mq4build.com

mq4build documentation

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

Stoyan Ivanov

[Date]

Chapter One “Fast start”

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.

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.

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 page.

Vision

General view of technical analysis chart with attached mq4build panel

Mq4build is a tool for visual analysis of 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.

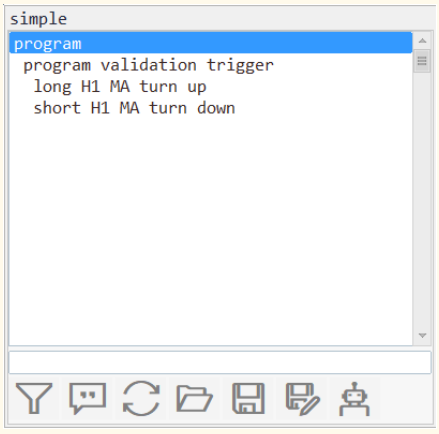
Analysis is done by adding technical indicators on the chart and further defining a structured set of conditions and relationships, called elements. The ability of mq4build to do this with simple mouse manipulations and to give immediate feedback on defined elements makes the utility a powerful analysis solution. 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.

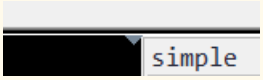
Interface

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

mq4build Panel



mq4build Panel with a simple program



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

The panel have heading, code window, comment line and a toolbar with context sensitive commands.

The heading contains current program name.

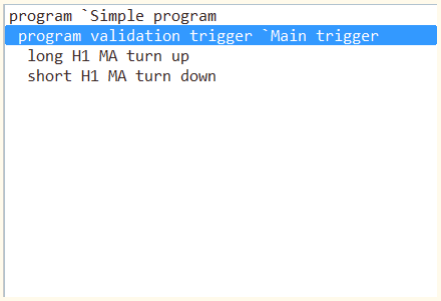
The code window contains program elements.

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

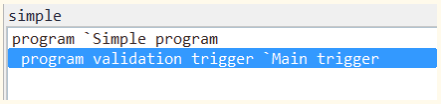
The toolbar contains buttons for the commands, applicable to the currently selected element.

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

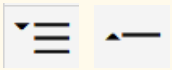
The panel width fits the space between chart shift and right border of the chart. This allows panel width to be customized by dragging the chart shift mark.



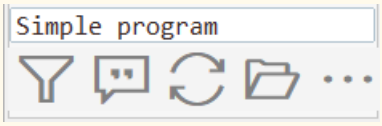
Code window with selected program validation trigger



Fragment of code window with program validation trigger details hidden

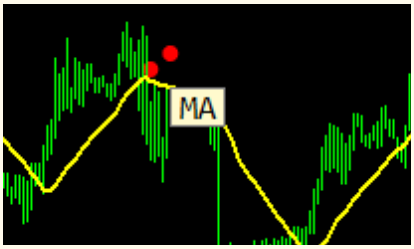


Buttons Expand and Fold



Comment line and the toolbar with button More

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



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

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 by some higher order element.

Comment line

The comment line is located below 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 displays the button More, allowing shifting of visualized buttons, similar as this is done on mobile devices.

Buffers interaction

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

To capture indicators: Function “Update” **must** be used after adding or removing indicators on chart, when modifying values of input parameters and when defining indicator levels.

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

When user is selecting buffers, the system will choose the indicator buffer which is nearest to mouse pointer and will display a label with selected buffer name. The lower right corner of the label will exactly touch the selected buffer.

The system will not select 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).

Evaluation engine

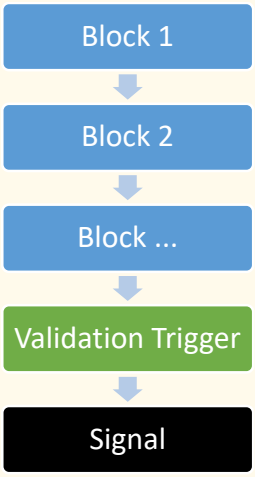
```
simple
program `Simple program
 block `Stage 1 establish trend
 block `Stage 2 wait retrace
  condition `counter trend move
  condition `continuation turn
  validation trigger
  invalidation trigger
 program validation trigger `Main trigger
```

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.

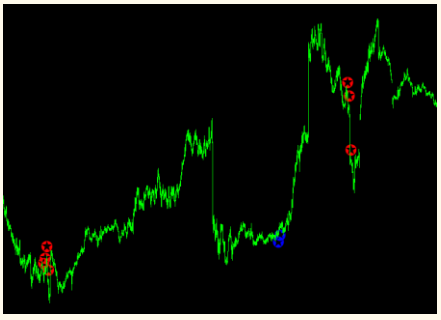
Element comments follow the symbol ` .



Schematic representation of a program evaluation sequence.

Failure on each stage before reaching next stage resets the process to Block 1.

This process is evaluated independent for long and short directions.



Visualization of signals, generated by a sample program.

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 single Boolean value.

Evaluation is defined by element rules. 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 program evaluation are signals for long and short direction, evaluated independent. 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

Program is defined as a sequence of blocks and a validation trigger. The block sequence can be empty.

Evaluation rules

If there are defined blocks, they are evaluated as sequence following 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 evaluates 'true' next block in the sequence becomes active – advance;
3. Reset: If the block before active block evaluates 'false' the first block in the sequence becomes active – reset;
4. Signal: If the last block in the sequence becomes active and the validation trigger becomes active – the system indicate signal in the corresponding direction and make active the first block;

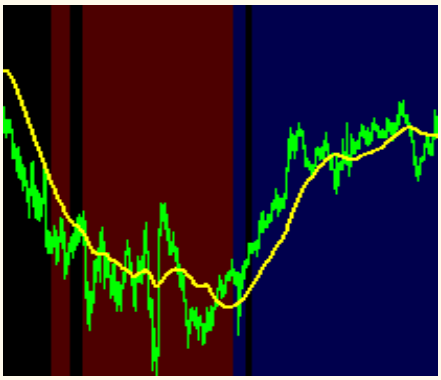
This sequence construction allows describing complex history patterns. The informal interpretation will sound like so:

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 evaluate as ordered sequence.
In block:	Conditions evaluate as unordered set.

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

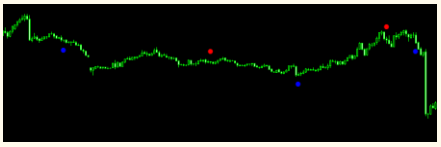
Code fragment with definition of a block, defined by two conditions, validation trigger and invalidation trigger.



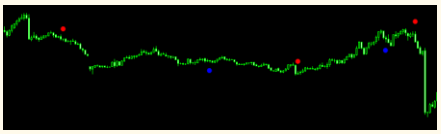
Visualization of block evaluation results.

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

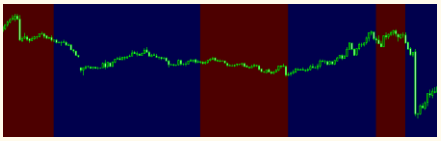
Visualization appear after selecting the block in the code window.



Visualization of validation trigger



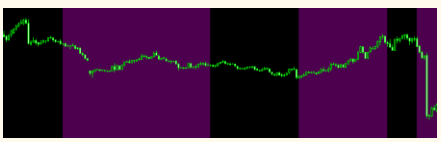
Visualization of invalidation trigger



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

The result of block evaluation for respective direction is 'true' for the period of time, starting with evaluating positive Validation rule and ending with evaluating positive Invalidation rule.

Structure

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

Blocks are part of structure of the program.

Evaluation rules

The block remembers the result of previous evaluation as state.

Validation: If the block state is 'false', if all the conditions and the validation trigger are evaluated 'true', the block evaluation result is 'true'.

Invalidation: If the block state is 'true', if 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

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'.

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

Conditions are very similar to the blocks as a structure element, except, 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 very high expressive power of elements. Validation and invalidation triggers could be arbitrary, based on different indicators, evaluated on a different time frames. This flexibility gives a powerful weapon for the researcher.

Same time, this power comes for a price. Correct definitions will require careful selection of 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.

```
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 a trigger evaluation results

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>
```

trigger

Triggers function 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 about that.

Structure

A trigger is defined by two direction elements – one for long direction 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

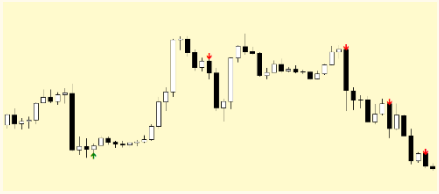
Direction elements are defined by an indicator buffers condition and a time frame.

Direction elements are part of trigger definitions and come always by pair – one element for the long evaluation direction and another one for the short evaluation direction.

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 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.

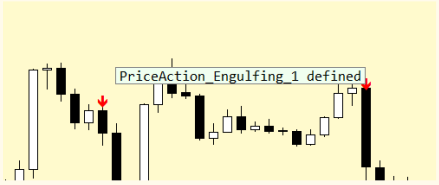
Best practice: Use comments to specify the design logic of every element.



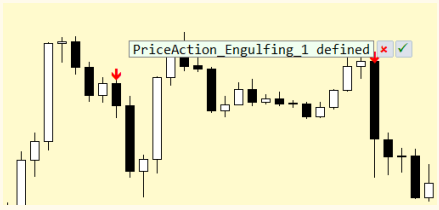
Technical indicator with two arrow buffers – green and red arrows.

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.



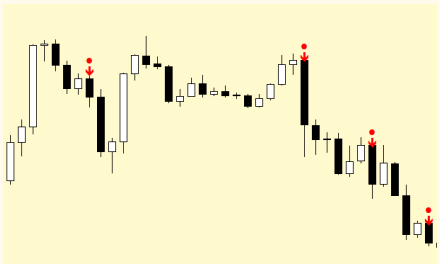
Text label, proposing condition definition.



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

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:

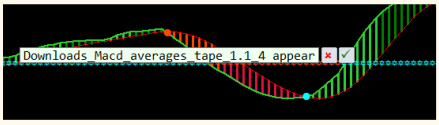
To define the condition, use Condition command ⌧ to put the system in condition definition mode. The system will start scanning the available indicator buffers.

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.

Having proposed condition definition, click the left mouse button. The system will lock the currently identified condition and will ask for final acceptance.

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

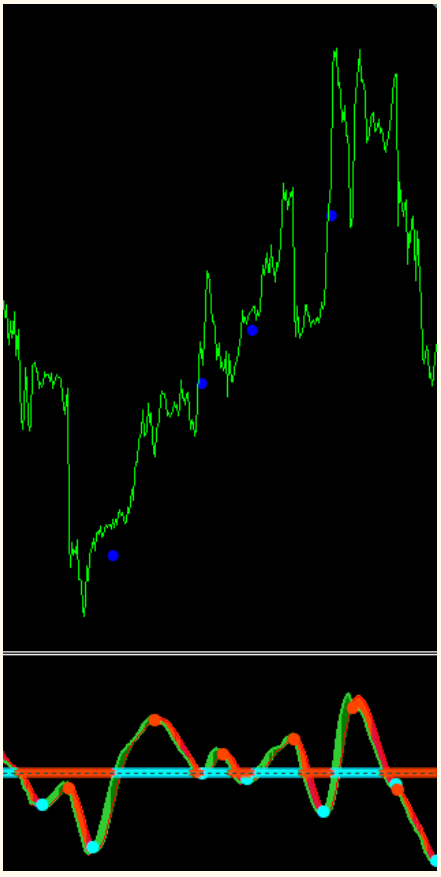
Click 2: Navigate mouse pointer to an area where buffer values are not defined. The system will change the color of text label and will propose definition of condition “appear”. Mouse click in this state.



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 “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:

To implement buffer condition “appear” use Condition command ☒ to put the system in condition definition mode. The system will start scanning the available indicator buffers.

Hover the mouse pointer around an indicator buffer defined value. The system will identify the buffer and will show the buffer name in the text label.

Mouse click to lock the buffer selection and navigate 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”.

Having proposed condition definition, click the left mouse button. The system will lock the currently identified condition and will ask for final acceptance.

How to:

The process for implementing buffer condition “disappear” is the same, except after locking the buffer we navigate right direction from the selection point to an area, 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 point an area, where the target condition to be defined actually happens.

How to:

The same sketching technique 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.

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 “turn down”. Same for “cross down”.

I will not further challenge the reader’s patience with repetitive detailed explanations of these similar techniques. Much more useful will be to work out the corresponding functionality in the actual environment.

Condition “cross up / down”

Condition “turn up / down”