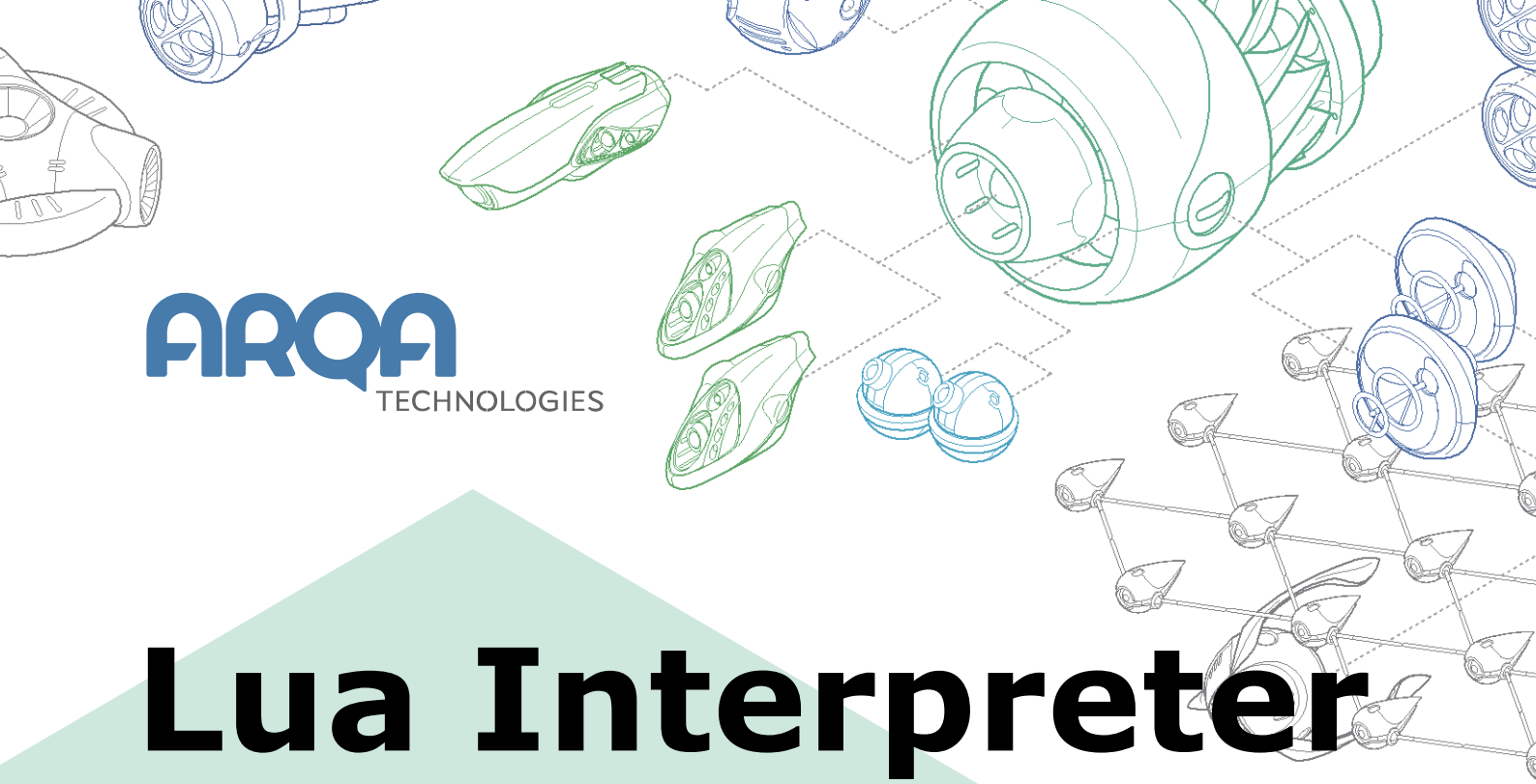
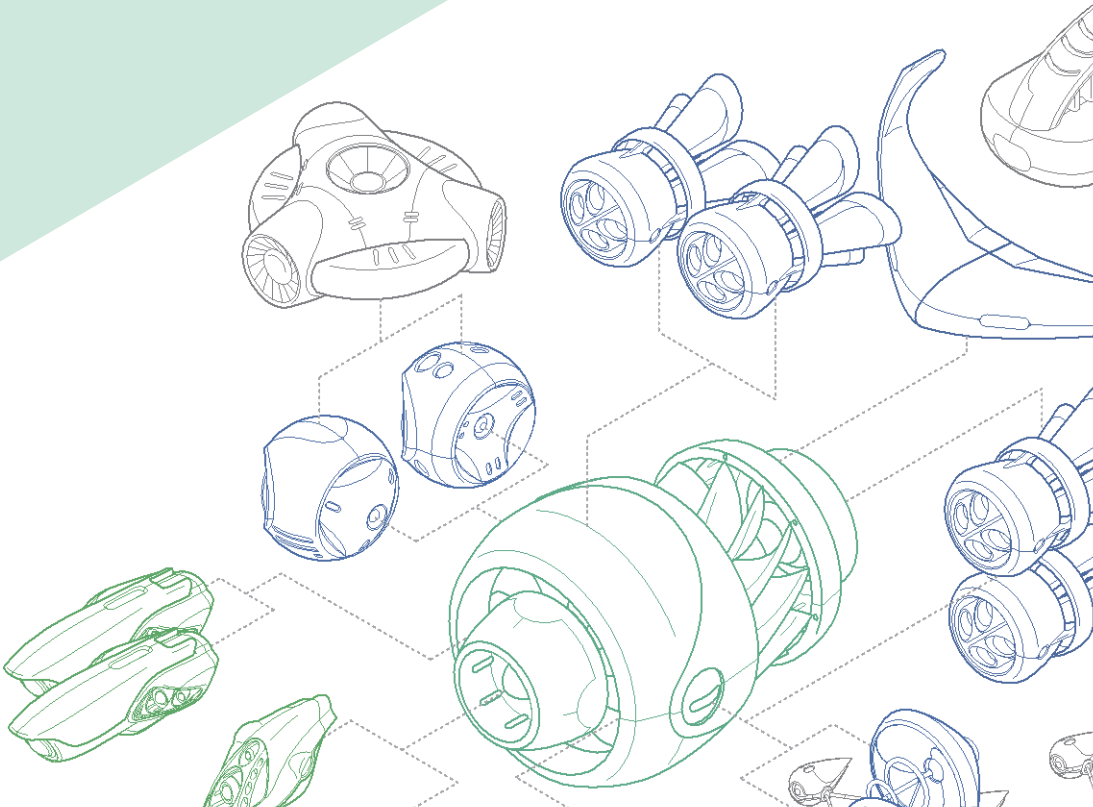


Lua Interpreter

User Manual

Version 2.2



Contents

Send your feedback and comments regarding this Instruction to: quiksupport@arqatech.com

1. About

Lua Interpreter (**QLua**) is a library that allows users to interact with a QUIK workstation using scripts written in Lua. For further details on Lua, visit www.lua.org.

1.1 Capabilities

QLua provides access to internal data of a QUIK workstation and can send client transactions from a Lua script to the QUIK sever. **QLua** enables (unlike the QPILE language) asynchronous processing in a script as it receives data, as well as the capability to constantly request new data from a client workstation. Asynchronous processing is based on the use of special callback functions defined in the script.

QLua can process the following events from QUIK workstations:

- reception of new data;
- connection to the QUIK server;
- disconnection from the QUIK server;
- a script stops;
- a QUIK workstation is closed.

All of these events might call an appropriate callback in the script.

QLua also provides the possibility of loading libraries written in other programming languages unto scripts.

> **Incorrect operation of third party libraries loaded into a script can cause errors in QUIK workstation operation.**

1.2 Getting Started

QLua is an optional component of a QUIK workstation.

QLua is provided in the qlua.dll file, which must be in the working directory of the QUIK workstation, for example, C:\Program Files\QUIK.

1.3 Working With the Program

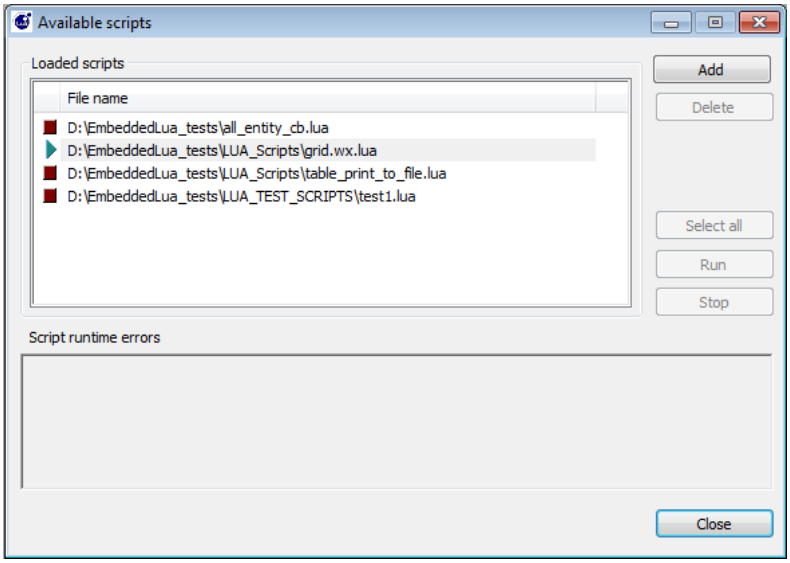
QLua functions are available from the program main menu. When the component is installed, new item **Tables / LUA / Available scripts** appears in the program menu.

1.3.1 Purpose

In the **Available scripts** form, you can add, remove, reload or launch scripts.

1.3.2 Window settings

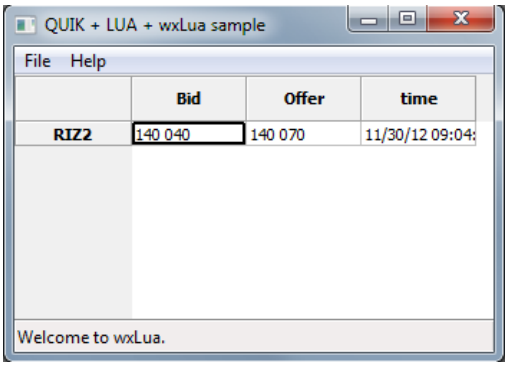
The window has the following control elements:



- Click **Add** to load a new script from a file.
- Click **Delete** to delete loaded scripts.
- Click **Select all** to select all available scripts.
- Click **Start** to run a loaded scripts.
- Click **Stop** to stop execution of a selected script.

The **Start** and **Stop** buttons are enabled depending on a script's current state.

Example of the result of a script execution:



Errors occurred during script execution are displayed in the error area.

2. Workstation Functions Accessible from Lua Scripts

2.1 Service functions

2.1.1 isConnected

This function is used to get the connection status between the workstation and the server.
The function returns '1' if a connection is established and '0' otherwise.

Function call syntax:

NUMBER isConnected()

2.1.2 getScriptPath

This function returns the path of the script from which it is called, without the final backslash ('\').
For example, C:\QuikFront\Scripts.

Function call syntax:

STRING getScriptPath()

Example:

```
path = getScriptPath()
```

2.1.3 getInfoParam

This function returns the values of the settings in the information window (menu **Connection / Information window**).

Function call syntax:

STRING getInfoParam (STRING param_name)

The param_name parameter can take the values presented in the following table.

Parameter value	Description
VERSION	Program version
TRADEDATE	Trading date
SERVERTIME	Server time
LASTRECORDTIME	Time of last record

Parameter value	Description
NUMRECORDS	Number of records
LASTRECORD	Last record
LATERECORD	Delayed record
CONNECTION	Connection

Parameter value	Description
IPADDRESS	Server IP address
IPPORT	Server port
IPCOMMENT	Connection description
SERVER	Server info
SESSIONID	Trade session identifier
USER	User
USERID	User ID
ORG	Organisation
MEMORY	Memory in use
LOCALTIME	Current time
CONNECTIONTIME	Connection time
MESSAGESSENT	Messages sent
ALLSENT	Bytes sent
BYTESSENT	Bytes sent (only with data)

Parameter value	Description
BYTESPERSECSENT	Sent per second
MESSAGESRECV	Messages received
BYTESRECV	Bytes received (only with data)
ALLRECV	Bytes received (total)
BYTESPERSECRECV	Received per second
AVGSENT	Average transfer rate
AVGRECV	Average receive rate
LASTPINGTIME	Last successfull ping time
LASTPINGDURATION	Data delay
AVGPINGDURATION	Average ping time
MAXPINGTIME	Maximum ping time
MAXPINGDURATION	Maximum ping duration

Example:

```
function main(  )
    params = {"VERSION", "TRADEDATE", "SERVERTIME",
                "LASTRECORDTIME", "NUMRECORDS", "LASTRECORD","LATERECORD",
                "CONNECTION", "IPADDRESS", "IPPORT", "IPCOMMENT",
                "SERVER", "SESSIONID", "USER", "USERID", "ORG", "MEMORY",
                "LOCALTIME", "CONNECTIONTIME", "MESSAGESSENT", "ALLSENT",
                "BYTESSENT", "BYTESPERSECSENT", "MESSAGESRECV", "BYTESRECV",
                "ALLRECV", "BYTESPERSECRECV", "AVGSENT", "AVGRECV",
                "LASTPINGTIME", "LASTPINGDURATION", "AVGPINGDURATION",
                "MAXPINGTIME", "MAXPINGDURATION"}
    file = io.open("res.txt", "w+t")
    for key,v in ipairs(params) do
        file:write(v .. " = " .. GetInfoParam(v) .. "\n")
    end
    file:close()
```

```
end
```

2.1.4 message

This function displays messages in a QUIK workstation. The function returns ‘nil’ in case of an exception or when input parameters are incorrect. Otherwise, the function returns ‘1’.

> **The maximum length of messages about errors arising when executing the script and strings translated to the message() function cannot exceed 900 symbols.**

Function call syntax:

NUMBER message(STRING message, NUMBER icon_type)

Parameters:

Parameter	Type	Description
message	STRING	String displayed in a QUIK workstation message window.
icon_type	NUMBER	The icon displayed in the message. Possible values and icons are: _ 1 (by default) –  ; _ 2 –  ; _ 3 –  Optional parameter

Example:

```
message("test message", 1)
message("test\nmessage", 2)
message("connection state is " .. tostring(isConnected()), 3)
```

2.1.5 sleep

This function pauses a script. The function returns ‘nil’ if input parameters are incorrect. If successful, the function returns the timeout specified by the ‘time’ parameter.

Function call syntax:

NUMBER sleep(NUMBER time)

Parameters:

Parameter	Type	Description
time	NUMBER	The length of time to sleep in milliseconds.

Example:

```
Sleep(1000)       a script is paused for one second
```

Using the sleep function in callback functions is not recommended.

2.1.6 getWorkingFolder

This function returns the path to **info.exe** that executes the script, without a backslash ('\') at the end. For example, c:\QuikFront.

Function call syntax:

STRING getWorkingFolder()

Example:

```
path = getWorkingFolder
```

2.1.7 PrintDbgStr

This function displays debug information.

Function call syntax:

PrintDbgStr(STRING s)

Example:

```
function main()
       PrintDbgStr("test1")
       PrintDbgStr("test2")
       PrintDbgStr("dbg from " ..getScriptPath())
End
```

2.2 Callback Functions

The functions described in this section are called when a QUIK terminal receives data from the server or to handle other external events.

> **IMPORTANT! Callback functions are processed in the main thread of the QUIK terminal. Therefore, users should optimize execution time of these functions.**

2.2.1 OnFirm

A QUIK terminal calls this function when it receives the description of a new firm from the server.

Function call syntax:

OnFirm(TABLE firm)

Parameters:

Parameter	Type	Description
firm	TABLE	Firms

2.2.2 OnAllTrade

A QUIK terminal calls this function when an anonymous trade is received.

Function call syntax:

OnAllTrade(TABLE alltrade)

Parameters:

Parameter	Type	Description
alltrade	TABLE	Anonymous Trades

2.2.3 OnTrade

This function is called by a QUIK terminal when a trade is received.

Function call syntax:

OnTrade(TABLE trade)

Parameters:

Parameter	Type	Description
trade	TABLE	Trade parameters table

2.2.4 OnOrder

This function is called by a QUIK terminal when it receives a new order or when the parameters of an existing order are changed.

Function call syntax:

OnOrder(TABLE order)

Parameters:

Parameter	Type	Description
order	TABLE	Orders

2.2.5 OnAccountBalance

This function is called by a QUIK terminal when it receives changes in a current position for an account.

Function call syntax:

OnAccountBalance(TABLE acc_bal)

Parameters:

Parameter	Type	Description
acc_bal	TABLE	Current Positions for Client Accounts

2.2.6 OnFuturesLimitChange

This function is called by a QUIK terminal when it receives changes in derivatives market limits.

Function call syntax:

OnFuturesLimitChange(TABLE fut_limit)

Parameters:

Parameter	Type	Description
fut_limit	TABLE	Futures Limits

2.2.7 OnFuturesLimitDelete

This function is called by a QUIK terminal when a derivatives market limit is deleted.

Function call syntax:

OnFuturesLimitDelete(TABLE lim_del)

Parameters:

Parameter	Type	Description
lim_del	TABLE	Deleting Futures Limit

2.2.8 OnFuturesClientHolding

This function is called by a QUIK terminal when a derivatives market position is changed.

Function call syntax:

OnFuturesClientHolding(TABLE fut_pos)

Parameters:

Parameter	Type	Description
fut_pos	TABLE	Client Account Positions (Futures)

2.2.9 OnMoneyLimit

This function is called by a QUIK terminal when it receives changes in a client's cash limit.

Function call syntax:

OnMoneyLimit(TABLE mlimit)

Parameters:

Parameter	Type	Description
mlimit	TABLE	Cash Limits

2.2.10 OnMoneyLimitDelete

This function is called by a QUIK terminal when a cash limit is removed.

Function call syntax:

OnMoneyLimitDelete(TABLE mlimit_del)

Parameters:

Parameter	Type	Description
mlimit_del	TABLE	Removing Cash Limit

2.2.11 OnDepoLimit

This function is called by a QUIK terminal when it receives changes in a limit for securities.

Function call syntax:

OnDepoLimit(TABLE dlimit)

Parameters:

Parameter	Type	Description
dlimit	TABLE	Limits for Securities

2.2.12 OnDepoLimitDelete

This function is called by a QUIK terminal when a client's limit for securities is removed.

Function call syntax:

OnDepoLimitDelete(TABLE dlimit_del)

Parameters:

Parameter	Type	Description

Parameter	Type	Description
dlimit_del	TABLE	Deleting Security Limit

2.2.13 OnAccountPosition

This function is called by a QUIK terminal when an account cash position changes.

Function call syntax:

OnAccountPosition(TABLE acc_pos)

Parameters:

Parameter	Type	Description
acc_pos	TABLE	Cash Positions

2.2.14 OnNegDeal

This function is called by a QUIK terminal when it receives an order for OTC securities.

Function call syntax:

OnNegDeal(TABLE neg_deals)

Parameters:

Parameter	Type	Description
neg_deals	TABLE	Negotiated Deal Orders

2.2.15 OnNegTrade

This function is called by a QUIK terminal when it receives a trade for execution.

Function call syntax:

OnNegTrade(TABLE neg_trade)

Parameters:

Parameter	Type	Description

Parameter	Type	Description
neg_trade	TABLE	Trades for Execution

2.2.16 OnStopOrder

This function is called by a QUIK terminal when it receives a new stop order or when the parameters of an existing stop order are changed.

Function call syntax:

OnStopOrder(TABLE stop_order)

Parameters:

Parameter	Type	Description
stop_order	TABLE	Current Positions for Instruments

2.2.17 OnTransReply

This function is called by a QUIK terminal when it receives a response for a user transaction.

Function call syntax:

OnTransReply(TABLE trans_reply)

Parameters:

Parameter	Type	Description
trans_reply	TABLE	Transactions

2.2.18 OnParam

This function is called by a QUIK terminal when current parameters change.

OnParam (STRING class_code, STRING sec_code)

Parameters:

Parameter	Type	Description
class_code	STRING	Class code

Parameter	Type	Description
sec_code	STRING	Security code

> **In this function, the user can call the getParamEx function to obtain the value of the desired parameter.**

Example:

```
function OnParam( class, sec )
if class =="SPBFUT" and sec == "RIZ2" then
tbid = getParamEx(class, sec, "bid")
if tbid.param_value >=130000 then
'
'
End
```

2.2.19 OnQuote

This function is called by a QUIK terminal when it receives changes in Level II quotes table.

OnQuote(STRING class_code, STRING sec_code)

> **To obtain the desired Level II Quotes table, call getQuoteLevel2().**

Example:

```
function OnQuote(class, sec )

if class =="SPBFUT" and sec == "RIZ2" then
ql2 = getQuoteLevel2(class, sec)
end
```

2.2.20 OnDisconnected

This function is called by a QUIK terminal when it is disconnected from the QUIK server.

OnDisconnected()

2.2.21 OnConnected

This function is called by a QUIK terminal when upon establishing a connection to the QUIK server.

OnConnected()

2.2.22 OnCleanUp

This function is called by a QUIK terminal when the session changes or when qlua.dll is unloaded.

OnCleanUp()

2.2.23 OnClose

This function is called before closing a QUIK terminal.

OnClose()

2.2.24 OnStop

This function is called by a QUIK terminal when the script is stopped from the control dialogue and when the QUIK terminal is closed.

[NUMBER time_out] OnStop(NUMBER signal)

The function returns the quantity of milliseconds that is available for a script to finish its operation. If the function doesn't return a number, timeout for script termination will be equal to 5 seconds.

> **After the time interval given to a script to terminate expires, the main() function is terminated. This might cause a loss of system resources.**

Example:

```
function OnStop(s)
       stopped = true
       return 3000 – timeout of 3 seconds is set
end
function OnStop(s)
       stopped = true
       return '3000' – return value is not a number, timeout is 5 seconds
end
```

2.2.25 OnInit

This function is called by a QUIK terminal before calling the **main()** function. It receives the full path to the script being launched as an input parameter.

OnInit(STRING script_path)

> **In this function, the user can initialize all required variables and libraries before launching the main() thread.**

2.2.26 main

Implements the main thread of execution of a script. The QUIK terminal creates a separate thread for this function. The script is running as long as the main() function is running. After the function exits, the script's status changes to 'stopped'. When the script is stopped, the event callback functions defined in the script are not called.

Notifications are only passed to the script, if parameters for this security are requested from the server (as a part of a smart request or manually though the **Connection / Available securities** dialogue) or if the corresponding Level II quotes table is opened.

3. Functions for Interactions between Lua Scripts and QUIK Workstation

3.1 Functions for Accessing Rows in QUIK Tables

The functions from this group are used to access data contained in QUIK tables.

3.1.1 getItem

This function returns a Lua table with the data on the row with number 'Index' from the table named 'TableName'.

getItem (STRING TableName, DOUBLE Index)

Index of elements in a table begins from 0.

3.1.2 getOrderByNumber

This function returns a Lua table with the description of the parameters of the

Orders and an order's index in the terminal's repository.

getOrderByNumber(STRING class_code, NUMBER order_id)

If there is no order with the specified number, the function returns nil.

3.1.3 getNumberOf

This function returns the number of rows in the table named 'TableName'.

getNumberOf(STRING TableName)

3.1.4 SearchItems

This function allows for quick selection of elements from the terminal's repository and returns a table with indices of elements that match the search criteria.

TABLE SearchItems(STRING table_name, NUMBER start_index, NUMBER end_index, FUNCTION fn [, STRING params])

Parameters:

- **table_name** – the table name to search in;
- **start_index** – the index of the first element to be searched;
- **end_index** – the index of the last element to be searched;
- **fn** – a callback function that returns true or false. The params parameter, if specified, defines the input parameters of the function:
- **params** defines the fields of an element of the **table_name** table that will be passed to the **fn** function. The fields should be separated by commas; spaces are ignored. If **params** is not set, the entire element will be passed to **fn**. An optional parameter.
  For the examples of use of **params**, refer to Appendix 4.

3.1.5 The tables used in the functions 'getItem', 'getNumberOf', 'getOrderByNumber' and 'SearchItems'

TableName	Table
firms	Firms
classes	Classes
securities	Instruments
trade_accounts	Trading accounts
client_codes	Client codes*
all_trades	Anonymous Trades
account_positions	Cash Positions
orders	Orders
futures_client_holding	Client Account Positions (Futures)
futures_client_limits	Futures Limits
money_limits	Cash Limits

TableName	Table
depo_limits	Limits for Securities
trades	Trades
stop_orders	Stop Orders
neg_deals	Negotiated Deal Orders
neg_trades	Trades for Execution
neg_deal_reports	Reports on Trades for Execution
firm_holding	Current Positions for Instruments
account_balance	Current Positions for Client Accounts
ccp_positions	Cash Commitments and Claims
ccp_holdings	Asset Commitments and Claims

* – getNumberOf("client_codes") returns the number of client codes available in the terminal; getItem("client_codes", i) returns a string that contains a client code with index i, where i can have values from 0 to getNumberOf("client_codes") -1

Example:

```
function main()
n = getNumberOf(“orders”)
order={}
message(“total “.. tostring(n) .. “ of all orders”, 1)
for i=0,n-1 do
       order = getItem(“orders”, i)
       message(“order: num=” .. tostring(order[“order_num”]) .. “ qty=” ..
tostring(order[“qty”]) .. “ value=” .. tostring(order[“value”]), 1)
end
```

```
end
```

This script displays information about all orders.

3.2 Functions for Accessing Available Parameters Lists

3.2.1 getClassesList

Returns a list of codes of classes that were transferred from the server during a connection session. Class codes in this list are separated by a comma, ‘,’. The character ‘,’ is always added to the end of a received string.

getClassesList ()

Example:

```
list = getClassesList ()
```

In this example, the list variable will contain the following string:

```
FUTEVN,OPTEVN,RTSSTEVN,OPTEXP,USDRUB,REPORTSST,
```

3.2.2 getClassInfo

Returns information about a class.

getClassInfo (STRING)

Returns a Lua table with a class description:

Parameter	Type	Description
firmid	NUMBER	Firm code
name	STRING	Class name
code	STRING	Class code
npars	NUMBER	Number of parameters in the class
nsecs	NUMBER	Number of securities in the class

3.2.3 getClassSecurities

This function returns a list of security codes for a specified list of class codes. Security codes in the list are separated by a comma, ‘,’. The character ‘,’ is always added to the end of a received string.

getClassSecurities (STRING)

Example:

```
sec_list = getClassSecurities(""SPBFU"")
```

In this example, the sec_list variable will contain the following string:

```
RSH3,VBZ2,O4Z2,O2Z2,SiM3,SiH3,SiF3,RIH3,RIM3,LKH3,LKZ2,GDZ2,GMZ2,GZH3,GZZ2,EuZ2,EDZ2,
SiZ2,RIZ2,
```

3.3 Function for Obtaining Cash Data

3.3.1 getMoney

Returns cash limit information.

getMoney (STRING client_code, STRING firmid, **STRING** tag, STRING currcode)

The function returns a Lua table with the following parameters:

Parameter	Type	Description
money_open_limit	NUMBER	Open cash limit
money_limit_locked_nonmarginal_value	NUMBER	Value of non-margin securities in buy orders
money_limit_locked	NUMBER	Cash volume locked in buy orders
money_open_balance	NUMBER	Open cash balance
money_current_limit	NUMBER	Current cash limit
money_current_balance	NUMBER	Current cash balance
money_limit_available	NUMBER	Available cash value

3.3.2 getMoneyEx

Returns the information on cash limits of a specified type.

TABLE getMoneyEx(STRING firm_id, STRING client_code, STRNIG tag, STRING curr_code, NUMBER limit_kind)

The function returns a Lua table with the Cash limits table parameters when processing the cash limit change.

If an error occurs, the function returns ‘nil’.

3.4 Function for Obtaining Security Limit Information

3.4.1 getDepo

Returns securitiy limit information.

getDepo (STRING client_code, STRING firmid, STRING sec_code, STRING account)

The function returns a Lua table with the following parameters:

Parameter	Type	Description
depo_limit_locked_buy_value	NUMBER	Value of instruments locked for buying
depo_current_balance	NUMBER	Current securities balance
depo_limit_locked_buy	NUMBER	Number of lots of securities locked for buying
depo_limit_locked	NUMBER	Number of locked lots of securities
depo_limit_available	NUMBER	Quantity of available securities
depo_current_limit	NUMBER	Current limit for securities
depo_open_balance	NUMBER	Opening balance for securities
depo_open_limit	NUMBER	Open limit for securities

3.4.2 getDepoEx

Returns the information on security limits of a specified type.

TABLE getDepoEx(STRING firm_id, STRING client_code, STRNIG sec_code, STRING acc_id, NUMBER limit_kind)

The function returns a Lua table with the Securities limits table parameters when processing the security limit change.

If an error occurs, the function returns ‘nil’.

3.5 Function for Obtaining Futures Limits Information

3.5.1 getFuturesLimit

Returns the information on Futures Limits.

TABLE getFuturesLimit(STRING firm_id, STRING acc_id, NUMBER limit_type)

The function returns a Lua table with the parameters of a futures limit when processing the security limit change.

If an error occurs, the function returns 'nil'.

3.6 Function for Obtaining Futures Positions Information

3.6.1 getFuturesHolding

Returns the information on Futures Positions.

TABLE getFuturesHolding(STRING form_id, STRING acc_id, STRING sec_code, NUMBER pos_type)

The function returns a Lua table with the parameters of the Table of futures positions when processing the security limit change.

If an error occurs, the function returns 'nil'.

3.7 Function for Obtaining Instrument Information

3.7.1 getSecurityInfo

This function returns information on a security.

getSecurityInfo (STRING class_code, STRING sec_code)

The function returns a Lua table with the following parameters:

Parameter	Type	Description
Sec_code	STRING	Security code
name	STRING	Instrument name
short_name	STRING	Abbreviated name
class_code	STRING	Class code

Parameter	Type	Description
class_name	STRING	Class name
face_value	STRING	Face-value
face_unit	STRING	Nominal currency code
scale	NUMBER	Number of significant digits after decimal point
mat_date	NUMBER	Expiration date
lot_size	NUMBER	Lot size

3.8 Function for Obtaininga trading session date

3.8.1 getTradeDate

Returns the date of the current trading session.

getTradeDate ()

The function returns a Lua table with the following parameters:

Parameter	Type	Description
date	STRING	Trading date in the form of <DD.MM.YYYY>
year	NUMBER	Year
month	NUMBER	Month
day	NUMBER	Day

3.9 Function for Obtaining Level II Quotes Table for Specified Class and Security

3.9.1 getQuoteLevel2

This function returns Level II quotes table for the specified class and security.

getQuoteLevel2 (STRING class_code, STRING sec_code)

The function returns a Lua table with the following parameters:

Parameter	Type	Description

Parameter	Type	Description
bid_count	STRING	Number of bids
offer_count	STRING	Number of offers
bid	TABLE	Bids (buying)
offer	TABLE	Offers (selling)

The ‘bid’ and ‘offer’ tables have the following structure:

Parameter	Type	Description
price	STRING	Buy/Sell price
quantity	STRING	Quantity in lots

3.10 Functions for Working with Charts

3.10.1 getLinesCount

This function returns the number of lines on a chart (indicator) for a specified identifier.

NUMBER getLinesCount (STRING tag)

Returns the number of lines on a chart.

3.10.2 getNumCandles

This function returns the number of candlesticks for a specified identifier.

NUMBER getNumCandles (STRING tag)

Returns the number of candlesticks for a specified identifier.

3.10.3 getCandlesByIndex

This function returns information about candlesticks for an identifier (this plug-in does not request data for chart plotting; therefore, to get access to the chart, it must be open):

Function call syntax:

TABLE t, NUMBER n, STRING l getCandlesByIndex (STRING tag, NUMBER line, NUMBER first_candle, NUMBER count)

Parameters:

- **tag** - a string identifier of a chart or an indicator;
- **line** - a line number in a chart or an indicator. The number of the first line is 0;
- **first_candle** - the index of the first candlestick. The index of the first (leftmost) candlestick is 0;
- **count** - the number of candlesticks to be returned.

Returns:

- **t** – the table that contains requested candlesticks;
- **n** – the number of candlesticks in **t**;
- **l** – the chart legend (caption).

3.10.4 CreateDataSource

The function is intended to create a Lua table and provides working with candlesticks received from QUIK server and reacting to its changes.

Function call syntax:

> TABLE data_source, STRING error_desc CreateDataSource (STRING class_code, STRING sec_code, NUMBER interval [, STRING param])

Parameters:

- **class_code** – code of a class;
- **sec_code** – code of a security;
- **interval** – interval of the required chart;
- **param** – optional parameter. If not specified, the data is required on the basis of Time and sales table; if the parameter is specified then data is required on this parameter.

The function returns **data_source** table if executed successfully. If incorrect class code or parameter is specified than the function returns “nil”. At that **data_sourc** contains description of the error.

List of constants for transmission to parameter **interval**:

Parameter	Value of interval
INTERVAL_TICK	Tick data
INTERVAL_M1	1 minute
INTERVAL_M2	2 minutes
INTERVAL_M3	3 minutes
INTERVAL_M4	4 minutes
INTERVAL_M5	5 minutes

Parameter	Value of interval
INTERVAL_M6	6 minutes
INTERVAL_M10	10 minutes
INTERVAL_M15	15 minutes
INTERVAL_M20	20 minutes
INTERVAL_M30	30 minutes
INTERVAL_H1	1 hour

Parameter	Value of interval
INTERVAL_H2	2 hours
INTERVAL_H4	4 hours
INTERVAL_D1	1 day

Parameter	Value of interval
INTERVAL_W1	1 week
INTERVAL_MN1	1 month

Function **CreateDataSource** returns Lua table with parameters:

Parameter	Type	Description
SetUpdateCallback	function	Allows a user to set callback function for processing changed candlesticks
O	function	Get value Open for the selected candlestick
H	function	Get value High for the selected candlestick
L	function	Get value Low for the selected candlestick
C	function	Get value Close for the selected candlestick
V	function	Get value Volume for the selected candlestick
T	function	Get value Time for the selected candlestick
Size	function	Returns the current size (number of candlesticks in the data source)
Close	function	Deletes the data source, unsubscribes from the received data

For example:

```
ds1 = CreateDataSource("SPBFUT", "RIU3", INTERVAL_M1, "last")
ds2 = CreateDataSource("SPBFUT", "RIU3", INTERVAL_M1)
ds3 = CreateDataSource("SPBFUT", "RIU3", INTERVAL_M1, "bid")
```

Detailed description of all functions is given below.

Function SetUpdateCallback

Function call syntax:

BOOLEAN res SetUpdateCallback (FUNCTION callback_function)

As a parameter takes the callback function:

Format of callback function:

function call_back(NUMBER index)

Parameters:

- **index** – number of changes candkestick. Candlestick indexes start from 1.

The function returns “true” is executed successfully, otherwise “false”.

Example of receiving time from a candlestick:

```
function cb( index )
local t = ds:T(index)
local _str = string.format"”#%d of %d\t%.4f\t%.4f\t%.4f\t%.4f\t%.4f %02d.%02d.%04d
%02d:%02d:%02d.%04”",
index, ds:Size(),ds:O(index), ds:H(index), ds:L(index),
ds:C(index), ds:V(index),
t.day, t.month, t.year, t.hour, t.min, t.sec, t.ms)
Log(file, _str)
end
ds: SetUpdateCallback (cb)
```

Functions O, H, L, C, V, T

Functions as a parameter takes a candlestick index and return a corresponding value. Time of candlestick returns within milliseconds in the form of a table with fields:

{year, month, day, week_day, hour, min, sec, ms, count }

Where

- **count** – a quantity of tick intervals per second. Must be a value between 1 and 10000 inclusively.

For example:

```
Open = ds:O(1)
High = ds:H(1)
Low = ds:L(1)
Close = ds:C(1)
Volume = ds:V(1)
week_day = ds:T(1).week_day
count = ds:T(1).count
```

Function Size

Function returns a current number of candlesticks in the data source.

Function call syntax:

NUMBER Size()

For example:

```
ds:Size()
```

Function Close
Function closes the data source and the terminal stops receiving the data from server.

Function call syntax:

BOOLEAN Close()

For example:

```
ds:Close()
```

The function returns “true” if executed successfully.

Function SetEmptyCallback
Function allows to receive the data from the server.

Function call syntax:

BOOLEAN SetEmptyCallback()

The function returns “true” if executed successfully, otherwise “false”.

For example:

```
ds:SetEmptyCallback()
```

3.11 Functions for Working with Orders

These functions are intended for creating orders and sending them into the trading system.

3.11.1 sendTransaction

This function sends a transaction to the QUIK server. If an error occurs during processing a transaction in the QUIK terminal, the function returns the description of the error. Otherwise, the transaction will be sent to the server.

The result of the transaction is returned in the **OnTransReply** callback function.

The input parameter for this function is a table in which the names and values of the fields correspond to the parameters used in the .tri file (see QUIK User Manual, Section 6: Working with other programs, paragraph 6.11.3).

> **IMPORTANT! For correct data processing, numeric values (price, quantity, transaction identifier, etc) must be passed as strings.**

3.11.2 CalcBuySell

The function is intended to calculate the maximum possible number of lots in order.

Function call syntax:

> NUMBER qty, NUMBER comission CalcBuySell(STRING class_code, STRING sec_code, STRING client_code, STRING account, NUMBER price, BOOLEAN is_buy, BOOLEAN is_market)

Parameters:

- **class_code** – class code;
- **sec_code** – security code;
- **client_code** – client code;
- **account** – depo account;
- **price** – price;
- **is_buy** – attribute of a buy order (“true” – to buy, otherwise – to sell);
- **is_market** – attribute of a market order (“true” – market order, otherwise limit order). Optional parameter, value by default: “false”.

Example:

```
function main()
       local bs = CalcBuySell
       assert(bs, "No funstion!!")
       while not stopped do
       qty, comiss = bs("BQUOTE", "AFLT", "Q3", "S01-00000F00", 10, true, false)
       message("qty = " .. qty .. ", COM = " .. comiss, 2)
       sleep(1000)
       end
end
```

3.12 Function for Obtaining Values from Quotes Table

3.12.1 getParamEx

This function returns all exchange parameters from a Quotes Table. This function can be used to obtain any value from a Quotes Table for specified class and security codes.

Function call syntax:

getParamEx (STRING class_code, STRING sec_code, STRING param_name)

The function returns a Lua table with the following parameters:

Parameter	Type	Description
param_type	STRING	The data type of a parameter from the Quotes Table. Possible values include: _ 1 – DOUBLE; _ 2 – LONG; _ 3 – CHAR; _ 4 – enumerable type; _ 5 – time; _ 6 – date
param_value	STRING	Parameter value. For param_type = 3, value of the parameter is equal to 0, in other cases numerical data type. For enumerable types, the value equals sequence number in the enumeration
param_image	STRING	The string value of the parameter as it is presented in the tables. The string representation takes into account numeric and decimal separators. For enumerable types, their corresponding string values are returned

3.13 Functions for Obtaining Parameters of Client Portfolio Table

3.13.1 getPortfolioInfo

This function is used for obtaining parameter values from the ‘Client portfolio’ table for the specified firm identifier ‘firm_id’ and client code ‘client_code’.

Function call syntax:

TABLE getPortfolioInfo (STRING firm_id, STRING client_code)

The function returns a Lua table with the following parameters:

№	Parameter	Type	Description	
1	is_leverage	STRING	Attribute of monitoring positions type. Possible values include: _ MLim: scheme of monitoring a position “by leverage” is used, the leverage is calculated based on the Incoming limit value; _ MP: scheme of monitoring a position “by leverage” is used when the leverage is expressly stated; _ Mpos: positions monitoring scheme “open position limit” is used; _ <blank>: positions monitoring scheme “by limit” is used	Client type
2	in_assets	STRING	Estimation of a client's assets before beginning of trading	Open assets
3	leverage	STRING	Leverage If it is not specified explicitly, the value equals the ratio of Opening limit to Opening assets	Leverage
4	open_limit	STRING	Estimation of the maximum value of borrowed assets before beginning of trading	Open limit
5	val_short	STRING	Estimated value of short positions This value is always negative.	ValShort
6	val_long	STRING	Estimated value of long positions	ValLong
7	val_long_margin	STRING	Estimated value of long positions for marginal securities used as collateral	ValLongMargin
8	val_long_asset	STRING	Estimated value of long positions for non-marginal securities included as collateral	ValLongAsset
9	assets	STRING	Estimated value of a client's assets for current positions and prices	Cur. assets
10	cur_leverage	STRING	Current leverage	Cur. leverage
11	margin	STRING	Margin ratio as a percentage	Margin ratio
12	lim_all	STRING	Current estimated value of maximum quantity of borrowed assets	Cur. Limit
13	av_lim_all	STRING	Estimated value of borrowed assets available for further opening of positions	AvLimAll
14	locked_buy	STRING	Estimated value of assets in buy orders	Lock. buy

№	Parameter	Type	Description	
15	locked_buy_margin	STRING	Estimated value of assets in buy orders for marginal securities used as collateral	Lock. margin. buy
16	locked_buy_asset	STRING	Value of assets in buy orders for non-marginal securities accepted as collateral	Lock. coll. buy
17	locked_sell	STRING	Estimated value of assets in sell orders for marginal securities	Lock. sell
18	locked_value_coef	STRING	Value of assets in buy orders for non-marginal securities	Lock. non-margin. buy
19	in_all_assets	STRING	Value of all client positions adjusted to the closing prices from the preceding trading session including positions for non-marginal securities	InAllAssets
20	all_assets	STRING	Current estimated value for all positions of a client	AllAssets
21	profit_loss	STRING	Absolute value of change in price of all positions of a client	ProfitLoss
22	rate_change	STRING	Relative value of change in price of all positions of a client	RateChange
23	lim_buy	STRING	Estimated cash volume available for buying marginal securities	LimBuy
24	lim_sell	STRING	Estimated value of marginal securities available for selling	LimSell
25	lim_non_margin	STRING	Estimated cash volume available for buying non-marginal securities	LimNonMargin
26	lim_buy_asset	STRING	Estimated cash assets available for purchasing securities used as collateral	LimBuyAsset
27	val_short_net	STRING	Estimated value of short positions The discount coefficient is not used in calculations*	Short (net)
28	val_long_net	STRING	Estimated value of long positions. The discount coefficient is not used in calculations*	Long (net)
29	total_money_bal	STRING	Sum of funds balance for all limits, excluding funds locked for commitments, in the selected settlement currency	Total cash balance
30	total_locked_money	STRING	Sum of locked funds from all cash limits of a client calculated in the settlement currency using cross rates from the server	Total locked cash

№	Parameter	Type	Description	
31	haircuts	STRING	Total discounts on the value of long (only for collateral securities) and short security positions, discounts of the correlation between instruments, as well as discounts on owed currencies not covered by security collateral in the same currencies	Haircuts
32	assets_without_hc	STRING	Total value of cash balances, prices of long positions for the securities held as collateral, and prices of short positions, without taking into account discount factors, netting of securities value within a unified position or correlation between instruments	Assets w/o HC
33	status_coef	STRING	Ratio between the sum of discounts and current assets without discounts	Status coef.
34	varmargin	STRING	Current variation margin for a client's positions for all instruments	Variat. margin
35	go_for_positions	STRING	Cash paid to cover all open positions on the derivatives market	Curr. clear pos.
36	go_for_orders	STRING	Estimated value of assets in orders on the derivatives market	Curr. clear ord.
37	rate_futures	STRING	Ratio between disposal value of a portfolio and collateral on the derivatives market	Assets/Curr. clear pos.
38	is_qual_client	STRING	This attribute shows that the client is a 'qualified' client who can borrow assets with leverage of 1:3. Possible values include: 'HighRisk' means qualified, <blank> means not qualified	HighRisk
39	is_futures	STRING	Client account on FORTS, if there is a unified position; otherwise this field will be empty	Fut. trade account
40	curr_TAG	STRING	Actual current settlement parameters for this row in the format <Currency>-<Trading session identifier>. Example: 'SUR-EQTV'	Calc. parameters

> **(*) For more details on discount factors, see Section 7 of Administrator's guide 'Dealer library configuration'.**

3.13.2 getPortfolioInfoEx

This function is used for obtaining parameters from the 'Client portfolio' table for the specified firm identifier 'firm_id', client code 'client_code' and limit type 'limit_kind'.

Function call syntax:

TABLE getPortfolioInfoEx (STRING firm_id, STRING client_code, DOUBLE limit_kind)

Returns a Lua table with the parameters from the 'Client portfolio' table. For the description of parameters, see sub-section 3.13.1.

The function also returns the following parameters:

№	Parameter	Type	Description	
1	init_margin	STRING	Value of the initial margin. The parameter is filled for MD clients	Init.margin
2	min_margin	STRING	Value of the minimum margin. The parameter is filled for MD clients	Min.margin
3	corrected_margin	STRING	Value of the corrected margin. The parameter is filled for MD clients	Corr.margin
4	client_type	STRING	Client type	Client type
5	portfolio_value	STRING	Portfolio value. For MD clients the value for rows with the maximum limit kind returns	Portfolio value

3.14 Functions for Obtaining Parameters from 'Buy/Sell' Table

3.14.1 getBuySellInfo

This function is used for obtaining parameters from the 'Buy/Sell' table.

Function call syntax:

TABLE getBuySellInfo (STRING firm_id, STRING client_code, STRING class_code, STRING sec_code, NUMBER price)

This functions returns a Lua table that contains parameters from the 'Buy/Sell' QUIK table which indicate the possibility of buying or selling the specified security 'sec_code' of class 'class_code' by the client 'client_code' of the firm 'firmid' at the price 'price'. If the price is '0', the best bid/offer values are used.

Parameters:

№	Parameter	Type	Description
1	is_margin_sec	STRING	This attribute specifies if the instrument is marginal. Possible values include: _ 0 – not marginal; _ 1 – marginal
2	is_asset_sec	STRING	This attribute shows if the instrument belongs to the list of securities used as collateral. Possible values include: _ 0 – is not used; _ 1 – is used
3	balance	STRING	Current instrument position, in lots
4	can_buy	STRING	Estimated number of lots available for buying at the specified price *
5	can_sell	STRING	Estimated number of lots available for selling at the specified price *
6	position_valuation	STRING	Valuation of an instrument position according to bid/offer prices
7	value	STRING	Evaluated position value according to last trade price
8	open_value	STRING	Evaluated client's position value calculated according to the closing price of the previous trading session
9	lim_long	STRING	Maximum position size for this instrument that can be accepted as collateral for long positions
10	long_coef	STRING	Discount factor for long positions for this instrument
11	lim_short	STRING	The maximum size of a short position for this instrument
12	short_coef	STRING	Discount factor for short positions for this instrument
13	value_coef	STRING	Evaluated position value according to the last trade price taking into account the discount factor
14	open_value_coef	STRING	Evaluated value of client's position calculated according to the closing price of the previous trading session taking into account discount factors
15	share	STRING	Percentage of the position value for this instrument to the total value of the client's assets calculated according current prices
16	short_wa_price	STRING	Weighted average price of short positions for instruments
17	long_wa_price	STRING	Weighted average price of long positions for instruments
18	profit_loss	STRING	Difference between the weighted average price of buying securities and their market price
19	spread_hc	STRING	Coefficient of correlation between instruments

№	Parameter	Type	Description
20	can_buy_own	STRING	Maximum number of securities in a buy order for this instrument in this class using client's own assets at the best offer price
21	can_sell_own	STRING	Maximum number of securities in a sell order for this instrument in this class from client's own assets at the best bid price

(*) Depending on QUIK server settings, this value can be expressed in lots or in units Contact your broker for information about the unit of measurement.

3.14.2 getBuySellInfoEx

This function is used for obtaining parameters from the 'Buy/Sell' table.

Function call syntax:

TABLE getBuySellInfoEx (STRING firm_id, STRING client_code, STRING class_code, STRING sec_code, NUMBER price)

This function returns a Lua table that contains parameters from the 'Buy/Sell' QUIK table which indicate the possibility of buying or selling the specified security 'sec_code' of class 'class_code' by the client 'client_code' of the firm 'firmid' at the price 'price'. If the price is '0', the best bid/offer values are used.

For the description of returned parameters, see sub-section 3.14.1.

The function also returns the following parameters:

№	Parameter	Type	Description
1	limit_kind	NUMBER	Limit type Possible values include: _ 0 – T0; _ 1 – T1; _ 2 – T2
2	d_long	STRING	Effective initial discount for a long position. The parameter is filled for MD clients
3	d_min_long	STRING	Effective minimum discount for a long position. The parameter is filled for MD clients
4	d_short	STRING	Effective initial discount for a short position. The parameter is filled for MD clients
5	d_min_short	STRING	Effective minimum discount for a short position. The parameter is filled for MD clients

№	Parameter	Type	Description
6	client_type	STRING	Client type
7	is_long_allowed	STRING	Attribute of a security allowed to be bought at borrowed funds. Valid values: _ 1 – allowed; _ 0 – not allowed. The parameter is filled for MD clients
8	is_short_allowed	STRING	Attribute of a security allowed to be sold at borrowed funds. Valid values: _ 1 – allowed; _ 0 – not allowed. The parameter is filled for MD clients

3.15 Functions for Working with QUIK Workstation Tables

QUIK workstation tables created with Lua scripts provide the following features:

- drag-and-drop mode;
- user filters;
- conditional formatting;
- tabs;
- searching within a table;
- previewing and printing.

Below is the list of features that are not supported for the tables created in Lua:

- tables cannot be saved into a configuration file;
- editing dialogue window is not available;
- no table context menu (except for the **Move to tab** item);
- tables cannot be copied;
- a default table window title cannot be set;
- data from tables cannot be exported;
- tables do not support hotkeys.

3.15.1 AddColumn

This function adds columns to a table with the ‘t_id’ identifier.

Function call syntax:

NUMBER AddColumn (NUMBER t_id, NUMBER iCode, STRING name, BOOLEAN is_default, NUMBER par_type, NUMBER width)

Parameters:

- **iCode** – code of the parameter displayed in a column;
- **name** – the name of a column;
- **is_default** – this parameter is not used;
- **par_type** – data type in a column; the value can be one of the following constants:
  - QTABLE_INT_TYPE – integer;
  - QTABLE_DOUBLE_TYPE – double;
  - QTABLE_INT64_TYPE – 64-bit integer;
  - QTABLE_CACHED_STRING_TYPE – cached string;
  - QTABLE_TIME_TYPE – time;
  - QTABLE_DATE_TYPE – date;
  - QTABLE_STRING_TYPE – string.
- **width** – width in nominal units.

This function returns '1' if the column is added to a table and '0' otherwise.

3.15.2 AllocTable

This function creates a structure that defines a table.

Function call syntax:

NUMBER AllocTable()

This function returns an integer table identifier that can be used to work with this table.

3.15.3 Clear

This function deletes the content of the table with the id 't_id'.

Function call syntax:

NUMBER Clear (NUMBER t_id)

The function returns 'false' in case of failure.

3.15.4 CreateWindow

This function creates a window for the table with the id ‘t_id’.

Function call syntax:

NUMBER CreateWindow(NUMBER t_id)

This function returns ‘1’ if a window is created successfully, and ‘0’ otherwise.

3.15.5 DeleteRow

This function deletes the row with the specified key from the table with the id ‘t_id’.

Function call syntax:

BOOLEAN DeleteRow(NUMBER t_id, NUMBER key)

The function returns ‘false’ in case of failure.

3.15.6 DestroyTable

This function closes the window of the table with the specified id.

Function call syntax:

NUMBER DestroyTable(NUMBER t_id)

All display data is deleted when a window is closed.

This function returns ‘true’ if executed successfully and ‘false’ otherwise.

3.15.7 InsertRow

This function inserts a row with the specified key to the table with the id ‘t_id’.

Function call syntax:

NUMBER InsertRow(NUMBER t_id, NUMBER key)

> **To add data to a new table, first execute this function with the ‘key’ parameter equal to ‘-1’. The row will be added to the end of the table.**

The function returns the index of the added row if executed successfully and -1 otherwise.

If the key is larger than the current number of rows, the new row will be added to the end of the table.

3.15.8 IsWindowClosed

This function returns ‘true’ if the window with the table ‘t_id’ is closed.

> **The IsWindowClosed function will return "false" if it is called inside the callback function which is set using SetTableNotificationCallback().**

Function call syntax:

```
BOOLEAN IsWindowClosed(NUMBER t_id)
```

The function returns 'nil' in case of failure.

The window can be opened again with the **CreateWindow** function (for further details, see sub-section 3.15.4).

3.15.9 GetCell

This function returns a table with the data from a cell with the specified row key and column code in the table 't_id'.

Function call syntax:

```
TABLE GetCell(NUMBER t_id, NUMBER key, NUMBER code)
```

Table parameters:

- **image** – string representation of the cell value,
- **value** – the numeric cell value.

If input parameters were set incorrectly, the function returns 'nil'.

3.15.10 GetTableSize

This function returns the number of rows and columns in the table with the id 't_id'.

Function call syntax:

```
NUMBER rows, NUMBER col GetTableSize (NUMBER t_id)
```

User filters applied to the table do not affect the number of rows returned. Headers and the first fixed column are not included in the returned values.

The function returns 'nil' in case of failure.

3.15.11 GetWindowCaption

This function gets a current window caption.

Function call syntax:

```
STRING GetWindowCaption(NUMBER t_id)
```

The function returns 'nil' in case of failure.

3.15.12 GetWindowRect

This function returns the coordinates of the upper left and lower right corners of the window that contains the table ‘t_id’.

Function call syntax:

> NUMBER top, NUMBER left, NUMBER bottom, NUMBER right
> GetWindowRect(NUMBER t_id)

The function returns ‘nil’ in case of failure.

3.15.13 SetCell

This function sets the value of the cell with the specified row key and column code in the table ‘t_id’.

Function call syntax:

> BOOLEAN SetCell(NUMBER t_id, NUMBER key, NUMBER code, STRING text,
> NUMBER value)

The ‘text’ parameter is the string representation of the ‘value’ parameter. The ‘value’ parameter is optional and equals ‘0’ by default. The ‘value’ parameter is not set for columns with string data types.

If the ‘value’ parameter is not set for cells of other types, then sorting, filtering and conditional formatting will not work correctly for columns that contain such cells (see Appendix 2).

If successful, this function returns ‘true’ and ‘false’ otherwise.

3.15.14 SetWindowCaption

This function sets a new window caption.

Function call syntax:

> SetWindowCaption(NUMBER t_id, STRING str)

This function returns ‘true’ if executed successfully and ‘false’ otherwise.

3.15.15 SetWindowPos

This function sets the position for the window with the table ‘t_id’. The coordinates of the upper left corner are set to x and y, and the sizes of the window are set to dx, dy.

Function call syntax:

> BOOLEAN SetWindowPos(NUMBER t_id, NUMBER x, NUMBER y, NUMBER dx,
> NUMBER dy)

This function returns ‘true’ if executed successfully and ‘false’ otherwise.

3.15.16 SetTableNotificationCallback

Defines a callback function to process table events.

Function call syntax:

NUMBER SetTableNotificationCallback (NUMBER t_id, FUNCTION f_cb)

Parameters:

- **t_id** – a table identifier;
- **f_cb** – a callback function to process table events.

This function returns ‘1’ if executed successfully and ‘0’ otherwise.

Table event callback call syntax:

FUNCTION (NUMBER t_id, NUMBER msg, NUMBER par1, NUMBER par2)

Parameters:

- **t_id** – the identifier of the table whose message is to be processed;
- **par1** and **par2** – the values of these parameters are determined by the message type;
- **msg** – a message code.

Possible event codes:

- QTABLE_LBUTTONDOWN – the table is clicked; par1 contains the row number, and par2 contains the column number;
- QTABLE_RBUTTONDOWN – the table is right-clicked; par1 contains the row number, and par2 contains the column number;
- QTABLE_LBUTTONDBLCLK – the table is double-clicked; par1 contains the row number, and par2 contains the column number;
- QTABLE_RBUTTONDBLCLK – the table is right double-clicked, par1 contains the row number, and par2 contains the column number;
- QTABLE_SELCHANGED – the current (highlighted) row is changed; par1 is the number of the new highlighted row;
- QTABLE_CHAR – a character key is pressed; par2 contains the key code and par1 contains the current highlighted row;
- QTABLE_VKEY – a key is pressed; par2 contains the key code and par1 contains the current highlighted row;
- QTABLE_MBUTTONDOWN – the middle mouse button is clicked; par1 contains the row number, and par2 contains the column number;

- QTABLE_MBUTTONDBLCLK – the middle mouse button is double-clicked, par1 contains the row number, and par2 contains the column number;
- QTABLE_LBUTTONUP – the left mouse button is released, par1 contains the row number, and par2 contains the column number;
- QTABLE_RBUTTONUP – the right mouse button is released, par1 contains the row number, and par2 contains the column number;
- QTABLE_CLOSE – the table is closed; par1 and par2 are equal to 0.

3.15.17 RGB

This function transforms RGB components (red, green, blue) into a single number for further use in the SetColor function (see sub-section 3.15.18).

Function call syntax:

NUMBER RGB(NUMBER red, NUMBER green, NUMBER blue)

3.15.18 SetColor

This function sets the colour for a cell, column or row in the table with the id 't_id'.

Function call syntax:

BOOLEAN SetColor(NUMBER t_id, NUMBER row, NUMBER col, NUMBER b_color, NUMBER f_color, NUMBER sel_b_color, NUMBER sel_f_color)

Parameters returned by this function:

- **b_color** – background colour;
- **f_color** – text colour;
- **sel_b_color** – background colour of the selected cell;
- **sel_f_color** – text colour of the selected cell.

Depending on the **row** and **col** parameters, you can change colour of the entire table or of a specific column, row or cell.

If the colour parameter is set to QTABLE_DEFAULT_COLOR, the colour specified in the Windows colour scheme is used.

The function uses the QTABLE_NO_INDEX constant equal to '-1'.

Colour setting options for tables are as follows:

row	col	Result
Number of lines from 1 to N	Number of columns from 1 to M	A cell is highlighted with the specified colour

row	col	Result
Number of lines from 1 to N	QTABLE_NO_INDEX	A row is highlighted with the specified colour
QTABLE_NO_INDEX	Number of columns from 1 to M	A column is highlighted with the specified colour
QTABLE_NO_INDEX	QTABLE_NO_INDEX	The entire table is highlighted with the specified colour

3.15.19 Highlight

This function highlights a selected cell range in the table 't_id' with the background and text colour for a specified period; then, highlighting gradually fades out.

Function call syntax:

BOOLEAN Highlight(NUMBER t_id, NUMBER row, NUMBER col, NUMBER b_color, NUMBER f_color, NUMBER timeout)

Parameters:

- **b_color** – background colour;
- **f_color** – text colour;
- **timeout** – highlighting time in milliseconds.

To cancel highlighting, call this function with the timeout of 0. In this case, the parameters 'b_color' and 'f_color' can have any values.

Options for highlighting cells in a table are the same as the colour parameters used in the SetColor function (see sub-section 3.15.18).

3.15.20 SetSelectedRow

Function selects a certain row of table.

NUMBER row SetSelectedRow((NUMBER table_id, NUMBER row)

Parameters:

- **table_id** – table identifier;
- **row** – row number.

> **If the set value is row=-1 the last visible row of table is selected.**

If executed successfully the function returns number of the selected row, otherwise "-1".

The function works with visible presentation of table considering user filters and sorting.

3.16 Function for Working with Labels

Functions are intended to form labels and its setting up on graph.

3.16.1 AddLabel

Function adds a label with specified parameters.

NUMBER AddLabel(STRING chart_tag, TABLE label_params)

Parameters:

- **chart_tag** – tag of the graph to which the label is bound;
- **label_params** – table with label parameters.

The function returns the numerical identifier of label. If executed unsuccessfully, the function returns “nil”.

Format of table with label parameters:

No.	Parameter	Type	Description
1	TEXT	STRING	Label signature (if not required, this is an empty string)
2	IMAGE_PATH	STRING	Path to the image displayed as a label (if the image is not required, this is an empty string)
3	ALIGNMENT	STRING	Text position relative to the image (four variants are possible: LEFT, RIGHT, TOP, BOTTOM)
4	YVALUE	DOUBLE	Y-axis value of the parameter to which the label is bound
5	DATE	DOUBLE	YYYYMMDD date format to which the label is bound
6	TIME	DOUBLE	HHMMSS time format to which the label is bound
7	R	DOUBLE	Red color component in RGB format, which is a number within an interval [0;255]
8	G	DOUBLE	Green color component in RGB format, which is a number within an interval [0;255]
9	B	DOUBLE	Blue color component in RGB format, which is a number within an interval [0;255]
10	TRANSPARENCY	DOUBLE	Label transparency as a percentage. The value should fall within a range [0; 100]

No.	Parameter	Type	Description
11	TRANSPARENT_BACKGROUND	DOUBLE	Market transparency. Possible values include 0 for transparency disabled or 1 for transparency enabled.
12	FONT_FACE_NAME	STRING	Font name (e.g., Arial)
13	FONT_HEIGHT	DOUBLE	Font size
14	HINT	STRING	Popup hint

3.16.2 DelLabel

Function deletes a label with specified parameters.

BOOLEAN DelLabel(STRING chart_tag, NUMBER label_id)

Parameters:

- **chart_tag** – tag of the graph to which the label is bound;
- **label_id** – label identifier.

If executed successfully the function returns “true”, otherwise “false”.

3.16.3 DelAllLabels

Function deletes all labels on chart with the specified graph.

BOOLEAN DelAllLabels(STRING chart_tag)

Parameters:

- **chart_tag** – tag of the graph to which the label is bound.

If executed successfully the function returns “true”, otherwise “false”.

3.16.4 GetLabelParams

Function allows getting label parameters

TABLE GetLabelParams(STRING chart_tag, NUMBER label_id)

Parameters:

- **chart_tag** – tag of the graph to which the label is bound;
- **label_id** – label identifier.

Functions returns the table with label parameters. If executed unsuccessfully, the function returns “nil”.

3.16.5 SetLabelParams

Function sets parameters for label with specified identifier.

BOOLEAN SetLabelParams(STRING chart_tag, NUMBER label_id, TABLE label_params)

Parameters:

- **chart_tag** – tag of the graph to which the label is bound;
- **label_id** – label identifier;
- **label_params** – table with parameters of new label.

If executed successfully the function returns “true”, otherwise “false”.

3.17 Functions for Ordering Level II Quotes Table

3.17.1 Subscribe_Level_II_Quotes

This function orders receiving the Level II Quotes table from the server for a specified class and security.

BOOLEAN Subscribe_Level_II_Quotes(STRING class_code, STRING sec_code)

Parameters:

- **class_code** – class code;
- **sec_code** – security code.

If executed successfully the function returns “true”.

3.17.2 Unsubscribe_Level_II_Quotes

This function cancels the order for receiving the Level II Quotes table from the server for a specified class and security.

BOOLEAN Unsubscribe_Level_II_Quotes(STRING class_code, STRING sec_code)

Parameters:

- **class_code** – class code;
- **sec_code** – security code.

If executed successfully the function returns “true”.

3.17.3 IsSubscribed_Level_II_Quotes

This function allows to know whether the Level II Quotes table for a specified class and security is ordered from the server.

BOOLEAN IsSubscribed_Level_II_Quotes (STRING class_code, STRING sec_code)

Parameters:

- **class_code** – class code;
- **sec_code** – security code.

This function returns 'true' if the Level II Quotes table for **class_code** class and **sec_code** security has already been ordered.

4. Data Structures

4.1 Classes

Class table parameters

Parameter	Type	Description
firmid	STRING	Firm ID
name	STRING	Class name
class_code	STRING	Class code
npars	NUMBER	Number of parameters in the class
nsecs	NUMBER	Number of securities in the class

Example:

```
t={
        ["firmid"]="NC0038900000",
        ["name"]="Broker quotes",
        ["code"]="BQUOTE",
        ["npars"]=38,
        ["nsecs"]=28
}
```

See also the descriptions of getItem and getNumberOf.

4.2 Firms

Firm table parameters

Parameter	Type	Description
firmid	STRING	Firm ID
firm_name	STRING	Class name
status	NUMBER	Status
exchange	STRING	Trading venue

See also the descriptions of getItem and getNumberOf.

4.3 Anonymous Trades

Anonymous trades table parameters

Parameter	Type	Description
trade_num	NUMBER	Trade identifier
flags	NUMBER	Flags for Anonymous Trades Table
price	NUMBER	Price
qty	NUMBER	Volume
value	NUMBER	Trade volume
accruedint	NUMBER	Accrued interest
yield	NUMBER	Yield
settlecode	STRING	Settlement code
reporate	NUMBER	REPO rate
repovalue	NUMBER	REPO sum
repo2value	NUMBER	REPO buyback volume
repoterm	NUMBER	REPO period in days
sec_code	STRING	Security code
class_code	STRING	Class code

Parameter	Type	Description
datetime	TABLE	Date and Time Format Used in Tables
period	NUMBER	Period of trading session. Valid values: _ 0 – opening; _ 1 – normal; _ 2 – closing

4.4 Trades

Trades table parameters

Parameter	Type	Description
trade_num	NUMBER	Trade number in the trading system
order_num	NUMBER	Order number in the trading system
brokerref	STRING	Comment, usually: <client code>/<order number>
userid	STRING	Trader identifier
firmid	STRING	Dealer identifier
account	STRING	Trading account
price	NUMBER	Price
qty	NUMBER	Number of instruments in the last trade, lots
value	NUMBER	Volume in cash
accruedint	NUMBER	Accrued interest
yield	NUMBER	Yield
settlecode	STRING	Settlement code
cpfirmid	STRING	Partner's firm code
flags	NUMBER	Flags for the Orders, Negdeal Orders, Trades, and Trades for Execution Tables
price2	NUMBER	Buyback price
reporate	NUMBER	REPO rate (%)
client_code	STRING	Client code

Parameter	Type	Description
accrued2	NUMBER	Profit (%) of the buyback date
repovalue	NUMBER	REPO sum
repo2value	NUMBER	REPO buyback value
start_discount	NUMBER	Start discount (%)
lower_discount	NUMBER	Lower discount (%)
upper_discount	NUMBER	Upper discount (%)
block_securities	NUMBER	Indicates if collateral is locked (Yes/No)
clearing_comission	NUMBER	Clearing commission of the exchange
exchange_comission	NUMBER	Exchange commission
tech_center_comission	NUMBER	Technical centre commission
settle_date	NUMBER	Settlement date
settle_currency	STRING	Settle currency
trade_currency	STRING	Currency
exchange_code	STRING	Exchange code in the trading system
station_id	STRING	Workstation identifier
sec_code	STRING	Code of the instrument in an order
class_code	STRING	Class code
datetime	TABLE	Date and Time Format Used in Tables
bank_acc_id	STRING	Settlement account id/clearing organization code
broker_comission	NUMBER	Brokerage commission, accurate to 2 digits. This field is reserved for use in the future.
linked_trade	NUMBER	Number of showcase trade in the trading system for REPO trades with CCP and SWAP
period	NUMBER	Period of trading session. Valid values: _ 0 – opening; _ 1 – normal; _ 2 – closing

4.5 Orders

Orders table parameters

Parameter	Type	Description
order_num	NUMBER	Order number in the trading system
* flags	NUMBER	Flags for the Orders, Negdeal Orders, Trades, and Trades for Execution Tables
brokerref	STRING	Comment, usually: <client code>/<order number>
userid	STRING	Trader identifier
firmid	STRING	Firm ID
account	STRING	Trading account
price	NUMBER	Price
qty	NUMBER	Quantity in lots
balance	NUMBER	Balance
value	NUMBER	Volume in cash
accruedint	NUMBER	Accrued interest
yield	NUMBER	Yield
trans_id	NUMBER	Transaction identifier
client_code	STRING	Client code
price2	NUMBER	Buyback price
settlecode	STRING	Settlement code
uid	NUMBER	User identifier
exchange_code	STRING	Exchange code in the trading system
activation_time	NUMBER	Time of activation
linkedorder	NUMBER	Order number in the trading system
expiry	NUMBER	Order expiry date
sec_code	STRING	Code of the instrument in an order
class_code	STRING	Order class code
datetime	TABLE	Date and Time Format Used in Tables

Parameter	Type	Description
withdraw_datetime	TABLE	Date and Time Format Used in Tables of order cancellation
bank_acc_id	STRING	Settlement account id/clearing organization code
value_entry_type	NUMBER	Methods of specifying an order volume Possible values include: _ 0 – by quantity; _ 1 – by volume
repoterm	NUMBER	REPO period in calendar days
repovalue	NUMBER	REPO amount as of the current date, accurate to 2 digits.
repo2value	NUMBER	REPO buyback volume, accurate to 2 digits.
repo_value_balance	NUMBER	REPO amount excluding the sum of cash raised or borrowed REPO funds in the unfilled amount of the order as of the current date, accurate to 2 digits.
start_discount	NUMBER	Start discount, %
reject_reason	STRING	The reason why the broker rejected the order
ext_order_flags	NUMBER	A bit field reserved for specific parameters from western venues.
min_qty	NUMBER	The minimum acceptable quantity that can be specified in an order for this instrument. 0 means that the quantity limit is not set
exec_type	NUMBER	Order execution type. 0 means that the value is not set
side_qualifier	NUMBER	A field reserved for the parameters of western trading venues 0 means that the value is not set
acnt_type	NUMBER	A field reserved for the parameters of western trading venues 0 means that the value is not set
capacity	NUMBER	A field reserved for the parameters of western trading venues 0 means that the value is not set
passive_only_order	NUMBER	A field reserved for the parameters of western trading venues If it has the value of 0, then the value is not set

4.6 Current Positions for Client Accounts

Description of the Table of current positions for accounts:

Parameter	Type	Description

Parameter	Type	Description
firmid	STRING	Firm ID
sec_code	STRING	Security code
trdaccid	STRING	Trading account
depaccid	STRING	Depo account
openbal	NUMBER	Opening position
currentpos	NUMBER	Current position
plannedpossell	NUMBER	Planned selling
plannedposbuy	NUMBER	Planned buying
planbal	NUMBER	Check balance of a simple clearing, equals to the opening balance minus the volume of the planned sell position included in that simple clearing
usqtyb	NUMBER	Buy
usqtys	NUMBER	Sell
planned	NUMBER	Planned balance, equals to the current balance minus the volume of the planned sell position
settlebal	NUMBER	Planned position after settlements
bank_acc_id	STRING	Settlement account id/clearing organization code
firmuse	NUMBER	Indicates a collateral account. Possible values include: _ '0' refers to normal accounts; _ '1' refers to a collateral account

4.7 Current Positions for Instruments

Description of the Table of current positions for instruments:

Parameter	Type	Description
firmid	STRING	Firm
seccode	STRING	Security code
openbal	NUMBER	Opening position
currentpos	NUMBER	Current position

Parameter	Type	Description
plannedposbuy	NUMBER	Volume of active buy orders, in securities
plannedpossell	NUMBER	Volume of active sell orders, in securities
usqtyb	NUMBER	Buy
usqtys	NUMBER	Sell

4.8 Stop Orders

Description of the Stop orders table parameters:

Parameter	Type	Description
order_num	NUMBER	Registration number of a stop order on the QUIK server
ordertime	NUMBER	Creation time
flags	NUMBER	Flags for Anonymous Trades Table
brokerref	STRING	Comment, usually: <client code>/<order number>
firmid	STRING	Dealer identifier
account	STRING	Trading account
condition	NUMBER	Stop price direction Possible values include: _ 4 means '<='; _ 5 means '>='
condition_price	NUMBER	Stop price
price	NUMBER	Price
qty	NUMBER	Quantity in lots
linkedorder	NUMBER	Order number in the trading system placed after a stop price is reached
expiry	NUMBER	Order expiry date
trans_id	NUMBER	Transaction identifier
client_code	STRING	Client code
co_order_num	NUMBER	Linked order
co_order_price	NUMBER	Linked order price

Parameter	Type	Description
stop_order_type	NUMBER	Stop order type. Possible values include: _ 1 – stop limit; _ 2 – condition by another instrument; _ 3 – with a linked order; _ 6 – take-profit; _ 7 – 'if done' stop limit; _ 8 – 'if done' take profit; _ 9 – take-profit and stop limit
orderdate	NUMBER	Creation date
alltrade_num	NUMBER	Trade of the primary order
stopflags	NUMBER	Additional Flags for Stop Orders Table
offset	NUMBER	Offset from min/max
spread	NUMBER	Protective spread
balance	NUMBER	Active amount
uid	NUMBER	User identifier
filled_qty	NUMBER	Filled quantity
withdraw_time	NUMBER	Order cancellation time
condition_price2	NUMBER	Stop limit price (for orders of the 'Take profit and stop limit' type)
active_from_time	NUMBER	Time when an order of the 'Take profit and stop limit' type becomes active
active_to_time	NUMBER	Time when an order of the 'Take profit and stop limit' type expires
sec_code	STRING	Code of the instrument in an order
class_code	STRING	Order class code
condition_sec_code	STRING	Stop price instrument code
condition_class_code	STRING	Stop price class code

4.9 Futures Limits

Description of a futures limit parameters:

Parameter	Type	Description
firmid	STRING	Firm ID
trdaccid	STRING	Trading account
limit_type	NUMBER	Type of limit Possible values include: _ 1 – Cash; _ 2 – Secured funds; _ 3 – Total; _ 4 – Clearing roubles; _ 5 – Clearing secured roubles; _ 6 – Limit for open positions on the spot market
liquidity_coef	NUMBER	Liquidity coefficient
cbp_prev_limit	NUMBER	Previous limit for open positions
cbplimit	NUMBER	Limit for open positions
cbplused	NUMBER	Current net positions
cbplplanned	NUMBER	Planned net positions
varmargin	NUMBER	Variation margin
accruedint	NUMBER	Accrued coupon interest
cbplused_for_orders	NUMBER	Current net positions (for orders)
cbplused_for_positions	NUMBER	Current net positions (for open orders)
options_premium	NUMBER	Options premium
ts_comission	NUMBER	Exchange commission
kgo	NUMBER	Client collateral coefficient
currcode	STRING	Currency in which a limit is transmitted
real_varmargin	NUMBER	Actual variation margin calculated during clearing, accurate to 2 digits. The ‘varmargin’ transmits a variation margin calculated taking into account the set price alteration limits

4.10 Client Account Positions (Futures)

Description of the Table of positions for futures:

Parameter	Type	Description
firmid	STRING	Firm ID
trdaccid	STRING	Trading account
sec_code	STRING	Futures contract code
type	NUMBER	Type of limit Possible values include: _ Main account; _ Client and additional accounts; _ Accounts of all traders; _ <blank>
startbuy	NUMBER	Opening long positions
startsell	NUMBER	Opening short positions
startnet	NUMBER	Opening net positions
todaybuy	NUMBER	Current long positions
todaysell	NUMBER	Current short positions
totalnet	NUMBER	Current net positions
openbuys	NUMBER	Open for buying
opensells	NUMBER	Open for selling
cbplused	NUMBER	Estimated current net positions
cbplplanned	NUMBER	Planned net positions
varmargin	NUMBER	Variation margin
avrposnprice	NUMBER	Effective price of positions
positionvalue	NUMBER	Position value
real_varmargin	NUMBER	Actual variation margin calculated during clearing, accurate to 2 digits. The existing 'varmargin' field contains a variation margin calculated with regard to the specified price variation limits.
total_varmargin	NUMBER	Total variation margin after main clearing calculated for all positions, accurate to 2 digits

4.11 Cash Limits

Cash limits table parameters:

Parameter	Type	Description
currcode	STRING	Currency code
tag	STRING	Settlement tag
firmid	STRING	Firm ID
client_code	STRING	Client code
openbal	NUMBER	Opening cash balance
openlimit	NUMBER	Opening cash limit
currentbal	NUMBER	Current cash balance
currentlimit	NUMBER	Current cash limit
locked	NUMBER	Locked quantity
locked_value_coef	NUMBER	Value of the buy orders for non-marginal securities
locked_margin_value	NUMBER	Value of the buy orders for marginal securities
leverage	NUMBER	Leverage
limit_kind	NUMBER	Type of limit Possible values include: _ 0 – usual limits; _ otherwise – technological limits

4.12 Removing Cash Limit

Description of the Cash limit removal table

Parameter	Type	Description
currcode	STRING	Currency code
tag	STRING	Settlement tag
client_code	STRING	Client code
firmid	STRING	Firm ID

Parameter	Type	Description
limit_kind	NUMBER	Type of limit Possible values include: _ 0 – usual limits; _ otherwise – technological limits

4.13 Deleting Security Limit

Description of the Deleting security limit table

Parameter	Type	Description
sec_code	STRING	Security code
trdaccid	STRING	Trading account code
firmid	STRING	Firm ID
client_code	STRING	Client code
limit_kind	NUMBER	Type of limit Possible values include: _ 0 – usual limits; _ value other than ‘0’ – technological limits

4.14 Deleting Futures Limit

Description of the Deleting futures limit table:

Parameter	Type	Description
firmid	STRING	Firm ID
limit_type	STRING	Limit type

4.15 Limits for Securities

Description of the Limits for securities table

Parameter	Type	Description
sec_code	STRING	Security code
trdaccid	STRING	Depo account

Parameter	Type	Description
firmid	STRING	Firm ID
client_code	STRING	Client code
openbal	NUMBER	Opening balance for securities
openlimit	NUMBER	Open limit for securities
currentbal	NUMBER	Current securities balance
currentlimit	NUMBER	Current limit for securities
locked_sell	NUMBER	Locked for selling a number of lots
locked_buy	NUMBER	Locked for buying a number of lots
locked_buy_value	NUMBER	Value of instruments locked for buying
locked_sell_value	NUMBER	Value of instruments locked for selling
awg_position_price	NUMBER	WA.position price
limit_kind	NUMBER	Type of limit Possible values include: _ 0 – usual limits; _ value other than '0' – technological limits

4.16 Cash Positions

Description of the Cash positions table:

Parameter	Type	Description
firmid	STRING	Firm ID
currcode	STRING	Currency code
tag	STRING	Settlement tag
description	STRING	Description
openbal	NUMBER	Opening position
currentpos	NUMBER	Current position
plannedpos	NUMBER	Planned balance
limit1	NUMBER	External cash limit
limit2	NUMBER	Internal cash limit

Parameter	Type	Description
orderbuy	NUMBER	In sell orders
ordersell	NUMBER	In buy orders
netto	NUMBER	Netto position
plannedbal	NUMBER	Planned position
debit	NUMBER	Debit
credit	NUMBER	BankAccID
bank_acc_id	STRING	Account ID
margincall	NUMBER	Marginal requirement at the beginning of trading
settlebal	NUMBER	Planned position after settlements

4.17 Negotiated Deal Orders

Description of the Negotiated deal orders table:

Parameter	Type	Description
neg_deal_num	NUMBER	Number
neg_deal_time	NUMBER	Time when an order was placed
flags	NUMBER	Flags for the Orders, Negdeal Orders, Trades, and Trades for Execution Tables
brokerref	STRING	Comment, usually: <client code>/<order number>
userid	STRING	Trader
firmid	STRING	Dealer identifier
cpuserid	STRING	Partner's trader
cpfirmid	STRING	Partner's firm code
account	STRING	Account
price	NUMBER	Price
qty	NUMBER	Volume
matchref	STRING	Reference
settlecode	STRING	Settlement code

Parameter	Type	Description
yield	NUMBER	Yield
accruedint	NUMBER	Coupon rate (%)
value	NUMBER	Volume
price2	NUMBER	Buyback price
reporate	NUMBER	REPO rate (%)
refundrate	NUMBER	Refund rate (%)
trans_id	NUMBER	Trans ID
client_code	STRING	Client code
repoentry	NUMBER	REPO order entry type Possible values include: _ Price1+Rate; _ Rate+Price2; _ Price1+Price2
repovalue	NUMBER	REPO sum
repo2value	NUMBER	REPO buyback value
repoterm	NUMBER	REPO period
start_discount	NUMBER	Start discount (%)
lower_discount	NUMBER	Lower discount (%)
upper_discount	NUMBER	Upper discount (%)
block_securities	NUMBER	Indicates if collateral is locked (Yes/No)
uid	NUMBER	User identifier
withdraw_time	NUMBER	Order cancellation time
neg_deal_date	NUMBER	Order placement date
balance	NUMBER	Balance
origin_repovalue	NUMBER	Original REPO sum
origin_qty	NUMBER	Original quantity
origin_discount	NUMBER	Original discount percent
neg_deal_activation_date	NUMBER	Order activation date
neg_deal_activation_time	NUMBER	Order activation time

Parameter	Type	Description
quoteno	NUMBER	Non-negotiated counter-order
settle_currency	NUMBER	Settle currency
sec_code	STRING	Security code
class_code	STRING	Class code
bank_acc_id	STRING	Settlement account id/clearing organization code
withdraw_date	NUMBER	Data when an negotiated order was cancelled, in the format YYYYMMDD
linkedorder	NUMBER	Number of the previous order Displayed with '0' accuracy

4.18 Trades for Execution

Description of the Trades for execution table

Parameter	Type	Description
trade_num	NUMBER	Trade number
trade_date	NUMBER	Trading date
settle_date	NUMBER	Settlement date
flags	NUMBER	Flags for the Orders, Negdeal Orders, Trades, and Trades for Execution Tables
brokerref	STRING	Comment, usually: <client code>/<order number>
firmid	STRING	Dealer identifier
account	STRING	Depo account
cpfirmid	STRING	Partner's firm code
cpaccount	STRING	Partner's DEPO account
price	NUMBER	Price
qty	NUMBER	Volume
value	NUMBER	Volume
settlecode	STRING	Settlement code
report_num	NUMBER	Report
cpreport_num	NUMBER	Partner's report

Parameter	Type	Description
accruedint	NUMBER	Coupon rate (%)
repotradeno	NUMBER	Trade number of the first leg of REPO
price1	NUMBER	Prices of the first leg of REPO
reporate	NUMBER	REPO rate (%)
price2	NUMBER	Buyback price
client_code	STRING	Client code
ts_comission	NUMBER	Trading system's commission
balance	NUMBER	Balance
settle_time	NUMBER	Execution time
amount	NUMBER	Amount of liability
repovalue	NUMBER	REPO sum
repoterm	NUMBER	REPO period
repo2value	NUMBER	REPO buyback value
return_value	NUMBER	REPO return value
discount	NUMBER	Discount (%)
lower_discount	NUMBER	Lower discount (%)
upper_discount	NUMBER	Upper discount (%)
block_securities	NUMBER	Indicates if collateral is locked (Yes/No)
urgency_flag	NUMBER	Execute (Yes/No)
type	NUMBER	Type. Possible values include: _ 0 – negotiated deal; _ 1 – first leg of REPO; _ 2 – second leg of REPO; _ 3 – compensation payment; _ 4 – defaulter: deferred commitments and claims; _ 5 – aggrieved person: deferred commitments and claims

Parameter	Type	Description
operation_type	NUMBER	Direction Possible values include: _ 1 – 'Credit'; _ 2 – 'Write off'
expected_discount	NUMBER	Discount after payment (%)
expected_quantity	NUMBER	Quantity after payment
expected_repovalue	NUMBER	REPO sum after payment
expected_repo2value	NUMBER	REPO buyback value after payment
expected_return_value	NUMBER	REPO return sum after payment
order_num	NUMBER	Order number
report_trade_date	NUMBER	Date of settlement
settled	NUMBER	Trade settlement status Possible values include: _ 1 – Processed; _ 2 – Not processed; _ 3 – Is processing
clearing_type	NUMBER	Type of clearing. Possible values include: _ 1 – 'Not set'; _ 2 – 'Simple'; _ 3 – 'Multilateral'
report_comission	NUMBER	Report comission
coupon_payment	NUMBER	Coupon payment
principal_payment	NUMBER	Principal debt payment
principal_payment_date	NUMBER	Date of principal debt payment
nextdaysettle	NUMBER	Date of the next settlement day
settle_currency	STRING	Settle currency
sec_code	STRING	Security code
class_code	STRING	Class code

4.19 Trading accounts

Description of the Trading accounts table:

Parameter	Type	Description
class_codes	STRING	List of class codes separated by the '|' character
firmid	STRING	Firm ID
trdaccid	STRING	Trading account code

4.20 Reports on Trades for Execution

Description of the Reports on trades for execution table:

Parameter	Type	Description
report_num	NUMBER	Report
report_date	NUMBER	Report date
flags	NUMBER	Flags for the Orders, Negdeal Orders, Trades, and Trades for Execution Tables
userid	STRING	User identifier
firmid	STRING	Firm ID
account	STRING	Depo account
cpfirmid	STRING	Partner's firm code
cpaccount	STRING	Partner's trading account code
qty	NUMBER	Quantity of securities, lots
value	NUMBER	Trade volume, in roubles
withdraw_time	NUMBER	Order cancellation time
report_type	NUMBER	Report type
report_kind	NUMBER	Report kind
commission	NUMBER	Trade commission, in roubles

4.21 Instruments

Description of the Instruments table:

Parameter	Type	Description
code	STRING	Security code

Parameter	Type	Description
name	STRING	Instrument name
short_name	STRING	Short name for an instrument
class_code	STRING	Instrument class code
class_name	STRING	Instrument class name
face_value	NUMBER	Face-value
face_unit	STRING	Face-value currency
scale	NUMBER	Precision (a number of significant digits after a decimal point)
mat_date	NUMBER	Expiration date
lot_size	NUMBER	Lot size
isin_code	STRING	ISIN
min_price_step	NUMBER	Minimum price step

4.22 Chart Candlesticks

Parameters of a chart candlestick:

Parameter	Type	Description
open	NUMBER	Opening price
close	NUMBER	Closing price
high	NUMBER	Highest tolerable price
low	NUMBER	Lowest tolerable price
volume	NUMBER	Last trade volume
datetime	TABLE	Date and time
doesExist	NUMBER	Specifies whether an indicator is calculated if there is a candlestick. Possible values include: _ 0 – indicator is not calculated; _ 1 – indicator is calculated

4.23 Date and Time Format Used in Tables

The description of date and time format used in some tables:

Parameter	Type	Description
mcs	NUMBER	Microseconds
ms	NUMBER	Milliseconds
sec	NUMBER	Seconds
min	NUMBER	Minutes
hour	NUMBER	Hours
day	NUMBER	Day
week_day	NUMBER	Weekday number
month	NUMBER	Month
year	NUMBER	Year

These parameters must be set for correct display of date and time

4.24 Transactions

Transactions parameters description.

Parameter	Type	Description
trans_id	NUMBER	Transaction's user ID
status	NUMBER	Status
result_msg	STRING	Message text
time	NUMBER	Time
uid	NUMBER	Identifier
flags	NUMBER	Transaction's flags (not used)
server_trans_id	NUMBER	Transaction ID on the server
*order_num	NUMBER	Order number
*price	NUMBER	Price
*quantity	NUMBER	Volume
*balance	NUMBER	Balance
*firm_id	STRING	Firm ID

Parameter	Type	Description
*account	STRING	Trading account
*client_code	STRING	Client code
*brokerref	STRING	Comment
*class_code	STRING	Class code
*sec_code	STRING	Security code

* – the value of this parameter can be **nil.**

5. Description of Bit Flags

5.1 Flags for the Orders, Negdeal Orders, Trades, and Trades for Execution Tables

Flag is set	Value
bit 0 (0x1)	The order is active, otherwise it is inactive
bit 1 (0x2)	The order was cancelled. If this flag is not set and the value of the bit 0 equals 0, then this order is filled
bit 2 (0x4)	A sell order; otherwise it is a buy order. This flag determines a trade's direction (BUY/SELL) for trades and trades for execution
bit 3 (0x8)	A limit order; otherwise it is a market order
bit 4 (0x10)	Allow / forbid trades at different prices
bit 5 (0x20)	Fill the order immediately, or cancel it (FILL OR KILL)
bit 6 (0x40)	Market-maker's order For negotiated orders, this means that the order was sent to a counterparty
bit 7 (0x80)	For negotiated orders, this means that the order was received from a counterparty
bit 8 (0x100)	Kill remain
bit 9 (0x200)	An iceberg order

5.2 Flags for Anonymous Trades Table

Flag is set	Value
bit 0 (0x1)	Sell trade
bit 1 (0x2)	Buy trade

If the flags are not set, the order direction is not determined.

5.3 Flags for Stop Orders Table

Flag is set	Value
bit 0 (0x1)	Order is active, otherwise is inactive
bit 1 (0x2)	The order was cancelled. If this flag is not set and the value of the bit 0 equals 0, then this order is filled
bit 2 (0x4)	A sell order; otherwise it is a buy order
bit 3 (0x8)	A limit order
bit 5 (0x20)	A stop order awaiting activation
bit 6 (0x40)	A stop order from another server
bit 8 (0x100)	This flag is set for an 'if done' take-profit order in case when the original order is partially filled and the activation condition of the take-profit order for the filled amount is met
bit 9 (0x200)	A stop order is activated manually
bit 10 (0x400)	A stop order was executed, but rejected by the trading system
bit 11 (0x800)	A stop order was executed, but didn't pass the limit control
bit 12 (0x1000)	A stop order was killed, since the linked order was killed
bit 13 (0x2000)	A stop order is killed, since the linked order was filled
bit 15 (0x8000)	Calculation of minimum/maximum is in the process

5.4 Additional Flags for Stop Orders Table

Flag is set	Value
bit 0 (0x1)	Use balance of the primary order

Flag is set	Value
bit 1 (0x2)	Kill the stop order if this order is filled partially
bit 2 (0x4)	Activate the stop-order if the linked order is filled partially
bit 3 (0x8)	The offset is specified as a percentage; otherwise, in price points
bit 4 (0x10)	Protective spread is specified as a percentage, otherwise in price points
bit 5 (0x20)	A stop order expires today
bit 6 (0x40)	A stop order's validity interval is set
bit 7 (0x80)	A take profit order is executed at the marked price
bit 8 (0x100)	A stop order is executed at the marked price

5.5 Cash Commitments and Claims

Description of the Cash commitments and claims table:

Parameter	Type	Description
firmid	STRING	Firm ID
bank_acc_id	STRING	Settlement account id/clearing organization code
settle_date	NUMBER	Settlement date
netto	NUMBER	Netto position
debit	NUMBER	Debit
credit	NUMBER	BankAccID

5.6 Asset Commitments and Claims

Description of the Asset commitments and claims table:

Parameter	Type	Description
firmid	STRING	Firm ID
depo_account	STRING	Number of the depo account in the Depository (NDC)
account	STRING	Trading account
bank_acc_id	STRING	Settlement account id/clearing organization code

Parameter	Type	Description
settle_date	NUMBER	Settlement date
qty	NUMBER	Quantity of securities in trades
qty_buy	NUMBER	Quantity of security in buy orders
qty_sell	NUMBER	Quantity of security in sell orders
netto	NUMBER	Netto position
debit	NUMBER	Debit
credit	NUMBER	BankAccID
sec_code	STRING	Code of the instrument in an order
class_code	STRING	Order class code

6. Functions for Working with Bit Masks in Data Structures

6.1 bit.tohex

This function converts the first argument into a hexadecimal string. A number of characters in the string is defined by the optional second parameter.

Function call syntax:

STRING bit.tohex(NUMBERx [, NUMBER n])

6.2 bit.bnot

This function returns the result of the bitwise NOT operation performed on the **x** argument.

Function call syntax:

NUMBER bit.bnot(NUMBER x)

6.3 bit.band

This function returns the result of the bitwise AND operation performed on the arguments. There can be several arguments; the arguments **x1** and **x2** are required.

Function call syntax:

NUMBER bit.band(NUMBER x1, NUMBER x2, ...)

6.4 bit.bor

This function returns the result of the bitwise OR operation performed on the arguments. There can be several arguments; the arguments **x1** and **x2** are required.

Function call syntax:

NUMBER bit.bor(NUMBER x1, NUMBER x2,...)

6.5 bit.bxor

This function returns the result of the bitwise XOR operation (exclusive OR) performed on the arguments. There can be several arguments; the arguments **x1** and **x2** are required.

Function call syntax:

NUMBER bit.bxor(NUMBER x1, NUMBER x2,...)

7. Indicators of the Technical Analysis

7.1 General Information

Technical analysis indicators represent a separate class of scripts that meet certain conditions and are located in **LuaIndicators** folder in the terminal's directory. If the file doesn't exist in the directory create it manually. List of scripts is available in dialogue **Tables / LUA / Available scripts.**

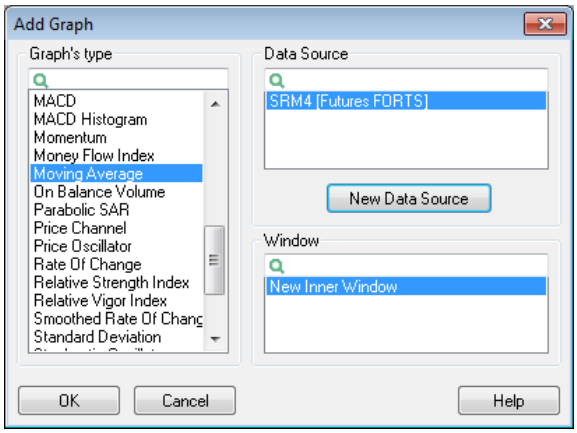
When a new indicator is added to a chart the **qlua** plugin scans **LuaIndicators** folder and checks the files with lua and luac extention (compied lua scripts) to meet the following requirements:

- Function Init is defined;
- Function OnCalculate is defined;
- Lua table named Settings that contains Name field.

Example of the minimum correct code for an indicator:

```
Settings={}
Settings.Name = "minimal"
function Init()
return 1
end
function OnCalculate(index)
return 1
end
```

List of the available indicators is transmitted in **qchart** module and further it is available in the standard dialogue of adding an indicator to a chart:



List of chart types is sorted in alphabetical order except of types Price and Volume that are always at the top of the list.

7.2 Functions and Global Variables of Script's Indicator

7.2.1 Init

The function is once called when adding an indicator on a chart, returns the value defining the number of lines in indicator.

Function call syntax:

```
NUMBER Init()
```

For example, for indicator Alligator the function returns the value 3.

7.2.2 OnCalculate

The function is called when appearing a new or changing an existent candlestick in the data source for an indicator.

Function call syntax:

```
NUMBER v1 [, NUMBER vn] OnCalculate(NUMBER index)
```

Parameters:

- **index** – index of a candlestick in the data source. Start value is 1.

If the value v_i is not defined then the function returns nil as the value of a line in interval index.

For example:

```
function Init()
	myDEMA = cached_DTEMA()
	return 2
end
function OnCalculate(index)
	x, y = myDEMA(index, Settings.period, Settings.calc_mode) --exponential
	return x, y
end
```

7.2.3 OnDestroy

The function is called when removing an indicator from a chart or when closing the window of a chart. This function is not obligatory for an indicator.

Function call syntax:

OnDestroy ()

7.2.4 Functions for Access to the Data Source

- Functions for access to the data source **O**, **H**, **L**, **C**, **V**, **T** take a candlestick index as the value of parameter and return an appropriate value in format:

  NUMBER <name of function>(NUMBER index)

- Function **Size** returns a current number of candlesticks in the data source. Function call syntax:

  NUMBER Size()

  Description the the function's values **O**, **H**, **L**, **C**, **V**, **T**, **Size** is the same as given in sub-section 3.10.4.

Example of a script realizing Moving Average indicator:

```
Settings={}
Settings.Name = "SimpleMA"
Settings.mode = "C"
Settings.period = 5
Settings.str_field = "STRING field"


function dValue(i,param)
	local v = param or "C"
	if v == "O" then
		return O(i)
	elseif v == "H" then
		return H(i)
```

```
        elseif v == "L" then
                return L(i)
        elseif v == "C" then
                return C(i)
        elseif v == "V" then
                return V(i)
        elseif v == "M" then
                return (H(i) + L(i))/2
        elseif v == "T" then
                return (H(i) + L(i)+C(i))/3
        elseif v == "W" then
                return (H(i) + L(i)+2*C(i))/4
        else
        return C(i)
        end
end

function Init()
        return 1
end
function OnCalculate(idx)
        local per = Settings.period
        local mode = Settings.mode
        local lValue = iValue
        if idx >= per then
                local ma_value=0
                for j = (idx-per)+1, idx do
                ma_value = ma_value+dValue(j, mode)
                end
                return ma_value/per
        else
                return nil
        end
end
```

7.2.5 getDataSourceInfo

The function is intended to get information on the data source for an indicator.

TABLE info getDataSourceInfo()

The function returns the Lua table with parameters:

Parameter	Type	Description
interval	NUMBER	Current interval (time frame) of a chart

Parameter	Type	Description
class_code	STRING	Class code of the data source
sec_code	STRING	Security code of the data source
param	STRING	Name of the Quotes table parameter on which a chart is created. If the field is empty then a chart is created on the basis of the Time and Sales table

Valid values of Interval field:

Returned value	Interval
0	Tick
1	1 minute
2	2 minutes
3	3 minutes
4	4 minutes
5	5 minutes
6	6 minutes
10	10 minutes
15	15 minutes

Returned value	Interval
20	20 minutes
30	30 minutes
60	1 hour
120	2 hours
240	4 hours
-1	1 day
-2	1 week
-3	1 month

7.2.6 SetValue

The function is intended to set a specified value on a selected line for a certain indicator's candlestick:

BOOLEAN SetValue(NUMBER index, NUMBER line_number, NUMBER value)

The parameters:

- **index** – candlestick's index;
- **line_number** – line number;
- **value** – a value to be set.

If executed successfully the function returns "true", otherwise "false".

The example is given below.

7.2.7 GetValue

The function is intended to determine the value set on a selected line for a certain indicator's candlestick:

NUMBER value GetValue(NUMBER index, NUMBER line_number)

The parameters:

- **index** – candlestick's index;
- **line_number** – line number.

The function returns **value** set for a **line_number** of the candlestick **index**. If an error occurs, the function returns 'nil'.

For example:

```
function OnCalculate(i)
	local ret_value = 0
	if i == 1 then
		ret_value = 1
	else
		ret_value = GetValue(i-1, 1)+2
	end
	if i%3 == 0 then
		ret_value = SetValue(i-1, 1, 2)
	end
	return ret_value
end
```

7.2.8 SetRangeValue

The function is intended to set a specified value on a selected line for a certain indices' interval of the indicator's candlestick:

BOOLEAN SetRangeValue(NUMBER line_number, NUMBER start_index, NUMBER end_index, NUMBER value)

The parameters:

- **line_number** – line number;
- **start_index** – index of the starting interval candlestick;
- **end_index** – index of the ending interval candlestick;
- **value** – a value to be set.

The function sets **value** for a **line_number** from the **start_index** to the **end_index** indices inclusively.

If executed successfully the function returns “true”, otherwise “false”.

For example:

```
function OnCalculate(index)
	local range = Settings.range
	if index >= range then
		SetValue(index-range, 1, nil)
		SetValue(index-range, 2, nil)

		SetValue(index-range+1, 1, H(index-range+1))
		SetValue(index-range+1, 2, L(index-range+1))
		SetRangeValue(1, index-range+2, index-1, nil)
		SetRangeValue(2, index-range+2, index-1, nil)

		--[[
		for i = index-range+2, index-1 do
			SetValue(i, 1, nil)
			SetValue(i, 2, nil)
		end
		--]]

		return H(index), L(index)
	else
		return nil, nil
	end
end
```

7.2.9 Settings Table

Table **Settings** contains settings of an indicator, in the script it is taken as global.

List of predefined fields with examples:

- **STRING Name** – line with name of an indicator;

```
Settings.Name = "Two MA"
```

- **STRING line[n].Name** – line with name of a line with number N. Indexes of lines start with 1;

```
Settings.line[1].Name = "First MA"
Settings.line[2].Name = "Second MA"
```

- **NUMBER line[n].Type** – type of displaying of a line. It is set using predefined constants: TYPE_LINE, TYPE_DASH, TYPE_DOT, TYPE_HISTOGRAM, TYPE_TRIANGLE_UP, TYPE_TRIANGLE_DOWN.

```
Settings.line[1].Type = LINE --lines
Settings.line[1].Type = DASH -- dashes
Settings.line[1].Type = DOT – dots
```

- **NUMBER line[n].Width** – line thickness;

```
Settings.line[1].Width = 5
```

- **NUMBER line[n].Color** – line colour. Result of execution of function RGB;

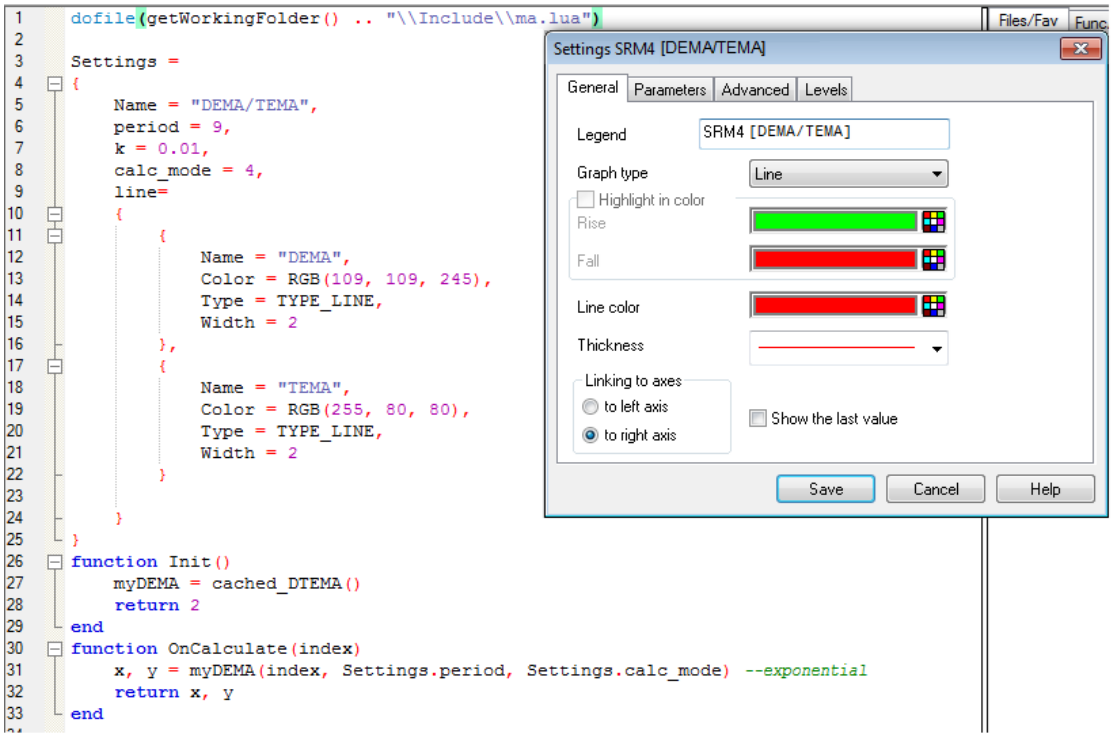
```
Settings.line[1].Color = RGB(255, 0, 0)
Settings.line[2].Color = RGB(0, 255, 0)
```

Fields in table **Settings** are displayed in settings dialogue in User settings section.

Types of user parameters: numbers and lines.

Fields not defined in the script will be initialized with default values.

7.2.10 Example of Settings Dialogue Linked to Setting Table



Changing setting in the dialogue leads to changes of the values of Settings table fields on working Lua computer without source code changes.

7.2.11 List of Functions Available from Script

- **getWorkingFolder** – returns the path on which info.exe file executing the script is located;
- **getScriptPath** – returns the path on which the run script is located;
- **getNumberOf** – returns the number of records in TableName table;
- **getItem** – returns the Lua table containing informations on data from a line with number Index from a table with name “TableName”;
- **getParamEx** – receives the values of all exchange data parameters from the Quoted table;
- **message** – displays the values QUIK terminal;
- **isConnected** – defines the state of workstation’s connection to server;
- **getTradeDate** – received the data of a trading session;
- **getInfoParam** – allows receiving parameters for information window (Connection / Information window);
- **getClassSecurities** – receives a list of securities codes for the list of classes set by codes list;
- **getClassInfo** – receives information on a class;
- **getClassesList** – receives a list of classes codes received from server during the session;
- **getSecurityInfo** – receives information on a security;
- **getQuoteLevel2** – receives the Level II quotes table for selected class and security;
- **getMoney** – receives information on cash limits;
- **getDepo** – receives information on securities limits;

- **sendTransaction** – function for working with orders;
- **SearchItems** – allows fast selecting of elements from the terminal storage and returns a table with indexes of elements that meet the search consitions;
- **getPortfolioInfo** – receives the values of parameters of Client portfolio table;
- **getBuySellInfo** – receives the values of parameters of Buy / Sell table;
- **getPortfolioInfoEx** – receives the values of parameters of Client portfolio table accounting the limit kind;
- **getBuySellInfoEx** – receives the values of parameters of Buy / Sell table accounting the limit kind;
- **getOrderByNumber** – returns the Lua table, containing description of parameters of the Orders table and index of an order to the terminal's storage;
- **RGB** – converts components RGB (red, green, blue) into a single number;
- **AddLabel** – adds label with specified parameters;
- **DelLabel** – deletes label with specified parameters;
- **DelAllLabels** –deletes all labels on chart with the specified graph;
- **GetLabelParams** – gets label parameters;
- **SetLabelParams** – sets parameters for label with the specified identifier;
- **SetValue** – sets a specified value on a selected line for a certain indicator's candlestick;
- **GetValue** – gets a value set on a selected line for a specified indicator's candlestick;
- **SetRangeValue** – sets a specified value on a selected line for a certain indices' interval of the indicator's candlesticks.

7.2.12 Loading and Saving of an Indicator Configuration to File

When selecting the menu item **Settings / Save configuration to file** all values from **Settings** table are saved in .wnd file.

When loading configuration from a file **qchart** module receives a list of indicators from **qlua** module and creates an indicator on its name automatically.

If the loaded indicator in not in the list (for example the file is deleted or the value of Settings. Name filed is changed) the indicator is not displayed. To avoid disappearing of indicators when changing settings in Lua code the chart displays its legend and the dialogue of settings editing is available.

8. Appendices

8.1 Appendix 1. Example of a Lua Script

This application provides an example of a Lua script that creates a table in a QUIK workstation.

Table_object.lua:

```
dofile (getScriptPath() .. "\\quik_table_wrapper.lua")
dofile (getScriptPath() .. "\\ntime.lua")
stopped = false
function format1(data)
	return string.format("0x%08X", data)
end

function format2(data)
	return string.format("%06d", data)
end


function OnStop(s)
	stopped = true
end

function main()
	-- turning ‘sticks’ in the table's heading
	local palochki = {"-","\\", "|", "/"}
	-- create a QTable instance
	t = QTable.new()
	if not t then
		message("error!", 3)
		return
	else
		message("table with id = " ..t.t_id .. " created", 1)
	end

	-- add two columns with formatting functions
	-- the first column is for hex values, the second is for integers
	t:AddColumn("test1", QTABLE_INT_TYPE, 10,format1)
	t:AddColumn("test2", QTABLE_INT_TYPE, 10, format2)
	-- add columns without formatting
	t:AddColumn("test3", QTABLE_CACHED_STRING_TYPE, 50)
	t:AddColumn("test4", QTABLE_TIME_TYPE, 50)
	t:AddColumn("test5", QTABLE_CACHED_STRING_TYPE, 50)

	t:SetCaption("Test")
	t:Show()
	i=1
	-- loop until the user stops the script from the control dialogue
	while not stopped do
		-- display the table again if it was closed
		-- all previous data will be purged
		if t:IsClosed() then
```

```
                    t:Show()
                end
                -- turn the 'stick' by 45 degrees on each iteration
                t:SetCaption("QLUA TABLE TEST " .. palochki[i%4 +1])
                -- this method will add a new row to the table and return its number
                local row = t:AddLine()
                t:SetValue(row, "test1", row, i)
                t:SetValue(row, "test2", row, i)

                -- insert the current table header into the cell
                -- the column has the string type, so the last parameter is ignored
                SetCell(t.t_id, row, 3, GetWindowCaption(t.t_id))

                _date = os.date("*t")
                -- fill out the 4th column with time data (a number in the format
<HHMMSS>)
                -- the function that returns a string presentation of time is defined
in ntime.lua
                -- The NiceTime function returns a string
                SetCell(t.t_id, row, 4,
                NiceTime(_date) .. string.format(" (%02d:%02d:%02d)", _date.hour,
_date.min, _date.sec),
                _date.hour*10000+_date.min*100 +_date.sec)
                -- the fifth column has the string type, and it is filled out with the
results of the NiceTime function
                -- the function's source code is taken from the Conky Lua widget for
Ubuntu
                SetCell(t.t_id, row, 5, NiceTime(_date))
                sleep(1000)
                i=i+1
        end
        message("finished")
end
```

Quik_table_wrapper.lua:

```
-- Reloading the message function with an optional second parameter
old_message = message
function message(v, i)
        old_message(tostring(v), i or 1)
end


QTable ={}
QTable.__index = QTable
```

```
-- Create and initialize a QTable instance
function QTable.new()
	local t_id = AllocTable()
	if t_id ~= nil then
		q_table = {}
		setmetatable(q_table, QTable)
		q_table.t_id=t_id
		q_table.caption = ""
		q_table.created = false
		q_table.curr_col=0
		-- a table with column parameters description
		q_table.columns={}
		return q_table
	else
		return nil
	end
end

function QTable:Show()
	-- display a window with the created table in the terminal
	CreateWindow(self.t_id)
	if self.caption ~="" then
		-- set the window's caption
		SetWindowCaption(self.t_id, self.caption)
	end
	self.created = true
end
function QTable:IsClosed()
	-- if the table window is closed, 'true' is returned
	return IsWindowClosed(self.t_id)
end

function QTable:delete()
	-- delete the table
	DestroyTable(self.t_id)
end

function QTable:GetCaption()
	if IsWindowClosed(self.t_id) then
		return self.caption
	else
		-- returns a string that contains the table caption
		return GetWindowCaption(self.t_id)
	end
end
```

```
-- Set the table caption
function QTable:SetCaption(s)
	self.caption = s
	if not IsWindowClosed(self.t_id) then
		res = SetWindowCaption(self.t_id, tostring(s))
	end
end

-- Add the description of the <name> column of the <c_type> type into the table
-- <ff> - the function that format data for display
function QTable:AddColumn(name, c_type, width, ff )
	local col_desc={}
	self.curr_col=self.curr_col+1
	col_desc.c_type = c_type
	col_desc.format_function = ff
	col_desc.id = self.curr_col
	self.columns[name] = col_desc
	-- <name> is often used as a table title
	AddColumn(self.t_id, self.curr_col, name, true, c_type, width)
end

function QTable:Clear()
	-- clear the table
	Clear(self.t_id)
end

-- set values in the cell
function QTable:SetValue(row, col_name, data)
	local col_ind = self.columns[col_name].id or nil
	if col_ind == nil then
		return false
	end
	-- if the formatting function is set for this column, it will be used
	local ff = self.columns[col_name].format_function

	if type(ff) == "function" then
		-- the result of the formatting function is used
		-- a string representation
		SetCell(self.t_id, row, col_ind, ff(data), data)
		return true
	else
		SetCell(self.t_id, row, col_ind, tostring(data), data)
	end
end

function QTable:AddLine()
```

```
	-- adds an empty line to the end of the table and returns its number
	return InsertRow(self.t_id, -1)
end

function QTable:GetSize()
	-- returns the table size
	return GetTableSize(self.t_id)
end

-- Retrieve data from a cell by row and column numbers
function QTable:GetValue(row, name)
	local t={}
	local col_ind = self.columns[name].id
	if col_ind == nil then
		return nil
	end
	t = GetCell(self.t_id, row, col_ind)
	return t
end

-- Set window's coordinates
function QTable:SetPosition(x, y, dx, dy)
	return SetWindowPos(self.t_id, x, y, dx, dy)
end

-- This function returns the window's coordinates
function QTable:GetPosition()
	top, left, bottom, right = GetWindowRect(self.t_id)
	return top, left, right-left, bottom-top
end
```

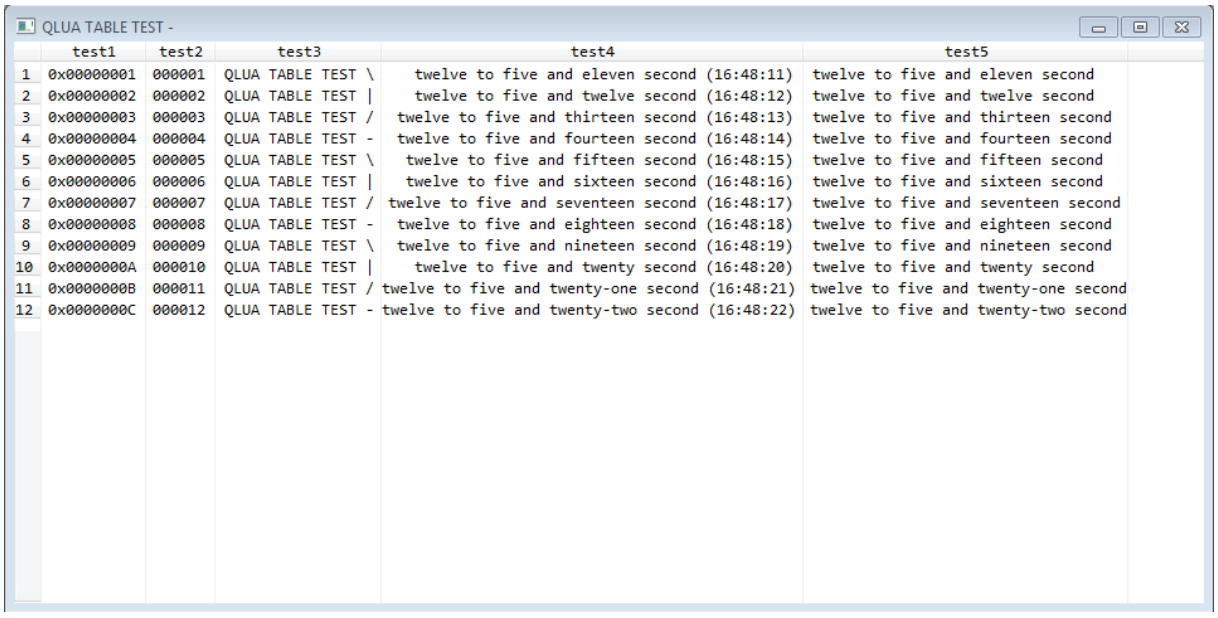
Ntime.lua:

```
words = {"one ", "two ", "three ", "four ", "five ", "six ", "seven ", "eight ",
"nine "}
levels = {"thousand ", "million ", "billion ", "trillion ", "quadrillion ",
"quintillion ", "sextillion ", "septillion ", "octillion ", [0] = ""}
iwords = {"ten ", "twenty ", "thirty ", "forty ", "fifty ", "sixty ", "seventy ",
"eighty ", "ninety "}
twords = {"eleven ", "twelve ", "thirteen ", "fourteen ", "fifteen ", "sixteen ",
"seventeen ", "eighteen ", "nineteen "}

function digits(n)
	local i, ret = -1
	return function()
	i, ret = i + 1, n % 10
```

```
        if n > 0 then
        n = math.floor(n / 10)
        return i, ret
        end
        end
end

level = false
function getname(pos, dig)
        level = level or pos % 3 == 0
        if(dig == 0) then return "" end
        local name = (pos % 3 == 1 and iwords[dig] or words[dig]) .. (pos % 3 == 2 and
"hundred " or "")
        if(level) then name, level = name .. levels[math.floor(pos / 3)], false end
        return name
end

function numberToWord(number)
        if(number == 0) then return "zero" end
        vword = ""
        for i, v in digits(number) do
        vword = getname(i, v) .. vword
        end

        for i, v in ipairs(words) do
        vword = vword:gsub("ty " .. v, "ty-" .. v)
        vword = vword:gsub("ten " .. v, twords[i])
        end
        return vword
end

function _Time(t)
        hour = t.hour
        minute = t.min
        hour = hour % 12
        if(hour == 0) then
        hour, nextHourWord = 12, "one "
        else
        nextHourWord = numberToWord(hour+1)
        end
        hourWord = numberToWord(hour)
        if(minute == 0 ) then
        return hourWord .. "o'clock"
        elseif(minute == 30) then
        return "half past " .. hourWord
        elseif(minute == 15) then
```

```
        return "a quarter past " .. hourWord
        elseif(minute == 45) then
        return "a quarter to " .. nextHourWord
        else
        if(minute < 30) then
        return numberToWord(minute) .. "past " .. hourWord
        else
        return numberToWord(60-minute) .. "to " .. nextHourWord
        end
        end
end

function _Seconds(s)
        return numberToWord(s)
end

function NiceTime(t)
        return _Time(t) .."and ".. _Seconds(t.sec) .. "second"
end
```

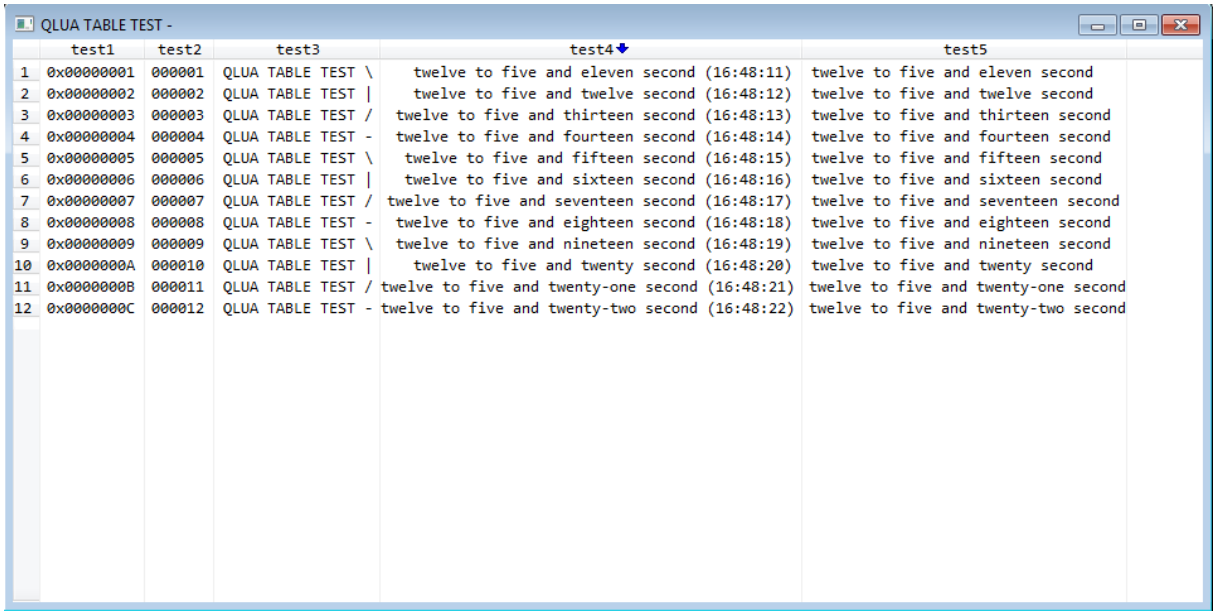
This script creates the following table in a QUIK workstation:



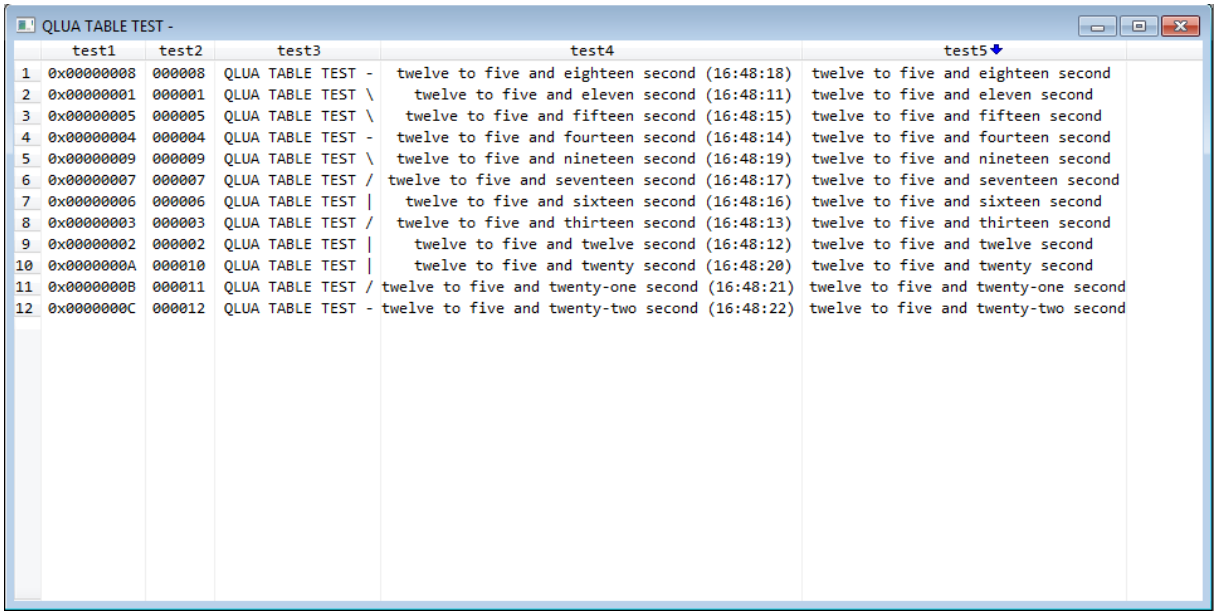
	test1	test2	test3	test4	test5
1	0x00000001	000001	QLUA TABLE TEST \	twelve to five and eleven second (16:48:11)	twelve to five and eleven second
2	0x00000002	000002	QLUA TABLE TEST |	twelve to five and twelve second (16:48:12)	twelve to five and twelve second
3	0x00000003	000003	QLUA TABLE TEST /	twelve to five and thirteen second (16:48:13)	twelve to five and thirteen second
4	0x00000004	000004	QLUA TABLE TEST -	twelve to five and fourteen second (16:48:14)	twelve to five and fourteen second
5	0x00000005	000005	QLUA TABLE TEST \	twelve to five and fifteen second (16:48:15)	twelve to five and fifteen second
6	0x00000006	000006	QLUA TABLE TEST |	twelve to five and sixteen second (16:48:16)	twelve to five and sixteen second
7	0x00000007	000007	QLUA TABLE TEST /	twelve to five and seventeen second (16:48:17)	twelve to five and seventeen second
8	0x00000008	000008	QLUA TABLE TEST -	twelve to five and eighteen second (16:48:18)	twelve to five and eighteen second
9	0x00000009	000009	QLUA TABLE TEST \	twelve to five and nineteen second (16:48:19)	twelve to five and nineteen second
10	0x0000000A	000010	QLUA TABLE TEST |	twelve to five and twenty second (16:48:20)	twelve to five and twenty second
11	0x0000000B	000011	QLUA TABLE TEST /	twelve to five and twenty-one second (16:48:21)	twelve to five and twenty-one second
12	0x0000000C	000012	QLUA TABLE TEST -	twelve to five and twenty-two second (16:48:22)	twelve to five and twenty-two second

8.2 Appendix 2. Examples of sorting in tables

Examples of sorting in a table's column that contains data of numeric (‘test4’) and string (‘test5’) types:



	test1	test2	test3	test4	test5
1	0x00000001	000001	QLUA TABLE TEST \	twelve to five and eleven second (16:48:11)	twelve to five and eleven second
2	0x00000002	000002	QLUA TABLE TEST |	twelve to five and twelve second (16:48:12)	twelve to five and twelve second
3	0x00000003	000003	QLUA TABLE TEST /	twelve to five and thirteen second (16:48:13)	twelve to five and thirteen second
4	0x00000004	000004	QLUA TABLE TEST -	twelve to five and fourteen second (16:48:14)	twelve to five and fourteen second
5	0x00000005	000005	QLUA TABLE TEST \	twelve to five and fifteen second (16:48:15)	twelve to five and fifteen second
6	0x00000006	000006	QLUA TABLE TEST |	twelve to five and sixteen second (16:48:16)	twelve to five and sixteen second
7	0x00000007	000007	QLUA TABLE TEST /	twelve to five and seventeen second (16:48:17)	twelve to five and seventeen second
8	0x00000008	000008	QLUA TABLE TEST -	twelve to five and eighteen second (16:48:18)	twelve to five and eighteen second
9	0x00000009	000009	QLUA TABLE TEST \	twelve to five and nineteen second (16:48:19)	twelve to five and nineteen second
10	0x0000000A	000010	QLUA TABLE TEST |	twelve to five and twenty second (16:48:20)	twelve to five and twenty second
11	0x0000000B	000011	QLUA TABLE TEST /	twelve to five and twenty-one second (16:48:21)	twelve to five and twenty-one second
12	0x0000000C	000012	QLUA TABLE TEST -	twelve to five and twenty-two second (16:48:22)	twelve to five and twenty-two second



	test1	test2	test3	test4	test5
1	0x00000008	000008	QLUA TABLE TEST -	twelve to five and eighteen second (16:48:18)	twelve to five and eighteen second
2	0x00000001	000001	QLUA TABLE TEST \	twelve to five and eleven second (16:48:11)	twelve to five and eleven second
3	0x00000005	000005	QLUA TABLE TEST \	twelve to five and fifteen second (16:48:15)	twelve to five and fifteen second
4	0x00000004	000004	QLUA TABLE TEST -	twelve to five and fourteen second (16:48:14)	twelve to five and fourteen second
5	0x00000009	000009	QLUA TABLE TEST \	twelve to five and nineteen second (16:48:19)	twelve to five and nineteen second
6	0x00000007	000007	QLUA TABLE TEST /	twelve to five and seventeen second (16:48:17)	twelve to five and seventeen second
7	0x00000006	000006	QLUA TABLE TEST |	twelve to five and sixteen second (16:48:16)	twelve to five and sixteen second
8	0x00000003	000003	QLUA TABLE TEST /	twelve to five and thirteen second (16:48:13)	twelve to five and thirteen second
9	0x00000002	000002	QLUA TABLE TEST |	twelve to five and twelve second (16:48:12)	twelve to five and twelve second
10	0x0000000A	000010	QLUA TABLE TEST |	twelve to five and twenty second (16:48:20)	twelve to five and twenty second
11	0x0000000B	000011	QLUA TABLE TEST /	twelve to five and twenty-one second (16:48:21)	twelve to five and twenty-one second
12	0x0000000C	000012	QLUA TABLE TEST -	twelve to five and twenty-two second (16:48:22)	twelve to five and twenty-two second

8.3 Appendix 3. Examples of Processing Table Events

Examples of processing mouse and keyboard events

```
stopped = false
t_id = nil

old_message = message
local fmt = string.format
function message(v, t)
	t= t or 1
	old_message(tostring(v), t)
end

function OnStop(s)
```

```
	stopped = true
	if t_id~= nil then
		DestroyTable(t_id)
	end
end
event_table = {
		[QTABLE_LBUTTONDOWN] ='Left mouse button is clicked',
		[QTABLE_RBUTTONDOWN] ='Right mouse button is clicked',
		[QTABLE_LBUTTONDBLCLK] = 'Left mouse button is double-clicked',
		[QTABLE_RBUTTONDBLCLK] = 'Right mouse button is double-clicked',
		[QTABLE_SELCHANGED] ='Row is changed',
		[QTABLE_CHAR] = "Character key",
		[QTABLE_VKEY] = "Some other key",
		[QTABLE_CONTEXTMENU] = "Shortcut menu",
		[QTABLE_MBUTTONDOWN] = "Mouse wheel is clicked",
		[QTABLE_MBUTTONDBLCLK] = "Mouse wheel is double-clicked",
		[QTABLE_LBUTTONUP] ='Left mouse button is released',
		[QTABLE_RBUTTONUP] ='Right mouse button is released',
		[QTABLE_CLOSE] = "Table is closed"
		}
function event_callback_str(t_id, msg, par1, par2)
	local str = fmt("%s, par1 = %d, par2 = %d", event_table[msg], par1, par2)
	SetWindowCaption(t_id, str)
	message(str)
end

local p_row = -1
local p_col = -1
function event_callback_color(t_id, msg, par1, par2)
	if par1==3 and par2 == 1 then
		os.exit()
	end
	if msg == QTABLE_LBUTTONDOWN then
		if p_col ~= -1 and p_col ~= -1 then
			SetColor(t_id, p_row, p_col, QTABLE_DEFAULT_COLOR,
QTABLE_DEFAULT_COLOR, QTABLE_DEFAULT_COLOR, QTABLE_DEFAULT_COLOR)
		end
		SetColor(t_id, par1, par2, RGB(240, 128, 128), QTABLE_DEFAULT_COLOR,
QTABLE_DEFAULT_COLOR, QTABLE_DEFAULT_COLOR)
		p_row = par1
		p_col = par2
	end
end


function main()
```

```
	data = {
			{"1", 2, 20130530},
			{"4", 5, 20130529},
			{"7", 8, 20130528}
			}

	t_id = AllocTable()
	message (t_id)
	AddColumn(t_id, 1, "string", true, QTABLE_CACHED_STRING_TYPE, 10)
	AddColumn(t_id, 2, "number", true, QTABLE_INT_TYPE, 10)
	AddColumn(t_id, 3, "date", true, QTABLE_DATE_TYPE, 10)
	CreateWindow(t_id)

	for _, v in pairs(data) do
		row = InsertRow(t_id, -1)
		SetCell(t_id, row, 1, v[1])
		SetCell(t_id, row, 2, string.format("value = %d",v[2]), v[2])
		SetCell(t_id, row, 3, string.format("%04d - %02d - %02d",v[3]/10000,
(v[3]%10000)/100, v[3]%100), v[3])
	end
	SetWindowCaption(t_id, "EXAMPLE")
	SetTableNotificationCallback(t_id, event_callback_str)
	sleep(5000)
	SetTableNotificationCallback(t_id, event_callback_color)
	SetCell(t_id, 3, 1, "DO NOT CLICK ME")
	SetTableNotificationCallback(t_id, dummy)

	while not stopped do
		sleep(100)
	end
end
```

Example of the 'Tic tac toe' game's implementation

```
--[[ TIC-TAC-TOE
by Evan Hahn (http://evanhahn.com/how-to-code-tic-tac-toe-and-a-lua-implementation/
--]]

---------------------------------------------
-- Configuration (change this if you wish!) --
---------------------------------------------
t_id=nil --grid
-- Are they playable by human or computer-controlled?
PLAYER_1_HUMAN = true
PLAYER_2_HUMAN = false
```

```
-- Board size
BOARD_RANK = 3      -- The board will be this in both dimensions.

-- Display stuff
PLAYER_1 = "[x]"    -- Player 1 is represented by this. Player 1 goes first.
PLAYER_2 = "[o]"    -- Player 2 is represented by this.
EMPTY_SPACE = "[ ]" -- An empty space is displayed like this.
DISPLAY_HORIZONTAL_SEPARATOR = "-"      -- Horizontal lines look like this.
DISPLAY_VERTICAL_SEPARATOR = " | "      -- Vertical lines look like this


--[[ ########################################################################
     ####   Don't mess with things below here unless you are brave  ####
     ######################################################################## --]]

------------------------
-- More configuration --
------------------------

MAX_BOARD_RANK = 100-- Won't run above this number. Prevents crashes.

-------------------------------------------------------
-- Don't run if the board is larger than the maximum --
-------------------------------------------------------

if BOARD_RANK > MAX_BOARD_RANK then os.exit(0) end

-----------------------------
-- Create board (2D table) --
-----------------------------

space = {}
for i = 0, (BOARD_RANK - 1) do
       space[i] = {}
       for j = 0, (BOARD_RANK - 1) do
              space[i][j] = nil   -- start each space with nil
       end
end

---------------------
-- Board functions --
---------------------

-- get the piece at a given spot
function getPiece(x, y)
```

```
	return space[x][y]
end

-- get the piece at a given spot; if nil, return " "
-- this is useful for output.
function getPieceNoNil(x, y)
	if getPiece(x, y) ~= nil then
		return getPiece(x, y)
	else
		return EMPTY_SPACE
	end
end

-- is that space empty?
function isEmpty(x, y)
	if getPiece(x, y) == nil then
		return true
	else
		return false
	end
end

-- place a piece there, but make sure nothing is there already.
-- if you can't play there, return false.
function placePiece(x, y, piece)
	if isEmpty(x, y) == true then
		space[x][y] = piece
		return true
	else
		return false
	end
end

-- is the game over?
function isGameOver()
	if checkWin() == false then	-- if there is no win...
		for i = 0, (BOARD_RANK - 1) do	-- is the board empty?
			for j = 0, (BOARD_RANK - 1) do
				if isEmpty(i, j) == true then return false end
			end
		end
		return true
	else	-- there is a win; the game is over
		return true
	end
end
```

```
-- create a string made up of a certain number of smaller strings
-- this is useful for the display.
function repeatString(to_repeat, amount)
	if amount <= 0 then return "" end
	local to_return = ""
	for i = 1, amount do
		to_return = to_return .. to_repeat
	end
	return to_return
end

-- display the board.
-- this uses the configuration file pretty much entirely.
function displayBoard()
	for i = (BOARD_RANK - 1), 0, -1 do
		for j = 0, (BOARD_RANK - 1) do    -- generate that row
			local piece = getPieceNoNil(j, i)
			SetCell(t_id, i+1, j+1, piece)
		end
	end
end

------------------------------------------------
-- Create regions (I admit this is a bit ugly) --
------------------------------------------------

-- declare region and a number to increment
region = {}
region_number = 0

-- vertical
for i = 0, (BOARD_RANK - 1) do
	region[region_number] = {}
	for j = 0, (BOARD_RANK - 1) do
		region[region_number][j] = {}
		region[region_number][j]["x"] = i
		region[region_number][j]["y"] = j
	end
	region_number = region_number + 1
end

-- horizontal
for i = 0, (BOARD_RANK - 1) do
	region[region_number] = {}
	for j = 0, (BOARD_RANK - 1) do
```

```
                region[region_number][j] = {}
                region[region_number][j]["x"] = j
                region[region_number][j]["y"] = i
        end
        region_number = region_number + 1
end

-- diagonal, bottom-left to top-right
region[region_number] = {}
for i = 0, (BOARD_RANK - 1) do
        region[region_number][i] = {}
        region[region_number][i]["x"] = i
        region[region_number][i]["y"] = i
end
region_number = region_number + 1

-- diagonal, top-left to bottom-right
region[region_number] = {}
for i = (BOARD_RANK - 1), 0, -1 do
        region[region_number][i] = {}
        region[region_number][i]["x"] = BOARD_RANK - i - 1
        region[region_number][i]["y"] = i
end
region_number = region_number + 1

-----------------------
-- Region functions --
-----------------------

-- get a region
function getRegion(number)
        return region[number]
end

-- check for a win in a particular region.
-- returns a number representation of the region. occurrences of player 1
-- add 1, occurrences of player 2 subtract 1. so if there are two X pieces,
-- it will return 2. one O will return -1.
function checkWinInRegion(number)
        local to_return = 0
        for i, v in pairs(getRegion(number)) do
                local piece = getPiece(v["x"], v["y"])
                if piece == PLAYER_1 then to_return = to_return + 1 end
                if piece == PLAYER_2 then to_return = to_return - 1 end
        end
        return to_return
```

```
end

-- check for a win in every region.
-- returns false if no winner.
-- returns the winner if there is one.
function checkWin()
	for i in pairs(region) do
		local win = checkWinInRegion(i)
		if math.abs(win) == BOARD_RANK then
			if win == math.abs(win) then
				return PLAYER_1
			else
				return PLAYER_2
			end
		end
	end
	return false
end

------------------
-- UI Functions --
------------------

-- human play
function humanPlay(piece)
	message("Human turn")
	displayBoard()
	local placed = false
	while placed == false do   -- loop until they play correctly
		sleep(100)
		if g_X ~= -1 and g_Y ~=-1 then
			local x = tonumber(g_Y)-1
			local y = tonumber(g_X)-1
			g_X = -1
			g_Y= -1
			message("clicked in " .. x .. " and " .. y)
			placed = placePiece(x, y, piece)
			if placed == false then
				message("I'm afraid you can't play there!")
			end
		end
	end
	displayBoard()

end
```

```
-- AI play
function AIPlay(piece)

	-- am I negative or positive?
	local me = 0
	if piece == PLAYER_1 then me = 1 end
	if piece == PLAYER_2 then me = -1 end

	-- look for a region in which I can win
	for i in pairs(region) do
		local win = checkWinInRegion(i)
		if win == ((BOARD_RANK - 1) * me) then
			for j, v in pairs(getRegion(i)) do
				if isEmpty(v["x"], v["y"]) == true then
					placePiece(v["x"], v["y"], piece)
					return
				end
			end
		end
	end

	-- look for a region in which I can block
	for i in pairs(region) do
		local win = checkWinInRegion(i)
		if win == ((BOARD_RANK - 1) * (me * -1)) then
			for j, v in pairs(getRegion(i)) do
				if isEmpty(v["x"], v["y"]) == true then
					placePiece(v["x"], v["y"], piece)
					return
				end
			end
		end
	end

	-- play first empty space, if no better option
	for i = 0, (BOARD_RANK - 1) do
		for j = 0, (BOARD_RANK - 1) do
			if placePiece(i, j, piece) ~= false then return end
		end
	end

end
g_X=-1
g_Y=-1
function event_callback(t_id, msg, par1, par2)
	if msg == QTABLE_LBUTTONDOWN then
```

```
                g_X = par1
                g_Y = par2
        end
end

old_message = message
local fmt = string.format
function message(v, t)
        t= t or 1
        old_message(tostring(v), t)
end

function main()
        t_id = AllocTable()
        AddColumn(t_id, 1, "", true, QTABLE_CACHED_STRING_TYPE, 5)
        AddColumn(t_id, 2, "", true, QTABLE_CACHED_STRING_TYPE, 5)
        AddColumn(t_id, 3, "", true, QTABLE_CACHED_STRING_TYPE, 5)
        CreateWindow(t_id)
        for i=1, 3 do
                row = InsertRow(t_id, -1)
                SetCell(t_id, row, 1, "[ ]")
                SetCell(t_id, row, 2, "[ ]")
                SetCell(t_id, row, 3, "[ ]")
        end
        SetTableNotificationCallback(t_id, event_callback)
        message("Welcome to Tic-Tac-Toe!")

-- play the game until someone wins
        while true do
                sleep(100)
        -- break if the game is won
                if isGameOver() == true then
                 break
                end
        -- player 1
                if PLAYER_1_HUMAN == true then
                        humanPlay(PLAYER_1)
                else
                        AIPlay(PLAYER_1)
                end

                if isGameOver() == true then
                        break
                end

                if PLAYER_2_HUMAN == true then
```

```
                    humanPlay(PLAYER_2)
                else
                    AIPlay(PLAYER_2)
                end
        end

-- show the final board
        displayBoard()

-- write who won, or if there is a tie
        win = checkWin()
        if win == false then
                message("Tie game!\n")
        else
                message(win)
                message(" wins!\n")
        end
end
```

8.4 Appendix 4. Examples of Using the ‘params’ Parameter in the ‘SearchItems’ Function

Example 1

If the last parameter in the **SearchItems** function is not set, an anonymous trade is passed to the **fn** callback function as a Lua table:

```
function fn(t)
                if t.qty == 103 then
                return true
        else
                return false
        end
end
t1 = SearchItems("all_trades", 0, getNumberOf(“all_trades”)-1, fn)
```

Example 2

If the list of fields is set, the parameters are passed to the **fn** function in the same order in which they are listed in the list of parameters. In this example, par1 contains the qty field, par2 contains class_code, par3 contains sec_code.

If the element doesn't have these fields, ‘nil’ is passed as a parameter.

```
function fn(par1, par2, par3)
              if par1 == 103 and par2 == “SPBFUT” and par3 == “RIM3” then
              return true
       else
              return false
       end
end
t1 = SearchItems("all_trades", 0, getNumberOf(“all_trades”)-1, fn, “qty,class_code,
sec_code”)
```

Example 3

In this example, par1 will be equal nil, par2 – class_code, par3 – sec_code.

```
function fn(par1, par2, par3)
       if par1 == 103 and par2 == “SPBFUT” and par3 == “RIM3” then
              return true
       else
              return false
       end
end
t1 = SearchItems("all_trades", 0, getNumberOf(“all_trades”)-1, fn, “test,class_code,
sec_code”)
```

Example 4

Elements of embedded tables are separated by a period, for example:

```
function fn(par1, par2)
       if par1 == 17 and par2 == 5 then
              return true
       else
              return false
       end
end
t1 = SearchItems("all_trades", 0, getNumberOf(“all_trades”)-1, fn, “datetime.hour,
datetime.min”)
```