD_M5 | 5 | 5 分钟 |
| PERIOD_M15 | 15 | 15 分钟 |
| PERIOD_M30 | 30 | 30 分钟 |
| PERIOD_H1 | 60 | 1 小时 |
| PERIOD_H4 | 240 | 4 小时 |
| PERIOD_D1 | 1440 | 每天 |
| PERIOD_W1 | 10080 | 每星期 |
| PERIOD_MN1 | 43200 | 每月 |
| 0 (zero) | 0 | 在图表中使用的时间周期 |

# Trade operations 交易操作

对于 OrderSend()函数的交易类型。可以是以下任意值:

| 常数 | 值 | 描述 |
|---|---|---|
| OP_BUY | 0 | 买仓 |
| OP_SELL | 1 | 卖仓 |
| OP_BUYLIMIT | 2 | 买挂单交易 |
| OP_SELLLIMIT | 3 | 卖挂单交易 |
| OP_BUYSTOP | 4 | 买停挂单交易 |
| OP_SELLSTOP | 5 | 卖停挂单交易 |

# Price constants 价格常数

提供的价格常数，它可以是以下的任意值:

| 常数 | 值 | 描述 |
|---|---|---|
| PRICE_CLOSE | 0 | 平仓价 |
| PRICE_OPEN | 1 | 开仓价 |
| PRICE_HIGH | 2 | 最高价 |
| PRICE_LOW | 3 | 最低价 |
| PRICE_MEDIAN | 4 | 中间价(high+low)/2. |
| PRICE_TYPICAL | 5 | 典型价格 (high+low+close)/3. |
| PRICE_WEIGHTED | 6 | 价格 (high+low+close+close)/4. |

# MarketInfo 市场信息识别符

市场信息识别符，使用 MarketInfo()函数。
可以是以下任意值:

| 常数 | 值 | 描述 |
|---|---|---|
| MODE_LOW | 1 | 价格最低日。 |
| MODE_HIGH | 2 | 价格最高日。 |
| MODE_TIME | 5 | 最后进入替克的时间 (服务器显示时间)。 |
| MODE_BID | 9 | 最后进入的买价。对于、当前货币对预定变量存储的 买价。 |
| MODE_ASK | 10 | 最后进入的卖价。对于、当前货币对预定变量存储的 卖价。 |

| | | |
|---|---|---|
| MODE_POINT | 11 | 当前价位的大小点。对于当前货币对预定变量储存的点。 |
| MODE_DIGITS | 12 | 在货币对值中小数点后的计数点。对于当前货币对预定变量存储的小数点计数。 |
| MODE_SPREAD | 13 | 差价点。 |
| MODE_STOPLEVEL | 14 | 停止水平点。 |
| MODE_LOTSIZE | 15 | 基本货币的标准手大小。 |
| MODE_TICK 值 | 16 | 在存款货币中的替克值。 |
| MODE_TICKSIZE | 17 | 在当前报价中的替克大小。 |
| MODE_SWAPLONG | 18 | 看涨仓位掉期。 |
| MODE_SWAPSHORT | 19 | 卖空仓位掉期。 |
| MODE_STARTING | 20 | 市场开始日期 (通常用作将来)。 |
| MODE_EXPIRATION | 21 | 市场时间周期 (通常用作将来)。 |
| MODE_TRADEALLOWED | 22 | 交易允许货币对。 |
| MODE_MINLOT | 23 | 最小允许标准手数。 |
| MODE_LOTSTEP | 24 | 改变标准手步骤。 |
| MODE_MAXLOT | 25 | 最大允许标准手数。 |
| MODE_SWAPTYPE | 26 | 掉期计算方法. 0 - 点; 1 -基本货币对; 2 - 兴趣; 3 - 货币保证金。 |
| MODE_PROFITCALCMODE | 27 | 赢利计算模式 0 - Forex; 1 - CFD; 2 - Futrues。 |
| MODE_MARGINCALCMODE | 28 | 保证金计算模式. 0 - Forex; 1 - CFD; 2 - Futrues; 3 - CFD for indices。 |
| MODE_MARGININIT | 29 | 对于 1 各标准手的初始保证金需求。 |
| MODE_MARGINMAINTENANCE | 30 | 对于 1 各标准手开仓的保证金。 |
| MODE_MARGINHEDGED | 31 | 对于 1 标准手的护盘保证金。 |
| MODE_MARGINREQUIRED | 32 | 对于购买一个标准手开仓的自由保证金。 |
| MODE_FREEZELEVEL | 33 | 冻结定单水平点。如果执行的价格在冻结水平点范围内，定单将会被注销或关闭。 |

# Drawing styles 画线风格

SetIndexStyle()函数画线风格的列举。
可以是以下任意值:

| 常数 | 值 | 描述 |
|---|---|---|

| | | |
|---|---|---|
| DRAW_LINE | 0 | 画线 |
| DRAW_SECTION | 1 | 线条 |
| DRAW_HISTOGRAM | 2 | 画柱状图 |
| DRAW_ARROW | 3 | 画箭头(货币对)。 |
| DRAW_ZIGZAG | 4 | 在添加的缓冲器之间画线条 |
| DRAW_NONE | 12 | 没有画线 |

画线风格。有效宽度为 1.。可以是以下任意值：

| 常数 | 值 | 描述 |
|---|---|---|
| STYLE_SOLID | 0 | 实线 |
| STYLE_DASH | 1 | 断线 |
| STYLE_DOT | 2 | 点线 |
| STYLE_DASHDOT | 3 | 断线与点线交替. |
| STYLE_DASHDOTDOT | 4 | 断线与双点线交替 |

# Arrow codes 预定义箭头

预定义箭头列举。箭头代码。 可以是以下的任意值：

| 常数 | 值 | 描述 |
|---|---|---|
| SYMBOL_THUMBSUP | 67 | 大拇指向上 (👍). |
| SYMBOL_THUMBSDOWN | 68 | 大拇指向下 (👎). |
| SYMBOL_ARROWUP | 241 | 箭头向上 (⇧). |
| SYMBOL_ARROWDOWN | 242 | 箭头向下 (⇩). |
| SYMBOL_STOPSIGN | 251 | 禁止货币对(✖). |
| SYMBOL_CHECKSIGN | 252 | 检验货币对 (✓). |

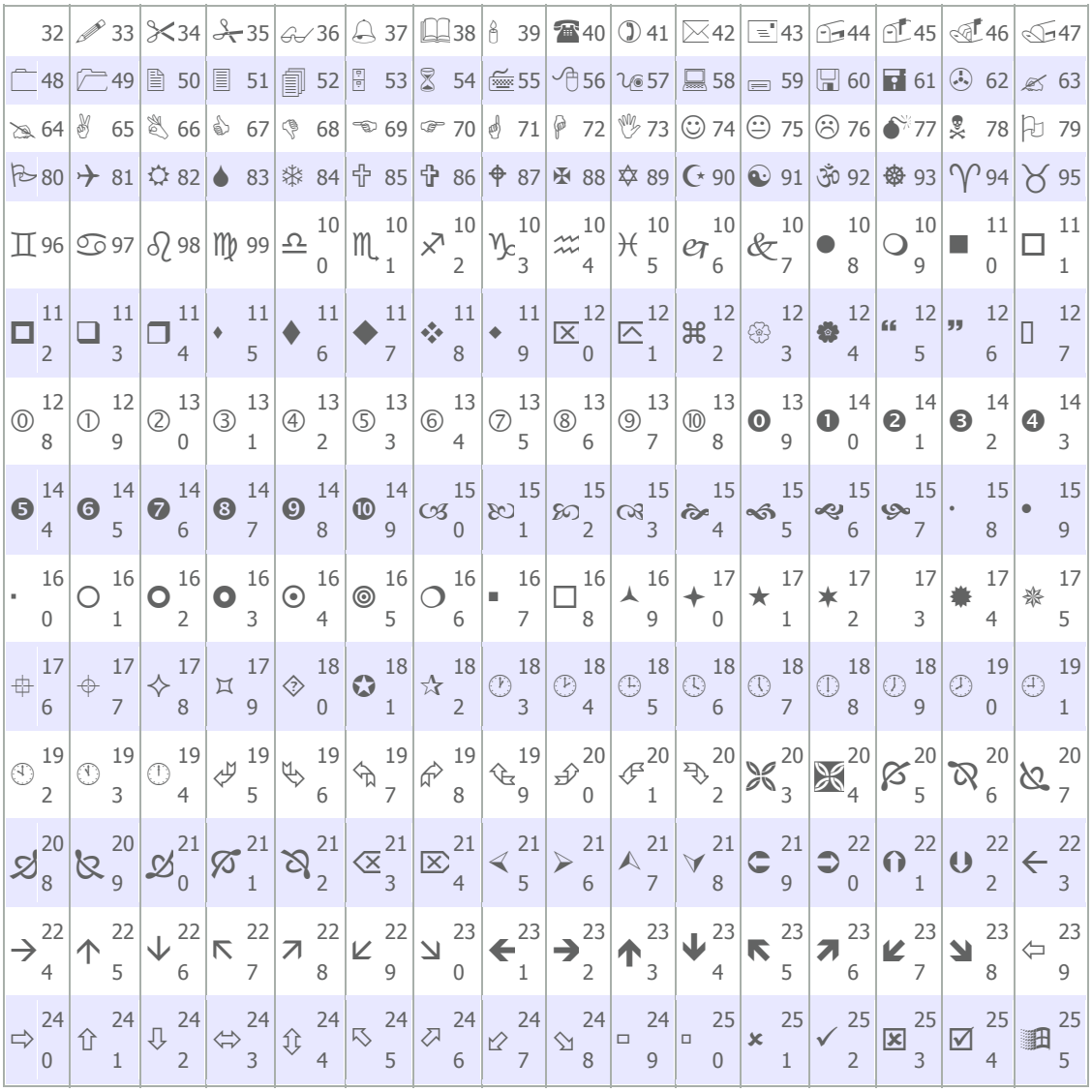
对于价格和时间的特殊箭头代码。可以是以下任意值:

| 常数 | 值 | 描述 |
|---|---|---|
| | 1 | 右上转箭头 (↱). |
| | 2 | 右下转箭头(↳). |
| | 3 | 左指向三角 (◀). |
| | 4 | 破折号(–). |

| | | |
|---|---|---|
| SYMBOL_LEFTPRICE | 5 | 价格左侧标签 |
| SYMBOL_RIGHTPRICE | 6 | 价格右侧标签 |

# Wingdings 宋体

使用在箭头对象的 Wingdings 铅字符。




# Web colors 颜色常数

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Black | DarkGreen | DarkSlateGray | Olive | Green | Teal | Navy | Purple |
| Maroon | Indigo | MidnightBlue | DarkBlue | DarkOliveGreen | SaddleBrown | ForestGreen | OliveDrab |
| SeaGreen | DarkGoldenrod | DarkSlateBlue | Sienna | MediumBlue | Brown | DarkTurquoise | DimGray |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| LightSeaGreen | DarkViolet | FireBrick | MediumVioletRed | MediumSeaGreen | Chocolate | Crimson | SteelBlue |
| Goldenrod | MediumSpringGreen | LawnGreen | CadetBlue | DarkOrchid | YellowGreen | LimeGreen | OrangeRed |
| DarkOrange | Orange | Gold | Yellow | Chartreuse | Lime | SpringGreen | Aqua |
| DeepSkyBlue | Blue | Magenta | Red | Gray | SlateGray | Peru | BlueViolet |
| LightSlateGray | DeepPink | MediumTurquoise | DodgerBlue | Turquoise | RoyalBlue | SlateBlue | DarkKhaki |
| IndianRed | MediumOrchid | GreenYellow | MediumAquamarine | DarkSeaGreen | Tomato | RosyBrown | Orchid |
| MediumPurple | PaleVioletRed | Coral | CornflowerBlue | DarkGray | SandyBrown | MediumSlateBlue | Tan |
| DarkSalmon | BurlyWood | HotPink | Salmon | Violet | LightCoral | SkyBlue | LightSalmon |
| Plum | Khaki | LightGreen | Aquamarine | Silver | LightSkyBlue | LightSteelBlue | LightBlue |
| PaleGreen | Thistle | PowderBlue | PaleGoldenrod | PaleTurquoise | LightGray | Wheat | NavajoWhite |
| Moccasin | LightPink | Gainsboro | PeachPuff | Pink | Bisque | LightGoldenrod | BlanchedAlmond |
| LemonChiffon | Beige | AntiqueWhite | PapayaWhip | Cornsilk | LightYellow | LightCyan | Linen |
| Lavender | MistyRose | OldLace | WhiteSmoke | Seashell | Ivory | Honeydew | AliceBlue |
| LavenderBlush | MintCream | Snow | White | | | | |

# Indicator lines 指标线

指标线识别在 iMACD(), iRVI() 和 iStochastic() 指标中使用。
可以是以下的任意值:

| 常数 | 值 | 描述 |
|---|---|---|
| MODE_MAIN | 0 | 基本指标线。 |
| MODE_SIGNAL | 1 | 信号线。 |

指标线识别符在 iADX()指标中使用。

| 常数 | 值 | 描述 |
|---|---|---|
| MODE_MAIN | 0 | 基本指标线。 |

| | | |
|---|---|---|
| MODE_PLUSDI | 1 | +DI 指标线。 |
| MODE_MINUSDI | 2 | -DI 指标线。 |

指标线识别符在 iBands(), iEnvelopes(), iEnvelopesOnArray(), iFractals() 和 iGator() 指标中使用。

| 常数 | 值 | 描述 |
|---|---|---|
| MODE_UPPER | 1 | 上面线。 |
| MODE_LOWER | 2 | 下面线。 |

# Ichimoku Kinko Hyo

Ichimoku Kinko Hyo 识别符使用在 iIchimoku() 指标中作为请求数据代码调用。
可以为以下任意值:

| 常数 | 值 | 描述 |
|---|---|---|
| MODE_TENKANSEN | 1 | Tenkan-sen. |
| MODE_KIJUNSEN | 2 | Kijun-sen. |
| MODE_SENKOUSPANA | 3 | Senkou Span A. |
| MODE_SENKOUSPANB | 4 | Senkou Span B. |
| MODE_CHINKOUSPAN | 5 | Chinkou Span. |

# Moving Average methods 移动平均方法

移动平均计算在 iAlligator(), iEnvelopes(), iEnvelopesOnArray, iForce(), iGator(), iMA(), iMAOnArray(), iStdDev(), iStdDevOnArray(), iStochastic()指标中使用。
可以使以下任意值:

| 常数 | 值 | 描述 |
|---|---|---|
| MODE_SMA | 0 | 简单移动平均数 |
| MODE_EMA | 1 | 指数移动平均数 |
| MODE_SMMA | 2 | 平滑移动平均数 |
| MODE_LWMA | 3 | 线性移动平均数 |

# MessageBox 信息箱

The MessageBox() function return codes.
如果选择信箱中 Cancel 按键或是 ESC 取消选择，函数返回 IDCANCEL 值 。 如果信箱中不存在取消按键，按 ESC 无效。

注解：消息框返回的代码在 WinUser32.mqh 文件中。

| 常数 | 值 | 描述 |
| --- | --- | --- |
| IDOK | 1 | 选择**确定** 按钮. |
| IDCANCEL | 2 | 选择**取消**按钮. |
| IDABORT | 3 | 选择**中止**按钮. |
| IDRETRY | 4 | 选择**重试** 按钮. |
| IDIGNORE | 5 | 选择**忽略** 按钮. |
| IDYES | 6 | 选择**是** 按钮. |
| IDNO | 7 | 选择**否** 按钮. |
| IDTRYAGAIN | 10 | 选择**再次尝试**按钮. |
| IDCONTINUE | 11 | 选择**继续** 按钮. |

MessageBox 函数是一个具有特殊功能的对话框。可以和以下值结合并用。
在邮箱中按键显示的意义

| 常数 | 值 | 描述 |
| --- | --- | --- |
| MB_OK | 0x00000000 | 消息框中包含的一个按钮：确定 这是默认值. |
| MB_OKCANCEL | 0x00000001 | 消息框中包含的两个按钮：确定和取消. |
| MB_ABORTRETRYIGNORE | 0x00000002 | 消息框中包含的三个按钮：中止，重试和忽略. |
| MB_YESNOCANCEL | 0x00000003 | 消息框中包含的三个按钮：是，否和取消 |
| MB_YESNO | 0x00000004 | 消息框中包含的两个按钮：是和否 |
| MB_RETRYCANCEL | 0x00000005 | 消息框中包含的两个按钮：重试和取消 |
| MB_CANCELTRYCONTINUE | 0x00000006 | Windows 2000：消息框中包含的三个按钮：取消，重试，继续. 使用这个消息框类型代替 |

To display an icon in the message box, specify one of the following values.
要显示在消息框中的图标，指定下列值之一。

| 常数 | 值 | 描述 |
| --- | --- | --- |
| MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND | 0x00000010 | 禁止消息图标在邮箱内显示. |
| MB_ICONQUESTION | 0x00000020 | 问号图标出现在消息框内. |
| MB_ICONEXCLAMATION, MB_ICONWARNING | 0x00000030 | 感叹号图标出现在消息框内. |
| MB_ICONINFORMATION, MB_ICONASTERISK | 0x00000040 | 图标组成的短信息显示在消息框内. |

在消息框内显示的图标是以下值之一。

| 常数 | 值 | 描述 |
| --- | --- | --- |
| MB_DEFBUTTON1 | 0x00000000 | 第一个按钮为默认。 MB_DEFBUTTON1 是默认的，MB_DEFBUTTON2, MB_DEFBUTTON3, MB_DEFBUTTON4 是指定的. |
| MB_DEFBUTTON2 | 0x00000100 | 第二个按钮为默认. |
| MB_DEFBUTTON3 | 0x00000200 | 第三个按钮为默认. |
| MB_DEFBUTTON4 | 0x00000300 | 第四个按钮为默认. |

MessageBox() 功能被指定在 WinUser32.mqh 文件内， 这就是为什么这个文件程序必须通过 #include <WinUser32.mqh>。不是所有的都可以在列表中列出。详细细节请参阅 Win32 API。

# Object types 对象类型

定单类型常数在 ObjectCreate(), ObjectsDeleteAll() 和 ObjectType() 函数中使用。可以使以下任意值:
对象可能有 1-3 个坐标。

| 常数 | 值 | 描述 |
| --- | --- | --- |
| OBJ_VLINE | 0 | 垂直线。使用第一坐标部分时间。 |
| OBJ_HLINE | 1 | 水平线。使用第一坐标部分价格。 |
| OBJ_TREND | 2 | 趋势线。 应用 2 个坐标。 |
| OBJ_TRENDBYANGLE | 3 | 趋势角度。应用 1 个坐标。应用 ObjectSet() 功能设置线的角度。 |
| OBJ_REGRESSION | 4 | 回归。应用头两个坐标的时间部分。 |
| OBJ_CHANNEL | 5 | 通道。应用 3 个坐标。 |
| OBJ_STDDEVCHANNEL | 6 | 标准偏离通道。应用头两个坐标的时间部分。 |
| OBJ_GANNLINE | 7 | 甘氏线。应用 2 个坐标，但第二个坐标的价格部分。 |
| OBJ_GANNFAN | 8 | 甘氏扇形线。应用 2 个坐标，但第二个坐标的价格部分。 |
| OBJ_GANNGRID | 9 | 甘氏网格线。应用 2 个坐标，但第二个坐标的价格部分 |
| OBJ_FIBO | 10 | 斐波纳契撤回。应用 2 个坐标。 |
| OBJ_FIBOTIMES | 11 | 斐波纳契时间周期线。应用 2 个坐标。 |
| OBJ_FIBOFAN | 12 | 斐波纳契扇形线。应用 2 个坐标。 |
| OBJ_FIBOARC | 13 | 斐波纳契弧线。应用 2 个坐标。 |
| OBJ_EXPANSION | 14 | 斐波纳契扩展。应用 3 个坐标。 |
| OBJ_FIBOCHANNEL | 15 | 斐波纳契通道。应用 3 个坐标。 |
| OBJ_RECTANGLE | 16 | 矩形。应用 2 个坐标。 |

| | | |
|---|---|---|
| OBJ_TRIANGLE | 17 | 三角形。应用 3 个坐标。 |
| OBJ_ELLIPSE | 18 | 椭圆形。应用 2 个坐标。 |
| OBJ_PITCHFORK | 19 | 安德鲁分叉线。应用 3 个坐标。 |
| OBJ_CYCLES | 20 | 周期。应用 2 个坐标。 |
| OBJ_TEXT | 21 | 文本。应用 1 坐标。 |
| OBJ_ARROW | 22 | 字行。应用 1 个坐标。 |
| OBJ_LABEL | 23 | 文本标签。应用 1 个坐标。 |

# Object properties 对象属性

对象值函数同 ObjectGet() 和 ObjectSet() 功能一起使用。它可能是以下的任意值:

| 常数 | 值 | 类型 | 描述 |
|---|---|---|---|
| OBJPROP_TIME1 | 0 | datetime | 日期时间值设置为第一协调时间部分。 |
| OBJPROP_PRICE1 | 1 | double | 双重值设置为第一协调价格部分。 |
| OBJPROP_TIME2 | 2 | datetime | 日期时间值设置为第二协调时间部分。 |
| OBJPROP_PRICE2 | 3 | double | 双重值设置为第二协调价格部分。 |
| OBJPROP_TIME3 | 4 | datetime | 日期时间值设置为第三协调时间部分。 |
| OBJPROP_PRICE3 | 5 | double | 双重值设置为第三协调价格部分。 |
| OBJPROP_COLOR | 6 | color | 颜色值设置对象颜色。 |
| OBJPROP_STYLE | 7 | int | STYLE_SOLID, STYLE_DASH, STYLE_DOT, STYLE_DASHDOT, STYLE_DASHDOTDOT 常数中的一个设置为对象线风格。 |
| OBJPROP_WIDTH | 8 | int | 设置对象线宽度的整数值。可以从 1 到 5。 |
| OBJPROP_BACK | 9 | bool | 设定对象背景的布尔值。 |
| OBJPROP_RAY | 10 | bool | 设定对象射线的布尔值。 |
| OBJPROP_ELLIPSE | 11 | bool | 设置椭圆状的弧形布尔值。 |
| OBJPROP_SCALE | 12 | double | 设置对象属性的双重值。 |
| OBJPROP_ANGLE | 13 | double | 在级别上设置对象属性的双重值。 |
| OBJPROP_ARROWCODE | 14 | int | 设置对象属性箭头代码的整数值和箭头计数 。 |
| OBJPROP_TIMEFRAMES | 15 | int | 设置对象属性的时间范围一个值或者可见性对象常数的组合值。 |
| OBJPROP_DEVIATION | 16 | double | 为标准离差对象设定双重值的离差属性。 |

| | | | |
|---|---|---|---|
| OBJPROP_FONTSIZE | 100 | int | 对于对象文本字体大小设定整数值。 |
| OBJPROP_CORNER | 101 | int | 对标记对象设定整数值的固定装置角。必须是从 0-3。 |
| OBJPROP_XDISTANCE | 102 | int | 在像点设定整数值固定装置 X 间隔对象。 |
| OBJPROP_YDISTANCE | 103 | int | 在像点设定整数值固定装置 Y 间隔对象。 |
| OBJPROP_FIBOLEVELS | 200 | int | 设置斐波纳契对象水平数为整数值。可以从 0 到 32 。 |
| OBJPROP_LEVELCOLOR | 201 | color | 设置对象水平颜色线的颜色值。 |
| OBJPROP_LEVELSTYLE | 202 | int | STYLE_SOLID, STYLE_DASH, STYLE_DOT, STYLE_DASHDOT, STYLE_DASHDOTDOT 常数中的一个设置为对象线风格。 |
| OBJPROP_LEVELWIDTH | 203 | int | 设置对象线宽度的整数值。可以从 1 到 5。 |
| OBJPROP_FIRSTLEVEL+*n* | 210+n | int | 斐波纳契水平函数是设置 n 的水平函数。可以是从 0 到 31 。 |

# Object visibility

时间范围将在货币对处显示。在 ObjectSet()应用函数中设置 OBJPROP_TIMEFRAMES 属性。

| 常数 | 值 | 描述。 |
|---|---|---|
| OBJ_PERIOD_M1 | 0x0001 | 对象只在 1 分钟图表中显示。 |
| OBJ_PERIOD_M5 | 0x0002 | 对象只在 5 分钟图表中显示。。 |
| OBJ_PERIOD_M15 | 0x0004 | 对象只在 15 分钟图表中显示。 |
| OBJ_PERIOD_M30 | 0x0008 | 对象只在 30 分钟图表中显示。 |
| OBJ_PERIOD_H1 | 0x0010 | 对象只在 1 小时图表中显示。 |
| OBJ_PERIOD_H4 | 0x0020 | 对象只在 4 小时图表中显示。 |
| OBJ_PERIOD_D1 | 0x0040 | 对象只在天图表中显示。 |
| OBJ_PERIOD_W1 | 0x0080 | 对象只在星期图表中显示。 |
| OBJ_PERIOD_MN1 | 0x0100 | 对象只在月图表中显示。 |
| OBJ_ALL_PERIODS | 0x01FF | 对象在所有时间周期图表中显示。 |
| NULL | 0 | 对象在所有时间周期图表中显示。 |
| EMPTY | -1 | 在所有时间周期图表中显示。 |

# Uninitialize reason codes 撤销初始化原因代码

| 常数 | 值 | 描述 |
|---|---|---|

| | | |
|---|---|---|
| | 0 | 脚本独立执行完成 |
| REASON_REMOVE | 1 | 从图表中删除。 |
| REASON_RECOMPILE | 2 | 重新编译交易。 |
| REASON_CHARTCHANGE | 3 | 在图表上改变货币对和时间周期。 |
| REASON_CHARTCLOSE | 4 | 关闭图表 |
| REASON_PARAMETERS | 5 | 用户改变了输入参量 |
| REASON_ACCOUNT | 6 | 其他账户已激活 |

# Special constants 特别常数

特别常数使用于指定参量和变量状态。可以是以下任意值:

| 常数 | 值 | 描述 |
|---|---|---|
| NULL | 0 | 指示空状态的字行。 |
| EMPTY | -1 | 指示空状态的参量。 |
| EMPTY_值 | 0x7FFFFFFF | 指示自定义空值。 |
| CLR_NONE | 0xFFFFFFFF | 指示空状态的颜色。 |
| WHOLE_ARRAY | 0 | 应用数组功能。指示数组将被处理。 |

# Error codes 错误代码

此 GetLastError()函数返回代码。错误代码被指定在 stderror.mqh 文件里。打印文本信息使用在 stdlib.mqh 文件中指定错误描述()函数。

```
#include <stderror.mqh>
#include <stdlib.mqh>
void SendMyMessage(string text)
  {
   int check;
   SendMail("some subject", text);
   check=GetLastError();
   if(check!=ERR_NO_ERROR) Print("Cannot send message, error: ",Error 描述(check));
  }
```

从服务器返回的错误代码

| 常数 | 值 | 描述 |
|---|---|---|
| ERR_NO_ERROR | 0 | 没有错误返回。 |
| ERR_NO_RESULT | 1 | 没有错误返回但结果不明。 |

| | | |
|---|---|---|
| ERR_COMMON_ERROR | 2 | 一般错误。 |
| ERR_INVALID_TRADE_PARAMETERS | 3 | 无效交易参量。 |
| ERR_SERVER_BUSY | 4 | 交易服务器繁忙。 |
| ERR_OLD_VERSION | 5 | 客户终端旧版本。 |
| ERR_NO_CONNECTION | 6 | 没有连接服务器。 |
| ERR_NOT_ENOUGH_RIGHTS | 7 | 没有权限。 |
| ERR_TOO_FREQUENT_REQUESTS | 8 | 请求过于频繁。 |
| ERR_MALFUNCTIONAL_TRADE | 9 | 交易运行故障。 |
| ERR_ACCOUNT_DISABLED | 64 | 账户禁止。 |
| ERR_INVALID_ACCOUNT | 65 | 无效账户 |
| ERR_TRADE_TIMEOUT | 128 | 交易超时。 |
| ERR_INVALID_PRICE | 129 | 无效价格。 |
| ERR_INVALID_STOPS | 130 | 无效停止。 |
| ERR_INVALID_TRADE_VOLUME | 131 | 无效交易量。 |
| ERR_MARKET_CLOSED | 132 | 市场关闭。 |
| ERR_TRADE_DISABLED | 133 | 交易被禁止。 |
| ERR_NOT_ENOUGH_MONEY | 134 | 资金不足。 |
| ERR_PRICE_CHANGED | 135 | 价格改变。 |
| ERR_OFF_QUOTES | 136 | 开价。 |
| ERR_BROKER_BUSY | 137 | 经纪繁忙。 |
| ERR_REQUOTE | 138 | 重新开价。 |
| ERR_ORDER_LOCKED | 139 | 定单被锁定。 |
| ERR_LONG_POSITIONS_ONLY_ALLOWED | 140 | 只允许看涨仓位。 |
| ERR_TOO_MANY_REQUESTS | 141 | 过多请求。 |
| ERR_TRADE_MODIFY_DENIED | 145 | 因为过于接近市场，修改否定。 |
| ERR_TRADE_CONTEXT_BUSY | 146 | 交易文本已满。 |
| ERR_TRADE_EXPIRATION_DENIED | 147 | 时间周期被经纪否定。 |
| ERR_TRADE_TOO_MANY_ORDERS | 148 | 开单和挂单总数已被经纪限定。 |

MQL4 运行错误代码

| 常数 | 值 | 描述 |
|---|---|---|

| | | |
|---|---|---|
| ERR_NO_MQLERROR | 4000 | 没有错误。 |
| ERR_WRONG_FUNCTION_POINTER | 4001 | 错误函数指示。 |
| ERR_ARRAY_INDEX_OUT_OF_RANGE | 4002 | 数组索引超出范围。 |
| ERR_NO_MEMORY_FOR_CALL_STACK | 4003 | 对于调用堆栈储存器函数没有足够内存。 |
| ERR_RECURSIVE_STACK_OVERFLOW | 4004 | 循环堆栈储存器溢出。 |
| ERR_NOT_ENOUGH_STACK_FOR_PARAM | 4005 | 对于堆栈储存器参量没有内存。 |
| ERR_NO_MEMORY_FOR_PARAM_STRING | 4006 | 对于字行参量没有足够内存。 |
| ERR_NO_MEMORY_FOR_TEMP_STRING | 4007 | 对于字行没有足够内存。 |
| ERR_NOT_INITIALIZED_STRING | 4008 | 没有初始字行。 |
| ERR_NOT_INITIALIZED_ARRAYSTRING | 4009 | 在数组中没有初始字串符。 |
| ERR_NO_MEMORY_FOR_ARRAYSTRING | 4010 | 对于数组没有内存。 |
| ERR_TOO_LONG_STRING | 4011 | 字行过长。 |
| ERR_REMAINDER_FROM_ZERO_DIVIDE | 4012 | 余数划分为零。 |
| ERR_ZERO_DIVIDE | 4013 | 零划分。 |
| ERR_UNKNOWN_COMMAND | 4014 | 不明命令。 |
| ERR_WRONG_JUMP | 4015 | 错误转换(没有常规错误)。 |
| ERR_NOT_INITIALIZED_ARRAY | 4016 | 没有初始数组。 |
| ERR_DLL_CALLS_NOT_ALLOWED | 4017 | 禁止调用 DLL 。 |
| ERR_CANNOT_LOAD_LIBRARY | 4018 | 数据库不能下载。 |
| ERR_CANNOT_CALL_FUNCTION | 4019 | 不能调用函数。 |
| ERR_EXTERNAL_CALLS_NOT_ALLOWED | 4020 | 禁止调用智能交易函数。 |
| ERR_NO_MEMORY_FOR_RETURNED_STR | 4021 | 对于来自函数的字行没有足够内存。 |
| ERR_SYSTEM_BUSY | 4022 | 系统繁忙 (没有常规错误)。 |
| ERR_INVALID_FUNCTION_PARAMSCNT | 4050 | 无效计数参量函数。 |
| ERR_INVALID_FUNCTION_PARAM 值 | 4051 | 无效参量值函数。 |
| ERR_STRING_FUNCTION_INTERNAL | 4052 | 字行函数内部错误。 |
| ERR_SOME_ARRAY_ERROR | 4053 | 一些数组错误。 |
| ERR_INCORRECT_SERIESARRAY_USING | 4054 | 应用不正确数组。 |
| ERR_CUSTOM_INDICATOR_ERROR | 4055 | 自定义指标错误。 |
| ERR_INCOMPATIBLE_ARRAYS | 4056 | 不协调数组。 |
| ERR_GLOBAL_VARIABLES_PROCESSING | 4057 | 整体变量过程错误。 |

| | | |
|---|---|---|
| ERR_GLOBAL_VARIABLE_NOT_FOUND | 4058 | 整体变量未找到。 |
| ERR_FUNC_NOT_ALLOWED_IN_TESTING | 4059 | 测试模式函数禁止。 |
| ERR_FUNCTION_NOT_CONFIRMED | 4060 | 没有确认函数 |
| ERR_SEND_MAIL_ERROR | 4061 | 发送邮件错误。 |
| ERR_STRING_PARAMETER_EXPECTED | 4062 | 字行预计参量。 |
| ERR_INTEGER_PARAMETER_EXPECTED | 4063 | 整数预计参量。 |
| ERR_DOUBLE_PARAMETER_EXPECTED | 4064 | 双预计参量。 |
| ERR_ARRAY_AS_PARAMETER_EXPECTED | 4065 | 数组作为预计参量。 |
| ERR_HISTORY_WILL_UPDATED | 4066 | 刷新状态请求历史数据。 |
| ERR_TRADE_ERROR | 4067 | 交易函数错误。 |
| ERR_END_OF_FILE | 4099 | 文件结束。 |
| ERR_SOME_FILE_ERROR | 4100 | 一些文件错误。 |
| ERR_WRONG_FILE_NAME | 4101 | 错误文件名称。 |
| ERR_TOO_MANY_OPENED_FILES | 4102 | 打开文件过多。 |
| ERR_CANNOT_OPEN_FILE | 4103 | 不能打开文件。 |
| ERR_INCOMPATIBLE_FILEACCESS | 4104 | 不协调文件。 |
| ERR_NO_ORDER_SELECTED | 4105 | 没有选择定单。 |
| ERR_UNKNOWN_SYMBOL | 4106 | 不明货币对。 |
| ERR_INVALID_PRICE_PARAM | 4107 | 无效价格。 |
| ERR_INVALID_TICKET | 4108 | 无效定单编码。 |
| ERR_TRADE_NOT_ALLOWED | 4109 | 不允许交易。 |
| ERR_LONGS_NOT_ALLOWED | 4110 | 不允许长期。 |
| ERR_SHORTS_NOT_ALLOWED | 4111 | 不允许短期。 |
| ERR_OBJECT_ALREADY_EXISTS | 4200 | 定单已经存在。 |
| ERR_UNKNOWN_OBJECT_PROPERTY | 4201 | 不明定单属性。 |
| ERR_OBJECT_DOES_NOT_EXIST | 4202 | 定单不存在。 |
| ERR_UNKNOWN_OBJECT_TYPE | 4203 | 不明定单类型。 |
| ERR_NO_OBJECT_NAME | 4204 | 没有定单名称。 |
| ERR_OBJECT_COORDINATES_ERROR | 4205 | 定单坐标错误。 |
| ERR_NO_SPECIFIED_SUBWINDOW | 4206 | 没有指定子窗口。 |
| ERR_SOME_OBJECT_ERROR | 4207 | 定单一些函数错误。 |

# Predefined variables 预定义变量

对于每个执行的 MQL4 程序,一定数量的变量设定可以轻松应对图表中的价格状态：智能交易，脚本或者是客户指标。.
资料应用到可变量的模型中来。
可以由预定义的变量分开处理。 这些数据会在开启后应用 RefreshRates()函数更新。

## Ask 最新卖价

double Ask
对于当前货币对的最新卖价格。使用 RefreshRates()函数更新。
参见 MarketInfo().
示例:

```
   if(iRSI(NULL,0,14,PRICE_CLOSE,0)<25)
      {
       OrderSend(Symbol(),OP_BUY,Lots,Ask,3,Bid-StopLoss*Point,Ask+TakeProfit*Point,
                       "My order #2",3,D'2005.10.10 12:30',Red);
       return;
      }
```

## Bars 柱数

int Bars
返回图表中的柱数。
参见 iBars().
示例:

```
   int counter=1;
   for(int i=1; i<=Bars; i++)
      {
       Print(关闭[i-1]);
      }
```

## Bid 最新买价

double Bid
对于当前货币对的最新买价格。使用 RefreshRates()函数更新。
参见 MarketInfo().
示例:

```
   if(iRSI(NULL,0,14,PRICE_CLOSE,0)>75)
      {
       OrderSend("EURUSD",OP_SELL,Lots,Bid,3,Ask+StopLoss*Point,Bid-TakeProfit*Point,
```

```
                    "My Order #2",3,D'2005.10.10 12:30',Red);
   return(0);
  }
```

## Close[]收盘价

double Close[]
系列数组包含当前图表每个柱的收盘价格。
系列数组元素被索引入倒序的定单，即从最后一个到第一个。. 当前最后一个柱在数组中的索引为 0。图表中的第一个柱的索引为 Bars-1.
参见 iClose().
示例:

```
int handle = FileOpen("file.csv", FILE_CSV|FILE_WRITE, ";");
if(handle>0)
  {
   // 表格栏标题记录
   FileWrite(handle, "Time;Open;High;Low;Close;Volume");
   // 数据记录
   for(int i=0; i<Bars; i++)
      FileWrite(handle, Time[i], Open[i], High[i], Low[i], Close[i], Volume[i]);
   FileClose(handle);
  }
```

## Digits 汇率小数位

int Digits
返回当前货币对的汇率小数位
参见 MarketInfo().
示例:

```
Print(DoubleToStr(Close[0], 小数位));
```

## High[]最高价

double High[]
系列数组包含当前图表每个柱的最高价格。
系列数组元素被索引入倒序的定单，即从最后一个到第一个。. 当前最后一个柱在数组中的索引为 0。图表中的第一个柱的索引为 Bars-1.
参见 iHigh().
示例:

```
//---- 最大值
   i=Bars-KPeriod;
   if(counted_bars>KPeriod) i=Bars-counted_bars-1;
```

```
    while(i>=0)
      {
       double max=-1000000;
       k = i + KPeriod-1;
       while(k>=i)
          {
           price=High[k];
           if(max<price) max=price;
           k--;
          }
       HighesBuffer[i]=max;
       i--;
      }
//----
```

# Low[]最低价

double Low[]
系列数组包含当前图表每个柱的最低价格。
系列数组元素被索引入倒序的定单，即从最后一个到第一个。. 当前最后一个柱在数组中的索引为 0。图表中的第一个柱的索引为 Bars-1.
参见 iLow().
示例:

```
//---- 最小值
    i=Bars-KPeriod;
    if(counted_bars>KPeriod) i=Bars-counted_bars-1;
    while(i>=0)
      {
       double min=1000000;
       k = i + KPeriod-1;
       while(k>=i)
          {
           price=Low[k];
           if(min>price) min=price;
           k--;
          }
       LowesBuffer[i]=min;
       i--;
      }
//----
```

# Open[]开盘价

double Open[]
系列数组包含当前图表每个柱的开盘价格。
系列数组元素被索引入倒序的定单，即从最后一个到第一个。. 当前最后一个柱在数组中的索引为 0。图表中的第一个柱的索引为 Bars-1.
参见 iOpen().
示例:

```
i = Bars - counted_bars - 1;
while(i>=0)
  {
   double high   = High[i];
   double low    = Low[i];
   double open   = Open[i];
   double close = Close[i];
   AccumulationBuffer[i] = (close-low) - (high-close);
   if(AccumulationBuffer[i] != 0)
     {
      double diff = high - low;
      if(0==diff)
          AccumulationBuffer[i] = 0;
      else
        {
         AccumulationBuffer[i] /= diff;
         AccumulationBuffer[i] *= Volume[i];
        }
     }
   if(i<Bars-1) AccumulationBuffer[i] += AccumulationBuffer[i+1];
   i--;
  }
```

# Point 点值

double Point
返回当前图表的点值
参见 MarketInfo().
示例:

```
OrderSend(Symbol(),OP_BUY,Lots,Ask,3,0,Ask+TakeProfit*Point);
```

# Time[]开盘时间

datetime Time[]

系列数组包含当前图表的每个柱开盘时间。数据像日期时间一样呈现时间，从 1979 年 1 月 1 日零点开始以秒计算。

系列数组元素被索引入倒序的定单，即从最后一个到第一个。当前最后一个柱在数组中的索引为 0。图表中的第一个柱的索引为 Bars-1.

参见 iTime().
示例:

```
for(i=Bars-2; i>=0; i--)
  {
   if(High[i+1] > LastHigh) LastHigh = High[i+1];
   if(Low[i+1] < LastLow)      LastLow    = Low[i+1];
   //----
   if(TimeDay(Time[i]) != TimeDay(Time[i+1]))
     {
      P = (LastHigh + LastLow + Close[i+1])/3;
      R1 = P*2 - LastLow;
      S1 = P*2 - LastHigh;
      R2 = P + LastHigh - LastLow;
      S2 = P - (LastHigh - LastLow);
      R3 = P*2 + LastHigh - LastLow*2;
      S3 = P*2 - (LastHigh*2 - LastLow);
      LastLow    = Open[i];
      LastHigh = Open[i];
     }
   //----
   PBuffer[i]    = P;
   S1Buffer[i] = S1;
   R1Buffer[i] = R1;
   S2Buffer[i] = S2;
   R2Buffer[i] = R2;
   S3Buffer[i] = S3;
   R3Buffer[i] = R3;
  }
```

# Volume[]成交量

double Volume[]
系列数组包含当前图表每个柱替克成交量。
系列数组元素被索引入倒序的定单，即从最后一个到第一个。. 当前最后一个柱在数组中的索引为 0。图表中的第一个柱的索引为 Bars-1.
参见 iVolume().
示例:

```
   if(i==0 && time0<i_time+periodseconds)
```

```
   {
     d_volume += Volume[0];
     if(Low[0]<d_low)      d_low = Low[0];
     if(High[0]>d_high) d_high = High[0];
     d_close = Close[0];
   }
last_fpos = FileTell(ExtHandle);
last_volume = Volume[i];
FileWriteInteger(ExtHandle, i_time, LONG_VALUE);
FileWriteDouble(ExtHandle, d_open, DOUBLE_VALUE);
FileWriteDouble(ExtHandle, d_low, DOUBLE_VALUE);
FileWriteDouble(ExtHandle, d_high, DOUBLE_VALUE);
FileWriteDouble(ExtHandle, d_close, DOUBLE_VALUE);
FileWriteDouble(ExtHandle, d_volume, DOUBLE_VALUE);
```

# Program Run 程序运行

使 MQL4 程序运行，必须进行编写("编写" 按钮或 F5) 。编写过程中不允许出现任何错误。必须在 terminal_dir\experts、 terminal_dir\experts\indicators, 或 terminal_dir\experts\scripts, 有相同名称和引伸 EX4 的文件中创建。这样可以开启这个文件。
智能交易,客户指标和脚本的相应图表(Drag'n'Drop 技术)需要从客户端内的导航窗口打开. MQL4 程序只有在客户端开启的基础上运行.
使智能交易停止运行,必须从图表使用的菜单中删除它。"智能交易开启"的状态栏将会直接影响智能交易的运行.
使客户指标停止运行,必须将它从图表中删除.
客户指标和智能交易在图表中的管理直到被删除.关于附属在客户指标和智能交易的信息将储存于客户端内. 脚本在完成执行一次任务后或者当前图表发生改变/关闭时,再或者客户端中断,脚本被自动删除.相应的附属信息不被保存.
在同一个图表内,智能交易,脚本和更多数量的指标可以同时运行.

## Program Run 程序运行

程序在附加图表之后，它开始应用 init()作用运行。在客户端开始和历史数据加载，图表周期改变，由 MetaEditor 编译程序， 智能和客户指标的输入数据设定后，附加到图表上的智能交易和客户指标将会运行。 在账户发生改变后交易将会初始化。
每一个附加到图表上的交易必须在 deinit()的作用协助下工作。 deinit()作用运行在客户端内停止，在图表中关闭 在商品或图表周期改变之前，成功重新编译程序，改变输入数据或改变账户。 在 deinit()函数执行中,您可以使用 UninitializeReason()函数查看原因.deinit()函数必须在 2.5 秒内执行. 如果函数没有在指定的时间段内执行,它将强制完成.脚本除外,它的运行不取决于任何外界的命令. 如果脚本工作时间过长, 可以应用外部命令结束运行 (从图表的菜单中删除脚本,在原有的图表上添加新的脚本,改变图表的商品或图表周期).在这种情况下,deinit()函数限制在 2.5 秒内.
新报价格的输入,start()函数将会添加到智能交易和客户指标上执行.当新报价格进入时如果 start()函数没有开启运行,新的报价格会被忽略. function launched at the preceding quote was running when a new quote came, 随后,只有当新报价格进入后 start()函数才会运行. 对于客户指标,由于新报价格的输入当前图表商品或期限发生改变, start()函数将会开启重新计算.在交易属性窗口打开时 start()函数 停止运行.迟些在交易执行中不会打开.
从图表中拆卸程序, 改变商品或图表周期,改变账户,关闭图表,客户端的改变将会中断程序的执行.如果 start()函数在给出停止命令的时刻执行 , 时间限制在 2.5 秒. 程序能够尝试关闭 IsStopped()函数并结束.
脚本的执行不取决于报价格的输入.在商品或图表周期发生改变时,脚本将停止运行并且中断从客户端上下载.
脚本和交易的运行在自己的界面.客户指标则是在主界面上运行. 如果客户指标中出现 iCustom() 函数, 这个指标的运行是在程序中显示的. 资料库函数在程序界面.

# Imported functions call 输入函数调用

输入函数在 mql4 程序中执行，即所谓的约束应用。这就意味着，直到输入函数被应用，相应的模板(ex4 or dll)将不会加载。 MQL4 和 DLL 资料在模板的命令下被调用。
这里不推荐使用模板被加载的 全名 Drive:\Directory\FileName.Ext 。 MQL4 资料是 从 terminal_dir\experts\libraries 文件夹中下载。如果资料中未找到，将会接受来自 terminal_dir\experts 文件夹的下载。
资料系统(DLLs)按照以下规则运行。如果资料已经被下载 (从其他智能交易。例如，从其他相同时间开启的客户端下载)，资料下在以满。会按照以下规则命令执行:
1. terminal_dir\experts\libraries 目录 。
2. 从 terminal_dir 客户端被开启的目录。
3. 当前目录。
4.windows_dir\SYSTEM32 的目录系统 (或是对于 windows_dir\SYSTEM for Win98)。
5. 在 windows_dir 执行系统中安装的目录。
6. 录中列出的 PATH 环境系统变量。
如果 DLL 应用另一个 DLL 运行，在后者省缺的情况下前者不会加载。
与资料系统不同，客户资料(MQL4)加载是对于每个分开模板，独立地从模板上加载。举例来说，ex4 模板可以从 lib1.ex4 和 lib2.ex4 资料中调用。反过来 lib1.ex4 资料可以从 lib2.ex4 资料中调用函数。在这种情况下，需要复制一份 lib1.ex4 资料和两分 lib2.ex4 资料，所有来自相同 caller.ex4 模板的资料。
从 DLL 转入到 mql4 程序的输入函数必须提供接受 Windows API 函数。 提供这样公约，关键词 __stdcall 对于微软（r）公司的编译器，在编写的 C 或 C++ 语言中源代码中使用。上述公约有以下特点:
- 呼叫函数(这种情况下，是 mql4 程序) 必须"参见"(来自 DLL 输入函数)适应参量；
- 呼叫函数(这种情况下，是 mql4 程序) 在反向命令中放置参量等等，从左到右；它是在读输入函数参量通过的命令；
- 参量根据他们的价格值通过，除了一些直接的界限(存在一些界限)；
- 参量通过，输入函数本身将会变大。
在描述输入函数 模型时，基本不使用参量默认值。因为所有的默认值必须直接通过输入函数。
如果调用输入函数失败(智能交易设置不允许 DLL 输入，或相关的资料由于一些原因不能下载)，智能交易会停止运行并在“停止运行”处提出相关信息。另外，智能交易只有在重新初始化之后才会开启。智能交易的初始化由于需要重新编译或开启属性可以按确定键。

# Runtime errors 运行错误

如果是由于一个关键错误停止了程序的运行，这个错误代码可以在下次开启时使用 GetLastError () 函数。 在未开启之前，错误变量不会归零。 在客户终端执行子系统时，其发生在 mql4 程序执行错误代码可以储存 。对于每一个 mql4 程序执行，存在一个特殊的 last_error。在 init 函数运行之前，last_error 变量必须归零。如果在计算过程中或内建函数时发生错误，last_error 变量会给出相应的错误代码。存储在这个变量中的值可以应用 GetLastError 函数。 另外，last_error 变量将归零。
存在一些智能或客户指标执行直接导致的关键错误， 在智能或客户指标未被重新初始化后

不会开启使用。：

| 常数 | 值 | 描述 |
| --- | --- | --- |
| ERR_WRONG_FUNCTION_POINTER | 4001 | 在调用内部函数时，发现错误指示物 |
| ERR_NO_MEMORY_FOR_CALL_STACK | 4003 | 在调用内部函数时，可能出现储存器 |
| ERR_RECURSIVE_STACK_OVERFLOW | 4004 | 在调用循环函数时，数据存储器溢出 |
| ERR_NO_MEMORY_FOR_PARAM_STRING | 4006 | 在调用内部函数时，可能作为功参量被分配 |
| ERR_NO_MEMORY_FOR_TEMP_STRING | 4007 | 不能分配临时缓冲字符 |
| ERR_NO_MEMORY_FOR_ARRAYSTRING | 4010 | 在转让时，数组不能重新记忆 |
| ERR_TOO_LONG_STRING | 4011 | 在转让时，字串符过长可能导致被送到服务器缓冲（不能再次发送到服务器缓冲处理） |
| ERR_REMAINDER_FROM_ZERO_DIVIDE | 4012 | 用剩余的部分除以 0 |
| ERR_ZERO_DIVIDE | 4013 | 除以 0 |
| ERR_UNKNOWN_COMMAND | 4014 | 无效指令 |

如果在错误生成时程序停止工作，这些错误代码可能被下一个开启的程序读取。应用 GetLastError()函数。 在程序开始之前 last_error 变量不会归零。
存在一些相关 输入函数调用 的错误会立即停止智能交易或客户指标的起初执行直至被初始化。

| 常数 | 值 | 描述 |
| --- | --- | --- |
| ERR_CANNOT_LOAD_LIBRARY | 4018 | 输入数据时，生成 dll 或 ex4 l 错误 |
| ERR_CANNOT_CALL_FUNCTION | 4019 | 输入数据时, dll 或 ex4 不包含调用功能 |
| ERR_DLL_CALLS_NOT_ALLOWED | 4017 | 输入数据时，dll 数据禁止 |
| ERR_EXTERNAL_CALLS_NOT_ALLOWED | 4020 | 输入数据时，ex4 数据禁止 |

其他错误，不中断程序执行。

| 常数 | 值 | 描述 |
| --- | --- | --- |
| ERR_ARRAY_INDEX_OUT_OF_RANGE | 4002 | 企图获取数组项目，其中一些是出于数组范围 |
| ERR_NOT_INITIALIZED_STRING | 4008 | 没有初始化字符;没有值被分配到字符作为运算中的表达 |
| ERR_NOT_INITIALIZED_ARRAYSTRING | 4009 | 没有初始化字符;没有值被分配到字符作为运算中的表达 |
| ERR_NO_MEMORY_FOR_RETURNED_STR | 4021 | 不可能重新记忆字符 |

The ERR_NO_MQLERROR (4000) 代码不会生成。
有一些错误可能只是由于软件或硬件故障。如果一些错误文本反复出现，应与开发商联络。

| 常数 | 值 | 描述 |
| --- | --- | --- |
| ERR_WRONG_FUNCTION_POINTER | 4001 | 在调用一个内部函数时，发现错误指示物 |

| | | |
|---|---|---|
| ERR_UNKNOWN_COMMAND | 4014 | 无效指令 |
| ERR_NOT_INITIALIZED_ARRAY | 4016 | 数组没有初始化 |
| ERR_INVALID_FUNCTION_PARAMSCNT | 4050 | 无效参量 |
| ERR_STRING_FUNCTION_INTERNAL | 4052 | 字串符函数内部错误 |
| ERR_TRADE_ERROR | 4067 | 交易函数内部错误 |
| ERR_SOME_OBJECT_ERROR | 4207 | 窗体函数内部错误 |

| 函数 | 错误代码 |
|---|---|
| AccountFreeMarginCheck | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_UNKNOWN_SYMBOL (4106), ERR_NOT_ENOUGH_MONEY (134) |
| OrderSend | ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INVALID_PRICE_PARAM (4107), ERR_UNKNOWN_SYMBOL (4106), ERR_TRADE_NOT_ALLOWED (4109), ERR_LONGS_NOT_ALLOWED (4110), ERR_SHORTS_NOT_ALLOWED (4111), 用交易服务器返回源代码 |
| OrderClose | ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INVALID_PRICE_PARAM (4107), ERR_INVALID_TICKET (4108), ERR_UNKNOWN_SYMBOL (4106), ERR_TRADE_NOT_ALLOWED (4109), 用交易服务器返回源代码 |
| OrderCloseBy | ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INVALID_TICKET (4108), ERR_UNKNOWN_SYMBOL (4106), ERR_TRADE_NOT_ALLOWED (4109), 用交易服务器返回源代码 |
| OrderDelete | ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INVALID_TICKET (4108), ERR_UNKNOWN_SYMBOL (4106), ERR_TRADE_NOT_ALLOWED (4109), 用交易服务器返回源代码 |
| OrderModify | ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051), |

| | |
|---|---|
| | ERR_INVALID_PRICE_PARAM (4107), ERR_INVALID_TICKET (4108), ERR_UNKNOWN_SYMBOL (4106), ERR_TRADE_NOT_ALLOWED (4109), 用交易服务器返回源代码 |
| GetLastError | ERR_NO_ERROR (0) |

last_error 变量的一些功能的价格值改变只能产生一种错误。

| 函数 | 错误代码 |
|---|---|
| ArrayBsearch | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| ArrayCopy | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INCOMPATIBLE_ARRAYS (4056), ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| ArrayCopyRates | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INCOMPATIBLE_ARRAYS (4056), ERR_STRING_PARAMETER_EXPECTED (4062), |
| ArrayCopySeries | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INCORRECT_SERIESARRAY_USING (4054), ERR_INCOMPATIBLE_ARRAYS (4056), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_HISTORY_WILL_UPDATED (4066), ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| ArrayDimension | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053) |
| ArrayGetAsSeries | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053) |
| ArrayInitialize | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| ArrayIsSeries | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053) |
| ArrayMaximum | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| ArrayMinimum | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| ArrayRange | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), |

| | |
|---|---|
| | ERR_SOME_ARRAY_ERROR (4053), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| ArrayResize | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| ArraySetAsSeries | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053) |
| ArraySize | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053) |
| ArraySort | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INCORRECT_SERIESARRAY_USING (4054), ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| FileClose | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| FileDelete | ERR_WRONG_FILE_NAME (4101), ERR_SOME_FILE_ERROR (4100) |
| FileFlush | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| FileIsEnding | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| FileIsLineEnding | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| FileOpen | ERR_TOO_MANY_OPENED_FILES (4102), ERR_WRONG_FILE_NAME (4101), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_SOME_FILE_ERROR (4100), ERR_CANNOT_OPEN_FILE (4103) |
| FileOpenHistory | ERR_TOO_MANY_OPENED_FILES (4102), ERR_WRONG_FILE_NAME (4101), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_SOME_FILE_ERROR (4100), ERR_CANNOT_OPEN_FILE (4103) |
| FileReadArray | ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_ARRAY_ERROR (4053), ERR_SOME_FILE_ERROR (4100), ERR_END_OF_FILE (4099) |
| FileReadDouble | ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_END_OF_FILE (4099) |
| FileReadInteger | ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_END_OF_FILE (4099) |

| | |
|---|---|
| FileReadNumber | ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100), ERR_END_OF_FILE (4099) |
| FileReadString | ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100), ERR_TOO_LONG_STRING (4011), ERR_END_OF_FILE (4099) |
| FileSeek | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| FileSize | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| FileTell | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| FileWrite | ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_SOME_FILE_ERROR (4100) |
| FileWriteDouble | ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100) |
| FileWriteInteger | ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100) |
| FileWriteString | ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100), ERR_STRING_PARAMETER_EXPECTED (4062) |
| FileWriteArray | ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100), |
| GlobalVariableCheck | ERR_STRING_PARAMETER_EXPECTED (4062) |
| GlobalVariableDel | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_GLOBAL_VARIABLES_PROCESSING (4057) |
| GlobalVariableGet | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_GLOBAL_VARIABLE_NOT_FOUND (4058) |
| GlobalVariablesDeleteAll | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_GLOBAL_VARIABLES_PROCESSING (4057) |
| GlobalVariableSet | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_GLOBAL_VARIABLES_PROCESSING (4057), ERR_GLOBAL_VARIABLE_NOT_FOUND (4058) |
| GlobalVariableSetOnCondition | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_GLOBAL_VARIABLE_NOT_FOUND (4058) |

| | |
|---|---|
| iCustom | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| technical indicators, series access functions | ERR_HISTORY_WILL_UPDATED (4066) |
| technical indicators OnArray | ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053) |
| IndicatorBuffers | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| IndicatorDigits | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| IndicatorShortName | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| MarketInfo | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_UNKNOWN_SYMBOL (4106), ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| MathArccos | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| MathArcsin | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| MathMod | ERR_ZERO_DIVIDE (4013) |
| MathSqrt | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| MessageBox | ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_STRING_PARAMETER_EXPECTED (4062) |
| ObjectCreate | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_UNKNOWN_OBJECT_TYPE (4203), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_OBJECT_ALREADY_EXISTS (4200), ERR_NO_SPECIFIED_SUBWINDOW (4206) |
| ObjectDelete | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202) |
| ObjectDescription | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202) |
| ObjectFind | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204) |
| ObjectGet | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_UNKNOWN_OBJECT_PROPERTY (4201) |

| | |
|---|---|
| ObjectGetFiboDescription | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_UNKNOWN_OBJECT_TYPE (4203), ERR_UNKNOWN_OBJECT_PROPERTY (4201) |
| ObjectGetShiftByValue | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_OBJECT_COORDINATES_ERROR (4205) |
| ObjectGetValueByShift | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_OBJECT_COORDINATES_ERROR (4205) |
| ObjectMove | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_OBJECT_DOES_NOT_EXIST (4202) |
| ObjectName | ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_ARRAY_INDEX_OUT_OF_RANGE (4002) |
| ObjectSet | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_UNKNOWN_OBJECT_PROPERTY (4201) |
| ObjectSetText | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202) |
| ObjectSetFiboDescription | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_UNKNOWN_OBJECT_TYPE (4203), ERR_UNKNOWN_OBJECT_PROPERTY (4201) |
| ObjectType | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202) |
| OrderClosePrice | ERR_NO_ORDER_SELECTED (4105) |
| OrderCloseTime | ERR_NO_ORDER_SELECTED (4105) |
| OrderComment | ERR_NO_ORDER_SELECTED (4105) |
| OrderCommission | ERR_NO_ORDER_SELECTED (4105) |

| | |
|---|---|
| OrderExpiration | ERR_NO_ORDER_SELECTED (4105) |
| OrderLots | ERR_NO_ORDER_SELECTED (4105) |
| OrderMagicNumber | ERR_NO_ORDER_SELECTED (4105) |
| OrderOpenPrice | ERR_NO_ORDER_SELECTED (4105) |
| OrderOpenTime | ERR_NO_ORDER_SELECTED (4105) |
| OrderPrint | ERR_NO_ORDER_SELECTED (4105) |
| OrderProfit | ERR_NO_ORDER_SELECTED (4105) |
| OrderStopLoss | ERR_NO_ORDER_SELECTED (4105) |
| OrderSwap | ERR_NO_ORDER_SELECTED (4105) |
| OrderSymbol | ERR_NO_ORDER_SELECTED (4105) |
| OrderTakeProfit | ERR_NO_ORDER_SELECTED (4105) |
| OrderTicket | ERR_NO_ORDER_SELECTED (4105) |
| OrderType | ERR_NO_ORDER_SELECTED (4105) |
| PlaySound | ERR_WRONG_FILE_NAME (4101) |
| SendFTP | ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_STRING_PARAMETER_EXPECTED (4062) |
| SendMail | ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_FUNCTION_NOT_CONFIRMED (4060), ERR_SEND_MAIL_ERROR (4061) |
| SetIndexArrow | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| SetIndexBuffer | ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCORRECT_SERIESARRAY_USING (4054), ERR_INCOMPATIBLE_ARRAYS (4056) |
| SetIndexDrawBegin | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| SetIndexEmptyValue | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| SetIndexLabel | ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_STRING_PARAMETER_EXPECTED (4062) |
| SetIndexShift | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| SetIndexStyle | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| SetLevelValue | ERR_INVALID_FUNCTION_PARAMVALUE (4051) |
| Sleep | ERR_CUSTOM_INDICATOR_ERROR (4055) |

| | |
|---|---|
| StringFind | ERR_STRING_PARAMETER_EXPECTED (4062) |
| StringGetChar | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NOT_INITIALIZED_STRING (4008), ERR_ARRAY_INDEX_OUT_OF_RANGE (4002) |
| StringLen | ERR_STRING_PARAMETER_EXPECTED (4062) |
| StringSetChar | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_NOT_INITIALIZED_STRING (4008), ERR_TOO_LONG_STRING (4011), ERR_ARRAY_INDEX_OUT_OF_RANGE (4002) |
| StringSubstr | ERR_STRING_PARAMETER_EXPECTED (4062), ERR_TOO_LONG_STRING (4011) |
| StringTrimLeft | ERR_STRING_PARAMETER_EXPECTED (4062) |
| StringTrimRight | ERR_STRING_PARAMETER_EXPECTED (4062) |
| WindowIsVisible | ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059) |
| WindowFind | ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NOT_INITIALIZED_STRING (4008) |
| WindowHandle | ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NOT_INITIALIZED_STRING (4008) |
| WindowScreenShot | ERR_WRONG_FILE_NAME (4101), ERR_INVALID_FUNCTION_PARAMVALUE (4051) |

其他函数在 last_error 可变物价格值下不会改变。

AccountBalance, AccountCompany, AccountCredit, AccountCurrency, AccountEquity, AccountFreeMargin, AccountLeverage, AccountMargin, AccountName, AccountNumber, AccountProfit, AccountServer, Alert, CharToStr, Comment, Day, DayOfWeek, DayOfYear, DoubleToStr, GetTickCount, HideTestIndicators, Hour, IndicatorCounted, IsConnected, IsDemo, IsDllsAllowed, IsExpertEnabled, IsLibrariesAllowed, IsOptimization, IsStopped, IsTesting, IsTradeAllowed, IsTradeContextBusy, IsVisualMode, MathAbs, MathArctan, MathCeil, MathCos, MathExp, MathFloor, MathLog, MathMax, MathMin, MathPow, MathRand, MathRound, MathSin, MathSrand, MathTan, Minute, Month, NormalizeDouble, ObjectsDeleteAll, ObjectsTotal, OrderSelect, OrdersHistoryTotal, Period, Print, RefreshRates, Seconds, SetLevelStyle, StringConcatenate, StrToTime, StrToDouble, Symbol, TerminalCompany, TerminalName, TerminalPath, TimeCurrent, TimeDay, TimeDayOfWeek, TimeDayOfYear, TimeHour, TimeLocal, TimeMinute, TimeMonth, TimeSeconds, TimeToStr, TimeYear, UninitializeReason, WindowBarsPerChart, WindowFirstVisibleBar, WindowPriceOnDropped, WindowRedraw, WindowTimeOnDropped, WindowsTotal, WindowOnDropped, WindowXOnDropped, WindowYOnDropped, Year

# Account information 账户信息

激活账户信息的一组函数。

## AccountBalance( )账户余额

double AccountBalance( )
返回账户余额(账户中相当数量的价格值金钱) .
示例:
Print("账户余额= ",AccountBalance());

## AccountCredit( )账户信用点数

double AccountCredit( )
返回账户信用点数.
示例:
Print("账户点数 ", AccountCredit());

## AccountCompany( )账户公司名

string AccountCompany( )
返回账户公司名。
示例:
Print("账户公司名", AccountCompany());

## AccountCurrency( )货币对

string AccountCurrency( )
返回账户所用的通货名称。
示例:
Print("账户货币对", AccountCurrency());

## AccountEquity( )账户资产净值

double AccountEquity( )
对于当前账户返回资产净值。资产净值取决于交易服务器的设置。
示例:
Print("账户净值 = ",AccountEquity());

# AccountFreeMargin( )账户免费保证金

double AccountFreeMargin( )
返回当前帐户的免费保证金价格值。
示例:
Print("账户免费保证金 = ",AccountFreeMargin());

# AccountFreeMarginCheck()账户当前价格自由保证金

double AccountFreeMarginCheck( string symbol, int cmd, double volume)
当前账户的当前价格上在指定开仓的仓位返回自由保证金。如果免费保证金不够，将会生成错误 134(ERR_NOT_ENOUGH_MONEY) 。
参量:
symbol - 交易业务货币对。
cmd - 交易类型。可能是 OP_BUY 或者 OP_SELL。
volume - 份额数。
示例:
if(AccountFreeMarginCheck(Symbol(),OP_BUY,Lots)<=0 || GetLastError()==134) return;

# AccountFreeMarginMode( )账户免费保证金模式

double AccountFreeMarginMode( )
在当前开仓位置的账户上计算免费保证金的模式。计算方式可能采取以下价格值:
0 - 浮动 profit/loss 不使用
1 - 两个浮动赢利和损失在开仓位置上使用计算自由保证金;
2 - 只有赢利值被使用计算，不考虑当前开仓的亏损;
3 - 只有亏损值被使用计算, 不考虑当前开仓的亏损。
示例:

```
if(AccountFreeMarginMode()==0)
    Print("浮点盈利/亏损不使用。");
```

# AccountLeverage( )账户杠杆

int AccountLeverage( )
返回当前账户杠杆比率。
示例:
Print("账户#",AccountNumber(), " 杠杆比率", AccountLeverage());

# AccountMargin( )账户保证金

double AccountMargin( )

返回当前帐户的保证金。
示例:
Print("账户保证金 ", AccountMargin());

## AccountName( )账户名称

string AccountName( )
返回当前帐户名称。
示例:
Print("账户名称", AccountName());

## AccountNumber( )账户数字

int AccountNumber( )
返回当前帐户的数字。
示例:
Print("账户数字", AccountNumber());

## AccountProfit( )账户利润

double AccountProfit( )
返回账户利润。
示例:
Print("账户利润", AccountProfit());

## AccountServer( )账户连接服务器

string AccountServer( )
返回连接服务器的名称。
示例:
Print("服务器名称", AccountServer());

## AccountStopoutLevel( )账户停止水平值

int AccountStopoutLevel( )
返回停止水平值。
示例:
Print("停止水平 = ", AccountStopoutLevel());

# AccountStopoutMode( )账户停止返回模式

int AccountStopoutMode( )
对于停止水平返回的的运算方式。运算方式值如下:
0 - 计算保证金和净值之间的百分比;
1 - 比较自由保证金水平和绝对值。
示例:

```
int level=AccountStopoutLevel();
if(AccountStopoutMode()==0)
    Print("停止水平= ",水平, "%");
else
    Print("停止水平= ",  水平, " ", AccountCurrency());
```

# Array functions 数组函数

使用数组的一组函数。
数组的最大维数为四维。每个维数被索引编为从 0 至维度-1。事实上，第一维数组的 50 个，在调用时第一个数组显示为[0],最后一个数组显示为[49]。
使用这些函数(除那些改变定量和定性的数组外) 能够预定义时间系列 Time[], Open[], High[], Low[], Close[], Volume[]

## ArrayBsearch()数组搜索

int ArrayBsearch( double array[], double value, void count, void start, void direction)
如果没有发现事件，值会返回到第一个维度的数组或者最近的一个数组。
此函数不能用在字符型或连续数字的数组上(除打开柱的连续数组)。
注解: 双元查找只能够存储数。存储数字数组使用 ArraySort() 函数。
参量:
array[] - 需要搜索的数组.
value - 将要搜索的值
count - 搜索的数量，默认搜索所有的数组.
start - 搜索的开始点，默认从头开始.
direction - 搜索的方向:
MODE_ASCEND 顺序搜索,
MODE_DESCEND 倒序搜索.
示例:

```
datetime daytimes[];
int          shift=10,dayshift;
// 全部 Time[] 数组被排列在后面的形式
ArrayCopySeries(daytimes,MODE_TIME,Symbol(),PERIOD_D1);
if(Time[shift]>=daytimes[0]) dayshift=0;
else
  {
   dayshift=ArrayBsearch(daytimes,Time[shift],WHOLE_ARRAY,0,MODE_DESCEND);
   if(Period()<PERIOD_D1) dayshift++;
  }
Print(TimeToStr(Time[shift])," corresponds to ",dayshift," day bar opened at ",
        TimeToStr(daytimes[dayshift]));
```

## ArrayCopy()数组复制

int ArrayCopy( void dest[], object source[], void start_dest, void start_source, void count)
复制一个数组到另外一个数组。 只有 double[], int[], datetime[], color[], 和 bool[] 这些类型

的数组可以被复制。
返回复制元素总量。
参量:
dest[] - 目标数组 。
source[] - 源数组 。
start_dest - 从目标数组的第几位开始写入，默认为 0 。
start_source - 从源数组的第几位开始读取，默认为 0 。
count - 读取多少位的数组 。默认值为 WHOLE_ARRAY 常数。
示例:

```
double array1[][6];
double array2[10][6];
// 数组 2 被相同数据添满
ArrayCopyRates(array1);
ArrayCopy(array2,array1,0,0,60);
// 现在数组 2 的前 10 个柱来自历史(前 10 个柱包括索引[Bars-1])
ArrayCopy(array2,array1,0,Bars*6-60,60);
// 现在数组 2 的后 10 个柱来自历史(后 10 个柱包括索引[0])
```

# ArrayCopyRates()数组复制走势

int ArrayCopyRates( void dest_array[], void symbol, void timeframe)
复制一段走势图上的数据到一个二维数组，并返回复制柱总量，如果是-1 表示失败。数组的第二维只有 6 个项目分别是:
0 - 时间,
1 - 开盘价格,
2 - 最低价格,
3 - 最高价格,
4 - 收盘价格,
5 - 成交量.
如果数据(货币对名称/不同于当前的时间周期) 拒绝其他图表，这种情况下相应的图表不能够在客户端内打开，数据自然会拒绝服务器。这种情况， 错误 ERR_HISTORY_WILL_UPDATED (4066 - 拒绝刷新历史数据) 将被放置到 last_error 变量中，并且将再次拒绝(查看范例 ArrayCopySeries()).
注解:此数组通常用于到 DLL 函数的通过数据 。
对于数据数组内存没有真正的分配，没有真正地执行复制。当数组访问时，将会改变方向。
参量:
dest_array[] - 在二维数组上的双重目标数组。
symbol - 货币对名称
timeframe - 时间周期. 可以是列出时间周期的任意值。
示例:

```
double array1[][6];
ArrayCopyRates(array1,"EURUSD", PERIOD_H1);
Print("当前柱 ",TimeToStr(array1[0][0]),"开盘价格", array1[0][1]);
```

# ArrayCopySeries()数组复制系列走势

int ArrayCopySeries( void array[], int series_index, void symbol, void timeframe)
复制一个系列的走势图数据到数组上.
对于数据数组内存没有真正的分配，没有真正地执行复制。当数组访问时，将会改变方向。在客户指标内禁止数组作为 数组下标分配。这种情况下，数组被真正复制。
如果数据从不同货币对/时间周期图表复制，数据可能缺乏。这种情况下， 错误 ERR_HISTORY_WILL_UPDATED (4066 - 拒绝刷新历史数据)将被放置到 last_error 变量中，并且在确定的时间周期内将重新尝试复制。
注解: 如果 series_index 是 MODE_TIME, 那么第一个参量必须是日期型的数组
参量:
array[] - 目标第一维数字数组。
series_index - 连续数组识别符。必须是连续数组列表识别符其中的值。
symbol - 当前货币对名称
timeframe - 图表时间周期。可以是 列出时间周期的任意值。
示例:

```
datetime daytimes[];
int          shift=10,dayshift,error;
//---- 此 Time[] 数组被排列在后面的命令
ArrayCopySeries(daytimes,MODE_TIME,Symbol(),PERIOD_D1);
error=GetLastError();
if(error==4066)
  {
   //---- 使两个以上接受只读
   for(int i=0;i<2; i++)
      {
       Sleep(5000);
       ArrayCopySeries(daytimes,MODE_TIME,Symbol(),PERIOD_D1);
       //---- 检查但前每日柱时间
       datetime last_day=daytimes[0];
       if(Year()==TimeYear(last_day)          &&          Month()==TimeMonth(last_day)          &&
Day()==TimeDay(last_day)) break;
      }
  }
if(Time[shift]>=daytimes[0]) dayshift=0;
else
  {
   dayshift=ArrayBsearch(daytimes,Time[shift],WHOLE_ARRAY,0,MODE_DESCEND);
   if(Period()<PERIOD_D1) dayshift++;
  }
Print(TimeToStr(Time[shift])," 相 应 ",dayshift," day bar opened at ",
TimeToStr(daytimes[dayshift]));
```

# ArrayDimension()返回数组维数

int ArrayDimension( object array[])
返回数组的多元维数
参量:
array[]　-　将要返回的数组。
示例:

```
int num_array[10][5];
int dim_size;
dim_size=ArrayDimension(num_array);
// dim_size=2
```

# ArrayGetAsSeries()返回数组序列

bool ArrayGetAsSeries( object array[])
返回 TRUE，如果数组有组织序列的数组(是否从最后到最开始排序过的)，否则返回 FALSE。
参量:
array[]　-　需要检查的数组。
示例:

```
   if(ArrayGetAsSeries(array1)==true)
      Print("数组 1 是作为连续指数被编入索引");
   else
      Print("数组 1  正常编入索引(从左到右)");
```

# ArrayInitialize()数组初始化

int ArrayInitialize( void array[], double value)
对数组进行初始化，返回经过初始化的数组项的个数 。
注解:在客户指标中的 init()函数不建议使用到初始化缓冲，在这种函数自动初始化"空值"将自动分配和缓冲重新分配。
参量:
array[]　-　需要初始化的数组。
value　-　新的数组项的值 。
示例:

```
   //----初始化所有带有 2.1 的数组
   double myarray[10];
   ArrayInitialize(myarray,2.1);
```

# ArrayIsSeries()判断数组连续

bool ArrayIsSeries( object array[])

如果检查数组是连续的(Time[],Open[],Close[],High[],Low[], or Volume[]), 返回 TRUE，否则返回 FALSE。
参量:
array[] - 需要检查的数组 。
示例:

```
if(ArrayIsSeries(array1)==false)
   ArrayInitialize(array1,0);
else
   {
    Print("连续数组不能被初始化!");
    return(-1);
   }
```

# ArrayMaximum()数组最大值定位

int ArrayMaximum( double array[], void count, void start)
找出数组中最大值的定位 。在数组中函数返回最大值位置。
参量:
array[] - 搜索数字数组。
count - 搜索数组中项目的个数。
start - 搜索的开始指数。
示例:

```
double num_array[15]={4,1,6,3,9,4,1,6,3,9,4,1,6,3,9};
int       maxValueIdx=ArrayMaximum(num_array);
Print("最大值 = ", num_array[maxValueIdx]);
```

# ArrayMinimum()数组最小值定位

int ArrayMinimum( double array[], void count, void start)
找出数组中最小值的定位 。在数组中函数返回最小值位置。
参量:
array[] - 搜索数字数组。
count - 搜索数组中项目的个数。
start - 搜索的开始指数。
示例:

```
double num_array[15]={4,1,6,3,9,4,1,6,3,9,4,1,6,3,9};
int       minValueidx=ArrayMinimum(num_array);
Print("最小值 = ", num_array[minValueIdx]);
```

# ArrayRange()返回数组指定维数数量

int ArrayRange( object array[], int range_index)

取数组中指定维数中项目的数量。 索引以零为基础，维度的大小要大于最大索引 1 个点。
参量:
array[] - 需要检查的数组。
range_index - 指定的维数 。
示例:

```
int      dim_size;
double num_array[10,10,10];
dim_size=ArrayRange(num_array, 1);
```

# ArrayResize()改变数组维数

int ArrayResize( void array[], int new_size)
设定第一维度的大小。如果成功执行，在重新设定后返回包含的全部个数。如果数组没有重设，返回 -1。
注解:在函数完成执行后，在函数内数组地方水平化并且重设将保留不变。 在函数被重新调用后，一些数组将不同于表明的数组。
参量:
array[] - 需要重设的数组。
new_size - 第一维中数组的新大小 。
示例:

```
double array1[][4];
int      element_count=ArrayResize(array1, 20);
// 新大小 - 80 个
```

# ArraySetAsSeries()设定系列数组

bool ArraySetAsSeries( void array[], bool set)
设定指数数组为系列数组。如果设置参量值为 TRUE，数组将被编入索引。数组 0 位的值是最后的值。 其 FALSE 值设定一个标准的索引命令。此函数返回先前状态。
参量:
array[] - 需要设置的数组。
set - 索引数组命令。
示例:

```
double macd_buffer[300];
double signal_buffer[300];
int      i,limit=ArraySize(macd_buffer);
ArraySetAsSeries(macd_buffer,true);
for(i=0; i<limit; i++)
    macd_buffer[i]=iMA(NULL,0,12,0,MODE_EMA,PRICE_CLOSE,i)-iMA
(NULL,0,26,0,MODE_EMA,PRICE_CLOSE,i);
for(i=0; i<limit; i++)
    signal_buffer[i]=iMAOnArray(macd_buffer,limit,9,0,MODE_SMA,i);
```

# ArraySize()返回数组项目数

int ArraySize( object array[])
返回数组的项目数 。对于第一维数组，用 ArraySize 函数返回的 ArrayRange(array,0)。
参量:
array[] - 任何类型数组。
示例:

```
int count=ArraySize(array1);
for(int i=0; i<count; i++)
  {
   // 一些计算.
  }
```

# ArraySort()数组排序

int ArraySort( void array[], void count, void start, void sort_dir)
对数组进行排序，系列数组不能 ArraySort()使用进行排序 。
参量:
array[] - 被排列的数组。
count - 对多少个数组项进行排序。
start - 排序的开始点。
sort_dir - 排序方式，
MODE_ASCEND 顺序排列，
MODE_DESCEND 倒序排列。

示例:

```
  double num_array[5]={4,1,6,3,9};
  // 新数组包含值 4,1,6,3,9
  ArraySort(num_array);
  // 被排列新数组 1,3,4,6,9
  ArraySort(num_array,WHOLE_ARRAY,0,MODE_DESCEND);
  // 被排列新数组 9,6,4,3,1
```

# Checkup 检查

一组可以检测当前客户端状态（包括 MQL4 程序的环境状态）的函数。

## GetLastError( )返回最后错误

int GetLastError( )
函数返回最后生成错误，随后特殊值 last_error 变量的代码存储归零。 所以， 对于 GetLastError() 调用文本将返回 0。
示例:

```
int err;
int handle=FileOpen("somefile.dat", FILE_READ|FILE_BIN);
if(handle<1)
  {
   err=GetLastError();
   Print("错误(",err,"): ",ErrorDescription(err));
   return(0);
  }
```

## IsConnected( )返回联机状态

bool IsConnected( )
在客户终端和服务器执行数据之间函数返回主要连接状态。如果连接服务器成功，返回 TRUE。否则，返回 FALSE。
示例:

```
if(!IsConnected())
  {
   Print("没有连接!");
   return(0);
  }
// 需要打开连接
// ...
```

## IsDemo( )返回模拟账户

bool IsDemo( )
如果智能交易在模拟账户运行，返回 TRUE 。否则，返回 FALSE。
示例:

```
if(IsDemo()) Print("在模拟账户运行");
else Print("在真实账户运行");
```

# IsDllsAllowed( )返回 dll 允许调用

bool IsDllsAllowed( )
如果智能交易函数 DLL 允许调用，返回 TRUE。否则，返回 FALSE。
参见 IsLibrariesAllowed(), IsTradeAllowed().
示例:

```
#import "user32.dll"
    int       MessageBoxA(int hWnd, string szText, string szCaption,int nType);
...
...
if(IsDllsAllowed()==false)
  {
   Print("DLL 不允许调用。智能交易没有运行。");
   return(0);
  }
// 智能交易外部调用 DLL 函数
  MessageBoxA(0,"an message","Message",MB_OK);
```

# IsExpertEnabled( )返回智能交易开启状态

bool IsExpertEnabled( )
如果智能交易开启运行，返回 TRUE。否则，返回 FALSE。
示例:

```
while(!IsStopped())
  {
   ...
   if(!IsExpertEnabled()) break;
  }
```

# IsLibrariesAllowed( )返回数据库函数调用

bool IsLibrariesAllowed( )
如果智能交易允许调用数据库函数，返回 TRUE 。否则，返回 FALSE。 参见 IsDllsAllowed(), IsTradeAllowed().
示例:

```
#import "somelibrary.ex4"
    int somefunc();
...
...
if(IsLibrariesAllowed()==false)
  {
   Print("不允许调用数据库");
```

```
      return(0);
    }
  // 智能交易调用外部 DLL 函数
  somefunc();
```

# IsOptimization( )返回策略测试中优化模式

bool IsOptimization( )
如果在策略测试中智能交易为优化模式，返回 TRUE。否则，返回 FALSE。
示例:

```
  if(IsOptimization()) return(0);
```

# IsStopped( )返回终止业务

bool IsStopped( )
如果程序（智能交易或脚本）得到命令中止业务，返回 TRUE。否则，返回 FALSE。 在客户端中止执行之前程序业务会继续运行 2.5 秒。
示例:

```
  while(expr!=false)
    {
     if(IsStopped()==true) return(0);
     // 长运行时间循环
     // ...
    }
```

# IsTesting( )返回测试模式状态

bool IsTesting( )
如果智能交易在测试模式中运行，返回 TRUE 。否则，返回 FALSE。
示例:

```
  if(IsTesting()) Print("测试中");
```

# IsTradeAllowed( )返回允许智能交易

bool IsTradeAllowed( )
如果智能交易允许交易，返回 TRUE 。否则，返回 FALSE。
参见 IsDllsAllowed(), IsLibrariesAllowed(), IsTradeContextBusy()。
示例:

```
  if(IsTradeAllowed()) Print("允许交易");
```

# IsTradeContextBusy( )返回其他智能交易忙

bool IsTradeContextBusy( )
如果其他智能交易交易忙，返回 TRUE。否则，返回 FALSE。
参见 IsTradeAllowed().
示例:

```
if(IsTradeContextBusy()) Print("交易文本忙，请稍等");
```

# IsVisualMode( )返回智能交易“图片模式”

bool IsVisualMode( )
如果智能交易用"图片模式"测试，返回 TRUE 。否则，返回 FALSE。
示例:

```
if(IsVisualMode()) Comment("Visual mode turned on");
```

# UninitializeReason( )返回智能交易初始化原因

int UninitializeReason( )
返回智能交易，自定义指标和脚本的未初始化原因代码。返回值为未初始化原因代码之一。
此函数同样可以在函数 init() 中调用分析先前开启初始化原因。
示例:

```
// 这是范例
int deinit()
  {
   switch(UninitializeReason())
     {
      case REASON_CHARTCLOSE:
      case REASON_REMOVE:        CleanUp(); break; // 清理和抽空所有源代码
      case REASON_RECOMPILE:
      case REASON_CHARTCHANGE:
      case REASON_参量:
      case REASON_ACCOUNT:       StoreData(); break;   // 准备重新开始
     }
   //...
  }
```

# Client terminal 客户端信息

函数返回的客户终端信息。

## TerminalCompany( )返回客户端所属公司

string TerminalCompany( )
返回所属客户端公司名称。
示例:
Print("公司名称 ",TerminalCompany());

## TerminalName( )返回客户端名称

string TerminalName( )
返回客户端名称。
示例:
Print("终端名称",TerminalName());

## TerminalPath( )返回客户端文件路径

string TerminalPath( )
从被开启的客户端返回文件目录。
示例:
Print("工作目录",TerminalPath());

# Common functions 常规命令函数

常规命令函数不包括特殊函数。

## Alert 弹出警告窗口

void Alert( ...)
弹出一个显示信息的警告窗口。 参量可以使任意类型。通过参量总数不得超过 64。
对于警报函数数组不能通过。数组可以作为输出元素。
双重数据类型可以输入到小数点后 4 位。 输入数据使用 DoubleToStr()函数更为精确。
bool 数据, 时间和颜色类型警作为数字类型输入。
时间类型值作为数组使用 TimeToStr()函数输入。
参见 Comment() 和 Print() 函数。
参量:
... - 任意值，如有多个可用逗号分割 。最多为 64 个参量。
示例:

```
if(Close[0]>SignalLevel)
   Alert("收盘价进入 ", Close[0],"!!!");
```

## Comment 显示信息在走势图左上角

void Comment( ...)
显示信息在走势图左上角 。参量可以使任意类型。通过参量总数不得超过 64。
对于警报函数数组不能通过。数组可以作为输出元素。
双重数据类型可以输入到小数点后 4 位。 输入数据使用 DoubleToStr()函数更为精确。
bool 数据, 时间和颜色类型警作为数字类型输入。
bool 数据, 时间和颜色类型警作为数字类型输入。
时间类型值作为数组使用 TimeToStr()函数输入。
参见 Comment() 和 Print() 函数。
参量:
... - =任意值，如有多个可用逗号分割 。最多为 64 个参量。
示例:

```
double free=AccountFreeMargin();
Comment(" 账 户 自 由 保 证 金 ",DoubleToStr(free,2),"\n","Current time is
",TimeToStr(TimeCurrent()));
```

## GetTickCount 获取时间标记

int GetTickCount( )
使用 GetTickCount()函数取时间标记，函数取回用毫秒标示的时间标记。

示例:

```
int start=GetTickCount();
// 计算...
Print("Calculation time is ", GetTickCount()-start, " milliseconds.");
```

# MarketInfo 在市场观察窗口返回不同数据保证金列表

double MarketInfo( string symbol, int type)
在市场观察窗口返回不同数据保证金列表。 当前保证金的部分描述存储在预定义变量。
参量:
symbol - 货币对保证金。
type - 指定类别的请求识别符信息返回。可以是 请求识别码的任意值。

示例:

```
double bid      =MarketInfo("EURUSD",MODE_BID);
double ask      =MarketInfo("EURUSD",MODE_ASK);
double point =MarketInfo("EURUSD",MODE_POINT);
int       digits=MarketInfo("EURUSD",MODE_DIGITS);
int       spread=MarketInfo("EURUSD",MODE_SPREAD);
```

# MessageBox 创建信息窗口

int MessageBox( void text, void caption, void flags)
在信息箱内可以创建，展示和控制信息箱。信息箱包含信息和题头。如果函数成功运行，MessageBox 函数 返回代码值为其中值之一。
此函数从客户端的工作页面不能调用执行。
参量:
text - 窗口显示的文字。
caption - 窗口上显示的标题。如果参量为 NULL, 智能交易名称将被隐藏。
flags - 窗口选项开关。选项 开关存在组。
示例:

```
#include <WinUser32.mqh>
if(ObjectCreate("text_object", OBJ_TEXT, 0, D'2004.02.20 12:30', 1.0045)==false)
   {
    int ret=MessageBox(" ObjectCreate() function returned the "+GetLastError()+" error\nContinue?", "Question", MB_YESNO|MB_ICONQUESTION);
    if(ret==IDNO) return(false);
   }
// 继续
```

# PlaySound 播放声音

void PlaySound( string filename)
函数播放声音文件。文件必须载入目录 terminal_dir\sounds 或子目录内。
参量:
filename - 音频文件名 。
示例:

```
if(IsDemo()) PlaySound("alert.wav");
```

# Print 窗口中显示文本

void Print( ...)
将文本打印在结果窗口内 。 参量可以使任意类型。通过参量总数不得超过 64。
对于 Print()函数数组不能通过。数组可以作为输出元素。
双重数据类型可以输入到小数点后 4 位。 输入数据使用 DoubleToStr()函数更为精确。
bool 数据, 时间和颜色类型警作为数字类型输入。
时间类型值作为数组使用 TimeToStr()函数输入。
参见 Comment() 和 Print() 函数。
参量:
... - 任意值，如有多个可用逗号分割 。最多为 64 个。
示例:

```
Print("当前自由保证金 ", AccountFreeMargin());
Print("当前时间 ", TimeToStr(TimeCurrent()));
double pi=3.141592653589793;
Print("PI number is ", DoubleToStr(pi,8));
// 输入数据: PI number is 3.14159265
// 数组打印
for(int i=0;i<10;i++)
   Print(关闭[i]);
```

# SendFTP 设置 FTP

bool SendFTP( string filename, void ftp_path)
设置在工具>选项>公开标签内发送文件到 FTP 服务器。如果尝试失败, 返回 FALSE。
在测试的模式下作用不能控制。作用可以从客户指标或其他中运作。
发送的文件必须储存在 terminal_directory\experts\files 文件夹或子文件夹内。
如果不存在 FTP 地址或者指定密码，文件不会传送。
参量:
filename - 发送文件。
ftp_path - FTP 通道。如果没有制定通道，会应用设置中的描述通道。
示例:

```
int lasterror=0;
```

```
if(!SendFTP("report.txt"))
   lasterror=GetLastError();
```

# SendMail 设置 Email

void SendMail( string subject, string some_text)
设置在工具>选项 >EMail 标签内发送电子邮件。
可以设置禁止此项功能， 或者是省略电子邮件地址。获得详细错误信息，查看 GetLastError() 函数。
参量:
subject - 文本。
some_text - 邮件。
示例:

```
double lastclose=Close[0];
if(lastclose<my_signal)
   SendMail("从你的智能交易", "价格下降到"+DoubleToStr(lastclose,Digits));
```

# Sleep 指定的时间间隔内暂停交易业务

void Sleep( int milliseconds)
The Sleep()函数是指在指定的时间间隔内暂停交易业务
The Sleep()函数不能在客户指标内计算。
当进入函数停止状态 智能交易每 0.1 second 秒会检测。
参量:
milliseconds - 毫秒之内的内部睡眠。.
示例:

```
//---- 等待 10 秒
  Sleep(10000);
```

# Conversion functions 格式转换函数

从一种格式转换到另一种格式提供数据的一组函数。
必须特别注意 NormalizeDouble()函数它提供了价格介绍的必要的准确性。 在交易的操作中，如果当前的数字超出交易服务器的需求，意味着没有任何标准的价格可以使用。

## CharToStr 字符转换成字符串

string CharToStr( int char_code)
将字符型转换成字符串型结果。
参量:
char_code - 字符的 ACSII 码 。
示例:

```
string str="WORL" + CharToStr(44); // 'D'的 44 个代码。
// 结果字串符将被 WORLD
```

## DoubleToStr 双精度浮点转换成字符串

string DoubleToStr( double value, int digits)
将双精度浮点型转换成字符串型的结果返回。
参量:
value - 浮点型数字。
digits - 精确格式，小数点后位(0-8)。
示例:

```
string value=DoubleToStr(1.28473418, 5);
// 值为"1.28473"
```

## NormalizeDouble 给出环绕浮点值的精确度

double NormalizeDouble( double value, int digits)
给出环绕浮点值的精确度。返回双重类型的正常化。
计算止损和赢利值，挂单交易交易的开盘价必须正常化。精确值需要在小数点中预定义。
参量:
value - 浮点值。
digits - 精确格式，小数点之后的精确数字 (0-8)。
示例:

```
double var1=0.123456789;
Print(DoubleToStr(NormalizeDouble(var1,5),8));
// 输入信息: 0.12346000
```

# StrToDouble 字符串型转换成双精度浮点型

double StrToDouble( string value)
将字符串型转换成双精度浮点型结果。
参量:
value - 数字的字符串转换格式 。
示例:

```
double var=StrToDouble("103.2812");
```

# StrToInteger 字符串型转换成整型

int StrToInteger( string value)
将字符串型转换成整型结果。
参量:
value - 数字的字符串转换格式。
示例:

```
int var1=StrToInteger("1024");
```

# StrToTime 字符串型转换成时间型

datetime StrToTime( string value)
将字符串型转换成时间型，输入格式为 "yyyy.mm.dd hh:mi"
参量:
value - 日期时间的字串符格式为 "yyyy.mm.dd hh:mi"。
示例:

```
datetime var1;
var1=StrToTime("2003.8.12 17:35");
var1=StrToTime("17:35");          // 返回当前给出日期
var1=StrToTime("2003.8.12");   // 返回午夜时间日期"00:00"
```

# TimeToStr 时间类型转换为 "yyyy.mm.dd hh:mi"格式

string TimeToStr( datetime value, void mode)
时间类型(从 1970 1 月 1 日通过的相当数量秒数）转换为 "yyyy.mm.dd hh:mi"格式。
参量:
value - 自 1970 年 1 月所通过的数量秒数。
mode - 选择数据输出模式可以是以下的一个或者组合:
TIME_DATE 结果格式为 "yyyy.mm.dd",
TIME_MINUTES 结果格式为 "hh:mi",
TIME_SECONDS 结果格式为 "hh:mi:ss"。
示例:

```
string var1=TimeToStr(TimeCurrent(),TIME_DATE|TIME_SECONDS);
```

# Custom indicators 自定义指标

自定义指标中使用的一组函数。
这些函数不能在智能交易和脚本中使用。

## IndicatorBuffers

void IndicatorBuffers( int count)
对于缓冲储存器分配记忆应用自定义指标计算。缓冲储存器的总数不能超过 8 或者是小于自定义缓冲 属性中所给出的值。 如果客户指标要求另外的缓冲器计数，那么这个功能必须使用为指定总额缓冲。
参量:
count - 在指标缓冲器和 8 缓冲储存器之间分配缓冲储存器的总量。
示例:

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_color1 Silver
//---- 自定义参量
extern int FastEMA=12;
extern int SlowEMA=26;
extern int SignalSMA=9;
//---- 自定义缓冲
double ind_buffer1[];
double ind_buffer2[];
double ind_buffer3[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function |
//+------------------------------------------------------------------+
int init()
  {
//---- 使用 2 个添加缓冲。
   IndicatorBuffers(3);
//---- 画出设定
   SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3);
   SetIndexDrawBegin(0,SignalSMA);
   IndicatorDigits(MarketInfo(Symbol(),MODE_DIGITS)+2);
//---- 绘制 3 个添加缓冲位置
   SetIndexBuffer(0,ind_buffer1);
   SetIndexBuffer(1,ind_buffer2);
   SetIndexBuffer(2,ind_buffer3);
//---- DataWindow 和自定义窗口标签名称
```

```
    IndicatorShortName("OsMA("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- 初始化结束
    return(0);
  }
```

# IndicatorCounted

int IndicatorCounted( )
在自定义最后一次开启之后，函数返回柱的总数不会改变。计算过的柱数无须重新计算。大多数情况下，同样数额的索引值不需要重估。函数应用到优化计算中。
注解: 最近的柱无须考虑计算，在多数情况下，这个柱是要被重估的。不过，自定义指标显示交易中的新柱的第一替克。 可能先前柱的最后一个替克没有处理的结果（因为在最后 一个替克进入时倒数第二个没有 处理 完成)客定义标将不会显示和计算。 在这样的情况下为了避免错误， IndicatorCounted()函数会返回前一个柱。
示例:

```
  int start()
    {
      int limit;
      int counted_bars=IndicatorCounted();
//---- 检验可能出现错误
      if(counted_bars<0) return(-1);
//---- 最后数的柱将被重数
      if(counted_bars>0) counted_bars--;
      limit=Bars-counted_bars;
//---- 主环
      for(int i=0; i<limit; i++)
        {
         //---- ma_shift set to 0 because SetIndexShift called abowe
         ExtBlueBuffer[i]=iMA(NULL,0,JawsPeriod,0,MODE_SMMA,PRICE_MEDIAN,i);
         ExtRedBuffer[i]=iMA(NULL,0,TeethPeriod,0,MODE_SMMA,PRICE_MEDIAN,i);
         ExtLimeBuffer[i]=iMA(NULL,0,LipsPeriod,0,MODE_SMMA,PRICE_MEDIAN,i);
        }
//---- 完成
      return(0);
    }
```

# IndicatorDigits

void IndicatorDigits( int digits)
设置精确格式(计数数字在小数点以后)使自定义值直观化。货币对精确价格 为默认值。 指标会添加到图表中。
参量:
digits - 精确格式。

示例:

```
int init()
  {
//---- 使用及计算 2 个添加缓冲。
    IndicatorBuffers(3);
//---- 画出参量设置
    SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3);
    SetIndexDrawBegin(0,SignalSMA);
    IndicatorDigits(Digits+2);
//----一个自定义的 3 个缓冲
    SetIndexBuffer(0,ind_buffer1);
    SetIndexBuffer(1,ind_buffer2);
    SetIndexBuffer(2,ind_buffer3);
//---- DataWindow 和自定义子窗口"简称"
    IndicatorShortName("OsMA("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- 初始化完成
    return(0);
  }
```

# IndicatorShortName

void IndicatorShortName( string name)

设置显示在数据窗口和子窗口中自定义指标的"简称"。

参量:

name - 新简称。

示例:

```
int init()
  {
//----使用计算 2 个添加缓冲
    IndicatorBuffers(3);
//---- 画出设定
    SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3);
    SetIndexDrawBegin(0,SignalSMA);
    IndicatorDigits(MarketInfo(Symbol(),MODE_DIGITS)+2);
//---- 绘制 3 个添加缓冲位置
    SetIndexBuffer(0,ind_buffer1);
    SetIndexBuffer(1,ind_buffer2);
    SetIndexBuffer(2,ind_buffer3);
//---- DataWindow 和自定义子窗口标签名称
    IndicatorShortName("OsMA("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- 初始化完成
    return(0);
  }
```

# SetIndexArrow

void SetIndexArrow( int index, int code)
设置 DRAW_ARROW 类型的自定义线为一个箭头货币对。
箭头代码范围限于 33 到 255 之间。
参量:
index - 索引线。必须在 0 至 7 之间。
code - 来自 Wingdings 或 数组常数的货币对代码。
示例:

```
int init()
  {
//---- 2 个自定义缓冲
      SetIndexBuffer(0,ExtUppperBuffer);
      SetIndexBuffer(1,ExtLowerBuffer);
//---- 绘制参量设置
      SetIndexStyle(0,DRAW_ARROW);
      SetIndexArrow(0,217);
      SetIndexStyle(1,DRAW_ARROW);
      SetIndexArrow(1,218);
//---- 在 DataWindow 窗口显示
      SetIndexLabel(0,"Fractal Up");
      SetIndexLabel(1,"Fractal Down");
//---- 初始化完成
    return(0);
  }
```

# SetIndexBuffer

bool SetIndexBuffer( int index, double array[])
对于自定义指标预定义的缓冲器绑定全球水平。需要使用 IndicatorBuffers() 函数计算缓冲器的总数并且不能超过 8。如果成功，返回 TRUE，否则将返回 FALSE。获得详细信息，请查看 GetLastError()函数。
参量:
index - 索引线。必须在 0 至 7 之间。
array[] - 数组存储计算指标值。
示例:

```
  double ExtBufferSilver[];
  int init()
    {
      SetIndexBuffer(0, ExtBufferSilver); // 第一线缓冲
      // ...
    }
```

# SetIndexDrawBegin

void SetIndexDrawBegin( int index, int begin)
从给出指标线画出必须开始设置柱数字(从数据开始) 。指标线会从左到右画出所给出指标数组值左边部分不会显示在图表或数据窗口中。设置0作为默认值, 随后，所有数据将得出。
参量:
index - 索引线。必须在 0 至 7 之间。
begin - 第一个画出柱的数字位置。
示例:

```
int init()
  {
//----使用计算 2 个添加缓冲
   IndicatorBuffers(3);
//----画出设定
   SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3);
   SetIndexDrawBegin(0,SignalSMA);
   IndicatorDigits(MarketInfo(Symbol(),MODE_DIGITS)+2);
//---- 绘制 3 个添加缓冲位置
   SetIndexBuffer(0,ind_buffer1);
   SetIndexBuffer(1,ind_buffer2);
   SetIndexBuffer(2,ind_buffer3);
//---- DataWindow 和自定义子窗口标签名称
   IndicatorShortName("OsMA("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- 初始化完成
   return(0);
  }
```

# SetIndexEmptyValue

void SetIndexEmptyValue( int index, double value)
设置图画线省缺值。省缺值不得出或不被显示在 DataWindow 。省缺值是 EMPTY_VALUE。
参量:
index - 索引线。必须在 0 至 7 之间。
value - 新 "省缺值"。
示例:

```
int init()
  {
//----2 个添加缓冲
    SetIndexBuffer(0,ExtUppperBuffer);
    SetIndexBuffer(1,ExtLowerBuffer);
//---- 画出参量设置
    SetIndexStyle(0,DRAW_ARROW);
    SetIndexArrow(0,217);
```

```
      SetIndexStyle(1,DRAW_ARROW);
      SetIndexArrow(1,218);
//---- 值 0 不显示
      SetIndexEmptyValue(0,0.0);
      SetIndexEmptyValue(1,0.0);
//---- 在 DataWindow 窗口不显示
      SetIndexLabel(0,"Fractal Up");
      SetIndexLabel(1,"Fractal Down");
//---- 初始化完成
    return(0);
  }
```

# SetIndexLabel

void SetIndexLabel( int index, string text)
在 DataWindow 和 tooltip 中设置图画线描述。
参量:
index - 索引线。必须在 0 至 7 之间。
text - 标签文本。 NULL 表示索引值在 DataWindow 不显示。
示例:

```
//+------------------------------------------------------------------+
//| Ichimoku Kinko Hyo initialization function                       |
//+------------------------------------------------------------------+
int init()
  {
//----
    SetIndexStyle(0,DRAW_LINE);
    SetIndexBuffer(0,Tenkan_Buffer);
    SetIndexDrawBegin(0,Tenkan-1);
    SetIndexLabel(0,"Tenkan Sen");
//----
    SetIndexStyle(1,DRAW_LINE);
    SetIndexBuffer(1,Kijun_Buffer);
    SetIndexDrawBegin(1,Kijun-1);
    SetIndexLabel(1,"Kijun Sen");
//----
    a_begin=Kijun; if(a_begin<Tenkan) a_begin=Tenkan;
    SetIndexStyle(2,DRAW_HISTOGRAM,STYLE_DOT);
    SetIndexBuffer(2,SpanA_Buffer);
    SetIndexDrawBegin(2,Kijun+a_begin-1);
    SetIndexShift(2,Kijun);
//---- 在 DataWindow 窗口 Up Kumo 线不显示
    SetIndexLabel(2,NULL);
    SetIndexStyle(5,DRAW_LINE,STYLE_DOT);
```

```
   SetIndexBuffer(5,SpanA2_Buffer);
   SetIndexDrawBegin(5,Kijun+a_begin-1);
   SetIndexShift(5,Kijun);
   SetIndexLabel(5,"Senkou Span A");
//----
   SetIndexStyle(3,DRAW_HISTOGRAM,STYLE_DOT);
   SetIndexBuffer(3,SpanB_Buffer);
   SetIndexDrawBegin(3,Kijun+Senkou-1);
   SetIndexShift(3,Kijun);
//---- 在 DataWindow 窗口 Down Kumo 线不显示
   SetIndexLabel(3,NULL);
//----
   SetIndexStyle(6,DRAW_LINE,STYLE_DOT);
   SetIndexBuffer(6,SpanB2_Buffer);
   SetIndexDrawBegin(6,Kijun+Senkou-1);
   SetIndexShift(6,Kijun);
   SetIndexLabel(6,"Senkou Span B");
//----
   SetIndexStyle(4,DRAW_LINE);
   SetIndexBuffer(4,Chinkou_Buffer);
   SetIndexShift(4,-Kijun);
   SetIndexLabel(4,"Chinkou Span");
//----
   return(0);
  }
```

# SetIndexShift

void SetIndexShift( int index, int shift)

撤销画线设置。对于仓位值，画线将会平移到右侧或是平移到左侧。在当前柱计算值将被平移到相应的柱。

参量:

index - 索引线。必须在 0 至 7 之间。

shift - 平移植。

示例:

```
//+------------------------------------------------------------------+
//| Alligator initialization function                                |
//+------------------------------------------------------------------+
int init()
  {
//----当画出时线平移
   SetIndexShift(0,JawsShift);
   SetIndexShift(1,TeethShift);
   SetIndexShift(2,LipsShift);
```

```
//---- 当画出时越过地一个位置
    SetIndexDrawBegin(0,JawsShift+JawsPeriod);
    SetIndexDrawBegin(1,TeethShift+TeethPeriod);
    SetIndexDrawBegin(2,LipsShift+LipsPeriod);
//---- 绘制 3 个添加缓冲位置
    SetIndexBuffer(0,ExtBlueBuffer);
    SetIndexBuffer(1,ExtRedBuffer);
    SetIndexBuffer(2,ExtLimeBuffer);
//---- 画出设定
    SetIndexStyle(0,DRAW_LINE);
    SetIndexStyle(1,DRAW_LINE);
    SetIndexStyle(2,DRAW_LINE);
//---- 索引标签
    SetIndexLabel(0,"Gator Jaws");
    SetIndexLabel(1,"Gator Teeth");
    SetIndexLabel(2,"Gator Lips");
//---- 初始化完成
    return(0);
  }
```

# SetIndexStyle

void SetIndexStyle( int index, int type, void style, void width, void clr)
设置新型、样式、宽度和颜色为一条指定的显示线。
参量:
index - 索引线。必须在 0 至 7 之间。
type - 样式风格。可以是划线风格列表其中一个。
style - 画线风格。可以应用单线。可以是划线风格列表其中一个。 EMPTY 值表示风格不变。
width - 线的宽度。线的宽度可以是 1,2,3,4,5。 EMPTY 值表示着风格不变。
clr - 线的颜色。.现存的参量表示颜色将不会改变。
示例:

```
SetIndexStyle(3, DRAW_LINE, EMPTY, 2, Red);
```

# SetLevelStyle

void SetLevelStyle( int draw_style, int line_width, void clr)
函数设置指标水平线输入新型、样式、宽度和颜色输入数据。
参量:
draw_style - 画线风格。可以应用单线。可以是划线风格列表其中一个。 EMPTY 值表示风格不变。
line_width - 线的宽度。线的宽度可以是 1,2,3,4,5。 EMPTY 值表示着风格不变。
clr - 线的颜色。.现存的参量表示颜色将不会改变。

示例:

```
//---- 红色单线显示水平
    SetLevelStyle(STYLE_SOLID,2,Red)
```

# SetLevelValue

void SetLevelValue( int level, double value)

函数设置输入数据指标的水平线。

参量:

level - 水平索引(0-31)。

value - 给出的指标水平值。

示例:

```
SetLevelValue(1,3.14);
```

# Date & Time functions 日期时间函数

表示时间类型数据的一组函数(从 1970 年 1 月 1 日午夜开始以秒为单位计算)。

## Day

int Day( )
返回这个月的当天,最后一次访问服务器的时间。
注解: 在测试中, 时间格式为最后设定的服务器模式。
示例:

```
if(Day()<5) return(0);
```

## DayOfWeek

int DayOfWeek( )
返回这周的星期数，(0-星期天,1,2,3,4,5,6 以此类推)来自最后已知的服务器上的时间。
注解: 在测试中, 时间格式为最后设置的服务器模式。
示例:

```
// 假期不工作
if(DayOfWeek()==0 || DayOfWeek()==6) return(0);
```

## DayOfYear

int DayOfYear( )
返回年的当天(1 代表 1 月 1 日.., 365(6) 就是 12 月 31 日)， 最后访问服务器的时间。
注解: 在测试中, 时间格式为最后已知的服务器模式。
示例:

```
if(DayOfYear()==245)
   return(true);
```

## Hour

int Hour( )
在程序开始以前的片刻，返回小时数(0,1,2,..23) 最后访问的服务器时间(在程序执行之内的时期这个值不会改变) 。
注解: 在测试中, 时间格式为最后设置的服务器模式。
示例:

```
bool is_siesta=false;
if(Hour()>=12 || Hour()<17)
```

```
is_siesta=true;
```

# Minute

int Minute( )
在程序开始以前的片刻，返回当前的分钟(0,1,2,..59)最后访问的服务器时间(在程序执行期间这个值不会改变) 。
示例:

```
if(Minute()<=15)
   return("first quarter");
```

# Month

int Month( )
在程序开始以前的片刻，返回当前的月数(1,2,..12)最后访问的服务器时间(在程序执行期间这个值不会改变) 。
注解: 在测试中, 时间格式为最后设定的服务器模式。
示例:

```
if(Month()<=5)
   return("the first half year");
```

# Seconds

int Seconds( )
在程序开始以前的片刻，返回当前的秒数作为数字最后访问的服务器时间(在程序执行期间这个值不会改变) 。
示例:

```
if(Seconds()<=15)
   return(0);
```

# TimeCurrent

datetime TimeCurrent( )
返回最后访问的服务器时间(最新的行情输入时间)作为秒钟数字从 00:00 1970 年 1 月 1 日开始。
注解: 在测试中, 时间格式为最后设定的服务器模式。
示例:

```
if(TimeCurrent()-OrderOpenTime()<360) return(0);
```

# TimeDay

int TimeDay( datetime date)
返回输入日期中的日期（1-31）
参量:
date - 作为秒钟的数字从 00:00 1970 年 1 月 1 日开始。
示例:

```
int day=TimeDay(D'2003.12.31');
// 天数为 31
```

# TimeDayOfWeek

int TimeDayOfWeek( datetime date)
返回从零开始的星期中的第几天(0 代表星期天, 1, 2, 3, 4, 5, 6) 为指定日期。
参量:
date - 作为秒钟的数字，从 00:00 1970 年 1 月 1 日开始。
示例:

```
int weekday=TimeDayOfWeek(D'2004.11.2');
// 数字 2 - 星期二
```

# TimeDayOfYear

int TimeDayOfYear( datetime date)
返回一年中的日数(1 意味 1 月 1 日.., 365(6) 表示 12 月 31 日)为指定日期。
参量:
date - 作为秒钟的数字，从 00:00 1970 年 1 月 1 日开始。
示例:

```
int day=TimeDayOfYear(TimeCurrent());
```

# TimeHour

int TimeHour( datetime time)
返回小时为指定的时间。
参量:
time - 作为秒钟的数字，从 00:00 1970 年 1 月 1 日开始。
示例:

```
int h=TimeHour(TimeCurrent());
```

# TimeLocal

datetime TimeLocal( )

返回当前电脑时间，从 00:00 1970 年 1 月 1 日开始。
注解: 在测试中, 时间格式为最后设定的服务器模式。
示例:

```
if(TimeLocal()-OrderOpenTime()<360) return(0);
```

# TimeMinute

int TimeMinute( datetime time)
返回分钟为指定的时间。
参量:
time - 作为秒钟的数字，从 00:00 1970 年 1 月 1 日开始。
示例:

```
int m=TimeMinute(TimeCurrent());
```

# TimeMonth

int TimeMonth( datetime time)
返回月数为指定的时间。
参量:
time - 作为秒钟的数字，从 00:00 1970 年 1 月 1 日开始。
示例:

```
int m=TimeMonth(TimeCurrent());
```

# TimeSeconds

int TimeSeconds( datetime time)
返回秒数为指定的时间。
参量:
time - 作为秒钟的数字，从 00:00 1970 年 1 月 1 日开始。
示例:

```
int m=TimeSeconds(TimeCurrent());
```

# TimeYear

int TimeYear( datetime time)
返回年数为指定的时间。返回值的范围可以在 1970 到 2037 之间。
参量:
time - 作为秒钟的数字，从 00:00 1970 年 1 月 1 日开始。
示例:

```
int y=TimeYear(TimeCurrent());
```

# Year

int Year( )
返回本年度的年数字，即，服务器的年数时间。
注解: 在测试中, 时间格式为最后设定的服务器模式。
示例:

```
// 如果时间范围在 2006 年 1 月到 4 月 30 日之间，返回。
    if(Year()==2006 && Month()<5)
    return(0);
```

# File functions 文件函数

一组文件运行函数。
三个文件目录(补充指南)放置的地方:
/HISTORY/<current broker> - FileOpenHistory 函数;
/EXPERTS/FILES - 常规状况;
/TESTER/FILES - 专门测试.
来自其他目录的工作文件禁止。

## FileClose 关闭文件

void FileClose( int handle)
用 FileOpen() 函数打开先前已关闭的文件。
参量:
handle - 用 FileOpen()函数返回句柄。
示例:

```
int handle=FileOpen("filename", FILE_CSV|FILE_READ);
if(handle>0)
  {
   // 运行文件 ...
   FileClose(handle);
  }
```

## FileDelete 删除文件

void FileDelete( string filename)
删除指定的文件名。获得详细的错误信息, 查看 GetLastError()函数。
如果他们是在 terminal_dir\experts\files 目录 (terminal_directory\tester\files, 在测试的情况下)或它的补充指南，只删除单个文件。

```
int lastError;
FileDelete("my_table.csv");
lastError=GetLastError();
if(laseError!=ERR_NOERROR)
  {
   Print("错误 (",lastError,") 删除文件 my_table.csv");
   return(0);
  }
```

参量:
filename - 目录和文件名 。
示例:

```
//文件 my_table.csv 将从目录 terminal_dir\experts\files directory 删除
```

# FileFlush 将缓存中的数据刷新到磁盘上去

void FileFlush( int handle)
将缓存中的数据刷新到磁盘上去 。
注解：FileFlush() 函数只有在文件被读或写中显示。
所有关闭的文件会自动从储存缓冲器上删除。所以在调用 FileClose() 函数之前不需要调用 FileFlush() 函数。
参量:
handle - 用 FileOpen()函数返回的句柄。
示例:

```
int bars_count=Bars;
int handle=FileOpen("mydat.csv",FILE_CSV|FILE_WRITE);
if(handle>0)
  {
   FileWrite(handle, "#","OPEN","CLOSE","HIGH","LOW");
   for(int i=0;i<bars_count;i++)
      FileWrite(handle, i+1,Open[i],Close[i],High[i], Low[i]);
   FileFlush(handle);
   ...
   for(int i=0;i<bars_count;i++)
      FileWrite(handle, i+1,Open[i],Close[i],High[i], Low[i]);
   FileClose(handle);
  }
```

# FileIsEnding 文件结尾

bool FileIsEnding( int handle)
如果文件指针是在文件的末端,返回逻辑配齐， 否则返回 false。 获得详细的错误信息, 查看 GetLastError() 函数。 如果文件末端在只读期间到达， GetLastError() 函数将返回 错误 ERR_END_OF_FILE (4099)。
参量:
handle - 用 FileOpen()函数返回的句柄。
示例:

```
if(FileIsEnding(h1))
  {
   FileClose(h1);
   return(false);
  }
```

# FileIsLineEnding

bool FileIsLineEnding( int handle)
如果 CSV 文件指针是在文件的末端,返回逻辑配齐， 否则返回 false. 获得详细的错误信息, 查看 GetLastError() 函数。
参量:
handle - 用 FileOpen()函数返回的句柄。
示例:

```
  if(FileIsLineEnding(h1))
    {
     FileClose(h1);
     return(false);
    }
```

# FileOpen 打开文件

int FileOpen( string filename, int mode, void delimiter)
为输入或输出信息打开文件。如果函数失败，返回打开文件或-1 。获得详细的错误信息, 查看 GetLastError() 函数。
注解: 文件可能只在 terminal_directory\experts\files 文件夹(terminal_directory\tester\files 或在它的子文件夹内被打开。
FILE_BIN 和 FILE_CSV 格式不能同时使用。
如果 FILE_WRITE 与 FILE_READ 不结合, 被打开的文件长度为零。如果还有一些包含数据的文件，它们将被删除 。如果需要对现存文件添加数据，必须使用 FILE_READ 和 FILE_WRITE 文件组合打开 。
如果 FILE_READ 与 FILE_WRITE 不结合, 仅仅会打开现存文件。如果文件不存在, 可以使用 FILE_WRITE 创建。
在一个板块内最多能够同时执行 32 个文件。
参量:
filename - 文件名称
mode - 打开模式。可以是以下的一种或是组合: FILE_BIN, FILE_CSV, FILE_READ, FILE_WRITE。
delimiter - csv 文件的限定。默认值为';' 符号。
示例:

```
  int handle;
  handle=FileOpen("my_data.csv",FILE_CSV|FILE_READ,';');
  if(handle<1)
    {
     Print("未找到 my_data.dat 文件，错误", GetLastError());
     return(false);
    }
```

# FileOpenHistory 历史目录中打开文件

int FileOpenHistory( string filename, int mode, void delimiter)
在当前的历史目录(terminal_directory\history\server_name)或在它的子文件内打开文件 。如果函数失败，返回文件描述部分或-1 。获得详细的错误信息，查看 GetLastError()函数。
注解: 客户终端可能连接到不同经纪公司的服务器。每个经纪公司的历史数据(HST 文件)会存储在 terminal_directory\history 相对应的子文件夹内。
文件在脱机时同样可以打开，不会有数据进入。
参量:
filename - 文件名称
mode - 打开模式。可以是以下的一种或是组合：FILE_BIN，FILE_CSV，FILE_READ，FILE_WRITE。
delimiter - csv 文件的限定。默认值为';' 符号。
示例:

```
int handle=FileOpenHistory("USDX240.HST",FILE_BIN|FILE_WRITE);
if(handle<1)
  {
   Print("不能创建 USDX240.HST 文件");
   return(false);
  }
// 运行文件
// ...
FileClose(handle);
```

# FileReadArray 将二进制文件读取到数组中

int FileReadArray( int handle, void array[], int start, int count)
将二进制文件读取到数组中，返回读取的条数。
获得详细的错误信息，查看 GetLastError() 函数。
参量:
handle - 用 FileOpen()函数返回的句柄。
array[] - 写入的数组。
start - 在数组中存储的开始点。
count - 读取多少个对象 。
示例:

```
int handle;
double varray[10];
handle=FileOpen("filename.dat", FILE_BIN|FILE_READ);
if(handle>0)
  {
   FileReadArray(handle, varray, 0, 10);
   FileClose(handle);
  }
```

# FileReadDouble 从文件中读取浮点型数据

double FileReadDouble( int handle, void size)
从文件中读取浮点型数据，数字可以是 8byte 的 double 型或者是 4byte 的 float 型。
获得错误信息，请查看 GetLastError() 函数。
参量:
handle - 用 FileOpen()函数返回的句柄。
size - 数字格式大小，DOUBLE_VALUE(8 bytes) 或者 FLOAT_VALUE(4 bytes)。
示例:

```
int handle;
double value;
handle=FileOpen("mydata.dat",FILE_BIN);
if(handle>0)
  {
   value=FileReadDouble(handle,DOUBLE_VALUE);
   FileClose(handle);
  }
```

# FileReadInteger 从当前二进制文件读取整形型数据

int FileReadInteger( int handle, void size)
函数从当前二进制文件读取整形型数据，数字可以是 1,2,4byte 的长度 。如果格式大小不被指定，系统设法读 4 字节的值。获得详细的错误信息，请查看 GetLastError() 函数。
参量:
handle - 用 FileOpen()函数返回的句柄。
size - 数字格式大小，CHAR_VALUE(1 byte), SHORT_VALUE(2 bytes) 或者 LONG_VALUE(4 bytes)。
示例:

```
int handle;
int value;
handle=FileOpen("mydata.dat", FILE_BIN|FILE_READ);
if(handle>0)
  {
   value=FileReadInteger(h1,2);
   FileClose(handle);
  }
```

# FileReadNumber

double FileReadNumber( int handle)
从当前文件位置在符号之前读取数字。只能为 CSV 文件。
获得详细的错误信息，查看 GetLastError()函数。

参量:
handle - 用 FileOpen()函数返回的句柄。
示例:

```
int handle;
int value;
handle=FileOpen("filename.csv", FILE_CSV, ';');
if(handle>0)
  {
   value=FileReadNumber(handle);
   FileClose(handle);
  }
```

# FileReadString 从当前文件位置读取字串符

string FileReadString( int handle, void length)
函数从当前文件位置读取字串符。适用于 CSV 和二进制文件。在文本文件中字串符在符号之前被读取。对于二进制文件，被测量的计数将读取字串符。获得详细的错误信息，查看 GetLastError()函数。
参量:
handle - 用 FileOpen()返回的句柄。
length - 读取字符串长度 。
示例:

```
int handle;
string str;
handle=FileOpen("filename.csv", FILE_CSV|FILE_READ);
if(handle>0)
  {
   str=FileReadString(handle);
   FileClose(handle);
  }
```

# FileSeek 文件指针移动

bool FileSeek( int handle, int offset, int origin)
在字节上函数移动文件指针是垂距的，从开始的一个新的位置指向末端或者当前文件位置。接下来读取或写入放置在一个新位置。
如果文件指针成功地被移动了，函数返回 TRUEe， 否则，它返回 FALSE. 获得详细的错误信息，查看 GetLastError() 函数。
参量:
handle - 用 FileOpen()函数返回的句柄。
offset - 设置的原点。
origin - 最初的位置。 值可以是以下任意常数:
SEEK_CUR - 当前位置,

SEEK_SET - 开始位置
SEEK_END - 结束位置。
示例:

```
int handle=FileOpen("filename.csv", FILE_CSV|FILE_READ|FILE_WRITE, ';');
if(handle>0)
  {
   FileSeek(handle, 0, SEEK_END);
   //----在文件末端添加数据
   FileWrite(handle, data1, data2);
   FileClose(handle);
   handle=0;
  }
```

# FileSize 文件大小

int FileSize( int handle)
函数返回文件大小字节。获得详细的错误信息，查看 GetLastError() 函数。
参量:
handle - 用 FileOpen()函数返回的句柄。
示例:

```
int handle;
int size;
handle=FileOpen("my_table.dat", FILE_BIN|FILE_READ);
if(handle>0)
  {
   size=FileSize(handle);
   Print("my_table.dat 大小为 ", 大小 " bytes");
   FileClose(handle);
  }
```

# FileTell 文件指针的当前位置

int FileTell( int handle)
返回文件指针的当前位置。获得详细的错误信息，查看 GetLastError() 函数。
参量:
handle - 用 FileOpen()函数返回的句柄。
示例:

```
int handle;
int pos;
handle=FileOpen("my_table.dat", FILE_BIN|FILE_READ);
// 读取数据
pos=FileTell(handle);
Print("current position is ", pos);
```

# FileWrite 写入文件

int FileWrite( int handle, ...)
该函数的目的是便于书写的数据转换为 CSV 格式文件，自动插入限定符。在写入文件后，每行的尾端将会添加"\r\n" 符号。数字将会被转变成文本(查看 Print() 函数)。
如果生成错误，返回书写字或负值的计数。
获得详细的错误信息，查看 GetLastError() 函数。
参量:
handle - 用 FileOpen()函数返回的句柄。
... - 写入的数据，由逗号分离。 它可以是 63 个参量。当他们作为整数时，int 数据和双重类型自动地被转换成串，但不自动转换颜色、日期-时间和 bool typesare 和写出的文件。数组无法作为参量
示例:

```
int handle;
datetime orderOpen=OrderOpenTime();
handle=FileOpen("filename", FILE_CSV|FILE_WRITE, ';');
if(handle>0)
  {
   FileWrite(handle, Close[0], Open[0], High[0], Low[0], TimeToStr(orderOpen));
   FileClose(handle);
  }
```

# FileWriteArray 一个二进制文件写入数组

int FileWriteArray( int handle, object array[], int start, int count)
函数给一个二进制文件写入数组。一些 int、bool、日期-时间和颜色类型书面元素作为 4 字节整数。 一些双重类型书面元素，作为 8 字节浮动小数点数字。一些字串符类型将自动地在每串末尾添加符号 "\r\n" 。
如果错误生成，返回书写字或负值的计数。 获得详细的错误信息，查看 GetLastError()函数。
参量:
handle - 用 FileOpen()函数返回的句柄。
array[] - 写数组。
start - 在数组内(第一个书面元素数字)开始索引。
count - 书写字的计数。
示例:

```
int handle;
double BarOpenValues[10];
//  复制前十个柱到数组
for(int i=0;i<10; i++)
   BarOpenValues[i]=Open[i];
// 写入数组到文件
handle=FileOpen("mydata.dat", FILE_BIN|FILE_WRITE);
if(handle>0)
```

```
  {
   FileWriteArray(handle, BarOpenValues, 3, 7); // 写入最后 7 个元素
   FileClose(handle);
  }
```

# FileWriteDouble 一个二进制文件以浮动小数点写入双重值

int FileWriteDouble( int handle, double value, void size)
函数给一个二进制文件以浮动小数点写入双重值。 如果格式指定为 FLOAT_VALUE，值将作为 4 字节浮动小数点数字写入 (的浮游物类型)。否则，它以 8 字节浮动小数点格式将被写入(双重类型)。
如果错误生成，返回实际上书面字节数或负值。
获得详细的错误信息，查看 GetLastError() 函数。
参量:
handle - 用 FileOpen()函数返回的句柄。
value - 双精度值。
size - 选择格式。可以是以下的任意值:
DOUBLE_VALUE (8 字节，默认值)
FLOAT_VALUE (4 字节)。
示例:

```
int handle;
double var1=0.345;
handle=FileOpen("mydata.dat", FILE_BIN|FILE_WRITE);
if(handle<1)
  {
   Print("不能打开错误文件-",GetLastError());
   return(0);
  }
FileWriteDouble(h1, var1, DOUBLE_VALUE);
//...
FileClose(handle);
```

# FileWriteInteger 一个二进制文件写入整数值

int FileWriteInteger( int handle, int value, void size)
函数给一个二进制文件写入整数值。 如果大小是 SHORT_VALUE，值将作为 2 字节整数(短的类型)被写入。 如果大小是 CHAR_VALUE，值将作为 1 字节整数(炭灰类型)写入，并且，如果大小是 LONG_VALUE， 值将作为 4 字节整数(长的 int 类型)写入。
如果错误生成，返回实际上书面字节数或负值。
获得详细的错误信息，查看 GetLastError() 函数。
参量:
handle - 用 FileOpen()函数返回的句柄。
value - 写入值

size - 选择格式。可以是以下任意值:
CHAR_VALUE (1 字节),
SHORT_VALUE (2 字节),
LONG_VALUE (4 字节，默认值)。
示例:

```
int handle;
int value=10;
handle=FileOpen("filename.dat", FILE_BIN|FILE_WRITE);
if(handle<1)
  {
   Print("不能打开错误文件-",GetLastError());
   return(0);
  }
FileWriteInteger(handle, value, SHORT_VALUE);
//...
FileClose(handle);
```

# FileWriteString 当前文件位置函数写入一个二进制文件字串符

int FileWriteString( int handle, string value, int length)
从当前文件位置函数写入一个二进制文件字串符。
如果错误生成，返回实际上书面字节数或负值。
获得详细的错误信息,查看 GetLastError() 函数。
参量:
handle - 用 FileOpen()函数返回的句柄。
value - 写入字串符。
length - 写的字串符的长度。如果字串符长度超出被测量的值，它将被削减。 如果它较短，它将由二进制 0s 延伸由特定长度决定。
示例:

```
int handle;
string str="some string";
handle=FileOpen("filename.bin", FILE_BIN|FILE_WRITE);
  if(handle<1)
  {
   Print("不能打开错误文件-",GetLastError());
   return(0);
  }
FileWriteString(h1, str, 8);
FileClose(handle);
```

# Global variables 全局变量

和整体变量一起使用的一组函数。
客户端整体变量不应该与 MQL4 程序中的变量混合。
最后访问的整体变量可以在客户端内保存 4 个星期，随后将被自动删除。 对于整体变量的范文不仅仅是新值的设定，同样可以对整体变量进行读取。
在客户端从 MQL4 程序开启的客户端整体变量非常相似。

## GlobalVariableCheck

bool GlobalVariableCheck( string name)
如果整体变量存在，返回 TRUE.否则，返回 FALSE。要获得详细的错误信息，查看 GetLastError() 函数。
参量:
name - 整体变量名称。
示例:

```
// 检查先前使用变量
if(!GlobalVariableCheck("g1"))
   GlobalVariableSet("g1",1);
```

## GlobalVariableDel

bool GlobalVariableDel( string name)
删除整体变量。 如果函数成功，返回值将是真实的，否则，它将是错误的。 要获得详细的错误信息，查看 GetLastError() 函数。
参量:
name - 整体变量名称。
示例:

```
// 删除名称为 "gvar_1"的整体变量("gvar_1");
```

## GlobalVariableGet

double GlobalVariableGet( string name)
如果错误生成，返回值为现有整体变量或 0。获得详细的错误信息，查看 GetLastError()函数。
参量:
name - 整体变量名称。
示例:

```
double v1=GlobalVariableGet("g1");
//---- 检查函数调用结果
if(GetLastError()!=0) return(false);
```

```
   //---- 继续程序
```

# GlobalVariableName

string GlobalVariableName( int index)
由函数索引在整体变量名单返回一个整体变量的名字。 获得详细的错误信息，查看 GetLastError()函数。
参量:
index - 索引在整体变量名单。 它必须超出或相等于 0 或少于 GlobalVariablesTotal()。
示例:

```
   int       var_total=GlobalVariablesTotal();
   string name;
   for(int i=0;i<var_total;i++)
      {
       name=GlobalVariableName(i);
       Print(i,": 整体变量名称 - ",name);
      }
```

# GlobalVariableSet

datetime GlobalVariableSet( string name, double value)
设置整体变量的新的价格值。如果它不存在，系统创造一个新的整体变量。如果函数成功，返回值将是最后存取时间。 否则，返回值将是 0。获得详细的错误信息,查看 GetLastError() 函数。
参量:
name - 整体变量名称。
value - 新的数值。
示例:

```
   //---- 尝试设定新值
   if(GlobalVariableSet("BarsTotal",Bars)==0)
      return(false);
   //---- 继续程序
```

# GlobalVariableSetOnCondition

bool GlobalVariableSetOnCondition( string name, double value, double check_value)
如果当前值均等对第三参量 check_value，设置现有的整体变量的新值。 如果没有整体变量，函数将生成 错误 ERR_GLOBAL_VARIABLE_NOT_FOUND (4058) 并且返回 FALSE。当成功地执行，函数返回 TRUE，否则，它返回 FALSE。 获得详细的错误信息，查看 GetLastError() 函数。
如果整体变量的当前值与 check_value 不同，函数将返回 FALSE。
函数将为整体变量提供自动通道，这就是为什么在一个客户终端内几个智能交易可以同时运

行的原因。
参量:
name - 整体变量名称。
value - 新值。
check_value - 值与当前整体变量值比较。
示例:

```
int init()
  {
   //---- 创建整体变量
   GlobalVariableSet("DATAFILE_SEM",0);
   //...
  }
  int start()
  {
   //---- 尝试锁住源代码
   while(!IsStopped())
     {
      //---- 锁住
      if(GlobalVariableSetOnCondition("DATAFILE_SEM",1,0)==true)    break;
      //---- 可以删除变量吗?
      if(GetLastError()==ERR_GLOBAL_VARIABLE_NOT_FOUND) return(0);
      //---- 睡眠状态
      Sleep(500);
     }
   //---- 源代码被锁
   // ...做同样工作
   //----未锁源代码
   GlobalVariableSet("DATAFILE_SEM",0);
  }
```

# GlobalVariablesDeleteAll

int GlobalVariablesDeleteAll( void prefix_name)
删除整体变量。 如果命名前缀没有指定，所有整体变量将被删除。 否则，仅有那些可变物将被删除。从指定的前缀开始。 函数返回被删除的可变物计数。
参量:
prefix_name - 将被删除的整体变量的命名前缀。
示例:

```
Print(GlobalVariablesDeleteAll("test_")," 整体删除测试");
```

# GlobalVariablesTotal

int GlobalVariablesTotal( )

函数返回整体变量总值。
示例:

```
Print(GlobalVariablesTotal(),"探测整体变量");
```

# Math & Trig 数学和三角函数

一组数学和三角设置函数。

## MathAbs

double MathAbs( double value)
返回绝对值(模数)的指定的数值。
参量:
value - 数字值.
示例:

```
double dx=-3.141593, dy;
// calc MathAbs
dy=MathAbs(dx);
Print("The absolute value of ",dx," is ",dy);
// 输入数据: -3.141593 的绝对值为 3.141593
```

## MathArccos

double MathArccos( double x)
MathArccos 函数在范围 0 之内返回 x 反余弦到 π π (在弧度上)。如果 x 少于-1 是或超出 1，MathArccos 返回 FALSE。
参量:
x - 在-1 和 1 范围的值反余弦将被计算。
示例:

```
double x=0.32696, y;
y=asin(x);
Print("正弦",x," = ",y);
y=acos(x);
Print("余弦 ",x," = ",y);
//输入数据: 正弦 0.326960=0.333085
//输入数据:余弦 0.326960=1.237711
```

## MathArcsin

double MathArcsin( double x)
在 -π/2 到 π/2 范围内函数 MathArcsin 正弦 x 。如果 x 小于 -1 或超过 1, 正弦返回 NaN 。
参量:
x - 计算正弦的值。

示例:

```
double x=0.32696, y;
y=MathArcsin(x);
Print("正弦",x," = ",y);
y=acos(x);
Print("余弦  ",x," = ",y);
//输入数据:正弦  0.326960=0.333085
//输入数据:  余弦  0.326960=1.237711
```

# MathArctan

double MathArctan( double x)

函数 MathArctan 返回 x 的正切线值。 如果 x 为 0, MathArctan 返回 0。 MathArctan 返回值必须在- π /2 to π /2 弧度范围内。

参量:

x - 表示正切线的数字。

示例:

```
double x=-862.42, y;
y=MathArctan(x);
Print("正切线  ",x," is ",y);
//输入数据:正切线  -862.42 is -1.5696
```

# MathCeil

double MathCeil( double x)

MathCeil 函数返回一个最小超过或等于 x 的整数值。

参量:

x - 数值。

示例:

```
double y;
y=MathCeil(2.8);
Print("上限  2.8 is ",y);
y=MathCeil(-2.8);
Print("上限  -2.8 is ",y);
/*输入数据:
   2.8  的上限为 3
    -2.8  的上限为-2*/
```

# MathCos

double MathCos( double value)

返回指定的余弦角。

参量:
value - 角度测量。
示例:

```
double pi=3.1415926535;
double x, y;
x=pi/2;
y=MathSin(x);
Print("正弦(",x,") = ",y);
y=MathCos(x);
Print("余弦(",x,") = ",y);
//输入数据: 正弦(1.5708)=1
//          余弦(1.5708)=0
```

# MathExp

double MathExp( double d)
返回 e 的值升级到 d 的乘方。在溢出的情况下，函数返回 INF (无限定)，并且在底线返回 0。
参量:
d - 数字指定乘方。
示例:

```
double x=2.302585093,y;
y=MathExp(x);
Print("MathExp(",x,") = ",y);
//输入数据: MathExp(2.3026)=10
```

# MathFloor

double MathFloor( double x)
MathFloor 函数返回一个最大小于或等于 x 的整数值。
参量:
x - 数值。
示例:

```
double y;
y=MathFloor(2.8);
Print("下限  2.8 is ",y);
y=MathFloor(-2.8);
Print("下限  -2.8 is ",y);
/*输入数据:
   下限 2.8  为  2
  下限  -2.8 为-3*/
```

# MathLog

double MathLog( double x)
如果成功，MathLog 函数返回 x 的自然数。如果 x 是负值，这些函数返回 NaN (不确定值)。如果 x 是 0， 他们返回 INF (无限定)。
参量:
x - 发现的自然数值。
示例:

```
double x=9000.0,y;
y=MathLog(x);
Print("MathLog(",x,") = ", y);
//输入数据: MathLog(9000)=9.10498
```

# MathMax

double MathMax( double value1, double value2)
返回两个数字值的最大值。
参量:
value1 - 第一个数字值。
value2 - 第二个数字值。
示例:

```
double result=MathMax(1.08,Bid);
```

# MathMin

double MathMin( double value1, double value2)
返回两个数字值的最小值。
参量:
value1 - 第一个数字值。
value2 - 第二个数字值。
示例:

```
double result=MathMin(1.08,Ask);
```

# MathMod

double MathMod( double value, double value2)
此函数返回两位数除法的保留浮点。
MathMod 函数计算 x / y 的保留浮点 f ，这样 x = i * y + f ，i 是整数， f 与 x 是一样的标志， 并且 f 的绝对值小于 y 的绝对值。
参量:
value - 被除值。

value2 - 除值。
示例:

```
double x=-10.0,y=3.0,z;
z=MathMod(x,y);
Print("保留数 ",x," / ",y," 为 ",z);
//输入数据: -10 / 3 的保留数为 -1
```

# MathPow

double MathPow( double base, double exponent)
返回上升的基数指定的乘方(方次数值)。
参量:
base - 基数。
exponent - 方次数值。
示例:

```
double x=2.0,y=3.0,z;
z=MathPow(x,y);
Printf(x," 的",y,"次乘方为", z);
//输入数据: 2 的 3 次乘方为 8
```

# MathRand

int MathRand( )
在 0 到 32767 的范围内 MathRand 函数返回一个随机整数。在调用 MathRand 之前，需要使用 MathSrand 函数找寻随机整数。
示例:

```
MathSrand(TimeLocal());
// 显示 10 个数字。
for(int i=0;i<10;i++ )
   Print("随机值 ", MathRand());
```

# MathRound

double MathRound( double value)
返回最近的四舍五入整数值。
参量:
value - 四舍五入值。
示例:

```
double y=MathRound(2.8);
Print("2.8 的四舍五入值为 ",y);
y=MathRound(2.4);
Print(" -2.4 的四舍五入值为 ",y);
```

```
//输入数据:  2.8 的四舍五入值为 3
//           -2.4 的四舍五入值为 -2
```

# MathSin

double MathSin( double value)
返回指定角的正弦。
参量:
value - 弧度角测量。
示例:

```
double pi=3.1415926535;
double x, y;
x=pi/2;
y=MathSin(x);
Print("MathSin(",x,") = ",y);
y=MathCos(x);
Print("MathCos(",x,") = ",y);
//输入数据: MathSin(1.5708)=1
//          MathCos(1.5708)=0
```

# MathSqrt

double MathSqrt( double x)
MathSqrt 函数返回 x 的平方根。如果 x 为负值，MathSqrt 返回不确定值(与 NaN 相同)。
参量:
x - 否定数值。
示例:

```
double question=45.35, answer;
answer=MathSqrt(question);
if(question<0)
   Print("错误: MathSqrt 返回",答案," 答案");
else
   Print("",问题,"的平方根为 ", 答案);
//输入数据: 45.35 的平方根为 6.73
```

# MathSrand

void MathSrand( int seed)
MathSrand() 函数设置一系列随机整数的开始点。重新初始化生成，使用 1 作为自变数。找到的其他数值设置一个随机开始点。 MathRand 检测出生成的随机数字。调用 MathRand 之前， 任何 MathSrand 的生成调用需要按照找寻通过 1 的顺序调用 MathSrand 。
参量:

seed - 找寻生成的随机数字。
示例:

```
MathSrand(TimeLocal());
// 显示 10 数字。
for(int i=0;i<10;i++ )
   Print("随机值 ", MathRand());
```

# MathTan

double MathTan( double x)
MathTan 返回 x 的正切线。如果 x 大于等于 263 或者小于等于 -263，结果错误丢失，函数返回不确定值(与 NaN 相同)。
参量:
x - 弧度角
示例:

```
double pi=3.1415926535;
double x,y;
x=MathTan(pi/4);
Print("MathTan(",pi/4," = ",x);
//输入数据: MathTan(0.7856)=1
```

# Object functions 目标函数

对于当前图表有关的图表物件的一组函数。

## ObjectCreate 建立目标

bool ObjectCreate( string name, int type, int window, datetime time1, double price1, void time2, void price2, void time3, void price3)
物件创建的指定名称、类型和最初坐标的指定窗口。计数坐标与物件的关联可以是从 1 到 3 物件类型。 如果函数成功，返回值将是 TRUE，否则，它将是 FALSE。获得详细的错误信息，查看 GetLastError()函数。 OBJ_LABEL 类型的物件忽略坐标。 使用 ObjectSet() 设定 OBJPROP_XDISTANCE 和 OBJPROP_YDISTANCE 属性。
注解: 子窗口图表(如果子窗口带有指标)编号从 1 开始。主窗口的存在的索引为零。
必须通过的坐标： 时间和价格。 例如， OBJ_VLINE 只物件需要时间，但必须通过价格(任何值)。
参量:
name - 物件唯一名称。
type - 物件类型。它可以是物件类型列举的任意值。
window - 件窗口将增加的索引。窗口索引必须多于或等于 0 并且小于 WindowsTotal()。
time1 - 第一点的时间部分。
price1 - 第一点的值部分。
time2 - 第二点的时间部分。
price2 - 第二点的值部分。
time3 - 第三点的时间部分。
price3 - 第三点的值部分。
示例:

```
// 新文本物件
if(!ObjectCreate("text_object", OBJ_TEXT, 0, D'2004.02.20 12:30', 1.0045))
  {
   Print("错误:不能创建文本! 代码 #",GetLastError());
   return(0);
  }
// 新文本标签
if(!ObjectCreate("label_object", OBJ_LABEL, 0, 0, 0))
  {
   Print("错误:不能创建文本! 代码 #",GetLastError());
   return(0);
  }
ObjectSet("label_object", OBJPROP_XDISTANCE, 200);
ObjectSet("label_object", OBJPROP_YDISTANCE, 100);
```

# ObjectDelete 删除目标

bool ObjectDelete( string name)
删除物件已有的指定名称。 如果函数成功，返回值将是 TRUE，否则，它将是 FALSE。
获得详细的错误信息，查看 GetLastError() 函数。
参量:
name - 被删除的物件名称。
示例:
ObjectDelete("text_object");

# ObjectDescription 目标描述

string ObjectDescription( string name)
返回物件描述。 对于 OBJ_TEXT 和 OBJ_LABEL 类型物件，这些物件文本将返回。
获得详细的错误信息，查看 GetLastError() 函数。
参见 ObjectSetText() 函数。
参量:
name - 物件名称。
示例:

```
// 对于文件写下图表物件
int       handle, total;
string obj_name,fname;
// 文件名称
fname="objlist_"+Symbol();
handle=FileOpen(fname,FILE_CSV|FILE_WRITE);
if(handle!=false)
   {
    total=ObjectsTotal();
    for(int i=-;i<total;i++)
       {
        obj_name=ObjectName(i);
        FileWrite(handle,"Object "+obj_name+" description= "+ObjectDescription(obj_name));
       }
    FileClose(handle);
   }
```

# ObjectFind 查找目标

int ObjectFind( string name)
查找指定的物件名称。窗口的索引包含所找到的物件。如果它失败，返回值将是-1 。获得详细的错误信息，查看 GetLastError() 函数。子窗口图表(如果子窗口带有指标)编号从 1 开始。主窗口的索引为零。

参量:
name - 查找的物件名称。
示例:
if(ObjectFind("line_object2")!=win_idx) return(0);

# ObjectGet 目标属性

double ObjectGet( string name, int index)
函数返回指定物件的属性。检查错误，查看 GetLastError() 函数。
参见 ObjectSet()函数。
参量:
name - 物件名称。
index - 物件属性索引。它可以是物件属性列举值的任意。
示例:
color oldColor=ObjectGet("hline12", OBJPROP_COLOR);

# ObjectGetFiboDescription 斐波纳契描述

string ObjectGetFiboDescription( string name, int index)
函数返回对斐波纳契物件的平实描述。相当数量斐波纳契水平取决于物件类型。 最大斐波纳契水平是 32。
获得详细的错误信息,查看 GetLastError() 函数。
参见 ObjectSetFiboDescription() 函数。
参量:
name - 斐波纳契物件名称。
index - 斐波纳契索引水平(0-31)。
示例:

```
#include <stdlib.mqh>
  ...
  string text;
  for(int i=0;i<32;i++)
     {
      text=ObjectGetFiboDescription(MyObjectName,i);
      //---- 检查物件少于 32 水平线
      if(GetLastError()!=ERR_NO_ERROR) break;
      Print(MyObjectName,"水平: ",i," description: ",text);
     }
```

# ObjectGetShiftByValue

int ObjectGetShiftByValue( string name, double value)
函数计算并返回索引柱（移动当前相关的柱）给出的值。 索引柱由第一和第二坐标应用线

性方程计算。适用于趋势线和相似的物件。获得详细的错误信息，查看 GetLastError() 函数。
参见 ObjectGetValueByShift() 函数。
参量:
name - 物件名称。
value - 价格值。
示例:
```
int shift=ObjectGetShiftByValue("MyTrendLine#123", 1.34);
```

# ObjectGetValueByShift

double ObjectGetValueByShift( string name, int shift)
函数计算并返回指定柱的值(转移当前相关的柱)。索引柱由第一和第二坐标应用线性方程计算。 适用于趋势线和相似的物件。获得详细的错误信息，查看 GetLastError() 函数。
参见 ObjectGetShiftByValue() 函数。
参量:
name - 物件名称。
shift - 柱索引。
示例:
```
double price=ObjectGetValueByShift("MyTrendLine#123", 11);
```

# ObjectMove 移动目标

bool ObjectMove( string name, int point, datetime time1, double price1)
函数在图移动一个物件座标。 物件可能根据他们的 类型 有一个到三个座标。 如果函数成功 ，返回值将是 TRUE， 否则，它将是 FALSE。获得详细的错误信息，查看 GetLastError() 函数。物件坐标的开始数字必须是 0。
参量:
name - 物件名称。
point - 坐标索引(0-2)。
time1 - 新时间值。
price1 - 新值。
示例:
```
ObjectMove("MyTrend", 1, D'2005.02.25 12:30', 1.2345);
```

# ObjectName 目标名

string ObjectName( int index)
在物件列表中用它的索引函数返回物件名称。获得详细的错误信息，查看 GetLastError() 函数。
参量:
index - 在物件列表中的物件索引。物件索引必须超过或等于 0 并且小于 ObjectsTotal()。
示例:

```
int       obj_total=ObjectsTotal();
string name;
for(int i=0;i<obj_total;i++)
   {
    name=ObjectName(i);
    Print(i,"物件名称为 " + name);
   }
```

# ObjectsDeleteAll 删除所有目标

int ObjectsDeleteAll( void window, void type)
在图表的子窗口删除全部类型物件。函数返回删除物件数。获得详细的错误信息，查看 GetLastError() 函数。
注解: 子窗口图表(如果子窗口带有指标)编号从 1 开始。主窗口的存在的索引为零。如果窗口索引错误或值为-1，物件 会从现有的图表中删除。
如果类型 值等与-1 或这个参量是错误的，在子窗口的全部指定物件将被删除。
参量:
window - 选择参量。 物件的索引窗口将被删除。 必须超过或等于 -1 （EMPTY 为默认值）并且小于 WindowsTotal()。
type - 选择参量。被删除的物件类型。它可以是 任意列举值的物件类型或 EMPTY 常数删除所有物件类型。
示例:

```
ObjectsDeleteAll(2, OBJ_HLINE); // 从第二子窗口移除全部水平线。
ObjectsDeleteAll(2);                    // 从第二子窗口移除全部物件。
ObjectsDeleteAll();                     //从图表中移除全部物件。
```

# ObjectSet 改变目标属性

bool ObjectSet( string name, int index, double value)
改变指定物件属性的值。如果函数成功，返回值将是 TRUE。否则，它将是 FALSE. 获得详细的错误信息，查看 GetLastError() 函数。
参见 ObjectGet() 函数。
参量:
name - 物件名称。
index - 物件索引值。 它可以是列举的任意物件属性值。
value - 新的属性值。
示例:

```
// moving the first coord to the last bar time
ObjectSet("MyTrend", OBJPROP_TIME1, Time[0]);
// setting the second fibo level
ObjectSet("MyFibo", OBJPROP_FIRSTLEVEL+1, 1.234);
// setting object visibility. object will be shown only on 15 minute and 1 hour charts
ObjectSet("MyObject", OBJPROP_TIMEFRAMES, OBJ_PERIOD_M15 | OBJ_PERIOD_H1);
```

# ObjectSetFiboDescription 改变目标斐波纳契指标

bool ObjectSetFiboDescription( string name, int index, string text)
函数分配一个新的描述到斐波纳契物件的水平。 相当数量斐波纳契水平取决于物件类型。最大金额斐波纳契水平是 32。
获得详细的错误信息,查看 GetLastError() 函数。
参量:
name - 物件名称。
index - 斐波纳契索引水平(0-31)。
text - 新的水平描述
示例:
ObjectSetFiboDescription("MyFiboObject",2,"Second line");

# ObjectSetText 改变目标说明

bool ObjectSetText( string name, string text, int font_size, void font, void text_color)
改变物件描述。对于 OBJ_TEXT 和 OBJ_LABEL 物件的描述作为图表的文本显示。如果函数成功, 返回的值将是 TRUE。 否则, 它是 FALSE。获得详细的错误信息, 查看 GetLastError() 函数。
只有字体大小，字体名称和文本颜色参量使用为 font_size, font_name 和 text_color 物件。为其它类型 物件, 这些参量被忽略。
参见 ObjectDescription() 函数。
参量:
name - 物件名称。
text - 描述物件文本。
font_size - 字体大小点数。
font - 字体名称。
text_color - 文本颜色。
示例:
ObjectSetText("text_object", "Hello world!", 10, "Times New Roman", Green);

# ObjectsTotal 返回目标总量

int ObjectsTotal( void type)
在图表中返回指定物件类型总量。
参量:
type - 选择参量。将计数的物件类型。 它可以是 物件类型列举的任意值或 EMPTY 常数计算全部类型物件。
示例:

```
int       obj_total=ObjectsTotal();
string name;
for(int i=0;i<obj_total;i++)
```

```
{
  name=ObjectName(i);
  Print(i,"对于 #的物件名称",i," is " + name);
}
```

# ObjectType 返回目标类型

int ObjectType( string name)

函数返回 物件类型值。获得详细的错误信息，查看 GetLastError() 函数。

参量:

name - 物件名称。

示例:

```
if(ObjectType("line_object2")!=OBJ_HLINE) return(0);
```

# String functions 字符串函数

字串符类型数据的一组函数。

## StringConcatenate 字符串连接

string StringConcatenate( ...)
数据的字串符形式通过并且返回。 参量可以为任意类型。通过参量的总数不得超过 64 个字符。
作为应用到 Print(), Alert() 和 Comment()函数的参量按照同样规则传送。从函数参量返回获取的字符串作为连接结果。
当字串符连续使用(+)添加时，StringConcatenate() 运行较快并且会存储。
参量:
... - 所有价格值由逗号分开。 它可以是 64 个参量。
示例:

```
   string text;
   text=StringConcatenate("Account free margin is ", AccountFreeMargin(), "Current time is ", TimeToStr(TimeCurrent()));
// 文本="Account free margin is " + AccountFreeMargin() + "Current time is " + TimeToStr(TimeCurrent())
   Print(text);
```

## StringFind 字符串搜索

int StringFind( string text, string matched_text, void start)
搜索子字串符。如果未找到子字串符，从搜索子字串符开始返回字串符中的位置，或是 -1。
参量:
text - 被搜索的字符串。
matched_text - 需要搜索的字符串。
start - 搜索开始索引位置 。
示例:

```
   string text="快速的棕色小狗跨越过懒惰的狐狸";
   int index=StringFind(text, "小狗跨越", 0);
   if(index!=16)
      Print("oops!");
```

## StringGetChar 字符串指定位置代码

int StringGetChar( string text, int pos)
从字串符指定位置返回代码。

参量:
text - 字串符。
pos - 取字符的位置 。可以自 0 至 StringLen(text)-1。
示例:

```
int char_code=StringGetChar("abcdefgh", 3);
// 取出代码 'c' 是 99
```

# StringLen 字符串长度

int StringLen( string text)
在字串符中返回代码数。 Returns character count in a string.
参量:
text - 计算字符串长度。
示例:

```
string str="some text";
if(StringLen(str)<5) return(0);
```

# StringSetChar

string StringSetChar( string text, int pos, int value)
在指定位置返回带有改变代码的字串符复本。
参量:
text - 改变的字串符代码。
pos - 字串符种代码的位置。可以自 0 至 StringLen(text)。
value - 新取得 ASCII 代码。
示例:

```
string str="abcdefgh";
string str1=StringSetChar(str, 3, 'D');
// str1 is "abcDefgh"
```

# StringSubstr 提取子字符串

string StringSubstr( string text, int start, void length)
从给出的位置的文本字串符开端提取字串符。
如果可能此函数返回提取字串符的副本，否则返回空字串符。
参量:
text - 将被提取的字串符。
start - 字串符开始索引。可以是自 0 至 StringLen(text)-1。
length - 字串符提取的宽度。如果参量值超过或等于 0 或者参量没有指定，字串符将被提取。
示例:

```
string text="快速的棕色小狗跨越过懒惰的狐狸";
```

```
string substr=StringSubstr(text, 4, 5);
// 减去字串符是"快速"单词
```

# StringTrimLeft

string StringTrimLeft( string text)
在字串符左侧部分函数剪切空间和图表。如果可能函数返回一个剪切的复本。否则返回空字串符。
参量:
text - 左侧剪切的字串符。
示例:

```
string str1="    Hello world     ";
string str2=StringTrimLeft(str);
// 在剪切 str2 将是 "Hello World     "
```

# StringTrimRight

string StringTrimRight( string text)
在字串符右侧部分函数剪切空间和图表。如果可能函数返回一个剪切的复本。否则返回空字串符。
参量:
text - 右侧剪切的字串符。
示例:

```
string str1="    Hello world     ";
string str2=StringTrimRight(str);
// 在剪切 str2 之后将是 "    Hello World"
```

# Technical indicators 技术指标

标准和自定义指标的一组计算函数。
对于交易(或其他 MQL4 程序)接受其他指标的值，这个值不可能存在于图表之内。 这个请求的指标将在嗲用模件中被加载并计算。
不仅可以计算当前图表中的任何指标，同样可以计算任何有效的货币对/时间周期数据。如果请求数据(货币对名称/时间周期不同于当前图表)来自其他图表，这种情况可能使相应的图表不能在客户端内打开，并且需要从服务器上请求数据。这种情况下， 错误 ERR_HISTORY_WILL_UPDATED (4066 - 请求历史数据并刷新)将被放置于 last_error 变量中，并且可以重新请求(查看 ArrayCopySeries()范例)。

## iAC 比尔.威廉斯的加速器或减速箱振荡器

double iAC( string symbol, int timeframe, int shift)
计算比尔.威廉斯的加速器或减速箱振荡器。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前)。
示例:

```
double result=iAC(NULL, 0, 1);
```

## iAD 离散指标

double iAD( string symbol, int timeframe, int shift)
计算离散指标并且返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期.可以是时间周期列举任意值.0 表示当前图表的时间周期.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前)。
示例:

```
double result=iAD(NULL, 0, 1);
```

## iAlligator 比尔・威廉斯的鳄鱼指标

double iAlligator( string symbol, int timeframe, int jaw_period, int jaw_shift, int teeth_period, int teeth_shift, int lips_period, int lips_shift, int ma_method, int applied_price, int mode, int shift)
计算比尔・威廉斯的鳄鱼指标并且退回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.

timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
jaw_period - 平均周期(鳄鱼的下颌)的蓝线。
jaw_shift - 蓝线转移相对图。
teeth_period - 平均周期(鳄鱼的牙)的红线。
teeth_shift - 红线转移相对图。
lips_period - 平均周期(鳄鱼的嘴唇)的绿线。
lips_shift - 绿线转移相对图。
ma_method - MA 方法。 它可以是其中任意个滑动平均法.
applied_price - 应用的价格。 它可以是 应用价格列举 的任意值.
mode - 数据来源，显示线的标识符。 它可以是以下值中的任意:
MODE_GATORJAW - Gator 下颌(蓝色)平衡线路,
MODE_GATORTEETH - Gator 牙(红色)平衡线路，
MODE_GATORLIPS - Gator 嘴唇(绿色)平衡线路。
shift - 转移相对当前柱(期间的数字)应该采取数据从的地方。
示例:

```
double jaw_val=iAlligator(NULL, 0, 13, 8, 8, 5, 5, 3, MODE_SMMA, PRICE_MEDIAN, MODE_GATORJAW, 1);
```

# iADX 移动定向索引

double iADX( string symbol, int timeframe, int period, int applied_price, int mode, int shift)
计算移动定向索引并且退回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
period - 计算平均周期。
applied_price - 应用的价格。它可以是应用价格列举的任意值.
mode - 指标索引行。 它可以是指标索引列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前) .
示例:

```
if(iADX(NULL,0,14,PRICE_HIGH,MODE_MAIN,0)>iADX(NULL,0,14,PRICE_HIGH,MODE_PLUSDI,0))
return(0);
```

# iATR 平均真实范围

double iATR( string symbol, int timeframe, int period, int shift)
计算平均真实范围的指标并且返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
period - 计算平均周期。
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前)。
示例:

if(iATR(NULL,0,12,0)>iATR(NULL,0,20,0)) return(0);

# iAO 比尔.威廉斯的振荡器

double iAO( string symbol, int timeframe, int shift)
计算比尔.威廉斯的振荡器并且退回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前)。
示例:
double val=iAO(NULL, 0, 2);

# iBearsPower 熊功率指标

double iBearsPower( string symbol, int timeframe, int period, int applied_price, int shift)
计算熊功率指标并且返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
period - 计算平均周期。
applied_price - 应用的价格。 它可以是应用价格列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前) .
示例:
double val=iBearsPower(NULL, 0, 13,PRICE_CLOSE,0);

# iBands 保力加通道技术指标

double iBands( string symbol, int timeframe, int period, int deviation, int bands_shift, int applied_price, int mode, int shift)
计算保力加通道技术指标并返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
period - 计算平均周期的主线。
deviation - 与主线的偏差。
bands_shift - 指标相对图转移。
applied_price - 应用的价格。 它可以是应用价格列举的任意值.
mode - 显示索引行。 它可以是指标线识别符列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:
if(iBands(NULL,0,20,2,0,PRICE_LOW,MODE_LOWER,0)>Low[0]) return(0);

# iBandsOnArray 保力加通道指标

double iBandsOnArray( double array[], int total, int period, int deviation, int bands_shift, int mode, int shift)
计算保力加通道指标在不同数组上的数据存储。不同于 iBands (...)， iBandsOnArray 函数，不由货币对名字，时间周期，应用的价格采取数据 。 必须提前准备价格数据。从左到右计算指标。 要对数组元素访问至系列列阵(即，从右到左)，你必须使用 ArraySetAsSeries 函数.
参量:
array[] - 数据数组。
total - 将计数的项目的数量。 0 意味整体列阵。
period - 计算主线的平均周期。
deviation - 与主线的偏差。
bands_shift - 指标相对图转移。
mode - 显示索引行。 它可以是指标线识别符列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:
if(iBands(ExtBuffer,total,2,0,MODE_LOWER,0)>Low[0]) return(0);

# iBullsPower 牛市指标

double iBullsPower( string symbol, int timeframe, int period, int applied_price, int shift)
计算牛市指标并返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
period - 计算平均周期。
applied_price - 应用的价格。 它可以是应用价格列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前) .
示例:

```
double val=iBullsPower(NULL, 0, 13,PRICE_CLOSE,0);
```

# iCCI 商品通道索引指标

double iCCI( string symbol, int timeframe, int period, int applied_price, int shift)
计算商品通道索引指标并且返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
period - 计算平均周期.
applied_price - 应用的价格。 它可以是应用价格列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前) .
示例:

if(iCCI(NULL,0,12,PRICE_TYPICAL,0)>iCCI(NULL,0,20,PRICE_TYPICAL,0)) return(0);

# iCCIOnArray 商品通道索引指标

double iCCIOnArray( double array[], int total, int period, int shift)
计算在不同数组存储的商品通道索引指标。不同于 iCCI (...)， iCCIOnArray 函数， 不由标志名字，时间周期，应用的价格采取数据。 必须提前准备价格数据。指标从左到右被计算。 要对数组元素至于系列列阵访问(即，从右到左)，你必须使用 ArraySetAsSeries 函数.
参量:
array[] - 数据数组.
total - 计算项目数字.
period - 计算平均周期.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前）.
示例:
if(iCCIOnArray(ExtBuffer,total,12,0)>iCCI(NULL,0,20,PRICE_TYPICAL, 0)) return(0);

# iCustom 指定的客户指标

double iCustom( string symbol, int timeframe, string name, ..., int mode, int shift)
计算指定的客户指标并且退回它的值。 必须在 terminal_directory\experts\indicators 目录内编写客户指标(*.EX4 文件)。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
name - 客户指标完成程序名称.
... - 参量设置(如果需要)。通过的参量和他们的顺序必须与 desclaration 命令和客户指标的外部可变物的种类对应。
mode - 索引行。 从 0 到 7 并且必须对应以其中一个使用的索引的 SetIndexBuffer 函数.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:
double val=iCustom(NULL, 0, "示例 Ind",13,1,0);

# iDeMarker

double iDeMarker( string symbol, int timeframe, int period, int shift)
计算 DeMarker 指标并返回它的值.
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
period - 计算平均周期。
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前) .

示例:

```
double val=iDeMarker(NULL, 0, 13, 1);
```

# iEnvelopes 包络指标

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

计算包络指标并返回它的值。

参量:

symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.

timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.

ma_period - 主线的平均周期计算.

ma_method - MA 方法。 它可以是其中任意 滑动平均值列举 值.

ma_shift - MA 转移。 指标线垂直与图表的时间周期.

applied_price - 应用的价格。 它可以是应用价格列举的任意值.

deviation - 与主线的偏差。

mode - 指标行数组索引。它可以是 指标识别符列举的任意值.

shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).

示例:

```
double val=iEnvelopes(NULL, 0, 13,MODE_SMA,10,PRICE_CLOSE,0.2,MODE_UPPER,0);
```

# iEnvelopesOnArray 包络指标

double iEnvelopesOnArray( double array[], int total, int ma_period, int ma_method, int ma_shift, double deviation, int mode, int shift)

计算包络指标在不同数组上的数据存储。与不同 iEnvelopes (...)， iEnvelopesOnArray 函数不由标志名字， 时间周期，应用的价格采取数据。 必须提前准备价格数据。 指标从左到右被计算。 要对数组元素至于系列列阵(即，从右到左)访问，你必须使用 ArraySetAsSeries 函数.

参量:

array[] - 数据数组.

total - 计算项目数字.

ma_period - 主线的平均周期计算。

ma_method - MA 方法。 它可以是其中任意 滑动平均值列举 值.

ma_shift - MA 转移。 指标线垂直与图表的时间周期。

deviation - 与主线的偏差。

mode - 指标行数组索引。它可以是 指标识别符列举的任意值.

shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).

示例:

```
double val=iEnvelopesOnArray(ExtBuffer, 0, 13, MODE_SMA, 0.2, MODE_UPPER,0 );
```

# iForce 强力索引指标

double iForce( string symbol, int timeframe, int period, int ma_method, int applied_price, int shift)
计算强力索引指标并返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
period - 计算平均周期。
ma_method - MA 方法。 它可以是其中任意 滑动平均值列举 值.
applied_price - 应用的价格。 它可以是应用价格列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:

```
double val=iForce(NULL, 0, 13,MODE_SMA,PRICE_CLOSE,0);
```

# iFractals 分形索引指标

double iFractals( string symbol, int timeframe, int mode, int shift)
计算分形索引指标并返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
mode - 指标行数组索引。它可以是 指标识别符列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:

```
double val=iFractals(NULL, 0, MODE_UPPER, 3);
```

# iGator 随机震荡指标

double iGator( string symbol, int timeframe, int jaw_period, int jaw_shift, int teeth_period, int teeth_shift, int lips_period, int lips_shift, int ma_method, int applied_price, int mode, int shift)
计算随机震荡指标。 震荡指标在鳄鱼 红色和蓝线(上部直方图)和那之间的区别在于红色和绿线(更低的直方图)之间。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
jaw_period - 平均周期(鳄鱼的下颌)的蓝线.
jaw_shift - 蓝线转移相对图.
teeth_period - 平均周期(鳄鱼的牙)的红线.
teeth_shift - 红线转移相对图.
lips_period - 平均周期(鳄鱼的嘴唇)的绿线.
lips_shift - 绿线转移相对图.

ma_method - MA 方法。 它可以是其中任意 滑动平均值列举 值.
applied_price - 应用的价格。 它可以是应用价格列举的任意值.
mode - 指标行数组索引。它可以是 指标识别符列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:

```
double jaw_val=iGator(NULL, 0, 13, 8, 8, 5, 5, 3, MODE_SMMA, PRICE_MEDIAN, MODE_UPPER,
1);
```

# iIchimoku

double iIchimoku( string symbol, int timeframe, int tenkan_sen, int kijun_sen, int senkou_span_b, int mode, int shift)
计算 Ichimoku Kinko Hyo 并且返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
tenkan_sen - Tenkan Sen 平均周期.
kijun_sen - Kijun Sen 平均周期.
senkou_span_b - Senkou SpanB 平均周期.
mode - 数据源代码。它可以是 Ichimoku Kinko Hyo 列举模式的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:

```
double tenkan_sen=iIchimoku(NULL, 0, 9, 26, 52, MODE_TENKANSEN, 1);
```

# iBWMFI 比尔.威廉斯市场斐波纳契指标

double iBWMFI( string symbol, int timeframe, int shift)
计算比尔.威廉斯市场斐波纳契指标并且返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:

```
double val=iBWMFI(NULL, 0, 0);
```

# iMomentum 动量索引指标

double iMomentum( string symbol, int timeframe, int period, int applied_price, int shift)
计算动量索引指标并返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.

period - 价格改变的周期计算.
applied_price - 应用的价格。 它可以是应用价格列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:
if(iMomentum(NULL,0,12,PRICE_CLOSE,0)>iMomentum(NULL,0,20,PRICE_CLOSE,0)) return(0);

# iMomentumOnArray

double iMomentumOnArray( double array[], int total, int period, int shift)
计算动量指标在不同数组上的数据存储。与不同 iMomentum(...), the iMomentumOnArray 函数不由标志名字， 时间周期，应用的价格采取数据。 必须提前准备价格数据。 指标从左到右被计算。 要对数组元素至于系列列阵(即，从右到左)访问，你必须使用 ArraySetAsSeries 函数.
参量:
array[] - 数据数组.
total - 将计数的项目的数量.
period - 计算价格变化的周期.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:
if(iMomentumOnArray(mybuffer,100,12,0)>iMomentumOnArray(mubuffer,100,20,0)) return(0);

# iMFI 资金流量索引指标

double iMFI( string symbol, int timeframe, int period, int shift)
计算资金流量索引指标并返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
period - 指标的周期计算.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:
if(iMFI(NULL,0,14,0)>iMFI(NULL,0,14,1)) return(0);

# iMA 移动平均指标

double iMA( string symbol, int timeframe, int period, int ma_shift, int ma_method, int applied_price, int shift)
计算移动平均指标并返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
period - 平均周期计算。

ma_shift - MA 转移。 指标线垂直与图表的时间周期.
ma_method - MA 方法。 它可以是其中任意 滑动平均值列举 值.
applied_price - 应用的价格。 它可以是应用价格列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:

```
AlligatorJawsBuffer[i]=iMA(NULL,0,13,8,MODE_SMMA,PRICE_MEDIAN,i);
```

# iMAOnArray

double iMAOnArray( double array[], int total, int period, int ma_shift, int ma_method, int shift)
计算移动平均指标在不同数组上的数据存储。与不同 iMA(...), the iMAOnArray 函数不由标志名字， 时间周期，应用的价格采取数据。 必须提前准备价格数据。 指标从左到右被计算。要对数组元素至于系列列阵(即，从右到左)访问，你必须使用 ArraySetAsSeries 函数.
参量:
array[] - 数据数组.
total - 将计数的项目的数量.
period - 平均周期计算.
ma_shift - MA 移动.
ma_method - MA 方法。 它可以是其中任意 滑动平均值列举 值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:

```
    double macurrent=iMAOnArray(ExtBuffer,0,5,0,MODE_LWMA,0);
    double macurrentslow=iMAOnArray(ExtBuffer,0,10,0,MODE_LWMA,0);
    double maprev=iMAOnArray(ExtBuffer,0,5,0,MODE_LWMA,1);
    double maprevslow=iMAOnArray(ExtBuffer,0,10,0,MODE_LWMA,1);
    //----
    if(maprev<maprevslow && macurrent>=macurrentslow)
       Alert("穿过");
```

# iOsMA 移动振动平均震荡器指标

double iOsMA( string symbol, int timeframe, int fast_ema_period, int slow_ema_period, int signal_period, int applied_price, int shift)
计算移动振动平均震荡器指标并退回它的值。有时在一些系统中显示为 MACD 直方图。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
fast_ema_period - 对于快速移动平均值周期数的计算.
slow_ema_period - 对于缓慢移动平均值周期数计算.
signal_period - 对于信号移动平均值周期数计算.
applied_price - 应用的价格。 它可以是应用价格列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:

if(iOsMA(NULL,0,12,26,9,PRICE_OPEN,1)>iOsMA(NULL,0,12,26,9,PRICE_OPEN,0)) return(0);

# iMACD 移动平均数汇总/分离指标

double iMACD( string symbol, int timeframe, int fast_ema_period, int slow_ema_period, int signal_period, int applied_price, int mode, int shift)
计算移动平均数汇总/分离指标并退回它的值。在系统中， OsMA 称 MACD 直方图， 这个指标被作为二条线。 在客户终端，移动平均数汇总/分离被画作为直方图。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
fast_ema_period - 对于快速移动平均值周期数的计算.
slow_ema_period - 对于缓慢移动平均值周期数计算.
signal_period - 对于信号移动平均值周期数计算.
applied_price - 应用的价格。 它可以是应用价格列举的任意值.
mode - 指标行数组索引。它可以是 指标识别符列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:
if(iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_MAIN,0)>iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_SIGNAL,0)) return(0);

# iOBV 能量潮指标

double iOBV( string symbol, int timeframe, int applied_price, int shift)
计算能量潮指标并返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
applied_price - 应用的价格。 它可以是应用价格列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:

```
double val=iOBV(NULL, 0, PRICE_CLOSE, 1);
```

# iSAR 抛物线状止损和反转指标

double iSAR( string symbol, int timeframe, double step, double maximum, int shift)
计算抛物线状止损和反转指标并返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
step - 增值，通常 0.02.
maximum - 最大值，通常 0.2.

shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:
if(iSAR(NULL,0,0.02,0.2,0)>Close[0]) return(0);

# iRSI 相对强弱索引指标

double iRSI( string symbol, int timeframe, int period, int applied_price, int shift)
计算相对强弱索引指标并返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
period - 周期数字的计算.
applied_price - 应用的价格。 它可以是应用价格列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:
if(iRSI(NULL,0,14,PRICE_CLOSE,0)>iRSI(NULL,0,14,PRICE_CLOSE,1)) return(0);

# iRSIOnArray

double iRSIOnArray( double array[], int total, int period, int shift)
计算相对强弱索引指标在不同数组上的数据存储。与不同 iRSI(...), the iRSIOnArray 函数不由标志名字， 时间周期，应用的价格采取数据。 必须提前准备价格数据。 指标从左到右被计算。 要对数组元素至于系列列阵(即，从右到左)访问，你必须使用 ArraySetAsSeries 函数.
参量:
array[] - 数据数组.
total - 将计数的项目的数量.
period - 计算价格变化的周期.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:
if(iRSIOnArray(ExtBuffer,1000,14,0)>iRSI(NULL,0,14,PRICE_CLOSE,1)) return(0);

# iRVI 相对活力索引指标

double iRVI( string symbol, int timeframe, int period, int mode, int shift)
计算相对活力索引指标并返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
period - 周期数字的计算.
mode - 指标行数组索引。它可以是 指标识别符列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:

double val=iRVI(NULL, 0, 10,MODE_MAIN,0);

# iStdDev 标准偏差指标

double iStdDev( string symbol, int timeframe, int ma_period, int ma_shift, int ma_method, int applied_price, int shift)
计算标准偏差指标并返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
ma_period - MA 周期.
ma_shift - MA 移动.
ma_method - MA 方法。 它可以是其中任意 滑动平均值列举 值.
applied_price - 应用的价格。 它可以是应用价格列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:
double val=iStdDev(NULL,0,10,0,MODE_EMA,PRICE_CLOSE,0);

# iStdDevOnArray

double iStdDevOnArray( double array[], int total, int ma_period, int ma_shift, int ma_method, int shift)
计算标准离差索引指标在不同数组上的数据存储。与不同 iStdDev(...), the iStdDevOnArray 函数不由标志名字， 时间周期，应用的价格采取数据。 必须提前准备价格数据。 指标从左到右被计算。 要对数组元素至于系列列阵(即，从右到左)访问，你必须使用 ArraySetAsSeries 函数.
参量:
array[] - 数据数组.
total - 将计数的项目的数量.
ma_period - MA 周期.
ma_shift - MA 移动.
ma_method - MA 方法。 它可以是其中任意 滑动平均值列举 值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:
double val=iStdDevOnArray(ExtBuffer,100,10,0,MODE_EMA,0);

# iStochastic 随机震荡指标

double iStochastic( string symbol, int timeframe, int %Kperiod, int %Dperiod, int slowing, int method, int price_field, int mode, int shift)
计算随机震荡指标并返回它的值。
参量:

symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
%Kperiod - %K 周期线。.
%Dperiod - %D 周期线.
slowing - 滚动值.
method - MA 方法。 它可以是其中任意 滑动平均值列举 值.
price_field - 价格参量.可以是以下值: 0 - Low/High 或者 1 - Close/Close.
mode - 指标行数组索引。它可以是 指标识别符列举的任意值.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:

```
if(iStochastic(NULL,0,5,3,3,MODE_SMA,0,MODE_MAIN,0)>iStochastic(NULL,0,5,3,3,MODE_SMA,
0,MODE_SIGNAL,0))
    return(0);
```

# iWPR 威廉指标

double iWPR( string symbol, int timeframe, int period, int shift)
计算威廉指标并返回它的值。
参量:
symbol - 计算指标数据上的货币对名称. NULL 表示当前货币对.
timeframe - 时间周期。 可以时间周期列举任意值. 0 表示当前图表的时间周期.
period - 周期数字计算.
shift - 从显示缓冲采取的值的索引(转移相对当前柱特定相当数量期间前).
示例:

```
if(iWPR(NULL,0,14,0)>iWPR(NULL,0,14,1)) return(0);
```

# Timeseries access 时间序列图表数据

任何可见货币对/时间周期的价格数据的一组函数。
如果请求数据(货币对名称/时间周期不同于当前图表)来自其他图表，这种情况可能使相应的图表不能在客户端内打开，并且需要从服务器上请求数据。这种情况下， 错误 ERR_HISTORY_WILL_UPDATED (4066 - 请求历史数据并刷新)将被放置于 last_error 变量中，并且可以重新请求(查看 ArrayCopySeries()范例)。
在测试中，相同货币对但不同时间周期的价格价位被塑造（除成交量外）。其他货币对的价格数据不被塑造。这些情况下，在时间数组的 柱总数被塑造。

## iBars 柱的数量

int iBars( string symbol, int timeframe)
在指定的图表内返回柱的数量。
对于当前图表柱总量的信息在预定义的变量中命名为 Bars。
参量:
symbol - 需应用到计算指标的货币对数据 NULL 意味当前货币对名称。
timeframe - 时间周期。可以是时间周期列举的任意值。0 意味着当前图表的时间周期。
示例:

```
Print("在货币对'EUROUSD'带有 PERIOD_H1 柱数",iBars("EUROUSD",PERIOD_H1));
```

## iBarShift 开始时间的柱

int iBarShift( string symbol, int timeframe, datetime time, void exact)
搜索柱开始的时间。函数返回指定开始时间的柱。如果柱的指定开始时间是省缺值， 函数将返回-1 或 最近的柱 exact。
参量:
symbol - 需应用到计算指标的货币对数据 NULL 意味当前货币对名称。。
timeframe - 时间周期。可以是时间周期列举的任意值。0 意味着当前图表的时间周期。
time - 查找值 (柱的开始时间)。
exact - 未发现柱的返回模式。false - iBarShift 返回最近。 true - iBarShift 返回 -1。
示例:

```
datetime some_time=D'2004.03.21 12:00';
int      shift=iBarShift("EUROUSD",PERIOD_M1,some_time);
Print("带有打开时间平移柱 ",TimeToStr(some_time)," 是 ",shift);
```

## iClose

double iClose( string symbol, int timeframe, int shift)
对于带有时间周期和平移指定货币对的柱返回 关闭值。如果加载历史为空，函数返回 0。

对于当前图表，关于收盘价格的信息在预定义数组中命名为 Close[]。
参量:
symbol - 需应用到计算指标的货币对数据 NULL 意味当前货币对名称。
timeframe - 时间周期。可以是时间周期列举的任意值。0 意味着当前图表的时间周期。
shift - 从指标缓冲器上获取的索引值。
示例:

```
Print(" 对 于 USDCHF H1 当 前 柱 : ",iTime("USDCHF",PERIOD_H1,i),", ", iOpen("USDCHF",PERIOD_H1,i),", ",
                                    iHigh("USDCHF",PERIOD_H1,i),", ", iLow("USDCHF",PERIOD_H1,i),", ",
                                    iClose("USDCHF",PERIOD_H1,i),", ", iVolume("USDCHF",PERIOD_H1,i));
```

# iHigh

double iHigh( string symbol, int timeframe, int shift)
对于带有时间周期和平移指定货币对 的柱返回 高值。如果加载历史为空，函数返回 0。
对于当前图表，关于高价格的信息在预定义数组中命名为 High[].
参量:
symbol - 需应用到计算指标的货币对数据 NULL 意味当前货币对名称。
timeframe - 时间周期。可以是时间周期列举的任意值。0 意味着当前图表的时间周期。
shift - 从指标缓冲器上获取的索引值。
示例:

```
Print(" 对 于 USDCHF H1 当 前 柱 : ",iTime("USDCHF",PERIOD_H1,i),", ", iOpen("USDCHF",PERIOD_H1,i),", ",
                                    iHigh("USDCHF",PERIOD_H1,i),", ", iLow("USDCHF",PERIOD_H1,i),", ",
                                    iClose("USDCHF",PERIOD_H1,i),", ", iVolume("USDCHF",PERIOD_H1,i));
```

# iHighest

int iHighest( string symbol, int timeframe, int type, void count, void start)
根据类型返回最大值转移的一个具体数字。
参量:
symbol - 需应用到计算指标的货币对数据 NULL 意味当前货币对名称。。
timeframe - 时间周期。可以是时间周期列举的任意值。0 意味着当前图表的时间周期。
type - 系列数组的识别符。它可以是系列数据识别符列举 的任意值。
count - 周期数字。
start - 移动显示与当前相关的柱，采取数据。
示例:

```
double val;
// 在范围内 20 个连续柱计算最大值
```

```
// 在当前图表上从第 4 个至第 23 个的索引
val=High[iHighest(NULL,0,MODE_HIGH,20,4)];
```

# iLow

double iLow( string symbol, int timeframe, int shift)
对于带有时间周期和平移指定货币对 的柱返回 低值。如果加载历史为空，函数返回 0。
对于当前图表，关于低价格的信息在预定义数组中命名为 Low[].
参量:
symbol - 需应用到计算指标的货币对数据 NULL 意味当前货币对名称。。
timeframe - 时间周期。可以是时间周期列举的任意值。0 意味着当前图表的时间周期。
shift - 从指标缓冲器上获取的索引值。
示例:

```
Print(" 对 于 USDCHF H1 当 前 柱 : ",iTime("USDCHF",PERIOD_H1,i),", ",
iOpen("USDCHF",PERIOD_H1,i),", ",
                                    iHigh("USDCHF",PERIOD_H1,i),", ",
iLow("USDCHF",PERIOD_H1,i),", ",
                                    iClose("USDCHF",PERIOD_H1,i),", ",
iVolume("USDCHF",PERIOD_H1,i));
```

# iLowest

int iLowest( string symbol, int timeframe, int type, void count, void start)
根据类型返回最小值转移的一个具体数字。
参量:
symbol - 需应用到计算指标的货币对数据 NULL 意味当前货币对名称。。
timeframe - 时间周期。可以是时间周期列举的任意值。0 意味着当前图表的时间周期。
type - 系列数组的识别符。它可以是系列数据识别符列举 的任意值。
count - 时间周期。
start - 移动显示与当前相关的柱，采取数据。
示例:

```
// 在范围内计算连续 10 个柱的最低值
// 在当前图表从第 10 个到第 19 个的索引
double val=Low[iLowest(NULL,0,MODE_LOW,10,10)];
```

# iOpen

double iOpen( string symbol, int timeframe, int shift)
对于带有时间周期和平移指定货币对 的柱返回 开价格值。如果加载历史为空，函数返回 0。
对于当前图表，关于开价格的信息在预定义数组中命名为 Open[].
参量:
symbol - 需应用到计算指标的货币对数据 NULL 意味当前货币对名称。。

timeframe - 时间周期。可以是时间周期列举的任意值。0 意味着当前图表的时间周期。
shift - 从指标缓冲器上获取的价格值指数。
示例:

```
Print(" 对 于 USDCHF H1 当 前 柱 : ",iTime("USDCHF",PERIOD_H1,i),", ", iOpen("USDCHF",PERIOD_H1,i),", ",
                                    iHigh("USDCHF",PERIOD_H1,i),", ", iLow("USDCHF",PERIOD_H1,i),", ",
                                    iClose("USDCHF",PERIOD_H1,i),", ", iVolume("USDCHF",PERIOD_H1,i));
```

# iTime

datetime iTime( string symbol, int timeframe, int shift)
对于带有时间周期和平移指定货币对 的柱返回 时间值。如果加载历史为空，函数返回 0。
对于当前图表，关于时间的信息在预定义数组中命名 Time[].
参量:
symbol - 需应用到计算指标的货币对数据 NULL 意味当前货币对名称。。
timeframe - 时间周期。可以是时间周期列举的任意值。0 意味着当前图表的时间周期。
shift - 从指标缓冲器上获取的价格值指数。
示例:

```
Print(" 对 于 USDCHF H1 当 前 货 币 对 : ",iTime("USDCHF",PERIOD_H1,i),", ", iOpen("USDCHF",PERIOD_H1,i),", ",
                                    iHigh("USDCHF",PERIOD_H1,i),", ", iLow("USDCHF",PERIOD_H1,i),", ",
                                    iClose("USDCHF",PERIOD_H1,i),", ", iVolume("USDCHF",PERIOD_H1,i));
```

# iVolume

double iVolume( string symbol, int timeframe, int shift)
对于带有时间周期和平移指定货币对 的柱返回 替克成交量值。如果加载历史为空，函数返回 0。
对于当前图表，关于成交量的信息在预定义数组中命名 Volume[].
参量:
symbol - 需应用到计算指标的货币对数据 NULL 意味当前货币对名称。。
timeframe - 时间周期。可以是时间周期列举的任意值。0 意味着当前图表的时间周期。
shift - 从指标缓冲器上获取的价格值指数。
示例:

```
Print(" 对 于 USDCHF H1 的 当 前 柱 : ",iTime("USDCHF",PERIOD_H1,i),", ", iOpen("USDCHF",PERIOD_H1,i),", ",
                                    iHigh("USDCHF",PERIOD_H1,i),", ", iLow("USDCHF",PERIOD_H1,i),", ",
                                    iClose("USDCHF",PERIOD_H1,i),", ",
```

```
iVolume("USDCHF",PERIOD_H1,i));
```

# Trading functions 交易函数

交易管理的一组函数。
从自定义指标中不能调用 OrderSend(), OrderClose, OrderCloseBy, OrderDelete 和 OrderModify 交易函数。
交易函数应用于智能交易和脚本中。如果检验智能交易的“允许实事交易”属性，交易函数不能调用。
来自智能交易和脚本的交易在程序中只能有一个开启。这就是为什么如果交易业务忙，其他交易或脚本在此时不能调用的原因，由于错误 146 (ERR_TRADE_CONTEXT_BUSY)。 使用 IsTradeAllowed() 函数检测交易或没有交易。 弄清交易访问模式，可以使用改变 GlobalVariableSetOnCondition()函数整体变量值。

## Execution errors

任何交易业务(OrderSend(), OrderClose, OrderCloseBy, OrderDelete 和 OrderModify 函数)都会因为一些原因导致失败，并且返回负值票据数 或 FALSE。 您可以查看 GetLastError()函数得知错误的问题所在。 每一个错误必须以不同的方式加以处理。最常见的建议列举如下：
从交易服务器返回的错误代码
数据 应用 RefreshRates 函数重试。

| 常数 | 值 | 描述 |
|---|---|---|
| ERR_NO_ERROR | 0 | 交易业务成功。 |
| ERR_NO_RESULT | 1 | OrderModify 尝试去还原已经设定好的相同值。一个或多个值必须改变，然后修改尝试重复. |
| ERR_COMMON_ERROR | 2 | 常规错误。直到错误清晰为止，所有交易必须停止运行。如果需要客户端的交易系统必须重启。 |
| ERR_INVALID_TRADE_参量 | 3 | 无效参量， 例如，货币对错误,未知 交易业务，不存在票数等等。程序逻辑必须修改。 |
| ERR_SERVER_BUSY | 4 | 交易服务器忙。稍后请重新尝试。 |
| ERR_OLD_VERSION | 5 | 客户端的旧版本。客户端的最新版本必须初始化。 |
| ERR_NO_CONNECTION | 6 | 交易服务器没有联接。需要确认连接没有断开(例如，应用 IsConnected 函数) 在 5 秒之后重试。 |
| ERR_TOO_FREQUENT_REQUESTS | 8 | 请求过于频繁。过于频繁的请求必须减少， |

| | | |
|---|---|---|
| | | 程序逻辑需要改变。 |
| ERR_ACCOUNT_DISABLED | 64 | 账户被禁止。所有运行交易必须停止。 |
| ERR_INVALID_ACCOUNT | 65 | 账号无效。所有运行交易必须停止。 |
| ERR_TRADE_TIMEOUT | 128 | 交易超时。在重试前必须确认交易业务确实没有成功（存在未修改或未删除的定单） |
| ERR_INVALID_PRICE | 129 | 无效开价格或报价格。稍后必须刷新 数据 应用 /RefreshRates.html">RefreshRates 函数重试。 如果错误没有消失，尝试停止所有运行交易，改变程序逻辑。 |
| ERR_INVALID_STOPS | 130 | Stops 太近或是价格计算错误。 需要刷新 数据 /RefreshRates.html">RefreshRates 函数重试 如果错误没有消失，尝试停止所有运行交易，改变程序逻辑。 |
| ERR_INVALID_TRADE_VOLUME | 131 | 无效交易值。 尝试停止所有运行交易，改变程序逻辑。 |
| ERR_MARKET_CLOSED | 132 | 市场关闭。稍后重新尝试。 |
| ERR_TRADE_DISABLED | 133 | 交易被禁止。所有运行交易必须停止。 |
| ERR_NOT_ENOUGH_MONEY | 134 | 没有足够的资金。带有相同参量的交易必须重复。稍后用小额的资金重试， 确定没有足够的资金完成交易。 |
| ERR_PRICE_CHANGED | 135 数据 应用 RefreshRates 函数重试。 | |
| ERR_OFF_QUOTES | 136 | 没有报价格 数据 应用 RefreshRates 函数重试。 |
| ERR_REQUOTE | 138 | 重新请求报价格。刷新数据 可以应用 RefreshRates 函数重试。 如果错误没有消失，尝试停止所有运行交易，改变程序逻辑。 |
| ERR_ORDER_LOCKED | 139 | 交易定单被锁住。尝试停止所有运行交易，改变程序逻辑。 |
| ERR_LONG_POSITIONS_ONLY_ALLOWED | 140 | 只允许买进。SELL 不再重复。 |
| ERR_TOO_MANY_REQUESTS | 141 | 请求过多。 过多的请求必须减少，程序逻辑需要改变。 |
| | 142 | 定单按次序排列。它不是一个错误,而是客户端和服务器交易之间一个代码。当断开 |

| | | |
|---|---|---|
| | | 或重新连接执行交易时，这种代码的出现次数非常少。此代码与误差 128 一样处理。 |
| | 143 | 定单已经被执行交易商接受。它不是一个错误,而是客户端和服务器交易之间一个代码。当断开或重新连接执行交易时， 这种代码的出现次数非常少。此代码与误差 128 一样处理。 |
| | 144 | 在手动确认期间定单已经被客户放弃。它不是一个错误,而是客户端和服务器交易之间一个代码。 |
| ERR_TRADE_MODIFY_DENIED | 145 | 修改被否定。由于太近或被锁定。数据 应用 RefreshRates 函数重试。 |
| ERR_TRADE_CONTEXT_BUSY | 146 | 交易繁忙。只有在 IsTradeContextBusy 函数错误返回后重试。 |
| ERR_TRADE_EXPIRATION_DENIED | 147 | 否定挂单交易期限。如果期限为零可以重试。 |
| ERR_TRADE_TOO_MANY_ORDERS | 148 | 开仓和挂单交易总数已经达到经纪人设定。只有在现有仓位关闭或删除之后才可以开新仓位或挂单。 |

# OrderClose

bool OrderClose(int ticket, double lots, double price, int slippage, void Color)
对定单进行平仓操作。如果函数成功，返回的值是真实的。如果函数失败，返回的值是假的。获得详细错误信息，请查看 GetLastError()函数。
参量:
ticket - 定单编号。
lots - 手数。
price - 收盘价格。
slippage - 最高划点数。
Color - 图表中标记颜色。如果参量丢失，CLR_NONE 值将不会在图表中画出。
示例:

```
  if(iRSI(NULL,0,14,PRICE_CLOSE,0)>75)
     {
      OrderClose(order_id,1,Ask,3,Red);
      return(0);
     }
```

# OrderCloseBy

bool OrderCloseBy( int ticket, int opposite, void Color)
用相反定单对打开仓位进行平仓操作。如果函数成功，返回的值是真实的。如果函数失败，返回的值是假的。获得详细错误信息，请查看 GetLastError()函数。
参量:
ticket - 定单编号。
opposite - 相对定单编号
Color - 图表中标记颜色。如果参量丢失，CLR_NONE 值将不会在图表中画出
示例:

```
  if(iRSI(NULL,0,14,PRICE_CLOSE,0)>75)
    {
     OrderCloseBy(order_id,opposite_id);
     return(0);
    }
```

# OrderClosePrice

double OrderClosePrice()
对于当前选择定单返回收盘价格。
注解:定单必须用 OrderSelect()函数提前选定。
示例:

```
  if(OrderSelect(ticket,SELECT_BY_POS)==true)
    Print("对于定单 ", 定单编号" = ",OrderClosePrice()的收盘价格);
  else
    Print("OrderSelect 失败错误代码是",GetLastError());
```

# OrderCloseTime

datetime OrderCloseTime()
对于当前选择定单返回平仓时间。如果定单时间不是 0，所选定单会从账户历史重新尝试。开仓和挂单交易平仓时间必须等于 0。
注解:定单必须用 OrderSelect()函数提前选定。
示例:

```
  if(OrderSelect(10,SELECT_BY_POS,MODE_HISTORY)==true)
    {
     datetime ctm=OrderOpenTime();
     if(ctm>0) Print("定单 10 "开仓时间, ctm);
     ctm=OrderCloseTime();
     if(ctm>0) Print("定单 10 "平仓时间, ctm);
    }
  else
```

```
    Print("OrderSelect 失败错误代码是",GetLastError());
```

# OrderComment

string OrderComment( )
返回定单的注释。
注解:定单必须用 OrderSelect()函数提前选定。
示例:

```
  string comment;
  if(OrderSelect(10,SELECT_BY_TICKET)==false)
    { Print("OrderSelect  失败错误代码是",GetLastError());
      return(0);
    }
  comment = OrderComment();
  // ...
```

# OrderCommission

double OrderCommission( )
返回定单的佣金数。
注解:定单必须用 OrderSelect()函数提前选定。
示例:

```
  if(OrderSelect(10,SELECT_BY_POS)==true)
    Print("定单 10 "佣金,OrderCommission());
  else
   Print("OrderSelect  失败错误代码是",GetLastError());
```

# OrderDelete

bool OrderDelete( int ticket, void Color)
删除先前打开挂单。如果函数成功，返回的值是真实的。如果函数失败，返回的值是假的。获得详细错误信息，请查看 GetLastError()函数。
参量:
ticket - 定单编号。
Color - 图表中标记颜色。如果参量丢失，CLR_NONE 值将不会在图表中画出。
示例:

```
  if(Ask>var1)
    {
     OrderDelete(order_ticket);
     return(0);
    }
```

# OrderExpiration

datetime OrderExpiration( )
返回挂单的有效日期。
注解:定单必须用 OrderSelect()函数提前选定。
示例:

```
if(OrderSelect(10, SELECT_BY_TICKET)==true)
   Print("定单 #10 有效日期为",OrderExpiration());
else
   Print("OrderSelect 返回的",GetLastError()错误);
```

# OrderLots

double OrderLots( )
返回选定定单的手数。
注解:定单必须用 OrderSelect()函数提前选定。
示例:

```
if(OrderSelect(10,SELECT_BY_POS)==true)
   Print("定单 10 "手数,OrderLots());
else
   Print("OrderSelect 返回的 ",GetLastError()错误);
```

# OrderMagicNumber

int OrderMagicNumber( )
返回选定订单的指定编号
注解:定单必须用 OrderSelect()函数提前选定。
示例:

```
if(OrderSelect(10,SELECT_BY_POS)==true)
   Print("定单 10 "指定编号, OrderMagicNumber());
else
   Print("OrderSelect 返回的 ",GetLastError()错误);
```

# OrderModify

bool OrderModify( int ticket, double price, double stoploss, double takeprofit, datetime expiration, void arrow_color)
对于先前的开仓或挂单进行特性修改。如果函数成功，返回的值为 TRUE。如果函数失败，返回的值为 FALSE。 获得详细的错误信息,查看 GetLastError()函数。
注解: 开价格和有效时间的改变只对挂单而言。
如果未改变的值作为函数参量通过，将会生成错误 1 (ERR_NO_RESULT)。

在一些服务器中挂单的有效时间会被隐藏。这种情况下，当一个非零值在 有效参量被指定时，将生成错误 147 (ERR_TRADE_EXPIRATION_DENIED)。
参量:
ticket - 定单编号。
price - 收盘价格
stoploss - 新止损水平。
takeprofit - 新赢利水平。
expiration - 挂单有效时间。
arrow_color - 在图表中允许对止损/赢利颜色进行修改。如果参量丢失或存在 CLR_NONE 值，在图表中将不会显示。
示例:

```
  if(TrailingStop>0)
    {
     OrderSelect(12345,SELECT_BY_TICKET);
     if(Bid-OrderOpenPrice()>Point*TrailingStop)
       {
        if(OrderStopLoss()<Bid-Point*TrailingStop)
          {

OrderModify(OrderTicket(),OrderOpenPrice(),Bid-Point*TrailingStop,OrderTakeProfit(),0,Blue);
           return(0);
          }
       }
    }
```

# OrderOpenPrice

double OrderOpenPrice( )
对于当前选择定单返回开价格。
定单必须由 OrderSelect() 函数首先选定。
示例:

```
  if(OrderSelect(10, SELECT_BY_POS)==true)
    Print("对于定单 10 开价格",OrderOpenPrice());
  else
    Print("OrderSelect 返回错误",GetLastError());
```

# OrderOpenTime

datetime OrderOpenTime( )
对于当前选择定单返回买入时间。
注解:定单必须用 OrderSelect()函数提前选定。
示例:

```
  if(OrderSelect(10, SELECT_BY_POS)==true)
```

```
    Print("定单 10  买入时间",OrderOpenTime());
  else
    Print("OrderSelect  返回的错误  ",GetLastError());
```

# OrderPrint

void OrderPrint( )
按照以下形式打印选择定单信息:
定单编号; 买入时间; 交易业务;  手数总数; 开盘价格; 止损; 赢利; 平仓时间; 收盘价格; 佣金; 掉期; 盈利; 注释; 指定编码; 挂单有效日期
定单必须用 OrderSelect()函数提前选定。
示例:

```
  if(OrderSelect(10, SELECT_BY_TICKET)==true)
    OrderPrint();
  else
   Print("OrderSelect  失败错误代码是",GetLastError());
```

# OrderProfit

double OrderProfit( )
对于选择定单返回净盈利值  (除掉期和佣金外)。对于开仓位当前不真实盈利。对于平仓为固定盈利。
对于当前选择定单返回盈利。
注解:定单必须用 OrderSelect()函数提前选定。
示例:

```
  if(OrderSelect(10, SELECT_BY_POS)==true)
    Print("定单  10  盈利",OrderProfit());
  else
    Print("OrderSelect 返回的错误",GetLastError());
```

# OrderSelect

bool OrderSelect( int index, int select, void pool)
函数选择定单。如果函数成功，返回的值为 TRUE。如果函数失败，返回的值为 FALSE。获得详细错误信息，请查看 GetLastError()函数。
如果定单编号被选定，此 pool 参量被认知。此定单编号为唯一识别符。找出所选定单的列表，它的平仓时间必须进行分析。如果定单卖出时间为零， 开单和挂单将从终端位置列表打开。可以从定单类型区别开挂单和开单。 如果定单的卖出时间不等于 0， 平单和删除定单是在终端历史中被选择。他们同样可以区分定单类型。
参量:
index  -  定单索引。
select  -  选定模式。可以为以下的任意值:

SELECT_BY_POS
SELECT_BY_TICKET 。
pool - 可选择定单索引。当选择 SELECT_BY_POS 参量时使用。可以为以下的任意值:
MODE_TRADES (default)- 来自交易的定单(开单和挂单)，
MODE_HISTORY - 来自历史的定单(平仓和取消定单)。

示例:

```
if(OrderSelect(12470, SELECT_BY_TICKET)==true)
  {
   Print("定单 #12470 开价格", OrderOpenPrice());
   Print("定单 #12470 收盘价格 ", OrderClosePrice());
  }
else
  Print("OrderSelect 返回的错误 ",GetLastError());
```

# OrderSend

int OrderSend(string symbol, int cmd, double volume, double price, int slippage, double stoploss, double takeprofit, void comment, void magic, void expiration, void arrow_color)
参量:
symbol - 交易货币对。
cmd - 购买方式。可以是购买方式列举的任意值。
volume - 购买手数。
price - 收盘价格。
slippage - 最大允许滑点数。
stoploss - 止损水平。
takeprofit - 赢利水平。
comment - 注解文本。注解的最后部分可以由服务器改变。
magic - 定单指定码。可以作为用户指定识别码使用。
expiration - 定单有效时间(只限挂单)。
arrow_color - 图表上箭头颜色。如果参量丢失或存在 CLR_NONE 价格值不会在图表中画出。
对于 OrderSend()函数的交易类型。可以是以下任意值:

| 常数 | 值 | 描述 |
|---|---|---|
| OP_BUY | 0 | 买仓 |
| OP_SELL | 1 | 卖仓 |
| OP_BUYLIMIT | 2 | 买挂单交易 |
| OP_SELLLIMIT | 3 | 卖挂单交易 |
| OP_BUYSTOP | 4 | 买停挂单交易 |
| OP_SELLSTOP | 5 | 卖停挂单交易 |

示例:

```
int ticket;
if(iRSI(NULL,0,14,PRICE_CLOSE,0)<25)
  {
```

```
      ticket=OrderSend(Symbol(),OP_BUY,1,Ask,3,Ask-25*Point,Ask+25*Point,"My order
#2",16384,0,Green);
      if(ticket<0)
        {
         Print("OrderSend 失败错误 #",GetLastError());
         return(0);
        }
    }
```

# OrdersHistoryTotal

int OrdersHistoryTotal( )
在账户历史返回关闭定单数加载进入终端。历史列表的大小取决于终端的"帐户历史" 表格的当前的设置.
示例:

```
  // 来自交易历史的恢复信息
  int i,hstTotal=OrdersHistoryTotal();
  for(i=0;i<hstTotal;i++)
    {
     //---- 检查选择结果
     if(OrderSelect(i,SELECT_BY_POS,MODE_HISTORY)==false)
       {
        Print("带有 (",GetLastError(),")错误的历史失败通道");
        break;
       }
     // 定单的一些工作
```

# OrderStopLoss

double OrderStopLoss( )
对于当前选择定单返回止损值。
注解:定单必须用 OrderSelect()函数提前选定。
示例:

```
  if(OrderSelect(ticket,SELECT_BY_POS)==true)
     Print("对于 10 止损值", OrderStopLoss());
  else
    Print("OrderSelect 失败错误代码是",GetLastError());
```

# OrdersTotal

int OrdersTotal( )
返回市场和挂单的总数

示例:

```
int handle=FileOpen("OrdersReport.csv",FILE_WRITE|FILE_CSV,"\t");
if(handle<0) return(0);
// 写标题
FileWrite(handle,"#","开价格","买入时间","货币对","手数");
int total=OrdersTotal();
// 编写定单命令
for(int pos=0;pos<total;pos++)
   {
    if(OrderSelect(pos,SELECT_BY_POS)==false) continue;

FileWrite(handle,OrderTicket(),OrderOpenPrice(),OrderOpenTime(),OrderSymbol(),OrderLots());
   }
FileClose(handle);
```

# OrderSwap

double OrderSwap( )
对于当前选择定单返回掉期值。
注解:定单必须用 OrderSelect()函数提前选定。
示例:

```
if(OrderSelect(order_id, SELECT_BY_TICKET)==true)
   Print("对于定单 #掉期", order_id, " ",OrderSwap());
else
  Print("OrderSelect 失败错误代码是",GetLastError());
```

# OrderSymbol

string OrderSymbol( )
对于选择定单返回定单货币对值。
注解:定单必须用 OrderSelect()函数提前选定。
示例:

```
if(OrderSelect(12, SELECT_BY_POS)==true)
   Print("定单 #货币对", OrderTicket(), " is ", OrderSymbol());
else
  Print("OrderSelect 失败错误代码是",GetLastError());
```

# OrderTakeProfit

double OrderTakeProfit( )
对于当前选择定单返回赢利值。

注解:定单必须用 OrderSelect()函数提前选定。
示例:

```
if(OrderSelect(12, SELECT_BY_POS)==true)
   Print("定单 #",OrderTicket()," 盈利: ", OrderTakeProfit());
else
   Print("OrderSelect() 返回错误 - ",GetLastError());
```

# OrderTicket

int OrderTicket( )
对于当前选择定单返回定单编号。
注解:定单必须用 OrderSelect()函数提前选定。
示例:

```
if(OrderSelect(12, SELECT_BY_POS)==true)
   order=OrderTicket();
else
  Print("OrderSelect 失败错误代码",GetLastError());
```

# OrderType

int OrderType( )
对于当前选择定单返回定单类型。可以是以下的任意值:
OP_BUY -买进,
OP_SELL - 卖出,
OP_BUYLIMIT - 挂单买入限定,
OP_BUYSTOP - 挂单停止限定,
OP_SELLLIMIT - 挂单卖出限定,
OP_SELLSTOP - 挂单停止限定。
注解: 定单必须由 OrderSelect()函数选择。
示例:

```
int order_type;
if(OrderSelect(12, SELECT_BY_POS)==true)
   {
    order_type=OrderType();
    // ...
   }
else
   Print("OrderSelect() 返回错误 - ",GetLastError());
```

# Window functions 窗口函数

当前图表窗口的一组函数。

## HideTestIndicators 隐藏指标

void HideTestIndicators( bool hide)
函数设置使用智能交易隐藏指标。在交易被测试以后打开相应的图表，标出的指标将不会出现在测试图表中。 查看每个指标需应用当前隐藏的标记和第一个标记。
必须注明只有这些指标才可以在测试图表中画出。
参量:
hide - 如果需要隐藏指标为 TRUE，否则为 FALSE。
示例:

```
HideTestIndicators(true);
MaCurrent=iMA(NULL,0,56,0,MODE_EMA,PRICE_CLOSE,0);
MaPrevious=iMA(NULL,0,56,0,MODE_EMA,PRICE_CLOSE,1);
HideTestIndicators(false);
```

## Period 使用周期

int Period( )
返回使用 周期 (图表周期)的分钟总数。
示例:

```
Print("时间周期 ", Period());
```

## RefreshRates 刷新预定义变量和系列数组的数据

bool RefreshRates( )
刷新预定义变量和系列数组的数据。在智能交易计算时间过长时，这个功能可以自动更新数据。如果数据刷新，返回到 TRUE， 否则返回到 FALSE。只有在客户端内的数据不被更新。如果数据已经更新，接下来输入的行情也一样被更新。
智能交易和脚本只管理本身历史数据的复制本。在智能交易和脚本第一次开启 的时候，当前的商品数据已经复制。 每次智能或脚本开启时，最初的复制本会更新。智能和脚本运作时，数据可能已经过期。
示例:

```
int ticket;
while(true)
  {
   ticket=OrderSend(Symbol(),OP_BUY,1.0,Ask,3,0,0,"expert comment",255,0,CLR_NONE);
   if(ticket<=0)
```

```
        {
          int error=GetLastError();
          //---- 资金不足
          if(error==134) break;
          //---- 10 秒钟等待
          Sleep(10000);
          //---- 刷新价格数据
          RefreshRates();
          break;
        }
      else
        {
          OrderSelect(ticket,SELECT_BY_TICKET);
          OrderPrint();
          break;
        }
    }
```

# Symbol 当前货币对

string Symbol( )
带有当前货币对名称返回字串符文本。
示例:

```
  int total=OrdersTotal();
  for(int pos=0;pos<total;pos++)
    {
     // 因为此时可能平单或删除定单，检测选择结果!
     if(OrderSelect(pos, SELECT_BY_POS)==false) continue;
     if(OrderType()>OP_SELL || OrderSymbol()!=Symbol()) continue;
     // 执行过程...
    }
```

# WindowBarsPerChart 可见柱总数

int WindowBarsPerChart( )
在图表上函数返回可见柱总数。
示例:

```
// 对于可见柱工作。
int bars_count=WindowBarsPerChart();
int bar=WindowFirstVisibleBar();
for(int i=0; i<bars_count; i++,bar--)
  {
   // ...
```

```
}
```

# WindowExpertName 智能交易系统名称

string WindowExpertName( )
从调用函数返回 MQL4 程序中独立执行智能交易，脚本，客户指标和数据库的名称。
示例:

```
string name=WindowExpertName();
GlobalVariablesDeleteAll(name);
```

# WindowFind 返回名称

int WindowFind( string name)
如果发现指标 名称，函数返回包含特殊指标的窗口索引，否则返回 -1。
注解: 如果当 init()函数运行时，客户指标搜索到本身，则 WindowFind()函数返回 -1 。
参量:
name - 指标简称。
示例:

```
int win_idx=WindowFind("MACD(12,26,9)");
```

# WindowFirstVisibleBar 第一个可见柱

int WindowFirstVisibleBar( )
在当前图表窗口函数返回第一个可见柱。必须考虑到价格柱的逆序编号，即从最后一 价格数组中的最后一个指示为 0。最老得柱被索引为柱-1。如果第一个柱的编码为 2 或少于图表中可见柱的总数，意味着图表窗口没有被完整填充。
示例:

```
// 可见柱的工作
int bars_count=WindowBarsPerChart();
int bar=WindowFirstVisibleBar();
for(int i=0; i<bars_count; i++,bar--)
  {
   // ...
  }
```

# WindowHandle

int WindowHandle( string symbol, int timeframe)
返回包含特定图表的系统窗口。 如果货币对和 时间周期的图表暂时还没有开启，显示为 0。
参量:
symbol - 货币对名称。
timeframe - 时间周期。可以是时间周期列举的任意值。 0 意味着当前图表的时间周期。

示例:

```
int win_handle=WindowHandle("USDX",PERIOD_H1);
if(win_handle!=0)
   Print("发现带有 USDX,H1 的窗口。数组将会被立即复制。");
```

# WindowIsVisible 图表在子窗口中可见

bool WindowIsVisible( int index)

如果图表在子窗口中可见，返回 TRUE，否则返回 FALSE。 子图表窗口可以隐藏于指标的可见属性位置。

参量:

index - 图表自窗口索引。

示例:

```
int maywin=WindowFind("MyMACD");
if(maywin>-1 && WindowIsVisible(maywin)==true)
   Print("MyMACD 窗口可见");
else
   Print(" MyMACD 窗口未发现或不可见");
```

# WindowOnDropped

int WindowOnDropped( )

返回智能交易，客户指标和脚本的绑定窗口索引。 只有在智能交易、客户指标或脚本应用鼠标的帮助下，绑定的值是准确的。

注解：对于客户指标初始化(调用 init()函数)，此索引不被定义。

返回的索引被窗口编码(0 为主菜单图表，指标子窗口的开始数字为 1 )。 在运行期间客户指标可以创建子窗口，子窗口的编码不同于。

参见 WindowXOnDropped(), WindowYOnDropped()

示例:

```
if(WindowOnDropped()!=0)
  {
   Print("指标'MyIndicator'必须被主图表窗口接受!");
   return(false);
  }
```

# WindowPriceMax

double WindowPriceMax( void index)

返回当前图表指定子窗口的最大垂直标度的值(0 为主菜单图表，指标子窗口的开始数字为 1 )。 如果子窗口没有指定，最大价格标度的值返回图表窗口。

参见 WindowPriceMin(), WindowFirstVisibleBar(), WindowBarsPerChart()

参量:

index - 图表子窗口索引 (0 -主图表窗口)。
示例:

```
double     top=WindowPriceMax();
double     bottom=WindowPriceMin();
datetime left=Time[WindowFirstVisibleBar()];
int          right_bound=WindowFirstVisibleBar()-WindowBarsPerChart();
if(right_bound<0) right_bound=0;
datetime right=Time[right_bound]+Period()*60;
//----
ObjectCreate("Padding_rect",OBJ_RECTANGLE,0,left,top,right,bottom);
ObjectSet("Padding_rect",OBJPROP_BACK,true);
ObjectSet("Padding_rect",OBJPROP_COLOR,Blue);
WindowRedraw();
```

# WindowPriceMin

double WindowPriceMin( void index)
返回当前图表指定子窗口的最小垂直标度的价格值(0 为主菜单图表，指标子窗口的开始数字为 1 )。 如果子窗口没有指定，最小价格标度的价格值返回图表窗口。
参见 WindowPriceMax(), WindowFirstVisibleBar(), WindowBarsPerChart()
参量:
index - 图表子窗口索引 (0 - 主图表窗口)。
示例:

```
double     top=WindowPriceMax();
double     bottom=WindowPriceMin();
datetime left=Time[WindowFirstVisibleBar()];
int          right_bound=WindowFirstVisibleBar()-WindowBarsPerChart();
if(right_bound<0) right_bound=0;
datetime right=Time[right_bound]+Period()*60;
//----
ObjectCreate("Padding_rect",OBJ_RECTANGLE,0,left,top,right,bottom);
ObjectSet("Padding_rect",OBJPROP_BACK,true);
ObjectSet("Padding_rect",OBJPROP_COLOR,Blue);
WindowRedraw();
```

# WindowPriceOnDropped

double WindowPriceOnDropped( )
返回图表指出的智能交易或脚本价格下滑价格部分。 只有在智能交易、客户指标或脚本应用鼠标的情况下，绑定的值是准确的。注解: 对于客户指标的值是不确定的。
示例:

```
   double     drop_price=WindowPriceOnDropped();
   datetime drop_time=WindowTimeOnDropped();
```

```
//---- 可能未指定 (zero)
if(drop_time>0)
  {
   ObjectCreate("价格下滑水平", OBJ_HLINE, 0, drop_price);
   ObjectCreate("下滑时间", OBJ_VLINE, 0, drop_time);
  }
```

# WindowRedraw

void WindowRedraw( )
重新画出当前图表。在货币对属性改变之后应用。
示例:

```
//---- 对于货币对设置新属性
ObjectMove(object_name1, 0, Time[index], price);
ObjectSet(object_name1, OBJPROP_ANGLE, angle*2);
ObjectSet(object_name1, OBJPROP_FONTSIZE, fontsize);
ObjectSet(line_name, OBJPROP_TIME2, time2);
ObjectSet(line_name, OBJPROP_ANGLE, line_angle);
//---- 现在重画
WindowRedraw();
```

# WindowScreenShot

bool WindowScreenShot( string filename, int size_x, int size_y, void start_bar, void chart_scale, void chart_mode)
以 GIF 文件形式保存当前图像。如果失败，返回 FALSE。详细错误信息，查看 GetLastError() 函数。
图像被储存在 terminal_dir\experts\files (terminal_dir\tester\files 测试情况下) 目录中或是子目录。
参量:
filename - 屏幕映像文件名称。
size_x - 屏幕宽度映像点。
size_y - 屏幕高度影响点。
start_bar - I 第一个可见柱的屏幕映像。如果价格值设定为 0 ，当前的第一个可见柱 将被除去。如果价格值为负值， 结束图的映像将会产生。
chart_scale - 对于屏幕映像水平的图标度。 可以在范围从 0 到 5 之间。 如果没有值或者为负值，当前图表将被应用。
chart_mode - 图表显示模式。可以是以下价格值: ICHART_BAR (0 是柱的次序), CHART_CANDLE (1 是蜡烛柱的次序）,CHART_LINE (2 是收盘价格线)。如果没有价格值或者为负值，图表会以当前模式显示。
示例:

```
int lasterror=0;
//----测试者平仓或多个仓
```

```
if(IsTesting() && ExtTradesCounter<TradesTotal())
  {
   //---- 使 WindowScreenShot 进行检测
   if(!WindowScreenShot("shots\\tester"+ExtShotsCounter+".gif",640,480))
       lasterror=GetLastError();
   else ExtShotsCounter++;
   ExtTradesCounter=TradesTotal();
  }
```

# WindowTimeOnDropped

datetime WindowTimeOnDropped( )
返回图表指出的智能交易或脚本价格下滑时间部分。 只有在智能交易、客户指标或脚本应用鼠标的帮助下，绑定的值是准确的。
注解: 对于客户指标的价格值是不确定的。
示例:

```
double     drop_price=WindowPriceOnDropped();
datetime drop_time=WindowTimeOnDropped();
//---- 可能未指定 (zero)
if(drop_time>0)
  {
   ObjectCreate("Dropped price line", OBJ_HLINE, 0, drop_price);
   ObjectCreate("Dropped time line", OBJ_VLINE, 0, drop_time);
  }
```

# WindowsTotal 指标窗口数

int WindowsTotal( )
返回在图表中指标窗口数(包括主图表)。
示例:

```
Print("窗口数 = ", WindowsTotal());
```

# WindowXOnDropped

int WindowXOnDropped( )
当以映像点 X 轴图表窗口的客户区域智能交易或脚本下滑时，返回价格值。 只有在智能交易、客户指标或脚本应用鼠标("Drag'n'Drop") 的情况下，绑定的值是准确的。
参见 WindowYOnDropped(), WindowOnDropped()
示例:

```
Print("智能交易下滑点 x=",WindowXOnDropped()," y=",WindowYOnDropped());
```

# WindowYOnDropped

int WindowYOnDropped( )

当以映像点 Y 轴图表窗口的客户区域智能交易或脚本下滑时，返回价格值。 只有在智能交易、客户指标或脚本应用鼠标("Drag'n'Drop")的帮助下，绑定的值是准确的。

参见 WindowXOnDropped(), WindowPriceOnDropped(), WindowOnDropped()

示例:

```
Print" 被获取智能交易到窗口的点 x=",WindowXOnDropped()," y=",WindowYOnDropped());
```

# Obsolete functions 过时的函数

MQL4 程序在不断的完善发展中，由于系统化的需要，一些名称被重新命名。旧的功能名称与 MetaEditor 资料无法连接。有时，编辑器会以适当的途径接纳。不过，我们还是要求使用新名称。

| 旧名称 | 新名称 |
| --- | --- |
| BarsPerWindow | WindowBarsPerChart |
| ClientTerminalName | TerminalName |
| CurTime | TimeCurrent |
| CompanyName | TerminalCompany |
| FirstVisibleBar | WindowFirstVisibleBar |
| Highest | iHighest |
| HistoryTotal | OrdersHistoryTotal |
| LocalTime | TimeLocal |
| Lowest | iLowest |
| ObjectsRedraw | WindowRedraw, |
| PriceOnDropped | WindowPriceOnDropped |
| ScreenShot | WindowScreenShot |
| ServerAddress | AccountServer |
| TimeOnDropped | WindowTimeOnDropped |