



## MFT C# API

<b>Author</b>	<b>Vinay Dwivedi</b>
<b>Version</b>	<b>Draft</b>
<b>Version Date</b>	<b>06-May-2024</b>
<b>Revision</b>	
<b>Reviewer</b>	<b>Abhay Tewari</b>

This document outlines important attributes, and methods exposed by MFT C# API. As this is an initial draft version, changes, and updates will be done on continue basis, and a final version will be market as version 1.0, and subsequent updates will be marked as 1.1 as so on.



Table of Contents

1. Security.....	8
1.1. Security Class.....	8
1.2. Basket Class .....	8
1.2.1. Basket Security class .....	8
1.2.1.1. Attributes .....	9
1.3. GetStrike function.....	9
1.3.1. Decimal GetStrike(int token, int index, int strikeGap, out String errorMessage).....	9
1.3.2. Decimal GetStrike(String symbol, int expiry, int index, int strikeGap, out String errorMessage) .....	10
1.3.3. Decimal GetStrike(String symbol, String expiry, int index, int strikeGap, out String errorMessage) .....	10
1.3.4. Decimal GetStrikeByParity(String symbol, String expiry, int index, int strikeGap, out String errorMessage) .....	11
1.3.5. Decimal GetStrikeByParity(String symbol, int expiry, int index, int strikeGap, out String errorMessage) .....	11
1.3.6. Decimal[] GetAllStrikesByParity(String symbol, String expiry, int[] indexList, int strikeGap, out String errorMessage) .....	12
1.3.7. Decimal[] GetAllStrikesByParity(String symbol, int expiry, int[] indexList, int strikeGap, out String errorMessage).....	12
1.3.8. Decimal[] GetAllStrikes(int token, int[] indexList, int strikeGap, out String errorMessage)....	12
1.3.9. Decimal[] GetAllStrikes(String symbol, Int expiry, int[] indexList, int strikeGap, out String errorMessage) .....	13
1.3.10. Decimal[] GetAllStrikes (String symbol, String expiry, int[] indexList, int strikeGap, out String errorMessage).....	13
1.3.11. Decimal[] GetAllStrikes(String symbol, int expiry, out String errorMessage) .....	14
1.3.12. Decimal[] GetAllStrikes(String symbol, String expiry, out String errorMessage) .....	14
1.3.13. Decimal[] GetAllStrikes(String symbol, int expiry, int strikeGap, out String errorMessage) .....	14
1.3.14. Decimal[] GetAllStrikes(String symbol, String expiry, out String errorMessage) .....	15
1.3.15. Decimal GetStrikeByPrice(String symbol, int expiry, Decimal price, int strikeGap, out String errorMessage).....	15
1.3.16. Decimal GetStrikeByPrice(String symbol, String expiry, Decimal price, int strikeGap, out String errorMessage).....	16
1.3.17. Decimal GetStrikeByPercentageGap(String symbol, int expiry, Decimal percentageGap, int strikeGap, out String errorMessage) .....	16



1.3.18.	Decimal GetStrikeByPercentageGap(String symbol, string expiry, Decimal percentageGap, int strikeGap, out String errorMessage).....	17
1.3.19.	Decimal[] GetStrikeByPercentageGap(String symbol, int expiry, Decimal[] percentageGapList, int strikeGap, out String errorMessage).....	17
1.3.20.	Decimal[] GetStrikeByPercentageGap(String symbol, string expiry, Decimal[] percentageGapList, int strikeGap, out String errorMessage).....	18
1.3.21.	List<Decimal> GetAllStrikeByExpiry(String symbol, int expiry, int strikeGap, out String errorMessage) .....	18
1.3.22.	List<Decimal> GetAllStrikeByExpiry(String symbol, String expiry, int strikeGap, out String errorMessage) .....	18
1.3.23.	Dictionary<String, List<KeyValuePair<Int32, Decimal>>> GetAllOptionsWithDeltaRange(String symbol, int expiry, Decimal deltaLowerLimit, Decimal deltaUpperLimit, int strikeGap, out String err) .....	19
1.3.24.	Dictionary<String, List<KeyValuePair<Int32, Decimal>>> GetAllOptionsWithDeltaRange(String symbol, string expiry, Decimal deltaLowerLimit, Decimal deltaUpperLimit, int strikeGap, out String err) .....	20
1.3.25.	Dictionary<String, List<KeyValuePair<Int32, Decimal>>> GetAllOptionsCloseToDelta(String symbol, int expiry, Decimal deltaThreshold, int strikeDivisor, Boolean bothSides, out String err) .....	21
1.3.26.	Dictionary<String, List<KeyValuePair<Int32, Decimal>>> GetAllOptionsCloseToDelta(String symbol, string expiry, Decimal deltaThreshold, int strikeDivisor, Boolean bothSides, out String err) .....	22
1.3.27.	GetAllOptionsWithPriceRange, & GetAllOptionsCloseToPrice .....	23
1.4.	<b>GetExpiry function</b> .....	23
1.4.1.	int GetCurrentExpiry(String symbol, String expiryType = null) .....	23
1.4.2.	int GetCurrentExpiry(String symbol, String expiryType = null) .....	23
1.5.	<b>GetSecurity function</b> .....	24
1.5.1.	Security GetSecurity(String symbol) .....	24
1.5.2.	Security GetSecurity(int token) .....	24
1.5.3.	Security GetSecurity(String symbol, String expiry).....	24
1.5.4.	Security GetSecurity(String symbol, int expiry, Decimal strike, String type) .....	24
1.5.5.	Security GetSecurity(String symbol, string expiry, Decimal strike, String type).....	25
1.6.	<b>GetLotSize function</b> .....	25
2.	<b>Subscription, Data &amp; Events</b> .....	26
2.1.	<b>StrategyBuildMode</b> .....	26
2.1.1.	API.....	26



2.1.2.	GUI .....	26
2.2.	<b>StrategyTradingMode()</b> .....	26
2.2.1.	SingleSecurity .....	26
2.2.2.	BASKET .....	26
2.2.3.	MultiSecurity .....	26
2.3.	<b>AddSecurity Method</b> .....	27
2.3.1.	Tuple<Int32, Int32> AddSecurity(Security singleSecurity, out String errorMessage) .....	27
2.3.1.1.	<b>Details</b> .....	27
2.3.1.2.	<b>Parameters</b> .....	28
2.3.2.	Tuple<Int64, Int32> AddBasket(BasketSecurity[] securityList, out String errorMessage)....	28
2.3.2.1.	<b>Details</b> .....	28
2.3.2.2.	<b>Parameters</b> .....	29
2.3.3.	AddMultiSecurity(BasketSecurity[] securityList, out String errorMessage) .....	29
2.3.3.1.	<b>Details</b> .....	29
2.3.3.2.	<b>Parameters</b> .....	29
2.3.4.	Tuple<Int64, Int32> AppendMultiSecurity(BasketSecurity[] securityList, out String errorMessage) .....	30
2.3.5.	Tuple<Int64, Int32> RemoveMultiSecurity(BasketSecurity[] securityList, out String errorMessage) .....	30
2.3.5.1.	<b>Details</b> .....	31
2.3.6.	Tuple<Int64, Int32> EditMultiSecurity(BasketSecurity[] securityList, out String errorMessage) .....	31
2.3.6.1.	<b>Details</b> .....	32
2.4.	<b>GetSubscribedSecurity Method</b> .....	32
2.4.1.	SubscribedSingleSecurity .....	32
2.4.2.	SubscribedBasket .....	32
2.4.3.	SubscribedMultiSecurities .....	32
2.5.	<b>Data</b> .....	33
2.5.1.	<b>MBP Data</b> .....	33
2.5.2.	OHLC Data.....	35
2.5.3.	Indicator Data .....	36
2.5.4.	BarType Enum.....	36
2.5.5.	Filter Data .....	36



<b>2.6.</b>	<b>Data Events &amp; Functions.</b>	37
<b>2.6.1.</b>	<b>MBP Data</b>	37
2.6.1.1.	Decimal LTP(int token)	37
2.6.1.2.	Decimal OPEN(int token)	37
2.6.1.3.	Decimal CLOSE(int token)	37
2.6.1.4.	MBP GetGlobalMBPTick(int token, out String errorMessage)	37
2.6.1.5.	MBP GetSubscribedMBPTick (int token, out String errorMessage)	38
<b>2.6.2.</b>	<b>OHLC, Indicator, &amp; Filter data</b>	38
<b>2.6.2.1.</b>	<b>Definition.</b>	38
2.6.2.2.	void OnBarUpdate	39
2.6.2.3.	void OnHistoryBarUpdate	39
2.6.2.4.	OHLCVBar GetDataSetSeriesBar(int token, int index, BARType type)	39
2.6.2.5.	OHLCVBar GetDataSetSeriesBar(int token, DateTime dateTime, BARType type)	40
2.6.2.6.	List<OHLCVBar> DataSetSeriesHistory(int token)	40
2.6.2.7.	List<OHLCVBar> DataSetSeriesLive(int token)	40
2.6.2.8.	List<OHLCVBar> DataSetSeriesEOD(int token)	40
2.6.2.9.	EarliestFirst version.	40
2.6.2.10.	IndicatorBar GetIndicatorBar(DateTime dateTime, BARType type, String name)	41
2.6.2.11.	IndicatorBar GetIndicatorBar(DateTime dateTime, BARType type, String name)	42
2.6.2.12.	List<IndicatorBar> IndicatorHistory(int token, String name)	42
2.6.2.13.	List<IndicatorBar> IndicatorLive(int token, String name)	42
2.6.2.14.	List<IndicatorBar> IndicatorEOD(int token, String name)	42
2.6.2.15.	void OnFilterUpdate(FilterRes res)	43
<b>3.</b>	<b>Trading</b>	44
<b>3.1.</b>	<b>UI Trading Parameters</b>	44
<b>3.1.1.</b>	<b>Strategy (1):</b>	44
	Rule that is to be traded. List of strategy rules available to a user are set up by Admin.	44
3.1.1.1.	<b>StrategyID Api attribute</b>	44
3.1.1.2.	<b>StrategyName api attribute</b>	44
<b>3.1.2.</b>	<b>Type (2)</b>	44
<b>3.1.3.</b>	<b>Mode (3)</b>	44
<b>3.1.4.</b>	<b>(3-7)</b>	45



3.1.5.	(8): Pricing Method .....	45
3.1.5.1.	ABSOLUTE .....	45
3.1.5.2.	PERCENT .....	45
3.1.6.	(9): Base Quantity .....	45
3.1.6.1.	BaseQty() API method .....	45
3.1.7.	(10): Max Quantity.....	45
3.1.7.1.	MaxQty() API method .....	45
3.1.8.	(11): SizeType .....	45
3.1.8.1.	SizeType() API method .....	45
3.1.8.2.	Notional .....	46
3.1.8.3.	Delta .....	46
3.1.8.4.	Vega.....	46
3.1.8.5.	Gamma.....	46
3.1.8.6.	Lots.....	46
3.1.9.	(12/13): Exit on Close & Mins before Close .....	46
3.1.10.	(14): Order Tip Size.....	46
3.1.11.	(15): SqOff Tip Size.....	46
3.1.12.	(20/21): OMS Frequency (Sec) & Use OMS Frequency .....	46
3.1.13.	(16): SQRrunning .....	46
3.1.14.	(17): Max Delta/Lot.....	47
3.1.15.	(18): Max Risk .....	47
3.1.16.	(19): Timer Frequency in Second(s) .....	47
3.1.17.	(25): Send (OnSave) .....	47
3.2.	Strategy Rule Attributes .....	47
3.2.1.	Key.....	48
3.2.2.	Constraints .....	48
3.2.3.	Value.....	48
3.2.4.	VALIDATIONS .....	Error! Bookmark not defined.
3.2.4.1.	MinValue.....	48
3.2.4.2.	MaxValue.....	48
3.2.4.3.	NA .....	48
3.2.4.4.	Valid Numeric Type .....	48



3.2.5.	Search TextBox .....	48
3.2.6.	Update Attributes actions .....	48
3.2.6.1.	Refresh .....	49
3.2.6.2.	Reset .....	49
3.2.7.	Attributes API Methods .....	49
3.2.7.1.	void OnPresetAttributes().....	49
3.2.7.2.	ExecutionResult AddAttributeString .....	49
3.2.7.3.	ExecutionResult AddAttributeBool.....	50
3.2.7.4.	ExecutionResult AddAttributeDecimalMinMax.....	50
3.2.7.5.	ExecutionResult AddAttributeDecimalMinMax.....	51
3.2.7.6.	ExecutionResult AddAttributeCollection .....	51
3.2.7.7.	String GetAttribute .....	52
3.2.7.8.	Boolean GetAttributeBool .....	52
3.2.7.9.	Decimal GetAttributeDecimal .....	52
Glossary .....		67
1-	GetStrike Function index or Index List parameters:.....	67
2-	StrikeGap.....	67
3-	out errorMessage, or out error, or out err .....	67
4-	Int32 expiry, or String expiry .....	67
5-	String Symbol.....	67
6-	GetParamValue & IndicatorName in GetIndicatorBar functions. ....	67
7-	ExecutionResult .....	68





# 1. Security

## 1.1. Security Class

Security refers to an instrument that can be traded. Security in MFT API is available with class name "Security" with following attributes,

- A. Token - Int32, unique number to identity a Security.
- B. Symbol – String, to identify the security.
- C. Type - String, to identify the security type
  - **CE** - Call Option
  - **PE** - Put Option
  - **XX** - Future
  - **EQ** - Stock
- D. Strike - Int32 Strike of Call/Put option in paisa.
- E. Expiry – Int32 expiry of call/Put/Future. To get a date representation of expiry, call one of the following functions on Security class.
  - **expiryToString()** - Expiry in 'ddMMMyyyy' format.
  - **expMonthString()** - Expiry in 'MMMyy' format
  - **expToDate()** - Expiry in DateTime c# structure.

## 1.2. Basket Class

Basket encapsulates set of securities as an array of Security class encapsulated by **BasketSecurity** with set of rules to buy, or sell in a given ratio of quantities,

### 1.2.1. Basket Security class

BasketSecurity security encapsulates a security with a Buy or Sell direction, with quantity defined in terms of Ratio. Actual quantity or buy or sell is implied from BaseQuantity set on StrategyBuilder UI.

There are two defined signatures available to define a BasketSecurity,

```
BasketSecurity(Int32 token, Decimal ratio, BuyOrSell bsType, Boolean hasQdap = true,  
String legName = null, String basketName = null)
```

And

```
BasketSecurity(Security s, Decimal ratio, BuyOrSell bsType, Boolean hasQdap = true, String  
legName = null, String basketName = null)
```





Both differ by first Parameter i.e. Token, or Security. In case of 1<sup>st</sup> signature – Security is retrieved internally, if not found – IsValid attribute is set to false.

#### 1.2.1.1. Attributes

- A. **Ratio:** Decimal, Defines the ratio of size of the security w.r.t BaseQuantity defined on the Strategy builder. For example, in case ratio = 1, and BaseQuantity = 1 lot, then in every order 1 lot of order is placed. Ratio = 1.5 and baseQuantity = 1000 Vega, then 1500 Vega value order is placed.
- B. **BuyOrSell:** Defines the direction of trade. Buy places a buy order, Sell places a Sell order.
- C. **HasQdap:** True/False Defines whether technical data will be retrieved from infrastructure, and real time data would stream in.
- D. **BasketName:** Defines name of the encapsulating basket.
- E. **LegName:** Defines name of the BasketSecurity (or Security) entity also called a Leg.
- F. **HasIndex:** Returns true or false indicating if encapsulated security is an Index or Stock. E.g. Nifty underlying returns true.
- G. **HasStock:** Returns true or false indicating if encapsulated security is a stock. E.g. SBIN returns true.
- H. **ToString():** returns string in format  
`{token}:{Ratio}:{BSType}:{LegName}:{BasketName}`

### 1.3. GetStrike function

GetStrike function API queries infrastructure to find strike of an option based search parameters. There are 28 different variations of this search available in the API detailed below.

#### 1.3.1. Decimal GetStrike(int token, int index, int strikeGap, out String errorMessage)

```
var err = String.Empty;  
var security = GetSecurity("SBIN", "M1");  
if (security != null)  
{  
    var atm = GetStrike(security._token, 0, 0, out err);  
    if (!String.IsNullOrEmpty(err))  
        LogError($"Failed: {err}");  
}
```

This function retrieves ATM Strike of a token (+/- index, 0= ATM) for current month's expiry. ATM strike is calculated using future price, i.e. strike that is closest to the future's price.

- A. Token – Security identifier.
- B. Index – Strike Index. See [Glossary 1](#).



- C. StrikeGap – Strike Gap in rupees. See [Glossary 2](#).
- D. errorMessage – error message if any. See [Glossary 3](#).

### 1.3.2. Decimal GetStrike(String symbol, int expiry, int index, int strikeGap, out String errorMessage)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
    var atm = GetStrike(security._symbol, security._expiry, 0, 100, out err);
    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
}
```

This function retrieves ATM Strike of a token (+/- index, 0= ATM) & expiry. ATM strike is calculate using future price, i.e. strike that is closest to the future's price.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Index – Strike Index. See [Glossary 1](#).
- C. Expiry: expiry in integer. See [Glossary 4](#).
- D. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- E. errorMessage – error message if any. See [Glossary 3](#).

### 1.3.3. Decimal GetStrike(String symbol, String expiry, int index, int strikeGap, out String errorMessage)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
    var atm = GetStrike(security._symbol, "M1", 0, 100, out err);
    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
}
```

This function retrieves ATM Strike of a symbol (+/- index, 0= ATM) & expiry. ATM strike is calculate using future price, i.e. strike that is closest to the future's price.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Index – Strike Index. See [Glossary 1](#).
- C. Expiry: expiry in String. See [Glossary 4](#).



- D. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- E. errorMessage – error message if any. See [Glossary 3](#).

#### 1.3.4. Decimal GetStrikeByParity(String symbol, String expiry, int index, int strikeGap, out String errorMessage)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
    var atmStrikg = GetStrikeByParity(security._symbol, 0,
    "M1", 100, out err);

    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
}
```

This function retrieves ATM strike of a symbol, and expiry. This functions returns the strike price where diff of call/put prices is minimum. At least 5 strikes are evaluated, in case there is not enough data, ATM is calculated from price of the future for requested symbol, and expiry.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: expiry in String. See [Glossary 4](#).
- C. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- D. errorMessage – error message if any. See [Glossary 3](#).

#### 1.3.5. Decimal GetStrikeByParity(String symbol, int expiry, int index, int strikeGap, out String errorMessage)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
    var atmStrikg = GetStrikeByParity(security._symbol, 0,
    security._expiry, 100, out err);

    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
}
```



This function retrieves ATM strike of a symbol, and expiry. This functions returns the strike price where diff of call/put prices is minimum. At least 5 strikes are evaluated, in case there is not enough data, ATM is calculated from price of the future for requested symbol, and expiry.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: expiry in String. See [Glossary 4](#).
- C. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- D. errorMessage – error message if any. See [Glossary 3](#).

**1.3.6.** Decimal[] GetAllStrikesByParity(String symbol, String expiry, int[] indexList, int strikeGap, out String errorMessage)

Same as [1.2.4](#), except that multiple indexes can be requested.

**1.3.7.** Decimal[] GetAllStrikesByParity(String symbol, int expiry, int[] indexList, int strikeGap, out String errorMessage)

Same as [1.2.5](#), except that multiple indexes can be requested.

**1.3.8.** Decimal[] GetAllStrikes(int token, int[] indexList, int strikeGap, out String errorMessage)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
    Decimal[] strikes = GetAllStrikes(security._token, new int[2] { 0, 1 }, 100, out err);
    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
}
```

This function retrieves an array of Strikes of a token for current month's at list of strike indexes provided in the array indexList.

- A. Token – Security identifier
- B. IndexList – Strike Index list. See [Glossary 1](#).



- C. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- D. errorMessage – error message if any. See [Glossary 3](#).

### 1.3.9. Decimal[] GetAllStrikes(String symbol, Int expiry, int[] indexList, int strikeGap, out String errorMessage)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
    Decimal[] strikes = GetAllStrikes(security._symbol, security._expiry, new
    int[2] { 0, 1 }, 100, out err);
    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
}
```

This function retrieves an array of Strikes of a token for a expiry at list of strike indexes provided in the array indexList.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: int expiry. See [Glossary 4](#).
- C. IndexList – Strike Index list. See [Glossary 1](#).
- D. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- E. errorMessage – error message if any. See [Glossary 3](#).

### 1.3.10. Decimal[] GetAllStrikes (String symbol, String expiry, int[] indexList, int strikeGap, out String errorMessage)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
    Decimal[] strikes = GetAllStrikes(security._symbol, security._expiry,
    new int[2] { 0, 1 }, 100, out err);
    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
}
```

This function retrieves an array of Strikes of a token for a expiry at list of strike indexes provided in the array indexList.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: expiry in String. See [Glossary 4](#).
- C. IndexList – Strike Index list. See [Glossary 1](#).



- D. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- E. errorMessage – error message if any. See [Glossary 3](#).

#### 1.3.11. Decimal[] GetAllStrikes(String symbol, int expiry, out String errorMessage)

```
var err = String.Empty;  
var security = GetSecurity("SBIN", "M1");  
if (security != null)  
{  
    Decimal[] strikes = GetAllStrikes(security._symbol, security._expiry,  
    out err);  
    if (!String.IsNullOrEmpty(err))  
        LogError($"Failed: {err}");  
}
```

This function retrieves all the Strike of a given symbol & expiry.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: int expiry. See [Glossary 4](#).
- C. errorMessage – error message if any. See [Glossary 3](#).

#### 1.3.12. Decimal[] GetAllStrikes(String symbol, String expiry, out String errorMessage)

```
var err = String.Empty;  
var security = GetSecurity("SBIN", "M1");  
if (security != null)  
{  
    Decimal[] strikes = GetAllStrikes(security._symbol, "M1", out err);  
    if (!String.IsNullOrEmpty(err))  
        LogError($"Failed: {err}");  
}
```

This function retrieves all the strikes of a given symbol & expiry.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: int expiry. See [Glossary 4](#).
- C. errorMessage – error message if any. See [Glossary 3](#).

#### 1.3.13. Decimal[] GetAllStrikes(String symbol, int expiry, int strikeGap, out String errorMessage)

```
var err = String.Empty;  
var security = GetSecurity("SBIN", "M1");  
if (security != null)
```



```
{
    Decimal[] strikes = GetAllStrikes(security._symbol, security._expiry,
    100, out err);
    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
}
```

This function retrieves all the Strike of a given symbol & expiry.

- A. Symbol – String Security. See [Glossary 5](#).
- F. Expiry: int expiry. See [Glossary 4](#).
- B. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- C. errorMessage – error message if any. See [Glossary 3](#).

#### 1.3.14. Decimal[] GetAllStrikes(String symbol, String expiry, out String errorMessage)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
    Decimal[] strikes = GetAllStrikes(security._symbol, "M1", 100, out
    err);
    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
}
```

This function retrieves all the strikes of a given symbol & expiry.

- A. Symbol – String Security. See [Glossary 5](#).
- G. Expiry: String expiry. See [Glossary 4](#).
- B. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- C. errorMessage – error message if any. See [Glossary 3](#).

#### 1.3.15. Decimal GetStrikeByPrice(String symbol, int expiry, Decimal price, int strikeGap, out String errorMessage)

```
var err = String.Empty;
var security = GetSecurity("NIFTY", "M1");
if (security != null)
{
    var atmStrikg = GetStrikeByPrice(security._symbol, security._expiry,
    14449, 100, out err);
    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
}
```





This function retrieves Strike of Nifty50 index, and expiry of current month's future closest to 14449 rupees i.e. 14500.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Index – Strike Index. See [Glossary 1](#).
- C. Expiry: expiry in String. See [Glossary 4](#).
- D. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- D. errorMessage – error message if any. See [Glossary 3](#).

#### 1.3.16. Decimal GetStrikeByPrice(String symbol, String expiry, Decimal price, int strikeGap, out String errorMessage)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
var atmStrikg = GetStrikeByPrice(security._symbol, "M1", 14449, 100, out
err);
if (!String.IsNullOrEmpty(err))
LogError($"Failed: {err}");
}
```

This function retrieves Strike of a given symbol, and current month's expiry future closest to 14449 rupees i.e. 14500.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Index – Strike Index. See [Glossary 1](#).
- C. Expiry: expiry in String. See [Glossary 4](#).
- D. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- E. errorMessage – error message if any. See [Glossary 3](#).

#### 1.3.17. Decimal GetStrikeByPercentageGap(String symbol, int expiry, Decimal percentageGap, int strikeGap, out String errorMessage)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
```



```
var per10Abovestrike = GetStrikeByPercentageGap(security._symbol,
security._expiry, 10m, 100, out err);
    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
}
```

This function retrieves Strike of a given symbol, and expiry perchantageGap % above or below strike. Send -ve value for below strike.

- A. Symbol – String Security. See [Glossary 5](#).
- B. percentageGap – % gap from ATM strike above or below strike.
- C. Expiry: expiry in String. See [Glossary 4](#).
- D. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- E. errorMessage – error message if any. See [Glossary 3](#).

### 1.3.18. Decimal GetStrikeByPercentageGap(String symbol, string expiry, Decimal percentageGap, int strikeGap, out String errorMessage)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
    var per10Abovestrike = GetStrikeByPercentageGap(security._symbol,
security._expiry, 10m, 100, out err);
    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
}
```

This function retrieves Strike of a given symbol, and expiry perchantageGap % above or below strike. Send -ve value for below strike.

- A. Symbol – String Security. See [Glossary 5](#).
- B. percentageGap – % gap from ATM strike above or below strike.
- C. Expiry: expiry in String. See [Glossary 4](#).
- D. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- E. errorMessage – error message if any. See [Glossary 3](#).

### 1.3.19. Decimal[] GetStrikeByPercentageGap(String symbol, int expiry, Decimal[] percentageGapList, int strikeGap, out String errorMessage)



Same as [1.2.17](#). Returns multiple strikes at multiple gaps. E.g. {-10, 0, 10} returns strikes 10% below ATM, ATM, and 10% above ATM.

### 1.3.20. Decimal[] GetStrikeByPercentageGap(String symbol, string expiry, Decimal[] percentageGapList, int strikeGap, out String errorMessage)

Same as [1.2.18](#). Returns multiple strikes at multiple gaps. E.g. {-10, 0, 10} returns strikes 10% below ATM, ATM, and 10% above ATM.

### 1.3.21. List<Decimal> GetAllStrikeByExpiry(String symbol, int expiry, int strikeGap, out String errorMessage)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
    var allStrikes = GetAllStrikeByExpiry(security._symbol,
security._expiry, 100, out err);
    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
}
```

This function retrieves all the strikes of a given symbol, and expiry

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: expiry in String. See [Glossary 4](#).
- C. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- D. errorMessage – error message if any. See [Glossary 3](#).

### 1.3.22. List<Decimal> GetAllStrikeByExpiry(String symbol, String expiry, int strikeGap, out String errorMessage)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
    var allStrikes = GetAllStrikeByExpiry(security._symbol,
"M1", 100, out err);
    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
}
```



This function retrieves all the strikes of a given symbol, and expiry

- E. Symbol – String Security. See [Glossary 5](#).
- F. Expiry: expiry in String. See [Glossary 4](#).
- G. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- H. errorMessage – error message if any. See [Glossary 3](#).

### 1.3.23. Dictionary<String, List<KeyValuePair<Int32, Decimal>>> GetAllOptionsWithDeltaRange(String symbol, int expiry, Decimal deltaLowerLimit, Decimal deltaUpperLimit, int strikeGap, out String err)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
    Dictionary<String, List <KeyValuePair <Int32, Decimal>>>
    retult = GetAllOptionsWithDeltaRange(security._symbol, security._expiry, 0.3m, 0.7m,
    100, out err);

    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
    else if (retult.ContainsKey("CE"))
    {
        //Call options between 0.3, and 0.7 delta.
        //KeyValuePair<int32, Decimal> sorted by delta in
        increasing order.
    }

    if (retult.ContainsKey("PE"))
    {
        //Put options between 0.3, and 0.7 delta.
        //KeyValuePair<int32, Decimal> sorted by delta in
        increasing order.
    }
}
```

This function retrieves all the strikes of a given symbol, and expiry for both Call, and Put options with (Delta > 0), (greater or equal to lower range i.e. 0.3 Delta), & (less than or equal to upper range i.e. 0.7 Delta). The key "CE" in the dictionary contains a list of CALL option strikes, with corresponding delta. The key "PE" in the dictionary contains a list of PUT option strikes, with corresponding delta in keyvalue list with key as strike, and value as Delta.



- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: expiry in int. See [Glossary 4](#).
- C. deltaLowerLimit: Lower Delta value in positive.
- D. upperDeltaLimit: upper Delta value.
- E. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- F. err – error message if any. See [Glossary 3](#).

### 1.3.24. Dictionary<String, List<KeyValuePair<Int32, Decimal>>>

GetAllOptionsWithDeltaRange(String symbol, [string](#) expiry, Decimal deltaLowerLimit, Decimal deltaUpperLimit, [int](#) strikeGap, [out](#) String err)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
    Dictionary<String, List <KeyValuePair <Int32, Decimal>>>
    retult = GetAllOptionsWithDeltaRange(security._symbol, "M1", 0.3m, 0.7m, 100, out
    err);

    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
    else if (retult.ContainsKey("CE"))
    {
        //Call options between 0.3, and 0.7 delta.
        //KeyValuePair<int32, Decimal> sorted by delta in
        increasing order.
    }

    if (retult.ContainsKey("PE"))
    {
        //Put options between 0.3, and 0.7 delta.
        //KeyValuePair<int32, Decimal> sorted by delta in
        increasing order.
    }
}
```

This function retrieves all the strikes of a given symbol, and expiry for both Call, and Put options with (Delta > 0), (greater or equal to lower range i.e. 0.3 Delta), & (less than or equal to upper range i.e. 0.7 Delta). The key "CE" in the dictionary contains a list of CALL option strikes, with corresponding delta. The key "PE" in the dictionary contains a list of PUT option strikes, with corresponding delta in keyvalue list with key as strike, and value as Delta.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: string expiry. See [Glossary 4](#).



- C. deltaLowerLimit: Lower Delta value in positive.
- D. upperDeltaLimit: upper Delta value.
- E. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- F. err – error message if any. See [Glossary 3](#).

### 1.3.25. Dictionary<String, List<KeyValuePair<Int32, Decimal>>> GetAllOptionsCloseToDelta(String symbol, int expiry, Decimal deltaThreshold, int strikeDivisor, Boolean bothSides, out String err)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
    Dictionary<String, List <KeyValuePair <Int32, Decimal>>> retult =
    GetAllOptionsCloseToDelta(security._symbol, security._expiry, 0.5m,
    100, true, out err);
    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
    else if (retult.ContainsKey("CE"))
    {
        //Call options between 0.3, and 0.7 delta.
        //KeyValuePair<int32, Decimal> sorted by delta in increasing order.
    }

    if (retult.ContainsKey("PE"))
    {
        //Put options between 0.3, and 0.7 delta.
        //KeyValuePair<int32, Decimal> sorted by delta in increasing order.
    }
}
```

This function retrieves all the strikes of a given symbol, and expiry for both Call, and Put options with (Delta > 0), and Delta either less than or equal to 0.5 Delta, or closer to 0.5 Delta in case if it's bothSides is set to true. The key "CE" in the dictionary contains a list of CALL option strikes, with corresponding delta. The key "PE" in the dictionary contains a list of PUT option strikes, with corresponding delta in keyvalue list with key as strike, and value as Delta.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: expiry in int. See [Glossary 4](#).
- C. deltaThreshold: Delta value in positive.
- D. bothSides: false than between 0 & deltaThreshold, else closest to deltaThreshold but could be greater than or less than deltaThreshold.



- E. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- F. err – error message if any. See [Glossary 3](#).

### 1.3.26. Dictionary<String, List<KeyValuePair<Int32, Decimal>>> GetAllOptionsCloseToDelta(String symbol, string expiry, Decimal deltaThreshold, int strikeDivisor, Boolean bothSides, out String err)

```
var err = String.Empty;
var security = GetSecurity("SBIN", "M1");
if (security != null)
{
    Dictionary<String, List <KeyValuePair <Int32, Decimal>>> retult =
    GetAllOptionsCloseToDelta(security._symbol, "M1", 0.5m, 100, true, out
    err);
    if (!String.IsNullOrEmpty(err))
        LogError($"Failed: {err}");
    else if (retult.ContainsKey("CE"))
    {
        //Call options between 0.3, and 0.7 delta.
        //KeyValuePair<int32, Decimal> sorted by delta in increasing order.
    }

    if (retult.ContainsKey("PE"))
    {
        //Put options between 0.3, and 0.7 delta.
        //KeyValuePair<int32, Decimal> sorted by delta in increasing order.
    }
}
```

This function retrieves all the strikes of a given symbol, and expiry for both Call, and Put options with (Delta > 0), and Delta either less than or equal to 0.5 Delta, or closer to deltaThreshold in case if it's bothSides is set to true. The key "CE" in the dictionary contains a list of CALL option strikes, with corresponding delta. The key "PE" in the dictionary contains a list of PUT option strikes, with corresponding delta in keyvalue list with key as strike, and value as Delta.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: expiry in int. See [Glossary 4](#).
- C. deltaThreshold: Delta value in positive.
- D. bothSides: false than between 0 & deltaThreshold, else closest to deltaThreshold but could be greater than or less than deltaThreshold.
- E. StrikeGap – Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. See [Glossary 2](#).
- F. err – error message if any. See [Glossary 3](#).





### 1.3.27. GetAllOptionsWithPriceRange, & GetAllOptionsCloseToPrice

Both of these 2 functions are exactly similar in behavior as functions around delta, except that both these 2 overloads retrieve options close to a positive price in rupees instead of Delta.

## 1.4. GetExpiry function

GetStrike function API queries infrastructure to find expiry of an option or future based search parameters. There are 2 different variations of this search available in the API detailed below.

### 1.4.1. `int` GetCurrentExpiry(String symbol, String expiryType = `null`)

```
int expiry = GetCurrentExpiry("NIFTY", "W1");
```

This function retrieves the expiry (in native int32 format) of a given symbol, and expiry in string format e.g. Mn or Wn.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: expiry in String, Default M1. See [Glossary 4](#).
- C. 0 return value means expiry not found.

### 1.4.2. `int` GetCurrentExpiry(String symbol, String expiryType = `null`)

```
DateTime expiry = GetCurrentExpiryDate("NIFTY", "W1");
```

This function retrieves the expiry in DateTime format of a given symbol, and expiry in string format e.g. Mn or Wn.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: expiry in String. See [Glossary 4](#).
- C. 0 return value means expiry not found.



## 1.5. GetSecurity function

GetSecurity function API queries infrastructure to find a security based on search parameters. There are 5 different variations of this search available in the API detailed below.

### 1.5.1. Security GetSecurity(String symbol)

```
Security sec = GetSecurity("SBIN");
```

This function retrieves CM or EQ security for the given symbol.

- A. Symbol – String Security. See [Glossary 5](#).
- B. Null result means security not found.

### 1.5.2. Security GetSecurity(int token)

```
Security sec = GetSecurity(1234);
```

If Token is known, this function can be called to retrieve the security instance associated with token 1234.

- A. token – int32. See [1.1](#).
- B. Null result means security not found.

### 1.5.3. Security GetSecurity(String symbol, String expiry)

```
Security sec = GetSecurity("NIFTY", "W1");
```

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: expiry in String. See [Glossary 4](#).
- C. null return value means security not found.

### 1.5.4. Security GetSecurity(String symbol, int expiry, Decimal strike, String type)



```
Security sec = GetSecurity("NIFTY",164556120, 19900, "CE");
```

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: expiry in int. See [Glossary 4](#).
- C. Strike in rupees. See [1.1](#)
- D. Type “CE” for call option, and “PE” for put option.
- E. null return value means security not found.

#### 1.5.5. Security GetSecurity(String symbol, string expiry, Decimal strike, String type)

```
Security sec = GetSecurity("NIFTY","W2", 19900, "CE");
```

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: expiry in string. See [Glossary 4](#).
- C. Strike in rupees. See [1.1](#)
- D. Type “CE” for call option, and “PE” for put option.
- E. null return value means security not found.

### 1.6. GetLotSize function

GetLotSize function API queries infrastructure to find the lot size of a security.

```
var lotSize = GetLotSize("NIFTY", "W1");
```

- A. Symbol – String Security. See [Glossary 5](#).
- B. Expiry: expiry in String. See [Glossary 4](#).
- C. 0 means not found.



## 2. Subscription, Data & Events.

### 2.1. StrategyBuildMode

Securities can be enlisted in a strategy either via API, or via GUI. This enum is not exposed to the API, however any error message suggesting subscription could not complete would indicate API method is not allowed.

#### 2.1.1.API

This mode allows subscription from API. GUI mode is disabled.

#### 2.1.2.GUI

This mode allows subscription from GUI. API mode is disabled. This mode is not allowed for MultiSecurity mode listed below.

### 2.2. StrategyTradingMode()

Security Listing can one of the following types,

#### 2.2.1.SingleSecurity

Single security can be subscribe for Data, and trading. API could still take trades using random tokens, however API doesn't provide MBP, and QDAP (indicators, filters, and OHLC) data on unsubscribed securities. Security cannot be edited in case an existing security is already subscribed. Strategy slots becomes available next day for new subscription again, or support could reset the subscription on request. **Legacy, this will be removed in future.**

#### 2.2.2.BASKET

Multiple securities be subscribe for Data, and trading. API could still take trades using random tokens, however API doesn't provide MBP, and QDAP (indicators, filters, and OHLC) data on unsubscribed securities (i.e. securities that are not part of the basket). Security list cannot be edited in case an active basket is already subscribed. Strategy slots becomes available next day for new subscription again, or support could reset the subscription on request. **Legacy, this will be removed in future.**

#### 2.2.3.MultiSecurity

Multiple securities be subscribe for Data, and trading. API could still take trades using random tokens, however API doesn't provide MBP, and QDAP (indicators, filters, and OHLC) data on unsubscribed securities (i.e. securities that are not part of the basket). Securities can be edited (removed, updated, and added) during trade except that in order



to update/delete there should not have position (long or short) in the security that is being updated or removed.

## 2.3. AddSecurity Method

Add security method subscribes and enlists securities to the infrastructure to allow security's market data (MBP) and technical data such as OHLC, Indicator, & filters data to start flowing into the MFT strategy.

### 2.3.1. Tuple<Int32, Int32> AddSecurity(Security singleSecurity, out String errorMessage)

AddSecurity method can be used to subscribe a single security to API, provided strategy is setup in SingleSecurity mode, once subscribed – cannot be edited.

```
String err = String.Empty;
var expiry = GetCurrentExpiry("NIFTY", "M1");
var strike = GetStrikeByParity("NIFTY", expiry, 0, 100, out err);
if(!String.IsNullOrEmpty(err))
{
    var security = GetSecurity("NIFTY", expiry, strike, "CE");
    if(null != security)
    {
        if(StrategyTradingMode() == TradingMode.SingleSecurity)
        {
            var result = AddSecurity(security, out err);
            if(String.IsNullOrEmpty(err))
            {
                LogInfo($"Subscribed NIFTY Call option with Token:
{result.Item1}, StrategyID: {result.Item2}");
            }
            else LogInfo($"Subscribed NIFTY Call option with Token:
{security._token}, StrategyID: {StrategyID}");
        }
    }
}
else LogError($"Failed to find ATM Strike for NIFTY, error: {err}");
```

#### 2.3.1.1. Details

- A. Return Value: Is Tuple<Int32, Int32>, with Item1 as Token of Subscribed Security, and Item2 as StrategyID.
- B. In case a security is already subscribed in the strategy, this function returns a Tuple<Int32, Int32>, with Item1 as Token of Subscribed Security, and



Item2 as StrategyID and err message containing text “Security is already subscribed...”

### 2.3.1.2. Parameters

- A. Security: Security to subscribe.
- B. err: Error message, see [Glossary3](#).

### 2.3.2. Tuple<Int64, Int32> AddBasket(BasketSecurity[] securityList, out String errorMessage)

AddBasket method can be used to subscribe a group of Basket securities to API, provided strategy is setup in Basket mode, once subscribed – cannot be edited.

```
String err = String.Empty;
var security1 = GetSecurity("SBIN", "M1");
var security2 = GetSecurity("TCS", "M1");
if (null != security1 && null != security2)
{
    if (StrategyTradingMode() == TradingMode.Basket)
    {
        var sbinBasket = new BasketSecurity(security1, 1, BuyOrSell.SELL,
true, "SBINLeg", "MyBasket");
        var tcsBasket = new BasketSecurity(security1, 1, BuyOrSell.BUY, true,
"TCSLeg", "MyBasket");

        var basketList = new BasketSecurity[2] { sbinBasket, tcsBasket };
        var result = AddBasket(basketList, out err);
        if (String.IsNullOrEmpty(err))
        {
            LogInfo($"Subscribed basket with Details: {String.Join("|",
basketList.Select(x=>x.ToString()))}, StrategyID: {result.Item2}");
        }
        else LogInfo($"Failed to subscribe basket with Details:
{String.Join("|", basketList.Select(x => x.ToString()))}, StrategyID: {StrategyID}");
    }
}
```

### 2.3.2.1. Details

- A. Return Value: Is Tuple<Int32, Int32>, with Item1 as ID of Subscribed Basket, and Item2 as StrategyID.
- B. In case a Basket is already subscribed in the strategy, this function returns a Tuple<Int32, Int32>, with Item1 as ID of Subscribed basket, and Item2 as StrategyID and err message containing text “Security is already subscribed...”



### 2.3.2.2. Parameters

- C. Security: Security to subscribe.
- D. err: Error message, see [Glossary3](#).

### 2.3.3. AddMultiSecurity(BasketSecurity[] securityList, out String errorMessage)

AddMultiSecurity method can be used to subscribe a group of Basket securities to API, provided strategy is setup in MultiSecurity mode. Securities subscribed in this mode can be edited later using AppendMultiSecurity, RemoveMultiSecurity, & EditMultiSecurity.

```
String err = String.Empty;
var security1 = GetSecurity("SBIN", "M1");
var security2 = GetSecurity("TCS", "M1");
if (null != security1 && null != security2)
{
    if (StrategyTradingMode() == TradingMode.MultiSecurity)
    {
        var sbinBasket = new BasketSecurity(security1, 1, BuyOrSell.SELL,
true, "SBINLeg", "MyBasket");
        var tcsBasket = new BasketSecurity(security1, 1, BuyOrSell.BUY, true,
"TCSLeg", "MyBasket");
        var basketList = new BasketSecurity[2] { sbinBasket, tcsBasket };
        var result = AddMultiSecurity(basketList, out err);
        if (String.IsNullOrEmpty(err))
        {
            LogInfo($"Subscribed MultiSecurity with Details:
{String.Join("|", basketList.Select(x=>x.ToString()))}, StrategyID: {result.Item2}");
        }
        else LogInfo($"Failed to subscribe MultiSecurity basket with Details:
{String.Join("|", basketList.Select(x => x.ToString()))}, StrategyID: {StrategyID}");
    }
}
```

### 2.3.3.1. Details

- A. Return Value: Is Tuple<Int32, Int32>, with Item1 as ID of Subscribed Basket, and Item2 as StrategyID.
- B. In case a Basket is already subscribed in the strategy, this function returns a Tuple<Int32, Int32>, with Item1 as ID of Subscribed basket, and Item2 as StrategyID and err message containing text "Security is already subscribed..."

### 2.3.3.2. Parameters

- A. Security: Security to subscribe.
- B. err: Error message, see [Glossary3](#).





### 2.3.4. Tuple<Int64, Int32> AppendMultiSecurity(BasketSecurity[] securityList, out String errorMessage)

This method, and signature is same as [2.3.4](#), and it's used to add one or more BasketSecurities to existing list.

```
String err = String.Empty;
if (StrategyTradingMode() == TradingMode.MultiSecurity)
{
    if (null != SubscribedMultiSecurities && null !=
        SubscribedMultiSecurities.Securities)
    {
        var infyBasket = new BasketSecurity(GetSecurity("INFY",
            "M1"), 1, BuyOrSell.SELL, true, "InfyLeg", "MyBasket");
        if (infyBasket.IsValid)
        {
            var result = AppendMultiSecurity(new
BasketSecurity[1] { infyBasket }, out err); //Append infy security, pass that leg as
an array
            if (String.IsNullOrEmpty(err))
            {
                LogInfo($"Appended MultiSecurity with Details:
{infyBasket.ToString()}, StrategyID: {result.Item2}");
            }
            else LogInfo($"Failed to append MultiSecurity
basket with Details: {infyBasket.ToString()}, StrategyID: {StrategyID}");
        }
    }
}
```

### 2.3.5. Tuple<Int64, Int32> RemoveMultiSecurity(BasketSecurity[] securityList, out String errorMessage)

This method, and signature is same as [2.3.4](#), and it's used to remove one or more BasketSecurities from existing list.

```
String err = String.Empty;
if (StrategyTradingMode() == TradingMode.MultiSecurity)
{
    if (null != SubscribedMultiSecurities && null !=
        SubscribedMultiSecurities.Securities)
    {
        var existing =
        SubscribedMultiSecurities.Securities.ToList();
        var infyBasket = existing.FirstOrDefault(x => x.LegName
            == "InfyLeg");
```



```
        if (infyBasket != null)
        {
            var result = RemoveMultiSecurity(new
BasketSecurity[1] { infyBasket }, out err); //Remove infy security, pass that leg as
an array
            if (String.IsNullOrEmpty(err))
            {
                LogInfo($"Removed MultiSecurity with Details:
{infyBasket.ToString()}, StrategyID: {result.Item2}");
            }
            else LogInfo($"Failed to remove MultiSecurity
basket with Details: {infyBasket.ToString()}, StrategyID: {StrategyID}");
        }
    }
}
```

#### 2.3.5.1. Details

- A. An error message is returned in case an attempt is made to remove a leg that has an active position, and it's not allowed.

#### 2.3.6. Tuple<Int64, Int32> EditMultiSecurity(BasketSecurity[] securityList, out String errorMessage)

This method, and signature is same as [2.3.4](#), and it's used to Add or remove one or more BasketSecurities from existing list.

```
String err = String.Empty;

if (StrategyTradingMode() == TradingMode.MultiSecurity)
{
    if (null != SubscribedMultiSecurities && null !=
SubscribedMultiSecurities.Securities)
    {
        var existing =
SubscribedMultiSecurities.Securities.ToList();
        var infyBasket = existing.FirstOrDefault(x => x.LegName
== "InfyLeg"); //To Remove
        var niftyBasket = new
BasketSecurity(GetSecurity("NIFTY", "M1"), 1, BuyOrSell.BUY, true, "InfyLeg",
"MyBasket"); //To Add
        if (infyBasket != null)
        {
            existing.Remove(infyBasket);
            existing.Add(niftyBasket);
            var result = EditMultiSecurity(existing.ToArray(),

out err);

            if (String.IsNullOrEmpty(err))
            {
                LogInfo($"Subscribed MultiSecurity with
Details: {String.Join("|", existing.ToArray().Select(x => x.ToString()))},
StrategyID: {result.Item2}");
            }
        }
    }
}
```



```
        }  
        else LogInfo($"Failed to MultiSecurity basket with  
Details: {String.Join("|", existing.ToArray().Select(x => x.ToString()))},  
StrategyID: {StrategyID}");  
    }  
}
```

#### 2.3.6.1. Details

- A. An error message is returned in case an attempt is made to remove a leg that has an active position, and it's not allowed.

## 2.4. GetSubscribedSecurity Method

GetSubscribedSecurity has 3 different flavours to retrieve the security information subscribed based on StrategyTradingMode().

### 2.4.1. SubscribedSingleSecurity

Returns Security class that is subscribed. Null in case StrategyTradingMode is not set to SingleSecurity.

### 2.4.2. SubscribedBasket

Returns Basket class that is subscribed. Null in case StrategyTradingMode is not set to Basket.

### 2.4.3. SubscribedMultiSecurities

Returns Basket class that is subscribed. Null in case StrategyTradingMode is not set to MultiSecurity.

```
if(StrategyTradingMode() == TradingMode.SingleSecurity)  
{  
    Security security = SubscribedSingleSecurity;  
}  
else if (StrategyTradingMode() == TradingMode.Basket)  
{  
    Security[] security = (SubscribedBasket != null &&  
SubscribedBasket.Securities != null) ?  
SubscribedBasket.Securities.Select(x=>x.Security).ToArray() : null;
```



```
        BasketSecurity[] basSecurity = (SubscribedBasket != null &&  
SubscribedBasket.Securities != null) ? SubscribedBasket.Securities.ToArray() : null;  
    }  
    else if (StrategyTradingMode() == TradingMode.SingleSecurity)  
    {  
        Security[] security = (SubscribedMultiSecurities != null &&  
SubscribedMultiSecurities.Securities != null) ? SubscribedMultiSecurities.Securities.Select(x =>  
x.Security).ToArray() : null;  
        BasketSecurity[] basSecurity = (SubscribedMultiSecurities != null &&  
SubscribedMultiSecurities.Securities != null) ? SubscribedMultiSecurities.Securities.ToArray() :  
null;  
    }  
}
```

## 2.5. Data

Data is relayed to the strategy by the infrastructure through Data events fired periodically in the strategy. Data packets can be stored by the strategy, or can be accessed using methods exposed by the API detailed later in this document. MBP (Market by price) data API only stores latest data points – so it's upto the strategy writer to store the data for analytics. Technical data however, is stored by the API, and can be accessed through API functions and is available throughout the life time of the Strategy.

### 2.5.1. MBP Data

MBP data class in MFT API stores latest Market by price data for every instrument that is subscribed by the strategy. For example – in case if the strategy has subscribed SBIN future, and TCS future - **latest MBP data** is stored by the API for both the instruments, and is relayed to the as soon as a tick arrives from the exchange via an appropriate event details below.

Following is a list of attributes available in MBP class,

// All Prices are in rupees.

DateTime DtTime	DateTime Time of the tick
DateTime LTT	Last Traded Time
Decimal LTP	Last Traded Price
Decimal LTQ	Last Traded Quantity
Decimal Close	Close Price (last day)
Decimal High	Day's High
Decimal Low	Day's Low
Decimal Open	Day's Open
Decimal[] Bids	Best Open 5 bids



Decimal[] BidSize	Best Open 5 bids Sizes
Decimal[] Asks	Best Open 5 Asks
Decimal[] AskSize	Best Open 5 Ask sizes
Int32 Token	Token
Decimal TTE	Time to expiry in numeric
Decimal SyntheticFutPrice	Implied synthetic future price (available on Option instruments only)
Decimal Delta	Delta Greek
Decimal Gamma	Gamma Greek
Decimal Vega	Vega Greek
Decimal Theta	Theta Greek
Decimal Rho	Rho Greek
Decimal Bidlv	Bid IV, (available on Option instruments only)
Decimal Asklv	Ask IV, (available on Option instruments only)
Decimal Ltplv	LTP IV, (available on Option instruments only)
Int32 OI	Open Interest
Int32 Volume	Volume trade
Int64 TBQ	Total buy quantity
Int64 TSQ	Total sell quantity
Int32 HIGHOI	High OI
Int32 LOWOI	Low OI
ulong TimeOfTick	long Time of the tick
Int32 Expiry	int expiry
Decimal ATMStrike	ATM Strike, only available on Future instruments
Decimal ATMIV	ATM IV, only available on Future instruments
Decimal SKEW	Skew, only available on Future instruments
Decimal W1ATMIV	Week1... Week5 ATM IV, only available on Future instruments that has weekly options
Decimal W1SKEW	Week1... Week5 SKEW, only available on Future instruments that has weekly options
Int32 W1Expiry	Week1... Week5 Expiry, only available on Future instruments that has weekly options
Decimal W1ATMStrike	Week1... Week5 ATM Strike, only available on Future instruments that has weekly options
Decimal W1SyntheticPrice	Week1... Week5 Synthetic price, only available on Future instruments that has weekly options
Decimal W2ATMIV	
Decimal W2SKEW	
Int32 W2Expiry	
Decimal W2ATMStrike	
Decimal W2SyntheticPrice	



Decimal W3ATMIV	
Decimal W3SKEW	
Int32 W3Expiry	
Decimal W3ATMStrike	
Decimal W3SyntheticPrice	
Decimal W4ATMIV	
Decimal W4SKEW	
Int32 W4Expiry	
Decimal W4ATMStrike	
Decimal W4SyntheticPrice	
Decimal W5ATMIV	
Decimal W5SKEW	
Int32 W5Expiry	
Decimal W5ATMStrike	
Decimal W5SyntheticPrice	

### 2.5.2.OHLC Data

OHLCBar class stores OHLC (Open, High, Low, and Close) data in MFT API for granularity (Time interval) and every instrument that is subscribed by the strategy.

Following is a list of attributes available in OHLCBar class,

**// All Prices are in rupees.**

Double Open	Open price for each granularity
Double High	High price for each granularity
Double Low	Low price for each granularity
Double Close	Close price for each granularity
Double Volume	Volume traded for each granularity
DTTime	Time of bar
Symbol	Instrument's Symbol
Token	Instrument's Token



### 2.5.3. Indicator Data

IndicatorBar class stores technical indicator data such RSI, ATR, MovingAverages etc. data in MFT API for granularity (Time interval) and every instrument that is subscribed by the strategy.

Following is a list of attributes/functions available in IndicatorBar class,

// All Prices are in rupees (a complete list of keys associated with each indicator is detailed in attached document).

Double GetParamValue(String key) function	Returns value of indicator name requested. For example, for RSI key is RSI.
DTTime	Time of bar
Symbol	Symbol of bar
Token	Token Of bar

IndicatorDefinitions.pdf document has all indicators available in MFT API available, with dictionary key, or key name detailed in Output section.

### 2.5.4. BarType Enum

LIVE	Live Bar.
HISTORY	Time of bar
EOD	Symbol of bar

IndicatorDefinitions.pdf document has all indicators available in MFT API available, with dictionary key, or key name detailed in Output section.

### 2.5.5. Filter Data

FilterRes class represents filter data such as Top Market Gainers, Unusual Volume, 52 Weeks low, bearish etc. in MFT API for granularity (Time interval). This data is not stored by API. It's only relayed at regular time intervals until Market closes.

Following is a list of attributes/functions available in FilterRes class,

List<SEC_INFO> _secInfo	Stores list of Security information data (int token, string symbol)
Int FilterType	Filter Type





Following is a list of enum types (represented by int32) applicable on FilterRes for FilterType field.

- TOP\_MARKET\_GAINERS = 0,
- TOP\_MARKET\_LOSERS = 1,
- GAP\_UP = 2,
- GAP\_DOWN = 3,
- UNUSUAL\_VOL = 4,
- BULLISH = 5,
- BEARISH = 6,
- NEW\_52\_WEEK\_LOW = 7,
- NEW\_52\_WEEK\_HIGH = 8,
- CUSTOM\_FILTER = 9,
- BANNED\_SEC = 10,
- MWPL\_SEC = 11,
- RESULTS\_CALEDNDAR = 12,
- STOCKS\_IN\_ACTION = 13

## 2.6. Data Events & Functions.

Data events are set of pre-defined must implement, and optional functions. Role of these functions is to stream data into the strategy in real time – instantly once the data arrives into infrastructure. Listed below are set of function signatures, and their parameter attributes.

### 2.6.1.MBP Data

Following a list of functions to retrieve latest MBP parameters, or entire MBP structure from the API.

#### 2.6.1.1. Decimal LTP(int token)

Returns last traded price of Security's token.

#### 2.6.1.2. Decimal OPEN(int token)

Returns Open price of Security's token.

#### 2.6.1.3. Decimal CLOSE(int token)

Returns Close price of Security's token

#### 2.6.1.4. MBP GetGlobalMBPTick(int token, out String errorMessage)

Returns latest MBP class instance of any security's token, errorMessage returns a message indicating failure retrieving data from the API e.g. Invalid Token.



2.6.1.5. MBP GetSubscribedMBPTick (int token, out String errorMessage)  
Returns latest MBP class instance of subscribed security's token (securities that are subscribed to strategy only), errorMessage returns a message indicating failure retrieving data from the API e.g. Invalid Token.

## 2.6.2. OHLC, Indicator, & Filter data

Subscription definition is defined on the user interface while defining the strategy rule. System relays Live Data to OnBarUpdate(), and Historical data to OnHistoryBarUpdate() events in the strategy at interval defined in the screenshot below against TimeFrame, and for All the instruments subscribed to the Strategy using (HasQDap = true) defined in 1.2.1.1.

### 2.6.2.1. Definition.

The screenshot displays the configuration interface for a trading strategy. It features two dropdown menus for 'Indicators' and 'Filters'. The 'Indicators' dropdown is set to 'ATR15,RSI15,SMA15', and the 'Filters' dropdown is set to 'TopLoser,TopGainers'. Below these are two rows of buttons: 'ATR15', 'RSI15', and 'SMA15' in the first row, and 'TopLoser' and 'TopGainers' in the second row. At the bottom, there are four input fields: 'Time Frame' set to '1M', 'INTRADData to load (N)' set to '2', 'EODData to load (N)' set to '2', and 'Data Series Type' set to 'CANDLESTICK'.

- **Indicators** – Is list of system defined Indicator that could be subscribed in the strategy. For example in the screenshot above, Strategy has subscribe to ATR15 (Active Trade Range of 15 days), RSI 15 (Relative Strength Index of 15 days period), and SMA (Simple Moving average of 15 days period)
- **Filters** – Is list of system defined filters that could be subscribed in the strategy. For example in the screenshot above, Strategy has subscribe to TopGainers, and TopLosers.
- **TimeFrame** – Is the interval for which data sampling is done, and at which data is relayed to the strategy. Interval format is defined as nI, where n is total length of interval, and I is interval. I can be S – for second, M for minute, and D for day. In the example above, strategy has subscribed Time Frame as 1M (one minute)
- **INTRADData To Load** – In case historical data is needed, number of days could be set in this parameter. In the example above, Strategy has requested 2 days of 1 minute historical data.
- **EODData To Load** – In case EOD historical data is needed, number of days could be set in this parameter. In the example above, Strategy has requested 2 days of 1 minute historical data.
- **Data Series Type** – Defines the type of OHLC data required by the strategy. Possible options are “CANDLESTICK”, “HEIKENASHI”, & “REKNO”.



#### 2.6.2.2. void OnBarUpdate

OnBarUpdate is a **must implement** function in the strategy. This function is automatically called at every interval = Granularity subscribed by the strategy.

```
public override void OnBarUpdate()
{
}
```

#### 2.6.2.3. void OnHistoryBarUpdate

OnHistoryBarUpdate is a **must implement** function in the strategy. This function is automatically called when all the history bar data is downloaded by the strategy.

```
public override void OnHistoryBarUpdate()
{
}
```

#### 2.6.2.4. OHLCVBar GetDataSeriesBar(int token, int index, BARType type)

A. Returns OHLCVBar data for parameters supplied.

B. Index – index of bar. Index = 0 is latest, 1 is next, and so on.

C. BARType refer [2.5.4](#)

```
public override void OnBarUpdate()
{
    var err = String.Empty;
    var now = DateTime.Now;
    var time1130 = new DateTime(now.Year, now.Month, now.Day, 11, 30, 00);
    if (null != SubscribedMultiSecurities)
    {
        var sbinBasketSecurity =
SubscribedMultiSecurities.Securities.FirstOrDefault(x => x.LegName == "SBINLeg");
        if (null != sbinBasketSecurity)
        {
            //Get Latest Bar for SBIN Future.
            OHLCVBar bar = GetDataSeriesBar(sbinBasketSecurity.Token, 0,
BARType.LIVE);

            if (null == bar)
                LogError($"Failed: SBIN BAR Not found");
            else LogInfo($"Recieved: SBIN BAR: {bar.ToString()}");
        }
    }
}
```



#### 2.6.2.5. OHLCVBar GetDataSeriesBar(int token, DateTime dateTime, BARType type)

- A. Returns OHLCVBar data for parameters supplied.
- B. dateTime – time of bar.
- C. BARType refer [2.5.4](#)

```
//Get 11:30 Bar for SBIN Future.  
OHLCVBar bar1130 = GetDataSeriesBar(sbinBasketSecurity.Token, time1130,  
BARType.LIVE);  
if (null == bar)  
    LogError($"Failed: 11:30 AM SBIN BAR Not found");  
else LogInfo($"Recieved: 11:30 AM SBIN BAR: {bar.ToString()}");
```

#### 2.6.2.6. List<OHLCVBar> DataSeriesHistory(int token)

Returns collection of all the Historical bars for the Token requested by the strategy rule. Bars are sorted by time in descending order, **Latest first**. E.g. if granularity is 1 minute, then 15:29, 15:28, and so on.

#### 2.6.2.7. List<OHLCVBar> DataSeriesLive(int token)

Returns collection of all the Live bars for the Token requested by the strategy rule. Bars are sorted by time in descending order, **Latest first**.

#### 2.6.2.8. List<OHLCVBar> DataSeriesEOD(int token)

Returns collection of all the EOD (end of the day or 15:29 PM) bars for the Token requested by the strategy rule. Bars are sorted by time in descending order, **Latest first**.

#### 2.6.2.9. EarliestFirst version.

Returns collection of all the Live bars for the Token requested by the strategy rule. Bars are sorted by time in Ascending order, **Earliest first**. E.g.

**DataSeriesHistoryEarliestFirst function** returns all history bar of the token in ascending order of time. E.g. if granularity is 1 minute, then 9:15, 9:16, and so on.



#### 2.6.2.10. IndicatorBar GetIndicatorBar(DateTime dateTime, BARType type, String name)

- A. Returns IndicatorBar data for parameters supplied.
- B. Index – index of bar. Index = 0 is latest, 1 is next, and so on.
- C. BARType refer [2.5.4](#)
- D. String name: Name of indicator setup on user interface. E.g RSI15 set up in section [2.6.2.1](#) and [Glossary 6](#)

```
public override void OnBarUpdate()
{
    var err = String.Empty;
    var now = DateTime.Now;
    var time1130 = new DateTime(now.Year, now.Month, now.Day, 11, 30, 00);
    if (null != SubscribedMultiSecurities)
    {
        var sbinBasketSecurity =
SubscribedMultiSecurities.Securities.FirstOrDefault(x => x.LegName == "SBINLeg");
        if (null != sbinBasketSecurity)
        {
            //Get Latest Bar for SBIN Future.
            IndicatorBar bar = GetIndicatorBar(sbinBasketSecurity.Token, 0,
BARType.LIVE, "RSI15");
            if (null == bar)
                LogError($"Failed: SBIN BAR Not found");
            else
            {
                var rsi = bar.GetParamValue("RSI");
                LogInfo($"Recieved: latest 15 days interval RSI value for SBIN
: {rsi.ToString()}");
            }
        }
    }
}
```



#### 2.6.2.11. IndicatorBar GetIndicatorBar(DateTime dateTime, BARType type, String name)

- A. Returns IndicatorBar data for parameters supplied.
- B. dateTime – time of bar.
- C. BARType refer [2.5.4](#)
- D. String name: Name of indicator setup on user interface. E.g RSI15 set up in section [2.6.2.1](#) and [Glossary 6](#)

```
//Get 11:30 Bar for SBIN Future.  
IndicatorBar bar1130 = GetIndicatorBar(sbinBasketSecurity.Token,  
time1130, BARType.LIVE, "RSI15");  
if (null == bar)  
    LogError($"Failed: SBIN BAR Not found");  
else  
{  
    var rsi = bar.GetParamValue("RSI");  
    LogInfo($"Recieved: 11:30 AM 15 days interval RSI value for  
SBIN : {rsi.ToString()}");  
}
```

#### 2.6.2.12. List<IndicatorBar> IndicatorHistory(int token, String name)

Returns collection of all the Historical indicator bars for the Token, and name requested by the strategy rule. Bars are sorted by time in descending order, **Latest first**. E.g. if granularity is 1 minute, then 15:29, 15:28, and so on.

#### 2.6.2.13. List<IndicatorBar> IndicatorLive(int token, String name)

Returns collection of all the Live indicator bars for the Token, and name requested by the strategy rule. Bars are sorted by time in descending order, **Latest first**.

#### 2.6.2.14. List<IndicatorBar> IndicatorEOD(int token, String name)

Returns collection of all the EOD (end of the day or 15:29 PM) indicator bars for the Token, and name requested by the strategy rule. Bars are sorted by time in descending order, **Latest first**.



#### 2.6.2.15. void OnFilterUpdate(FilterRes res)

OnHistoryBarUpdate is optional **overridable** function in the strategy. This function is automatically called when all the filter bar data is relayed by the strategy at interval = TimeFrame.


```
public void OnFilterRes(FilterRes filterRes)
{
    switch ((FilterTypes)filterRes.filter_type)
    {
        case FilterTypes.TOP_MARKET_GAINERS:
        case FilterTypes.TOP_MARKET_LOSERS:
        case FilterTypes.GAP_UP:
        case FilterTypes.GAP_DOWN:
        case FilterTypes.BULLISH:
        case FilterTypes.BEARISH:
        case FilterTypes.NEW_52_WEEK_HIGH:
        case FilterTypes.NEW_52_WEEK_LOW:
        case FilterTypes.UNUSUAL_VOL:
            {
                foreach (SEC_INFO sec in filterRes._secInfo)
                {
                    var securityToken = sec.token;
                    var securitySymbol = sec.symbol;
                    var framTime = filterRes._time_frame;
                }
                break;
            }
    }
}
```



## 3. Trading

### 3.1. UI Trading Parameters

Strategy builder is used to add initial parameters to a strategy.



The screenshot shows the 'WIZARD BUILDER: ADD NEW' window. It includes a table of attributes on the left, a main configuration area with dropdowns for Strategy, Type, and Mode, and a grid of input fields for various trading parameters. Red numbers 1 through 25 are overlaid on the interface to highlight specific elements.

Key	Constraints	Value
Symbol		SBIN
FutureExpiry		M1
OptionExpiry		M1
StrikeDivisor	Min: 0 Max: NA	0
DeltaCE	Min: 0 Max: NA	0.2
ResDeltaTh	Min: 0 Max: NA	0.4
DeltaGap	Min: 0 Max: NA	0.3
UpperDeltaTH	Min: 0 Max: NA	0.5
LowerDeltaTH	Min: 0 Max: NA	0.5
StraddlePer	Min: 0 Max: NA	10
StraddleEnterTime		

Strategy: Atlas **1** Type: MultiSecurity **2** Mode: API **3**

Indicators: Filters:

Time Frame: 1M **4** INTRADData to load (N): 0 **5** EODData to load (N): 0 **6** Data Series Type: CANDLESTICK **7**

Pricing Method: ABSOLUTE **8** Base Qty (L): 1 **9** Max Qty (L): 2 **10** Size Type: NOTIONAL **11**

Exit On Close: ☒ **12** Mins Before Close: 10 **13** Order Tip Size: 100 **14** SqOff Tip Size: 100 **15**

Max Delta/Lot (T): 10 **17** Max Risk (L): 10 **19** Timer Frequency (Sec): 1 **19** SQRunning: ☒ **16**

Exec Method (Buy): NONE **22** Exec Method (Sell): NONE **23** Default Expiry (FOCM): M1 **24** OMS Frequency (Sec): .25 **20**

Use OMS Frequency: ☐ **21** Send (OnSave): ☒ **25**

ADD

#### 3.1.1.Strategy (1)

Rule that is to be traded. List of strategy rules available to a user are set up by Admin.

##### 3.1.1.1. StrategyID Api attribute

Returns unique numeric ID of the strategy rule.

##### 3.1.1.2. StrategyName api attribute

Returns string name of the strategy rule.

#### 3.1.2.Type (2)

Strategy can be defined and code in 3 modes. See [section 2.2](#) for more details.

#### 3.1.3.Mode (3)

Set StrategyBuildMode. See [section 2.1](#) for more details.



### 3.1.4. (3-7)

These are defined in section 2.6.2.1.

### 3.1.5. (8): Pricing Method

Absolute, and Percentage.

#### 3.1.5.1. ABSOLUTE

All PnL changes in the strategy are calculated in absolute terms i.e. in money.

#### 3.1.5.2. PERCENT

All PnL changes in the strategy are calculated in percentage terms e.g. 10% loss or 10% profit.

### 3.1.6.(9): Base Quantity

Base Quantity in absolute terms to trade. For example – base quantity is set to 1000 delta (SizeType), every Buy/Sell order would execute options equivalent to 1000 delta per order.

#### 3.1.6.1. BaseQty() API method

Gets value set on GUI.

### 3.1.7.(10): Max Quantity

Maximum Quantity in absolute rule could trade. For example – max quantity is set to 100 lots (SizeType), strategy rule stops trading once traded quantity (open positions) + ordered quantity breaches 100 lots on new order.

#### 3.1.7.1. MaxQty() API method

Gets value set on GUI.

### 3.1.8.(11): SizeType

Sets size type to trade.

#### 3.1.8.1. SizeType() API method

Gets value set on GUI.

**3.1.8.2. Notional**

Lot Size \* LTP → Total notional to trade in one build order.

**3.1.8.3. Delta** → Total delta to trade in one build order.

**3.1.8.4. Vega** → Total vega to trade in one build order.

**3.1.8.5. Gamma** → Total gamma to trade in one build order.

**3.1.8.6. Lots** → Total number of lots to trade in one build order.

**3.1.9.(12/13): Exit on Close & Mins before Close**

If checked, strategy rule exits minutes (value set) before exchange close, and stops executing any more orders. For example if value is checked, and value is set to 10 minutes – all active positions square off at 15:20, and rule stops executing an new order after 15:20 (considering exchange closes at 15:30).

**3.1.10.(14): Order Tip Size**

Value in percentage to instruct oms build smaller orders from exist order. For example, base quantity is set to 100 lots, and Order Tip Size – case 1 100% - Sends 1 order of 100 Lots, Case 2 10% - sends 10 orders of 10 lots each.

**3.1.11.(15): SqOff Tip Size**

Value in percentage to instruct oms build smaller square off orders from exist square off order. For example, 100 lots square off quantity, and SqOff Tip Size – **Case 1** 100% - Sends 1 order of 100 Lots, **Case 2** 10% - sends 10 orders of 10 lots each.

**3.1.12.(20/21): OMS Frequency (Sec) & Use OMS Frequency**

In case Use OMS Frequency is checked, OMS waits for number of seconds between 2 consecutive orders. For example in case of Order Tip Size – in case 2 – every order from sequence 1-10 get executed as soon as previous order is executed in case Use OMS frequency is set to false, or else OMS waits for 250 milli seconds (based on GUI value .25 i.e. 250 mill secs) between previous and new order.

**3.1.13.(16): SQRunning**

Defines whether Running PnL, or TotalPnL is used for Square off Triggers within MFT.



### 3.1.14. (17): Max Delta/Lot

Defines maximum delta per lot a strategy rule could build.

### 3.1.15. (18): Max Risk

Defines maximum Risk strategy rule could build.

### 3.1.16. (19): Timer Frequency in Second(s)

Defines frequency of OnTimer() API function callbacks within a rule. Minimum value is .25 i.e. equal to 250 milli seconds.

### 3.1.17. (25): Send (OnSave)

If checked attributes are sent to engine on every Add/Update on Strategy builder GUI.

## 3.2. Strategy Rule Attributes

Strategy rule attributes are set of key value values that could be setup in a rule. These are versatile set of values that could be used anywhere in the strategy to make trading decisions. For example, String type attribute by name: **Symbol** with value “**NIFTY**” to make a trading decision to trade in Nifty Index. There are 4 different types of attributes that could be setup in a strategy,

The screenshot shows a 'WIZARD BUILDER: ADD NEW' window with a table of attributes. The table has columns for 'Key', 'Constraints', and 'Value'. The attributes listed are Symbol (Nifty), DeltaThreshold (30), MaximumDelta (Min: 0, Max: 100, 30), and Hedge (checkbox).

Key	Constraints	Value
Symbol		Nifty
DeltaThreshold		30
MaximumDelta	Min: 0 Max: 100	30
Hedge		<input type="checkbox"/>



### 3.2.1. Key

Name of the attribute (String).

### 3.2.2. Constraints

Applicable on Numeric types with Min/Max value constraints. In case constraint is NA for min value, attribute can have any minimum value, or else cannot be less than Min value setup on the constraint, and vice versa for Max value.

### 3.2.3. Value

Value in DataType setup on the attribute. For Collection Type attribute, value can be selected from a drop down, for Boolean type attribute, value can be set on a checkbox, and for all others value can be set on a text box.

### 3.2.4. Validations

Validations apply on numeric type of data on attributes panels. Numeric values always stored data in Decimal format, and value could be constrained one of the following,

#### 3.2.4.1. MinValue

Entered value should greater or equal to the value set on constraint. If value displayed is 'NA' that means it could hold value upto Decimal.MinValue.

#### 3.2.4.2. MaxValue

Entered value should smaller or equal to the value set on constraint. If value displayed is 'NA' that means it could hold value upto Decimal.MaxValue.

#### 3.2.4.3. NA

Entered value could be between Decimal.MinValue & Decimal.MaxValue.

#### 3.2.4.4. Valid Numeric Type

Entered value should be a valid numeric type.

### 3.2.5. Search TextBox

Enter value in search text box to search all Key names in attributes panel.

### 3.2.6. Update Attributes actions

By default – all attributes in panel do not change once the strategy is saved. However, a strategy developer could update strategy parameters such as KeyName, validations using OnPresetAttributes() function in the strategy. Using Update attributes function, all attributes can be updated in the strategy.



### 3.2.6.1. Refresh

Refresh removes deleted attributes, re-adds existing attributes with values set earlier on the strategy. In case attribute did not exist earlier, adds it with default value.

### 3.2.6.2. Reset

Removes existing attributes, and adds new attributes with default value.

## 3.2.7. Attributes API Methods.

### 3.2.7.1. void OnPresetAttributes()

overrideable method that can be implemented within the strategy to setup all the attributes required in the strategy.

```
protected override void OnPresetAttributes()
{
    var result = AddAttributeString("Symbol", "BANKNIFTY", true, true);
    if (result.Result == ExecutionResultType.VALIDATION)
        LogWarn($"Error: {result.ExecutionMessage}", true);

    result = AddAttributeString("Expiry", "W1", true, true);
    if (result.Result == ExecutionResultType.VALIDATION)
        LogWarn($"Error: {result.ExecutionMessage}", true);

    result = AddAttributeDecimalMinMax("OTMPrice", 5m, true, true, 0);
    if (result.Result == ExecutionResultType.VALIDATION)
        LogWarn($"Error: {result.ExecutionMessage}", true);

    result = AddAttributeBool("IsAuto", true, true, true);
    if (result.Result == ExecutionResultType.VALIDATION)
        LogWarn($"Error: {result.ExecutionMessage}", true);

    result = AddAttributeString("Strats", "", true, true);
    if (result.Result == ExecutionResultType.VALIDATION)
        LogWarn($"Error: {result.ExecutionMessage}", true);
}
```

### 3.2.7.2. ExecutionResult AddAttributeString

Used to add a string type attribute to the strategy rule. Following is a list of parameters,

- A. String attributeName: Name or key of the attribute.
- B. String defaultValue: Default value of the attribute.
- C. Boolean Editable – Defines whether attribute can be edited. Optional parameter, defaults to true.



- D. Boolean Visible – defines if attributes would be visible on the GUI. Optional parameter, defaults to true.
- E. Boolean isBulkUpdatable: defines if attribute would be available on the bulk update interface. Optional parameter, defaults to false.
- F. String defaultValueForBulkUpdate: String value that is used to setup default value on bulk update screen. This Value can be equal to defaultValue, as bulk update function goes through a different routine. Optional parameter, defaults to null.
- G. ExecutionResult return type. See [glossary 7](#) for details.

#### **3.2.7.3. ExecutionResult AddAttributeBool**

Used to add a Boolean type attribute to the strategy rule. Following is a list of parameters,

- A. String attributeName: Name or key of the attribute.
- B. Boolean defaultValue: Default value of the attribute.
- C. Boolean Editable – Defines weather attribute can be edited. Optional parameter, defaults to true.
- D. Boolean Visible – defines if attributes would be visible on the GUI. Optional parameter, defaults to true.
- E. Boolean isBulkUpdatable: defines if attribute would be available on the bulk update interface. Optional parameter, defaults to false.
- F. ExecutionResult return type. See [glossary 7](#) for details.

#### **3.2.7.4. ExecutionResult AddAttributeDecimalMinMax**

Used to add a Numeric type attribute to the strategy rule. This function can be used to define both MinValue, and/or MaxValue constraint. Following is a list of parameters,

- A. String attributeName: Name or key of the attribute.
- B. Decimal defaultValue: Default value of the attribute.
- C. Boolean Editable – Defines weather attribute can be edited. Optional parameter, defaults to true.
- D. Boolean Visible – defines if attributes would be visible on the GUI. Optional parameter, defaults to true.
- E. Decimal Min – MinValue constraint. Optional, defaults to Decimal.MinValue.
- F. Decimal Max – MaxValue constraint. Optional, defaults to Decimal.MaxValue.
- H. Boolean isBulkUpdatable: defines if attribute would be available on the bulk update interface. Optional parameter, defaults to false.





- I. Decimal defaultValueForBulkUpdate: Decimal value that is used to setup default value on bulk update screen. This Value can be equal to defaultValue, as bulk update function goes through a different routine. Optional parameter, defaults to 0.
- G. ExecutionResult return type. See [glossary 7](#) for details.

#### **3.2.7.5. ExecutionResult AddAttributeDecimalMinMax**

Used to add a Numeric type attribute to the strategy rule. This function can be used to define only MaxValue constraint. Following is a list of parameters,

- A. String attributeName: Name or key of the attribute.
- B. Decimal defaultValue: Default value of the attribute.
- C. Boolean Editable – Defines whether attribute can be edited. Optional parameter, defaults to true.
- D. Boolean Visible – defines if attributes would be visible on the GUI. Optional parameter, defaults to true.
- E. Decimal Max – MaxValue constraint. Optional, defaults to Decimal.MaxValue.
- F. Boolean isBulkUpdatable: defines if attribute would be available on the bulk update interface. Optional parameter, defaults to false.
- G. Decimal defaultValueForBulkUpdate: Decimal value that is used to setup default value on bulk update screen. This Value can be equal to defaultValue, as bulk update function goes through a different routine. Optional parameter, defaults to 0.
- H. ExecutionResult return type. See [glossary 7](#) for details.

#### **3.2.7.6. ExecutionResult AddAttributeCollection**

Used to add a Collection type attribute to the strategy rule. Following is a list of parameters,

- A. String attributeName: Name or key of the attribute.
- B. List<String> itemList: List of items to populate.
- C. Decimal defaultSelected: Default value selected. Should be one of the values from itemList.
- D. Boolean Editable – Defines whether attribute can be edited. Optional parameter, defaults to true.
- E. Boolean Visible – defines if attributes would be visible on the GUI. Optional parameter, defaults to true.
- I. Boolean isBulkUpdatable: defines if attribute would be available on the bulk update interface. Optional parameter, defaults to false.



- J. String defaultValueForBulkUpdate: String value that is used to setup default value on bulk update screen. This Value can be equal to defaultValue, as bulk update function goes through a different routine. Optional parameter, defaults to null.
- F. ExecutionResult return type. See [glossary 7](#) for details.

#### 3.2.7.7. ExecutionResult UpdateStrategyAttributes

This function can be used to bulk update list of attributes defined in a strategy rule,

- A. List<AttributeUpdate>: Collection of AttributeUpdates instances. See [glossary 8](#) for more details.
- B. ExecutionResult return type. See [glossary 7](#) for details.

#### 3.2.7.8. String GetAttribute

Retrieves String value of String or Collection type attribute by its attributeName.

Out errorMessage returns an error in case attribute were not found, or were not of type of String as defined by the strategy rule.

#### 3.2.7.9. Boolean GetAttributeBool

Retrieves Boolean value of Boolean type attribute by its attributeName.

Out errorMessage returns an error in case attribute were not found, or were not of type of Boolean as defined by the strategy rule.

#### 3.2.7.10. Decimal GetAttributeDecimal

Retrieves Decimal value of Decimal type attribute by its attributeName.

Out errorMessage returns an error in case attribute were not found, or were not of type of Decimal as defined by the strategy rule.

Example code,

```
var error = String.Empty;
var result = GetAttribute("Symbol", out error);
if (String.IsNullOrEmpty(error))
    LogInfo($"Value of attribute by name Symbol: {result}", true);
else LogError($"Value of attribute by name Symbol not found with error: {error}", true);
```



### 3.3. Order Execution

MFT order execution methods are used to execute a single token, or multi token orders. All methods return type is of type ExecutionResult documented in [Glossary 7](#).

#### 3.3.1. Buy/Sell

Use this API method to execute Single Security Orders. Executes an order for SubscribedSingleSecurity. Following is a list of parameters,

- A. OrderAlgoType: See [glossary 9](#) for more details.
- B. UserTag: Message to print on orders, and trades.
- C. ExecutionResult return type. See [glossary 7](#) for details.
- D. **Constraint:** Security to trade must be subscribed.

**Note:** Sell cannot be called once position is built using Buy until rule is squareoff and vice versa.

e.g.

```
if (StrategyTradingMode() == TradingMode.SingleSecurity)
{
    var exResult = Buy(OrderAlgoType.MKT, "SingleSecurity");
    if (exResult != null)
    {
        if (exResult.Result == ExecutionResultType.SUCCESS)
            LogWarn($"Executed BUY on SecDesc: {SubscribedSingleSecurity._secDesc}");
        else LogError($"failed to execute BUY on SecDesc:
{SubscribedSingleSecurity._secDesc}, error: {exResult.ExecutionMessage}");
    }
    else LogInfo($"Failed to execute BUY on SecDesc:
{SubscribedSingleSecurity._secDesc}");
}
else LogInfo($"Invalid Mode: Can not execute Buy in : {StrategyTradingMode()}
Mode");
```

#### 3.3.2. Submit/Reverse

Use this API method to execute Basket Orders (Rules subscribed as Basket). Following is a list of parameters,

- A. OrderAlgoType: See [glossary 9](#) for more details.
- B. UserTag: Message to print on orders, and trades.
- C. ExecutionResult return type. See [glossary 7](#) for details.
- D. **Constraint:** Basket to trade must be subscribed.



Refer to [1.2.1.1](#) for BasketSecurity Class, and [2.3.2](#) AddBasket method's example. Subscribed basket has 2 components,

SBIN M1 Future, with Ratio 1, and Direction = Sell.

TCS M1 Future, with Ratio 1, and Direction = Buy.

Also refer to [SizeType](#) in [3.1.8.1](#). Considering size type is set to Lots, and base quantity is set to 2 lots.

Following call to Submit – executes Sell 2 lots of SBIN M1 Future, and Buy 2 lots of M1 SBIN Future.

A call to Reverse - executes Buy 2 lots of SBIN M1 Future, and Buy 2 lots of M1 TCS Future.

**Note:** MultiReverse cannot be called once position is built using MultiSubmit until rule is squareoff and vice versa.

```
if (StrategyTradingMode() == TradingMode.Basket)
{
    if(null != SubscribedBasket)
    {
        var exResult = Submit(OrderAlgoType.MKT, "Sell 2 lots of SBIN future, and buy 2
lots of TCS Future");
        if (exResult != null)
        {
            if (exResult.Result == ExecutionResultType.SUCCESS)
                LogWarn($"Executed Submit on SecDesc: {SubscribedBasket.ToString()}");
            else LogError($"failed to execute Submit on SecDesc:
{SubscribedBasket.ToString()}, error: {exResult.ExecutionMessage}");
        }
        else LogInfo($"Failed to execute Submit on SecDesc:
{SubscribedBasket.ToString()}");
    }
    else LogInfo($"Invalid Security: Basket not subscribed.");
}

else LogInfo($"Invalid Mode: Cannot execute Submit in : {StrategyTradingMode()}
Mode");
```

### 3.3.3. MultiSubmit/MultiReverse

Use this API method to execute MultiSecurity Orders (Rules subscribed as MultiSecurity). Following is a list of parameters,

- A. BasketSecurity array: array of basket securities to execute.
- B. OrderAlgoType: See [glossary 9](#) for more details.
- C. UserTag: Message to print on orders, and trades.
- D. ExecutionResult return type. See [glossary 7](#) for details.
- E. **Constraint:** Security to trade must be subscribed.



Refer to [1.2.1.1](#) for BasketSecurity Class, and [2.3.3](#) AddMultiSecurity method's example. Subscribed MultiSecurity has 2 Legs,

SBIN M1 Future, with Ratio 1, and Direction = Sell.

TCS M1 Future, with Ratio 1, and Direction = Buy.

Also refer to [SizeType](#) in [3.1.8.1](#). Considering size type is set to Lots, and base quantity is set to 2 lots.

Following call to MultiReverse – executes Sell 2 lots of TCS M1 Future.

**Note:** Only one leg was used to execute MultiSubmit. MultiSubmit, & MultiReverse allows selecting user defined set of legs to Submit, or Reverse. However API doesn't allow building using reverse in case submit were executed and vice versa. So for example, once MultiSubmit is called to execute Sell 2 lots of TCS M1 future, API won't allow a call to MultiReverse Buy 2 lots of TCS M1 future, until square off is called.

A call to MultiSubmit - executes Buy 2 lots of TCS M1 Future.

```
if (StrategyTradingMode() == TradingMode.MultiSecurity)
{
    if(null != SubscribedMultiSecurities)
    {
        var toSubmitSecurities = SubscribedMultiSecurities.Securities.Where(x => x.LegName
== "TCSLeg");
        if(null != toSubmitSecurities)
        {
            var exResult = MultiReverse(toSubmitSecurities.ToArray(), OrderAlgoType.MKT,
"Sell 2 lots of TCS future");
            if (exResult != null)
            {
                if (exResult.Result == ExecutionResultType.SUCCESS)
                    LogWarn($"Executed MultiSubmit on SecDesc:
{toSubmitSecurities.ToString()}");
                else LogError($"failed to execute BUY on SecDesc:
{toSubmitSecurities.ToString()}, error: {exResult.ExecutionMessage}");
            }
            else LogInfo($"Failed to execute BUY on SecDesc:
{toSubmitSecurities.ToString()}");
        }
        else LogInfo($"Invalid Security: Leg with name InfyLeg not subscribed.");
    }
    else LogInfo($"Invalid Security: Basket not subscribed.");
}

else LogInfo($"Invalid Mode: Can not execute Buy in :
{StrategyTradingMode()} Mode");
```



Following call to MultiSubmit – executes 2 lots of Sell SBIN M1 future, and 2 lots of Buy TCS M1 future.

```
if (StrategyTradingMode() == TradingMode.MultiSecurity)
{
    if(null != SubscribedMultiSecurities)
    {
        var toSubmitSecurities = SubscribedMultiSecurities.Securities;
        if(null != toSubmitSecurities)
        {
            var exResult = MultiSubmit(toSubmitSecurities.ToArray(), OrderAlgoType.MKT, "Sell 2
lots of TCS future");
            if (exResult != null)
            {
                if (exResult.Result == ExecutionResultType.SUCCESS)
                    LogWarn($"Executed MultiSubmit on SecDesc:
{toSubmitSecurities.ToString()}");
                else LogError($"failed to execute BUY on SecDesc:
{toSubmitSecurities.ToString()}, error: {exResult.ExecutionMessage}");
            }
            else LogInfo($"Failed to execute BUY on SecDesc: {toSubmitSecurities.ToString()}");
        }
        else LogInfo($"Invalid Security: Leg with name InfyLeg not subscribed.");
    }
    else LogInfo($"Invalid Security: Basket not subscribed.");
}
else LogInfo($"Invalid Mode: Can not execute Buy in : {StrategyTradingMode()} Mode");
```

### 3.3.4. MultiSubmit/MultiReverse with BasketSize

Use this API method to execute orders on multiple legs on a rule subscribed in MultiSecurity mode,

- A. BasketSize array: array of BasketSize that contains reference to a BasketSecurity, and int size (quantity) to execute. Refer to [section 10](#) for more details.
- B. OrderAlgoType: See [glossary 9](#) for more details.
- C. UserTag: Message to print on orders, and trades.
- D. ExecutionResult return type. See [glossary 7](#) for details.

**Note:** This method can be used to execute orders in any direction. In case security is not subscribed to the rule, then trade rule appears on Unsubscribed Tab on StrategyMonitor.



In the example below, a strategy rule is executing 1 lot of SBIN M1 future Sell, and buy equivalent amount (in premium) of SBIN Stock. Calling MultiReverse in the same example executes Buy 1 lot of SBIN M1 future, and sell an equivalent premium equivalent in SBIN Stock.

```
//Execute SBIN M1 Future, 1 lot, Buy, and Sell SBIN Stock equal to Premium Bought.
String errF = String.Empty;
String errS = String.Empty;
var sbinFuture = GetSecurity("SBIN", "M1");
var sbinStock = GetSecurity("SBIN");
if (null != sbinFuture) {
    var sbinFutLeg = new BasketSize()
    {
        Security = new BasketSecurity(sbinFuture, 1, BuyOrSell.SELL, false, "SBINFut"),
        Size = sbinFuture._lotSize
    };
    var mbpFut = GetGlobalMBPTick(sbinFuture._token, out errF);
    var mbpStock = GetGlobalMBPTick(sbinStock._token, out errS);
    if (String.IsNullOrEmpty(errF) && String.IsNullOrEmpty(errS) && mbpFut.LTP != 0 &&
        mbpStock.LTP != 0)
    {
        var futPrem = mbpFut.LTP * sbinFuture._lotSize;
        var stockSize = (int)futPrem / mbpStock.LTP;
        var sbinStockLeg = new BasketSize()
        {
            Security = new BasketSecurity(sbinStock, 1, BuyOrSell.BUY, false, "SBINStock"),
            Size = sbinFuture._lotSize
        };

        var securityExecLog = $"SBIN Fut 1 lot, and SBIN Stock: {sbinStockLeg.Size}";
        var result = MultiSubmit(new BasketSize[2] { sbinFutLeg, sbinStockLeg },
            OrderAlgoType.MKT, $"Executing {securityExecLog}");
        if (result != null)
        {
            if (result.Result == ExecutionResultType.SUCCESS)
                LogWarn($"Executed MultiSubmit on SecDesc: {securityExecLog}");
            else LogError($"failed to execute MultiSubmit on SecDesc: {securityExecLog}, error:
{result.ExecutionMessage}");
        }
        else LogInfo($"Failed to execute MultiSubmit on SecDesc: {securityExecLog}");
    }
    else LogInfo($"Security tick not found.");
}
```





### 3.3.5. SquareOff

Use this API method to square off all existing positions,

- A. OrderAlgoType: See [glossary 9](#) for more details.
- B. UserTag: Message to print on orders, and trades.
- C. ExecutionResult return type. See [glossary 7](#) for details.

### 3.3.6. SquareOffPartial

Use this API method to Partial Square off existing positions,

- A. TokenSize Array: See [glossary 11](#) for more details.
- B. ExitMode: Quantity to see Size value in Token size as Quantity to trade, or Percent to compute size as % of existing position.
- C. OrderAlgoType: See [glossary 9](#) for more details.
- D. ExecutionResult return type. See [glossary 7](#) for details.

Example below, has a function from a rule that trades in Straddles, and has positions in both call & Put legs. Rule is configured to square off quantity = Base Quantity configured in the rule in case active position is less than Total active quantity in the token, or all the remaining quantity in case active position is less than total active quantity in the token.

```
private Tuple<Boolean, String> PartialSqOffBasket(string orderMsg = null)
{
    Tuple<Boolean, String> result = null;
    List<TokenSize> bas = new List<TokenSize> ();
    BasketSecurity call = SubscribedMultiSecurities.Securities.Where(x => x.LegName ==
"Call").FirstOrDefault();
    BasketSecurity put = SubscribedMultiSecurities.Securities.Where(x => x.LegName ==
"Put").FirstOrDefault();
    if (call != null && put != null)
    {
        var qty = BaseQty() * call.Security._lotSize;
        var callQTY = Math.Abs(Position(call.Token));
        var putQTY = Math.Abs(Position(put.Token));

        if(callQTY > 0 && qty <= callQTY)
        {
            bas.Add(new TokenSize() { Token = call.Token, Size = qty, OrderMessage =
orderMsg });
        }
        else if(callQTY > 0 && qty >= callQTY)
        {
            bas.Add(new TokenSize() { Token = call.Token, Size = callQTY, OrderMessage =
orderMsg });
        }
    }
}
```



```
        if (putQTY > 0 && qty <= putQTY)
        {
            bas.Add(new TokenSize() { Token = put.Token, Size = qty, OrderMessage =
orderMsg });
        }
        else if (putQTY > 0 && qty >= putQTY)
        {
            bas.Add(new TokenSize() { Token = put.Token, Size = putQTY, OrderMessage =
orderMsg });
        }

        if(bas.Count > 0)
        {
            result = GetExecutionResult(SquareOffPartial(bas.ToArray(), ExitMode.Quantity,
OrderAlgoType.MKT));
        }
        else
        {
            result = new Tuple<bool, string>(false, $"CQTY: {callQTY}, PQTY: {putQTY},
Partial SQ Failed, failed to compute quantity!");
        }
    }
    else
    {
        result = new Tuple<bool, string>(false, "Call or Put are null!");
    }

    return result;
}
```

## 4. Utility Methods & Events.

Utility method & events are set of API functions, and notification functions that assist in trading.

### 4.1. DateTime ResidualPositionEntryDate

In case a rule is configured to carry forward positions – this API function returns the date when entry happened in case position is greater than 0, else default date i.e. DateTime.MinValue.



#### **4.2. Boolean IsOrderExecInProgress**

Returns true in case an order execution is in progress.

#### **4.3. Int32 UserID**

Returns logged in userID.

#### **4.4. Int32 StrategyID**

Current rule's ID.

#### **4.5. String StrategyName**

Current rule's name.

#### **4.6. Boolean StrategyState**

Returns true if current rule is enabled or else false.

#### **4.7. Decimal DeltaPerLot**

Net Delta Per lot for all positions in the rule.

#### **4.8. Decimal EntryPrice(Int32 Token)**

Entry price of token in case holding a position, else zero.

#### **4.9. Decimal Expenses(Int32 Token = 0)**

Total expense of token, default value of token is 0 i.e. if not supplied this function return total expenses of the rule.

#### **4.10. Decimal OpenExpenses(Int32 Token=0)**

Total expense of open positions in a token, default value of token is 0 i.e. if not supplied this function return total open expenses of the rule.

#### **4.11. Decimal PnL(Int32 Token=0)**

Total PnL of open positions in a token, default value of token is 0 i.e. if not supplied this function return total open PnL of the rule.

#### **4.12. Decimal TotalPnL(Int32 Token=0)**

Total PnL of all the positions in a token, default value of token is 0 i.e. if not supplied this function return total PnL of the rule.

#### **4.13. Int64 Position(Int32 Token=0)**

Position of token if greater than 0, or else rule's.



#### 4.14.Int64 BuyPosition(Int32 token = 0)

Total Buy Positions in a token if token greater than 0, or else total buy positions in rule.

#### 4.15.Int64 SellPosition(Int32 token = 0)

Total Sell Positions in a token if token greater than 0, or else total buy positions in rule.

#### 4.16.Dictionary<Int64, Int64> AllPosition()

Net positions in all tokens. Tokens are represented in Int64 here. In future all tokens will be of type Int64.

#### 4.17.Boolean IsNetPositionZero()

Returns true in case there is a net position in any instrument traded from the rule, else false.

#### 4.18.void SetStrategyColor(System.Drawing.Color color)

Paints a rule's row on strategy monitor to parameter color.

#### 4.19.void SetStrategyNameColor(System.Drawing.Color color)

Paints a StrategyName column for the rule on strategy monitor to parameter color.

#### 4.20.SendStrategyDataToMonitor

Displays a custom value on a column cell. Please note that if there is strategy1, and strategy2 both pointing to column1 for example, last value overrides the data. Following is a list of attributes associated with this function,

- A. ExecutionResult: See [glossary 7](#) for more details.
- B. destinationColumn: of type StrategtGridColumnNames enum with values Column1- Column50.
- C. desiredColumnName: of type String - name of destination column.
- D. value: of type String - value to be printed.



## 4.21. LogError/LogWarn/LogInfo

Prints a set of data as Information, warning or error to file system, and UI.

- A. Message: of String type, Data to print.
- B. UI: if message is to be printed on miscellaneous window, set it to true.

## 4.22. AddOrUpdateValueToDB Overloads

Saves a custom set of values to DB.

- A. OverLoad1: String Key, String Value
- B. OverLoad2: String key, Decimal Value
- C. OverLoad3: String key, Boolean Value
- D. OverLoad4: List of String key, and String Values. Use this to bulk update values to DB.

e.g.

```
kv = new List<KeyValuePair<string, string>>();  
kv.Add(new KeyValuePair<String, String>("CallReEntryPrice",  
    $"{DateTime.Today.ToString("yyyy-MM-dd")}|{CallReEntryPrice}"));  
kv.Add(new KeyValuePair<String, String>("PutReEntryPrice",  
    $"{DateTime.Today.ToString("yyyy-MM-dd")}|{PutReEntryPrice}"));  
AddOrUpdateValueToDB(kv);
```

OR

```
AddOrUpdateValueToDB("ReEntryAttempted",  
    $"{DateTime.Today.ToShortDateString()}|{ReEntryAttempted}");
```

## 4.23. GetDecimalValueFromDB/ GetBooleanValueFromDB/

### GetStringValueFromDB(String Key)

Retrieves corresponding datatype value from DB based on it's key. In order to retrieve the value, right datatype value must be saved with corresponding overload, or a run cast is required. E.g.

```
String cep = GetStringValueFromDB("PutReEntryPrice");  
String pep = GetStringValueFromDB("CallReEntryPrice");
```



#### 4.24. void Schedule(System.Action<bool> scheduledAction, int hours, int minutes)

Schedule a function to run at a given time during the day time, and minute (must be during trading hours). Function to be scheduled should be of void type, and accept a boolean Parameter (isFirstRun – which is true, when function is executed for first time during the day)

e.g.

```
Schedule(FireOrderScheduled, 13, 10);
```

```
private void FireOrderScheduled(Boolean firstTime)
{
    if (StrategyState())
        EnterBasket("Scheduled");
    else LogInfo("Entry: Rule not enabled!", true);
}
```

#### 4.25. DisableStrategy()

Used to disable a strategy at runtime.

#### 4.26.Void OnStateChanged(StateChangeType type)

Rule is notified when one of the following events is triggered,

##### 4.26.1. SetDefaults

First ever event that is triggered, when rule is loaded from the database, however is still processing. This event can be used to set default values in the strategy if any required.

##### 4.26.2. StrategyStateChanged

This event is triggered when rule is enabled or disabled.

##### 4.26.3. StrategyModified

This event is triggered when rule is modified from the builder.

##### 4.26.4. StrategyModifiedAPI

This event is triggered when rule is modified while saving attributes from the rule using AddAttribute methods documented in [3.2.7](#).



#### 4.26.5. HistoryBarsArrived

Triggers after History QDAP data (documented in [2.5](#)) for all the subscribed securities is downloaded by the rule.

#### 4.26.6. LiveBarArrived

Triggers after Live QDAP data (documented in 2.5) for all the securities by the rule for the first time.

#### 4.26.7. FilterArrived

Triggers after Live filter data arrives for all the subscribed securities for the first time.

#### 4.26.8. PositionDownloaded

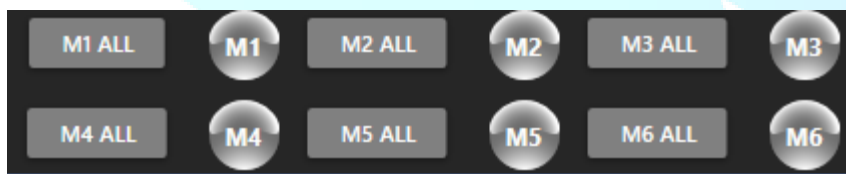
Triggers after rule finishes downloading positions data.

#### 4.26.9. LiveTickArrived

Triggers as soon as first live tick arrives on any subscribed security.

#### 4.26.10. ManualTrigger1....6

These events are fired when one of the corresponding (M1-M6) buttons on StrategyMonitor is pressed.



### 4.27. OnESQTrades(MFTTradeModel engineTrades)

Triggers when rule is squared off due to an RMS, or PNL Square off triggers squaring off positions automatically. Trades stream into this function one by one. For example – trades getting squared off due to StopLoss.





## 4.28. void OnStrategyTradeUpdate(MFTOrderType type, List<TradeInfo> trades, Boolean isPartiallyFilled, Boolean attemptsExhausted)

Triggers when rule's order fired using one of the API methods in section [3.3](#) is complete.

### 4.28.1.1. Parameters

- A. MFTOrderType: Original order that triggered this event. Could be one of the following types,
- SUBMIT
  - REVERSE
  - MULTISUBMIT
  - MULTIREVERSE
  - BUY
  - SELL
  - PARTIALSQOFF
  - MANUALSQOFF
  - SQOFF
- B. List<TradeInfo>: List of traded legs. For list of attributes in TradeInfo, see [Glossary 12](#)
- C. IsPartiallyFilled: Boolean, indicated if all the orders for all the legs were filled, true if yes.
- D. attemptsExhausted: Boolean, indicating in case of market orders that 5 attempts were made to execute the entire lot of orders. This an obsolete attribute, still holds valid, but will be removed in future.

## 4.29. OnTimer

OnTimer function gets call at frequency set up in 3.1.16. E.g. if it's set to .25, OnTimer gets called every 250 MS.

## 4.30. void OnPortfolioRiskUpdate(MFTRisk risk)

This event streams risk parameters of the USER (i.e. for the entire trading portfolio that consists of trades happened on all the rules, and Tokens).

### 4.30.1.1. Parameters

- A. MFTRisk: class that contains risk attributes-see [Glossary 13](#).



### 4.31. void OnRuleRiskUpdate(MFTRisk risk)

This event streams risk parameters of the RuleName (i.e. for the trades execute by the StrategyName: Eagle).

#### 4.31.1.1. Parameters

B. MFTRisk: class that contains risk attributes-see [Glossary 13](#).

### 4.32. void OnStrategyRiskUpdate(MFTRisk risk)

This event streams risk parameters of the Rule (i.e. for the trades execute by the current rule ID).

#### 4.32.1.1. Parameters

C. MFTRisk: class that contains risk attributes-see [Glossary 13](#).

## 5. Sample Strategy.

KTStrategy is a simple strategy rule that Subscribes to Future, and ATM Call Option, and builds short call, and square off every 10 seconds.



## Glossary

### 1- GetStrike Function index or Index List parameters:

- For ATM use Index = 0, for **OTM** Call option use index > 0 e.g. "1" means one strike above ATM, similarly for **ITM** Call option use index < 0 e.g. "-1" means one strike below ATM.
- indexList: e.g. new int[3] {-1,0,1} means one strike below atm, at ATM, and above atm.
- Can be both +ve or -ve.

### 2- StrikeGap

- Strike Gap in rupees while filtering out strikes of a given symbol to find the Strike at index. So for example

`var atm = GetStrike(1234, 0, 0, out );` could return ATM strike = 19000 or 19050 for Nifty's future's with Token = 1234, and strike gap = 0. Strike Gap = 0 instructs API to consider all the strikes to find the ATM strike. Conversely, using strike gap = 100 filters out strikes with 50 rupees. Therefore, the ATM strike would either be 18900, 19000 or 19100 based on the price of Nifty future.

- Positive value.

### 3- out errorMessage, or out error, or out err

- This parameter is populated with diagnostic error message in case something went wrong while executing a given function.

### 4- Int32 expiry, or String expiry

- Int32 expiry: Use Int32 \_expiry parameter in Security class.
- String expiry: Use M1, M2, M3, or W1, W2...W15. M refers to month, W refers to week. Number 1,2...15 - refers to nth expiry from now. For example, M1 means current week, W2 means second week from now.
- Positive value.

### 5- String Symbol

- Refers to String identifier of the security. For example SBIN is symbol for State Bank of India.

### 6- GetParamValue & IndicatorName in GetIndicatorBar functions.

- There are 2 set of names for Indicators provided by Indicators API functions available in MFT.
  - A. IndicatorName set on Trading UI.
  - B. Indicator parameter name configured documented in IndicatorDefinitions.pdf.



With GetIndicator function, IndicatorName function is used to get the bar. Once bar is retrieved, individual values associated with Indicator can be retrieved using parameter name using GetParamValue function. For example, parameter associated with RSI15 (Which is a unique name used to define a 15 days interval RSI indicator) is RSI as documented in IndicatorDefinitions document. Similarly if we setup BBands, we could retrieve values associated with UpperBand, LowerBand and MiddleBand using "UB", "LB" & "MB" respectively.

#### 7- ExecutionResult

- Return value of functions returning this type.
- Execution message – String Type, contains any validation error if exists in case of an error.
- Result – ExecutionResultType enum type.
  1. VALIDATION – Function called failed.
  2. DISABLESOURCE – Function call resulted in strategy disabled.
  3. SUCCESS – Function call executed successfully.
  4. NONE – Unknown execution type.

#### 8- AttributeUpdate

- Type used to bulk update strategy attributes.
  1. AttributeName: name of the attribute to be updated.
  2. AttributeValue: value of the attribute in string format.
  3. SendToEngine: True or False defines if a copy of the value is to be sent to Strategy Engine.

#### 9- OrderAlgoType

1. MKT: executes at current Market Price.
2. Best Algo: Buys at best bid, or sells at best Ask. Uses quoting, there is a time out of 5 seconds (default), or number of modifications (default: 5), if either fails – order executes at Market.

#### 10-BasketSize

1. BasketSecurity: Security to trade.
2. Size: Quantity to trade. Should be in lotSize. E.g. LotSize of BankNifty option is 25, then in order to execute 2 lots, size should be equal to 50. For Single Stocks, it's equal to number of stocks to buy or sell.

#### 11-TokenSize

1. Token: Token to trade.



2. Size: Quantity to trade. Should be in lotSize. E.g. LotSize of BankNifty option is 25, then in order to execute 2 lots, size should be equal to 50. For Single Stocks, it's equal to number of stocks to buy or sell.
3. OrderMessage: Message to print on trade & Order log.

## 12-TradeInfo

Following is a list of attributes,

1. Token: Traded Token.
2. StrategyID: Rule ID that triggered the trade.
3. Symbol: Traded Symbol
4. TradedTime: Time of trade.
5. FillPrice: TradedPrice.
6. FillQuantity: Traded quantity.
7. Side: 1 for Buy, -1 for Sell.
8. UserTag: String message (UserTag) sent while sending the order using any of the API functions documented in section [3.3](#).

## 13-MFTRisk

Following is a list of attributes,

1. PortNetDayPnL: Average day's PNL of all positions minus day expenses.
2. PortNetTotalPnL: Average total PNL of all positions minus Total expenses.
3. PortNetRunningPnL: Average total PnL of open positions minus running expenses
4. PortHistoricalPnL: PNL of carried over positions based in EOD LTP of y'day.
5. Greeks (PortDelta, PortGamma etc.): Total open position Greeks.
6. PortExpenses: Total Expenses of all positions.
7. Exposure: Total Exposure of Future, Options, and CM.
8. Turnover: Total Turnover of Future, Options, and CM.
9. PortSell/Buy Quantity: Total buy/Sell Quantity.



QUANTIZER

QUANTIZER RESEARCH PRIVATE LIMITED

Reimagining Quant Trading

