Expert Advisor Programming for MetaTrader 4

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 `\MQL4\Include\Mql4Book\` folder of your MetaTrader 4 installation. To include a file in your program, use the `#include` directive:

```
        #include <Mql4Book\Trade.mqh>
```

Trade.mqh

CTrade class

```
int CTrade::OpenBuyOrder(string pSymbol, double pVolume, string pComment = "Buy order",
       color pArrow = clrGreen)

int CTrade::OpenSellOrder(string pSymbol, double pVolume, string pComment = "Sell order",
       color pArrow = clrRed)
```

Opens a buy or sell order. Returns the order ticket number, or `-1` if the order is not placed.

- `pSymbol` - The symbol to open the order on.
- `pVolume` - The trade volume in lots.
- `pComment` - The trade comment (optional).
- `pArrow` - The arrow color (optional).

```
int CTrade::OpenBuyStopOrder(string pSymbol, double pVolume, double pPrice, double pStop,
       double pProfit, string pComment = "Buy stop order", datetime pExpiration = 0,
       color pArrow = clrBlue)

int CTrade::OpenSellStopOrder(string pSymbol, double pVolume, double pPrice, double pStop,
       double pProfit, string pComment = "Sell stop order", datetime pExpiration = 0,
       color pArrow = clrIndigo)

int CTrade::OpenBuyLimitOrder(string pSymbol, double pVolume, double pPrice, double pStop,
       double pProfit, string pComment = "Buy limit order", datetime pExpiration = 0,
       color pArrow = clrCornflowerBlue)

int CTrade::OpenSellLimitOrder(string pSymbol, double pVolume, double pPrice, double pStop,
       double pProfit, string pComment = "Sell limit order", datetime pExpiration = 0,
        color pArrow = clrMediumSlateBlue)
```

Opens a pending order of the specified type. Returns the order ticket number, or `-1` if the order is not placed.

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

`bool CTrade::CloseMarketOrder(int pTicket, double pVolume = 0, color pArrow = clrRed)`

Closes a single market order. Return `true` if the order was closed, `false` otherwise.

- `pTicket` - The ticket number of the order to close.
- `pVolume` - The trade volume to close. If `pVolume` is less than the order volume, the order will be partially closed if the broker supports it. If `pVolume` is greater than the order volume, or `pVolume` is 0, the entire order will be closed.
- `pArrow` - The arrow color (optional).

`bool CTrade::CloseAllBuyOrders()`
`bool CTrade::CloseAllSellOrders()`
`bool CTrade::CloseAllMarketOrders()`

Closes all orders of the specified type that match the magic number set by the `SetMagicNumber()` function. Return `true` if all orders were closed successfully, `false` otherwise.

`bool CTrade::DeletePendingOrder(int pTicket, color pArrow = clrRed)`

Deletes the specified pending order. Return `true` if the order was deleted, `false` otherwise.

- `pTicket` - The ticket number of the order to delete.
- `pArrow` - The arrow color (optional).

`bool CTrade::DeleteAllBuyStopOrders()`
`bool CTrade::DeleteAllSellStopOrders()`
`bool CTrade::DeleteAllBuyLimitOrders()`
`bool CTrade::DeleteAllSellLimitOrders()`

`bool CTrade::DeleteAllPendingOrders()`

Deletes all pending orders of the specified type that match the magic number set by the `SetMagicNumber()` function.

`static void CTrade::SetMagicNumber(int pMagic)`

Sets the magic number to be applied to all orders opened with this expert advisor.

- `pMagic` - The magic number to use.

`static int CTrade::GetMagicNumber()`

Returns the magic number that was set by the `SetMagicNumber()` function.

`static void CTrade::SetSlippage(int pSlippage);`

Sets the slippage to be used when placing market orders on instant execution brokers.

- `pSlippage` - The slippage to use.

Non-Class Functions

```
bool ModifyOrder(int pTicket, double pPrice, double pStop = 0, double pProfit = 0,
       datetime pExpiration = 0, color pArrow = clrOrange)
```

Modifies a single order. If the order price, stop loss, take profit or expiration will not be modified, you must pass the current value to this function. Returns `true` if the order was modified successfully, `false` otherwise.

- `pTicket` - The ticket number of the order to modify.
- `pPrice` - The new or current pending order price.
- `pStop` - The new or current stop loss price.
- `pProfit` - The new or current take profit price.
- `pExpiration` - The new or current expiration time, for pending orders.
- `pArrow` - The arrow color (optional).

`bool ModifyStopsByPoints(int pTicket, int pStopPoints, int pProfitPoints = 0, int pMinPoints = 10)`

Modifies the stop loss and/or take profit for the specified order, using the specified stop distance in points. The stop will be automatically adjusted if invalid.

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

- `pStopPoints` - The new stop loss distance in points.
- `pProfitPoints` - The new take profit distance in points.
- `pMinPoints` - The minimum stop distance, in case the stop loss or take profit is automatically adjusted (optional).

```
bool ModifyStopsByPrice(int pTicket, double pStopPrice, double pProfitPrice = 0,
       int pMinPoints = 10)
```

Modifies the stop loss and/or take profit for the specified order, using the specified prices. The stop will be automatically adjusted if invalid.

- `pTicket` - The ticket number of the order to modify.
- `pStopPrice` - The new stop loss price.
- `pProfitPrice` - The new take profit price.
- `pMinPoints` - The minimum stop distance, in case the stop loss or take profit is automatically adjusted (optional).

```
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.

CCount Class

`int CCount::Buy()`
`int CCount::Sell()`
`int CCount::BuyStop()`
`int CCount::SellStop()`
`int CCount::BuyLimit()`
`int CCount::SellLimit()`
`int CCount::TotalMarket()`
`int CCount::TotalPending()`
`int CCount::TotalOrders()`

Returns a count of all open orders of the specified type that match the magic number set by the `SetMagicNumber()` function in the `CTrade` class. The `TotalMarket()` function returns a count of all market orders (buy and sell), `TotalPending()` returns a count of all pending orders, and `TotalOrders()` returns a count of all orders, market and pending.

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.

TrailingStop.mqh

`bool TrailingStop(string pTicket, int pTrailPoints, int pMinProfit = 0, int pStep = 10)`
`bool TrailingStop(string pTicket, double pTrailPrice, int pMinProfit = 0, int pStep = 10)`

Applies a trailing stop to a single order. The first variant of the function takes a trailing stop value in points, and the second variant takes a trailing stop price. Returns a value of `true` if the trailing stop was adjusted, `false` otherwise.

- `pTicket` - The ticket number of the order to adjust the trailing stop for.
- `pTrailPoints` - The trailing stop distance in points.
- `pTrailPrice` - The 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).

`void TrailingStopAll(int pTrailPoints, int pMinProfit = 0, int pStep = 10)`
`void TrailingStopAll(double pTrailPrice, int pMinProfit = 0, int pStep = 10)`

Applies a trailing stop to all open market orders. The first variant of the function takes a trailing stop value in points, and the second variant takes a trailing stop price.

- `pTrailPoints` - The trailing stop distance in points.
- `pTrailPrice` - The 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 BreakEvenStop(int pTicket, int pMinProfit, int pLockProfit = 0)`

Applies a break even stop for a single order. The stop loss is moved 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.

- `pTicket` - The ticket number of the order to adjust the break even stop for.
- `pMinProfit` - 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).

`void BreakEvenStopAll(int pMinProfit, int pLockProfit = 0)`

Applies a break even stop to all open market orders.

- `pMinProfit` - 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.

`datetime CTimer::GetStartTime()`
`datetime CTimer::GetEndTime()`

Returns the current start time or end time 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

`void CIndicator::Init(string pSymbol,int pTimeFrame)`

Sets the symbol and timeframe for the current indicator. This function is called from the constructor of all classes derived from the `CIndicator` class.

- `pSymbol` - The symbol to use for the current indicator.
- `pTimeFrame` - The timeframe to use for the current indicator.

CiMA class

```
void CiMA::CiMA(string pSymbol, int pTimeFrame, int pMaPeriod, int pMaShift, int pMaMethod,
       int pAppliedPrice)
```

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.

`double CiMA::Main(int pShift = 0)`

Returns the indicator value for the specified bar.

- `pShift` - The bar to retrieve the indicator value for. 0 is the current bar.

CiRSI class

`void CiRSI::CiRSI(string pSymbol, int pTimeFrame, int pPeriod, int pAppliedPrice)`

Initializes the RSI indicator.

- `pSymbol` - The symbol to use.
- `pTimeFrame` - The chart period to use.
- `pPeriod` - The calculation period for the indicator.
- `pAppliedPrice` - The price series to use.

`double CiRSI::Main(int pShift=0)`

Returns the indicator value for the specified bar.

- `pShift` - The bar to retrieve the indicator value for. 0 is the current bar.

CiStochastic class

```
void CiStochastic::CiStochastic(string pSymbol, int pTimeFrame, int pKPeriod, int pDPeriod,
       int pSlowing, int pMethod, int pAppliedPrice)
```

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.
- `pMethod` - The calculation method of the indicator.
- `pAppliedPrice` - The price series to use.

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

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

- `pShift` - The bar to retrieve the indicator value for. 0 is the current bar.

CiBollinger class

```
void CiBands::CiBands(string pSymbol, int pTimeFrame, int pPeriod, double pDeviation, int pShift,
       int pAppliedPrice)
```

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.
- `pDeviation` - The standard deviation of the outer bands.
- `pShift` - The forward or backward shift of the indicator.
- `pAppliedPrice` - The price series to use.

```
double CiBollinger::Lower(int pShift = 0)
double CiBands::Main(int pShift = 0)
double CiBollinger::Upper(int pShift = 0)
```

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

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

CiMACD class

```
void CiMACD::CiMACD(string pSymbol, int pTimeFrame, int pFastEmaPeriod, int pSlowEmaPeriod,
       int pSignalPeriod, int pAppliedPrice)
```

Initializes the MACD indicator.

- `pSymbol` - The symbol to use.
- `pTimeFrame` - The chart period to use.
- `pFastEmaPeriod` - The calculation period to use for the histogram.
- `pSlowEmaPeriod` - The calculation period to use for the histogram.
- `pSignalPeriod` - The calculation period for the signal line.
- `pAppliedPrice` - The price series to use.

`double CiMACD::Main(int pShift = 0)`
`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.