IQL User Manual

[First Draft]

Table of Contents

Introduction

For Technical Analysis, many different indicators, candlesticks, and various statistical concepts are used which can prove to be advantageous if used adroitly. Many traders are alienated from these complex mathematical concepts and unable to fully leverage the possible advantages of technical analysis.

The Investar Custom Screener allows the use of both IQL(Investar Query Language) and IEL (Investar Easy Language) in a hybrid mode, IQL Only mode and IEL Only mode.

IEL (Investar Easy Language) is a proprietory language which allows the use of English to create user-friendly queries.

IQL (Investar Query Language) is a proprietary language used in Investar software for technical analysis. IQL (Investar Query Language) provides a simple and user-friendly medium to enable the users to use the inbuilt indicators, candlesticks, price action, and other many technical analysis tools by minimal coding. IQL is easy to pick up for those who are familiar with the C-language.

Reading Conventions

Many technical terms used in this document may mean different from other available academic papers, books and other reference material on the Internet. Hence, please refer this section to avoid any kind ambiguity of understanding the meaning of the word.

- Query: The word ‘query’ means the last ’BUY’/’SELL’ variable used in the IQL code. This “BUY/SELL” variable represents the whole structure of the query user wants to execute.
- SubQuery: The term 'SubQuery' suggests the expressions which are separated by "OR" and "AND" operators. The return type of "SubQuery" will always be Boolean.

  E.g. : BUY = (IQL1 OR IQL2) And (CND2 OR IND0)

Based on the above example, the "BUY" variable represents the whole structure of the Query which will be executed by the IQL compiler. The variables "IQL1", "IQL2", "CND2" and "IND0" represent the SubQuery.

Tokens

Tokens are the smallest unit and basic unit of the language. Different types of tokens are accepted in IQL:

- Identifiers
- Operators
- Semicolon (;)
- Decimal Point (.)
- Parenthesis "(" and ")"
- Numeric Constants (Integers and Double)

Comments

Commenting allows you to disable a specific section of code without deleting that code. The commented code is not detected by the compiler. Commenting in IQL is similar to C language which allows multiline and single line comments. Comments between symbols "/*" and "*/" will be considered as the multiline comments and the lines beginning with "//" will be considered a single line comment.

NOTE: Comments are only allowed in IQL mode, which can be accessed from Tools -> Options -> Custom Scan tab and select 'IQL mode'.

```
/*
 Multiline Comments
*/

// Single Line comments
```

Separators or Punctuators

IQL provides some special symbols which must be used for the specific purpose.

- Semicolon (;) - To end the statement or query.
- Comma (,) - To separate different arguments in the function call
- Square Brackets ('[' and ']') - To specify the time frames value or to specify the index value of an array.
- Parenthesis '(' and ')' - To assign the precedence of operation in binary expressions statements.
- Dot (.) - As decimal point in decimal numbers.

Identifiers

Identifiers are the names (user-defined or inbuilt) which represent some specific entity in the code. The information may vary based on the context in which the identifier is used.

The identifiers can be used as function names, variable names, inbuilt constant variables, and as arguments for the functions.

Example 1: In the below given example, different identifiers are used in different contexts

```
IQL1 = Close < Low;
IND2 = EMA (High, 50);
BUY = IQL OR IND2;
```

The identifiers **IQL1**, **IND2,** and **BUY** are used as output variables. **Close, Low** and **High** are the inbuilt identifiers. **EMA** is the identifier which represents the name of the function which computes the EMA based on the parameters passed to it.

Each identifier may have different return types depending on what they represent. For instance, the **Open, Close, Low, High** and **Volume** returns the array of double type and the **Candlestick** patterns return the Boolean type.

NOTE: All the keywords used in IQL are not case sensitive.

Boolean Data Type

The Boolean data type is used to represent if a particular condition is true or not. In IQL, each variable which begins with keyword IQL, IND, ASR, AT, PA and CND and moreover variable "BUY" and "Sell" must have Boolean return types.

Identifiers '**true**' and '**false**' are used to represent Boolean states in IQL language.

Example:

```
IQL1 = Hammer == true;   // for Boolean comparisons.
IQL1 = Hammer == false; // if we want to compare false condition
NOTE: In current version of IQL, “!=” is not supported.
```

Using inbuilt Functions

IQL provides number of inbuilt functions, which represents indicators, price action and operations, so that users can directly use them in their queries. The return types of these functions may be different based on their feature.

Example: The following example uses inbuilt function whose return types are different.

```
IQL1 = Cross (EMA (Close, 20), EMA (Close, 5));
```

Example: Close on current bar is crossing above close on previous bar
IQL1 = Cross(C, C[1]);

In above example, the Cross function returns bool value, while the EMA function returns array of numbers (double data type).

Currently, IQL does not support user-defined functions.

Operations

Users can perform different operations between two or more expressions in order to implement complex queries. These operations may be arithmetic, logical or inbuilt functions.

Arithmetic operations

- Multiplication (*)
- Addition (+)
- Subtraction (-)
- Division (/)

Logical Operations

- Greater than (>)
- Less than (<)
- Equals to (= =)

Functions

- Cross(op1, op2) : op1 is crossing above op2, or op2 is crossing below op1.
- BreakOut(op1, op2): op1 is breaking out above op2
- BreakDown(op1, op2): op1 is breaking down below op2

Bitwise operations and conditional statements are not supported in IQL currently.

NOTE: Arithmetic operations are only possible in IQL mode or IQL variables (IQL variables are discussed in upcoming chapters)

Basic Structure

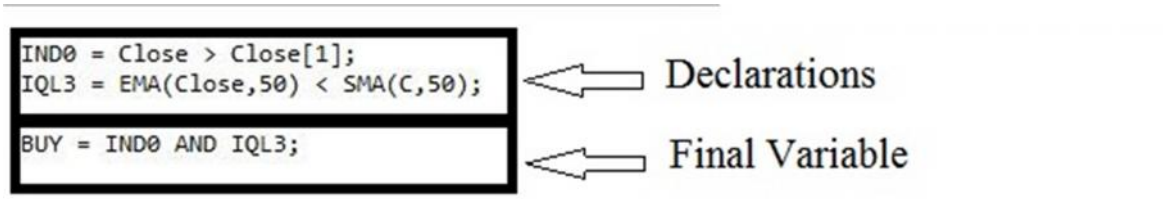
The language is based on using inbuilt functions and identifier directly and applying different operations on them. The language uses the syntax similar to C.

The structure of the language is divided into two sections:

1. Declarations
2. Final variable

Declarations

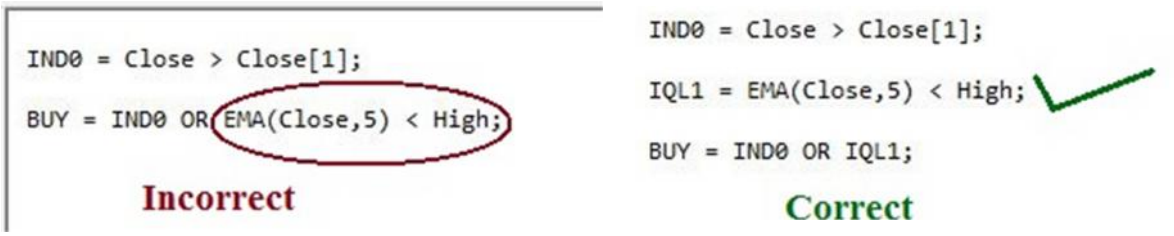
Declarations section will have all the variable assignment declarations and the **Final variable** section will have the variable present in declaration section.



Final Variable (BUY / SELL variable)

In the final variable section we can use the above-declared variables only, and we can only perform AND and OR operations between them. Users are allowed to use brackets to set the priority of AND and OR operations by using the parentheses '(' and ')'**. The last variable should always represent the final structure of the query.** The name of the final variable may be "BUY" or "SELL" which represents bullish and bearish custom scan query respectively.

One cannot directly use inbuilt functions or identifiers in the final variable. Consider the below example.



Modes

There are total three modes in which the Custom Scans feature of Investar can be used:

- IQL and IEL mode
- IEL mode
- IQL mode

IQL and IEL mode

This is the default mode for custom scan, and preferable for beginners/intermediate users. On clicking the **"Manage Scans -> Custom Scans -> Create Custom Scan"** you will be displayed a GUI for creating custom scans which enables you to create the custom scans using the simple dropdown lists. The window will also have a toggle switch which can to used to simply move to IQL editor.

On clinking the toggle switch, the GUI window will be replaced by a code editor which can be used to write IQL code. On switching from IEL window to IQL window, the query created in GUI will also be converted into equivalent IQL code.

Refer the below given example

Illustration 1: IEL Window

Illustration 2: IQL Window

There are many different features supported by the IEL mode and those same features can be used in IQL language also.

Such features are:

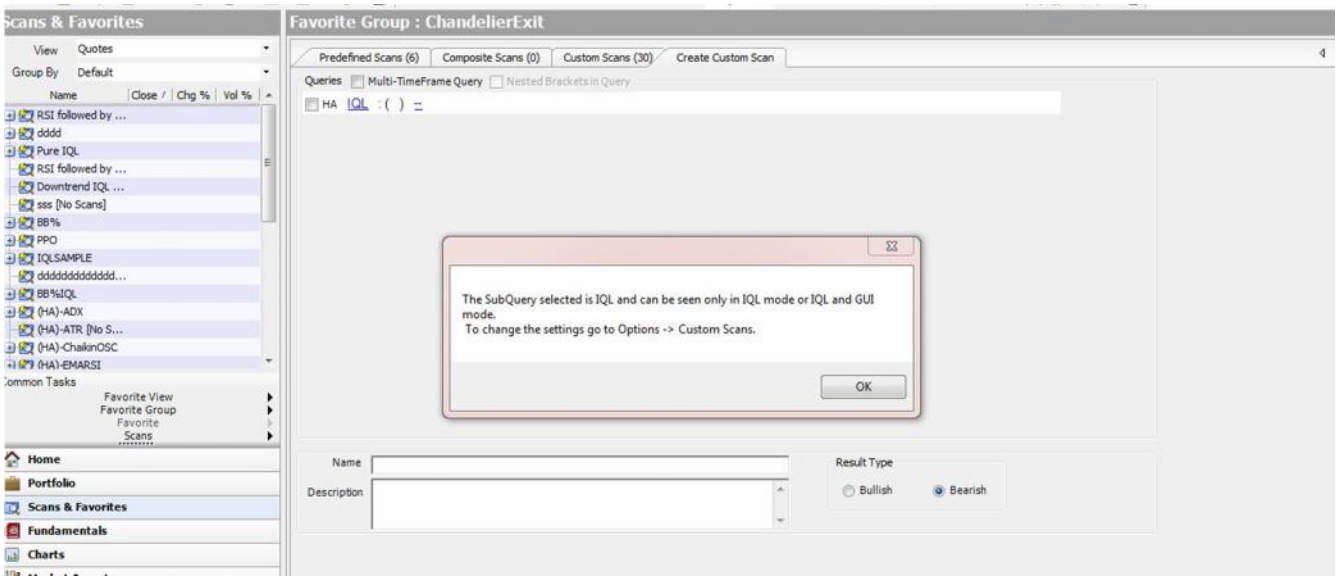
- Multi Time Frame
- Brackets
- Disable Query

All the above features will be discussed in further chapters.

IEL Mode

IEL mode is the basic way to create simple queries. This mode provides the dropdown lists which automatically displays relevant option to choose from. In GUI mode, every single line represents a subquery and they are logically connected using **AND** and **OR** operators.

In case someone tries to access or change the IQL subquery in **IEL Only** mode, an error message will be displayed.



IQL Mode

IQL mode is specially designed for advanced users who want to exploit the arithmetic operations and complex query structures. Users can use the different indicators, price action, Auto SR, inbuilt candlestick patterns and many other operations like break out, break down and cross over.

One can enable IQL mode the same way as the other two above modes. Go to "Tools -> Options " and in the Custom Scan Tab, select the IQL option.

In the IQL mode, we can directly use the inbuilt functions and identifier constants to write the code.

Language features

The main aim of this language is to provide simplified features which can be used by the user of any technical background. The use of IQL will be mainly based on variables (user-defined or inbuilt) and inbuilt functions.

IQL supports the following syntax:

- Inbuilt Identifiers
- Local variables
- Inbuilt functions
- Binary Operations
- Disable Subquery
- IQL File import

Inbuilt Identifiers

IQL provides inbuilt constant identifiers which can be used directly in the code. Refer the table of Identifiers for the detailed information. Below is an example which explains how different identifiers can be used in different ways.

Example 1: twice the Open on current bar is smaller than Low on Current bar.

```
IQL1 = 2 * Open < Low;
```

Here, the Open and Low are the array which can be accessed till 5^{th} element.

Example 2: Closing on the current bar is less than Closing on the previous bar.

```
IQL1 = Close < Close[1];
```

```
Here, "Close[1]" represents the  previous closing value, Close[2] is closing value of two bars back. The max index value possible is 5.
```

Example 3: Some values represent the array of numbers while some identifier represents the Boolean. Same is the case with the candlestick patterns.

```
IQL1 = Hammer == true; // here "Hammer" represents the candlestick pattern at current bar.
```

If the candlestick is forming then, the value is true otherwise it’s false.

Local Variables

Local variables are the variables which can be declared locally and can be used in the code. These variables are more often used in large code to avoid repetitions.

Example 1 : Variable 'a' and 'b' are the local variables which are used in the code.

```
a = EMA(Close, 5);
```

```
b = EMA(Close, 20);
```

```
IQL1 = Cross(a,b);
```

```
BUY = IQL1;
```

When you run this code, the values 'EMA(C, 5)' and 'EMA(C,20)' will be assigned to 'a' and 'b', which are further used in the Cross function. Parser will automatically use these values while parsing.

NOTE: There should not be any other variable in BUY variable except the SubQuery Variables.

SubQuery Variables

SubQuery variables are the variables which explicitly represent a Subquery which has the boolean data type.

The variable nomenclature used in IQL is divided into four parts

- The variables which begin with IQL keyword. These cannot be seen or edited in IEL window.
- The variables which begin with IND, PA, ASR, AT or CND (as shown in table below).
- Final Variables: "BUY" or "SELL" variables. The last variable of the code which indicates a Bullish or Bearish custom scan query.
- Temporary Local variables: Other variables which may have any name, return type. (eg, a, b myVar, temp, etc)

Name	Category	Description
IQL	Investar Query Language	For Custom strategies. Details cannot be seen or edited in IEL window
IND	Indicator	Indicators which can be viewed in IEL window
CND	Candlestick	Candlesticks types which can be viewed in IEL window
ASR	Auto- SR	Auto-SR which can be viewed in IEL window
AT	Auto-Trendline	Auto-Trendline which can be viewed in IEL window
PA	Price Action	Price Actions which can be viewed in IEL window

Inbuilt Functions

The inbuilt functions can be directly called from the code to leverage the complex indicators logic. Functions mainly use some arguments which can be numbers or identifiers.

The details of all the functions used in IQL language are given later in this document with examples.

Example 1: This example demonstrates the use of function Cross and EMA indicator. The below example is for the EMA(c,5) is crossing above EMA(c,20);

```
IQL1 = Cross(EMA(Close,5),EMA(Close,20));
BUY = IQL1;
```

There are different restrictions for arguments which vary from indicator to indicator. For instance, MACD is not allowed as the argument for EMA.

```
IQL1 = EMA(MACD(12,24,9) , 20) < 30;            // INCORRECT;
```

Binary Operations

The binary operations are the operations which use two operands and gives a single result.

<ANS> = <Operand1> <BinaryOperator> <Operand2>

Following binary operations are allowed in IQL:

- Addition (+)
- Multiplication(*)
- Division(/)
- Less than (<)
- Greater than (>)
- Less than equal to (<=)
- Greater than equals to (>=)
- Equals to (==)

Disable SubQuery

Inherently, the disable feature is for IEL only, but users can also leverage its advantages in IQL. Using the multiline comment syntax users can temporarily disable the existing query. This disabling sub query is used in the last variable (buy or sell).

Example: This example demonstrates how to disable a subquery correctly.

```
IQL1 = close > l;

IQL2 = EMA(Close,5) < EMA(Close,20);

BUY = IQL1 /* OR IQL2*/;
```

Notice the last buy variable, where we have commented out the IQL2 variable. If we switch back to IEL (only in IQL and IEL mode), the IQL2 subquery will be grey out (disabled). Also, note that the subquery which is being disabled should still have its assignment operation.

Example: Error will occur in case if we comment out the actual assignment statement.

```
IQL1 = close > l;

// IQL2 = EMA(Close,5) < EMA(Close,20);     // Incorrect

BUY = IQL1 /* OR IQL2*/;
```

In the above case, notice that the actual assignment statement is also commented which will give the error

Multi Time Frames

IQL allows users to select their desired time frame for each separate Sub Query. Multi time frames can be set to each separate sub query (SubQuery is the expression separated by the "AND" or "OR") operator.

To use the time frame you need to write the value in the square bracket preceding the subquery

expression. The expression to which we are applying the time frame should be inside a single parenthesis.

Example: In this example, we will use the 5- min time frame.

```
IQL1 = [5]( EMA(Close, 6) < Close );

Buy = IQL1;
```

Example: Incorrect way of using the multi-time frame.

```
IQL1 = [5]EMA(Close,6) < Close;     // Incorrect
```

Buy = IQL1;

Syntactically, the above example depicts that we are applying the 5 minute time frame to only the EMA function and it's not applied to Close, hence it is incorrect.

Each variable must have at the most only one-time frame.

```
IQL1 = [5] (EMA(Close,5) < Open)  OR [2](Cross(Open,Close));      // Incorrect because
IQL1 variable cannot have two different time frames.
```

The above code can be correctly written as:

```
IQL1 = [5] (EMA(Close, 5) < Open);                        //Correct

IQL2 = [2] (Cross(Open, Close));                          //Correct

IQL3 = IQL1 OR IQL2;
```

IQL File Import

User can import the IQL files from going to File->Import-> Import IQL. It will open the explorer window and allow to select the files with .iql extension. On successful import the scans will be displayed in the scan list.

Also, during the shutdown of software, IQL files will be created of all the custom scans so that the user can refer to the details of the scan when the software is off. The IQL files will be created in Documents-> Investar->IQLFiles.

Inbuilt Identifiers for IQL

Category	Identifier Name	Data Type	Description
Indicators	Close / C	Array	The closing value of the current bar
	High / H	Array	The highest value of the current bar
	Open / O	Array	Opening value of the current bar
	Low / L	Array	The lowest value of the current bar
	Volume / V	Array	Value of the volume of stock on the current bar
	DelVol / DV	Array	Delivery Volume
	OpenInt/OI	Array	Open Interest
	Change	Array	Change (difference bettween current and previous closing value)
	ChangePercent	Array	Percentage of Close
	EWO	Array	Elliotwave Oscillator
	WilliamsAD	Array	Williams AD indicator
	TypicalPrice	Array	Typical Price
Price Action	DayOpen	Array	Day Open
	DayClose	Array	Day Close
	DayLow	Array	Day Low
	DayHigh	Array	Day High
	DayOI	Array	Day Open Interest
	BullReversal	bool	Bullish Reversal Bar
	BearReversal	bool	Bearish Reversal Bar
	BullKeyReversal	bool	Bullish Key Reversal Bar
	BearKeyReversal	bool	Bearish Key Reversal Bar
	BullExhausting	bool	Bullish Exhausting Bar
	BullPin	bool	Bullish Pin Bar
	BearPin	bool	Bearish Pin Bar
	BearTwoBarReversal	bool	Bearish Two Bar Reversal
	BullThreeBarReversal	bool	Bullish Three Bar Reversal
	BearThreeBarReversal	bool	Bearish Three Bar Reversal

	BearThreeBarPull	bool	Bearish Three Bar Pullback
	BullThreeBarPull	bool	Bullish Three Bar Pullback
	InsideBar	bool	Inside Bar
	OutsideBar	bool	Outside Bar
	ThreeBarInsider	bool	Three Bar Insider
	NR4	bool	Narrow Range 4
	NR7	bool	Narrow Range 7
	R1	Integer	Resistance 1:- Used with pivot points function
	R2	Integer	Resistance 2:- Used with pivot points function
	R3	Integer	Resistance 3:- Used with pivot points function
	PP	Integer	Pivot Point:- Used with pivot points function
	S1	Integer	Support 1:- Used with pivot points function
	S2	Integer	Support 2:- Used with pivot points function
	S3	Integer	Support 3:- Used with pivot points function
Candlestick Patterns	BullishEngulfing	Bool	Bullish Engulfing
	PiercingLine	Bool	Piercing Line
	BullishHarami	Bool	Bullish Harami
	BullishHaramiCross	Bool	Bullish Harami Cross
	DragonflyDoji	Bool	Dragonfly Doji
	BearishEngulfing	Bool	Bearish Engulfing
	BearishHaramiCross	Bool	Bearish Harami Cross
	GravestoneDoji	Bool	Gravestone Doji

	Hammer	Bool	Hammer
	HangingMan	Bool	Hanging Man
	BullishKicker	Bool	Bullish Kicker
	BearishKicker	Bool	Bearish Kicker
	InvertedHammer	Bool	Inverted Hammer
	ShootingStar	Bool	Shooting Star
	BullishMarubozu	Bool	Bullish Marubozu
	BearishMarubozu	Bool	Bearish Marubozu
	Doji	Bool	Doji
Auto SR	VSR1	Number	Very Strong Resistance 1 (default: dark blue line)
	VSS1	Number	Very Strong Support 1 (default: light blue line)
	SS1	Number	Strong Support 1(default: light blue line)
	SR1	Number	Strong Resistance 1 (default: dark blue line)
Auto-Trendline	Uptrend	Bool	Uptrend
	Downtrend	Bool	Downtrend

Note: There are many other inbuilt Identifiers which will be discussed along with specific indicator and price action.

Inbuilt Functions Support

NOTE: The examples given section does not represent a valid trading strategy. They just represent the valid usage of syntax.

ADX: Average Directional Index

Syntax: ADX(number)

Return: Array

Category: Indicator

```
Example 1: ADX with number
IQL1 = ADX(14) < 20;
BUY = IQL1;
```

Only number is allowed in ADX argument. Number passed through any variable is not allowed.

Example 2: Incorrect argument

IQL1 = ADX(Close[0]); // Incorrect

DIM(Di-) and DIP(Di+)

DIM and DIP are sub indicators of ADX indicator.

Syntax: DIM(Number) / DIP(Number)

Returns: Array

Category: Indicator

```
Example:
IQL1 = DIM(20) > DIP(10);
```

Conditions:

- In same operation, the operands must be different in GUI variables
  IND1 = DIM(20) < DIM(10); // not allowed, because both are DIM.

ATR

Syntax: ATR(Number)

Returns: Array

Category: Indicator

CCI

Syntax: CCI(number)

Returns: Array

Category: Indicator

Ichimoku

Syntax: Ichimoku(subindicator, number,number,number,number,number)

Subindicators : Kijun, Tenkan, SenkouSpanA, SenkouSpanB

Returns: Array

Category: Indicator

MACD

Syntax: MACD(number,number,number)

Returns: Array

Category: Indicator

Signal

Syntax: Signal(number,number,number)

Returns: Array

Category: Indicator

Used as sub indicator of MACD

Histogram

Syntax: Histogram(number,number,number)

Return: Array

Category: Indicator

Trix

Syntax: Trix(number,number)

Return: Array

Category: Indicator

TrixSignal

Syntax: TrixSignal(number,number)

Return: Array

Category: Indicator

SMA: Simple Moving Average

Syntax: SMA(subindicator, number)

Subindicator:Close,Open,Low,High,Volume,DelVol

Category: Indicator

Returns: array

EMA: Exponential Moving Average

Syntax: EMA(subindicator, number)

*Subindicator:Close,Open,Low,High,Volume, DelVol, RSI***

Category: Indicator

Returns: array

Special Case: EMA supports the RSI indicator as its argument.

DEMA: Double Exponential Moving Average

Syntax: DEMA(subindicator, number)

Subindicator:Close,Open,Low,High,Volume,DelVol

Category: Indicator

Returns: array

TEMA: Triple Exponential Moving Average

Syntax: TEMA(subindicator, number)

Subindicator:Close,Open,Low,High,Volume,DelVol

Category: Indicator

Returns: array

WMA: Weighted Moving Average

Syntax: WMA(subindicator, number)

Subindicator:Close,Open,Low,High,Volume,DelVol

Category: Indicator

Returns: array

RSI:

Syntax: RSI(number)

Category: Indicator

Returns: array

StochD:

Syntax: StochD (number, number, number)

Category: Indicator

Returns: array

StochK:

Syntax: StochK (number, number, number)

Category: Indicator

Returns: array

BBandLower:

Syntax: BBandLower(number, number)

Category: Indicator

Returns: array

BBandUpper:

Syntax: BBandUpper(number, number)

Category: Indicator

Returns: array

BBandWidth:

Syntax: BBandWidth(number, number)

Category: Indicator

Returns: array

BBandPercent:

Syntax: BBandPercent(number, number)

Category: Indicator

Returns: array

Supertrend:

Syntax: Supertrend(number, number)

Category: Indicator

Returns: array

Max: Finding maximum value

This function finds the maximum value of given indicator for given number of bars.

Syntax: MAX(indicator, number)

indicator: Open, High, Low, Close, Volume, DelVol

Category: Indicator

Returns: Number

```
Example: Finds the max value of close for last 5 bars
a = Max(Close,5);
```

Min: Finding minimum value

Syntax: Min(indicator, number)

indicator: Open, High, Low, Close, Volume, DelVol

Category: Indicator

Returns: Number

BreakOut

Syntax: BreakOut(argument1,argument2)

Description: argument1 is breaking out above argument2

Returns: Boolean

BreakDown

Syntax: BreakDown(argument1,argument2)

Description: argument1 is breaking down below argument2

Returns: Boolean

BreakOut() and BreakDown() functions are used in Auto SR and AutoTrendlines.

IQL Examples

Example 1: EMA 5 is Crossing above EMA 20 on closing

```
IQL1 = Cross(EMA(Close,5), EMA(Close,20));
BUY = IQL1;
```

Example 2: RSI Followed by EMA

```
IQL0 = Cross(EMA(Close, 5) , EMA(Close, 20));
IQL1 = WithinBars(Cross(RSI(7),30), 3);
BUY = IQL0 AND IQL1;
```

Example 3: Boring Candle

```
IQL_BoringCndl = (abs(Open-Close) / abs(High-Low)) > 0.5;
BUY = IQL_BoringCndl;
```

Example 4: RSI followed by Supertrend (Sell)

```
IND0 = Cross(Supertrend(5,1),Close);
IND1 = WithinBars(Cross(70 , RSI(7)), 3);
SELL = IND0 AND IND1;
```

Example 5: Chandelier Sell or RSI followed by Supertrend

```
IND0 = Cross(ChandelierExit(Long,22,3),Close);
IND1 = WithinBars(Cross(70 , RSI(7)), 3);
IND2 = Cross(Supertrend(5,1.5),Close);
SELL = IND0 AND (IND1 AND IND2);
```

Example 6: RSI followed by EMA(5,20) crossover - Buy

```
IND0 = Cross(EMA(Close, 5) , EMA(Close, 20));
IND1 = WithinBars(Cross(RSI(7),30), 3);
BUY = IND0 AND IND1;
```