```
//+------------------------------------------------------------------+
//|                                            C_TradeManagement.mqh |
//|                                  Copyright 2017, M Wilson.       |
//|                                https://www.algotrader.blog       |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, M Wlson"
#property link      "https://www.algotrader.blog"
#property version   "1.00"
#property strict
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
class C_TradeManagement
  {
public:
   //Constructor/Destructor
   C_TradeManagement();
   C_TradeManagement(const int intMagicNo=0):m_intMagicNo(intMagicNo){};
  ~C_TradeManagement();
   //Public Functions
   int CountLiveTrades();
   double CalculateAtRisk1Lot(const double dblSpotStopLossDifference);
   double RoundLotSize(const double dblLots);
   bool DoWeContinueAttemptingToTrade(int intErr);
protected:
   int m_intMagicNo;
private:
  };
//+------------------------------------------------------------------+
//| Constructors                                                     |
//+------------------------------------------------------------------+
C_TradeManagement::C_TradeManagement()
  {
  }
//+------------------------------------------------------------------+
//| Destructors                                                      |
//+------------------------------------------------------------------+
C_TradeManagement::~C_TradeManagement()
  {
  }
//+------------------------------------------------------------------+
int C_TradeManagement::CountLiveTrades()
{
   //Count the number of open trades
   int intRet=0;
   int intNoTrades=OrdersTotal();

   for(int i=0;i<intNoTrades;i++)
   {
      if(OrderSelect(i,SELECT_BY_POS,MODE_TRADES))
      {
         if(OrderMagicNumber()==this.m_intMagicNo && OrderSymbol()==Symbol())  intRet++;
      }
      else
         Print(__FILE__+" : "+__FUNCTION__," Error Selecting Trade Position ",i);
   }

   return intRet;
}
double C_TradeManagement::CalculateAtRisk1Lot(const double dblSpotStopLossDifference)
{
   double dblMarketTickSize=MarketInfo(Symbol(),MODE_TICKSIZE);
   double dblMarketTickValue=MarketInfo(Symbol(),MODE_TICKVALUE);

   double dblRet=(MathAbs(dblSpotStopLossDifference)/dblMarketTickSize)*
dblMarketTickValue;

   return dblRet;
}
double C_TradeManagement::RoundLotSize(const double dblLots)
{
```

```
   double dblLotStepSize=MarketInfo(Symbol(),MODE_LOTSTEP);
   double dblLotMax=MarketInfo(Symbol(),MODE_MAXLOT);
   double dblLotMin=MarketInfo(Symbol(),MODE_MINLOT);

   double dblLotRet=MathRound(dblLots/dblLotStepSize)*dblLotStepSize;
   if(dblLotRet>dblLots)  dblLotRet-=dblLotStepSize;

   if(dblLotRet>dblLotMax) dblLotRet=dblLotMax;
   if(dblLotRet<dblLotMin) dblLotRet=0;

   return dblLotRet;
}
bool C_TradeManagement::DoWeContinueAttemptingToTrade(int intErr)
{
   switch(intErr)
   {

//If there is no error or no result, assume success and stop attempting to impliment the tra
      case ERR_NO_ERROR :
      case ERR_NO_RESULT :
         return True;

// Wrong Price, or the Price has changed, then refresh the rates and attempt to trade again.
      case ERR_INVALID_PRICE:         // Wrong price
      case ERR_PRICE_CHANGED:         // Price changed
         RefreshRates();
         return True;

//If Off Quotes, sleep for up to 10 seconds, refreshing the rates, but returning true if we
      case ERR_OFF_QUOTES:
         for(int i=0;i<10000;i++)
         {
            if(RefreshRates()!=False)
            {
               return True;
            }
            Sleep(1);
         }
         return False;

//If Trade Context is busy, sleep for 500 milliseconds, refresh the rate and then try to tra
      case ERR_TRADE_CONTEXT_BUSY:         // The trade subsystem is busy
         Sleep(500);                       // Simple solution
         RefreshRates();                   // Renew data
         return True;                      // Error is overcomable
      // Critical errors - do not attempt to continue to trade, but log the error
      case ERR_COMMON_ERROR :          // Common error
      case ERR_OLD_VERSION :           // Old version of the client terminal
      case ERR_ACCOUNT_DISABLED:       // Account blocked
      case ERR_TRADE_DISABLED:         // Trading is prohibited
      default:          // Other variants
         return False;
   }
}
```