Expert Advisor Programming for MetaTrader 5

Include Function Reference

This document is a quick reference to the functions inside the include files distributed in the source code download. The details and usage of these functions are described in the book.

The include files are located in the `\MQL5\Include\Mql5Book\` folder of your MetaTrader 5 installation. To include a file in your program, use the `#include` directive:

```
#include <Mql5Book\Trade.mqh>
```

Trade.mqh

CTrade class

```
bool CTrade::Buy(string pSymbol, double pVolume, double pStop = 0, double pProfit = 0,
       string pComment = NULL)

bool CTrade::Sell(string pSymbol, double pVolume, double pStop = 0, double pProfit = 0,
       string pComment = NULL)
```

Opens a buy or sell position. Returns `true` if the position is placed, `false` otherwise.

- `pSymbol` - The symbol to open the position on.
- `pVolume` - The trade volume in lots.
- `pStop` - The stop loss price (optional).
- `pProfit` - The take profit price (optional).
- `pComment` - The trade comment (optional).

```
bool CTrade::BuyStop(string pSymbol, double pVolume, double pPrice, double pStop = 0,
       double pProfit = 0, datetime pExpiration = 0, string pComment = NULL)

bool CTrade::SellStop(string pSymbol, double pVolume, double pPrice, double pStop = 0,
       double pProfit = 0, datetime pExpiration = 0, string pComment = NULL)

bool CTrade::BuyLimit(string pSymbol, double pVolume, double pPrice, double pStop = 0,
       double pProfit = 0, datetime pExpiration = 0, string pComment = NULL)

bool CTrade::SellLimit(string pSymbol, double pVolume, double pPrice, double pStop = 0,
       double pProfit = 0, datetime pExpiration = 0, string pComment = NULL)
```

Opens a pending order of the specified type. Returns `true` if the position is placed, `false` otherwise.

- `pSymbol` - The symbol to open the position on.
- `pVolume` - The trade volume in lots.
- `pPrice` - The order opening price.
- `pStop` - The stop loss price (optional).
- `pProfit` - The take profit price (optional).
- `pExpiration` - The order expiration time (optional).
- `pComment` - The trade comment (optional).

```
bool CTrade::BuyStopLimit(string pSymbol, double pVolume, double pPrice, double pStopLimit,
       double pStop = 0, double pProfit = 0, datetime pExpiration = 0,
       string pComment = NULL)

bool CTrade::SellStopLimit(string pSymbol, double pVolume, double pPrice, double pStopLimit,
       double pStop = 0, double pProfit = 0, datetime pExpiration = 0,
       string pComment = NULL);
```

Opens a buy or sell stop limit order. Returns `true` if the position is placed, `false` otherwise.

- `pSymbol` - The symbol to open the position on.
- `pVolume` - The trade volume in lots.
- `pPrice` - The order opening price.
- `pStopLimit` - The stop limit price.
- `pStop` - The stop loss price (optional).
- `pProfit` - The take profit price (optional).
- `pExpiration` - The order expiration time (optional).
- `pComment` - The trade comment (optional).

```
bool CTrade::ModifyPosition(string pSymbol, double pStop, double pProfit = 0)
```

Modifies the stop loss and take profit price of the current position. If either the stop loss or take profit price will not be changed, the current price must be passed to the function. You can retrieve the current price using the `PositionStopLoss()` and `PositionTakeProfit()` functions. Returns `true` if the position was modified, `false` otherwise.

- `pSymbol` - The symbol of the position to modify.
- `pStop` - The new or current stop loss price.
- `pProfit` - The new or current take profit price (optional).

bool CTrade::ModifyPending(ulong pTicket, double pPrice, double pStop, double pProfit, datetime pExpiration=0)

Modifies the opening price, stop loss or take profit price of a specified order. If either the opening price, stop loss or take profit price will not be changed, the current price must be passed to the function. You can retrieve the current prices of the order using the `OrderOpenPrice()`, `OrderStopLoss()` or `OrderTakeProfit()` functions from the `Pending.mqh` file. Returns `true` if the order was modified, `false` otherwise.

- `pTicket` - The ticket number of the order to modify.
- `pPrice` - The new or current order opening price.
- `pStop` - The new or current stop loss price.
- `pProfit` - The new or current take profit price.
- `pExpiration` - The new or current order expiration time (optional).

bool CTrade::Close(string pSymbol, double pVolume = 0, string pComment = NULL)

Closes the current position on the specified symbol. Returns `true` if the position was closed successfully, `false` otherwise.

- `pSymbol` - The symbol of the position to close.
- `pVolume` - The volume to partially close. If `pVolume` is greater than the current position volume, or `pVolume` is 0, the entire position will be closed.
- `pComment` - A comment to add to the deal.

bool CTrade::Delete(ulong pTicket)

Deletes the specified pending order. Returns `true` if the order was closed successfully.

- `pTicket` - The ticket number of the order to delete.

void CTrade::MagicNumber(ulong pMagic)
void CTrade::Deviation(ulong pDeviation)
void CTrade::FillType(ENUM_ORDER_TYPE_FILLING pFill)

Sets the magic number, deviation or fill type for all orders. Call the function once in the `OnInit()` event handler to set the relevant property.

- `pMagic` - The magic number to associate with all orders.
- `pDeviation` - The deviation in points.
- `pFill` - The fill type to use.

double BuyStopLoss(string pSymbol,int pStopPoints, double pOpenPrice = 0)
double SellStopLoss(string pSymbol,int pStopPoints, double pOpenPrice = 0)

Calculates and returns a stop loss price for a buy or sell order.

- `pSymbol` - The symbol to calculate the stop loss price for.
- `pStopPoints` - The distance between the stop loss and the order opening price in points.
- `pOpenPrice` - The order opening price. If 0, the current Bid or Ask price will be used.

double BuyTakeProfit(string pSymbol,int pProfitPoints, double pOpenPrice = 0)
double SellTakeProfit(string pSymbol,int pProfitPoints, double pOpenPrice = 0)

Calculates and returns a take profit price for a buy or sell order.

- `pSymbol` - The symbol to calculate the take profit price for.
- `pProfitPoints` - The distance between the take profit and the order opening price in points.
- `pOpenPrice` - The order opening price. If 0, the current Bid or Ask price will be used.

bool CheckAboveStopLevel(string pSymbol, double pPrice, int pPoints = 10)
bool CheckBelowStopLevel(string pSymbol, double pPrice, int pPoints = 10)

Verifies whether a pending order opening price, stop loss or take profit price is valid relative to the current Bid or Ask price. If the price is within the stop level (+/- pPoints), it is invalid. Returns `true` if the price is valid, `false` otherwise.

- `pSymbol` - The symbol to check.
- `pPrice` - The price to check.
- `pPoints` - Adds/subtracts a specified number of points to the stop level price.

double AdjustAboveStopLevel(string pSymbol, double pPrice, int pPoints = 10)
double AdjustBelowStopLevel(string pSymbol, double pPrice, int pPoints = 10)

Verifies whether a pending order opening price, stop loss or take profit price is valid relative to the current Bid or Ask price. If the price is within the stop level (+/- pPoints), it is invalid and will be automatically adjusted to a valid price. Returns `pPrice` if the price is valid, or an adjusted price otherwise.

- `pSymbol` - The symbol to check.
- `pPrice` - The price to check.
- `pPoints` - Adds/subtracts a specified number of points to the stop level price.

```
string PositionComment(string pSymbol = NULL)
long PositionType(string pSymbol = NULL)
long PositionIdentifier(string pSymbol = NULL)
double PositionOpenPrice(string pSymbol = NULL)
long PositionOpenTime(string pSymbol = NULL)
double PositionVolume(string pSymbol = NULL)
double PositionStopLoss(string pSymbol = NULL)
double PositionTakeProfit(string pSymbol = NULL)
double PositionProfit(string pSymbol = NULL)
```

Returns information about the current position.

- `pSymbol` - The position to return information for. If not specified, the current chart symbol will be used.

```
string CheckOrderType(ENUM_ORDER_TYPE pType)
```

Returns a string describing the order type, for example "buy" or "sell stop."

- `pType` - An order type constant.

Pending.mqh

CPending class

```
int CPending::BuyLimit(string pSymbol)
int CPending::SellLimit(string pSymbol)
int CPending::BuyStop(string pSymbol)
int CPending::SellStop(string pSymbol)
int CPending::BuyStopLimit(string pSymbol)
int CPending::SellStopLimit(string pSymbol)
int CPending::TotalPending(string pSymbol)
```

Returns the number of currently open orders of the specified type for the specified symbol.

- `pSymbol` - The symbol to return the number of open orders for.

```
void CPending::GetTickets(string pSymbol, ulong &pTickets[]);
```

Returns an array filled with the ticket numbers of open orders on the specified symbol.

- `pSymbol` - The symbol to return the order tickets for.
- `pTickets` - An array passed by reference that will be filled with the order tickets.

```
long OrderType(ulong pTicket)
long OrderExpirationTime(ulong pTicket)
long OrderExpirationType(ulong pTicket)
long OrderMagicNumber(ulong pTicket)
double OrderVolume(ulong pTicket)
double OrderOpenPrice(ulong pTicket)
double OrderStopLimit(ulong pTicket)
double OrderStopLoss(ulong pTicket)
double OrderTakeProfit(ulong pTicket)
string OrderComment(ulong pTicket)
```

Returns information about the specified pending order.

- `pTicket` - The ticket of the pending order to return information for.

Price.mqh

CBars class

```
void CBars::Update(string pSymbol,ENUM_TIMEFRAMES pPeriod)
```

Updates the price data for the last 100 bars on the specified symbol and period. The `MAX_BARS` constant in the `Price.mqh` file adjusts the number of bars to retrieve data for.

- `pSymbol` - The symbol to retrieve the price data for.
- `pPeriod` - The period of the chart to retrieve the price data for.

```
double CBars::Close(int pShift)
double CBars::High(int pShift)
double CBars::Low(int pShift)
double CBars::Open(int pShift)
datetime CBars::Time(int pShift)
long CBars::TickVolume(int pShift)
long CBars::Volume(int pShift)
```

Returns the specified price for the specified bar. The current bar has a shift value of 0, the previous bar is 1 and so on. The `Update()` function above must be called first to ensure the most current prices are used.

- `pShift` - The shift of the bar to retrieve the price for.

double Ask(string pSymbol=NULL)
double Bid(string pSymbol=NULL)

Retrieves the current Bid or Ask price for the specified symbol.

- `pSymbol` - The symbol to retrieve the price for. If no value is specified, the current chart symbol will be used.

long Spread(string pSymbol=NULL)

Retrieves the current spread for the specified symbol.

- `pSymbol` - The symbol to retrieve the spread for. If no value is specified, the current chart symbol will be used.

long StopLevel(string pSymbol=NULL)

Retrieves the stop level in points for the specified symbol.

- `pSymbol` - The symbol to retrieve the stop level for. If no value is specified, the current chart symbol will be used.

double HighestHigh(string pSymbol, ENUM_TIMEFRAMES pPeriod, int pBars, int pStart = 0)
double LowestLow(string pSymbol, ENUM_TIMEFRAMES pPeriod, int pBars, int pStart = 0)

Retrieves the highest high or lowest low price for the specified bar range.

- `pSymbol` - The symbol to use.
- `pPeriod` - The chart period to use.
- `pBars` - The number of bars to search,
- `pStart` - The starting bar. Default is the current bar.

MoneyManagement.mqh

double MoneyManagement(string pSymbol,double pFixedVol,double pPercent,int pStopPoints)

Calculates a trade volume using a specified percentage of the current balance, as well as the stop loss in points. Returns the calculated trade volume, or if no trade volume can be calculated, the `pFixedVol` value.

- `pSymbol` - The symbol to calculate the trade volume for.
- `pFixedVol` - A default trade volume, used if a trade volume cannot be calculated.
- `pPercent` - A percentage of the current balance to risk.
- `pStopPoints` - The desired stop loss in points.

double VerifyVolume(string pSymbol,double pVolume)

Verifies a trade volume against the minimum, maximum and step values enforced by the trade server. Returns `pVolume` if the trade volume is valid, or an adjusted value if not.

- `pSymbol` - The symbol to use.
- `pVolume` - The trade volume to verify.

double StopPriceToPoints(string pSymbol,double pStopPrice, double pOrderPrice)

Calculates and returns the difference in points between a stop loss price and an order opening price.

- `pSymbol` - The symbol to use.
- `pStopPrice` - The stop loss price.
- `pOrderPrice` - The order opening price.

Trailing.mqh

CTrailing class

```
bool CTrailing::TrailingStop(string pSymbol, int pTrailPoints, int pMinProfit = 0,
       int pStep = 10)

bool CTrailing::TrailingStop(string pSymbol, double pTrailPrice, int pMinProfit = 0,
       int pStep = 10)
```

Implements a trailing stop for open positions. The first variant of the function takes a trailing stop value in points. The second variant takes a trailing stop price. Returns a value of true if the trailing stop was adjusted, false otherwise.

- `pSymbol` - The symbol of the position to adjust the trailing stop for.
- `pTrailPoints` - The trailing stop distance in points.
- `pTrailPrice` - A trailing stop price.
- `pMinProfit` - The minimum profit in points required before the trailing stop will be adjusted (optional).
- `pStep` - Moves the trailing stop in increments. The minimum step is 10 points (optional).

```
bool CTrailing::BreakEven(string pSymbol, int pBreakEven, int pLockProfit = 0)
```

Implements a break even stop for open positions. The stop loss is moves to the order opening price once as soon as the specified break even profit is reached. Returns a value of `true` if the break even stop is adjusted successfully, `false` otherwise.

- `pSymbol` - The symbol of the position to adjust the break even stop for.
- `pBreakEven` - The break even profit in points. When the order profit in points meets or exceeds this amount, the break even stop will trigger.
- `pLockProfit` - Adds or subtracts a specified number of points to the break even stop price (optional).

Timer.mqh

CTimer class

```
bool CTimer::CheckTimer(datetime pStartTime, datetime pEndTime, bool pLocalTime = false)
```

Takes a start and end time and determines whether the current time falls between them. Returns a value of `true` if the current time is between the start and end time, and `false` otherwise.

- `pStartTime` - The start time.
- `pEndTime` - The end time.
- `pLocalTime` - If true, will use your computer's local time. Otherwise, the server time will be used (optional).

```
bool CTimer::DailyTimer(int pStartHour, int pStartMinute, int pEndHour, int pEndMinute,
      bool pLocalTime = false)
```

Calculates a start and end time using a start and end hour and minute. Returns a value of `true` if the current time is between the start and end times, and `false` otherwise.

- `pStartHour` - The start hour.
- `pStartMinute` - The start minute.
- `pEndHour` - The end hour.
- `pEndMinute` - The end minute.
- `pLocalTime` - If true, will use your computer's local time. Otherwise, the server time will be used (optional).

bool CTimer::BlockTimer(TimerBlock &pBlock[], bool pLocalTime = false)

Takes an array that holds several start and end times, and determines whether the current time falls between any of them. The array type uses the `TimerBlock` structure, described below. Returns a value of `true` if the current time falls within one of the start and end times, and `false` otherwise.

- `pBlock` - An array passed by reference that contains start and end times.
- `pLocalTime` - If `true`, will use your computer's local time. Otherwise, the server time will be used (optional).

```
struct TimerBlock
{
       bool enabled;
       int start_day;
       int start_hour;
       int start_min;
       int end_day;
       int end_hour;
       int end_min;
};
```

The `TimerBlock` structure is used for the `BlockTimer()` function. An array is declared using `TimerBlock` as the type, and loaded with data representing start and end times for the current week.

- `enabled` - Set to `true` if the timer for the current array element is enabled, `false` if disabled.
- `start_day` - The start day for the current timer. 0: Sunday, 5: Friday.
- `start_hour` - The start hour for the current timer.
- `start_min` - The start minute for the current timer.
- `end_day` - The end day for the current timer. 0: Sunday, 5: Friday.
- `end_hour` - The end hour for the current timer.
- `end_min` - The end minute for the current timer.

CNewBar class

bool CNewBar::CheckNewBar(string pSymbol, ENUM_TIMEFRAMES pTimeframe)

Checks to see if a new bar has opened since the last tick. Returns a value of `true` if a new bar has opened, and `false` otherwise.

- `pSymbol` - The symbol to use.
- `pTimeframe` - The period of the chart to check.

Indicators.mqh

CIndicator class

`double Main(int pShift=0)`

Returns the value of the first indicator buffer for the specified bar. All indicator classes based on `CIndicator` implement this function.

- `pShift` - The shift of the bar to retrieve the indicator value from.

`void Release()`

Releases the indicator from memory. All indicator classes based on `CIndicator` implement this function.

CiMA class

```
int CiMA::Init(string pSymbol, ENUM_TIMEFRAMES pTimeframe, int pMAPeriod, int pMAShift,
       ENUM_MA_METHOD pMAMethod, ENUM_APPLIED_PRICE pMAPrice)
```

Initializes the moving average indicator.

- `pSymbol` - The symbol to use.
- `pTimeframe` - The chart period to use.
- `pMAPeriod` - The calculation period for the indicator.
- `pMAShift` - The forward or backward shift for the indicator.
- `pMAMethod` - The calculation method for the indicator.
- `pMAPrice` - The price series to use.

CiRSI class

```
int CiRSI::Init(string pSymbol, ENUM_TIMEFRAMES pTimeframe, int pRSIPeriod,
       ENUM_APPLIED_PRICE pRSIPrice)
```

Initializes the RSI indicator.

- `pSymbol` - The symbol to use.
- `pTimeframe` - The chart period to use.
- `pRSIPeriod` - The calculation period for the indicator.

- `pRSIPrice` - The price series to use.

CiStochastic class

`int CiStochastic::Init(string pSymbol, ENUM_TIMEFRAMES pTimeframe, int pKPeriod, int pDPeriod, int pSlowing, ENUM_MA_METHOD pMAMethod, ENUM_STO_PRICE pPrice)`

Initializes the stochastic indicator.

- `pSymbol` - The symbol to use.
- `pTimeframe` - The chart period to use.
- `pKPeriod` - The calculation period for the %K line.
- `pDPeriod` - The calculation period for the %D line,
- `pSlowing` - The slowing period of the indicator.
- `pMAMethod` - The calculation method of the indicator.
- `pPrice` - The price series to use.

`double CiStochastic::Signal(int pShift=0)`

Returns the value for the *%D* line for the specified bar.

- `pShift` - The shift of the bar to retrieve the indicator value from.

CiBollinger class

`int CiBollinger::Init(string pSymbol, ENUM_TIMEFRAMES pTimeframe, int pPeriod, int pShift, double pDeviation, ENUM_APPLIED_PRICE pPrice)`

Initializes the Bollinger bands indicator.

- `pSymbol` - The symbol to use.
- `pTimeframe` - The chart period to use.
- `pPeriod` - The calculation period to use for the indicator.
- `pShift` - The forward or backward shift of the indicator.
- `pDeviation` - The standard deviation of the outer bands.
- `pPrice` - The price series to use.

double CiBollinger::Lower(int pShift=0)
double CiBollinger::Upper(int pShift=0)

Returns the lower or upper band price for the specified bar.

- `pShift` - The shift of the bar to retrieve the indicator value from.

CiMACD class

int CiMACD::Init(string pSymbol, ENUM_TIMEFRAMES pTimeframe, int pFastPeriod, int pSlowPeriod,
int pSignalPeriod, ENUM_APPLIED_PRICE pPrice)

Initializes the MACD indicator.

- `pSymbol` - The symbol to use.
- `pTimeframe` - The chart period to use.
- `pFastPeriod` - The calculation period to use for the histogram.
- `pSlowPeriod` - The calculation period to use for the histogram.
- `pSignalPeriod` - The calculation period for the signal line.
- `pPrice` - The price series to use.

double CiMACD::Signal(int pShift=0)

Returns the price of the signal line for the specified bar.

- `pShift` - The shift of the bar to retrieve the indicator value from.

CiSAR class

int CiSAR::Init(string pSymbol, ENUM_TIMEFRAMES pTimeframe, double pStep, double pMaximum)

Initializes the SAR indicator.

- `pSymbol` - The symbol to use.
- `pTimeframe` - The chart period to use.
- `pStep` - The step parameter for the SAR indicator.
- `pMaximum` - The maximum parameter for the SAR indicator.

CiADX class

`int CiADX::Init(string pSymbol, ENUM_TIMEFRAMES pTimeframe, int pPeriod)`

Initializes the ADX indicator.

- `pSymbol` - The symbol to use.
- `pTimeframe` - The chart period to use.
- `pPeriod` - The calculation period to use.

`double CiADX::Plus(int pShift=0)`
`double CiADX::Minus(int pShift=0)`

Returns the price for the plus and minus DI lines for the specified bar.

- `pShift` - The shift of the bar to retrieve the indicator value from.