MQL5

FQInclude

Documentazione

@fedetrade

FEDERICO QUINTIERI

FQInclude Librerie
Documentazione

First edition

This book was professionally typeset on Reedsy.
Find out more at reedsy.com

Contents

Preface

Tutti gli include che ho codificato sono nella cartella FQInclude.

Per accedere facilmente a tutte le classi ho creato un include di nome Main.mqh.

Basterà richiamare questo per ottenere accesso a tutte le classi.

#include <FQInclude\Main.mqh>

I

CInfo

Questa parte fornisce una spiegazione chiara e dettagliata di ciascun metodo della classe CInfo.

1

Strutture

```
struct DealInfo
{
   ulong ticket;
   string symbol;
   ENUM_DEAL_ENTRY tipo_entrata;
   ENUM_DEAL_TYPE tipo_deal;
   datetime time;
   long magic;
   double volume;
   double price;
   double profit;
   double fee;
   double commission;
   double swap;
   double take;
   double stop;
};
```

```
struct PositionInfo
{
   ulong ticket;
   string symbol;
   datetime time;
   ENUM_POSITION_TYPE tipo;
   long magic;
   double volume;
   double price_open;
   double stop;
   double take;
   double swap;
   double profit;
};
```

```
struct OrderInfo
{
   ulong ticket;
   string symbol;
   datetime time_inserimento;
   datetime time_cancellazione;
   ENUM_ORDER_TYPE tipo;
   long magic;
   double volume;
   double stop;
   double take;
};
```

2

NuovaCandela

```
bool NuovaCandela()
```

Ritorna true se sul grafico è stata generata una nuova candela altrimenti false.

Uso tipico: sincronizzare logiche di trading all'apertura di ogni candela.

3

Pips

```
double Pips()
```

Calcola il valore di un pip per il simbolo corrente.

Return: valore di 1 pip (double).
Se il simbolo è Forex con Digits >= 3, ritorna Point()*10.
Altrimenti ritorna Point().

4

CiSonoPosizioni

```
bool CiSonoPosizioni(int magic, string symbol)
bool CiSonoPosizioni(int magic, string symbol,
ENUM_POSITION_TYPE tipo)
```

Verifica se esistono posizioni aperte con determinate condizioni.
Parametri:

- magic → magic number da controllare.
- symbol → simbolo dello strumento.
- tipo (opzionale) → tipo di posizione (POSITION_TYPE_BUY o POSITION_TYPE_SELL).

Return: true se trovata almeno una posizione, false altrimenti.

5

CiSonoOrdini

```
bool CiSonoOrdini(int magic, string symbol)
bool CiSonoOrdini(int magic, string symbol,
ENUM_ORDER_TYPE tipo)
```

Controlla se ci sono ordini pendenti con determinate condizioni.

Parametri:

- magic → magic number.
- symbol → simbolo dello strumento.
- tipo (opzionale) → tipo di ordine (ORDER_TYPE_BUY_LIMIT, ORDER_TYPE_SELL_STOP, ecc.).

Return: true se trovato almeno un ordine valido, false altrimenti.

6

InfoUltimoDeal

```
DealInfo InfoUltimoDeal(string symbol, int magic)
```

Restituisce le informazioni sull'ultimo deal eseguito per un determinato symbol e magic.

Parametri:

- symbol → simbolo dello strumento.
- magic → magic number.

Return: struttura DealInfo con dettagli (ticket, volume, prezzo, profitto, commissioni, ecc.).

7

InfoUltimaPosizione

```
PositionInfo InfoUltimaPosizione(string symbol, int
magic)
```

Ritorna i dettagli dell'ultima posizione aperta.
Parametri:

- symbol → simbolo dello strumento.
- magic → magic number.

Return: struttura PositionInfo con dati (ticket, tipo, volume, prezzo, SL/TP, profitto, ecc.).

8

InfoUltimoOrdine

```
OrderInfo InfoUltimoOrdine(string symbol, int magic)
```

Ritorna i dettagli dell'ultimo ordine pendente registrato.
Parametri:

- symbol → simbolo dello strumento.
- magic → magic number.

Return: struttura OrderInfo con dati (ticket, volume, SL/TP, tempo inserimento, ecc.).

9

RisultatoDealsGiornalieri

```
double RisultatoDealsGiornalieri(int magic, string
symbol)
```

Calcola il risultato (profitto/perdita) dei deal chiusi nella giornata corrente.

Parametri:

- symbol → simbolo dello strumento.
- magic → magic number.

Return: somma di profitto + commissioni + swap.

10

RisultatoPosizioniAperte

```
double RisultatoPosizioniAperte(int magic, string
symbol)
```

Calcola il risultato attuale delle posizioni aperte.
Parametri:

- symbol → simbolo dello strumento.
- magic → magic number.

Return: somma di profitto + swap sulle posizioni aperte.

11

RisultatoGiornaliero

```
double RisultatoGiornaliero(int magic, string symbol)
```

Calcola il risultato complessivo della giornata:
RisultatoDealsGiornalieri + RisultatoPosizioniAperte.
Parametri:

- symbol → simbolo dello strumento.
- magic → magic number.

Return: profitto/perdita totale del giorno.

12

Massimo

```
double Massimo(int candele_totali, int candela_inizio)
```

Restituisce il massimo prezzo (High) in un range di candele.
Parametri:

- candele_totali → numero di candele da analizzare.
- candela_inizio → indice della candela da cui partire.

Return: valore massimo.

13

Minimo

```
double Minimo(int candele_totali, int candela_inizio)
```

Restituisce il minimo prezzo (Low) in un range di candele.
Parametri:

- candele_totali → numero di candele da analizzare.
- candela_inizio → indice della candela da cui partire.

Return: valore minimo.

14

Ask

```
double Ask()
```

Ritorna il prezzo **Ask** corrente del simbolo.

- **Return**: prezzo Ask (double).

15

Bid

```
double Bid()
```

Ritorna il prezzo **Bid** corrente del simbolo.

- **Return**: prezzo Bid (double).

II

CDatabase

Questa parte fornisce una spiegazione chiara e dettagliata di ciascun metodo della classe CDatabase.

16

ApriDatabase

```
int ApriDatabase(string filename)
```

Apre o crea un database nella cartella common.

Parametri:

- filename → Nome del file del database.

Ritorno:

- Handle del database se l'apertura/creazione ha successo.
- INVALID_HANDLE se fallisce (con log dell'errore).

17

ChiudiDatabase

```
void ChiudiDatabase(int database_handle)
```

Chiude il database specificato.

Parametri:

- database_handle → Handle del database da chiudere.

18

CreaTabella

```
bool CreaTabella(int database_handle, string
table_name, string query)
```

Crea una tabella nel database se non esiste già.

Parametri:

- database_handle → Handle del database.
- table_name → Nome della tabella.
- query → Query SQL di creazione.

Ritorno:

- true se la tabella viene creata o se esiste già.
- false se la creazione fallisce.

19

CancellaTabella

```
bool CancellaTabella(int database_handle, string
table_name)
```

Cancella una tabella dal database.

Parametri:

- database_handle → Handle del database.
- table_name → Nome della tabella da eliminare.

Ritorno:

- true se la cancellazione ha successo.
- false se fallisce.

20

InserisciQuery

```
bool InserisciQuery(int database_handle, string
query_inserimento)
```

Inserisce dati nel database usando una query SQL.
Gestisce transazioni con BEGIN, COMMIT e ROLLBACK in caso di errore.

Parametri:

- database_handle → Handle del database.
- query_inserimento → Query di inserimento (INSERT).

Ritorno:

- true se l'inserimento ha successo.
- false in caso di errore.

21

RitornaInteger

```
int RitornaInteger(int database_handle, string
query_selezione)
```

Esegue una query SELECT e ritorna l'ultimo valore intero trovato.

Parametri:

- database_handle → Handle del database.
- query_selezione → Query SQL (SELECT ...).

Ritorno:

- Ultimo valore int trovato.
- 0 se fallisce o non ci sono record.

22

RitornaStringa

```
string RitornaStringa(int database_handle, string
query_selezione)
```

Esegue una query SELECT e ritorna l'ultima stringa trovata.

Parametri:

- database_handle → Handle del database.
- query_selezione → Query SQL (SELECT ...).

Ritorno:

- Ultimo valore string trovato.
- Stringa vuota "" se fallisce.

23

RitornaDouble

```
double RitornaDouble(int database_handle, string
query_selezione)
```

Esegue una query SELECT e ritorna l'ultimo valore double trovato.

Parametri:

- database_handle → Handle del database.
- query_selezione → Query SQL (SELECT ...).

Ritorno:

- Ultimo valore double trovato.
- 0.0 se fallisce o non ci sono record.

24

RitornaLong

```
long RitornaLong(int database_handle, string
query_selezione)
```

Esegue una query SELECT e ritorna l'ultimo valore long trovato.

Parametri:

- database_handle → Handle del database.
- query_selezione → Query SQL (SELECT ...).

Ritorno:

- Ultimo valore long trovato.
- 0 se fallisce o non ci sono record.

25

RitornaSomma

```
double RitornaSomma(int database_handle, string
query_selezione)
```

Esegue una query SELECT e somma tutti i valori numerici restituiti.

Parametri:

- database_handle → Handle del database.
- query_selezione → Query SQL (SELECT ...).

Ritorno:

- Somma di tutti i valori double.
- 0.0 se fallisce o non ci sono risultati.

III

CAndamento

Questa parte fornisce una spiegazione chiara e dettagliata di ciascun metodo della classe CAndamento.

26

Strutture

```
struct DealData
{
   ulong ticketto;
   datetime orario;
   ENUM_DEAL_TYPE tipo;
   double prezzo;
   double volume;
   double commissione;
   double swap;
   double fee;
   double profitto;
   double take;
   double stop;
};
```

```
struct TableInfo
{
   string name;
   string query;
```

```
};
```

27

InitDatabase

```
void InitDatabase(int magic, string nome_ea)
```

Inizializza il database per l'Expert Advisor.

Parametri:

- magic → Numero identificativo univoco dell'EA.
- nome_ea → Nome dell'Expert Advisor.

Funzionamento:

- Apre/crea un database con nome nome_ea_magic.
- In modalità *tester* elimina le tabelle Daily e Calcoli per ricrearle da zero.
- Crea le tabelle necessarie (Daily e Calcoli) se non esistono.

28

UpdateDatabase

```
void UpdateDatabase(int magic, string nome_ea, int
lungo_periodo_giorni, int breve_periodo_giorni)
```

Aggiorna i dati giornalieri e calcola le medie sul database.

Parametri:

- magic → Numero identificativo univoco dell'EA.
- nome_ea → Nome dell'Expert Advisor.
- lungo_periodo_giorni → Numero di giorni considerati per il periodo lungo.
- breve_periodo_giorni → Numero di giorni considerati per il periodo breve.

Funzionamento:

- Apre il database associato all'EA (nome_ea_magic).

- Calcola il risultato giornaliero e lo inserisce nella tabella Daily se non è già presente.
- Genera e inserisce i valori di calcolo statistico nella tabella Calcoli tramite la funzione QueryCalcolo.
- Chiude la connessione al database.

IV

CNotizie

Questa parte fornisce una spiegazione chiara e dettagliata di ciascun metodo della classe CNotizie.

29

Strutture

```
struct InfoNotizia
{
   datetime data;
   string nome;
   double attuale;
   double previsto;
   double precedente;
   string attuale_previsto;
   string attuale_precedente;
};
```

30

Enumerazioni

```
enum ENUM_VALUTA
{
   AUD, //Dollaro autraliano
   BRL, //Real brasiliano
   CAD, //Dollaro canadese
   CHF, //Franco svizzero
   CNT, //Yuan cinese
   EUR, //Euro
   GBP, //Sterlina
   HKD, //Dollaro di Hong Kong
   INR, //Rupia indiana
   JPY, //Yen giapponese
   KRW, //Won sudcoreano
   MXN, //Peso messicano
   NOK, //Corona norvegese
   NZD, //Dollaro neozelandese
   SEK, //Corona svedese
   SGD, //Dollaro di Singapore
   USD, //Dollari americani
   ZAR //Rand sudafricano
};
```

31

InitDatabase

```
void InitDatabase(string valuta)
```

Input:

- valuta: codice valuta (es. “USD”, “EUR”)

Output: Nessuno

Funzione:

- Crea (se non esiste) un database Notizie_<valuta>.sqlite.
- All’interno crea le tabelle: CPI, Interest, NFP, GDP.

32

UpdateDatabase

```
void UpdateDatabase(string valuta, datetime from)
```

Input:

- valuta: codice valuta
- from: data/ora di inizio ricerca eventi

Output: Nessuno

Funzione:

- Legge dal calendario economico (funzioni CalendarValueHistory, CalendarEventById, CalendarCountryById).
- Filtra gli eventi per la valuta richiesta.
- Inserisce nel DB i valori Attuale, Previsto, Precedente + variazioni percentuali.

33

UltimaNotizia

```
InfoNotizia UltimaNotizia(int database_handle, string
table_name)
```

Input:

- database_handle: handle del DB aperto
- table_name: nome della tabella (“CPI”, “Interest”, “NFP”, “GDP”)

Output: Struttura InfoNotizia con:

- data, nome, attuale, previsto, precedente, attuale_previsto, attuale_precedente

Funzione:

- Recupera l’ultima notizia disponibile (con Data <= Time-

Current()).

- Restituisce i valori convertiti in tipi corretti (datetime, double).

34

Variazione

```
string Variazione(string valuta, string tabella, int
ore, string colonna, int valoreBUY, int valoreSELL)
```

Input:

- valuta: codice valuta
- tabella: nome tabella (“CPI”, “Interest”, “NFP”, “GDP”)
- ore: finestra temporale dopo la notizia in cui verificare variazione
- colonna: “previsto” o “precedente”
- valoreBUY: soglia percentuale positiva
- valoreSELL: soglia percentuale negativa

Output:

- “positivo”, “negativo”, “uguale” o “” (se impossibile calcolare)

Funzione:

- Controlla se l'attuale è superiore/inferiore alle soglie rispetto al previsto o precedente.
- Restituisce la direzione della variazione.

V

CGestione

Questa parte fornisce una spiegazione chiara e dettagliata di ciascun metodo della classe CGestione.

35

ChiudiPosizioni (Tutte)

```
void ChiudiPosizioni(string simbolo, int magic)
```

Chiude tutte le posizioni aperte su un determinato simbolo e con uno specifico **magic number**.

Parametri:

- simbolo → Simbolo del mercato (es. "EURUSD").
- magic → Magic number che identifica le operazioni aperte dall'EA.

Funzionamento:

- Scorre tutte le posizioni aperte.
- Se la posizione appartiene al simbolo e al magic number indicato → viene chiusa.

36

ChiudiPosizioni (Filtrati)

```
void ChiudiPosizioni(ENUM_POSITION_TYPE
tipo_posizione, string simbolo, int magic)
```

Chiude le posizioni aperte di un **tipo specifico** (BUY o SELL) su un simbolo e magic number.

Parametri:

- tipo_posizione → Tipo posizione (POSITION_TYPE_BUY o POSITION_TYPE_SELL).
- simbolo → Simbolo del mercato.
- magic → Magic number dell'EA.

Funzionamento:

- Scorre tutte le posizioni aperte.
- Se la posizione corrisponde al tipo, al simbolo e al magic

number → viene chiusa.

37

CancellaPendenti (Tutti)

```
void CancellaPendenti(int magic, string simbolo)
```

Elimina tutti gli **ordini pendenti** per un simbolo e un magic number.

Parametri:

- magic → Magic number dell'EA.
- simbolo → Simbolo del mercato.

Funzionamento:

- Scorre tutti gli ordini pendenti.
- Se l'ordine appartiene al simbolo e al magic number indicato → viene cancellato.

38

CancellaPendenti (Filtrati)

```
void CancellaPendenti(ENUM_ORDER_TYPE tipo_pendente,
int magic, string simbolo)
```

Elimina solo gli ordini pendenti di un **tipo specifico** (BUY_LIMIT, SELL_LIMIT, BUY_STOP, SELL_STOP).

Parametri:

- tipo_pendente → Tipo di ordine pendente da eliminare.
- magic → Magic number dell'EA.
- simbolo → Simbolo del mercato.

Funzionamento:

- Scorre tutti gli ordini pendenti.
- Se l'ordine corrisponde al tipo, al simbolo e al magic number → viene cancellato.

39

Azzeramento

```
void Azzeramento(int magic, string simbolo, double
puntiPerAzzerare)
```

Aggiorna dinamicamente lo **stop loss** portandolo al livello di apertura (break-even) quando il prezzo si muove a favore di un certo numero di punti.

Parametri:

- magic → Magic number dell'EA.
- simbolo → Simbolo del mercato.
- puntiPerAzzerare → Numero di punti di profitto da raggiungere prima di spostare lo stop loss a break-even.

Funzionamento:

- Scorre le posizioni aperte sul simbolo con il magic indicato.

- Se la posizione è **BUY** e il prezzo Bid raggiunge openprice + puntiPerAzzerare → lo stop loss viene spostato sopra il prezzo di apertura.
- Se la posizione è **SELL** e il prezzo Ask raggiunge openprice – puntiPerAzzerare → lo stop loss viene spostato sotto il prezzo di apertura.
- Viene applicato un margine minimo di sicurezza di **10 punti** per evitare modifiche troppo frequenti.

VI

CRischio

Questa parte fornisce una spiegazione chiara e dettagliata di ciascun metodo della classe CRischio.

40

Compounding

```
double Compounding(double Lotti_Iniziali)
```

Calcola il nuovo lottaggio in base al **compounding** del bilancio.

Parametri:

- Lotti_Iniziali → il lottaggio di partenza.

Funzionamento:

- Calcola il coefficiente = Balance Attuale / Balance Iniziale.
- Moltiplica i lotti iniziali per il coefficiente.
- Restituisce il nuovo valore di lotti (arrotondato a 2 decimali).

41

RischioinDenaro

```
double RischioinDenaro(double denaro, double
range_stop)
```

Calcola il numero di lotti da aprire rischiando un importo fisso in denaro.

Parametri:

- denaro → capitale in denaro che si vuole rischiare.
- range_stop → distanza dello stop loss in punti.

Funzionamento:

- Usa la funzione interna CalcolaLottaggio per trasformare il rischio monetario in lotti.
- Restituisce il numero di lotti calcolato.

42

RischioinPercentuale

```
double RischioinPercentuale(double percentuale,
double range_stop)
```

Calcola il numero di lotti da aprire rischiando una percentuale del saldo.

Parametri:

- percentuale → % del balance da rischiare (es. 2 = 2%).
- range_stop → distanza dello stop loss in punti.

Funzionamento:

- Converte la percentuale in valore monetario (balance * percentuale / 100).
- Calcola il numero di lotti tramite CalcolaLottaggio.
- Restituisce il risultato.

43

CalcoloLottiKellyFormula

```
double CalcoloLottiKellyFormula(double
Probablita_Successo, double Rischio_Rendimento,
double grandezza_stop)
```

Calcola i lotti ottimali usando la **formula di Kelly**.

Parametri:

- Probablita_Successo → probabilità stimata di successo in percentuale.
- Rischio_Rendimento → rapporto rischio/rendimento previsto.
- grandezza_stop → dimensione dello stop loss in punti.

Funzionamento:

- Converte la probabilità in valore decimale.

- Calcola la percentuale da rischiare con la formula di Kelly.
- Converte tale rischio in lotti tramite CalcolaLottaggio.
- Applica un fattore di sicurezza: restituisce solo il **30% del valore calcolato**.

VII

CNotizieLive

Questa parte fornisce una spiegazione chiara e dettagliata di ciascun metodo della classe CNotizieLive.

44

Strutture

```
struct notizia
{
   ulong id_valore;
   ulong event_id;
   datetime time;
   datetime period;
   int revision;
   long actual_value;
   long prev_value;
   long revised_prev_value;
   long forecast_value;
   ENUM_CALENDAR_EVENT_IMPACT impact_type;
   ulong id_evento;
   ENUM_CALENDAR_EVENT_TYPE type;
   ENUM_CALENDAR_EVENT_SECTOR sector;
   ENUM_CALENDAR_EVENT_FREQUENCY frequency;
   ENUM_CALENDAR_EVENT_TIMEMODE time_mode;
   ulong country_id;
   ENUM_CALENDAR_EVENT_UNIT unit;
   ENUM_CALENDAR_EVENT_IMPORTANCE importance;
   ENUM_CALENDAR_EVENT_MULTIPLIER multiplier;
```

```
    uint digits;
    string source_url;
    string event_code;
    string name;
    ulong id_paese;
    string name_paese;
    string code;
    string currency;
    string currency_symbol;
    string url_name;
};
```

id_valore: Identificativo univoco associato al valore dell'indicatore economico.

event_id: Identificativo univoco associato all'evento.

time: Data e ora dell'evento.

period: Periodo di riferimento per il report degli eventi.

revision: Numero di revisione dell'indicatore rispetto al periodo del report.

actual_value: Valore attuale dell'indicatore economico.

prev_value: Valore precedente dell'indicatore economico.

revised_prev_value: Valore precedente rivisto dell'indicatore economico.

forecast_value: Valore previsto dell'indicatore economico.

impact_type: Tipo di impatto potenziale sull'indicatore economico.

id_evento: ID dell'evento.

type: Tipo di evento (enumerazione ENUM_CALENDAR_EVENT_TYPE).

sector: Settore a cui è collegato l'evento.

frequency: Frequenza degli eventi.

time_mode: Modalità oraria dell'evento.

country_id: ID del Paese.

unit: Unità di misura del valore dell'indicatore economico.

importance: Importanza dell'evento.

multiplier: Moltiplicatore del valore dell'indicatore economico.

digits: Numero di posizioni decimali per il valore.

source_url: URL della fonte dell'evento.

event_code: Codice univoco dell'evento.

name: Nome dell'evento nella lingua del terminale corrente.

id_paese: ID del Paese secondo ISO 3166-1.

name_paese: Nome del Paese nella lingua del terminale corrente.

code: Codice del Paese (ISO 3166-1 alpha-2).

currency: Codice della valuta del Paese.

currency_symbol: Simbolo della valuta del Paese.

url_name: Nome del Paese utilizzato nell'URL di mql5.com.

45

News

```
void AllNews(datetime from, datetime to, notizia
&result[])
```

Recupera tutte le notizie economiche dallo storico del calendario economico MQL5 in un intervallo temporale definito e le inserisce in un **array di strutture notizia**.

Parametri:

- from → data/ora di inizio periodo da cui prelevare le notizie.
- to → data/ora di fine periodo.
- result[] → array di strutture notizia passato per riferimento che verrà popolato con i dati degli eventi.

Funzionamento:

1. Pulisce l'array **result**.

2. Chiama **CalendarValueHistory** per recuperare delle notizie nell'intervallo indicato.
3. Per ogni notizia trovata:
   Recupera i dettagli dell'evento con **CalendarEventByI**
   Recupera i dati del paese con **CalendarCountryById**
   Popola la struttura **notizia** con tutte le informazioni disponibili (valori, evento, paese)
4. In caso di errori (nessuna notizia, CalendarEventById o CalendarCountryById falliti), stampa un messaggio di log con Print.

Ritorno:

- Non ritorna un valore diretto (è void), ma popola l'array result con tutte le notizie dell'intervallo richiesto.
- Se non ci sono notizie, result rimane vuoto.

Globalmente dovrai creare un array di tipo struttura **notizia.**

Questo array verrà passato per riferimento al metodo **AllNews** che lo popolerà con oggetti di tipo notizia.