

```
#ifndef _SCSYMBOLDATA_H_
#define _SCSYMBOLDATA_H_
```

```
typedef uint32_t t_Volume;
```

```
#pragma pack(push, 8)
```

```
enum TickDirectionEnum
{ DOWN_TICK = -1
, NO_TICK_DIRECTION = 0
, UP_TICK = 1
};
```

```
// This structure is used in ACSIL studies. Therefore, back compatibility
// needs to be maintained.
```

```
struct s_SCBasicSymbolData
{
    float DailyOpen = 0;
    float DailyHigh = 0;
    float DailyLow = 0;
    unsigned int Old_DailyVolume = 0;
    int DailyNumberOfTrades = 0;
    float LastTradePrice = 0;
    uint32_t LastTradeVolume = 0;
    SCDatetime LastTradeDateTime;
    int TickDirection = NO_TICK_DIRECTION; // TickDirectionEnum
    int LastTradeAtSamePrice = 0;
    char LastTradeAtBidAsk = 0; // can be SC_TS_BID or SC_TS_ASK
    float Bid = 0;
    float Ask = 0;
    unsigned int AskQuantity = 0;
    unsigned int BidQuantity = 0;
    float CurrencyValuePerTick = 0;
    float SettlementPrice = 0;
    unsigned int OpenInterest = 0;
    unsigned int SharesOutstanding = 0;
    float EarningsPerShare = 0;
    float StrikePrice = 0;
    SCDatetime LastBidAskUpdateDateTime;
```

```
// This must correspond to the pricing format actually used after the
// price values are adjusted by the DisplayPriceMultiplier.
```

```
float TickSize = 0;
```

```
// Standard Sierra Chart price format code. This must correspond to the
// pricing format actually used after the price values are adjusted by
// the DisplayPriceMultiplier.
```

```
int PriceFormat = -1;
float ContractSize = 0;
float BuyRolloverInterest = 0;
float SellRolloverInterest = 0;
uint8_t TradeIndicator = 0;
```

```
int AccumulatedLastTradeVolume = 0;
SCDatetime LastTradingDateForFutures;
```

```
uint8_t TradingStatus = 0;
```

```
// The Date value (SCDatetime) that the current values in the
// BasicSymbolData object are for. This is going to be what is
// considered the trading day date. So this will be the following day in
// the case when the trading begins in the prior day at the beginning of
// the evening session.
```

```
SCDatetime TradingDayDate;
```

```
SCDateTimeMS LastMarketDepthUpdateDateTime;
```

```
float DisplayPriceMultiplier = 1.0f;
```

```
SCDateTime SettlementPriceDate;
```

```
SCDateTime DailyOpenPriceDate;
```

```
SCDateTime DailyHighPriceDate;
```

```
SCDateTime DailyLowPriceDate;
```

```
SCDateTime DailyVolumeDate;
```

```
int NumberOfTradesAtCurrentPrice = 0;
```

```
char VolumeValueFormat = 0;//Remove
```

```
float LowLimitPrice = 0;
```

```
float HighLimitPrice = 0;
```

```
uint64_t DailyVolume = 0;
```

```
SCDateTime RolloverDate;
```

```
struct s_ImbalanceData
```

```
{
```

```
    SCDateTime DateTimeUTC;
```

```
    double MatchingQuantity = 0;
```

```
    double NonMatchingQuantity = 0;
```

```
    double CurrentReferencePrice = 0;
```

```
    uint8_t ImbalanceDirection = 0;
```

```
    uint8_t CrossType = 0;
```

```
} m_ImbalanceData;
```

```
double DailyBidVolume = 0;
```

```
double DailyAskVolume = 0;
```

```
/*=====*/
```

```
s_SCBasicSymbolData()
```

```
{
```

```
}
```

```
void Clear()
```

```
{
```

```
    *this = s_SCBasicSymbolData();
```

```
}
```

```
/*=====*/
```

```
s_SCBasicSymbolData(const s_SCBasicSymbolData& Src)
```

```
{
```

```
    Copy(Src);
```

```
}
```

```
/*=====*/
```

```
s_SCBasicSymbolData& operator = (const s_SCBasicSymbolData& Src)
```

```
{
```

```
    Copy(Src);
```

```
    return *this;
```

```
}
```

```
/*=====*/
```

```
void Copy(const s_SCBasicSymbolData& Src)
```

```
{
```

```
    if (&Src == this)
```

```
        return;
```

```
    memcpy(this, &Src, sizeof(s_SCBasicSymbolData));
```

```
}
```

```
/*=====*/
```

```

bool operator == (const s_SCBasicSymbolData& That)
{
    return (memcmp(this, &That, sizeof(s_SCBasicSymbolData)) == 0);
}

/*=====*/
void MultiplyData(float Multiplier, bool InvertPrices)
{
    if (Multiplier != 1.0)
    {
        Ask *= Multiplier;
        Bid *= Multiplier;
        DailyHigh *= Multiplier;
        LastTradePrice *= Multiplier;
        DailyLow *= Multiplier;
        DailyOpen *= Multiplier;
        SettlementPrice *= Multiplier;
    }

    if (InvertPrices)
    {
        Ask = SafeInverse(Ask);
        Bid = SafeInverse(Bid);
        DailyHigh = SafeInverse(DailyHigh);
        LastTradePrice = SafeInverse(LastTradePrice);
        DailyLow = SafeInverse(DailyLow);
        DailyOpen = SafeInverse(DailyOpen);
        SettlementPrice = SafeInverse(SettlementPrice);
    }
}

float GetDailyPriceChange() const
{
    float DailyPriceChange = 0.0;
    if (SettlementPrice != 0.0)
    {
        DailyPriceChange = LastTradePrice - SettlementPrice;
    }
    return DailyPriceChange;
}

/*=====*/
static float SafeInverse(float Number)
{
    if (Number == 0.0f)
        return 0.0f;

    return 1.0f / Number;
}

};

#pragma pack(pop)

#endif

```