mq4build.com

# mq4build documentation

An Illustrated Guide for Fast Start and Informed Usage of Interactive Time Series Analysis and Source Code Generation Utility

# Preface

## Structure

This documentation is structured in two chapters. Chapter One "Use cases" will give practical introduction to mq4build by elaborating on examples of usage. Chapter Two "Reference" will give systematic and more complete description of the system functionality.

## Style

Due the highly interactive nature of the documented system, wherever possible, the textual descriptions are back plated with illustrations – most often, screen clippings from the software interface.

The author is not native English and is not much involved with the art of writing. We ask the reader for indulgence for the poor style and promise to put an effort for further improving it.

## Before start

mq4build *is an* analysis utility. The utility allows defining conditions over ready defined technical indicators and combining them for modelling of specific time series behavior - patterns.

mq4build, even being very flexible, *is not a* universal programming language. It is not programming language at all. There are no classes, objects, loops, variables, functions or complex expressions to deal with. Thus, the expressive power of the engine is constrained. Not everything the researcher could imagine, can be implemented with present engine. The tradeoff is that engine is optimized for the wide class of designs, related to history patterns modelling. Users, that do not have software programming experience will find most added value here – they are enabled to perform analysis, design and implement patterns on their own. Same time, experienced programmers could find out mq4build as useful productivity tool.

mq4build, even being fun, *is not a* game. The interactivity elements are provided to enable non programmer users to perform efficient exploration and analysis of existing data. However, the overall user experience heavily depends on performance of the underlying indicators. The internal evaluation in mq4build is organized in efficient way but when using many indicators or slow ones, the system will need time to perform evaluation and to visualize the results. So, do not expect 60 FPS interaction!

mq4build, even distributed as indicator, *is not a* technical indicator. It is not designed for performing analysis of real time events. For this purpose, the embedded source code generator will create a quality production code implementing evaluation logic, equivalent with the logic that is defined in mq4build.

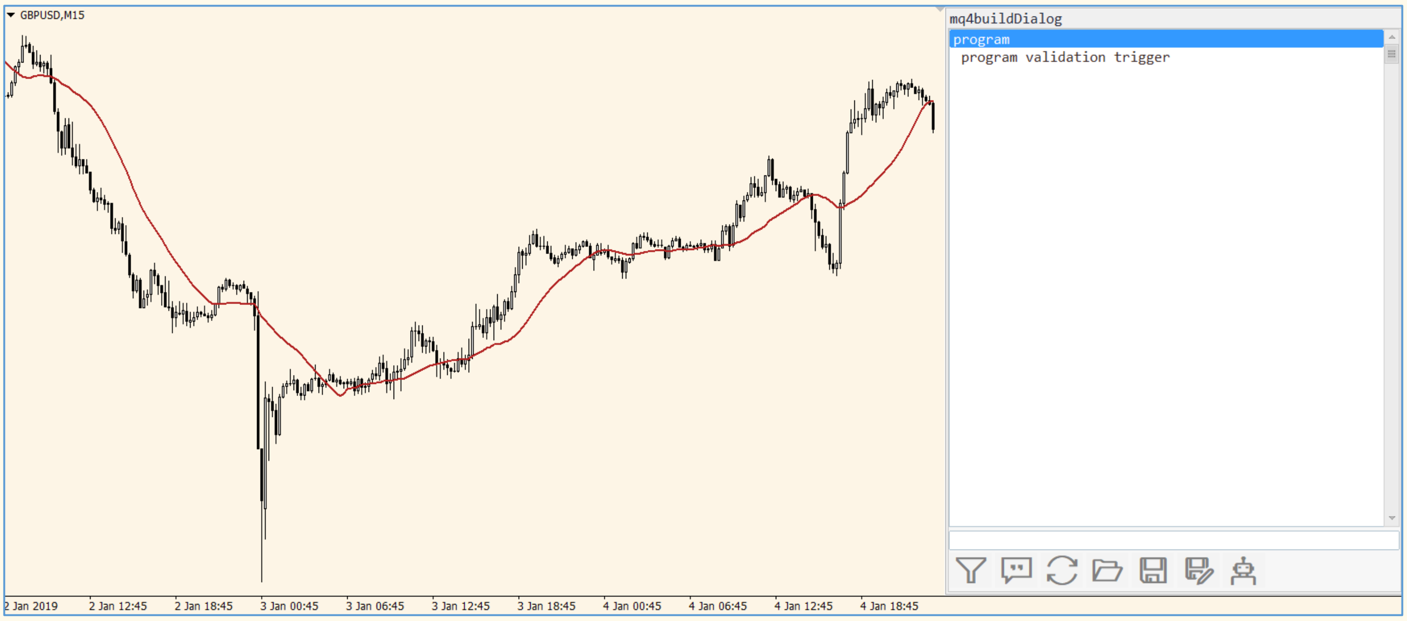
mq4build, even working with most custom indicators, *is not* capable to work with every custom technical indicator for Metatrader 4. Indicators, that does not create buffers could not be used with mq4build. Indicators that re-evaluate their buffers (repainting) will not work properly or will not work at all. Further, we could consider the wide existing variety of indicators implementation: there are indicators that call themselves; indicators with non-standard interface etc. most probably, these exotic implementations will have exotic problems to integrate.

# Chapter One “Use cases”

In this chapter we provide some basic examples for usage of the mq4build utility.

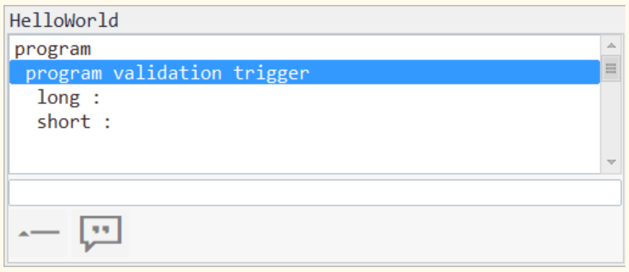
## Hello World

Let us follow the old tradition and start with very simple construction.

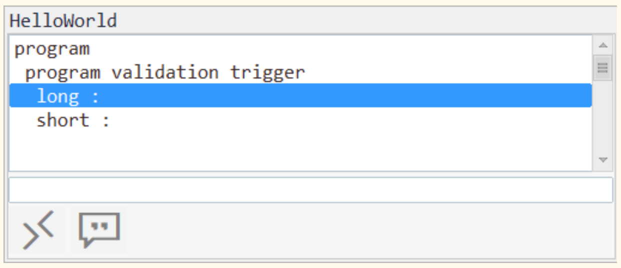


```
mq4buildDialog
program
 program validation trigger
```

1. Open an empty chart and add a Moving Average indicator and the mq4build indicator.
2. In the code window of mq4build select the line, named **`program`**.
3. Select button “Update” to capture the indicator buffer. The system will identify used indicator and will enable mq4build to work with the corresponding buffer. This operation takes half to one second typically. Nothing will change visually!
4. Select the line **`program validation trigger`**.
5. Select button “Expand” . The system will open two lines below the selected line: **`long`** and **`short`**, as below left.

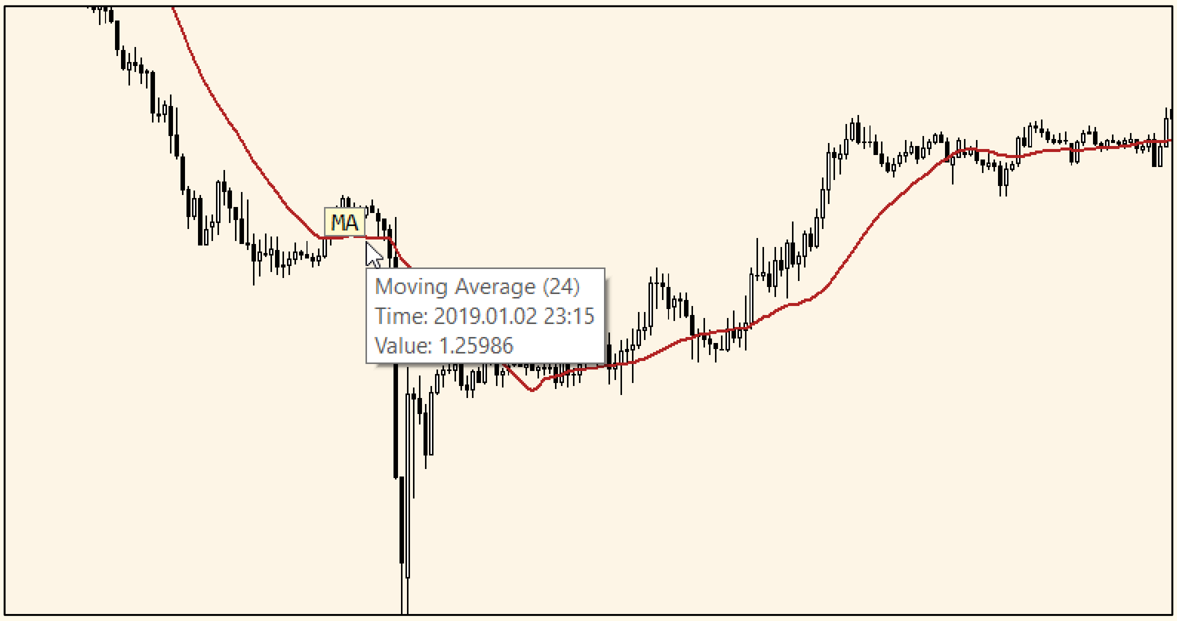


```
HelloWorld
program
 program validation trigger
   long :
   short :
```



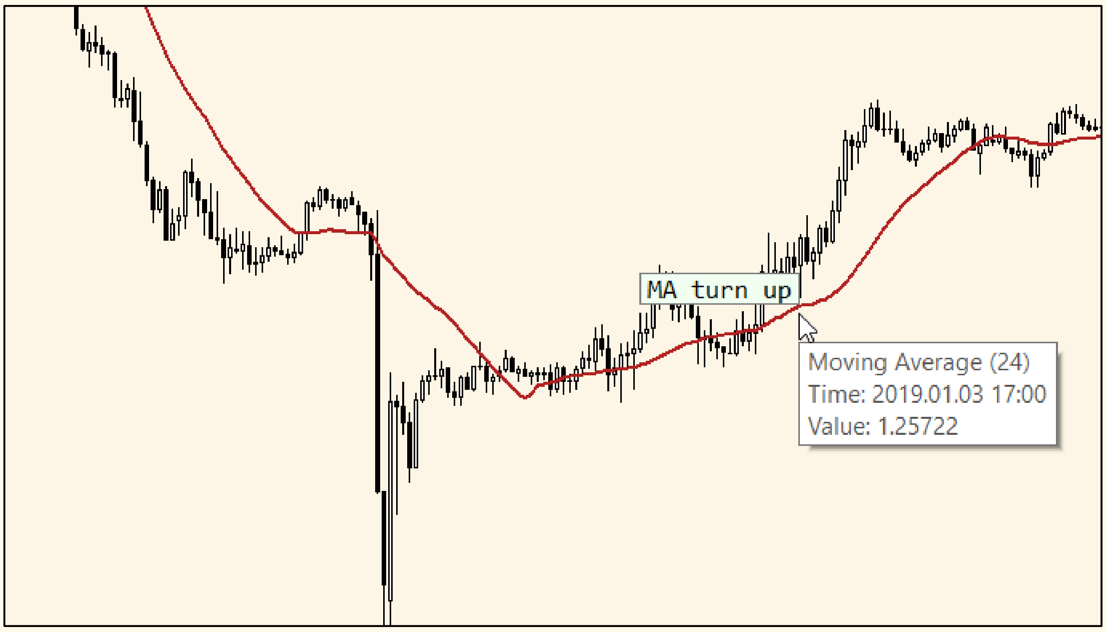
```
HelloWorld
program
 program validation trigger
   long :
   short :
```

6. Select line **`long`** to define condition for the long direction as above right.
7. Select button “Condition” to define the condition for the selected direction. This puts the system in buffer capturing mode.



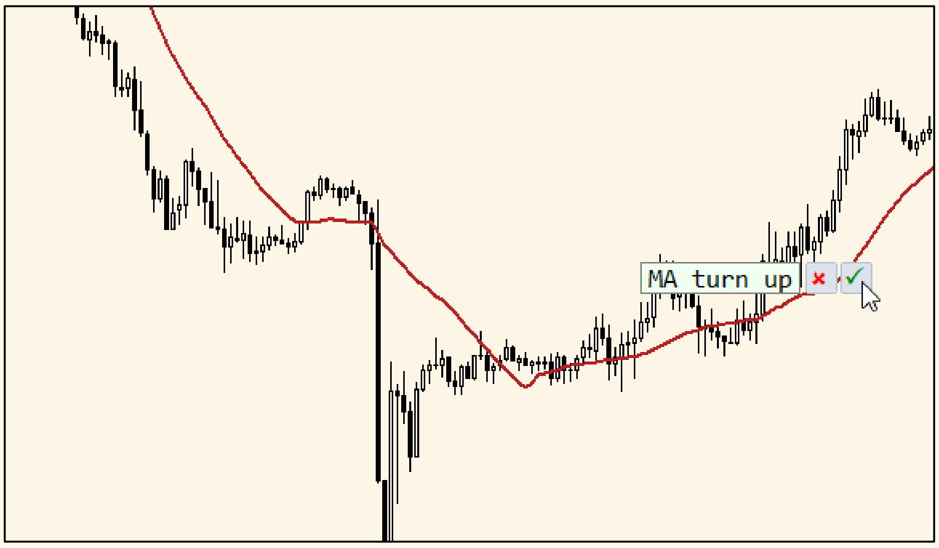
Moving Average (24)
Time: 2019.01.02 23:15
Value: 1.25986

8. Navigate mouse cursor to a chart area, where the moving average is decreasing. While in buffer selection mode, the system will identify the nearest to mouse cursor available buffer and will show a label, touching the buffer line, as in screenshot above.
9. Click mouse at this position. The system will remember this as first reference point.

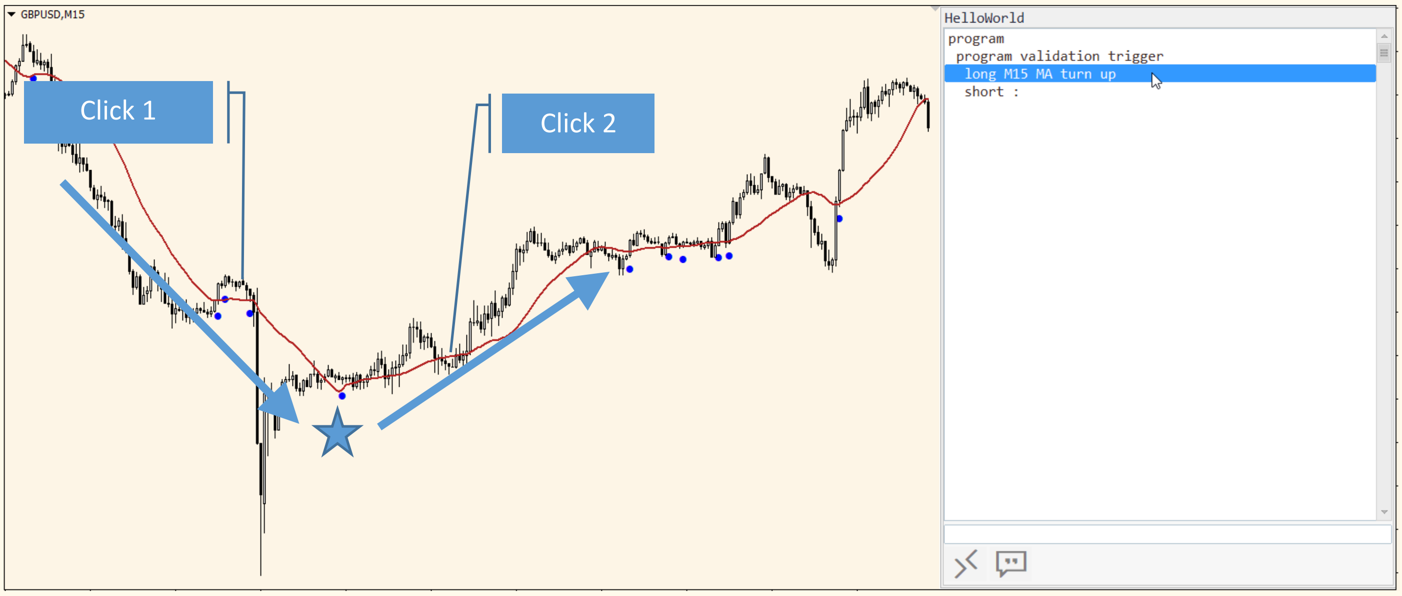


Moving Average (24)
Time: 2019.01.03 17:00
Value: 1.25722

10. Navigate mouse cursor to near area, where the moving average is increasing. The system will identify that we have changed buffer direction in the sketched area and will propose the corresponding definition in the label. In this case it is `MA turn up.`
11. Click mouse at this position. The system will lock the proposed condition and will ask for final acceptance, showing green check for accept and red cross for cancel buttons around the condition label.



12. Click the green checkbox to accept the proposed condition. The system will accept condition and will show it in the code line **`long M15 MA turn up`**. Further, system will evaluate the defined condition and will display the evaluation results on chart as dots.



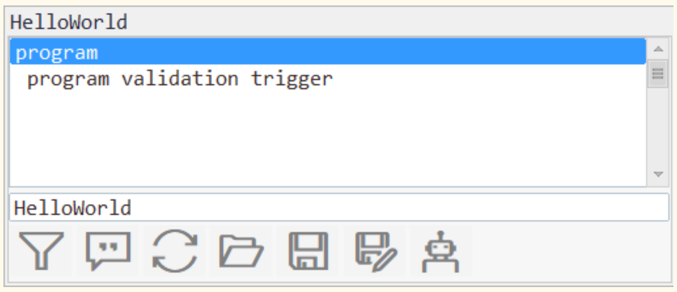
```
program
 program validation trigger
  long M15 MA turn up
  short :
```

Notice, how we defined the turn up condition with just two clicks. We actually sketched the condition specification by providing the first reference point in an area where the indicator is falling and the second reference point in an area where the indicator is raising, both reference points were around a pivot point which was not explicitly specified by user but the system identified it and take it into account to guess the desired condition.

13. Use the same process as above to define the **`short`** condition. To do this, select the **`short`** line, enter selection mode with button “Condition” and select two reference points around a peak of the indicator. Confirm the proposed condition.

14. To save the created code, click on **`program`** line.
15. Click the function button “Save” 🖫. The edit line above will become yellow. Click the edit line, enter text “HelloWorld” and press Enter on the keyboard.
16. You will see the mq4build panel with changed caption “Hello world” and the edit line with a white background again. This means that program is saved successfully.
17.

---------------------To be continued –



```
HelloWorld
program
  program validation trigger
HelloWorld
```

# Chapter Two “Reference”

This chapter contain introduction material about mq4build basic concepts and functionality. The topics covered are: interface of the software, evaluation engine concepts, elements for building visual research. At the end of chapter, description of buffer conditions will unveil the base for all evaluation process.

The concepts are explained and complemented with screenshots for easy transition to the live software. Same time, attempt was, text to be done as compact as possible.

# documentation

An Illustrated Guide for Fast Start and Informed Usage of mq4build, interactive time series analysis and source code generation utility

## Introduction

This documentation may appear big, because I do not have the talent to make it small. The intention is to keep text compact, with visual glues on the left side of the page.

## Vision

Mq4build is a tool for visual analysis of any time series.

Currently mq4build is implemented as a custom indicator for the popular platform Metatrader4. It appears as a panel in the right side of the main chart window.

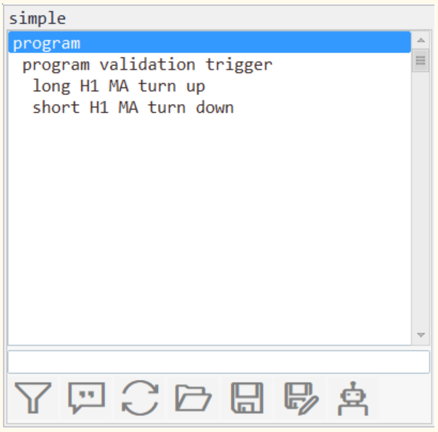
General view of technical analysis chart with attached mq4build panel

Analysis is done by adding technical indicators on the chart and further defining a structured set of conditions and relationships, called elements. The ability of mq4build to do this with simple mouse manipulations and to give immediate feedback on defined elements, makes this utility productive. Facilitation of a loopback between definitions and results increases quality and speeds up the process of system design.

In addition, mq4build is capable to generate production quality mql4 source code for evaluation of defined analysis. The generated code is independent of mq4build environment.

## Interface

Mq4build interface is implemented with a panel and through interaction functionality with indicator buffers.


```
simple
program
  program validation trigger
    long H1 MA turn up
    short H1 MA turn down
```
mq4build Panel with a simple program

### mq4build Panel

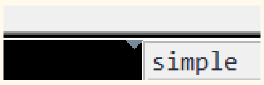
The panel consists of:   a heading, a code window, a comment line and a toolbar which contains context sensitive commands.

The heading contains the name of the current program.

The code window contains the elements of the program.

The comment line is used for editing comments of elements and also for input of the program name.

The toolbar contains command-buttons, applicable to the currently selected element.



Panel width can be modified by dragging the grey triangle in the middle of the image.

The panel height always fits the height of the main chart window.

The panel width fits the space between the chart shift and the right border of the chart. This allows panel width to be customized by  dragging the chart shift mark.

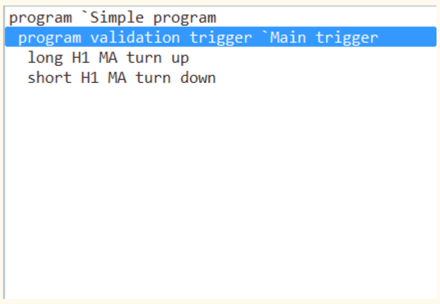
## Code Window

The code window contains brief description of all currently defined elements. Each element is showed on a separate line with indentation.

When an element is selected, the system performs the following actions:

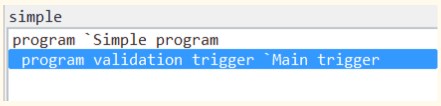
- the selected element is highlighted;
- the element comment is displayed in the comment line;
- the set of available user interface commands for the element is evaluated and synchronized with the toolbar;
- the selected element (with its sub elements) is evaluated;
- the results of evaluation are synchronized and displayed on chart;

The elements, that contain one or more other elements can be expanded or folded to show / hide the underlying content. The visualization state of the element – expanded or folded does not affect the evaluation process of the element and the contained elements. Also, the visualization state of the element does not affect the evaluation of the element when it is contained within some higher order element.



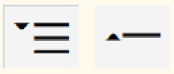
```
program `Simple program
 program validation trigger `Main trigger
  long H1 MA turn up
  short H1 MA turn down
```

Code window with selected program validation trigger



```
simple
program `Simple program
 program validation trigger `Main trigger
```

Fragment of code window with program validation trigger details hidden



Buttons Expand and Fold

## Comment line

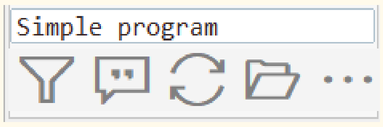
The comment line is located below the code window. The comment line is used for entering comments on elements and displaying them.

Also, the comment line is used to enter file names for saving or loading the current analysis.

## Toolbar

The toolbar holds buttons for the commands, available for currently selected element in the code window.

When the width of the panel does not allow all needed buttons to be displayed, the toolbar automatically the program displays the More button which allows the shifting of visualized buttons, in a similar manner to that of mobile devices.



Simple program

Comment line and the toolbar with the More button

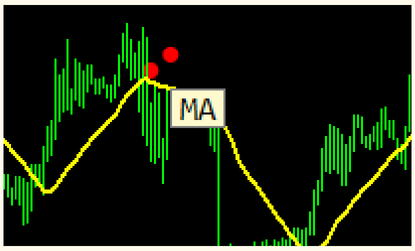
## Buffers interaction

Mq4build is able to capture technical indicators, applied on chart and to use their buffers for the analysis and source code generation purposes.

To capture indicators: The function "Update" **must** be used after adding or removing any indicators on the chart or when modifying any values of the input parameters or when defining some indicator levels.

Button **Update** for capturing indicators, available when **Program** element is selected.

Capture of the indicators is an expensive operation that cannot be done seamless in the Metatrader 4 environment. For synchronization, the function Update **must** be used manually every time the chart setup is changed.



Selecting an available buffer. A label with the name of the selected buffer is visible.

When the user is selecting any buffers, the system will choose the indicator buffer which is nearest to mouse pointer and will display a label with a selected buffer name. The lower right corner of the label will exactly touch the selected buffer. The system will not select any buffers if visualization for their indicator is disabled for the current chart time frame (using the tab Visualization and the time frame checkboxes from Indicator Setup dialog for the particular underlying indicator).

## Evaluation engine

## Elements

Mq4build features five elements to compose evaluation functionality.

These elements are: program, block, condition, trigger and direction.

All the elements except 'direction' define evaluation of two Boolean values. One value is for long direction and the other value is for short direction. Elements 'direction' are either long or short and evaluate a single Boolean value.

Any evaluation is defined by the rules of its elements. The Rules for every element are described in this section.

The system performs evaluation of defined elements and display the results for the currently selected element.

### Program

The results of any program evaluation consist of signals for long and short direction which are evaluated independently. Following the underlying logic, any combination of signals can appear. After a signal is evaluated, on next bar, the signal is reset and the evaluation process is started from initial state.

#### Structure

Any program consists of a sequence of blocks and a validation trigger. The block sequence can be empty.

#### Evaluation rules

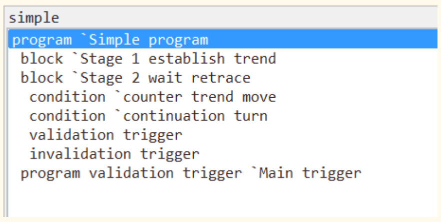
If there are any defined blocks, they are evaluated in a sequence which follows these rules:

1. Initialization: For both directions, long and short there is exactly one active block. Initially, the active blocks for both directions are the first ones;
2. Advance: If the active block is evaluated as 'true', then the next block in the sequence becomes active – (advancing takes place);
3. Reset: If the block before active block is evaluated as 'false', then the first block in the sequence becomes active – (reset takes place);
4. Signal: If the last block in the sequence becomes active and the validation trigger becomes active – the system displays a signal in the corresponding direction and activates the first block;

This sequence construction allows describing complex history patterns. The informal interpretation will sound like so:

To have a signal: I want to happen block 1 conditions and after that, while block 1 is not invalidated, block 2 must happen and having it valid it is needed block 3 etc. and finally, I want the validation trigger.

| | |
|---|---|
| In program: | Blocks are evaluated according to the ordered sequence. |
| In block: | Conditions are evaluated as an unordered set. |



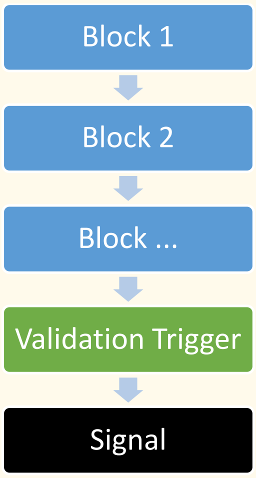
```
simple
program `Simple program
 block `Stage 1 establish trend
 block `Stage 2 wait retrace
  condition `counter trend move
  condition `continuation turn
  validation trigger
  invalidation trigger
 program validation trigger `Main trigger
```

Code fragment with different nested elements:

The program, containing two blocks and a validation trigger.

The expanded block 2, containing two conditions, validation trigger and invalidation trigger.

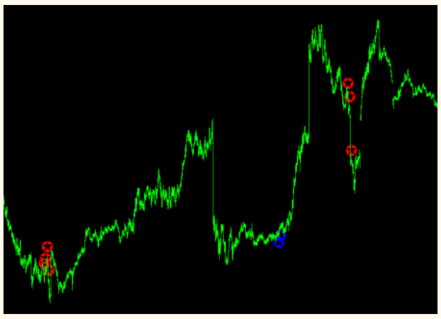
Element comments follow the symbol ` .



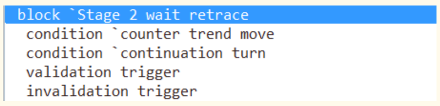
Schematic representation of a program evaluation sequence.

Failure on the evaluation of any stage before reaching next stage resets the process to Block 1.

This process is evaluated independently for long and short directions.

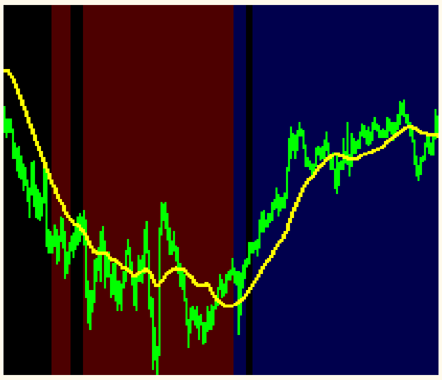


Visualization of signals, generated by a sample program.



```
block `Stage 2 wait retrace
  condition `counter trend move
  condition `continuation turn
  validation trigger
  invalidation trigger
```

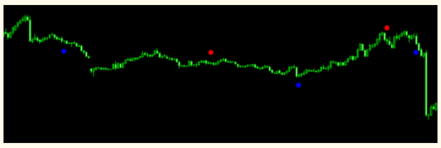
Code fragment with definition of a block, defined by two conditions, validation trigger and invalidation trigger.



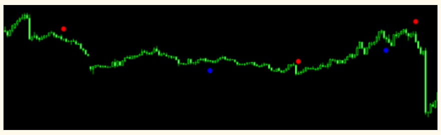
Visualization of block evaluation results.

Periods when block is evaluated for short direction are backgrounded red. Periods when block evaluated  long direction have blue background.

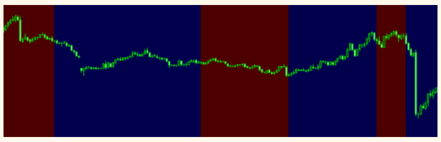
Visualization appear after selecting the block in the code window.



Visualization of validation trigger



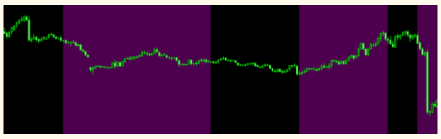
Visualization of invalidation trigger



Visualization of composed condition

Here the validation and invalidation triggers for long and short directions are mutually exclusive. This result is having always defined long or short direction in non-overlapping time periods.

Such a construction is by far not required.  For example, conditions that measure the time series volatility will have positive evaluation results for long and short directions aligned – ‘true’ or ‘false’ for both directions at same time periods, as in the visualization below:



## Block

Block is a structural element of the program. A block remembers the result of its previous evaluation as state.

### Structure

A Block is defined by a set of conditions, a validation trigger and an invalidation trigger. The set of conditions can be empty.

### Evaluation rules

Validation: If a block state is considered ‘false’ and all the conditions and the validation trigger are evaluated ‘true’, the block evaluation result is ‘true’.

Invalidation: If the block state is ‘true’ and the invalidation trigger is evaluated ‘true’, the block evaluation result is ‘false’.

Memory: If none of the above evaluation rules apply, the block state is not changed.

The Memory rule enables defining time periods, starting with a set of conditions. This as a structure element facilitates the history research of these time periods. Having the visualization feedback, blocks can be tuned and refined by editing the conditions in block definition. This process leads to more effective and precise definitions.

## Condition

Condition is a structural element of the Block. Same as blocks, the conditions have memory and remember their evaluation state.

### Structure

Condition is defined by a validation trigger and an invalidation trigger.

### Evaluation rules

Validation: When the condition is ‘false’ and validation trigger evaluates ‘true’, the condition evaluation result is ‘true’.

Invalidation: When the condition is ‘true’ and invalidation trigger evaluate ‘true’, the condition evaluation result is ‘false’.

Memory: If none of the above evaluation rules apply, the condition state remains unchanged.

Conditions are very similar to the blocks as a structure element, except the fact that the conditions do not utilize sub elements other than validation and invalidation triggers.

Our concept for evaluation of conditions and blocks, based on validation and invalidation triggers require some justification. We designed these evaluation rules to provide high expressive power on the elements. Validation and invalidation triggers could be arbitrary, based on different indicators, evaluated on a different time frames. This flexibility enables the researcher to specify time periods of interest easily and with a compact notation.

Same times, the power comes for a price! Correct definitions require careful selection of logically compatible triggers. Happily, visual feedback, provided by the environment helps to keep the constructed logic under control. Also, visual manipulation allows easy fixing of problems.

## Trigger

Trigger is structural element of Program, Block and Condition.

The only purpose of a Trigger element is to group two Direction elements in a single construct. Trigger elements do not have a memory, they fire a single event and immediately forget everything about it.

### Structure

A Trigger groups two Direction elements – one for the long and one for the short direction.

### Evaluation rules

Follow: For long direction, trigger evaluation result is the evaluation result of the long direction element.

Follow: For short direction, trigger evaluation result is the evaluation result of the short direction element.

## Direction

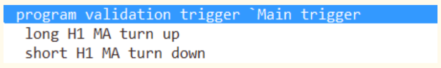
Direction element is structural element of Trigger.

Direction elements are defined by the Buffer condition and its time frame.

It has to be noted, that predefined evaluation direction does not imply constraints on the directional nature of the buffer event, used for element definition. The evaluated trigger could be part of overall design that has logic, indifferent to time series direction of change or even pointing conditions counter to the evaluated direction. For example, such a triggers could be dynamics filters, for which direction interpretation is not relevant. Another example are retracement identifications, looking for a conditions with direction opposite to the main evaluated direction. In the last case we will have direction element for example 'long', which will be defined by a buffer event, having nature to identify changes of time series in short direction.

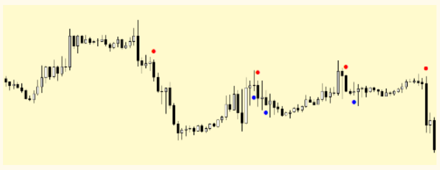
To avoid ambiguities, we strongly recommend commenting the created elements.

Best practice: Use comments to specify the design logic of every element.



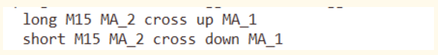
```
program validation trigger `Main trigger
  long H1 MA turn up
  short H1 MA turn down
```

Code fragment, showing a trigger definition with its two direction elements.



Visualization of the results of a trigger evaluation

Red dots represent positive evaluations of the trigger in short direction and blue dots in long direction.

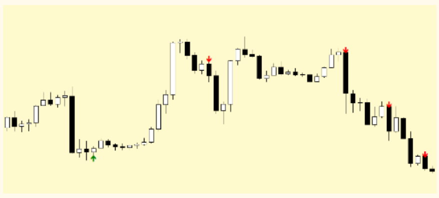


```
  long M15 MA_2 cross up MA_1
  short M15 MA_2 cross down MA_1
```

Code fragment, representing a pair of direction elements. In this example the direction elements are defined on M15 time frame and evaluate the cross over event of the two presented indicator buffers MA_2 and MA_1.

The general format for presenting a pair of direction elements in code is:

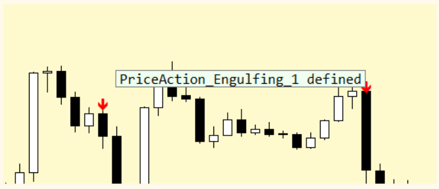
```
long <timeframe> <buffer event> <comment>
short <timeframe> <buffer event> <comment>
```



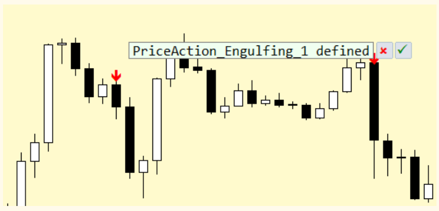
Technical indicator with two arrow buffers – green and red arrows.

Button **Condition** for defining conditions over indicator buffers.

The command is available when a `direction` element is selected in the code window. Using command Condition will put the system in condition definition mode.



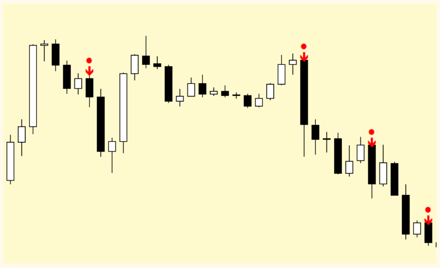
Text label, proposing condition definition.



After mouse click, the system locks current proposal and gives options to accept or reject the locked condition.

```
  short H1 PriceAction_Engulfing_1 defined
```

Code fragment with defined direction element, generated after confirmation of the proposed condition.



Feedback. The system will evaluate and display the defined direction element. Here the red dots represent the direction element and follow the original buffer arrows as expected.

The defined direction element is available for further processing by the evaluation engine, working together with the other defined elements.

## Buffer conditions

Buffer condition is structural element of Direction element. Buffer conditions are evaluated over available indicator buffers.

The buffer conditions can be:

- appearance of an arrow - keyword **`defined`**;
- change of a buffer value from defined to undefined or vice versa, keywords **`appear / disappear`**;
- change of a buffer direction – keywords **`turn up / down;`**
- crossing over of two buffers – keywords **`cross up / down`**;

The buffer conditions are defined by user visual interaction with the environment. The general sequence is:

1. User request condition definition using condition command ;
2. User points to an example of the event that has to be defined;
3. The system tries to guess the desired buffer event;
4. In case of identification, the system proposes it to user;
5. User either accept or reject the system proposal;

When proposed condition is accepted, the system will perform these actions:

6. Remember the condition;
7. Compose a definition for the current direction element with the defined condition, evaluated on the current chart time frame;
8. Perform evaluation of the newly defined direction element;
9. Visualize the evaluation results on the chart.

### Condition “defined”

This condition is based on an indicator buffer, providing single arrow signals.

How to define a buffer condition that identify an arrow appearance?

1. Select a direction element in the code window.
2. Use Condition command  to put the system in condition definition mode. The system will start scanning the available indicator buffers.
3. Hover the mouse pointer around an indicator arrow of the desired buffer. The system will identify the arrow and will propose the condition definition in the text label, pointing the indicator buffer.
4. Having proposed condition definition, click the left mouse button. The system will lock the currently identified condition and will ask for final acceptance.
5. Accept or reject the proposed definition. If accepted, the system will apply the definition to the Direction element selected in point 1. with timeframe equal to the current chart timeframe.

## Condition “appear / disappear”

This condition is based on indicator buffer, alternating periods with defined and undefined values. Often, indicators which change line color are implemented with two or more buffers of this type, using a separate buffer for each different line color.

### How to define a buffer condition that identify a buffer appearance?

1. Select a direction element in the code window.
2. Use Condition command to put the system in condition definition mode. The system will start scanning the available indicator buffers.
3. Hover the mouse pointer around the desired indicator buffer defined value. The system will identify the buffer and will show the buffer name in the text label.
4. Mouse click to lock the buffer selection.
5. Navigate mouse pointer left from the selected point to an area, where buffer values are not defined. (line disappear). The system will propose definition of buffer condition “appear”.
6. Having proposed condition definition, click the left mouse button. The system will lock the currently identified condition and will ask for final acceptance.
7. Accept or reject the proposed definition. If accepted, the system will apply the definition to the Direction element selected in point 1. with timeframe equal to the current chart timeframe.

### How to define a buffer condition that identify a buffer disappearance?

The process for implementing buffer condition “disappear” is the same, except, in point 5, after locking the buffer, we navigate right direction from the selection point to an area in the future, relative to first reference point, where the buffer line disappears.

The described definition process is a sketch for describing the desired buffer event by visual example on the actual buffer. In essence, we reference an area, where the target condition that we want to define, actually happens.

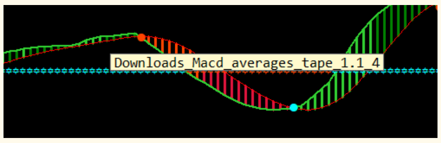
### How to define the other buffer conditions?

The same sketching technique as above is used for definition of the other buffer conditions:

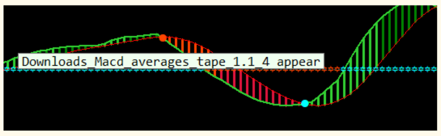
For the buffer condition “turn up” we sketch two points of a single buffer and in the area between them the buffer has local minimum and turns its direction from falling to raising. Symmetric process is used for defining buffer condition “turn down”.

For the buffer condition “cross up” we sketch two points on two buffers and in the area between them the two buffers cross their values. Same for “cross down”.

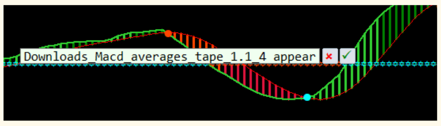
I will not further challenge the reader’s patience with repetitive detailed explanations of these similar techniques, instead detailed examples are given in section “Use cases”. Much more useful will be to work out the corresponding functionality in the actual environment.



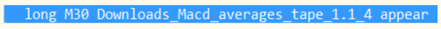
**Click 1**: Point the desired buffer with defined value and click to lock the buffer selection.



**Click 2**: Navigate mouse pointer to an area where buffer values are not defined. The system will change the color of text label and will propose definition of condition “appear”. Mouse click in this state.

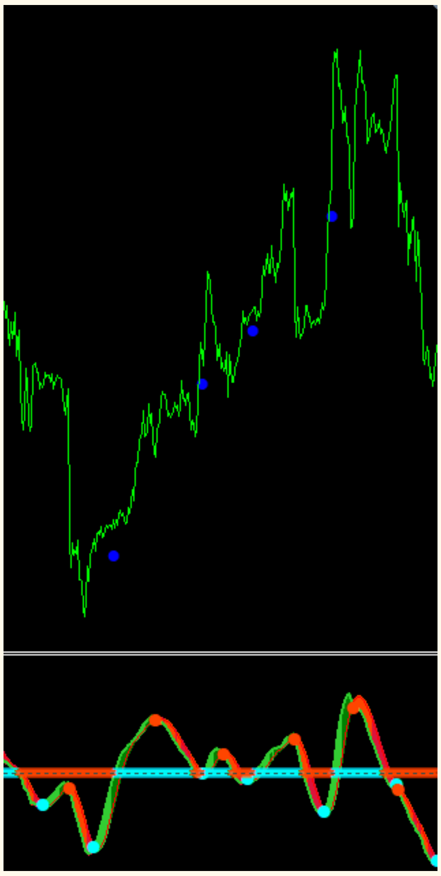


**Click 3**: The system will lock “appear” condition and will ask for confirmation / rejection.



```
long M30 Downloads_Macd_averages_tape_1.1_4 appear
```

Code fragment with defined direction element, evaluating buffer condition “appear” on time frame M30 for the buffer #4 of displayed technical indicator.



Visualization of evaluated element

Condition “cross up / down”

Condition “turn up / down”