quik

Lua Interpreter

User's manual



Version 13.2

2026

ARQA TECHNOLOGIES



Contents

1. About 5
   1.1 Capabilities 5
   1.2 Getting started 5
   1.3 Working with the program 5
2. Workstation functions accessible from Lua scripts 7
   2.1 Service functions 7
   2.2 Callback functions 11
3. Functions for interactions between Lua scripts and QUIK workstation 21
   3.1 Functions for accessing rows in QUIK tables 21
   3.2 Functions for accessing available parameters lists 23
   3.3 Function for obtaining cash data 24
   3.4 Function for obtaining positions in instruments 25
   3.5 Function for obtaining futures limits information 26
   3.6 Function for obtaining futures positions information 26
   3.7 Function for obtaining instrument information 26
   3.8 Function for obtaining trading session date 27
   3.9 Function for obtaining Level II Quotes table for specified class and instrument 27
   3.10 Functions for working with charts 29
   3.11 Functions for working with orders 33
   3.12 Function for obtaining values from Quotes table 35
   3.13 Functions for obtaining parameters of Client Portfolio table 38
   3.14 Functions for obtaining parameters from Buy/Sell table 44
   3.15 Functions for working with QUIK workstation tables 46
   3.16 Function for working with labels 54
   3.17 Functions for ordering Level II Quotes table 56
   3.18 Functions for ordering Quotes table parameters 57
   3.19 Functions for obtaining information of the unified cash position 58
4. Data structures 60
   4.1 Classes 60

4.2 Firms 61
4.3 Time and Sales 61
4.4 Trades 62
4.5 Orders 70
4.6 Participant's positions on trading accounts 75
4.7 Participant's positions in instruments 76
4.8 Stop orders 76
4.9 Client account limits 78
4.10 Client account positions (Futures) 79
4.11 Cash positions 81
4.12 Removing cash position 81
4.13 Deleting instrument position 82
4.14 Deleting futures limit 82
4.15 Positions in instruments 83
4.16 Participant's cash positions 84
4.17 Negotiated deal orders 85
4.18 Trades for execution 88
4.19 Trading accounts 92
4.20 Reports on trades for execution 93
4.21 Instruments 93
4.22 Chart candlesticks 95
4.23 Date and time format used in tables 96
4.24 Transactions 96
4.25 Asset commitments and claims 100
4.26 Currency: commitments and demands on assets 100

5. Description of bit flags 101
5.1 Flags for the Orders, Negdeal Orders tables 101
5.2 Flags for Trades, Trades for execution tables 102
5.3 Flags for Order reports for NDM trades table 102
5.4 Flags for Time and Sales table 102
5.5 Flags for Stop Orders table 103

5.6 Additional flags for Stop Orders table 103
5.7 Flags for Transactions table 104
6. Functions for working with bit masks in data structures 104
6.1 bit.tohex 104
6.2 bit.bnot 104
6.3 bit.band 104
6.4 bit.bor 104
6.5 bit.bxor 105
6.6 bit.test 105
7. Indicators of the technical analysis 105
7.1 General information 105
7.2 Functions and global variables of script's indicator 106
8. Thred–safe functions for working with Lua tables 117
9. Appendixes 119
9.1 Appendix 1. Example of a Lua script 119
9.2 Appendix 2. Examples of sorting in tables 125
9.3 Appendix 3. Examples of processing table events 126
9.4 Appendix 4. Examples of using the 'params' parameter in the 'SearchItems' function 136

You may email your comments on this Manual 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.

> 1. **64–bit third party libraries need to be used.**
> 2. **Incorrect operation of the 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

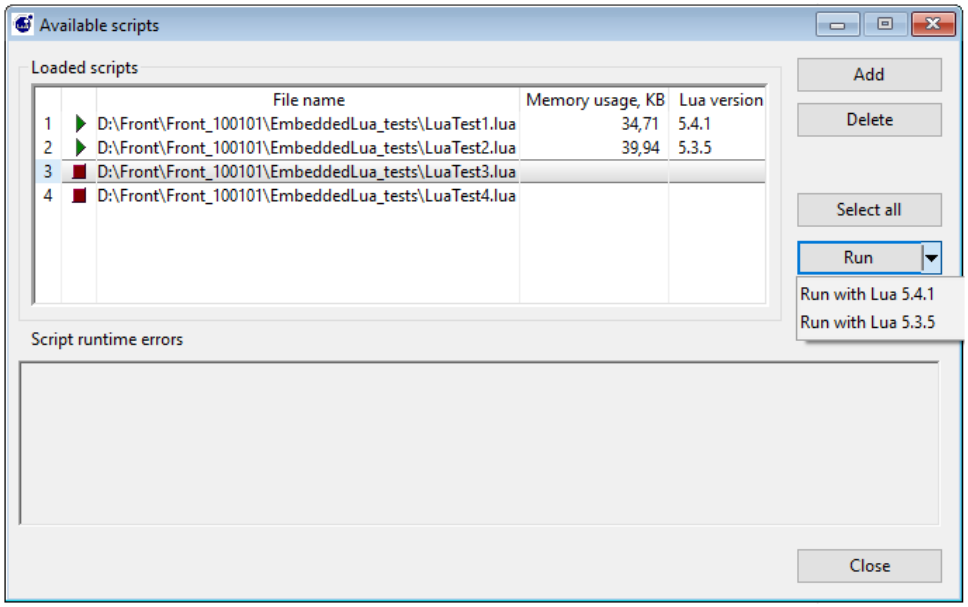
QLua functions are available from the program main menu. When the component is installed, new item **Services / Lua scripts** appears in the program menu.

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 **Run** to run loaded scripts, with the ability to choose the version of the Lua machine. Valid values:

  - Run with Lua 5.4.1;
  - Run with Lua 5.3.5.

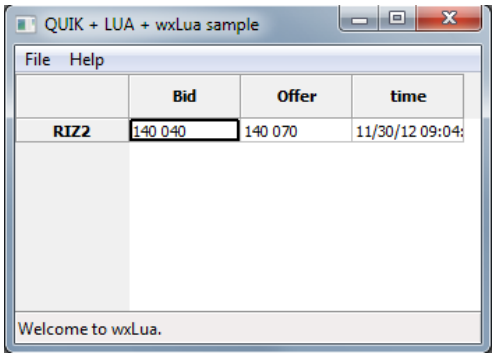
  The default value when clicking the **Run** button is Lua 5.4.1. The default value can be changed in the QUIK workstation settings in the **Lua scripts** section – **Lua version for user scripts**, the description is given in 2.10.7 sub– section Chapter 2 “Basic operating principles”.
- Click **Stop** to stop execution of a selected script.

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

> **The scripts are run automatically after restarting the terminal if they were running when the terminal was closed.**

> **The value of Memory usage, KB is calculated as a sum of the resulted values of functions lua_gc(LUA_GCCOUNT) and lua_gc(LUA_GCCOUNTB). In the Lua script memory usage can be determined by a standard LUA function collectgarbage("count").**

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 **System / About program / 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
NUMRECORDS	Number of records
LASTRECORD	Last record
LATERECORD	Delayed record
CONNECTION	Connection
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

Parameter value	Description
CONNECTIONTIME	Connection time
MESSAGESSENT	Messages sent
ALLSENT	Bytes sent
BYTESSENT	Bytes sent (only with data)
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+")
    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 the 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. Valid values and icons are: _ 1 (by default) – ; _ 2 – ; _ 3 – . Optional parameter

Example:

```
message("test message")
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
```

2.1.8 sysdate

The function returns the system date and time with an accuracy of microseconds.

Function call syntax:

TABLE os.sysdate()

The function returns a Lua table with parameters, which are described in 4.23.

> **Function uses system calls of the Windows operating system and the function values depend on the interval between the system timer interrupts.**

2.1.9 isDarkTheme

The function is used to determine which design theme of the interface is used: standard or dark.

Function call syntax:

BOOLEAN isDarkTheme()

The function returns **true**, if the dark theme is used, otherwise – **false**.

2.2 Callback functions

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

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

Trading data is only passed to the script, if parameters for this instrument are requested from the server (as a part of a smart request or manually though the **System/ Data request / Available instruments** dialogue) or if the corresponding Level II quotes table is opened.

> **Data can be requested from server by using QLua functions (see 3.17 about requesting a Level II Quotes table, and 3.18 about requesting the Quotes table's parameters).**

2.2.1 OnFirm

The QUIK workstation 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

The QUIK workstation calls this function when an anonymous trade is received.

Function call syntax:

OnAllTrade(TABLE alltrade)

Parameters:

Parameter	Type	Description
alltrade	TABLE	Table with parameters of an anonymous trade

2.2.3 OnTrade

This function is called by the QUIK workstation when a trade is received or when changing parameters of the existing trade.

Function call syntax:

OnTrade(TABLE trade)

Parameters:

Parameter	Type	Description
trade	TABLE	Trade parameters table

2.2.4 OnOrder

This function is called by the QUIK workstation 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 parameter table

2.2.5 OnAccountBalance

This function is called by the QUIK workstation 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	Participant's positions on trading accounts

2.2.6 OnFuturesLimitChange

This function is called by the QUIK workstation when it receives changes in derivatives market limits.

Function call syntax:

OnFuturesLimitChange(TABLE fut_limit)

Parameters:

Parameter	Type	Description
fut_limit	TABLE	Client account limits table

2.2.7 OnFuturesLimitDelete

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

Function call syntax:

OnFuturesLimitDelete(TABLE lim_del)

Parameters:

Parameter	Type	Description
lim_del	TABLE	Table with parameters of the futures limit being deleted

2.2.8 OnFuturesClientHolding

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

Function call syntax:

OnFuturesClientHolding(TABLE fut_pos)

Parameters:

Parameter	Type	Description
fut_pos	TABLE	Table of client account positions (futures)

2.2.9 OnMoneyLimit

This function is called by the QUIK workstation when it receives changes in a client's cash position.

Function call syntax:

OnMoneyLimit(TABLE mlimit)

Parameters:

Parameter	Type	Description
mlimit	TABLE	Cash positions

2.2.10 OnMoneyLimitDelete

This function is called by the QUIK workstation when a cash position is removed.

Function call syntax:

OnMoneyLimitDelete(TABLE mlimit_del)

Parameters:

Parameter	Type	Description
mlimit_del	TABLE	Table with parameters of cash position being deleted

2.2.11 OnDepoLimit

This function is called by the QUIK workstation when it receives changes in a position in instruments.

Function call syntax:

OnDepoLimit(TABLE dlimit)

Parameters:

Parameter	Type	Description
dlimit	TABLE	Positions in instruments

2.2.12 OnDepoLimitDelete

This function is called by the QUIK workstation when a client's position in instruments is removed.

Function call syntax:

OnDepoLimitDelete(TABLE dlimit_del)

Parameters:

Parameter	Type	Description
dlimit_del	TABLE	Table with parameters of position in instruments being deleted

2.2.13 OnAccountPosition

This function is called by the QUIK workstation when a participant’s cash position changes.

Function call syntax:

OnAccountPosition(TABLE acc_pos)

Parameters:

Parameter	Type	Description
acc_pos	TABLE	Participant's cash positions

2.2.14 OnNegDeal

This function is called by the QUIK workstation when receiving a negdeal order or when changing parameters of the existing one.

Function call syntax:

OnNegDeal(TABLE neg_deals)

Parameters:

Parameter	Type	Description
neg_deals	TABLE	Table with parameters of negotiated deal orders

2.2.15 OnNegTrade

This function is called by the QUIK workstation when it receives a trade for execution or when changing parameters of the existing one.

Function call syntax:

OnNegTrade(TABLE neg_trade)

Parameters:

Parameter	Type	Description
neg_trade	TABLE	Table of trades for execution

2.2.16 OnStopOrder

This function is called by the QUIK workstation 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	Table of stop order parameters

2.2.17 OnTransReply

The OnTransReply function is called by the QUIK workstation when it receives a response for a user transaction sent by means of any QUIK workstation plugin (including QLua). For

transactions sent by means of Trans2quik.dll, QPILE or dynamical transactions loading from file the function cannot be called.

Function call syntax:

OnTransReply(TABLE trans_reply)

Parameters:

Parameter	Type	Description
trans_reply	TABLE	Transactions

> **The simultaneously executed scripts receive the notifications of responses for their own transactions and for those which are placed by other scripts.**

2.2.18 OnParam

This function is called by the QUIK workstation when current parameters change.

OnParam (STRING class_code, STRING sec_code)

Parameters:

Parameter	Type	Description
class_code	STRING	Class code
sec_code	STRING	Instrument 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
			message("Bid " .. tbid.param_image)
		end
	end
end
```

2.2.19 OnQuote

This function is called by the QUIK workstation when it receives changes in **Level II Quotes** table.

OnQuote(STRING class_code, STRING sec_code)

Parameters:

Parameter	Type	Description
class_code	STRING	Class code
sec_code	STRING	Instrument 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
end
```

2.2.20 OnDisconnected

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

OnDisconnected()

2.2.21 OnConnected

This function is called by the QUIK workstation when upon establishing a connection to the QUIK server and receiving description at least of one class by terminal. If the terminal received a new class during the trading day the function is called again, at that the call parameter **flag** takes value **false**.

OnConnected(BOOLEAN flag)

2.2.22 OnCleanUp

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

OnCleanUp()

> **Change of session is understood as changed identifier of session when connecting to the QUIK server.**

2.2.23 OnClose

This function is called by the QUIK workstation in the following situations:

- Change of the QUIK server inside the trading session.
- Change of user who connects to the QUIK server inside the trading session.
- Change of session.

OnClose()

When performing several of the following conditions, the function OnCleanUp() is called by the QUIK terminal for each of them.

> **Uploading of qlua.dll file is understtod as getting disabled plygin QLua in dialog "Component and plugin versions" (see 1.8. of Section 1 of User's manual for QUIK).**

2.2.24 OnStop

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

[NUMBER time_out] OnStop(NUMBER flag)

The function returns the quantity of milliseconds that is available for a script to finish its operation. If the function does not return a number, timeout for script termination will be equal to 5 seconds. If the script does not finish operating in the allotted time returned in the OnStop function or in 5 seconds, the Lua Interpreter crash mode is initiated. If the script has not finished operating after 1 second, the Lua thread is forcibly stopped by the TerminateThread() function.

> **There may be failures in the operation of the QUIK terminal if stopped forcibly.**

After the running Lua script is stopped or deleted from the **Available scripts control** dialog the call parameter **flag** takes value 1. If the QUIK workstation is closed, it takes value 2.

Example:

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

```
end
```

2.2.25 OnInit

This function is called by the QUIK workstation 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 workstation 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.

main()

Example:

```
function main()
	while true do
		message(os.date()) --displays the current date and time once a second
		sleep(1000)
	end
end
```

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**.

TABLE getItem (STRING TableName, NUMBER Index)

If executed unsuccessfully, the function returns **nil**.

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

TABLE order NUMBER indx 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**.

NUMBER 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])

> **If an error occurs in fn function, SearchItems function stops working and returns "nil".**

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 one of the following values:

- true – current index is considered in the result;
- false – current index is not considered in the result;
- nil – the search stops, SearchItems function returns the table with indices found earlier including the current index.

- **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", and "SearchItems"

TableName	Table
firms	Firms
classes	Classes
securities	Instruments
trade_accounts	Trading accounts
client_codes	* Client codes
all_trades	Anonymous trades
account_positions	Participant's cash positions
orders	Orders
futures_client_holding	Client account positions (Futures)
futures_client_limits	Client account limits
money_limits	Cash positions
depo_limits	Positions in instruments
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	Participant's positions in instruments
account_balance	Participant's positions on trading accounts
ccp_holdings	Asset commitments and claims
rm_holdings	Currency: commitments and demands

* – 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 classe codes 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.

STRING getClassesList()

Example:

```
list = getClassesList()
```

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

```
OPTEXP,USDRUB,PSOPT,PSFUT,SPBFUT,
```

3.2.2 getClassInfo

Returns information about a class.

TABLE getClassInfo (STRING)

Returns a Lua table with a class description:

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

3.2.3 getClassSecurities

This function returns a list of instrument codes for a specified list of class codes. Instrument codes in the list are separated by a comma, ",". The character "," is always added to the end of a received string.

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 position information.

TABLE 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 instruments in buy orders

Parameter	Type	Description
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 positions of a specified type.

> TABLE getMoneyEx(STRING firmid, STRING client_code, STRNIG tag, STRING currcode, NUMBER limit_kind)

The function returns a Lua table with the **Cash positions** table parameters. If an error occurs, the function returns **nil**.

3.4 Function for obtaining positions in instruments

3.4.1 getDepo

Returns information about positions in instruments.

> TABLE getDepo (STRING client_code, STRING firmid, STRING sec_code, STRING trdaccid)

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 instruments balance
depo_limit_locked_buy	NUMBER	Number of lots of instruments locked for buying
depo_limit_locked	NUMBER	Number of locked lots of instruments
depo_limit_available	NUMBER	Quantity of available instruments
depo_current_limit	NUMBER	Current limit for instruments
depo_open_balance	NUMBER	Opening balance for instruments
depo_open_limit	NUMBER	Open limit for instruments

3.4.2 getDepoEx

Returns the information on positions in instruments of a specified type.

> TABLE getDepoEx(STRING firmid, STRING client_code, STRNIG sec_code, STRING trdaccid, NUMBER limit_kind)

The function returns a Lua table with the Positions in instruments table parameters. 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 firmid, STRING trdaccid, NUMBER limit_type, STRING currcode)

> **When it is necessary to obtain information on a futures limit without currency, curr_code value must be a blank row.**

The function returns a Lua table with the parameters of a Client account limits table. 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 firmid, STRING trdaccid, STRING sec_code, NUMBER type)

The function returns a Lua table with the parameters of the Client account positions table.
If an error occurs, the function returns **nil**.

3.7 Function for obtaining instrument information

3.7.1 getSecurityInfo

This function returns information on an instrument.

> TABLE getSecurityInfo (STRING class_code, STRING sec_code)

The function returns a Lua Instruments table.

3.8 Function for obtaining trading session date

3.8.1 getTradeDate

Returns the date of the current trading session.

TABLE getTradeDate()

The function returns a Lua table with the following parameters:

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

3.9 Function for obtaining Level II Quotes table for specified class and instrument

3.9.1 getQuoteLevel2

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

TABLE getQuoteLevel2 (STRING class_code, STRING sec_code)

The function returns a Lua table with the following parameters:

> **In the absence of bid and offer the function returns table without bid and offer parameters.**

Parameter	Type	Description
bid_count	STRING	Number of bids. In the absence of bid, 6 decimal places
offer_count	STRING	Number of offers. In the absence of offer, 6 decimal places
bid	TABLE	Bids (buying). In the absence of bid, a blank row is returned
offer	TABLE	Offers (selling). In the absence of offer, a blank row is returned

Each element of the **bid** and **offer** arrays contains the **price** and **quantity**:

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

Examples

There is no bid and offer:

```
{'bid_count': '0.000000',
'offer_count': '0.000000'
}
```

There is bid and offer:

```
{'bid_count': '3.000000',
'offer_count': '3.000000',
'bid': {{'price':'11.3', 'quantity':'10'},
        {'price':'11.4', 'quantity':'20'},
        {'price':'11.5', 'quantity':'30'},
       },
'offer': {
          {'price':'11.6', 'quantity':'10'},
          {'price':'11.7', 'quantity':'15'},
          {'price':'11.8', 'quantity':'18'}
       },
}
```

There is bid, but there is no offer:

```
{'bid_count': '3.000000',
'offer_count': '0.000000',
'bid': {{'price':'11.3', 'quantity':'10'},
        {'price':'11.4', 'quantity':'20'},
        {'price':'11.5', 'quantity':'30'},
       },
'offer': ''
}
```

There is offer, but there is no bid:

```
{'bid_count': '0.000000',
'offer_count': '3.000000',
'bid': '',
'offer': {
          {'price':'11.6', 'quantity':'10'},
          {'price':'11.7', 'quantity':'15'},
          {'price':'11.8', 'quantity':'18'}
       }
}
```

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 an instrument;
- **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, instrument code, interval or parameter is specified than the function returns "nil". At that **data_sourc** contains description of the error.

> **CreateDataSource function can be used only inside the main() and callback functions.**

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
INTERVAL_M6	6 minutes
INTERVAL_M10	10 minutes

Parameter	Value of interval
INTERVAL_M15	15 minutes
INTERVAL_M20	20 minutes
INTERVAL_M30	30 minutes
INTERVAL_H1	1 hour
INTERVAL_H2	2 hours
INTERVAL_H4	4 hours
INTERVAL_D1	1 day
INTERVAL_W1	1 week

Parameter	Value of interval
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
SetEmptyCallback	function	Allows a user to receive data from server without specifying the callback function

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 candlestick. 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.%03d",
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

The function is intended to send transactions to trading system.

Function call syntax:

STRING result sendTransaction(TABLE transaction)

Parameters:

- **result** – row containing an error text if it happened when processing a transaction.

- **transaction** – table with parameters of transaction.

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's manual, Chapter 6: Working with other programs / Transaction import).

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

Example of filling in fields of **transaction** table:

```
transaction = {
ACCOUNT="YY0070001234",
CLIENT_CODE="XXX",
TYPE="M",
TRANS_ID="7",
CLASSCODE="TQBR",
SECCODE="HYDR",
ACTION="NEW_ORDER",
OPERATION="B",
PRICE="0",
QUANTITY="15"
}
```

3.11.2 CalcBuySell

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

Function call syntax:

> TABLE 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** – instrument 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.

> **If is_market=true, then establish price=0, otherwise the maximum possible number of lots will be calculated in order at price of 'price' parameter.**

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 the **Quotes** table. This function can be used to obtain any value from the **Quotes** table for specified class and instrument codes. For details, see section 3.12.3.

Function call syntax:

TABLE 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. Valid values: _ 1 – DOUBLE; _ 2 – LONG; _ 3 – CHAR; _ 4 – enumerable type; _ 5 – time; _ 6 – date

Parameter	Type	Description
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 considers numeric and decimal separators. For enumerable types, their corresponding string values are returned
result	STRING	Result of executing operation. Valid values: _ 0 – error; _ 1 – parameter is found

3.12.2 getParamEx2

This function returns all exchange parameters from the **Quotes** table with a possibility to deny particular parameters requested by function ParamRequest, for details see section 3.12.3.
To deny receiving a parameter, use function CancelParamRequest.

Function call syntax:

TABLE getParamEx2 (STRING class_code, STRING sec_code, STRING param_name)

Parameters:

- **class_code** – class code;
- **sec_code** – instrument code;
- **param_name** – parameter code.

The function returns a Lua table with parameters analogical to those returned by function getParamEx (see 3.12.1).

3.12.3 Getting the values of Quotes table

To get the values of the Quotes table, use functions getParamEx() or getParamEx2(). Current values of parameters can be received if the data was requested. The methods to receive data depend on the options that are set in the general settings of the QUIK workstation (**System / Settings / General settings...** menu item, **Program / Receiving data / Quotes** sections):

- Manualy by turning on the **Selected classes...** option and specify the required parameters and instruments for a class;
- When the **smart data ordering** option is enabled and the **Quotes** table opened with the required parameters and instruments for a class;
- Automatically from Lua script using the ParamRequest or CreateDataSource functions, if the **smart data ordering** option in the QUIK workstation is enabled.

> **QUIK workstation automatically requests parameters required to correctly calculate the limits if the "smart data ordering" option is selected in the QUIK workstation.**

Getting service parameter identifiers

The set of parameters displayed in the **Quotes** table of the QUIK workstation may differ depending on the list of trading systems and trading modes provided by QUIK system server.

To access these parameters, you need to know their names (service identifiers), which can be obtained as follows:

- In the QUIK workstation, create the **Quotes** table with the required set of parameters. In the table, one row with the instrument is enough, since you only need to get a list of headers. In the table editing window, select the **Transpose** checkbox.
- Start MS Excel.
- In the context menu of the **Quotes** table, select the **Export to DDE Server**. In the opened dialog, specify the name of the Ms Excel workbook (as a rule, it is "Book1") and the sheet (Sheet1). It is also required to fill in the following fields with values:
  - Row – 1;
  - Column – 1;
  - The **With row headers** checkbox must be selected;
  - The **With column headers** checkbox must be selected;
  - The **Formal headers** setting must be disabled.
- Export the data to MS Excel by clicking the **Export now** button. The MS Excel table should be filled with data, with the first column containing the headings as they appear in the QUIK workstation.
- Return to the MS Excel export settings dialog in the QUIK workstation and change the following parameters:
  - Column – 2;
  - The **Formal headers** checkbox must be selected.
- Export the data again in MS Excel by clicking the **Export now** button. The first column contains the parameter headers, and the second column contains their service identifiers, which can be used as the param_name argument of the getParamEx() or getParamEx2() function, for example:

	A	B	C
1			MSNG
2	Face value	SEC_FACE_VALUE	1
3	Currency	SEC_FACE_UNIT	SUR
4	Scale	SEC_SCALE	3
5	Price step	SEC_PRICE_STEP	0,001

If MS Excel is not installed, you can add parameters in an alternative way in any text editor:

- In the QUIK workstation, create the **Quotes** table with the required set of parameters. In the table, one row with the instrument is enough, since you only need to get a list of headers. In the table editing window, select the **Transpose** checkbox.
- In the menu item **System/Settings/General settings...**, section Program/Clipboard, select the **Copy row headers**, **Copy column headers**, **Formal representation of data** and **Formal representation of row and column headers** checkboxes.
- Copy all data from the **Quotes** table (table context menu item **Copy entire table to clipboard**).
- Open any text editor and paste the copied data.

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**, client code **client_code** and **limit_kind** settlement period with the "0" value.

Function call syntax:

```
TABLE getPortfolioInfo (STRING firm_id, STRING client_code)
```

The function returns a Lua table with the following parameters:

№	Parameter	Type	Description	Short name
1	is_leverage	STRING	Attribute of monitoring positions type. Valid values: _ 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; _ MD: position monitoring scheme “by discounts” is used; _ F: position monitoring scheme “futures market” scheme is used. For clients of derivatives market without the unified cash position; _ <blank>: positions monitoring scheme “by limit” is used	Client type
2	in_assets	STRING	Estimation of client 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 instruments used as collateral	ValLongMargin
8	val_long_asset	STRING	Estimated value of long positions for non–marginal instruments included as collateral	ValLongAsset
9	assets	STRING	Estimated value of client 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

№	Parameter	Type	Description	Short name
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
15	locked_buy_margin	STRING	Estimated value of assets in buy orders for marginal instruments used as collateral	Lock. Margin. Buy
16	locked_buy_asset	STRING	Value of assets in buy orders for non–marginal instruments accepted as collateral	Lock. Coll. Buy
17	locked_sell	STRING	Estimated value of assets in sell orders for marginal instruments	Lock. Sell
18	locked_value_coef	STRING	Value of assets in buy orders for non–marginal instruments	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 instruments	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 instruments	LimBuy
24	lim_sell	STRING	Estimated value of marginal instruments available for selling	LimSell
25	lim_non_margin	STRING	Estimated cash volume available for buying non–marginal instruments	LimNonMargin
26	lim_buy_asset	STRING	Estimated cash assets available for purchasing instruments 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

№	Parameter	Type	Description	Short name
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
31	haircuts	STRING	Total discounts on the value of long (only for collateral instruments) and short positions in instruments, discounts of the correlation between instruments, as well as discounts on owed currencies not covered by instrument collateral in the same currencies	Haircuts
32	assets_without_hc	STRING	Total value of cash balances, prices of long positions in instruments held as collateral, and prices of short positions, without considering discount factors, netting of instrument 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 client 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. Valid values: 'HighRisk' means qualified, <blank> means not qualified	HighRisk
39	is_futures	STRING	Client account on the Moscow Exchange derivatives market, 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>–<Position code>. Example: "SUR–EQTV"	Calc. Parameters
41	init_margin	STRING	Value of the initial margin. The parameter is filled for MD clients	Init.margin
42	init_margin_net	STRING	Value of the initial margin calculated using the net method. The parameter is filled for MD+ clients	Init.margin net
43	min_margin	STRING	Value of the minimum margin. The parameter is filled for MD clients	Min.margin

№	Parameter	Type	Description	Short name
44	min_margin_net	STRING	Value of the minimum margin calculated using the net method. The parameter is filled for MD+ clients	Min.margin net
45	corrected_margin	STRING	Value of the corrected margin. The parameter is filled for MD clients	Corr.margin
46	client_type	STRING	Client type	Client type
47	portfolio_value	STRING	Portfolio value. For MD clients the value for rows with the maximum "limit kind" settlement period	Portfolio value
48	*start_limit_open_pos	STRING	Open positions limit for beginning of day	OpenPosLimBegin
49	*total_limit_open_pos	STRING	Open positions limit	OpenPosLim
50	*limit_open_pos	STRING	Planned net positions	PlanNetPos
51	*used_lim_open_pos	STRING	Current net positions	CurrNetPos
52	*acc_var_margin	STRING	Accrued variation margin	AccVarMarg
53	*cl_var_margin	STRING	Variation margin on the basis of intermediate clearing	AccVarMargIntCl
54	*opt_liquid_cost	STRING	Options liquidation value	OptLiquidVal
55	*fut_asset	STRING	Amount of estimated client's assets on derivatives market	FutMrkAssets
56	*fut_total_asset	STRING	Total amount of own client's assets on stock and derivatives markets	TotalMrkAssets
57	*fut_debt	STRING	Current debt on derivatives market	CurrDebdtFut
58	*fut_rate_asset	STRING	Funds adequacy	FundsAdeq
59	*fut_rate_asset_open	STRING	Funds adequacy (for open positions)	FundsAdeq (OpenPos)
60	*fut_rate_go	STRING	Liquidity coefficient of collateral	ColLiquidCoef
61	*planed_rate_go	STRING	Expected liquidity coefficient of collateral	ExColLiquidCoef
62	*cash_leverage	STRING	Cash Leverage	Cash Leverage
63	*fut_position_type	STRING	Type of position on derivatives market. Valid values: _ 0 – no position; _ 1 – futures; _ 2 – options; _ 3 – futures and options	PosTypeFutMrk

№	Parameter	Type	Description	Short name
64	*fut_accured_int	STRING	Accrued interest with consideration of premium on options and exchange fees	AccruedInt
65	rcv1	STRING	Risk coverage value 1. It is calculated as the difference between the Portfolio value and the Init. margin parameters. For MD and MD+ clients	RCS1
66	rcv1_net	STRING	Risk coverage value 1. It is calculated as the difference between the Portfolio value and the Init. margin net parameters. For MD+ clients	RCS1 net
67	rcv2	STRING	Risk coverage value 2. It is calculated as the difference between the Portfolio value and the Min. margin parameters. For MD and MD+ clients	RCS2
68	rcv2_net	STRING	Risk coverage value 2. It is calculated as the difference between the Portfolio value and the Min. margin net parameters. For MD+ clients	RCS2 net
69	demand	STRING	Demand, if Portfolio value < Init. Margin, this parameter is calculated as difference between the Init. margin and Portfolio value, otherwise the value is 0. For MD clients	Demand
70	current_bal	STRING	Current balance for all cash limits of the client	
71	open_pos	STRING	Estimated value of open client positions considering active orders. For MP clients	
72	money_locked	STRING	Number of blockings for active orders from cash limits	
73	is_marginal	STRING	For MLim clients (“by limits” lending type) the value is 1	
74	fundslevel	STRING	Available funds level. For MD and MD+ clients, otherwise the value is 9.99	Funds level

* – parameter is filled in for clients with the configured unified cash position and for clients of derivatives market without the unified cash position.

** – for details on discount factors, see Section 7 of Administrator's manual for Limits calculation library.

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_kind** settlement period with the value specified by user. The following optional arguments: position code **board_tag** and currency **currency**, specify the **Client portfolio** table calculation parameters, in the absence of these arguments position code and currency by default for the specified trade participant (firmid) are

used. In the table with the returned the curr_tag field parameters for which the calculation was performed in the <Currency>–<Position code> format are filled in.

Function call syntax:

> TABLE getPortfolioInfoEx (STRING firm_id, STRING client_code, NUMBER limit_kind), [STRING board_tag, STRING currency])

> **To receive parameters of Client portfolio table for clients of derivatives market without the unified cash position, use trading account on derivatives market as value of "client_code" and 0 as value of "limit_kind".**

Returns a Lua table with the parameters from the **Client portfolio** table. For the description of parameters, see 3.13.1. The function also returns the following parameters:

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 function returns a Lua table that contains parameters from the **Buy/Sell** QUIK table which indicate the possibility of buying or selling the specified instrument **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. Valid values: _ 0 – non margin; _ 1 – margin
2	is_asset_sec	STRING	This attribute shows if the instrument belongs to the list of instruments used as collateral. Valid values: _ 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 *

№	Parameter	Type	Description
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 considering 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 considering 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 instruments and their market price
19	spread_hc	STRING	Coefficient of correlation between instruments
20	can_buy_own	STRING	Maximum number of instruments 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 instruments 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 instrument **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 3.14.1.

The function also returns the following parameters:

№	Parameter	Type	Description
1	limit_kind	NUMBER	Position on date. Valid values: positive integers, starting from 0, corresponding to settlement periods from the Positions in instruments table: 0 – T0; 1 – T1; 2 – T2, etc.
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
6	client_type	STRING	Client type. Valid values: _ 1 – MLim; _ 2 – MLev; _ 3 – Mpos; _ 4 – MD; _ 5 – MD+
7	is_long_allowed	STRING	Attribute of an instrument 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 an instrument 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:

BOOLEAN Clear (NUMBER t_id)

The function returns **true** if completed successfully, otherwise **false**.

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 **true** if completed successfully, otherwise **false**.

3.15.6 DestroyTable

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

Function call syntax:

BOOLEAN 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**.

> **Addind rows is available only for the created window of the table after calling the CreateWindow function (for details, see 3.15.4).**

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 is used to get the status of the **t_id** table window.

> **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:

- **false** if the **t_id** table window is opened in the terminal;
- **true** if the **t_id** table window was closed manually by user. The window can be reopened by using the CreateWindow function;
- **nil** if the **t_id** table does not exist (nonexistent / incorrect identifier is specified or the table is deleted by using the DestroyTable function).

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:

BOOLEAN 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.

> **Calling Clear and DestroyTable functions for t_id inside the callback function f_cb, assigned for the table with this t_id is not allowed.**

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 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 color of the entire table or of a specific column, row or cell.

If the colour parameter is set to QTABLE_DEFAULT_COLOR, the color 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 color
Number of lines from 1 to N	QTABLE_NO_INDEX	A row is highlighted with the specified color
QTABLE_NO_INDEX	Number of columns from 1 to M	A column is highlighted with the specified color
QTABLE_NO_INDEX	QTABLE_NO_INDEX	The entire table is highlighted with the specified color

3.15.19 Highlight

This function highlights a selected cell range in the table **t_id** with the background and text color 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 color;
- **f_color** – text color;
- **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 color parameters used in the SetColor function (see 3.15.18).

3.15.20 SetSelectedRow

Function selects a certain row of table.

Function call syntax:

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.

Function call syntax:

NUMBER AddLabel(STRING chart_tag, TABLE label_params)

Parameters:

- **chart_tag** – tag of the graph to which the label is bound (for more details, see 4.2.4 of Section 4 of User's manual for QUIK, setting Magic);
- **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). Use images in format *.bmp or *.jpeg
3	ALIGNMENT	STRING	Text position relative to the image (four variants are possible: LEFT, RIGHT, TOP, BOTTOM)
4	YVALUE	NUMBER	Y–axis value of the parameter to which the label is bound

No.	Parameter	Type	Description
5	DATE	NUMBER	<YYYYMMDD> date format to which the label is bound
6	TIME	NUMBER	<HHMMSS> time format to which the label is bound
7	R	NUMBER	Red color component in RGB format, which is a number within an interval [0;255]
8	G	NUMBER	Green color component in RGB format, which is a number within an interval [0;255]
9	B	NUMBER	Blue color component in RGB format, which is a number within an interval [0;255]
10	TRANSPARENCY	NUMBER	Label transparency as a percentage. The value should fall within a range [0; 100]
11	TRANSPARENT_BACKGROUND	NUMBER	Transparency of image background. Valid values 0 for transparency disabled or 1 for transparency enabled
12	FONT_FACE_NAME	STRING	Font name (e.g., Arial)
13	FONT_HEIGHT	NUMBER	Font size
14	HINT	STRING	Popup hint

3.16.2 DelLabel

Function deletes a label with specified parameters.

Function call syntax:

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.

Function call syntax:

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.

Function call syntax:

```
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**.

> **Names of parameter labels in the returned table are specified in lowercase, all values are of STRING type.**

3.16.5 SetLabelParams

Function sets parameters for label with specified identifier.

Function call syntax:

```
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 instrument.

Function call syntax:

```
BOOLEAN Subscribe_Level_II_Quotes(STRING class_code, STRING sec_code)
```

Parameters:

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

Function call syntax:

BOOLEAN Unsubscribe_Level_II_Quotes(STRING class_code, STRING sec_code)

Parameters:

- **class_code** – class code;
- **sec_code** – instrument 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 instrument is ordered from the server.

Function call syntax:

BOOLEAN IsSubscribed_Level_II_Quotes (STRING class_code, STRING sec_code)

Parameters:

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

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

3.18 Functions for ordering Quotes table parameters

3.18.1 ParamRequest

This function orders receiving the **Quotes** table parameters.

Function call syntax:

BOOLEAN ParamRequest(STRING class_code, STRING sec_code, STRING db_name)

> **For correct work of the function, enable the attribute "smart data ordering" in settings of the QUIK workstation (System / Settings / General settings... menu item, Program / Receiving data / Quotes sections).**

Parameters:

- **class_code** – class code;
- **sec_code** – instrument code;
- **db_name** – parameter code.

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

3.18.2 CancelParamRequest

This function cancels a request for receiving the **Quotes** table parameters.

Function call syntax:

```
BOOLEAN CancelParamRequest(STRING class_code, STRING sec_code, STRING db_name)
```

> **For correct work of the function, enable the attribute "smart data ordering" in settings of the QUIK workstation (System / Settings / General settings... menu item, Program / Receiving data / Quotes sections).**

Parameters:

- **class_code** – class code;
- **sec_code** – instrument code;
- **db_name** – parameter code.

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

> **When using the ParamRequest() and CancelParamRequest() functions, it is possible to refuse from the requested parameters of the Quotes table if there is getParamEx2() function in the script.**
> **If getParamEx() fuction is used, then CancelParamRequest() function will not refuse from receiving parameters of the Quotes table.**

3.19 Functions for obtaining information of the unified cash position

3.19.1 getTrdAccByClientCode

This function returns the derivatives market trading account that corresponds to the stock market client code with a unified cash position.

Function call syntax:

```
STRING getTrdAccByClientCode(STRING firmid, STRING client_code)
```

Parameters:

- **firmid** – the stock market firm identifier;
- **client_code** – client code.

This function returns the line with the derivatives market trading accounts if the derivatives market client code has a unified cash position, otherwise – **nil**.

3.19.2 getTrdAccByClientCodeEx

This function returns the table containing all trading accounts that correspond to the stock market client code with a unified cash position.

Function call syntax:

```
TABLE getTrdAccByClientCodeEx(STRING firmid, STRING client_code)
```

Parameters:

- **firmid** – the stock market firm identifier;
- **client_code** – client code.

This function returns the table containing all trading accounts that correspond to the client code, if the stock market client code has a unified cash position. If the data is missing on the server, an empty table returns. If the data cannot be obtained due to an error, the function returns nil instead of the table.

3.19.3 getClientCodeByTrdAcc

This function returns the stock market client code with a unified cash position corresponding with derivatives market trading accounts.

Function call syntax:

```
STRING getClientCodeByTrdAcc(STRING firmid, STRING trdaccid)
```

Parameters:

- **firmid** – the stock market firm identifier;
- **trdaccid** – parameter code.

This function returns the line with the client code, if the specified trading account has a unified cash position, otherwise – **nil**.

3.19.4 IsUcpClient

The function is designed to receive the attribute indicating whether the client has a unified cash position.

Function call syntax:

BOOLEAN isUcpClient(STRING firmid, STRING client)

Parameters:

- **firmid** – the stock market firm identifier;
- **client** – client code of a stock market or trading account of derivatives market.

If specified client code has a unified cash position the function returns **true**, otherwise **false**.

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 instruments 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	Firm name
status	NUMBER	Status
exchange	STRING	Trading venue

See also the descriptions of getItem and getNumberOf.

4.3 Time and Sales

Time and Sales table parameters

Parameter	Type	Description
trade_num	NUMBER	Trade identifier
flags	NUMBER	Flags for Time and Sales 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	Instrument code
class_code	STRING	Class code
datetime	TABLE	Date and time

Parameter	Type	Description
period	NUMBER	Period of trading session. Valid values: _ 0 – opening; _ 1 – normal; _ 2 – closing
open_interest	NUMBER	Open interest
exchange_code	STRING	Exchange code in trading system
exec_market	STRING	Execution market
benchmark	STRING	Indicative rate identifier
trading_session_date	NUMBER	Exchange trading session date
z_spread	NUMBER	Z-spread by price

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
canceled_uid	NUMBER	ID of the user who has refused from trade
account	STRING	Trading account
price	NUMBER	Price
qty	NUMBER	Number of instruments, 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 Trades, Trades for execution tables

Parameter	Type	Description
price2	NUMBER	Buyback price
reporate	NUMBER	REPO rate (%)
client_code	STRING	Client code
accrued2	NUMBER	Profit (%) of the buyback date
repoterm	NUMBER	REPO term, in calendar days
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	Stock 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
bank_acc_id	STRING	Settlement account id/clearing organization code
broker_comission	NUMBER	Brokerage commission, accurate to 2 digits
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

Parameter	Type	Description
trans_id	NUMBER	Identifier of transaction. If the value 9 223 372 036 854 775 807 is exceeded, the field takes the value "-1". In this case, it is recommended to use the **trans_id_str** field
trans_id_str	STRING	Identifier of transaction
kind	NUMBER	Trade type. Valid values: _ 1 – Common; _ 2 – Negotiated; _ 3 – Initial placement; _ 4 – Transter of cash/ instruments; _ 5 – Negotiated trade with first REPO part; _ 6 – Settlement trade with SWAP operation; _ 7 – Settlement trade with OTC SWAP operation; _ 8 – Settlement trade with dual currency busket; _ 9 – Settlement OTC trade with dual currency busket; _ 10 – REPO with CCP tarde; _ 11 – First part of REPO with CCP tarde; _ 12 – Second part of REPO with CCP tarde; _ 13 – Negotiated REPO with CCP tarde; _ 14 – First part of negotiated REPO with CCP tarde; _ 15 – Second part of negotiated REPO with CCP tarde; _ 16 – Technical trade to return assets REPO with CCP; _ 17 – Trade with a spread between futures of different terms with the same asset; _ 18 – Technical trade of the first part of spread between futures; _ 19 – Technical trade of the second part of spread between futures; _ 20 – Negotiated trade with the first part of REPO busket; _ 21 Negotiated trade with the second part of REPO busket; _ 22 – rollover positions of derivatives market; _ 23 – Late correction–XLON;

Parameter	Type	Description
		_ 24 – Not to mark–XLON; _ 25 – Previous Day Contra; _ 26 – Ordinary trade immediate publication – XLON; _ 27 – Inter Fund Transfer delayed publication – XOFF; _ 28 – Negotiated trade delayed publication – XLON; _ 29 – Negotiated Trade immediate publication – XLON; _ 30 – OTC Late Correction – XOFF; _ 31 – Ordinary Trade delayed publication – XLON; _ 32 – Ordinary Trade Immediate publication – XOFF; _ 33 – SI Late Correction; _ 34 – SI Trade immediate publication; _ 35 – SI Trade delayed publication; _ 36 – OTC trade delayed publication – XOFF; _ 37 – OTC MTF TBA 1; _ 38 – OTC trade – delayed publication – XOFF; _ 39 – Inter fund cross – delayed publication requested MTF TBA 1; _ 40 – Cancellation of OTC trade after date of publication MTF TBA 2; _ 41 – OTC MTF TBA 1; _ 42 – OTC Trade – delayed publication MTF TBA 2; _ 43 – Inter fund cross – delayed publication requested MTF TBA 2; _ 44 – Cancellation of OTC trade after date of publication MTF TBA 2; _ 45 – OTC MTF TBA 3; _ 46 – OTC trade – delayed publication MTF TBA 3; _ 47 – Inter fund cross – delayed publication requested MTF TBA 3; _ 48 – Cancellation of OTC trade after date of publication MTF TBA 3; _ 49 – OTC MTF TBA 4; _ 50 – OTC trade – delayed publication MTF TBA 4; _ 51 – Inter fund cross – delayed publication requested MTF TBA 4;

Parameter	Type	Description
		_ 52 – Cancellation of OTC trade after date of publication MTF TBA 4; _ 53 – Delayed Publication Late Correction XLON; _ 54 – No to Mark Late Correction XLON; _ 55 – SWAP operation trade; _ 58 – SWAP negotiation operation trade; _ 59 – FX Non–deliverable Swap trade; _ 60 – FX Spot trade; _ 61 – FX Non–deliverable forward trade; _ 62 – FX deposits trade; _ 63 – FX Forward trade; _ 67 – Reverse EES trade; _ 68 – Direct covered EES trade; _ 69 – Direct uncovered EES trade; _ 70 – Direct aggregated EES trade; _ 71 – Futures exercise trade; _ 72 – Option expiration trade; _ 73 – Option lapse trade; _ 74 – Technical (prematching) trade with liquidity provider; _ 75 – Technical trade as a result of voluntary exiting perpetual futures (based on the submitted requests); _ 76 – Technical trade as a result of forced exiting perpetual futures (realization of unsatisfied demand); _ 77 – Technical trade with linked instrument as a result of exiting perpetual futures
clearing_bank_accid	STRING	Account identifier in NCC (settlement code)
canceled_datetime	TABLE	Date and time of order cancellation
clearing_firmid	STRING	Identifier of firm – clearing participant
system_ref	STRING	Additional information on trade transmitted by the trading system
uid	NUMBER	Identifier of user on QUIK server
lseccode	STRING	Preferred collateral
* order_revision_number	NUMBER	Revision number of an order for which the trade was concluded
* order_qty	NUMBER	Quantity in an order at the moment when the trade was concluded, in lots

Parameter	Type	Description
*order_price	NUMBER	Price in an order at the moment when the trade was concluded
order_exchange_code	STRING	Order Exchange code
exec_market	STRING	Execution market
liquidity_indicator	NUMBER	Liquidity indicator. Valid values: _ 0 – not defined; _ 1 – Add liquidity; _ 2 – Remove liquidity; _ 3 – Liquidity routed out; _ 4 – Auction
extref	STRING	External reference which is used to feedback to external systems
ext_trade_flags	NUMBER	Advanced flags received from the gateway directly with no QUIK server. The field is not populated
on_behalf_of_uid	NUMBER	UID of the user on whose behalf a trade was concluded
client_qualifier	NUMBER	Qualifier of the client on whose behalf a trade was executed. Valid values: _ 0 – not defined; _ 1 – Natural Person; _ 3 – Legal Entity
client_short_code	NUMBER	Short identifier of the client on whose behalf a trade was executed
investment_decision_maker_qualifier	NUMBER	Qualifier of the person or algorithm executed a trade. Valid values: _ 0 – not defined; _ 1 – Natural Person; _ 2 – Algorithm
investment_decision_maker_short_code	NUMBER	Short code to identify the person or algorithm executed a trade
executing_trader_qualifier	NUMBER	Determines if the execution of the order on which the trade was concluded was triggered by an algorithm or person. Valid values: _ 0 – not defined; _ 1 – Natural Person; _ 2 – Algorithm
executing_trader_short_code	NUMBER	Short code to identify the trader who executed an order on which the trade was executed

Parameter	Type	Description
waiver_flag	NUMBER	Indicates that the transaction was made according to pre–trade rules. Valid values of bit flags: _ bit 0 (0x1) – RFPT; _ bit 1 (0x2) – NLIQ; _ bit 2 (0x4) – OILQ; _ bit 3 (0x8) – PRC; _ bit 4 (0x10)– SIZE; _ bit 5 (0x20) – ILQD
mleg_base_sid	NUMBER	Identifier of a base instruments on the server for multileg instruments
side_qualifier	NUMBER	Operation qualifier. Valid values: _ 0 – not defined; _ 1 – Buy; _ 2 – Sell; _ 3 – Sell short; _ 4 – Sell short exempt; _ 5 – Sell undisclosed
otc_post_trade_indicator	NUMBER	OTC post–trade indicator. Valid values of bit flags: _ bit 0 (0x1) – Benchmark; _ bit 1 (0x2) – Agency cross; _ bit 2 (0x4) – Large in scale; _ bit 3 (0x8) – Illiquid instrument; _ bit 4 (0x10) – Above specified size; _ bit 5 (0x20) – Cancellations; _ bit 6 (0x40) – Amendments; _ bit 7 (0x80) – Special dividend; _ bit 8 (0x100) – Price improvement; _ bit 9 (0x200) – Duplicative; _ bit 10 (0x400) – Not contributing to the price discovery process; _ bit 11 (0x800) – Package; _ bit 12 (0x1000) – Exchange for Physical

Parameter	Type	Description
capacity	NUMBER	Role in order execution. Valid values: _ 0 – not defined; _ 1 – Agent; _ 2 – Principal; _ 3 – Riskless Principal; _ 4 – CFG give up; _ 5 – Cross as agent; _ 6 – Matched principal; _ 7 – Proprietary; _ 8 – Individual; _ 9 – Agent for other member; _ 10 – Mixed; _ 11 – Market maker
cross_rate	NUMBER	Cross rate of the trade price currency against the trade settlement currency
fixing_date	NUMBER	Quote fixing date for settlements. Filled in for the NDF contract type (see operation_type)
start_date	NUMBER	Date from which valuation is possible. Filled in for the FLEX FORWARD contract type (see operation_type)
operation_type	NUMBER	Operation type. Valid values: _ –1 – NOT_DEFINED; _ 0 – SPOT; _ 1 – FORWARD; _ 2 – SWAP; _ 6 – NDF; _ 7 – FLEX FORWARD
spot_rate	NUMBER	Quote price of a spot instrument at the trade conclusion moment
ts_commission_currency	STRING	Currency code of the trading system commission
broker_comission_currency	STRING	Currency code of the broker commission
trading_session	NUMBER	Trading session identifier. Va;id values: _ 0 – Not defined; _ 1 – Normal; _ 2 – Evening; _ 3 – Day results publishing
benchmark	STRING	Indicative rate identifier

Parameter	Type	Description
deposit_intent	NUMBER	Deposit trade type. Valid values: _ 0 – a trade is not the "deposit" type; _ 1 – Intention; _ 2 – Deposit; _ 3 – Reserve; _ 4 – Refund close; _ 5 – Deposit close; _ 6 – Refill shift
open_repo2date	NUMBER	Open REPO settlement date
open_repo2value	NUMBER	Open REPO buyback value
trading_session_date	NUMBER	Exchange trading session date
open_repo_evergreen_term	NUMBER	Period before the second part of the open REPO execution
z_spread	NUMBER	Z-spread by price
percent_period	NUMBER	Interest payments period

* – the parameter is used only for trades made by orders to which the order replacement transaction saving the number was applied.

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 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

Parameter	Type	Description
accruedint	NUMBER	Accrued interest
yield	NUMBER	Yield
trans_id	NUMBER	Identifier of transaction. If the value 9 223 372 036 854 775 807 is exceeded, the field takes the value -1. In this case, it is recommended to use the **trans_id_str** field
trans_id_str	STRING	Identifier of transaction
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
canceled_uid	NUMBER	ID of the user who withdrew an order
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
withdraw_datetime	TABLE	Date and time of order cancellation
bank_acc_id	STRING	Settlement account id/clearing organization code
value_entry_type	NUMBER	Methods of specifying an order volume Valid values: _ 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, %

Parameter	Type	Description
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. Valid values: _ 0 – not defined; _ 1 – Fill or Kill; _ 2 – Put in queue; _ 3 – Immediately or Cancel; _ 4 – Good till cancel; _ 5 – Good till date; _ 6 – Session; _ 7 – Open; _ 8 – Close; _ 9 – At the closing auction price; _ 11 – Till next session; _ 13 – While connected; _ 15 – Till time; _ 16 – Next scheduled intra–day auction
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	Role in order execution. Valid values: _ 0 – not defined; _ 1 – Agent; _ 2 – Principal; _ 3 – Riskless Principal; _ 4 – CFG give up; _ 5 – Cross as agent; _ 6 – Matched principal; _ 7 – Proprietary; _ 8 – Individual; _ 9 – Agent for other member; _ 10 – Mixed; _ 11 – Market maker
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

Parameter	Type	Description
visible	NUMBER	Visible quantity. Parameter of iceberg orders, 0 value is displayed for common orders
awg_price	NUMBER	Average execution price. It is used if the order was partially filled
expiry_time	NUMBER	Time the order expires in format <HHMMSS>. For GTT orders is is used in conjunction with the order expiration term (Expiry)
revision_number	NUMBER	Order revision number. It is used if the order was replaced saving the number
price_currency	STRING	Order currency
ext_order_status	NUMBER	Advanced order status. Valid values: _ 0 (by default) – not defined; _ 1 – New; _ 2 – Partially filled; _ 3 – Filled; _ 4 – Cancelled; _ 5 – Replaced; _ 6 – Pending cancel; _ 7 – Rejected; _ 8 – Suspended; _ 9 – Pending new; _ 10 – Expired; _ 11 – Pending replace
accepted_uid	NUMBER	UID of the manager user who confirmed an order in the confirmation mode
filled_value	NUMBER	Executed volume of an order in price currency for partially of fully executed orders
extref	STRING	External reference which is used to feedback to external systems
settle_currency	STRING	Settlement currency for an order
on_behalf_of_uid	NUMBER	UID of the user on whose behalf an order was submitted
client_qualifier	NUMBER	Qualifier of the client on whose behalf an order was submitted. Valid values: _ 0 – not defined; _ 1 – Natural Person; _ 3 – Legal Entity
client_short_code	NUMBER	Short identifier of the client on whose behalf an order was submitted

Parameter	Type	Description
investment_decision_maker_qualifier	NUMBER	Qualifier of the person or algorithm submitted an order. Valid values: _ 0 – not defined; _ 1 – Natural Person; _ 2 – Algorithm
investment_decision_maker_short_code	NUMBER	Short code to identify the person or algorithm submitted an order
executing_trader_qualifier	NUMBER	Determines if the execution of the order was triggered by an algorithm or person. Valid values: _ 0 – not defined; _ 1 – Natural Person; _ 2 – Algorithm
executing_trader_short_code	NUMBER	Short code to identify the trader who executed an order
settle_date	NUMBER	Settlement date. For swap trades, it is the date of the first part of a swap
settle_date2	NUMBER	Settlement date of the second part of a swap
start_date	NUMBER	Date from which valuation is possible. Filled in for the FLEX FORWARD contract type (see operation_type)
operation_type	NUMBER	Operation type. Valid values: _ –1 – NOT_DEFINED; _ 0 – SPOT; _ 1 – FORWARD; _ 2 – SWAP; _ 6 – NDF; _ 7 – FLEX FORWARD
qty2	NUMBER	Quantity of the second part of a swap
value2	NUMBER	Volume of the second part of a swap
visibility_factor	NUMBER	Visible part in the total order amount in percentage
visible_repo_value	NUMBER	Repo sum of a visible part (settlement currency accuracy of an order/instrument)
trading_session	NUMBER	Trading session identifier. Valid values: _ 0 – Not defined; _ 1 – Normal; _ 2 – Evening; _ 3 – Day results publishing
price_entry_type	NUMBER	Type of order price entry. Valid values: _ 1 – at price; _ 2 – at yield; _ 3 – at average weighted price

Parameter	Type	Description
lseccode	STRING	Code of an instrument considered preferable collateral
benchmark	STRING	Indicative rate identifier
external_qty	NUMBER	External quantity
upd_userid	STRING	Exchange identifier of the user who cancelled an order
trading_session_date	NUMBER	Exchange trading session date
z_spread	NUMBER	Z-spread by price

4.6 Participant's positions on trading accounts

Description of the Table of participant's positions for client on trading accounts:

Parameter	Type	Description
firmid	STRING	Firm ID
sec_code	STRING	Instrument 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

Parameter	Type	Description
firmuse	NUMBER	Indicates a collateral account. Valid values: _ 0 refers to normal accounts; _ 1 refers to a collateral account

4.7 Participant's positions in instruments

Description of the Table of Participant's positions in instruments:

Parameter	Type	Description
firmid	STRING	Firm
seccode	STRING	Instrument code
openbal	NUMBER	Opening position
currentpos	NUMBER	Current position
plannedposbuy	NUMBER	Number of instruments in active buy orders
plannedpossell	NUMBER	Number of instruments inactive sell orders
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 Stop Orders table
brokerref	STRING	Comment, usually: <client code>/<order number>
firmid	STRING	Dealer identifier
account	STRING	Trading account
condition	NUMBER	Stop price direction Valid values: _ 4 means <=; _ 5 means >=
condition_price	NUMBER	Stop price

Parameter	Type	Description
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. If the value 9 223 372 036 854 775 807 is exceeded, the field takes the value "-1". In this case, it is recommended to use the **trans_id_str** field
trans_id_str	STRING	Transaction identifier
client_code	STRING	Client code
co_order_num	NUMBER	Linked order
co_order_price	NUMBER	Linked order price
stop_order_type	NUMBER	Stop order type. Valid values: _ 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

Parameter	Type	Description
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
canceled_uid	NUMBER	Identifier of user who cancelled a stop order
order_date_time	TABLE	Time of stop order submission
withdraw_datetime	TABLE	Time of stop order cancellation
activation_date_time	TABLE	Date and time of stop order activation

4.9 Client account limits

Description of a futures limit parameters (**Client account limits** table):

Parameter	Type	Description
firmid	STRING	Firm ID
trdaccid	STRING	Trading account
limit_type	NUMBER	Type of limit. Valid values: _ 0 – Cash; _ 1 – Secured funds; _ 2 – By combined assets; _ 3 – Clearing cash; _ 4 – Clearing collateral cash; _ 5 – Limit for open positions on the spot market; _ 6 – Total deposit funds in foreign currency (in rubles); _ 7 – Deposit funds in foreign currency
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

Parameter	Type	Description
accruedint	NUMBER	Accrued 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 considering the set price alteration limits
risk_level	NUMBER	Client risk level. Valid values: _ 0 (empty, by default) – risk level is not specified; _ 1 – low–risk client; _ 2 – medium–risk client; _ 3 – high–risk client; _ 4 – exceptional–risk client
go_without_orders	NUMBER	Collateral without orders, considering the current realized risk (considering current prices) on risk parameters at the beginning of the trading session. Displayed with an accuracy of 2 digits after the separator
go_planned	NUMBER	Collateral without orders, considering current risk parameters and market data, in cash. Displayed with an accuracy of 2 digits after the separator
indicative_varmargin	NUMBER	Indicative variation margin considering current indicative currency rate, rub. (calculated similarly to the current indicative variation margin considering the variation margin on close positions). Calculated by the trading system of Moscow Exchange derivatives market. Displayed with an accuracy of 2 digits after the separator
net_option_value	NUMBER	Net value of options

4.10 Client account positions (Futures)

Description of the Table of positions for futures:

Parameter	Type	Description
firmid	STRING	Firm ID

Parameter	Type	Description
trdaccid	STRING	Trading account
sec_code	STRING	Futures contract code
type	NUMBER	Type of limit Valid values: _ 0 – not defined; _ 1 – main account; _ 2 – client and additional accounts; _ 4 – Accounts of all traders
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
session_status	NUMBER	Actual status of trading session. Valid values: _ 0 – not defined; _ 6 – main clearing finished
premium	NUMBER	Accrued premium on options

4.11 Cash positions

Cash positions table parameters:

Parameter	Type	Description
currcode	STRING	Currency code
tag	STRING	Position code
firmid	STRING	Firm ID
client_code	STRING	Client code
openbal	NUMBER	Opening balance
openlimit	NUMBER	Opening limit
currentbal	NUMBER	Current balance
currentlimit	NUMBER	Current limit
locked	NUMBER	Locked. Sum of assets blocked for client orders
locked_value_coef	NUMBER	Value of the buy orders for non–marginal instruments
locked_margin_value	NUMBER	Value of the buy orders for marginal instruments
leverage	NUMBER	Leverage
limit_kind	NUMBER	Position on date. Valid values: _ positive integers, starting from 0, corresponding to settlement periods from the Cash positions table: 0 – T0; 1 – T1; 2 – T2, etc.; _ negative integers – technological limits (used for internal operation of the QUIK system)
wa_position_price	NUMBER	Weighted average purchasing price of position
orders_collateral	NUMBER	Orders collateral
positions_collateral	NUMBER	Positions collateral

4.12 Removing cash position

Description of the Cash position removal table

Parameter	Type	Description
currcode	STRING	Currency code
tag	STRING	Position code
client_code	STRING	Client code

Parameter	Type	Description
firmid	STRING	Firm ID
limit_kind	NUMBER	Position on date. Valid values: _ positive integers, starting from 0, corresponding to settlement periods from the Cash positions table: 0 – T0; 1 – T1; 2 – T2, etc.; _ negative integers – technological limits (used for internal operation of the QUIK system)

4.13 Deleting instrument position

Description of the Deleting instrument position table

Parameter	Type	Description
sec_code	STRING	Instrument code
trdaccid	STRING	Trading account code
firmid	STRING	Firm ID
client_code	STRING	Client code
limit_kind	NUMBER	Position on date. Valid values: _ positive integers, starting from 0, corresponding to settlement periods from the Positions in instruments table: 0 – T0; 1 – T1; 2 – T2, etc.; _ negative integers – technological limits (used for internal operation of the QUIK system)

4.14 Deleting futures limit

Description of the Deleting futures limit table:

Parameter	Type	Description
firmid	STRING	Firm ID

Parameter	Type	Description
limit_type	NUMBER	_ Type of limit. Valid values: _ 0 – Cash; _ 1 – Secured funds; _ 2 – By combined assets; _ 3 – Clearing cash; _ 4 – Clearing collateral cash; _ 5 – Limit for open positions on the spot market; _ 6 – Total deposit funds in foreign currency (in rubles); _ 7 – Deposit funds in foreign currency

4.15 Positions in instruments

Description of the Positions in instruments table

Parameter	Type	Description
sec_code	STRING	Instrument code
trdaccid	STRING	Depo account
firmid	STRING	Firm ID
client_code	STRING	Client code
openbal	NUMBER	Opening balance
openlimit	NUMBER	Open limit
currentbal	NUMBER	Current balance
currentlimit	NUMBER	Current limit
locked_sell	NUMBER	In sell orders. Number of instruments locked for client's sell orders
locked_buy	NUMBER	In buy orders. Number of instruments locked for client's buy orders
locked_buy_value	NUMBER	Value of instruments locked for buying
locked_sell_value	NUMBER	Value of instruments locked for selling
wa_position_price	NUMBER	Purchasing price
wa_price_currency	STRING	The currency in which the purchasing price of the instrument is displayed. For bonds, the price of which is expressed as a percentage of face value, the % is returned. The parameter value can be <empty> if there is no data for calculating the purchasing price currency in the terminal at the time of data request

Parameter	Type	Description
limit_kind	NUMBER	Position on date. Valid values: _ positive integers, starting from 0, corresponding to settlement periods from the Position in instruments table: 0 – T0; 1 – T1; 2 – T2, etc.; _ negative integers – technological limits (used for internal operation of the QUIK system)
locked_out	NUMBER	The quantity of securities blocked in withdrawal instructions

4.16 Participant's cash positions

Description of the Participant's cash positions table:

Parameter	Type	Description
firmid	STRING	Firm ID
currcode	STRING	Currency code
tag	STRING	Position code
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
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 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
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	Identifier of transaction. If the value 9 223 372 036 854 775 807 is exceeded, the field takes the value -1. In this case, it is recommended to use the **trans_id_str** field
trans_id_str	STRING	Identifier of transaction
client_code	STRING	Client code

Parameter	Type	Description
repoentry	NUMBER	REPO order entry type Valid values: _ 0 – not defined; _ 1 – Price1+Rate; _ 2 – Rate+Price2; _ 3 – Price1+Price2; _ 4 – REPO sum+ Volume; _ 5 – REPO sum+Discount; _ 6 – Volume +Discount; _ 7 – REPO sum; _ 8 – Volume
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
quoteno	NUMBER	Non–negotiated counter–order
settle_currency	STRING	Settlement currency
sec_code	STRING	Instrument code
class_code	STRING	Class code
bank_acc_id	STRING	Settlement account id/clearing organization code
withdraw_date	NUMBER	Data when a negotiated order was cancelled, in the <YYYYMMDD> format

Parameter	Type	Description
linkedorder	NUMBER	Number of the previous order Displayed with "0" accuracy
activation_date_time	TABLE	Date and time of order activation
withdraw_date_time	TABLE	Date and time of order cancellation
date_time	TABLE	Date and time of order
lseccode	STRING	Preferred collateral
canceled_uid	NUMBER	ID of the user who withdrew an order
system_ref	STRING	System reference
price_currency	STRING	Order price currency
order_exchange_code	STRING	Order Exchange code
extref	STRING	External reference which is used to feedback to external systems
period	NUMBER	Trading session period when an order was submitted
client_qualifier	NUMBER	Qualifier of the client on whose behalf an order was submitted. Valid values: _ 0 – not defined; _ 1 – Natural Person; _ 3 – Legal Entity
client_short_code	NUMBER	Short identifier of the client on whose behalf an order was submitted
investment_decision_maker_qualifier	NUMBER	Qualifier of the person or algorithm submitted an order. Valid values: _ 0 – not defined; _ 1 – Natural Person; _ 2 – Algorithm
investment_decision_maker_short_code	NUMBER	Short code to identify the person or algorithm submitted an order
executing_trader_qualifier	NUMBER	Determines if the execution of the order was triggered by an algorithm or person. Valid values: _ 0 – not defined; _ 1 – Natural Person; _ 2 – Algorithm
executing_trader_short_code	NUMBER	Short code to identify the trader who executed an order
settle_date	NUMBER	Settlement date. For swap trades it is the settlement date of the first part of a swap

Parameter	Type	Description
benchmark	STRING	Indicative rate identifier
ext_negdeal_flags	NUMBER	Bit flags. Valid values: _ bit 0 (0x1) – The currencies on the negdeal order are filled in in accordance with the general rules for filling in currencies. _ bit 1 (0x2) – An order is Open REPO type
open_repo2date	NUMBER	Open REPO settlement date
open_repo2value	NUMBER	Open REPO buyback value
reject_reason	STRING	Cancel reason
trading_session_date	NUMBER	Exchange trading session date
filled_value	NUMBER	Executed volume of an order
valueentrytype	NUMBER	Type of an order entry
repovaluebalance	NUMBER	REPO sum balance
repo2valuebalance	NUMBER	REPO sum buyback balance
open_repo_evergreen_term	NUMBER	Period before the second part of the open REPO execution
percent_period	NUMBER	Interest payments period
z_spread	NUMBER	Z-spread by price

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 Trades, Trades for execution tables
brokerref	STRING	Comment, usually: <client code>/<order number>
firmid	STRING	Dealer identifier
account	STRING	Depo account

Parameter	Type	Description
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
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)

Parameter	Type	Description
type	NUMBER	Type. Valid values: _ 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
operation_type	NUMBER	Direction. Valid values: _ 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. Valid values: _ 1 – Processed; _ 2 – Not processed; _ 3 – In processing
clearing_type	NUMBER	Type of clearing. Valid values: _ 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	Instrument code
class_code	STRING	Class code
compval	NUMBER	Amount of compensation in currency of the trade

Parameter	Type	Description
parenttradeno	NUMBER	Parent trade ID
bankid	STRING	Settlement organization ID
bankaccid	STRING	Position code
precisebalance	NUMBER	Number of instruments for execution (in lots)
confirmtime	NUMBER	Time of confirmation in format <HHMMSS>
ex_flags	NUMBER	Extended flags of a trade for execution. Valid values: _ 1 – Confirmed by counterparty; _ 2 –Confirmed
confirmreport	NUMBER	Number of instruction
extref	STRING	External reference which is used to feedback to external systems
benchmark	STRING	Indicative rate identifier
benchmark_change_date	NUMBER	Date of the benchmark change in the <YYYYMMDD> format
benchmark_value	NUMBER	Benchmark value (%)
cancel_reason	STRING	Cancel reason
deposit_intent	NUMBER	Deposit trade type. Valid values: _ 0 – a trade is not the **deposit** type; _ 1 – Intention; _ 2 – Deposit; _ 3 – Reserve; _ 4 – Refund close; _ 5 – Deposit close; _ 6 – Refill shift
open_repo2date	NUMBER	Open REPO settlement date
open_repo2value	NUMBER	Open REPO buyback value
open_repo_report_no	NUMBER	Report for closing REPO with open date
open_repo_status	NUMBER	Open REPO status. Valid values: _ 0 – No; _ 1 – Yes; _ 2 – Finalized
trading_session_date	NUMBER	Exchange trading session date
open_repo_evergreen_term	NUMBER	Period before the second part of the open REPO execution
due_date	NUMBER	Due date of trade calculation
early_settle_status	NUMBER	Early execution status
early_settle_report_num	NUMBER	Report on early execution

Parameter	Type	Description
percent_period	NUMBER	Interest payments period
planned_interest	NUMBER	Amount of future interest payments
early_settle_cpreport_num	NUMBER	Counterparty report on early execution

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
description	STRING	Description
fullcoveredsell	NUMBER	Prohibition for uncovered sell. Valid values: _ 0 – No; _ 1 – Yes
main_trdaccid	STRING	Main trading account
bankid_t0	STRING	Settlement organization by T0
bankid_tplus	STRING	Settlement organization by T0
trdacc_type	NUMBER	Type of depo account
depunitid	STRING	Section of depo account
status	NUMBER	Status of trading account. Valid values: _ 0 – operations allowed; _ 1 – operations prohibited
firmuse	NUMER	Section type. Valid values: _ 0 – collateral section; _ otherwise – for trading sections
depaccid	STRING	Number of depo account in depository
bank_acc_id	STRING	Code of additional cash position
flags	NUMBER	Bit flags. Valid values: _ bit 0 (0x1) – Trading and clearing account for the deposit market

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 Order reports for NDM trades table
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 instruments, 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
sec_code	STRING	Instrument code
class_code	STRING	Class code
report_time	NUMBER	Report time
report_date_time	TABLE	Report date and time

4.21 Instruments

Description of the **Instruments** table:

Parameter	Type	Description
code	STRING	Instrument code
name	STRING	Instrument name
short_name	STRING	Short name for an instrument
class_code	STRING	Instrument class code

Parameter	Type	Description
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
bsid	STRING	Bloomberg ID
cusip_code	SRING	CUSIP
stock_code	STRING	StockCode
couponvalue	NUMBER	Coupon value
first_currcode	STRING	Quote currency code in a pair
second_currcode	STRING	Base currency code in a pair
base_active_classcode	STRING	Underlying asset classcode
base_active_seccode	STRING	Underlying asset
option_strike	NUMBER	Option strike
qty_multiplier	NUMBER	Quantity multiplicity
step_price_currency	STRING	Price step currency
sedol_code	STRING	SEDOL
cfi_code	STRING	CFI
ric_code	STRING	RIC
buybackdate	NUMBER	Offer date
buybackprice	NUMBER	Offer price
list_level	NUMBER	Listing level
qty_scale	NUMBER	Quantity scale
yieldatprevwaprice	NUMBER	Profitability according to the latest estimation
regnumber	STRING	Registration number

Parameter	Type	Description
trade_currency	STRING	Trade currency
second_curr_qty_scale	NUMBER	The accuracy of the amount of quoted currency
first_curr_qty_scale	NUMBER	The accuracy of the amount of underlying currency
accruedint	NUMBER	Accrued interest
stock_name	STRING	Code of derivative contract in QUIK format
nextcoupon	NUMBER	Date of coupon payment
couponperiod	NUMBER	Coupon period
settlecode	STRING	Current settlement code for the instrument
exp_date	NUMBER	Expiration date
settle_date	NUMBER	Settlement date
legs	TABLE	Legs of a composite instrument in format leg_<N>
_ leg_<N>	TABLE	The leg of a composite instrument, where N can be an integer value from 0
_ classcode	STRING	Parameter of a leg, indicating the class code of the linked instrument
_ seccode	STRING	Parameter of a leg, indicating the code of the linked instrument
_ CFI	STRING	Parameter of a leg, indicating CFI
_ ratio_qty	NUMBER	Parameter of a leg, indicating the number of position creation when executing transaction on a linked instrument
_ leg_side	STRING	Parameter of a leg, indicating the position direction when executing transaction on a linked instrument. Valid values: _ buy; _ sell
_ settle_date	NUMBER	Parameter of a leg, indicating the settlement date

4.22 Chart candlesticks

Parameters of a chart candlestick:

Parameter	Type	Description
open	NUMBER	Opening price
close	NUMBER	Closing price

Parameter	Type	Description
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. Valid values: _ 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 identifier. If the value 9 223 372 036 854 775 807 is exceeded, the field

Parameter	Type	Description
		takes the value -1. In this case, it is recommended to use the **trans_id_str** field
trans_id_str	STRING	Transaction's user identifier
status	NUMBER	Status of transaction. Valid values: _ 0 – transaction sent to server; _ 1 – transaction is received on QUIK server from client; _ 2 – error occurred when transferring transaction to the trading system. The transaction is not resent because connection to gateway of MOEX is broken; _ 3 – transaction executed; _ 4 – transaction is not executed by the trading system. Detailed description of the error in given in 'Message text' field; _ 5 – thansaction failed the control of QUIK server for certin criterias. For example, the control over user rights for sending transactions of this type; _ 6 – transaction faild limits control of QUIK server; _ 10 – transaction is not supported by the trading system; _ 11 – transaction faild validation control of the digital sugnature; _ 12 – no answer to transaction because the waiting timeout is expired. That may happen when submitting transaction from QPILE; _ 13 – transaction is rejected as its execution could result a cross trade (a trade with the same client code); _ 14 – transaction failed check of the additional restrictions set by broker; _ 15 – transaction is accepted after the violation of additional restrictions set by broker; _ 16 – transaction is canceled by user during the check of additional restrictions set by broker
result_msg	STRING	Message text
date_time	TABLE	Date and time
uid	NUMBER	Identifier

Parameter	Type	Description
flags	NUMBER	Transaction's flags
order_flags	NUMBER	Parameter of an order, indicating flags. Valid values: bit 2 (0x4) (order to sell, otherwise – to buy). This parameter value can be used only for transactions with the operation direction parameter (buy / sell), otherwise the parameter value is 0
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
*account	STRING	Trading account
*client_code	STRING	Client code
*brokerref	STRING	Comment
*class_code	STRING	Class code
*sec_code	STRING	Instrument code
*exchange_code	STRING	Order Exchange code
error_code	NUMBER	Error numerical code. The value is equal to 0 if the transaction was successful
error_source	NUMBER	Message source. Valid values: _ 1 – Trade system; _ 2 – QUIK server; _ 3 – Dealer library; _ 4 – Trade system gate
first_ordernum	NUMBER	Number of the first order that was created as a result of automatic client code replacing. It is used if the client code replacement is configured for a cross trade on the QUIK server
gate_reply_time	TABLE	Date and time at which a transaction response is received by the gate
sent_local_time	TABLE	Date and time at which a_transaction is sent, client local time in UTC

Parameter	Type	Description
got_local_time	TABLE	Date and time at which a transaction response is received, client local time in UTC
orders	TABLE	Orders. This parameter is added in response to a transaction only if there are two or more orders linked to the transaction
_ order_<N>	TABLE	The order, where N can be an integer value from 0
_ first_ordernum	NUMBER	Parameter of an order, indicating the number of the first order that was created as a result of automatic client code replacing. It is used if the client code replacement is configured for a cross trade on the QUIK server
_ order_num	NUMBER	Parameter of an order, indicating oder number
_ price	NUMBER	Parameter of an order, indicating price
_ quantity	NUMBER	Parameter of an order, indicating volume
_ balance	NUMBER	Parameter of an order, indicating balance
_ order_flags	NUMBER	Parameter of an order, indicating flags. Valid values: bit 2 (0x4) (order to sell, otherwise – to buy). This parameter value can be used only for transactions with the operation direction parameter (buy / sell), otherwise the parameter value is 0
_ firm_id	STRING	Parameter of an order, indicating frm ID
_ account	STRING	Parameter of an order, indicating trading account
_ client_code	STRING	Parameter of an order, indicating client code
_ brokerref	STRING	Parameter of an order, indicating comment
_ exchange_code	STRING	Parameter of an order, indicating order Exchange code
_ sec_code	STRING	Parameter of an order, indicating instrument code
_ class_code	STRING	Parameter of an order, indicating class code

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

4.25 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
settle_date	NUMBER	Settlement date
qty	NUMBER	Quantity of instruments in trades
qty_buy	NUMBER	Quantity of instruments in buy orders
qty_sell	NUMBER	Quantity of instruments in sell orders
netto	NUMBER	Netto position
debit	NUMBER	Debit
credit	NUMBER	Credit
sec_code	STRING	Code of the instrument in an order
class_code	STRING	Order class code
planned_covered	NUMBER	Planned T+ position
firm_use	NUMBER	Section type. Valid values: _ 0 – trading section; _ 1 – collateral section

4.26 Currency: commitments and demands on assets

Description of the **Currency: commitments and demand on assets** table:

Parameter	Type	Description
sec_code	STRING	Instrument code
class_code	STRING	Class code
firmId	STRING	Firm ID
account	STRING	Trading account
bank_acc_id	STRING	Identifier of settlement account in National Clearing Center

Parameter	Type	Description
date	NUMBER	Settlement date
debit	NUMBER	Commitment value
credit	NUMBER	Claim value
value_buy	NUMBER	Funds amount in buy orders
value_sell	NUMBER	Funds amount in sell orders
margin_call	NUMBER	Refundable amount of the compensation transfer
planned_covered	NUMBER	Planned T+ position
debit_balance	NUMBER	Commitment amount at the beginning of the day, with accuracy of two decimal points
credit_balance	NUMBER	Claim amount at the beginning of the day, with accuracy of two decimal points

5. Description of bit flags

5.1 Flags for the Orders, Negdeal Orders 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
bit 3 (0x8)	A limit order; otherwise it is a market order
bit 4 (0x10)	Execute trade 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)	Hidden order
bit 8 (0x100)	Immediately or cancel
bit 9 (0x200)	An iceberg order
bit 10 (0x400)	Order rejected by the trading system
bit 20 (0x100000)	The linkedorder field is filled in with the stop order number

5.2 Flags for Trades, Trades for execution tables

Flag is set	Value
bit 0 (0x1)	Margin trade
bit 2 (0x4)	Sell trade, otherwise – Buy trade
bit 3 (0x8)	Trade of iceberg order
bit 4 (0x10)	Canceled trade (Status – C), the **canceled_datetime** field is filled in with the date and time of trade cancellation
bit 5 (0x20)	Passive trade (Status – P)
bit 6 (0x40)	Active trade (Status – A)
bit 7 (0x80)	The first part of the swap operation
bit 8 (0x100)	The second part of the swap operation

5.3 Flags for Order reports for NDM trades table

Flag is set	Value
bit 6 (0x40)	Report sent by user (Sent status)
bit 7 (0x80)	The report was received by the user from another counterparty of the trade for execution (Received status)

If both flags are present, the status is **Sent** and **Received**. If no flag is set, the status is **Status unknown**.

5.4 Flags for Time and Sales 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.

5.5 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

5.6 Additional flags for Stop Orders table

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

5.7 Flags for Transactions table

Flag is set	Value
bit 20 (0x21)	Service transaction
bit 21 (0x22)	Order submitting transaction

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,…)

6.6 bit.test

This function checks the bit specified in a value. Returns **true** if a bit is equal to 1, returns **false** if a bit is equal to 0.

Function call syntax:

BOOLEAN bit.test(NUMBER x, NUMBER n)

where:

- x – value;
- n – bit number. Bit numbering starts from 0.

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 does not exist in the directory create it manually. List of scripts is available in dialogue **Servoces / LUA scripts...**

The 5.3.5 version of the Lua machine in which the indicators are launched by default can be changed in the QUIK workstation settings in the **Lua scripts** section – **Lua version for scripts indicator**, the description is given in 2.10.7 sub– section Chapter 2 "Basic operating principles".

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:

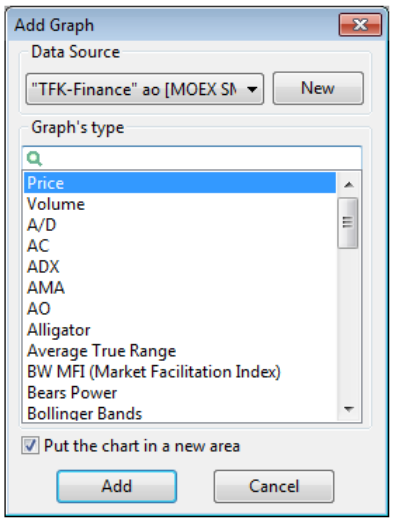
- Function Init is defined;
- Function OnCalculate is defined;
- Lua table named Settings that contains Name field;
- luac file is compiled for the version of the Lua machine selected in the settings.

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 called when adding an indicator on a chart (when pressing the **Add** button on a chart window), returns the value defining the number of lines in indicator.

> **The function is called also when rerunning the QUIK workstation and when loading .wnd or .tab files that stores the graph with 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 OnChangeSettings

The function is called when editing the indicator properties after clicking **Apply** or **OK**.

> **The function is called also when rerunning the QUIK workstation and when loading .wnd or .tab files that stores the graph with indicator.**

Function call syntax:

OnChangeSettings()

For example:

```
Settings={Name="test1"}
function Init()
	return 1
end
function OnChangeSettings()
```

```
	message(Settings.Name)
end
```

7.2.5 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 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
```

7.2.6 CandleExist

The function is designed to check for the existence of the candle on the price and volume chart:

BOOLEAN CandleExist(NUMBER index)

Parameters:

- **index** – candle index.

The function takes as a parameter the index of the candle and returns **true** if candle exists, otherwise **false**.

> **The function determines the candles that need not be considered in the calculation of the indicator. These candles appear if you add on one chart with the price and volume graph the graph of the parameter values from the Quotes Table or when the graph setting "Show empty intervals" is enabled. For such candles the O, H, L, C, V functions return "nil", and the T function returns empty candle's time.**

An example of use of a function for Moving Average indicator at close price:

```
Settings= {
  Name = "Moving Average Lua",
  Period = 9,
  line =
  {
    {
      Name = "Moving Average Lua",
```

```
      Color = RGB(90, 110, 200),
      Type = TYPE_LINE,
      Width = 1
    }
  }
}

function Init()
  QUEUE = {}
  SUM = 0
  return 1
end
function OnCalculate(index)
  return Average(index)
end

function Average(indx)
  --if the indicator is recalculated, queue and sum are reset
  if indx == 1 then
    QUEUE = {}
    SUM = 0
  end
  --if the candle is not empty, add it to the queue and sum up the close value
  if CandleExist(indx) then
    table.insert(QUEUE, {C_value = C(indx)})
    SUM = SUM + QUEUE[#QUEUE].C_value
    --if the queue reaches the required period, the average value is calculated
    if #QUEUE == Settings.Period then
      local avg = SUM/Settings.Period
      SUM = SUM - QUEUE[1].C_value
      table.remove(QUEUE, 1)
      return avg
    end
  else
    return nil
  end
end
```

7.2.7 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:

> **For correct work of the function getDataSourceInfo called from Init function, rerun the QUIK workstation after having added the indicator on graph.**

Parameter	Type	Description
interval	NUMBER	Current interval (time frame) of a chart
class_code	STRING	Class code of the data source
sec_code	STRING	Instrument 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

7.2.8 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. Numbering starts with 1;
- **line_number** – line number. Numbering starts with 1;
- **value** – a value to be set. Numbering starts with **nil**.

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

The example is given below.

7.2.9 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
```

7.2.10 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
```

7.2.11 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_POINT, TYPE_DASHDOT, TYPE_DASH, TYPE_TRIANGLE_UP, TYPE_TRIANGLE_DOWN, TYPE_CANDLE, TYPE_BAR.

```
Settings.line[1].Type = TYPE LINE --lines
Settings.line[2].Type = TYPE_HISTOGRAM --histograms
Settings.line[3].Type = TYPE_POINT --points
Settings.line[4].Type = TYPE_DASHDOT --dot-dash
Settings.line[5].Type = TYPE DASH -- dashes
Settings.line[6].Type = TYPE_TRIANGLE_UP --triangle_up
Settings.line[7].Type = TYPE_TRIANGLE_DOWN -- triangle_down
Settings.line[8].Type = TYPE_CANDLE --candles
Settings.line[9].Type = TYPE_BAR --bars
```

- **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.

> **To indicate that an indicator parameter is a real number, it is necessary to supplement its value with a suffix of the form <.N>. Otherwise, the parameter is considered an integer value, and when editing the indicator properties manually, you can enter only an integer value.**
> **Example:**

```
Settings.double_param1 = 1.0
Settings.double_param2 = 1.1
Settings.integer_param = 1
```

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

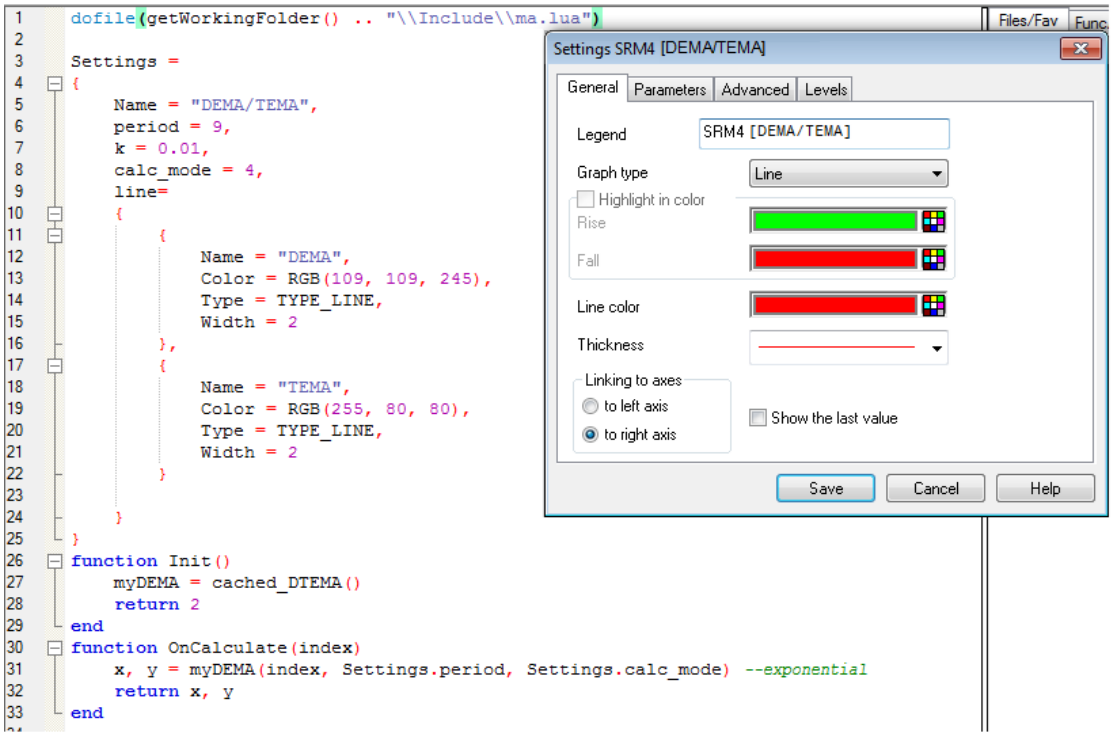
> **For parameters Settings.Name, Settings.line[n].Name (and any other custom string parameters), multi–string construction is not recommended to be used.**

When using a multi–string construction, only the first string is accounted, for example, for parameter of the following view:

```
Settings.Name = [[Two
MA]]
```

string “Two” becomes name of the indicator.

7.2.12 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.

7.2.13 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 **Quotes** table.
- **getParamEx2** – receives the values of all exchange data parameters from the **Quotes** table with a possibility to deny receiving particular parameters.
- **message** – displays the values of the QUIK workstation.
- **isDarkTheme** – allows receiving information about interface design theme currently used in the terminal: standard or dark.
- **isConnected** – defines the state of workstation connection to server.

- **isUcpClient** – receives the attribute indicating whether the client has a unified cash position.
- **PrintDbgStr** – function displays debug information.
- **getTradeDate** – received the data of a trading session.
- **getInfoParam** – allows receiving parameters for information window (Connection / Information window).
- **getClassSecurities** – receives a list of instrument 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 an instrument.
- **getQuoteLevel2** – receives the Level II quotes table for selected class and instrument.
- **getMoney** – receives information on cash positions.
- **getMoneyEx** – receives information on cash positions of a specified type.
- **getDepo** – receives information on positions in instruments.
- **getDepoEx** – receives information on positions in instruments of a specified type.
- **Subscribe_Level_II_Quotes** – orders receiving the **Level II Quotes** table from the server for a specified class and instrument.
- **Unsubscribe_Level_II_Quotes** – cancels the order for receiving the **Level II Quotes** table from the server for a specified class and instrument.
- **IsSubscribed_Level_II_Quotes** – function allows to know whether the **Level II Quot**es table for a specified class and instrument is ordered from the server.
- **ParamRequest** – orders receiving the **Quotes** table parameters.
- **CancelParamRequest** – cancels a request for receiving the **Quotes** table parameters.
- **sendTransaction** – function for working with orders.
- **CalcBuySell** – calculates the maximum possible number of lots in order.
- **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 portf**olio table.
- **getBuySellInfo** – receives the values of parameters of **Buy / Sell** table.
- **getPortfolioInfoEx** – receives the values of parameters of **Client portfolio** table accounting the position on date.
- **getFuturesLimit** – receives information on Futures Limits.
- **getFuturesHolding** – receives information on futures positions.
- **getBuySellInfoEx** – receives the values of parameters of **Buy / Sell** table accounting the position on date.
- **getTrdAccByClientCode** – returns the derivatives market trading account that corresponds to the stock market client code with a unified cash position.
- **getTrdAccByClientCodeEx** – returns the table containing all trading accounts that correspond to the stock market client code with a unified cash position.
- **getClientCodeByTrdAcc** – returns the stock market client code with a unified cash position corresponding with derivatives market trading accounts.
- **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.
- **getLinesCount** – receives the number of lines on a chart (indicator) for a specified identifier.
- **getNumCandles** – receives the number of candlesticks for a specified identifier.
- **getCandlesByIndex** – receives information about candlesticks for an identifier.

7.2.14 Loading and saving of an indicator configuration to file

When selecting the menu item **System / Save windows 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. Thred–safe functions for working with Lua tables

Simultaneous work with tables of the callback script functions and main() function can cause uncertain situations. To solve this problem qlua.dll provides thread–safe analogues of standard Lua functions.

> **Performing thread–safe function blocks execution of code on another thread before the end of the function.**

Call format of thread–safe function coincides with call format of the analogic standard Lua function.

The table below shows the standard Lua functions and thread–safe functions corresponding to them:

Standard Lua function	Thred–safe function
concat	sconcat
remove	sremove
insert	sinsert
sort	ssort

9. Appendixes

9.1 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

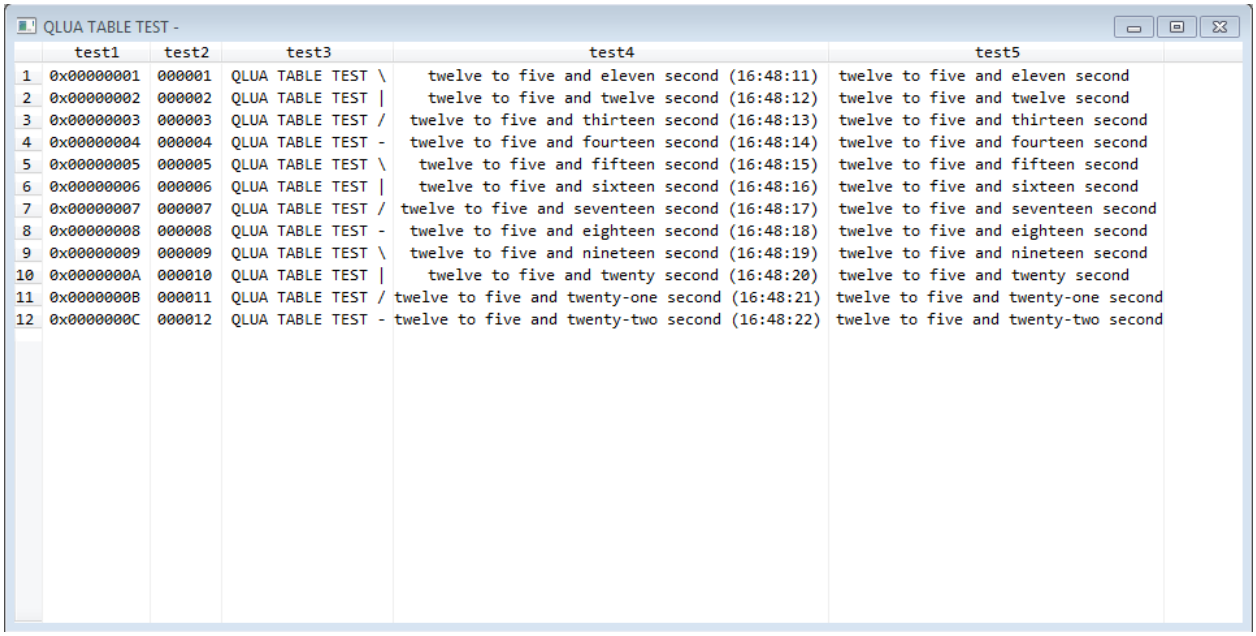
function _Time(t)
	hour = t.hour
	minute = t.min
	hour = hour % 12
	if(hour == 0) then
	hour, nextHourWord = 12, "one "
	else
	nextHourWord = numberToWord(hour+1)
	end
	hourWord = numberToWord(hour)
	if(minute == 0 ) then
	return hourWord .. "o'clock"
```

```
        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
```

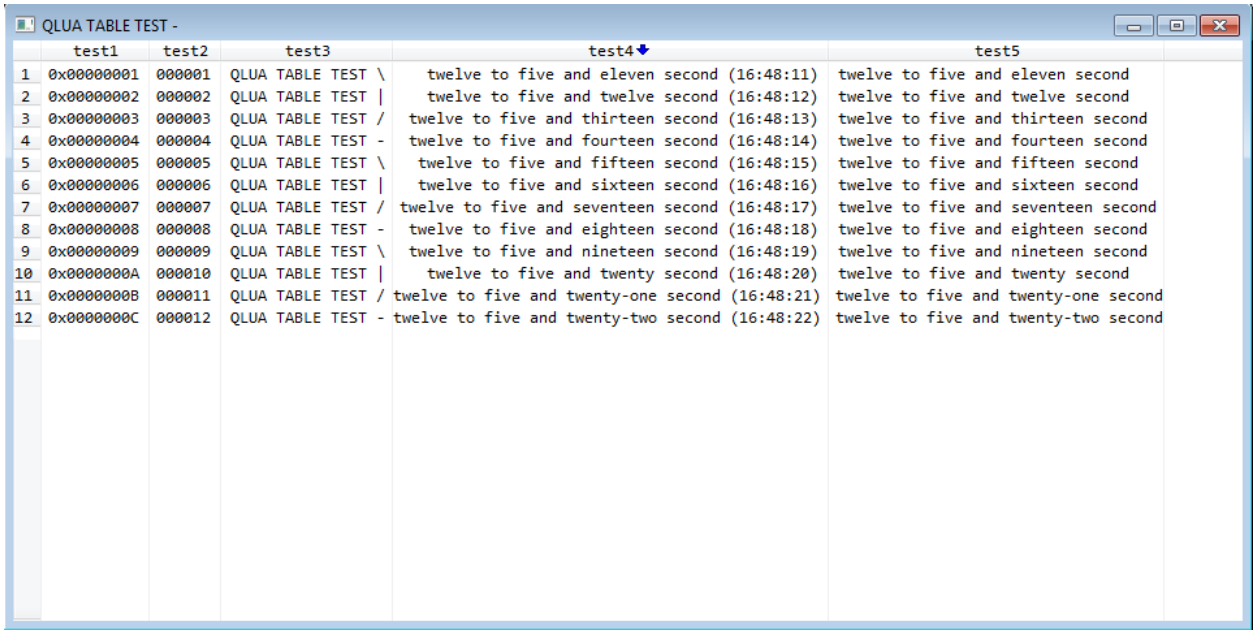
This script creates the following table in the QUIK workstation:



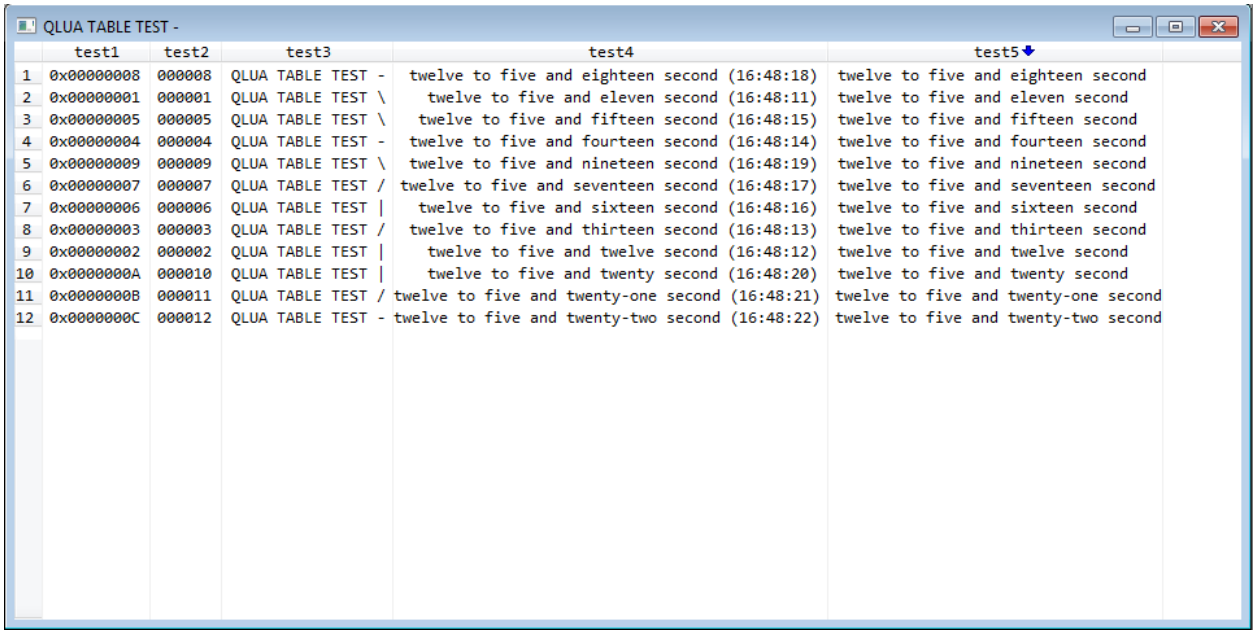
	test1	test2	test3	test4	test5
1	0x00000001	000001	QLUA TABLE TEST \	twelve to five and eleven second (16:48:11)	twelve to five and eleven second
2	0x00000002	000002	QLUA TABLE TEST |	twelve to five and twelve second (16:48:12)	twelve to five and twelve second
3	0x00000003	000003	QLUA TABLE TEST /	twelve 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

9.2 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

9.3 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
		message("Event on click to cell 'DO NOT CLICK ME'!", 3)
	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")

	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
```

9.4 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 does not 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")
```