```mql4
//+------------------------------------------------------------------+
//|                                        MMU_Goto-Day_csv_exp.mq4 |
//|                                    Copyright 2021, Zyuna32246 |
//|                                          https://zyunafx.com/ |
//+------------------------------------------------------------------+
#property copyright "Copyright 2021, Zyuna32246"
#property link      "https://zyunafx.com/"
#property version   "1.00"
#property strict

//---パラメーター設定
input int jisa=7;                    //冬季での日本時間とMT4時間の誤差
input int pMinutes=0;                //基準となる時間（分）
input int pHour=10;                  //基準となる時間（時）
//集計範囲
input string p="----------------";  //集計範囲
input int pFuture=2;                 //基準から未来の時間（時）
input int pPast=10;                  //基準から過去の時間（時）

//---全体に関わる変数/配列
double Pips;             //価格をpipsに置き換える為の値を入れる変数
double rate[][1000];          //各分毎の価格を入れる配列
string rateTime[][1000];      //各分毎の「時:分」を入れる配列
int flags[];             //各分毎に価格を入れた回数を入れる配列
int nDays=0;             //計測した日数を入れる変数
int pNum=0;              //配列の数を入れる変数
string sDate,eDate;      //計測開始と終りの時間を入れる変数
string gDays[1000];   //
//+------------------------------------------------------------------+
int OnInit() {

   //価格をpipsに置き換える為の値を入れる
   Pips=AdjustPoint(_Symbol);

   //配列に必要な数を計算
   pNum=(pFuture+pPast)*60;     //計測範囲の時間を分にして入れる

   //配列のサイズを指定
   ArrayResize(rate,pNum+1);
   ArrayResize(rateTime,pNum+1);
   ArrayResize(flags,pNum+1);

   //計測開始年月日として取得
   sDate=StringConcatenate(Year(),IntegerToString(Month(),2,'0'),IntegerToString(Day(),2,'0'));

   return(INIT_SUCCEEDED);
}
//+------------------------------------------------------------------+
void OnTick() {

   //バー始値制御
   static int nBars=0;
   if(nBars!=Bars){

      //日本時間を取得
      datetime jTime=GetJapanTime(TimeCurrent(),jisa);
      int jD=TimeDay(jTime);
      int jH=TimeHour(jTime);
      int jM=TimeMinute(jTime);

      if(jD%5==0){                                  //ゴトー日確認
         if(jH==pHour+pFuture && jM==pMinutes){     //計測開始時分の確認

            double kOpen=Open[pFuture*60];          //基準時間の始値を入れる
            datetime cTime=Time[0];                 //計測開始の時間を入れる
            int Blank=0;                            //データがないローソク足計測用

            //計測する分数をループ処理
            for(int i=0;i<=pNum;i++){

               if(TimeMinute(cTime)==TimeMinute(Time[i-Blank])){  //計測時間と取得時間が同じか確認
                  rate[i][nDays]=(Open[i-Blank]-kOpen)/Pips;      //取得時間の価格と基準価格の差を取得。Pipsに変換。
                  //rate[i][nDays]=Open[i-Blank];                   //取得時間の始値を取得
                  flags[i]++;                                     //計測出来た時間の回数をカウント
               }else{                     //計測時間と取得時間が違う場合
                  rate[i][nDays]+=0;      //取得時間のデータが無かったので0を入れる
                  Blank++;                //データがない足の数をカウント
               }

               int tHour=TimeHour(jTime-60*i);      //計測開始時間から1分毎の時間（日本時間）
               //int tHour=TimeHour(TimeCurrent()-60*i);     //計測開始時間から1分毎の時間（サーバー時間）
               rateTime[i][nDays]=StringConcatenate(tHour,"時",TimeMinute(cTime),"分");   //計測時分を入れる
               cTime-=60;  //次の計測時間を1分戻す
            }
            //計測のゴトー日の年月日を入れる
            gDays[nDays]=StringSubstr((string)jTime,0,10);
            nDays++; //計測したゴトー日数をカウント

         }
      }
   }//nBars


   //ティック更新時のバー総数を記憶
   nBars=Bars;
}
//+------------------------------------------------------------------+
//ポジション調整
double AdjustPoint(string Currency){
   int SymbolDigits=(int)MarketInfo(Currency,MODE_DIGITS);
```

```
   switch(SymbolDigits){
      case 2:  return(0.01);     break;
      case 3:  return(0.01);     break;
      case 4:  return(0.0001);   break;
      case 5:  return(0.0001);   break;
   }
   return(0.0);
}
//+------------------------------------------------------------------+
//日本時間の取得（サマータイムを考慮）
datetime GetJapanTime(datetime CurrentTime,int offset,bool summerflag=true){

   if(summerflag){
      return(CurrentTime+PERIOD_H1*(offset-IsDST(CurrentTime))*60);
   }else
   if(!summerflag){
      return(CurrentTime+PERIOD_H1*offset*60);
   }

   return(0);
}
//+------------------------------------------------------------------+
//サマータイム判定
bool IsDST(datetime CurrentTime){
   bool DST;
   static datetime StartDate,EndDate;

   //最初の1回&年1回の処理制御
   static int nYear=0;
   if(nYear!=Year()){

   string CurrentYear = (string)(TimeYear(CurrentTime));
   StartDate = (datetime)(CurrentYear + ".3." + (string)(14 - TimeDayOfWeek((datetime)(CurrentYear + ".3.14"))));
   EndDate = (datetime)(CurrentYear + ".11." + (string)(7 - TimeDayOfWeek((datetime)(CurrentYear + ".11.7"))));

      nYear=Year();
   }


   if(CurrentTime>=StartDate && CurrentTime<EndDate){
      DST=true;   //ture=1 サマータイム
   }else{
      DST=false;  //false=0
   }
   return(DST);
}
//+------------------------------------------------------------------+
//csv出力
double OnTester(){

   //計測終了年月日として取得
   eDate=StringConcatenate(Year(),IntegerToString(Month(),2,'0'),IntegerToString(Day(),2,'0'));

   //ファイル名の頭
   string fName="Goto-Day_Exp";
   //ファイル名に（_開始年月日_終了年月日_基準時分.csv）を追加
   fName+=("_"+sDate+"_"+eDate+"_"+(string)pHour+IntegerToString(pMinutes,2,'0')+".csv");
   int fDelete=FileDelete(fName,0);
   int fOpen=FileOpen(fName,FILE_CSV | FILE_READ | FILE_WRITE,",");

   string subject;
   for(int i=0;i<nDays;i++){
      //if(i!=0)subject+=(","+gDays[i]+",価格");
      if(i!=0)subject+=(","+gDays[i]);
      else    subject=("時間"+gDays[i]);
   }
   FileWrite(fOpen,subject);
   FileSeek(fOpen,0,SEEK_END);

   string data;
   for(int i=pNum;i>=0;i--){

      for(int j=0;j<nDays;j++){
         //if(j!=0)data+=(","+rateTime[i][j]+","+(string)rate[i][j]);
         if(j!=0)data+=(","+(string)rate[i][j]);
         else    data=(rateTime[i][j]+","+(string)rate[i][j]);
      }

      FileWrite(fOpen,data);

   }
   FileClose(fOpen);
   return(true);
}
```