

ACSIL Interface Members - Historical Market Depth Data (c_ACSILDepthBars)

- [c_ACSILDepthBars](#)
- [c_ACSILDepthBars Methods](#)
 - [GetTickSize](#)
 - [PriceToTickIndex](#)
 - [TickIndexToPrice](#)
 - [NumBars](#)
 - [DepthDataExistsAt](#)
 - [GetBarHighestPriceTickIndex](#)
 - [GetBarLowestPriceTickIndex](#)
 - [GetNextHigherPriceTickIndex](#)
 - [GetMaxDominantSide](#)
 - [GetMaxQuantityForSide](#)
 - [GetMaxBidQuantity](#)
 - [GetMaxAskQuantity](#)
 - [GetMaxQuantity](#)
 - [GetLastDominantSide](#)
 - [GetLastQuantityForSide](#)
 - [GetLastBidQuantity](#)
 - [GetLastAskQuantity](#)
 - [GetLastQuantity](#)
- [Examples](#)
 - [Iterating Through All the Prices Within a Bar](#)

c_ACSILDepthBars

The c_ACSILDepthBars C++ class provides an interface for accessing historical market depth data bars loaded into a chart. Use the [Advanced Custom Study Interface and Language](#) functions [sc.GetMarketDepthBars\(\)](#) or [sc.GetMarketDepthBarsFromChart\(\)](#) to get an instance of this class.

Historical market depth data is stored at multiple price increments, as determined by the tick size for the symbol, for each bar in the chart. The different price increments are identified by their price tick index, where each adjacent price tick index has a price value difference exactly equal to the tick size on the chart.

For functions that take a **PriceTickIndex** index value, that can be determined from an actual float price value by using the [PriceToTickIndex](#) member function.

In order for there to be any historical market depth data to access, historical market depth data must be loaded into the chart and also should be recorded in real-time. To accomplish both of these, it is necessary to set [sc.MaintainHistoricalMarketDepthData](#) to TRUE in the study function. This normally will be done in the **sc.SetDefaults** code block.

For a study which uses this data, refer to [Market Depth Historical Graph](#) for details.

It is also supported to access the most recent market depth data. For complete documentation, refer to [Programmatically Accessing Historical and Current Market Depth Data](#). This is an alternative to access the current market depth data. You will need to access the current market depth data through these other methods whenever market depth data changes after a chart bar has completed but before there is a new trade for the next bar which has not yet been displayed because there has not yet been a trade for that new upcoming bar.

c_ACSILDepthBars Methods

GetTickSize

```
float GetTickSize();
```

Returns the tick size that was used for storing the historical market depth data.

PriceToTickIndex

```
int PriceToTickIndex(float Price);
```

Converts the given **Price** value into a price tick index. This is equivalent to the price divided by the tick size. Price tick indexes are used to discretely identify price levels in the historical market depth data.

TickIndexToPrice

```
float TickIndexToPrice(int PriceTickIndex);
```

Converts the given **PriceTickIndex** into the original price value. This is equivalent to the price tick index multiplied by the tick size. Price tick indexes are used to discretely identify price levels in the historical market depth data.

NumBars

```
int NumBars();
```

Returns the number of historical market depth data bars. This will generally match the number of bars in the chart, but may be less if there is less market depth data available. Bar indexes in the historical market depth data will correspond to bar indexes in the chart. Not every bar may contain historical market depth data.

DepthDataExistsAt

```
bool DepthDataExistsAt(int BarIndex);
```

Returns **true** if any historical market depth data exists within the bar at the given **BarIndex**. If the given **BarIndex** is outside the valid range of bars, this method will return **false**.

GetBarHighestPriceTickIndex

```
int GetBarHighestPriceTickIndex(int BarIndex);
```

Returns the highest price, as a price tick index, that has any known depth quantity within the bar at the given **BarIndex**. If no depth data exists within the bar at the given **BarIndex**, or the given **BarIndex** is outside the valid range of bars, this method will return **0**.

GetBarLowestPriceTickIndex

```
int GetBarLowestPriceTickIndex(int BarIndex);
```

Returns the lowest price, as a price tick index, that has any known depth quantity within the bar at the given **BarIndex**. If no depth data exists within the bar at the given **BarIndex**, or the given **BarIndex** is outside the valid range of bars, this method will return **0**.

GetNextHigherPriceTickIndex

```
bool GetNextHigherPriceTickIndex(int BarIndex, int& r_PriceTickIndex);
```

Increments the given **r_PriceTickIndex** up to the next price tick index that contains any known depth quantity within the bar at the given **BarIndex**. If the given **r_PriceTickIndex** is already at or beyond the highest price, or if no depth data exists within the bar at the given **BarIndex**, or the given **BarIndex** is outside the valid range of bars, this method will return **false**. Otherwise the method returns **true** to indicate that the given **r_PriceTickIndex** has been set to the next valid price tick index with data. The given **r_PriceTickIndex** may be incremented by more than one price level if there are price levels that do not contain any depth quantities.

GetMaxDominantSide

```
BuySellEnum GetMaxDominantSide(int BarIndex, int PriceTickIndex);
```

Returns the side (**BSE_BUY** for bid, **BSE_SELL** for ask) with the greatest maximum quantity within the bar at the given **BarIndex**, at the given **PriceTickIndex**. If both sides have equal maximum quantities, or no quantities are known, this method returns **BSE_UNDEFINED**.

GetMaxQuantityForSide

```
int GetMaxQuantityForSide(int BarIndex, int PriceTickIndex, BuySellEnum Side);
```

Returns the maximum quantity within the bar at the given **BarIndex**, at the given **PriceTickIndex**, on the given **Side** (**BSE_BUY** for bid, **BSE_SELL** for ask). If the maximum bid quantity is equal to the maximum ask quantity, and **BSE_UNDEFINED** is given for the **Side**, then the maximum bid quantity will be returned, since both sides are equal. This is to assist in the case that the dominant side is requested, but there is no dominant side. Otherwise, if any of these

parameters are outside the valid range of data, or there is no known quantity, this method returns **0**.

GetMaxBidQuantity

```
int GetMaxBidQuantity(int BarIndex, int PriceTickIndex);
```

Returns the maximum bid quantity within the bar at the given **BarIndex**, at the given **PriceTickIndex**. If any of these parameters are outside the valid range of data, or there is no known bid quantity, this method returns **0**.

GetMaxAskQuantity

```
int GetMaxAskQuantity(int BarIndex, int PriceTickIndex);
```

Returns the maximum ask quantity within the bar at the given **BarIndex**, at the given **PriceTickIndex**. If any of these parameters are outside the valid range of data, or there is no known ask quantity, this method returns **0**.

GetMaxQuantity

```
int GetMaxQuantity(int BarIndex, int PriceTickIndex);
```

Returns the maximum quantity, the greater of the maximum bid quantity or the maximum ask quantity, within the bar at the given **BarIndex**, at the given **PriceTickIndex**. If any of these parameters are outside the valid range of data, or there are no known quantities, this method returns **0**.

GetLastDominantSide

```
BuySellEnum GetLastDominantSide(int BarIndex, int PriceTickIndex);
```

Returns the side (**BSE_BUY** for bid, **BSE_SELL** for ask) with the greatest last quantity within the bar at the given **BarIndex**, at the given **PriceTickIndex**. If both sides have equal last quantities, or no quantities are known, this method returns **BSE_UNDEFINED**.

The [Market Depth Historical Graph](#) study simply defaults to the ask side in any case where the last bid quantity is not greater.

GetLastQuantityForSide

```
int GetLastQuantityForSide(int BarIndex, int PriceTickIndex, BuySellEnum Side);
```

Returns the last quantity for the bar at the given **BarIndex**, at the given **PriceTickIndex**, on the given **Side** (**BSE_BUY** for bid, **BSE_SELL** for ask). If the last bid quantity is equal to the last ask quantity, and **BSE_UNDEFINED** is given for the **Side**, then the last bid quantity will be returned, since both sides are equal. This is to assist in the case that the dominant side is requested, but there is no dominant side. Otherwise, if any of these parameters are outside the valid range of data, or there is no known quantity, this method returns **0**.

GetLastBidQuantity

```
int GetLastBidQuantity(int BarIndex, int PriceTickIndex);
```

Returns the last bid quantity for the bar at the given **BarIndex**, at the given **PriceTickIndex**. If any of these parameters are outside the valid range of data, or there is no known bid quantity, this method returns **0**.

GetLastAskQuantity

```
int GetLastAskQuantity(int BarIndex, int PriceTickIndex);
```

Returns the last ask quantity for the bar at the given **BarIndex**, at the given **PriceTickIndex**. If any of these parameters are outside the valid range of data, or there is no known ask quantity, this method returns **0**.

GetLastQuantity

```
int GetLastQuantity(int BarIndex, int PriceTickIndex);
```

Returns the last quantity, the greater of the last bid quantity or the last ask quantity, for the bar at the given **BarIndex**, at the given **PriceTickIndex**. If any of these parameters are outside the valid range of data, or there are no known quantities, this method returns **0**.

Examples

Iterating Through All the Prices Within a Bar

This code example demonstrates how to iterate through all of the price levels that contain market depth data within a single bar.

```
// Get access to the market depth bars in the chart the study instance is applied to.
c_ACSILDepthBars* p_DepthBars = sc.GetMarketDepthBars();
if (p_DepthBars == NULL)
    return;

// Do nothing if the bar at the current index has no data.
if (!p_DepthBars->DepthDataExistsAt(sc.Index))
    return;

// Iterate through each price index for the bar at the current index.
int PriceTickIndex = p_DepthBars->GetBarLowestPriceTickIndex(sc.Index);

do
{
    // Use PriceTickIndex to get the desired depth data at the current
    // price level. For example:
    const int MaxBidQuantityAtPriceTick =
        p_DepthBars->GetMaxBidQuantity(sc.Index, PriceTickIndex);
}
while (p_DepthBars->GetNextHigherPriceTickIndex(sc.Index, PriceTickIndex));
```

The [sc.GetMarketDepthBars\(\)](#) function is used for getting access to the [c_ACSILDepthBars](#)

object.

[sc.Index](#) is used to specify the bar index currently being processed.

Before attempting to get the first price tick index, the code first checks if any depth data exists in the bar using [DepthDataExistsAt\(\)](#).

[GetBarLowestPriceTickIndex\(\)](#) is used to get the first price tick index in the bar, and then [GetNextHigherPriceTickIndex\(\)](#) is used in a do-while loop to iterate through the rest of the price tick indexes in the bar.

A do-while loop is used so that the first price tick index is processed before moving on to the next price tick index.

*Last modified Monday, 10th January, 2022.