Algorithmica
Research AB

Quantlab Editor and Debugger

for Quantlab version 3.1.2067 and later

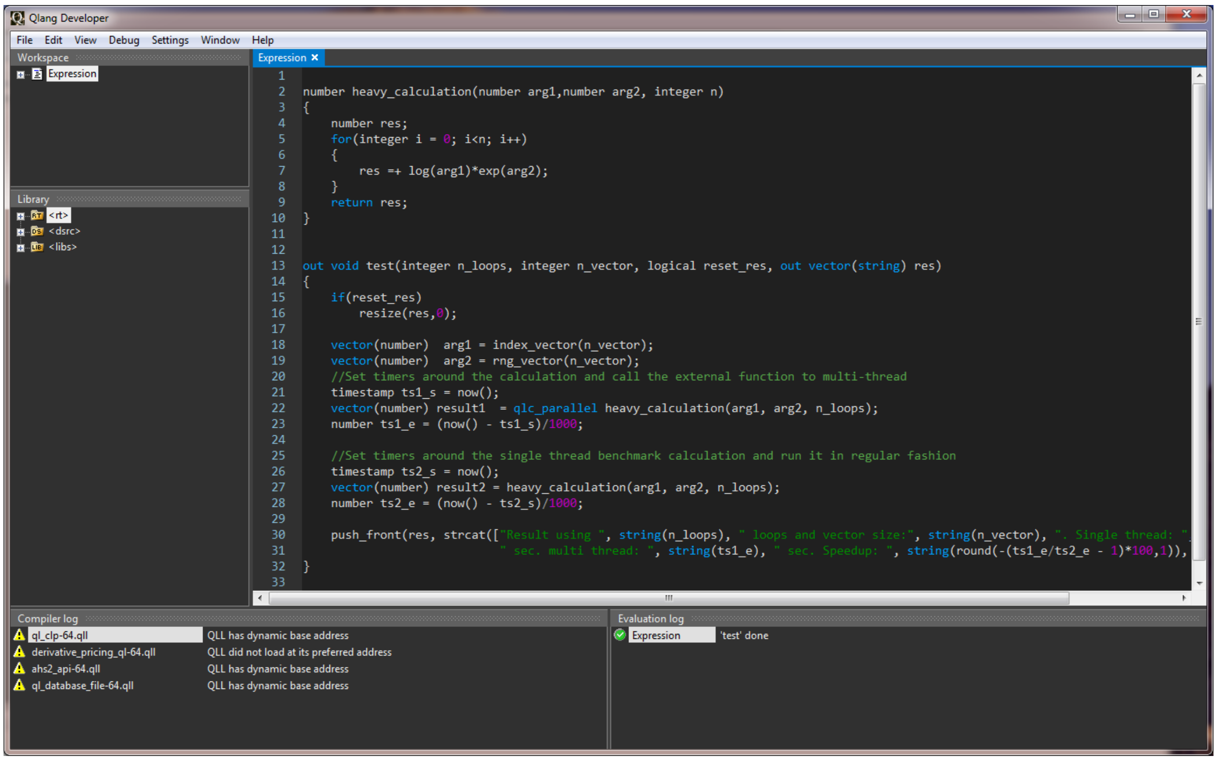
Table of Contents

Introduction to the Editor and Debugger environment

This document aims to highlight functions and features of the new editor and debugger environment in Quantlab introduced with version 3.1.2040 and later.

First, the environment is held in a separate windows structure and no longer as expressions within the Quantlab workspace itself. The workspace code is still saved in the workspace file.

To open the editor, press **Ctrl+L** or double click on the expression code in the workspace. This opens the editor environment in non-debug mode.



```

number heavy_calculation(number arg1,number arg2, integer n)
{
    number res;
    for(integer i = 0; i<n; i++)
    {
        res =+ log(arg1)*exp(arg2);
    }
    return res;
}


out void test(integer n_loops, integer n_vector, logical reset_res, out vector(string) res)
{
    if(reset_res)
        resize(res,0);

    vector(number)  arg1 = index_vector(n_vector);
    vector(number)  arg2 = rng_vector(n_vector);
    //Set timers around the calculation and call the external function to multi-thread
    timestamp ts1_s = now();
    vector(number) result1  = qlc_parallel heavy_calculation(arg1, arg2, n_loops);
    number ts1_e = (now() - ts1_s)/1000;

    //Set timers around the single thread benchmark calculation and run it in regular fashion
    timestamp ts2_s = now();
    vector(number) result2 = heavy_calculation(arg1, arg2, n_loops);
    number ts2_e = (now() - ts2_s)/1000;

    push_front(res, strcat(["Result using ", string(n_loops), " loops and vector size:", string(n_vector), ". Single thread: "
                            " sec. multi thread: ", string(ts1_e), " sec. Speedup: ", string(round(-(ts1_e/ts2_e - 1)*100,1)),
}

```

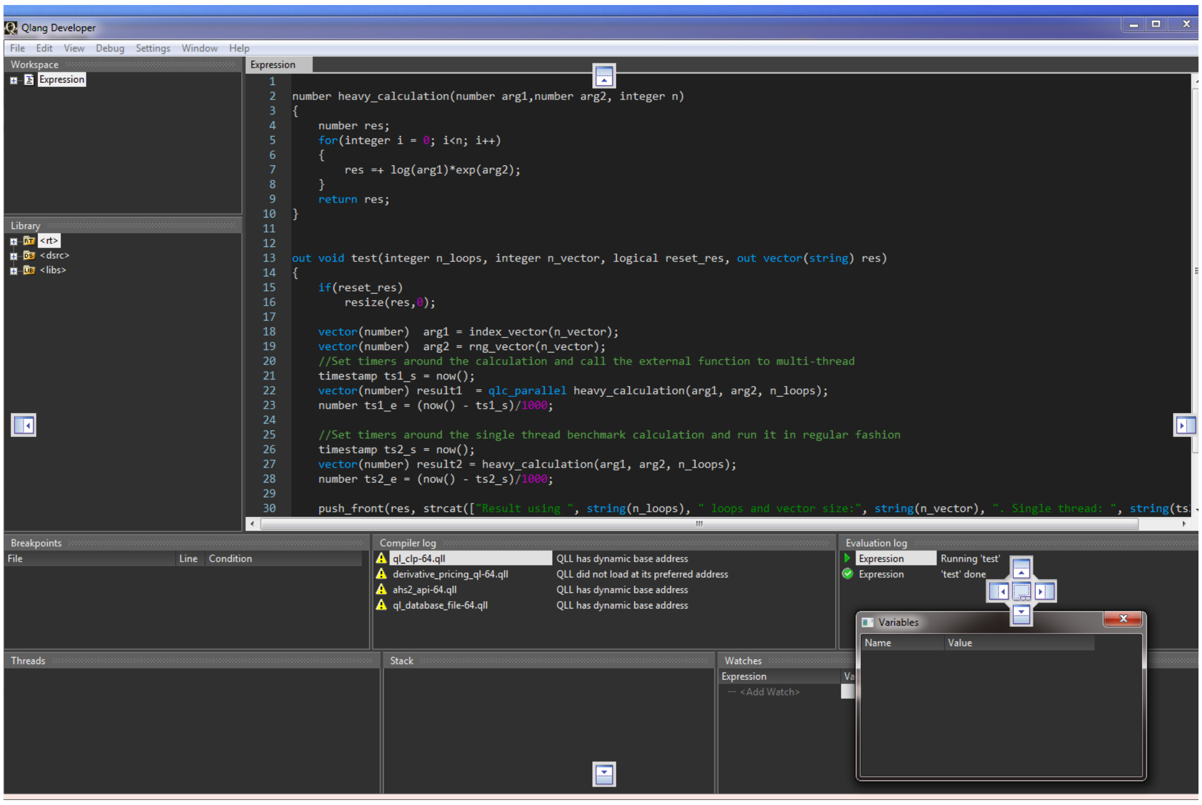
```
ql_clp-64.qll                QLL has dynamic base address
derivative_pricing_ql-64.qll QLL did not load at its preferred address
ahs2_api-64.qll              QLL has dynamic base address
ql_database_file-64.qll      QLL has dynamic base address
```

```
Expression   'test' done
```

Note that the compiler tab/log previously found in the workspace “view messages” window is now only found in the new editor.

The controls

By switching to debug mode using menu Debug | Toggle debugging (or **ctrl + shift + D**) will open a predefined set of small working windows. All can be moved around be simply grabbing the header frame and undocking or docking within the lager window. A set of guiding docking markers will assist.



```

number heavy_calculation(number arg1,number arg2, integer n)
{
    number res;
    for(integer i = 0; i<n; i++)
    {
        res =+ log(arg1)*exp(arg2);
    }
    return res;
}


out void test(integer n_loops, integer n_vector, logical reset_res, out vector(string) res)
{
    if(reset_res)
        resize(res,0);

    vector(number)  arg1 = index_vector(n_vector);
    vector(number)  arg2 = rng_vector(n_vector);
    //Set timers around the calculation and call the external function to multi-thread
    timestamp ts1_s = now();
    vector(number) result1  = qlc_parallel heavy_calculation(arg1, arg2, n_loops);
    number ts1_e = (now() - ts1_s)/1000;

    //Set timers around the single thread benchmark calculation and run it in regular fashion
    timestamp ts2_s = now();
    vector(number) result2 = heavy_calculation(arg1, arg2, n_loops);
    number ts2_e = (now() - ts2_s)/1000;

    push_front(res, strcat(["Result using ", string(n_loops), " loops and vector size:", string(n_vector), ". Single thread: ", string(ts
```

```
ql_clp-64.qll                  QLL has dynamic base address
derivative_pricing_ql-64.qll   QLL did not load at its preferred address
ahs2_api-64.qll                QLL has dynamic base address
ql_database_file-64.qll        QLL has dynamic base address
```

```
Expression   Running 'test'
Expression   'test' done
```

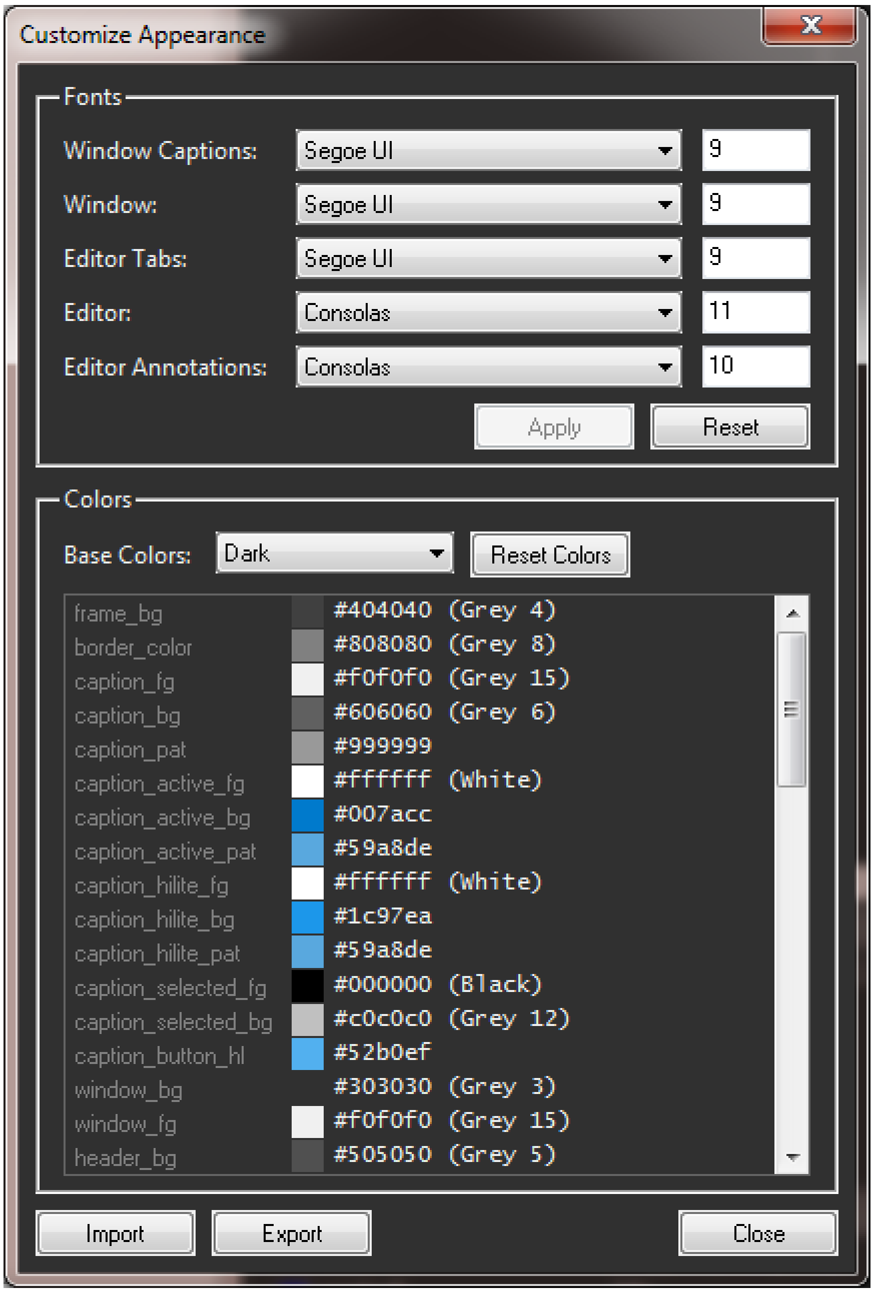
Windows can be switched on/off using the View menu or by simply detaching and closing them. Settings will be remembered between sessions in the user registry.

Following is a list of widows and their purpose:

Name	Usage
Main Editor	The editor window itself. Can be split into multiple windows both horizontal and vertically split.
Workspace	Displaying all Expression code files in a particular workspace
Library	Showing the library code tree. Right click in the tree to also see built-in functions.
Breakpoints	Display all break-points and set optional break conditions.
Threads	View running threads and manage in which thread to debug.
Compiler log	Show all compiler messages. Double click to go to any compilation errors.
Stack	View call stack when debugging.
Watches	Add watches for variables and expressions when stepping through code. (Replaces the "Immediate window" of the old debug environment.)
Evaluation log	Run-time messages from running the code.
Variables	List of variables and their in-scope values. Displays both scalar and vector values.
Profiler	Opens when toggling to File | Toggle Profiling (ctrl + shift + P). Used for performance profiling.

Environment appearance

Open menu Settings | Appearance to customize the environment. Font, fond size, and coloring can all be adapted to user preference.



Customize Appearance

Fonts

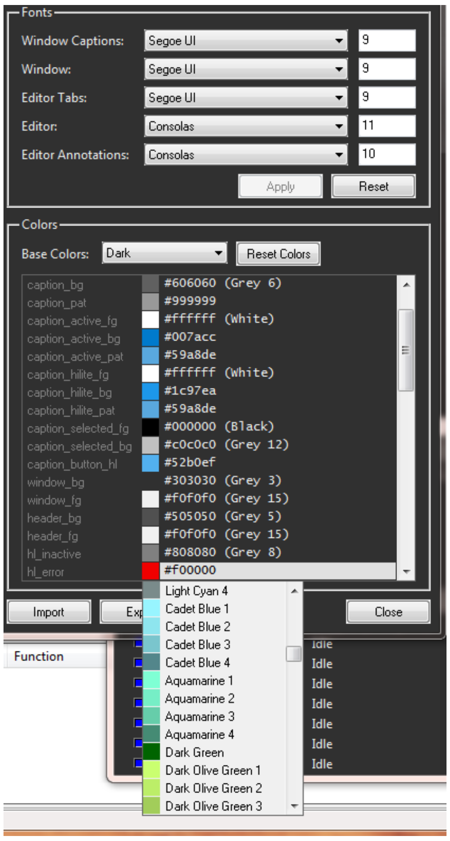
Window Captions:	Segoe UI	9
Window:	Segoe UI	9
Editor Tabs:	Segoe UI	9
Editor:	Consolas	11
Editor Annotations:	Consolas	10

Colors

Base Colors: Dark

frame_bg	⬛	#404040 (Grey 4)
border_color	⬛	#808080 (Grey 8)
caption_fg	⬜	#f0f0f0 (Grey 15)
caption_bg	⬛	#606060 (Grey 6)
caption_pat	⬛	#999999
caption_active_fg	⬜	#ffffff (White)
caption_active_bg	🟦	#007acc
caption_active_pat	🟦	#59a8de
caption_hilite_fg	⬜	#ffffff (White)
caption_hilite_bg	🟦	#1c97ea
caption_hilite_pat	🟦	#59a8de
caption_selected_fg	⬛	#000000 (Black)
caption_selected_bg	⬜	#c0c0c0 (Grey 12)
caption_button_hl	🟦	#52b0ef
window_bg	⬛	#303030 (Grey 3)
window_fg	⬜	#f0f0f0 (Grey 15)
header_bg	⬛	#505050 (Grey 5)

To get a predefined style, use the Base Colors drop-down. Using a base color scheme, all individual components can be adjusted. By double clicking on the color square, a custom color editor will open. By right-clicking on the color code itself, a menu of pre-defined named colors will open. (It is also possible to write the color code directly in the control box.



Fonts

Window Captions:	Segoe UI	9
Window:	Segoe UI	9
Editor Tabs:	Segoe UI	9
Editor:	Consolas	11
Editor Annotations:	Consolas	10

Colors

Base Colors: Dark

caption_bg	#606060 (Grey 6)
caption_pat	#999999
caption_active_fg	#ffffff (White)
caption_active_bg	#007acc
caption_active_pat	#59a8de
caption_hilite_fg	#ffffff (White)
caption_hilite_bg	#1c97ea
caption_hilite_pat	#59a8de
caption_selected_fg	#000000 (Black)
caption_selected_bg	#c0c0c0 (Grey 12)
caption_button_hl	#52b0ef
window_bg	#303030 (Grey 3)
window_fg	#f0f0f0 (Grey 15)
header_bg	#505050 (Grey 5)
header_fg	#f0f0f0 (Grey 15)
hl_inactive	#808080 (Grey 8)
hl_error	#F00000

Any alterations to color or fonts etc will be stored in the user registry. It is also possible to export the settings using the import / export buttons. This will enable sharing of a personalized scheme with others.

See appendix for the complete list and description of the settings.

Short-cuts for Windows, Editor, and Debugger

There are many short-cuts available. Some of these are context sensitive and only apply within that context. Below is a summary of the most import short-cuts.

Context	Short-cut key(s)	Description
Main	F1	Open function browser
	Ctrl + Shift + F, H	Open Global Find /Replace in library and/or Editor windows. See detail below.
	Ctrl + Shift + D	Toggle Debugging on/off (will generate re-compile)
	Ctrl + Shift + P	Toggle Profiling on/off
	Ctrl + P	Find library file (only)
	Ctrl + Page up/down	Cycle through open windows and expressions
	Ctrl + Tab	Cycle through Tabs in Editor window
	Ctrl + 2	Open additional Editor window vertical split
	Ctrl + W	Close current Editor Tab
	Ctrl + 3	Open additional Editor window horizontal split
	Ctrl + 1	Unsplit vertical / horizontal split
Glob. Find /Replace	Ctrl + I, C	Toggle case (in)sensitive search
	Ctrl + F	Match any string
	Ctrl + W	Match whole word only
	Ctrl + R	Search using regular expressions
	Ctrl + D	Search in current document only
	Ctrl + S	Search in selection only
	Ctrl + L	Search in Library only
	Ctrl + K	Search in Workspace code only
	Ctrl + O	Search in all open documents only
	Ctrl + U	Search in all documents
	Ctrl + Alt + Enter	Replace All
Editor	Ctrl + F	Quick find in editor
	Ctrl + H	Open Find and replace for current document only.
	Ctrl + A	Select All
	Ctrl + Z	Undo
	Ctrl + Y	Redo
	Ctrl + X, C, V	Cut, copy, paste
	Ctrl + Shift + X, C	Cut or copy entire line
	Ctrl + T	Transpose lines
	Tab	Indent using tab
	Shift + Tab	Reverse indent
	Ctrl + Space	Show tooltip when inside brackets ( )
	Alt + right/left arrow	In tooltip, cycle through function overloading
	Alt + Up/Down	To move line(s) up and down
	Ctrl (+ Shift) + U	Switch to lower / upper case
	Ctrl + D	For multiple line selection of words, add next occurrence to multi caret editing.
	Ctrl + mouse wheel	Zoom in and out in Editor code window
	Ctrl (+Shift) + 0	Go-to definition (when cursor is on variable, function, class or enum)
	Ctrl + Q	Jump back from previous go-to-definition
	Ctrl + Shift + I	To automatically indent the entire file or selection
Debugger	F7	Compile all
	Ctrl + F7	Check for compile errors in editor window
	F10	Step Over
	F11	Step Into (function)
	Shift + F11	Step out of function
	F5	Continue (for highlighted thread, when using multiple)
	Shift + F5	Continue all threads
	Ctrl + F5	Continue all other than active thread
	F9	Toggle break-point on row
	Ctrl + F9	Remove break-point

Many shortcuts used in Scintilla based editors will also work in the Qlang Editor.

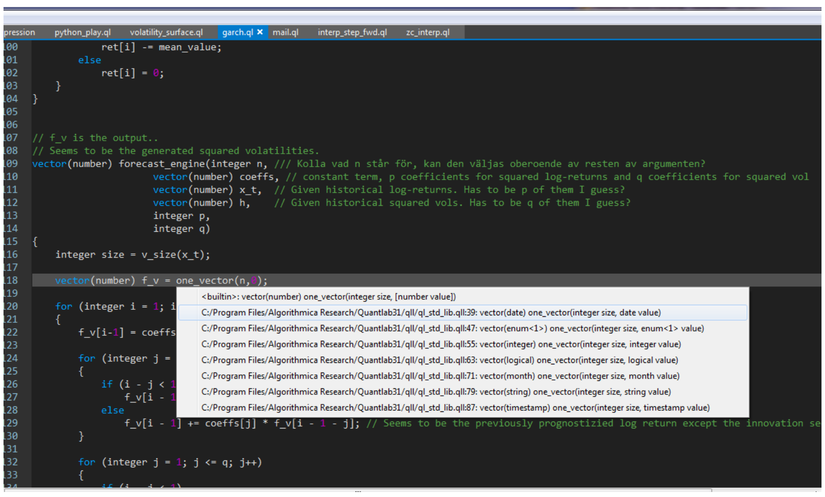
The Editor

Using the editor should work much similar to other popular editors. Many features from popular editors such as Notepad++, Visual Studio Code, and Sublime can be found in the new Quantlab Editor. The main reason for using the Quantlab built-in editor is its direct link to the (multi-threaded) debugger.

We have already addressed the possibility to work with personalized appearance, by setting fonts and coloring. Below are some of the key features described one-by-one.

Go-to-definition

By right-clicking (**ctrl + 0**) on an Enumerator, Function, Class, Class member, go-to-definition will appear in the menu. If the function is openly readable and only exist in one instance, it will jump straight to that place in the expression window or library. If there are multiple instances of the function, a selectable list will open. If the function is built-in or used from a compiled qll, the list will high-light <built-in> and the relevant qll will be highlighted.



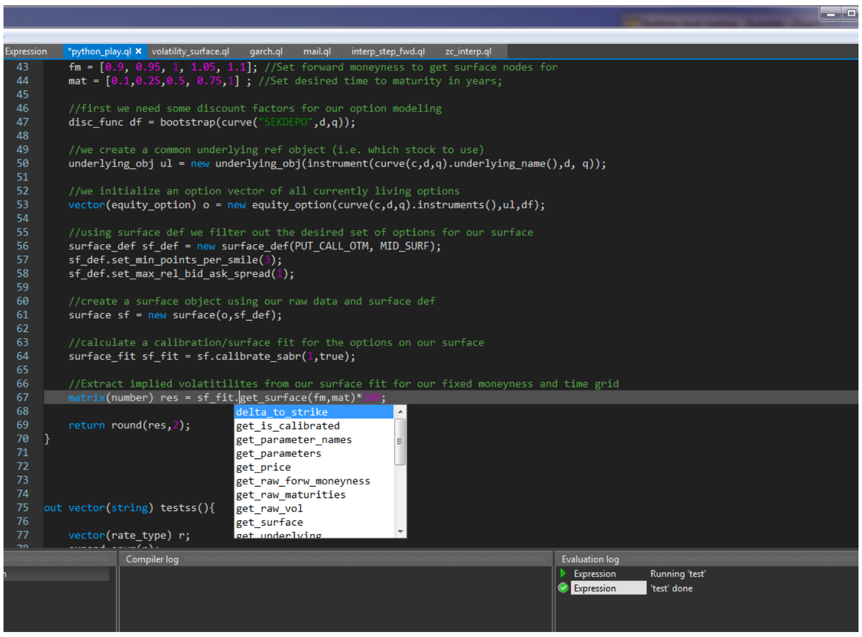
```
            ret[i] -= mean_value;
        else
            ret[i] = 0;
    }
}


// f_v is the output..
// Seems to be the generated squared volatilities.
vector(number) forecast_engine(integer n, /// Kolla vad n står för, kan den väljas oberoende av resten av argumenten?
                     vector(number) coeffs, // constant term, p coefficients for squared log-returns and q coefficients for squared vol
                     vector(number) x_t,  // Given historical log-returns. Has to be p of them I guess?
                     vector(number) h,    // Given historical squared vols. Has to be q of them I guess?
                     integer p,
                     integer q)
{
    integer size = v_size(x_t);

    vector(number) f_v = one_vector(n,0);

    for (integer i = 1; i
    {
        f_v[i-1] = coeffs

        for (integer j =
        {
            if (i - j < 1
                f_v[i - 1
            else
                f_v[i - 1] += coeffs[j] * f_v[i - 1 - j]; // Seems to be the previously prognostizied log return except the innovation se
        }

        for (integer j = 1; j <= q; j++)
        {
```

```
<builtin>: vector(number) one_vector(integer size, [number value])
C:/Program Files/Algorithmica Research/Quantlab31/qll/ql_std_lib.qll:39: vector(date) one_vector(integer size, date value)
C:/Program Files/Algorithmica Research/Quantlab31/qll/ql_std_lib.qll:47: vector(enum<1>) one_vector(integer size, enum<1> value)
C:/Program Files/Algorithmica Research/Quantlab31/qll/ql_std_lib.qll:55: vector(integer) one_vector(integer size, integer value)
C:/Program Files/Algorithmica Research/Quantlab31/qll/ql_std_lib.qll:63: vector(logical) one_vector(integer size, logical value)
C:/Program Files/Algorithmica Research/Quantlab31/qll/ql_std_lib.qll:71: vector(month) one_vector(integer size, month value)
C:/Program Files/Algorithmica Research/Quantlab31/qll/ql_std_lib.qll:79: vector(string) one_vector(integer size, string value)
C:/Program Files/Algorithmica Research/Quantlab31/qll/ql_std_lib.qll:87: vector(timestamp) one_vector(integer size, timestamp value)
```

Call-tip and member completion

When writing a function and setting the first function bracket, a call-tip with all alternative variants of the function will appear. By clicking on the right / left arrow (**alt + right/left arrow key**) will cycle through all the variants. The current variable will highlight. If the call-tip is lost, pressing ctrl + space will get it back when inside the brackets.

When working with an object, class or enumerator, its available and public members will appear in a drop-down in alphabetical order.



```
    fm = [0.9, 0.95, 1, 1.05, 1.1]; //Set forward moneyness to get surface nodes for
    mat = [0.1,0.25,0.5, 0.75,1] ; //Set desired time to maturity in years;

    //first we need some discount factors for our option modeling
    disc_func df = bootstrap(curve("SEKDEPO",d,q));

    //we create a common underlying ref object (i.e. which stock to use)
    underlying_obj ul = new underlying_obj(instrument(curve(c,d,q).underlying_name(),d, q));

    //we initialize an option vector of all currently living options
    vector(equity_option) o = new equity_option(curve(c,d,q).instruments(),ul,df);

    //using surface def we filter out the desired set of options for our surface
    surface_def sf_def = new surface_def(PUT_CALL_OTM, MID_SURF);
    sf_def.set_min_points_per_smile(3);
    sf_def.set_max_rel_bid_ask_spread(1);

    //create a surface object using our raw data and surface def
    surface sf = new surface(o,sf_def);

    //calculate a calibration/surface fit for the options on our surface
    surface_fit sf_fit = sf.calibrate_sabr(1,true);

    //Extract implied volatitilites from our surface fit for our fixed moneyness and time grid
    matrix(number) res = sf_fit.get_surface(fm,mat)*100;

    return round(res,2);
}



out vector(string) testss(){

    vector(rate_type) r;
```

```
Expression    Running 'test'
Expression    'test' done
```

Multi-caret/cursor editing

By double clicking on a variable word or other character in the editor, all instances will highlight. To enable multiple simultaneous editing, repeated use of **Ctrl + D** will add next appearance and leave a caret/cursor point at all those places. Then the all can be edited at the same time.

Using **Ctrl + selection** (using mouse left click) in code, will leave a caret/cursor at all places where clicked. Then multi-instance editing can be done.

Using **Alt + selection** (using mouse left click) will highlight a “box” of lines and open up for multi-caret/cursor editing at the point where the mouse cursor ends.

Revert all Changes / Revert to Compiled State

When coding in an old library or workspace, revert all changes will undo all changes in the current file/expression done in the current session. To un revert, either revert to latest compiled state, or “redo” using ctrl + shift + z.

Revert to compiled state will do just that – revert to latest compiled state in the current file and session.

Other Editor Settings

A number of visual features and settings can be found in the Setting menu. The most important are;

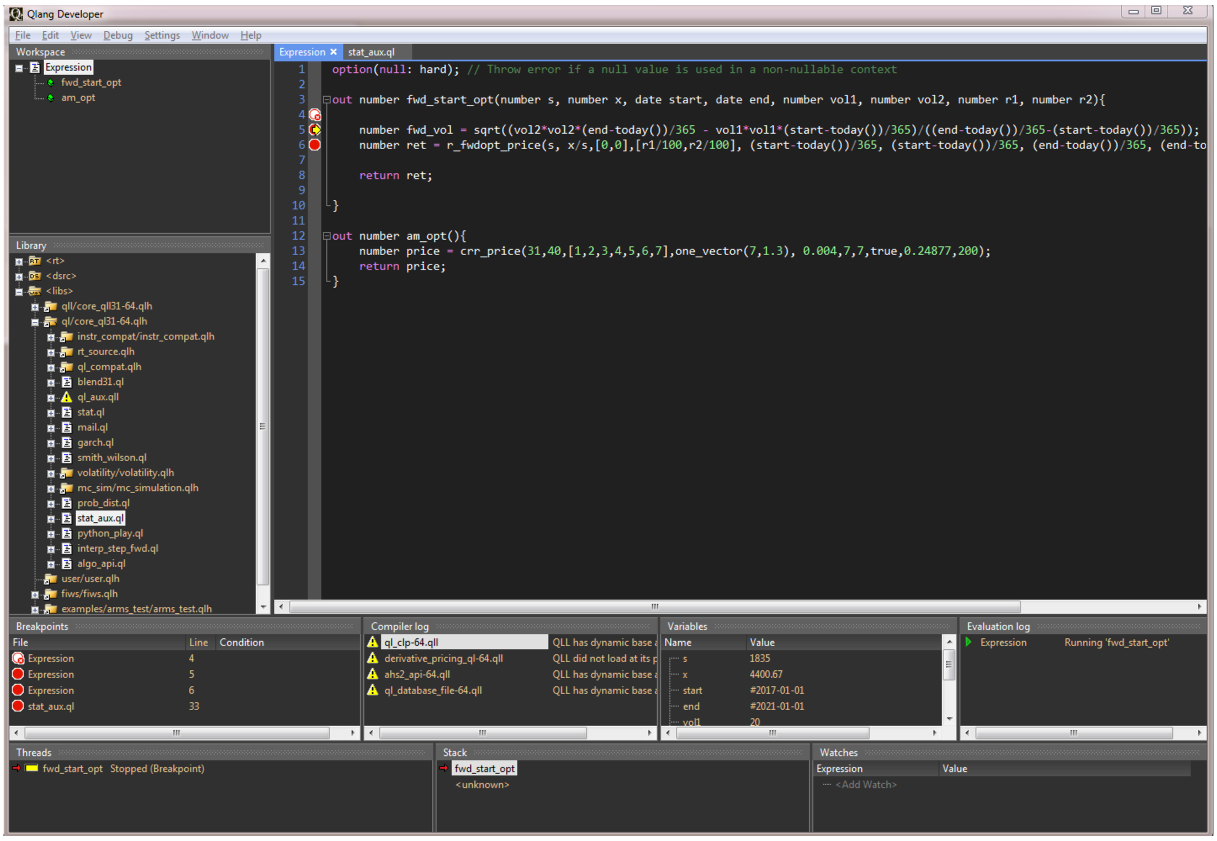
Setting	Description
Appearance	Open the menu for all font and coloring options
Show line numbers	Turn line numbers on/off
Show whitespace	Display tab / space in the editor
Show line endings	Display all line-endings
Highlight current line	Change display of current line highlighting, none, box, filled etc
Caret width	Number of pixel points to display the caret/cursor line with
Show diagnostics inline	Turn on to show message in-line with the code in the editor
Show diagnostics in the margin	Turn on to show a graphic element with mouse over diagnostics in the left column
Auto indent	Turn auto indentation on/off
Indent and tab size	Set indent and tab size
Indent with spaces	Turn on to set indents using space instead of tab
Show indentation guides	Show vertical lines to indicate where indentation is set on each row
Enable Code Folding	Show / hide the vertical folding markers on the left-hand side of the code. Can fold code per bracket tuple.
Auto-save library recovery backup	Turn on/off auto-save library file. A backup will be created in the same folder and with the same name as the original with a ~bu~ file type extension. Any time the library file is in uncompiled state the back-up will be created.

Debugging

Start by switching to debug mode using the menu Debug | Toggle Debugging (Ctrl + Shift + D). A default window layout will appear. Move, close and resize to your preferences.

Breakpoints and conditional breakpoints are set using the mouse on the vertical bar next to row numbering. Can also be switched on/off using F9 for the line where the cursor is at.

All breakpoints are displayed in the Breakpoints window. Also, breakpoints in the library files will show here. If an invalid breakpoint is set, it will show as an empty circle.



```
option(null: hard); // Throw error if a null value is used in a non-nullable context

out number fwd_start_opt(number s, number x, date start, date end, number vol1, number vol2, number r1, number r2){

    number fwd_vol = sqrt((vol2*vol2*(end-today())/365 - vol1*vol1*(start-today())/365)/((end-today())/365-(start-today())/365));
    number ret = r_fwdopt_price(s, x/s,[0,0],[r1/100,r2/100], (start-today())/365, (start-today())/365, (end-today())/365, (end-to

    return ret;

}

out number am_opt(){
    number price = crr_price(31,40,[1,2,3,4,5,6,7],one_vector(7,1.3), 0.004,7,7,true,0.24877,200);
    return price;
}
```

Breakpoints

File	Line	Condition
Expression	4	
Expression	5	
Expression	6	
stat_aux.ql	33	

Compiler log

ql_clp-64.qll	QLL has dynamic base a
derivative_pricing_ql-64.qll	QLL did not load at its p
ahs2_api-64.qll	QLL has dynamic base a
ql_database_file-64.qll	QLL has dynamic base a

Variables

Name	Value
s	1835
x	4400.67
start	#2017-01-01
end	#2021-01-01
vol1	20

Evaluation log

Expression	Running 'fwd_start_opt'

Threads

fwd_start_opt Stopped (Breakpoint)

Stack

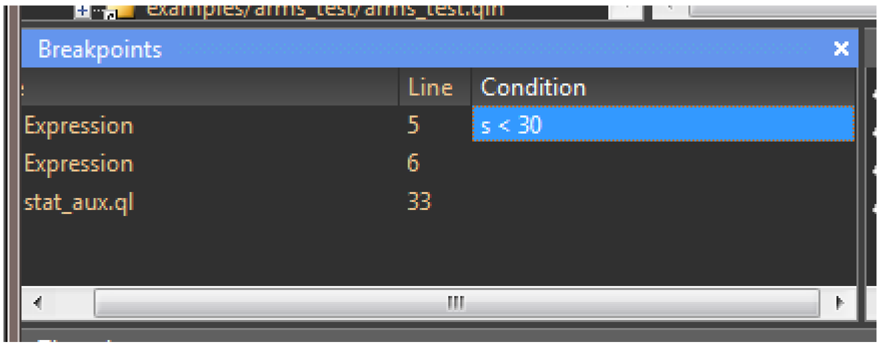
fwd_start_opt
<unknown>

Watches

Expression	Value
<Add Watch>	

Conditional breakpoints

To create a conditional breakpoint, simply enter a valid logical expression directly in the Breakpoints window box, in the Condition column.



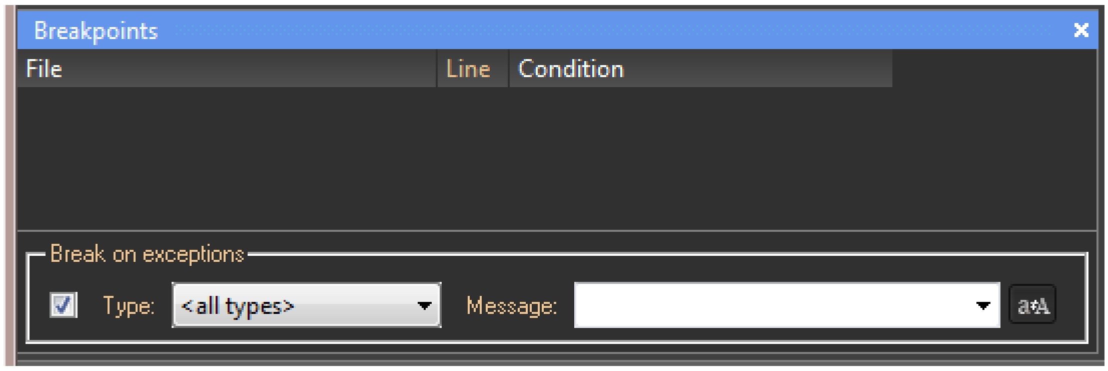
	Line	Condition
Expression	5	s < 30
Expression	6	
stat_aux.ql	33	

Conditional breakpoints will be market with a plus in the red circle.

Tip! As the conditional breakpoints are code lines that are actually compiled into the code at the point where inserted, any valid Qlang expression can be used given that it returns a logical.
Example: v_size(vect) > 2 && vect[0] != 0.

Breakpoints can be stored on file, shared and loaded into other Quantlab sessions. This can be done using the menu Debug | Import / Export Breakpoints.

As a complement to setting break points manually, "Break on exceptions" can be configured so that it automatically breaks on a certain exception. Filter can be set on a specific type and/or thrown message.



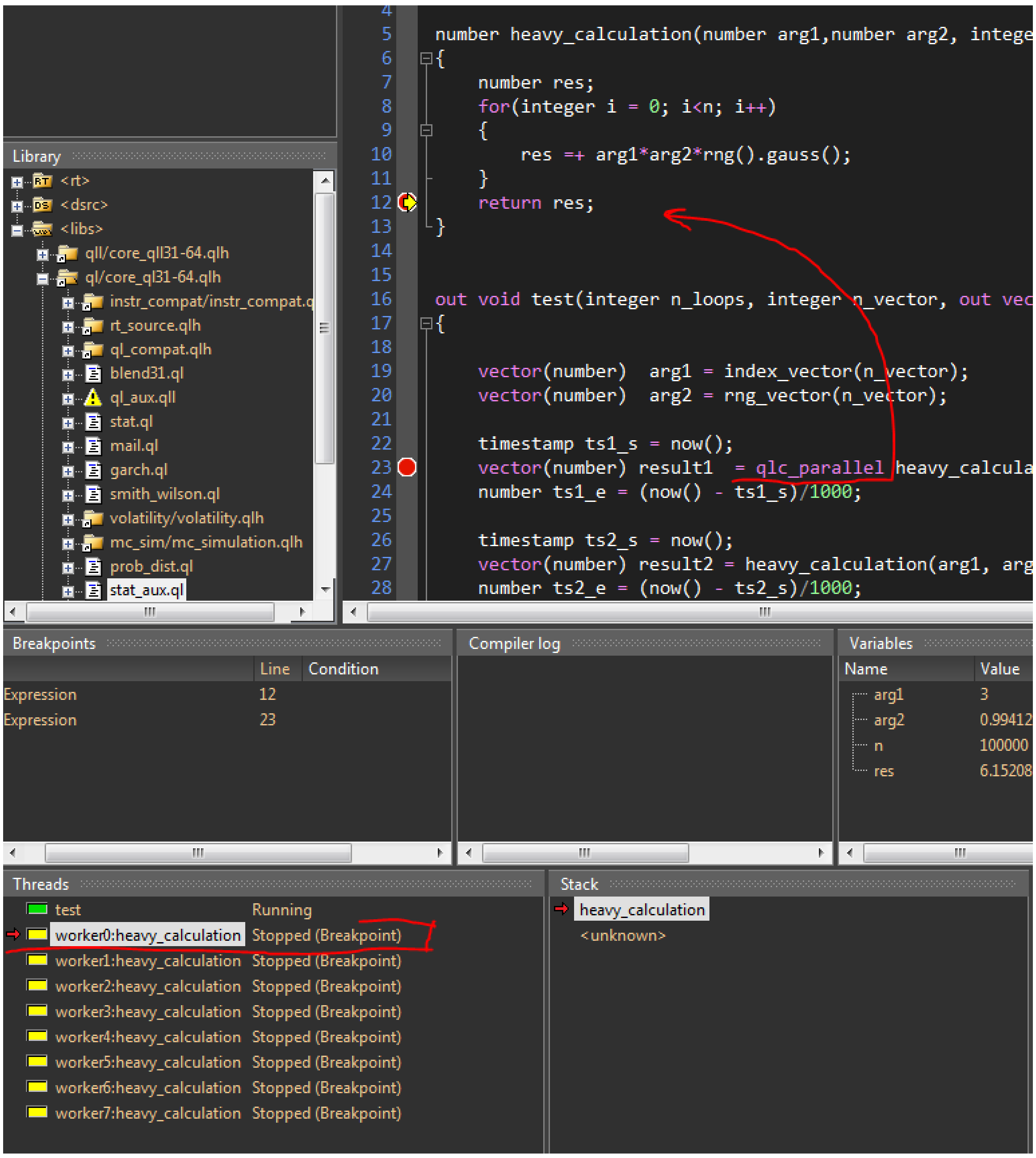
Break on exceptions: ☑ Type: <all types> Message:

Multi-thread debug

If running a multi-threaded program, each individual thread can be debugged using the list in the Threads window.

If a breakpoint is inserted in the function that is evaluated using multiple parallel threads. A list will appear. Initially, the pointer will be on the main worker thread, and not show the yellow stop cursor. By dubbel clicking on any of the threads in the window, its location and variable contents will show.

F5 will continue only the highlighted thread, Shift + F5 all threads, and Ctrl + F5 all but the highlighted thread. Note that when continuing a worker thread individually, it will finish the task for all threads. However, the results will not be released and displayed in the output until also the other threads are released from their respective breakpoints even though there is no more work to perform.



Library
- <rt>
- <dsrc>
- <libs>
  - qll/core_qll31-64.qlh
  - ql/core_ql31-64.qlh
    - instr_compat/instr_compat.q
    - rt_source.qlh
    - ql_compat.qlh
    - blend31.ql
    - ql_aux.qll
    - stat.ql
    - mail.ql
    - garch.ql
    - smith_wilson.ql
    - volatility/volatility.qlh
    - mc_sim/mc_simulation.qlh
    - prob_dist.ql
    - stat_aux.ql

```
5   number heavy_calculation(number arg1,number arg2, intege
6   {
7       number res;
8       for(integer i = 0; i<n; i++)
9       {
10          res =+ arg1*arg2*rng().gauss();
11      }
12      return res;
13  }
14
15
16  out void test(integer n_loops, integer n_vector, out vec
17  {
18
19      vector(number)  arg1 = index_vector(n_vector);
20      vector(number)  arg2 = rng_vector(n_vector);
21
22      timestamp ts1_s = now();
23      vector(number) result1  = qlc_parallel heavy_calcula
24      number ts1_e = (now() - ts1_s)/1000;
25
26      timestamp ts2_s = now();
27      vector(number) result2 = heavy_calculation(arg1, arg
28      number ts2_e = (now() - ts2_s)/1000;
```

Breakpoints

	Line	Condition
Expression	12	
Expression	23	

Compiler log

Variables

Name	Value
arg1	3
arg2	0.99412
n	100000
res	6.15208

Threads

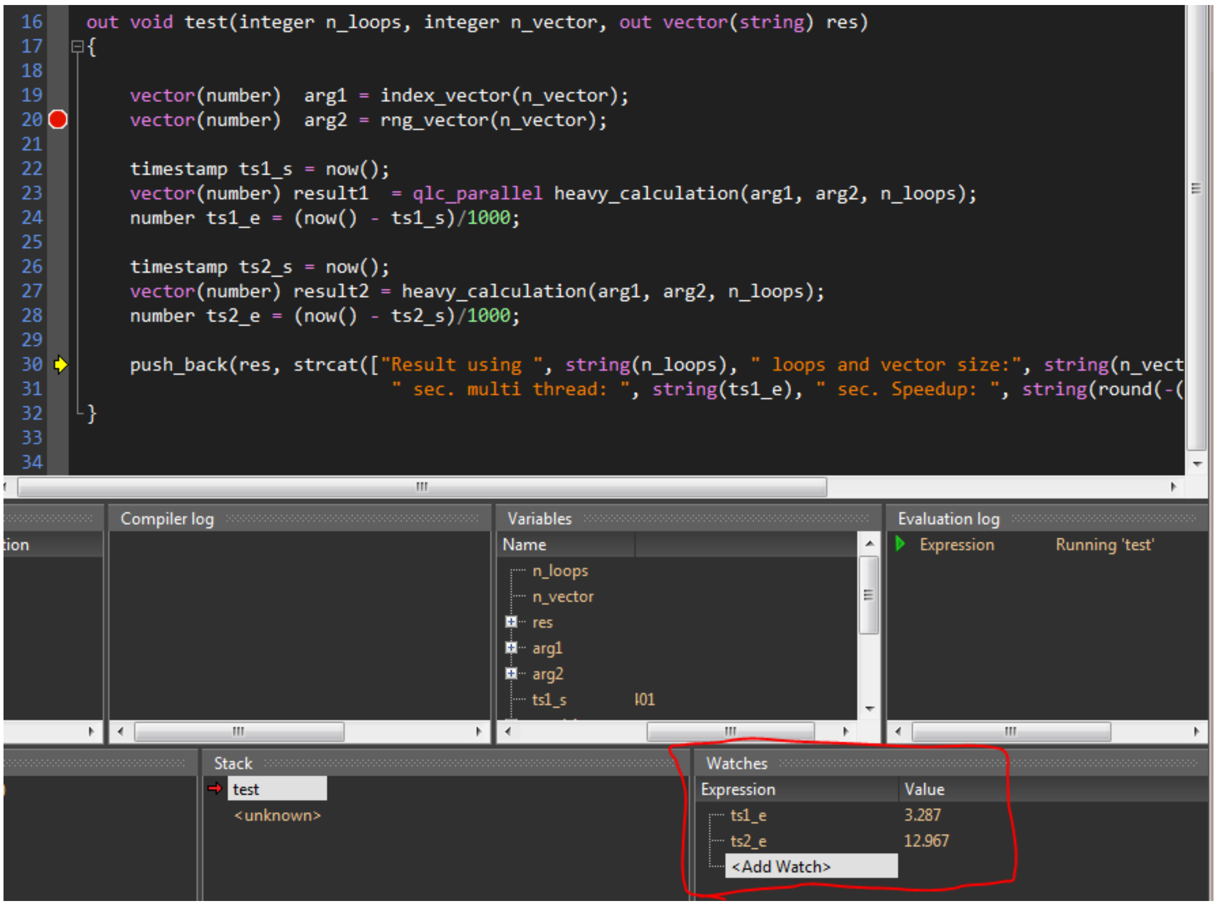
test	Running
worker0:heavy_calculation	Stopped (Breakpoint)
worker1:heavy_calculation	Stopped (Breakpoint)
worker2:heavy_calculation	Stopped (Breakpoint)
worker3:heavy_calculation	Stopped (Breakpoint)
worker4:heavy_calculation	Stopped (Breakpoint)
worker5:heavy_calculation	Stopped (Breakpoint)
worker6:heavy_calculation	Stopped (Breakpoint)
worker7:heavy_calculation	Stopped (Breakpoint)

Stack
- heavy_calculation
- <unknown>

Watches

In previous versions of Quantlab, the way to view and interact with a variable during debug was via the Immediate window. This has now been replaced by the Watches window, having the same functionality. The expression can be any type of valid Qlang code, at minimum showing variable's contents when stepping in the function.

Note that the watch expressions will stay in the Watch window even though stepping out of scope for that particular variable. It will then show "Undeclared identifier" if not yet passed or just null value.



```
out void test(integer n_loops, integer n_vector, out vector(string) res)
{

    vector(number)  arg1 = index_vector(n_vector);
    vector(number)  arg2 = rng_vector(n_vector);

    timestamp ts1_s = now();
    vector(number) result1  = qlc_parallel heavy_calculation(arg1, arg2, n_loops);
    number ts1_e = (now() - ts1_s)/1000;

    timestamp ts2_s = now();
    vector(number) result2 = heavy_calculation(arg1, arg2, n_loops);
    number ts2_e = (now() - ts2_s)/1000;

    push_back(res, strcat(["Result using ", string(n_loops), " loops and vector size:", string(n_vect
                          " sec. multi thread: ", string(ts1_e), " sec. Speedup: ", string(round(-(
}
```

Variables

Name	
n_loops	
n_vector	
res	
arg1	
arg2	
ts1_s	I01

Evaluation log

Expression   Running 'test'

Stack

test
<unknown>

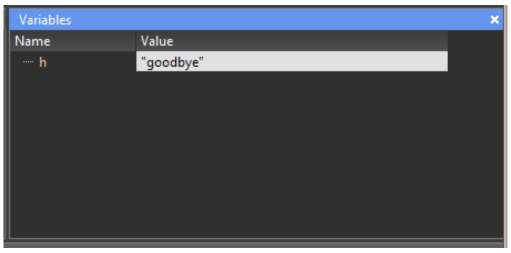
Watches

Expression	Value
ts1_e	3.287
ts2_e	12.967
<Add Watch>	

Variables

In the variables window, all variables in a debug function will display their contents. It will try to show both simple types and object types if possible. For viewing your own class members etc please view the section on "*Adding custom debug value browsing*" below.

Note that you can interact with the values in the variables by simply editing in the control-box. This is equivalent to how the "Immediate" window used to work.



Variables

Name	Value
h	"goodbye"

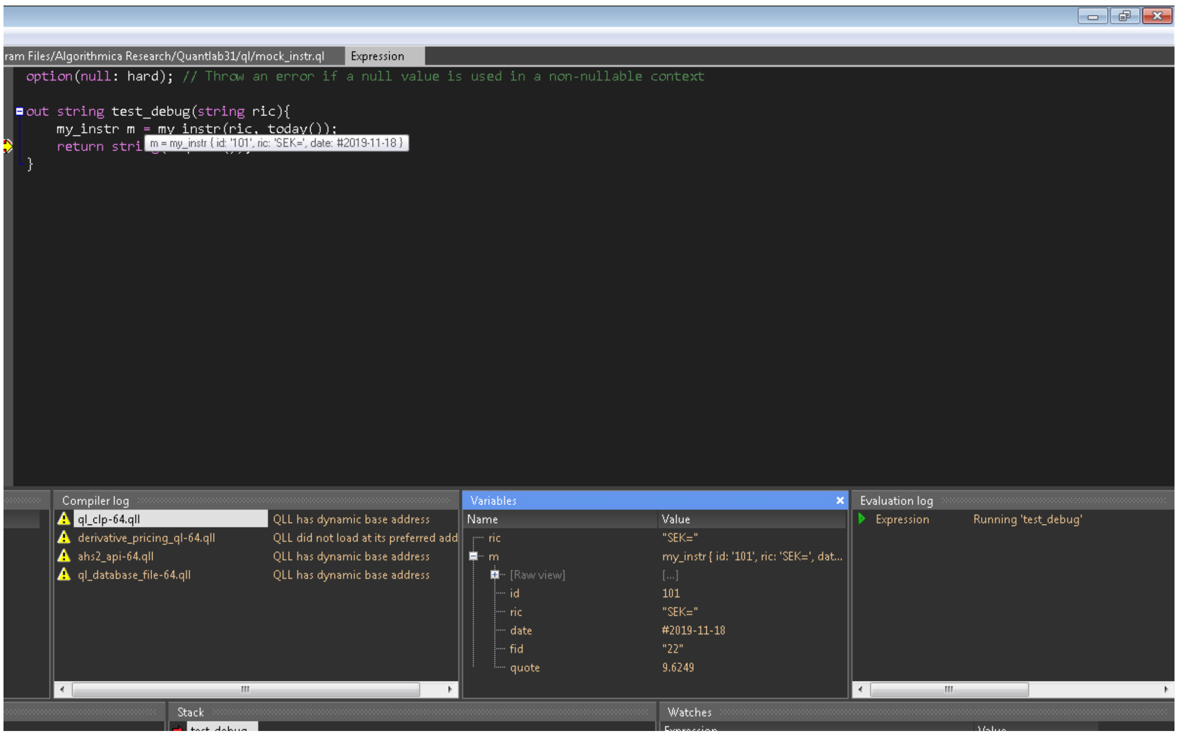
Adding custom debug value browsing

When working with objects and classes, the default is to show the class name in the Variables view and the Tooltip when debugging. By adding your own custom debug value functions, __dbg_print(), and __dbg_browse() to your class, the chosen member values will show in both Tooltips and Variable view. This can be very useful when working and debugging large data structures that is frequently visited. Using the auxiliary custom defined browse and print, a particular class data is easily found.

The double underscore prefix ensures that these member functions are automatically hidden from code completion and function browser view.

Below is a complete mock-up example of adding the __dbg_print() and __dbg_browse() to a simple class.

```
class my_instr
    option(category: "Curve and Instrument")
{
public:
    my_instr(string ric, date d, string fid);

    integer                 id();
    string                  name();
    number                  quote();

    void                    __dbg_print(__dbg_label);
    void                    __dbg_browse(__dbg_split);

public:
    integer                 id_;
    string                  ric_;
    date                    d_;
    string                  fid_;
    number                  quote_;
};

my_instr my_instr(string ric, date d, string fid = "22")
    option(category: "Curve and Instrument")
{
    return new my_instr(ric, d, fid);
}

my_instr.my_instr(string ric, date d, string fid)
    : id_(-1),
      ric_(ric),
      d_(d),
      fid_(fid),
      quote_(rt.get_num(ric, fid, "RFA", "IDN_SELECTFEED"))
{
    switch (ric) {
    case "SEK=": id_ = 101; break;
    case "EUR=": id_ = 102; break;
    case "GBP=": id_ = 103; break;
    case "NOK=": id_ = 104; break;
    case "JPY=": id_ = 105; break;
    case "AUD=": id_ = 106; break;
    }
}
integer my_instr.id()
{
    return id_;
}
string my_instr.name()
{
    return ric_;
}
number my_instr.quote()
{
    return quote_;
}
void my_instr.__dbg_print(__dbg_label l)
{
    l.set_text(strcat([ "my_instr { id: '", string(id_),
                        "', ric: '", ric_,
                        "', date: #", string(d_),
                        " }" ]));
}
void my_instr.__dbg_browse(__dbg_split s)
{
    s.resize(5);

    s.set_text(0, "id");
    s.set_value(0, id_);
    if (id_ < 0)
        s.set_hl(0, __dbg_hl.INACTIVE);

    s.set_text(1, "ric");
    s.set_value(1, ric_);
    s.set_text(2, "date");
    s.set_value(2, d_);
    s.set_text(3, "fid");
    s.set_value(3, fid_);

    s.set_text(4, "quote");
    s.set_value(4, quote_);
}
```



ram Files/Algorithmica Research/Quantlab31/ql/mock_instr.ql   Expression

```
option(null: hard); // Throw an error if a null value is used in a non-nullable context

out string test_debug(string ric){
    my_instr m = my_instr(ric, today());
    return stri
}
```

```
m = my_instr { id: '101', ric: 'SEK=', date: #2019-11-18 }
```

Compiler log

```
ql_clp-64.qll                  QLL has dynamic base address
derivative_pricing_ql-64.qll   QLL did not load at its preferred add
ahs2_api-64.qll                QLL has dynamic base address
ql_database_file-64.qll        QLL has dynamic base address
```

Variables

Name	Value
ric	"SEK="
m	my_instr { id: '101', ric: 'SEK=', dat...
[Raw view]	[...]
id	101
ric	"SEK="
date	#2019-11-18
fid	"22"
quote	9.6249

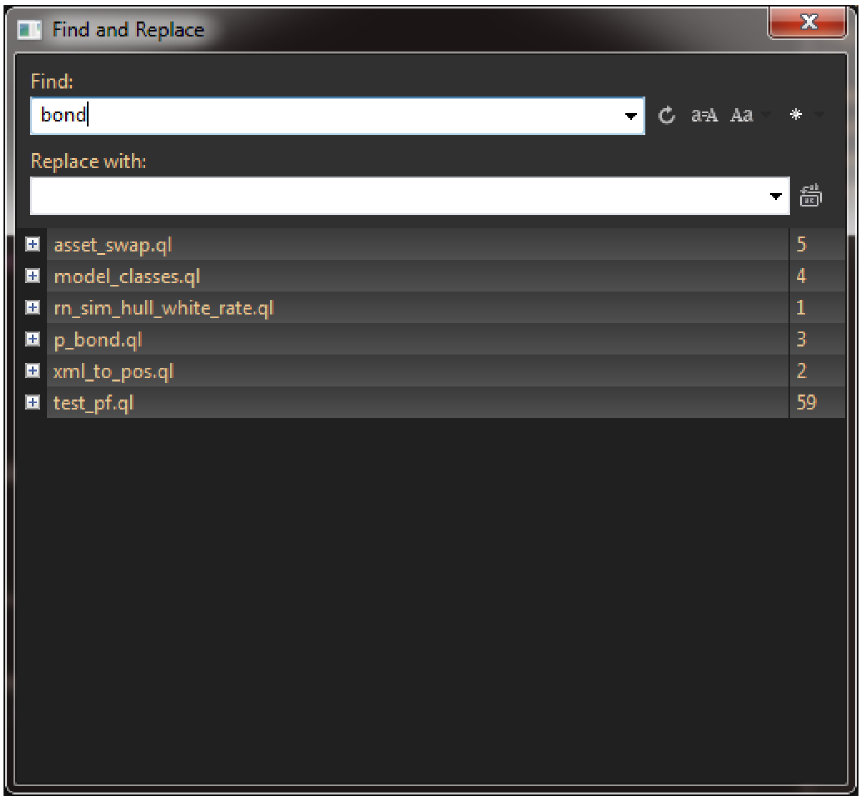
Evaluation log

```
Expression    Running 'test_debug'
```

Result of running the mock example above (note also the tooltip on mouse over)

Global Find and Replace

Ctrl + Shift + F / H will open the global find and replace window.



bond

asset_swap.ql	5
model_classes.ql	4
rn_sim_hull_white_rate.ql	1
p_bond.ql	3
xml_to_pos.ql	2
test_pf.ql	59

Searching can be performed in different contexts.

1) In the Current Document
2) In Selection only
3) In Library files
4) In Workspace expressions
5) All Open Documents
6) All Documents

Further, the search can be done using case sensitive or not, matching whole words or not.

Occurrences will be highlighted when the relevant file is expanded. Click on the line of code to go to / and open that file.

Advanced search can be done using regex syntax.

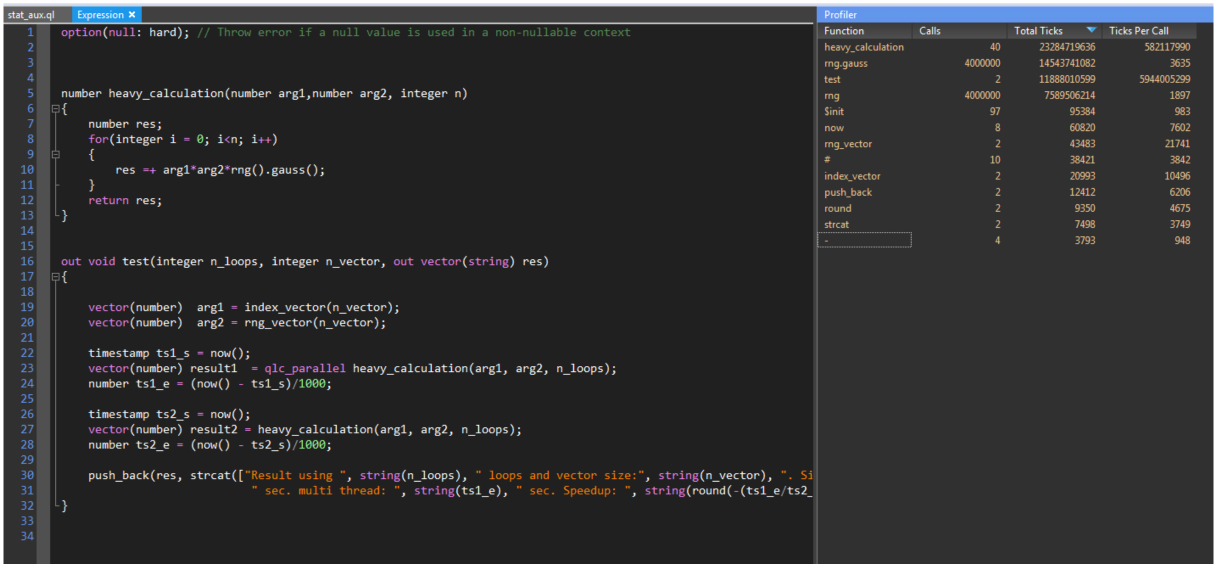
To replace all occurrences, press on the icon right of the Replace with dialog (or press Ctrl + Alt + H). To replace one found match at a time, simply press Ctrl + H, repeatedly.

To revert changes, open each file and use Ctrl + Z to revert the changes in that file.

Using the Profiler

Open the profiler using menu View | Profiler (or Ctrl + Shift + P).

In profile mode there are two options, showing the results for each run and then resetting the counter, or aggregated with total and average run times. To change between the two, right click on the Profiler pane and choose to Accumulate or not. Here you may also clear the results and start over.



```
option(null: hard); // Throw error if a null value is used in a non-nullable context



number heavy_calculation(number arg1,number arg2, integer n)
{
    number res;
    for(integer i = 0; i<n; i++)
    {
        res =+ arg1*arg2*rng().gauss();
    }
    return res;
}


out void test(integer n_loops, integer n_vector, out vector(string) res)
{

    vector(number)  arg1 = index_vector(n_vector);
    vector(number)  arg2 = rng_vector(n_vector);

    timestamp ts1_s = now();
    vector(number) result1  = qlc_parallel heavy_calculation(arg1, arg2, n_loops);
    number ts1_e = (now() - ts1_s)/1000;

    timestamp ts2_s = now();
    vector(number) result2 = heavy_calculation(arg1, arg2, n_loops);
    number ts2_e = (now() - ts2_s)/1000;

    push_back(res, strcat(["Result using ", string(n_loops), " loops and vector size:", string(n_vector), ". Si
                            " sec. multi thread: ", string(ts1_e), " sec. Speedup: ", string(round(-(ts1_e/ts2_
}


```

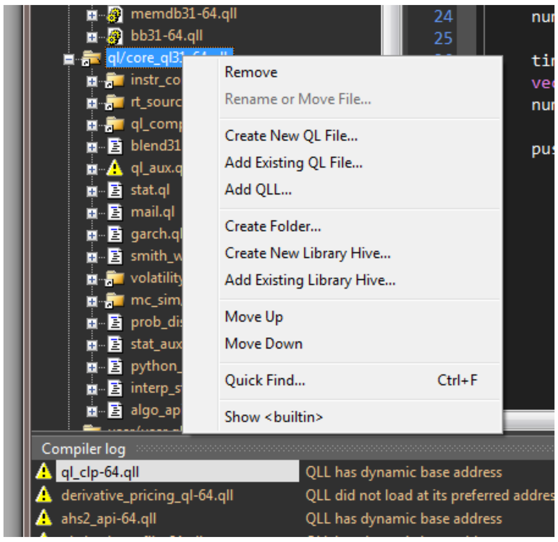
Function	Calls	Total Ticks	Ticks Per Call
heavy_calculation	40	23284719636	582117990
rng.gauss	4000000	14543741082	3635
test	2	11888010599	5944005299
rng	4000000	7589506214	1897
$init	97	95384	983
now	8	60820	7602
rng_vector	2	43483	21741
#	10	38421	3842
index_vector	2	20993	10496
push_back	2	12412	6206
round	2	9350	4675
strcat	2	7498	3749
-	4	3793	948

In the Profiler window you will see;

- Which function that is called
- How many calls that was made
- Total number of Ticks (CPU cycles)
- Number of Ticks (CPU cycles) per call

Sort by clicking on the appropriate column header.

Library file functions



```
ql_clp-64.qll                  QLL has dynamic base address
derivative_pricing_ql-64.qll   QLL did not load at its preferred addres
ahs2_api-64.qll                QLL has dynamic base address
```

Right clicking on a folder in the library tree opens the following functions:

Setting	Description
Remove	Removes the file from being included in startup of Quantlab. It does not delete the file from the windows directory.
Create New QL file	Create a library file in the windows directory and include it in the ini-file
Add existing QL file	Add an existing ql-file from the chosen directory
Add QLL	Add existing QLL file from chosen directory
Create folder	Create a new folder in the library tree (in the ini-file not on disc)
Create New Library Hive	A hive is a separate mini ini-file, that can contain one or several ql files or qll:s. The main library tree will point to include the hive file and not the individual code files.
Add Existing Library Hive	Include an already created hive file.
Move up/down	Move library files, qlls, or hives up or down in the ini file tree. Code will be compiled from top -> down so dependent code must be loaded after the files on which it depends.
Quick find (Ctrl + F)	Will search in the library only. Note that Ctrl + P will search library file names only.

Encryption

When right clicking on an ql-file, an option to encrypt the file will be given. It will encrypt the file both in the library tree and also the physical ql-file on disc. To recover a file with a forgotten password, please contact the Quantlab Support team.