

Chapter 8. QPILE Language

| | |
|---|----|
| 8.1 General..... | 4 |
| 8.2 Working with QPILE tables..... | 6 |
| 8.3 Program file structure | 10 |
| 8.4 QPILE language constructs | 14 |
| 8.5 General functions | 22 |
| 8.6 Mathematical functions | 24 |
| 8.7 Functions for working with collections (COLLECTION)..... | 26 |
| 8.8 Functions for working with associative arrays (MAP) | 27 |
| 8.9 Functions for accessing rows in arbitrary QUIK tables | 29 |
| 8.10 Functions for accessing a list of available parameters..... | 49 |
| 8.11 Functions for handling programmable tables..... | 50 |
| 8.12 Functions for getting values from the quotes table..... | 53 |
| 8.13 Functions for retrieving values from the Level II Quotes table | 62 |
| 8.14 Functions for retrieving values from the Positions in instruments table..... | 62 |
| 8.15 Functions for retrieving values from the Cash positions table | 65 |
| 8.16 Functions for the calculation of margin positions | 67 |
| 8.17 Functions for retrieving values from the Client Portfolio and Buy / Sell tables... | 68 |
| 8.18 File handling functions | 75 |
| 8.19 String handling functions..... | 77 |
| 8.20 Chart handling functions | 78 |
| 8.21 Order handling functions | 81 |
| 8.22 Label handling functions..... | 83 |
| 8.23 Service Functions | 86 |
| 8.24 QPILE program debugging | 89 |
| APPENDIX 1. QPILE command syntax | 91 |
| APPENDIX 2. Recommendations for writing programs in QPILE | 93 |

This chapter describes the use of QPILE, an algorithmic language built into QUIK system workstations. The first part of the chapter reviews main functions of the client terminal that are



intended for work with programmable tables. In this document, users interested in developing their own programs can find the descriptions of QPILE constructs and built-in functions, as well as implementation examples (calculation of margin lending according to the guidelines of Federal Financial Market Service).

8.1 General

8.1.1 Purpose

The QPILE (QUIK Programmable Interface and Logic Environment) language is a set of instructions interpreted on the QUIK workstation.

QPILE is intended for creation of new tables for real-time calculation of the user's individual indicators based on information obtained from other tables.

This functionality is primarily useful for brokers, because each broker uses his or her own strategy for calculating client's position indicators. A built-in language allows for implementation of virtually any algorithm.

QPILE application examples:

1. Dynamic revaluation of client assets at the broker's workstation.
2. Dynamic revaluation of assets in the client portfolio and their total value.
3. Calculation of indicators not available in QUIK or in the exchange trading system using custom algorithms.
4. Calculation of margin lending parameters in compliance with the standards currently in effect.
5. Programming of a trading strategy that would generate signals for changing positions of instruments.

The primary QPILE application is to calculate portfolio values; that is why the term 'portfolio' will be frequently used in the description of table types.

8.1.2 QPILE operation mode

1. The table structure (the function of columns and rows, and formulas for calculation of parameters) is described in the form of a QPILE program. The formulas for calculation allow the use of standard mathematical and logic operations, variables and data arrays, as well as data obtained from other QUIK tables.
2. The program code can be obtained from the QUIK server (**server code**) or from the user's drive (**local code**). The QPILE language interpreter processes this code on the QUIK workstation and recalculates formula values at regular intervals. This provides an internal data source for the values to be displayed in tables. Multiple tables can use one data source based on the same program. In this way, redundant calculations degrading system performance can be avoided.
3. The tables created in a program have the functions of standard QUIK tables.



4. A QUIK workstation has a built-in QPILE code debugger that allows single-step debugging and inspecting current variable values. For detailed information, see [8.24](#).

Rows in the QPILE tables are numbered starting from one, while characters in the strings in QPILE are numbered starting from zero.

8.1.3 Basic capabilities

QPILE basic functions are listed below:

1. Describing new tables of arbitrary structure;
2. Calculating table fields using mathematical formulas and Boolean expressions;
3. Highlighting table cells with different colours depending on the value;
4. Audio and text notifications.

A QPILE table supports all basic table operations available in QUIK:

- Editing, including selecting parameters to be displayed and their priority order;
- Hotkeys;
- Placement on screen tabs;
- Lookup of values in table cells;
- Table printout with preview;
- Copying data into the Windows Clipboard;
- Exporting data into Excel;
- Exporting data via ODBC.

Information from QUIK charts and tables listed below can be used as **source data** for calculating table parameters:

| No. | Table |
|-----|--------------------------|
| 1 | Quotes table |
| 2 | Time and Sales table |
| 3 | Orders table |
| 4 | Stop orders table |
| 5 | Trades table |
| 6 | Positions in instruments |
| 7 | Cash positions |
| 8 | Client account positions |
| 9 | Client account limits |
| 10 | Negdeal orders |

| No. | Table |
|-----|---|
| 11 | Trades for execution |
| 12 | Order reports for NDM trades table |
| 13 | Client portfolio table |
| 14 | Buy / Sell table |
| 15 | Participant's cash positions |
| 16 | Participant's positions in instruments |
| 17 | Participant's positions on trading accounts |
| 18 | Table created during calculation of the program |



8.1.4 Functional constraints

The current version of QPILE and the tables created using this version do not support the following operations:

- Sorting in tables;
- Filters;
- Using the table as a data source for constructing charts;
- Saving table data into a text file using a shortcut menu;
- Exporting data into technical analysis systems.

8.2 Working with QPILE tables

8.2.1 Loading a program

At this stage, the descriptions of user tables are added to the list of available types. If table descriptions will be handled on the server, this part can be skipped.

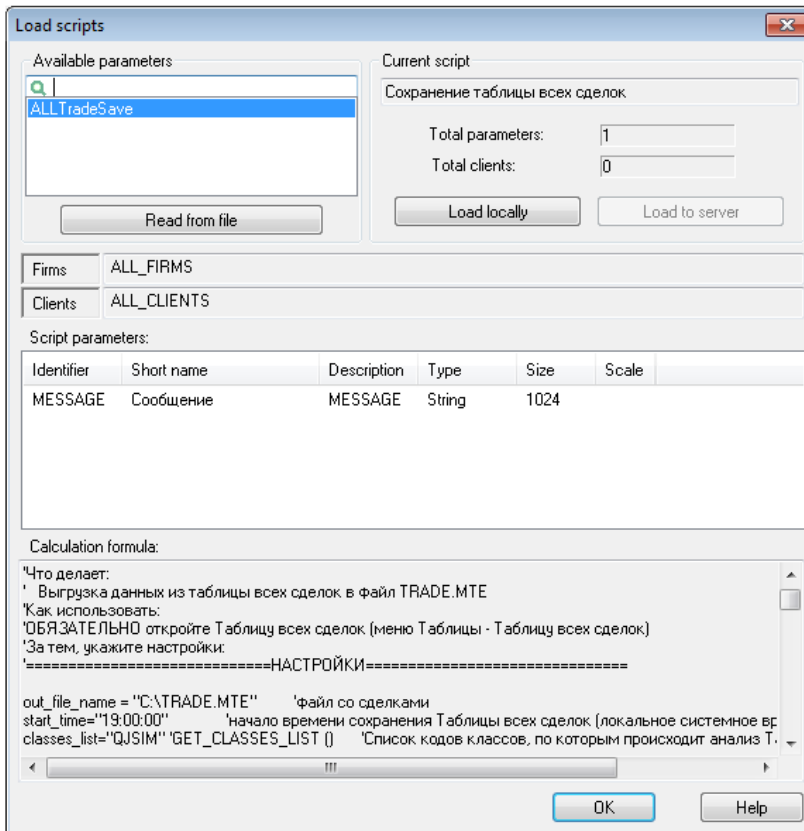
To load the program code, select **Services / QPILE scripts**, or press CTRL+F11.

1. Click **Add** and select the file on the drive to read the program from. The files have a standard extension. QPL. The table name will appear in the **Available scripts** list.
2. While the file is read, the program is checked for correctness. If the program contains errors, the system notifies the user in the **Message Window**. The error message shows the file name and the line number of the error.
3. If the file is read successfully, relevant parameters are shown in the table fields as follows:

| Field | Purpose |
|---------------------|---|
| Current script | Table name |
| Total parameters | The number of parameters (columns) described in the table structure |
| Total clients | The number of available client codes from those listed in the table structure. This parameter is not applicable in the new version of the language |
| Firms | The list of firm identifiers used in the trading system (values corresponding to the Dealer field in the Orders or Trades tables) |
| Clients | The list of client identifiers to be shown in the table |
| Script parameters | The list of described parameters and detailed information about them |
| Calculation formula | Program source code in QPILE |

4. Click **Load locally** to load the table that was read from a local file.
5. Click **Load to server** to load the program to the server, where it will be available to all server users. The permission to upload programs to the server is granted by the QUIK administrator.





When loaded locally, the code is executed on the current workstation, and the table based on the code can be viewed on this workstation only.

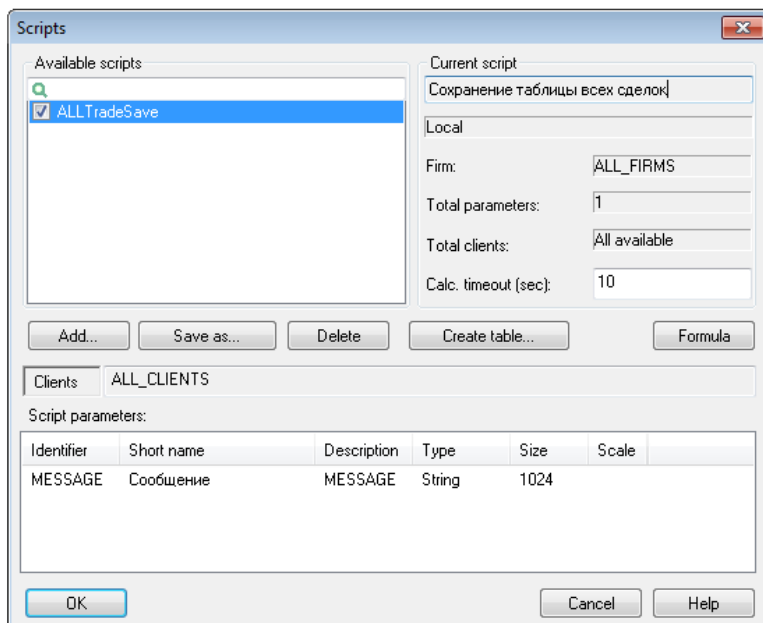
It is strongly recommended that you first load each new description locally, and only after reviewing it and verifying that it works and that the parameter are calculated correctly, upload it to the server.

8.2.2 Setting program parameters

At this stage, the list of processed programs and the periodicity of calculations is determined. If the programs were loaded locally, they are automatically selected.

To select available programs with descriptions of table type, select **Services / QPILE scripts** or press CTRL+F11.





1. The **Available scripts...** list shows all programs that can be used to create tables, available both on the local machine and from the server. Select a program from the list by checking its checkbox. The fields in this dialogue window will display parameters related to the structure of the table being programmed.
2. Set calculation period for the table. If the program to be executed involves a large number of calculations, it is recommended that the calculation period be set to a minimum of 5-10 seconds.
3. Save the settings by clicking the **Save as...** button.
4. Click **Delete** to delete a selected item from the list. A table that is loaded from the server cannot be deleted.
5. Click **Formula** to open the window containing the program source code and the description of the table (shown as an example).
6. Click **Create table...** to **Create script table** (see [8.2.3](#)).
7. Click **OK** to close the window and save the settings. Click **Cancel** button to close the window without saving any changes.

Settings made at this stage also apply to all previously created programmable tables. This stage can be used to enable/disable calculations in tables and to adjust the data update intervals.

Applying settings to existing tables:

- How can recalculation of table values be disabled?
 - Select **Suspend calculation** from the shortcut menu or press CTRL+F11 and clear the checkbox of the required table type. The table will show the last calculated values;
 - If more than one table was generated based on one program, the calculation will be disabled for all these tables;
 - Press CTRL+F11 and clear all checkboxes to disable recalculation in all tables.




- How the calculation interval in an existing table can be modified?
 - Select **Scripts settings** from the shortcut menu or press CTRL+F11, then, change the value of **Calc. timeout (sec)** and click **OK**.

8.2.3 Creating a table

At this stage, a table based on a program is created. One program can be used to as a basis for several tables for convenient display of these tables on the screen.

To create a programmable table, press **Create table...** in Scripts table (see [8.2.2](#)).

1. Select a program from the **Available scripts** list. Table structure data will be shown in the fields of the **Current script** section.
2. If required, use the **Clients filter** to limit the number of displayed rows in the table.
3. Create a list of table columns selecting them from the available parameters and specify the order in which they will be displayed in the table. The **Parameter description** field contains a note describing the selected parameter in detail.
4. Click **OK** to create a table.

The created table has the same control functions as other QUIK tables. For example, the user can click  on the toolbar or press CTRL+E to edit the table. RECOMMENDATIONS: Table parameters are calculated based, among other things, on the Quotes table. Ensure that data required for calculating parameters can be received from the server (that they are not filtered out from the list of received parameters and instruments).

8.2.4 Functions available for a table

Table data can be copied, exported to Excel, or exported via ODBC.

Functions available from the table's shortcut menu:

- **Suspend calculation** suspends calculation of the table's parameters (with the last calculated data shown in the table);
- **Start calculation** – restore calculation of the table parameters if it was suspended;
- **Restart calculation** clears all values and starts the calculation anew;
- **Start calculation in debug mode** clears all values, opens the debug window, and starts the calculation anew;
- **Save description to file** saves the program that describes the table to a text file;
- **View source** displays the table's program code in a window;
- **Script settings** opens a window that shows the program parameters including the calculation interval in seconds.

Description of the standard functions of the context menu for tables is given in sub-section 2.8.4 of Section 2: Basic Operating Principles.

Description of user filters and conditional formatting of tables is given in sub-sections 2.8.8 and 2.8.9 correspondently of Section 2: Basic Operating Principles.

The complete list of shortcut keys for all table types is shown in Appendix to Section 2.

8.3 Program file structure

A program code file is a text file encoded with in CP-1251 encoding. A single file describes one particular type of table. The files have standard extension. QPL.

The description consists of three parts:

1. The 'Header' contains the table name and the description of basic parameters;
2. The 'Program body' contains the program code;
3. The 'Description of table columns' contains the names of table columns and format definition for the relevant table cells.

All parts are mandatory and must follow strictly in the specified order. Each file must always begin with a header and end with the string END_PORTFOLIO (END_PORTFOLIO_EX is used in the new version of the language).

Examples of files see in the QPILE folder that comes with the User's Manual.



8.3.1 File header

The file header contains basic parameters of the table:

| Parameter | Purpose |
|----------------|--|
| PORTFOLIO | A table name: an alphanumerical identifier in Latin letters without spaces. |
| PORTFOLIO_EX | 'PORTFOLIO' is a parameter used in the first version of the language, 'PORTFOLIO_EX' is a program name for the extended language version (starting from QUIK 4.09) |
| DESCRIPTION | Text description of the table |
| * CLIENTS_LIST | A comma-separated list of client codes for which table values are calculated. ALL_CLIENTS means that all client codes are selected. Each table row contains values for a separate client account |
| FIRMS_LIST | A comma-separated list of firm (trader) identifiers whose clients have access to the table |

* – this parameter is used in the previous version of the language and is not required in the current version.

Each parameter must be described in a separate line and end with a semicolon (;).

Example of a header:

```
PORTFOLIO AVAILABLE_MONEY;  
DESCRIPTION Available cash of the client;  
CLIENTS_LIST ALL_CLIENTS;  
FIRMS_LIST MC0012300000;
```

This table will be shown under name AVAILABLE_MONEY in the list of available tables. When available table descriptions are selected, the 'Current script' field shows 'Available cash of the client'. The same name will be used as a default caption for the table. The parameters are calculated for all client accounts with the firm code MC0012300000 that are available for the user in the tables of positions.

8.3.2 Program body

This section contains the program code for calculation of values in table cells. The section begins with the string PROGRAM and ends with the string END_PROGRAM. To exit from the program body the RETURN statement is used.

- 1. The character case (upper or lower) in instructions is ignored by the interpreter. All string constants are automatically converted to uppercase. However, if necessary, the automatic conversion of characters to uppercase can be disabled. To do so, the following string has to be inserted between the header and the program body:**



Once this key is added, string variables will not be converted to upper case.

2. **The interpreter ignores multiple spaces, except for those inside string variables.**
3. **Long strings that do not fit into one line are split using combination '<space>_'. For example:**

```
CLIENTS_LIST 0001, 0002, 0003, 0004, 0005, 0006, 0007, 0008, 0009, 0010, _
0011, 0012, 0013, 0014, 0015;
```

4. **A single quotation mark (') is used to mark comments. A comment is effective to the end of the line.**
5. **Statements are separated by a carriage return. That is, each statement is described in an individual line.**
6. **The characters in a string are numbered beginning with 0.**

Program example:

```
PROGRAM
FirmCode = "MC0012300000"
CurrentBalance      = MONEY_CURRENT_BALANCE(ROWNAME, FirmCode, "EQTV", "SUR")
CurrentLimit        = MONEY_CURRENT_LIMIT(ROWNAME, FirmCode, "EQTV", "SUR")
Locked              = MONEY_LIMIT_LOCKED(ROWNAME, FirmCode, "EQTV", "SUR")
AvailableMoney      = MoneyCurrentBalance + MoneyCurrentLimit - MoneyLocked
If AvailableMoney > 0
Status = "Orders available"
Else
Status = "Orders unavailable"
SET_ROW_COLOR(ROWNAME, "RGB(255,138,138)", "DEFAULT_COLOR")
End If
END_PROGRAM
```

This example relates to the creation of the **Available cash of the client** table. The created table partially duplicates the fields from the **Cash positions** table and contains two calculated fields as well.

8.3.3 Definition of table columns

In order for the table to display calculated values, its columns must be described and the format of cells in each column defined. This is what this file section is intended for.



Each individual column of the table is described using the set of parameters given below:

| Parameter | Purpose |
|-----------------------|---|
| PARAMETER | Name of the program variable whose value will be shown in the given column (maximum length is 31 characters) |
| PARAMETER_TITLE | Column name shown in the table (maximum length is 32 characters) |
| PARAMETER_DESCRIPTION | Expanded description of a parameter (maximum length is 127 characters) |
| PARAMETER_TYPE | The data type of the cells in this column. Two data types can be used: <ul style="list-style-type: none">– NUMERIC(<number_of_digits>, <number_of_digits_after_the_point>) – double with floating point;– STRING(<string_length>) – string |

The description of each parameter ends with END.

The description of parameters must be followed by END_PORTFOLIO (in the first version of the language) or END_PORTFOLIO_EX (in the new version).

Example of description:

```
PARAMETER AvailableMoney;  
PARAMETER_TITLE Available;  
PARAMETER_DESCRIPTION Available cash of the client;  
PARAMETER_TYPE NUMERIC(10,2);  
END
```

The values of the variable 'AvailableMoney' will be shown in the **Available** column. During configuration, the **Parameter description** field shows 'Available cash of a client'. The column values will be displayed as numbers with two decimals. The same format is recommended for exporting data from the table via ODBC.

8.3.4 Including additional files

The section INCLUDE is used to include additional files with functions to the program. This section is located between the header and the body of the program and is described as follows:

```
INCLUDE file1, file2, ..., fileN;
```

where file1, file2, ..., fileN are relative or full paths to the files containing descriptions of functions (separated by commas).

Example of description:



8.4 QPILE language constructs

8.4.1 Data types

1. The following data types are used:

- STRING – string data;

| |
|-------------|
| 'Total bid' |
|-------------|

- DOUBLE – real or double data. Floating-point numbers accurate to 15 decimal places, but no more than 8 digits after the decimal point;

| |
|------------|
| 1234567.89 |
|------------|

- COLLECTION – collection;

Collection is a list of objects indexed using an integer key (starting from 0).

| 0 | 1 | 2 | 3 |
|--------|----|-------|-------|
| «HYDR» | 12 | 7.890 | «BUY» |

- MAP – associative array of data;

Associative Array (MAP) is a sequence of [key, value] pairs that allows for retrieving values by the key. Each key corresponds to only one value or, in other words, each key in an associative array is unique. A key is always a string.

| NUMBER | TIME | OPERATION |
|--------|----------------|-----------|
| 67890 | '12:34:56 , | BUY |

2. The COLLECTION and MAP data types are structural. They can:

- Be heterogeneous, that is, contain values of different types;
- Contain values of any types including COLLECTION and MAP variables.

3. All variables, except for formal parameters, have global scope defined by the execution context. If a variable was assigned a value in the process of execution, it will be considered as defined until the end of the program execution.

Variables retaining their values across program calculation intervals are called **global**. Global variables are described by the function `NEW_GLOBAL ()` (see [8.5.1](#)).

4. The number of variables in a program cannot exceed 1000.



8.4.2 Typcasting

1. Language variables have no types and can change the data type at runtime.
2. Applying operators '+', '-', '*', '/' to string variables converts them into real values. If the strings cannot be converted to real, the real value is assumed to be 0.0. The result of applying these operations to structural variables is undefined.
3. The comparison operation for real and string variables is allowed only for variables holding values of the same type. The result of comparison operation is not defined for COLLECTION and MAP variables.
4. The result of concatenating (operator &) real variables is a string.
5. Argument type conversion is always performed when external functions are called.
6. Values of structural type variables can be accessed and changed using special functions described in sections [8.7](#)–[8.8](#).

8.4.3 Expressions

1. Mathematical operations ('+', '-', '*', '/', and unary '-') are performed with a standard priority.
2. Boolean operations have equal priority (except AND and OR) and are executed from left to right. AND and OR operations are executed last, from left to right. For example:

A < B or A = C is equivalent to (A < B) or (A = C)

3. Parentheses can be used in expressions and comparisons.

Admissible expressions:

| Operation | Description |
|---------------------|---|
| Mathematical | |
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| Unary '-' | Value sign inversion |
| D or E | Exponential notation, 3D2 is equivalent to 3*10^2 |
| Boolean | |
| == | Equality |
| = | Equality, similar to the previous one |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal |



| Operation | Description |
|---------------|--|
| <= | Less than or equal |
| != | Not equal |
| <> | Not equal, similar to the previous one |
| AND | Boolean AND |
| OR | Boolean OR |
| String | |
| & | Joining or concatenation of strings |

Type casting must be used when assigning values to variables. For example, `A="3E2"+0` assigns number 300 to the variable, whereas `A="5E3"` assigns the string value "5E3".

8.4.4 Conditional statements

Conditional statement syntax:

```
IF condition
Sequence of instructions
ELSE
Sequence of instructions
END IF
```

A condition is a logical expression. The nesting depth of conditional statements is not restricted. The END IF statement can contain only a single space.

Example:

```
IF DealerMoney = 0
Margin = 100
ELSE
Margin=0
END IF
```

8.4.5 Loops

Loop statement syntax:

1. Executes a sequence of instructions for each variable value from the **list of values**. The **List of values** is defined by a STRING variable with a comma-separated list of values.



```
FOR variable IN list of values
Sequence of instructions
END FOR
```

2. Executes a sequence of instructions for each variable value in a range between 'value1' and 'value2' which can be represented by mathematical expressions, with a step equal to 1. If 'value2' < 'value1', the cycle is not processed.

```
FOR variable FROM value1 TO value2
Sequence of instructions
END FOR
```

The nesting depth of loop statements is not restricted. The END FOR statement can contain only a single space.

Example:

```
FkcbSecsList = "HYDR,SBER,MSNG,LKOH,YUKO,RTKM"

FOR Sec IN FkcbSecsList
DCPos = DEPO_CURRENT_BALANCE(ROWNAME, FirmList, Sec, DefDepoAcc)
DCLim = DEPO_CURRENT_LIMIT(ROWNAME, FirmList, Sec, DefDepoAcc)
DOLim = DEPO_OPEN_LIMIT(ROWNAME, FirmList, Sec, DefDepoAcc)
SecPos = DCPos + DCLim - DOLim
SecPos = SecPos * GET_PARAM(ClassCode, Sec, "LAST") * GET_PARAM(ClassCode, Sec,
"LOTSIZE")
DepoPos = DepoPos + ignore_negative(SecPos)
DealerMoney = DealerMoney + dealer(SecPos)
END FOR
```

This loop searches through all instruments from the «FkcbSecsList» list of instruments and processes the position for each instrument for the current client being estimated.

The BREAK statement is used to break the loop before its end. It breaks the execution of FOR and transfers control to the next statement.

The CONTINUE statement transfers execution control to the next iteration of FOR. In the FOR statement, the next iteration begins with evaluation of the FOR loop conditional expression. Following the evaluation of the conditional expression, the execution of the statement is either terminated or the statement's body is executed, depending on the computation result.



8.4.6 Functions

General:

1. Descriptions of functions may be located anywhere in the program.
2. All functions must have unique names, no overload by types or number of parameters is allowed.
3. Functions may return the value using a variable named RESULT.
4. All parameters are passed to functions by value.
5. A function may be a procedure. In this case, the returned value is not used at the place such function is called, so assigning the RESULT variable inside the function is not necessary.
6. The RETURN statement can be used to exit from the function's body.

1. **In the description of the syntax of functions returning or accepting a variable of any type, the type ANY is used for designation.**
2. **A table created in program calculation is designated as OWN.**

User-defined functions have the following syntax:

```
FUNC function (list of arguments)
Sequence of instructions
END FUNC
```

The END FUNC statement can contain only a single space.

Example:

```
FUNC ignore_negative(x)
If x > 0
RESULT = x
Else
RESULT = 0
End If
END FUNC
```

8.4.7 QPILE functions

Standard QPILE-language functions are intended for working with structural variables and obtaining values from the QUIK tables.

| Function | Purpose |
|------------|---------------------------------|
| NEW_GLOBAL | Initialises a global variable |
| MESSAGE | Outputs text in the Message Box |



| Function | Purpose |
|--|---|
| Mathematical functions | |
| ABS | Module |
| ACOS | Arc cosine |
| ASIN | Arc sine |
| ATAN | Arc tangent |
| CEIL | Rounding up |
| COS | Cosine |
| EXP | Exponent |
| FLOOR | Rounding down |
| LOG | Logarithm |
| POW | Raising to a power |
| RAND | Random number |
| RANDOMIZE | Random number generation |
| SIN | Sine |
| SQRT | Square root |
| TAN | Tangent |
| Functions for working with collections (COLLECTION) | |
| CREATE_COLLECTION | Defines a collection |
| GET_COLLECTION_COUNT | Returns the number of collection items |
| REMOVE_COLLECTION_ITEM | Removes a collection item |
| INSERT_COLLECTION_ITEM | Inserts a collection item |
| SET_COLLECTION_ITEM | Replaces the value of a collection item |
| GET_COLLECTION_ITEM | Reads the value of a collection item |
| Functions for working with associative arrays (MAP) | |
| CREATE_MAP | Defines an array |
| SET_VALUE | Adds a new element to the array |
| GET_VALUE | Reads an element from the array |
| Functions for accessing rows in arbitrary QUIK tables | |
| GET_ITEM | Returns the row with the specified number from a QUIK table |



| Function | Purpose |
|---------------|---|
| GET_NUMBER_OF | Returns the number of records in the specified QUIK table |

Functions for accessing a list of available parameters

| | |
|----------------------|--|
| GET_CLASSES_LIST | Returns a list of class codes available in the current session |
| GET_CLASS_SECURITIES | Returns a list of instrument codes for the specified list of classes |
| GET_SECURITY_INFO | Returns information for an instrument with the specified code from the specified class |

Functions for handling programmable tables

| | |
|------------------|--|
| ADD_ITEM | Adds a new row to the table |
| MODIFY_ITEM | Modifies the specified table row |
| DELETE_ITEM | Deletes the specified table row |
| DELETE_ALL_ITEMS | Deletes all values in the table |
| SET_ROW_COLOR | Sets the highlighting color for the table row |
| SET_ROW_COLOR_EX | Sets the background and font color for a table row |

Functions for getting values from the quotes table

| | |
|--------------|--|
| *GET_PARAM | Returns values from the Quotes table |
| GET_PARAM_EX | Returns all values from the Quotes table |

Functions for retrieving values from the Level II Quotes table

| | |
|--------------------------|---|
| GET_QUOTES_II_LEVEL_DATA | Returns values from the Level II Quotes table |
|--------------------------|---|

*Functions for retrieving values from the Positions in instruments table

| | |
|-----------------------------|---|
| DEPO_OPEN_BALANCE | Instruments opening balance |
| DEPO_OPEN_LIMIT | Instruments opening limit |
| DEPO_CURRENT_BALANCE | Instruments current balance |
| DEPO_CURRENT_LIMIT | Instruments current limit |
| DEPO_LIMIT_AVAILABLE | Number of available instruments |
| DEPO_LIMIT_LOCKED | Number of instrumentslots locked |
| DEPO_LIMIT_LOCKED_BUY | Number of instrumentslots locked to buy |
| DEPO_LIMIT_LOCKED_BUY_VALUE | Value of instrumentslocked to buy |



| Function | Purpose |
|--|--|
| * Functions for retrieving values from the Cash positions table | |
| MONEY_OPEN_BALANCE | Opening cash balance |
| MONEY_OPEN_LIMIT | Opening cash limit |
| MONEY_CURRENT_BALANCE | Current cash balance |
| MONEY_CURRENT_LIMIT | Current cash limit |
| MONEY_LIMIT_AVAILABLE | Available cash |
| MONEY_LIMIT_LOCKED | Amount of cash locked in buy orders |
| Functions for the calculation of margin positions | |
| SHORT_VALUE | Value of all short positions |
| LONG_VALUE | Value of all long positions |
| Functions for retrieving values from the Client Portfolio and Buy / Sell tables | |
| GET_CLIENT_MARGINAL_PORTFOLIO_INFO | Returns values of parameters in the Client Portfolio table |
| GET_CLIENT_MARGINAL_BUY_SELL_INFO | Return values of parameters in the Buy/Sell table |
| File handling functions | |
| CLEAR_FILE | Clears file |
| WRITE | Writes a string at the end of the file |
| WRITELN | Writes a string with a carriage return at the end of file |
| GET_FILE_LEN | Returns the number of rows in the file |
| READ_LINE | Reads the line with the specified number from the file |
| String handling functions | |
| LEN | Returns the number of characters in the string |
| TRIM | Trims spaces at the end of the string |
| SUBSTR | Returns a substring |
| FIND | Finds the substring position in the string |
| Chart handling functions | |
| GET_CANDLE | Return values of the candle prices, volumes and indicators (OHLCV) on the chart |
| GET_CANDLE_EX | Returns values of the candle prices, volumes and indicators (OHLCV) in the chart |



| Function | Purpose |
|---------------------------------|--|
| Order handling functions | |
| SEND_TRANSACTION | Entry of a new order |
| Label handling functions | |
| ADD_LABEL | Adds a label |
| DELETE_LABEL | Deletes a label |
| DELETE_ALL_LABELS | Deletes all labels |
| GET_LABEL_PARAMS | Returns label parameters |
| SET_LABEL_PARAMS | Sets label parameters |
| Service Functions | |
| GET_TRADE_DATE | Returns the date of the current trading session |
| GET_DATETIME | Returns the current date and time |
| APPLY_SCALE | Rounds with the specified accuracy |
| IS_CONNECTED | Determines the status of the connection between the client terminal and the server |
| GET_INFO_PARAM | Returns parameters for the information window (see menu System/About program/Information window) |
| BREAKPOINT | Breaks program execution and calls the Debug window |

* – functions from the previous version of the QPILE language retained for backward compatibility. Values returned by these functions can also be obtained by using function GET_ITEM to read a table row and function GET_VALUE to obtain a value from it

Functions from the previous versions no longer supported:

- MONEY_LIMIT_LOCKED_NONMARGINAL_VALUE.

8.5 General functions

8.5.1 NEW_GLOBAL

This function is intended for initialization of a global variable. A global variable retains its value across iterations of table values calculations. The initialization is performed by a variable of any type and results in a global variable of the corresponding type.

NEW_GLOBAL (STRING Name, ANY InitValue)



Parameters:

| No. | Parameter | Type | Description |
|-----|-----------|--------|---|
| 1 | Name | STRING | String name of the created variable |
| 2 | INITVALUE | ANY | Value for initialization of a global variable |

A string constant or a variable with a string value can be used as the first parameter. In the latter instance, the created global variable has the same name as the value of the string variable.

Example:

```
\nNEW_GLOBAL("GLOBAL","MyFirstGlobal")\nNEW_GLOBAL(Global,1)\n
```

Executing these two lines creates two global variables: A string variable named GLOBAL that has value MyFirstGlobal and a real type variable named MyFirstGlobal that has value 1.

8.5.2 MESSAGE

This function displays a Message Box with the specified text.

MESSAGE (STRING Text, DOUBLE Msg_type)

Parameters:

| No. | Parameter | Type | Description |
|-----|-----------|--------|---|
| 1 | TEXT | STRING | Message text |
| 2 | MSG_TYPE | DOUBLE | A message type defining the type of icon in the Window and a tone signal. Valid values: <ul style="list-style-type: none">– 1 – Information;– 2 – Attention;– 3 – Error |

Example:

```
\nMESSAGE ("Hello!",1)\n
```



8.6 Mathematical functions

8.6.1 ABS

Returns the absolute value of the number

DOUBLE ABS (DOUBLE Value)

8.6.2 ACOS

Returns the value of the argument's arc cosine.

DOUBLE ACOS (DOUBLE Value)

8.6.3 ASIN

Returns the value of the argument's arc sine.

DOUBLE ASIN (DOUBLE Value)

8.6.4 ATAN

Returns the value of the argument's arc tangent.

DOUBLE ATAN (DOUBLE Value)

8.6.5 CEIL

Returns the nearest integer greater than or equal to the argument.

DOUBLE CEIL (DOUBLE Value)

8.6.6 COS

Returns the value of the argument's cosine.

DOUBLE COS (DOUBLE Value)

8.6.7 EXP

Returns the argument's exponent.

DOUBLE EXP (DOUBLE Value)

8.6.8 FLOOR

Returns the nearest integer, smaller than the argument.

DOUBLE FLOOR (DOUBLE Value)

8.6.9 LOG

Returns the natural logarithm of the argument

DOUBLE LOG (DOUBLE Value)



8.6.10 POW

Raises the argument to a power.

DOUBLE POW (DOUBLE Value, DOUBLE Power)

8.6.11 RAND

Returns a random integer value in the range from 0 to 32767.

DOUBLE RAND ()

8.6.12 RANDOMIZE

Generation of random numbers.

DOUBLE RANDOMIZE ()

Initialises the random number generator to define a random sequence of the generated numbers. Function RANDOMIZE () must be called before the RAND() function is used.

8.6.13 SIN

Returns the value of the argument's sine.

DOUBLE SIN (DOUBLE Value)

8.6.14 SQRT

Returns the value of the argument's square root.

DOUBLE SQRT (DOUBLE Value)

8.6.15 TAN

Returns the value of the argument's tangent.

DOUBLE TAN (DOUBLE Value)

Example:

```
'  
MESSAGE ("ACOS 0.5 - " & acos(0.5),1)  
MESSAGE ("ASIN 0.5 - " & asin(0.5),1)  
MESSAGE ("ATAN 2 - " & atan(2),1)  
MESSAGE ("CEIL 2.2 - " & ceil(2.2),1)  
MESSAGE ("COS 0.5 - " & cos(0.5),1)  
MESSAGE ("EXP 2 - " & exp(2),1)  
MESSAGE ("FLOOR 4.5 - " & floor(4.5),1)  
MESSAGE ("LOG 0.5 - " & log(0.5),1)  
MESSAGE ("POW 2,3 - " & pow(2,3),1)  
MESSAGE ("RAND - " & rand(),1)  
MESSAGE ("SIN 0.5 - " & sin(0.5),1)  
MESSAGE ("SQRT 2 - " & sqrt(2),1)
```



```
MESSAGE ("TAN 0.5 - " &tan(0.5),1)
'
```

When the example is executed, a Message Box appears on the screen showing the function and the result of calculation of its value.

8.7 Functions for working with collections (COLLECTION)

Functions from this group are intended for working COLLECTION-type variables.

Collection is a list of objects indexed using an integer key (starting from 0). A collection may contain elements of any type including COLLECTION-type variables. A collection may be non-homogeneous, that is, contain objects of different types. In the current implementation, a collection element identified by an index is accessed in linear time.

8.7.1 CREATE_COLLECTION

The first function creates an empty collection and the second one is a copy constructor.

```
COLLECTION CREATE_COLLECTION ()
```

```
COLLECTION CREATE_COLLECTION (COLLECTION IntValue)
```

8.7.2 GET_COLLECTION_COUNT

Returns the number of elements in the collection.

```
DOUBLE GET_COLLECTION_COUNT (COLLECTION Name)
```

8.7.3 REMOVE_COLLECTION_ITEM

Removes the element indexed with index from the Name collection.

```
COLLECTION REMOVE_COLLECTION_ITEM (COLLECTION Name, DOUBLE index)
```

8.7.4 INSERT_COLLECTION_ITEM

Inserts variable value as an element indexed with index.

```
COLLECTION INSERT_COLLECTION_ITEM (COLLECTION Name, DOUBLE index,  
ANY value)
```

When the element indexed with "index" is inserted, the existing collection element indexed with "index" is shifted into the place of the element indexed with "index+1", and so on for all existing elements with indices equal to or greater than the value of the insertion index. See example in [8.7.6](#).

8.7.5 SET_COLLECTION_ITEM

Replaces the value of the element indexed with "index" with the "value".



COLLECTION SET_COLLECTION_ITEM (COLLECTION Name, DOUBLE index, ANY value)

8.7.6 GET_COLLECTION_ITEM

Returns the value of the element indexed with "index".

ANY GET_COLLECTION_ITEM (COLLECTION Name, DOUBLE index)

Parameters:

| No. | Parameter | Type | Description |
|-----|-----------|------------|--|
| 1 | INTVALUE | COLLECTION | Collection for initialization of a newly created object |
| 2 | Name | STRING | Collection name |
| 3 | INDEX | DOUBLE | Index for addressing a collection. It must have an integer value |
| 4 | VALUE | ANY | An arbitrary type value for inclusion in the collection |

Example:

```
\n
col=CREATE_COLLECTION()
FOR i FROM 0 TO 9
col=INSERT_COLLECTION_ITEM(col,0,0)
col=SET_COLLECTION_ITEM(col,0,i)
END FOR
s = GET_COLLECTION_ITEM(col,5)
len = GET_COLLECTION_COUNT(col)
\
```

An empty collection is created. Then, a null element with value "0" is inserted 10 times in the collection and immediately its value changes to the value of the loop variable. As a result, a collection of 10 elements is created. Upon execution of the program, "s"=4, "len"=10, and "col" has the following form:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

8.8 Functions for working with associative arrays (MAP)

Functions from this group are intended for working with MAP-type variables.



Associative Array (MAP) is a sequence of [key, value] pairs used to get values by the key. Only a single value corresponds to each key. An associative array may contain elements of any type including structural variables as well. Different elements of such array may contain various types of objects as values. A key is always a string. In the current implementation, access by a key to a collection element is performed in linear time.

8.8.1 CREATE_MAP

The first function creates an empty array and the second one is a copy constructor.

```
MAP CREATE_MAP ()
MAP CREATE_MAP (MAP IntValue)
```

8.8.2 SET_VALUE

This function adds the element having a value and a key named "Value" and "Key", respectively, to the array "Name." If the array contains an element with the key named "Key," the value of such element will be changed to "Value." The function returns a modified array.

```
MAP SET_VALUE(MAP Name, STRING Key, ANY Value)
```

8.8.3 GET_VALUE

This function returns the value of the element with a key named "Key" from the array "Name." If the element is not found, the function returns an empty string.

```
STRING GET_VALUE(MAP Name, STRING Key)
```

Parameters:

| No. | Parameter | Type | Description |
|-----|-----------|--------|---|
| 1 | INTVALUE | MAP | Array for initializing the created object |
| 2 | NAME | STRING | Array name |
| 3 | KEY | STRING | Value of the key used to insert or access an element in the array |
| 4 | VALUE | ANY | An arbitrary value for inclusion into the array |

Example:

```
`
map=CREATE_MAP ()
FOR i FROM 0 TO 9
map=SET_VALUE(map, "key" & i, i)
END FOR
s = GET_VALUE(map, "key5")
`
```



An empty array is created. Then, in a loop, elements that have the value equal to the loop variable and the key of the "'key' & i" type are inserted into the array. After the program executes, "s"=5.

This function returns a **STRING value, unless it is not explicitly converted into a **DOUBLE**-type value.**

For example:

```
RES=0+GET_VALUE
```

8.9 Functions for accessing rows in arbitrary QUIK tables

Functions from this group are intended for accessing data in QUIK workstation tables.

8.9.1 GET_ITEM

This function returns an associative array (MAP) containing data from the string with the number "Index" from the table named TableName.

MAP GET_ITEM(STRING TableName,DOUBLE Index)

The returned array contains values of table cells from the client terminal's table as its elements, whose keys are the names of the columns. Valid values of the TableName field and the keys are shown in tables below. Values of the keys for the programmable table **OWN** correspond to the names of columns specified in the parameter description section.

8.9.2 GET_NUMBER_OF

This function returns the number of records in the TableName table.

DOUBLE GET_NUMBER_OF(STRING TableName)

Parameters:

| No. | Parameter | Type | Description |
|-----|------------|--------|--|
| 1 | TABLERNAME | STRING | QUIK table name |
| 2 | INDEX | DOUBLE | Index for addressing a collection. It must have an integer value |

Example:

```
\n
n=GET_NUMBER_OF("ALL_TRADES")
value=0
FOR i FROM 1 to n
```



```
trade = GET_ITEM ("ALL_TRADES ", i)
value = value + GET_VALUE (trade, "VALUE")
END FOR
`
```

In this example, first the number of records in the **Time and Sales** table (anonymous trades) is queried, then a loop is executed in which a corresponding MAP is created for each record from which a value identified by the VALUE key is retrieved. As a result, the variable named "value" contains the total volume of anonymous trades for the moment.

8.9.3 Descriptions of tables and parameters

1. Tables used in the GET_NUMBER_OF and GET_ITEM functions:

| Table Name | Table |
|-------------------------|---|
| ORDERS | Table of orders |
| STOP_ORDERS | Table of stop orders |
| TRADES | Table of trades |
| ALL_TRADES | Time and Sales |
| MONEY_LIMITS | Cash positions |
| DEPO_LIMITS | Positions in instruments |
| FUTURES_CLIENT_HOLDINGS | Positions of client accounts (futures) |
| FUTURES_CLIENT_LIMITS | Client account limits (futures) |
| NEG_DEALS | Table of negdeal orders |
| NEGOTIATION_TRADES | Table of trades for execution |
| NEG_DEAL_REPORTS | Table of negotiated deal orders / reports |
| POSITIONS | Participant's cash positions |
| FIRM_HOLDING | Participant's positions in instruments |
| ACCOUNT_BALANCE | Participant's positions on trading accounts |
| OWN | Table created by program calculations |

2. Description of parameters from the **Orders Table** returned by GET_ITEM:

| No. | Parameter | Type | Description |
|-----|---------------|--------|---|
| 1 | NUMBER | DOUBLE | Number of the order in the trading system |
| 2 | EXCHANGE_CODE | STRING | Exchange code in the trading system |



| No. | Parameter | Type | Description |
|-----|-----------------|--------|---|
| 3 | DATE | DOUBLE | Date of order entry |
| 4 | TIME | DOUBLE | Time of order entry |
| 5 | ACTIVATION_TIME | DOUBLE | Time of activation |
| 6 | WITHDRAW_TIME | DOUBLE | Order kill time |
| 7 | SECURITY | STRING | Short name of the instrument |
| 8 | SECCODE | STRING | Instrument code in the order |
| 9 | CLASS | STRING | Short name of the instrument class |
| 10 | CLASSCODE | STRING | Order class code |
| 11 | OPERATION | STRING | Operation. Valid values: SELL or BUY |
| 12 | ACCOUNT | STRING | Trading account |
| 13 | PRICE | DOUBLE | Price |
| 14 | QUANTITY | DOUBLE | Quantity in lots |
| 15 | BALANCE | DOUBLE | Balance |
| 16 | VALUE | DOUBLE | Value in cash |
| 17 | TRADE_CURRENCY | STRING | Order currency |
| 18 | YIELD | DOUBLE | Yield |
| 19 | ACCRUEDINT | DOUBLE | Accrued coupon income |
| 20 | USERID | STRING | Trader's ID |
| 21 | FIRMID | STRING | Firm ID |
| 22 | CLIENTCODE | STRING | Client code |
| 23 | COMMENT | STRING | Comment |
| 24 | STATUS | STRING | Order status. Valid values: ACTIVE, KILLED or FILLED |
| 25 | TYPE | STRING | Order type. Sequence of three characters: <ul style="list-style-type: none"> – 1st: L - limit or M - market; – 2nd: S - settlement at any price or O - settlement at one price; – 3rd: N - fill or kill, W - withdraw / kill balance, or <space> - unconditional |
| 26 | TRANS_ID | DOUBLE | Transaction ID |
| 27 | SETTLECODE | STRING | Settlement code |
| 28 | PRICE2 | DOUBLE | Buyback price |



| No. | Parameter | Type | Description |
|-----|----------------------------|--------|--|
| 29 | IS_MARKET_MAKE R_ORDER | STRING | Order of a market maker. Valid values: YES or <space> |
| 30 | SESSION_DATE | DOUBLE | Date of the current trading session |
| 31 | TIME_MICROSEC | DOUBLE | Number of microseconds in the order placement period |
| 32 | WITHDRAW_DATE | DOUBLE | Order kill date |
| 33 | WITHDRAW_TIME_ MICROSEC | DOUBLE | Number of microseconds in the order kill period |
| 34 | PERIOD | DOUBLE | Trading session period. Valid values: _ 0 – opening; _ 1 – regular; _ 2 – closing |
| 35 | VISIBLE_QUANTIT Y | DOUBLE | Visible quantity. This parameter is used for “Iceberg” type orders. |
| 36 | LINKED_ORDER | DOUBLE | Order number in the trading system |
| 37 | SEC_CURRENCY | STRING | Settlement currency |
| 38 | EXPIRE_DATE | DOUBLE | Term |
| 39 | UID | DOUBLE | UID |

3. Description of parameters from the **Trades Table** returned by GET_ITEM:

| № | Parameter | Type | Description |
|----|---------------|--------|--|
| 1 | NUMBER | DOUBLE | Number of the trade in the trading system |
| 2 | EXCHANGE_CODE | STRING | Exchange code in the trading system |
| 3 | DATE | DOUBLE | Execution date |
| 4 | TIME | DOUBLE | Execution time |
| 5 | ORDER_NUMBER | DOUBLE | Number of the order in the trading system |
| 6 | SECURITY | STRING | Short name of the instrument |
| 7 | SECCODE | STRING | Instrument code |
| 8 | CLASS | STRING | Short name of the class |
| 9 | CLASSCODE | STRING | Class code |
| 10 | OPERATION | STRING | Operation. Valid values: _ SELL; _ BUY |



| № | Parameter | Type | Description |
|----------|---------------------------|-------------|---|
| 11 | TYPE | STRING | Type of trade. Valid value: MARGIN - margin trade |
| 12 | ACCOUNT | STRING | Trading account |
| 13 | PRICE | DOUBLE | Price |
| 14 | QUANTITY | DOUBLE | Quantity in lots |
| 15 | VALUE | DOUBLE | Value in cash |
| 16 | TRADE_CURRENCY | STRING | Currency |
| 17 | SETTLE_CURRENCY | STRING | Settlement currency |
| 18 | SETTLE_CODE | STRING | Settlement code |
| 19 | YIELD | DOUBLE | Yield |
| 20 | ACCRUEDINT | DOUBLE | Accrued coupon income |
| 21 | USERID | STRING | Trader's ID |
| 22 | STATION_ID | STRING | Workstation ID |
| 23 | FIRMID | STRING | Dealer's ID |
| 24 | FIRMNAME | STRING | Trader's firm's ID |
| 25 | CLIENTCODE | STRING | Client code |
| 26 | COMMENT | STRING | Comment |
| 27 | PARTNER_FIRMID | STRING | Partner's firm's ID |
| 28 | PARTNER_FIRM_NAME | STRING | Partner's firm's name |
| 29 | PRICE2 | DOUBLE | Buyback price |
| 30 | REPORATE | DOUBLE | REPO rate (%) |
| 31 | TS_COMISSION | DOUBLE | Trading system commission |
| 32 | CLEARING_ COMISSION | DOUBLE | Clearing commission (MOEX) |
| 33 | EXCHANGE_ COMISSION | DOUBLE | Stock exchange commission (MOEX) |
| 34 | TECH_CENTER_ COMISSION | DOUBLE | Technical center commission (MOEX) |
| 35 | ACCRUED2 | DOUBLE | Accrued interest (%) at the date of buyback |
| 36 | REPOVALUE | DOUBLE | REPO value |
| 37 | REPO2VALUE | DOUBLE | REPO buyback value |



| Nº | Parameter | Type | Description |
|----|------------------|--------|---|
| 38 | REPOTERM | DOUBLE | REPO term |
| 39 | START_DISCOUNT | DOUBLE | Initial discount (%) |
| 40 | LOWER_DISCOUNT | DOUBLE | Lower discount (%) |
| 41 | UPPER_DISCOUNT | DOUBLE | Upper discount (%) |
| 42 | BLOCK_SECURITIES | STRING | Block instruments. Valid values: <ul style="list-style-type: none"> – YES; – NO |
| 43 | SESSION_DATE | DOUBLE | Date of the current trading session |
| 44 | TIME_MICROSEC | DOUBLE | Number of microseconds in the order execution period |
| 45 | PERIOD | DOUBLE | Trading session period. Valid values: <ul style="list-style-type: none"> – 0 – opening; – 1 – regular; – 2 – closing |
| 46 | KIND | DOUBLE | Type of trade. Valid values: <ul style="list-style-type: none"> – 1 – regular; – 2 – targeted; – 3 – initial placement; – 4 – cash / instruments transfer; – 5 – targeted trade of the first REPO leg; – 6 – swap transaction settlement trade; – 7 – OTC swap transaction settlement trade; – 8 – dual currency basket settlement trade; – 9 – OTC dual currency basket settlement trade; – 10 – CC REPO transaction trade; – 11 – first leg of a CC REPO transaction trade; – 12 – second leg of a CC REPO transaction trade; – 13 – CC REPO transaction targeted trade; – 14 – first leg of a CC REPO transaction targeted trade; – 15 – second leg of a CC REPO transaction targeted trade; – 16 – CC REPO transaction asset returning technical trade; – 17 – Futures spread negotiation trade with same underlying loan and different expiry dates; – 18 – First part of tech negotiation trade for futures spread; – 19 – Second part of tech negotiation trade for futures spread; – 20 – First part of REPO negotiation trade with basket; – 21 – Second part of REPO negotiation trade with basket; – 22 – Derivatives market positions roll-over; – 23 – Late correction – XLON; – 24 – Not to mark – XLON; – 25 – Previous Day Contra; – 26 – Ordinary trade immediate publication – |

| № | Parameter | Type | Description |
|---|-----------|------|--|
| | | | <ul style="list-style-type: none"> 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 MTF TBA 1; – 39 – Inter fund cross - delayed publication requested MTF TBA 1; – 40 – Cancellation of OTC trade after date of publication MTF TBA 1; – 41 – OTC MTF TBA 2; – 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; |
| | | | <ul style="list-style-type: none"> – 49 – OTC MTF TBA 4; – 50 – OTC trade – delayed publication MTF TBA 4; – 51 – Inter fund cross - delayed publication requested MTF TBA 4; – 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. |



| No | Parameter | Type | Description |
|----|-----------|------|--|
| | | | <ul style="list-style-type: none"> _ 59 – FX Non-deliverable Swap trade; _ 60 – FX Spot trade; _ 61 – FX Non-deliverable forward trade; _ 62 – FX deposits trade; _ 63 – FX Forward trade |

4. Description of parameters from the **Time and Sales Table** retrieved by GET_ITEM:

| No. | Parameter | Type | Description |
|-----|---------------|--------|--|
| 1 | NUMBER | DOUBLE | Number for the transaction in the trading system |
| 2 | DATE | DOUBLE | Date of order entry |
| 3 | TIME | DOUBLE | Time of order entry |
| 4 | SECURITY | STRING | Short name of the instrument |
| 5 | SECCODE | STRING | Instrument code |
| 6 | CLASS | STRING | Short name of the class |
| 7 | CLASSCODE | STRING | Class code |
| 8 | PRICE | DOUBLE | Price |
| 9 | QUANTITY | DOUBLE | Quantity in lots |
| 10 | VALUE | DOUBLE | Value in cash |
| 11 | OPERATION | STRING | Direction of operation. Valid values: <ul style="list-style-type: none"> _ SELL; _ BUY |
| 12 | ACCRUEDINT | DOUBLE | Accrued coupon income |
| 13 | YIELD | DOUBLE | Yield |
| 14 | SETTLE_CODE | STRING | Settlement code |
| 15 | REPORATE | DOUBLE | REPO rate (%) |
| 16 | REPOVALUE | DOUBLE | REPO value |
| 17 | REPO2VALUE | DOUBLE | REPO buyback value |
| 18 | REPOTERM | DOUBLE | REPO term |
| 19 | SESSION_DATE | DOUBLE | Date of the current trading session |
| 20 | TIME_MICROSEC | DOUBLE | Number of microseconds in the order execution period |



| No. | Parameter | Type | Description |
|-----|-----------|--------|---|
| 21 | PERIOD | DOUBLE | Trading session period. Valid values: <ul style="list-style-type: none"> _ 0 – opening; _ 1 – regular; _ 2 – closing |

5. Description of parameters from the **Stop Orders Table returned by GET_ITEM:**

| No. | Parameter | Type | Description |
|-----|---------------------|--------|--|
| 1 | NUMBER | DOUBLE | Registration number of the stop order on the QUIK server |
| 2 | DATE | DOUBLE | Date of order entry |
| 3 | TIME | DOUBLE | Time of order entry |
| 4 | WITHDRAW_TIME | DOUBLE | Order kill time |
| 5 | STOP_ORDER_TYPE | DOUBLE | Stop order type. Valid values: <ul style="list-style-type: none"> _ 1 – stop-limit; _ 2 – stop price for a different instrument; _ 3 – contingent order; _ 6 – take-profit; _ 7 – 'if done' stop-limit; _ 8 – 'if done' take-profit; _ 9 – take-profit and stop-limit |
| 6 | TYPE | STRING | Order type. Sequence of three characters: <ul style="list-style-type: none"> _ 1st: L - limit or M – market; _ 2nd: S - settlement at any price or O - settlement at one price; _ 3rd: N - fill or kill, W - withdraw / kill balance, or <space> - unconditional |
| 7 | SECURITY | STRING | Short name of the instrument |
| 8 | SECCODE | STRING | Instrument code |
| 9 | CLASS | STRING | Short name of the class |
| 10 | CLASSCODE | STRING | Class code |
| 11 | OPERATION | STRING | Operation. Valid values: SELL or BUY |
| 12 | ACCOUNT | STRING | Trading account |
| 13 | CONDITION_SECURITY | STRING | Stop price instrument |
| 14 | CONDITION_SECCODE | STRING | Stop-price instrument code |
| 15 | CONDITION_CLASS | STRING | Class of the stop price |
| 16 | CONDITION_CLASSCODE | STRING | Class code for the stop price |



| No. | Parameter | Type | Description |
|-----|--------------------------------------|--------|---|
| 17 | CONDITION | STRING | Stop price direction. Valid values: LESS_OR_EQUAL_VALUE or GREATER_OR_EQUAL_VALUE |
| 18 | CONDITION_PRICE | DOUBLE | Stop price |
| 19 | CONDITION2 | STRING | Stop-limit price direction (for take-profit and stop-limit orders). Valid values: LESS_OR_EQUAL_VALUE or GREATER_OR_EQUAL_VALUE |
| 20 | CONDITION_PRICE2 | DOUBLE | Stop-limit price (for take-profit and stop-limit orders) |
| 21 | PRICE | DOUBLE | Price |
| 22 | MARKET_STOP_LIMIT | STRING | Stop-limit order settlement at market price (for take-profit and stop-limit orders). Valid values: YES or <space> |
| 23 | QUANTITY | DOUBLE | Quantity in lots |
| 24 | BALANCE | DOUBLE | Active volume |
| 25 | FILLED_VOLUME | DOUBLE | Volume filled |
| 26 | FIRMID | STRING | Dealer |
| 27 | UID | DOUBLE | UID |
| 28 | CLIENTCODE | STRING | Client code |
| 29 | COMMENT | STRING | Comment |
| 30 | LINKED_ORDER | DOUBLE | Number in the trading system for an order placed after the stop price condition occurs |
| 31 | ALL_TRADE_NUMBER | DOUBLE | Conditional trade |
| 32 | EXPIRE_DATE | DOUBLE | Term |
| 33 | EXPIRY_DATE_IS_TODAY | STRING | Expiry term is 'Today' |
| 34 | ACTIVE_IN_TIME_INTERVAL | STRING | Take-profit and stop-limit orders active during a time interval. Valid values: YES or <space> |
| 35 | ACTIVE_FROM_TIME | DOUBLE | Beginning of the time interval for a take-profit or stop-limit order |
| 36 | ACTIVE_TO_TIME | DOUBLE | End of the time interval for a take-profit or stop-limit order |
| 37 | USE_BASE_ORDER_BALANCE | STRING | Use the primary order balance for an entered stop order (for 'if done' orders). Valid values: YES or <space> |
| 38 | KILL_IF_LINKED_ORDER_PARTLY_FILLED | STRING | Cancel a stop order if the contingent order is partially filled (for 'if done' orders). Valid values: YES or <space> |
| 39 | ACTIVATE_IF_BASE_ORDER_PARTLY_FILLED | STRING | Enable if the primary order is partially filled (for 'if done' orders). Valid values: YES or <space> |



| No. | Parameter | Type | Description |
|-----|------------------------|--------|--|
| 40 | TYPE_DESCRIPTION | STRING | Type |
| 41 | STATUS | STRING | Order status. Valid values: ACTIVE, KILLED or FILLED |
| 42 | RESULT_DESCRIPTION | STRING | Result. Valid values: |
| | N | | <ul style="list-style-type: none"> – Rejected by TS; – Limit check failed; – Contingent order killed; – Contingent order filled; – Calculate min / max; – Awaiting activation; – Calculate min / max and await activation; – Killed; – Order sent to TS |
| 43 | CO_ORDER_NUMBER | DOUBLE | Contingent order |
| 44 | CO_ORDER_PRICE | DOUBLE | Price of contingent order |
| 45 | TRANS_ID | DOUBLE | Transaction ID |
| 46 | OFFSET | DOUBLE | Offset from min / max |
| 47 | OFFSET_UNITS | STRING | Offset units. Valid values: % or M |
| 48 | SPREAD | DOUBLE | Protective spread |
| 49 | USE_SPREAD_AS_PERCENTS | STRING | Protective spread as a percentage. Valid values: YES or <space> |
| 50 | MARKET_TAKE_PROFIT | STRING | Take-profit order settlement at market price (for take-profit and stop-limit orders). Valid values: |
| 51 | CO_ORDER_NUMBER | DOUBLE | Primary order |
| 52 | OWNER_SERVER | STRING | Server. Valid values: Other or Current |

6. Description of parameters from the **Cash positions table** returned by GET_ITEM:

| No. | Parameter | Type | Description |
|-----|-----------------|--------|----------------------|
| 1 | FIRMID | STRING | Firm ID |
| 2 | CURRCODE | STRING | Currency code |
| 3 | TAG | STRING | Calculation tag |
| 4 | CLIENT_CODE | STRING | Client code |
| 5 | OPEN_BALANCE | DOUBLE | Opening cash balance |
| 6 | OPEN_LIMIT | DOUBLE | Opening cash limit |
| 7 | CURRENT_BALANCE | DOUBLE | Current cash balance |



| No. | Parameter | Type | Description |
|-----|---------------|--------|--|
| 8 | CURRENT_LIMIT | DOUBLE | Current cash limit |
| 9 | LOCKED | DOUBLE | Locked value |
| 10 | AVAILABLE | DOUBLE | Available value |
| 11 | LOCKED_VALUE | DOUBLE | Cash value locked for the purchase of non-margin instruments |
| 12 | LIMIT_KIND | DOUBLE | Position on date |

7. Description of parameters from the **Positions in instruments table returned by GET_ITEM:**

| No. | Parameter | Type | Description |
|-----|-------------------|--------|---------------------------------|
| 1 | FIRMID | STRING | Firm ID |
| 2 | SECCODE | STRING | Instrument code |
| 3 | TRDACCID | STRING | Depo account |
| 4 | CLIENT_CODE | STRING | Client code |
| 5 | OPEN_BALANCE | DOUBLE | Opening balance for instruments |
| 6 | OPEN_LIMIT | DOUBLE | Opening limit for instruments |
| 7 | CURRENT_BALANCE | DOUBLE | Current balance for instruments |
| 8 | CURRENT_LIMIT | DOUBLE | Current limit for instruments |
| 9 | LOCKED_SELL | DOUBLE | Locked value |
| 10 | AVAILABLE | DOUBLE | Available quantity |
| 11 | WA_POSITION_PRICE | DOUBLE | Purchase price |
| 12 | LIMIT_KIND | DOUBLE | Position on date |

8. Description of parameters from the **Table of Limits for Client Accounts retrieved by GET_ITEM:**

| No. | Parameter | Type | Description |
|-----|-----------|--------|--|
| 1 | FIRMID | STRING | Firm ID |
| 2 | TRDACCID | STRING | Trading account |
| 3 | TYPE | STRING | Type of limit. Valid values: – Cash; – Total |



| No. | Parameter | Type | Description |
|-----|------------------------|--------|--|
| 4 | CBP_PREV_LIMIT | DOUBLE | Previous limit for open positions |
| 5 | CBPLIMIT | DOUBLE | Limit for open positions |
| 6 | CBPLUSED | DOUBLE | Current net positions |
| 7 | CBPLUSED_FOR_ORDERS | DOUBLE | Current net positions (for orders) |
| 8 | CBPLUSED_FOR_POSITIONS | DOUBLE | Current net positions (for open positions) |
| 9 | CBPLPLANNED | DOUBLE | Planned net positions |
| 10 | VARMARGIN | DOUBLE | Variation margin |
| 11 | ACCRUEDINT | DOUBLE | Accrued interest |
| 12 | OPTIONS_PREMIUM | DOUBLE | Options premium |
| 13 | TS_COMISSION | DOUBLE | Exchange fees |
| 14 | KGO | DOUBLE | Client's collateral coefficient |

9. Description of parameters from the **Client Account Positions Table** returned by GET_ITEM:

| No. | Parameter | Type | Description |
|-----|----------------|--------|---|
| 1 | FIRMID | STRING | Firm ID |
| 2 | TRDACCID | STRING | Trading account |
| 3 | SECCODE | STRING | Futures contract code |
| 4 | SEC_SHORT_NAME | STRING | Short name of the contract |
| 5 | TYPE | STRING | Type of limit. Valid values are: Main account, Clients and additional accounts, All traders' accounts, or <space> |
| 6 | START_BUY | DOUBLE | Opening long positions |
| 7 | START_SELL | DOUBLE | Opening short positions |
| 8 | START_NET | DOUBLE | Opening net positions |
| 9 | TODAY_BUY | DOUBLE | Current long positions |
| 10 | TODAY_SELL | DOUBLE | Current short positions |
| 11 | TOTAL_NET | DOUBLE | Current net positions |
| 12 | OPEN_BUYS | DOUBLE | Open buys |



| No. | Parameter | Type | Description |
|-----|---------------|--------|-----------------------------------|
| 13 | OPEN_SELLS | DOUBLE | Open sells |
| 14 | CBPLUSED | DOUBLE | Estimate of current net positions |
| 15 | CBPLPLANNED | DOUBLE | Planned net positions |
| 16 | VARMARGIN | DOUBLE | Variation margin |
| 17 | AVRPOSNPRICE | DOUBLE | Effective price of positions |
| 18 | POSITIONVALUE | DOUBLE | Position value |

10. Description of parameters from the **Table of negotiated deal orders** returned by GET_ITEM:

| No. | Parameter | Type | Description |
|-----|-----------------|--------|-----------------------------|
| 1 | NUMBER | DOUBLE | Number |
| 2 | QUOTENO | DOUBLE | Counter quote |
| 3 | DATE | STRING | Order entry date |
| 4 | TIME | STRING | Order entry time |
| 5 | ACTIVATION_DATE | DOUBLE | Order activation date |
| 6 | ACTIVATION_TIME | DOUBLE | Order activation time |
| 7 | SECURITY | STRING | Short name of an instrument |
| 8 | SECCODE | STRING | Instrument code |
| 9 | CLASS | STRING | Class |
| 10 | CLASSCODE | STRING | Class code |
| 11 | OPERATION | STRING | Operation: Buy or Sell |
| 12 | ACCOUNT | STRING | Account |
| 13 | PRICE | DOUBLE | Price |
| 14 | QUANTITY | DOUBLE | Quantity |
| 15 | USERID | STRING | Trader |
| 16 | FIRMID | STRING | Dealer ID |
| 17 | FIRMNAME | STRING | Trader's firm's ID |
| 18 | UID | DOUBLE | UID |
| 19 | CPUSERID | STRING | Partner's trader |



| No. | Parameter | Type | Description |
|-----|------------------|--------|---|
| 20 | CPFIRMID | STRING | Partner's ID |
| 21 | CPFIRMNAME | STRING | Partner firm |
| 22 | CLIENTCODE | STRING | Client code |
| 23 | COMMENT | STRING | Comment |
| 24 | MATCH_REFERENCE | STRING | Reference |
| 25 | STATUS | STRING | Status: Active, Filled or Killed |
| 26 | SETTLE_CODE | STRING | Settlement code |
| 27 | DIRECTION | STRING | Direction: Sent, Received, or Sent and received |
| 28 | YIELD | DOUBLE | Yield |
| 29 | VALUE | DOUBLE | Value |
| 30 | ACCRUEDINT | DOUBLE | Coupon % |
| 31 | PRICE2 | DOUBLE | Coupon yield |
| 32 | REFUNDRATE | DOUBLE | Refund rate (%) |
| 33 | REPORATE | DOUBLE | Repo rate (%) |
| 34 | TRANS_ID | DOUBLE | Transaction ID |
| 35 | REPOVALUE | DOUBLE | REPO value |
| 36 | REPO2VALUE | DOUBLE | REPO buyback value |
| 37 | REPOENTRY | STRING | REPO order entry type. Valid values: Price1 + Rate, Rate + Price2, Price1 + Price2, REPO total + Volume, REPO total + Discount, Volume + Discount, REPO Total, Volume |
| 38 | REPOTERM | DOUBLE | REPO term |
| 39 | START_DISCOUNT | DOUBLE | Initial discount (%) |
| 40 | LOWER_DISCOUNT | DOUBLE | Lower discount (%) |
| 41 | UPPER_DISCOUNT | DOUBLE | Upper discount (%) |
| 42 | BLOCK_SECURITIES | STRING | Block instruments. Valid values: Yes or No |
| 43 | ORIG_REPOVALUE | DOUBLE | Original REPO value |
| 44 | ORIG_VOLUME | DOUBLE | Original volume |
| 45 | ORIG_DISCOUNT | DOUBLE | Original discount |
| 46 | WITHDRAW_TIME | DOUBLE | Order kill time |

| No. | Parameter | Type | Description |
|-----|-----------------|--------|---------------------|
| 47 | BALANCE | DOUBLE | Balance |
| 48 | SETTLE_CURRENCY | STRING | Settlement currency |

11. Description of parameters from the **Table of Trades for Execution** returned by GET_ITEM:

| No. | Parameter | Type | Description |
|-----|--------------|--------|--|
| 1 | NUMBER | DOUBLE | Number |
| 2 | ORDER_NUMBER | DOUBLE | Order number |
| 3 | DATE | STRING | Trading date |
| 4 | SETTLEDATE | STRING | Settlement date |
| 5 | CLASS | STRING | Class |
| 6 | CLASSCODE | STRING | Class code |
| 7 | SECCODE | STRING | Instrument code |
| 8 | SECURITY | STRING | Short name of an instrument |
| 9 | OPERATION | STRING | Operation: Buy or Sell |
| 10 | CLIENTCODE | STRING | Client code |
| 11 | COMMENT | STRING | Comment |
| 12 | FIRMID | STRING | Dealer ID |
| 13 | FIRMNAME | STRING | Trader's firm ID |
| 14 | ACCOUNT | STRING | Depo account |
| 15 | CPFIRMNAME | STRING | Partner's name |
| 16 | CPFIRMID | STRING | Partner's ID |
| 17 | CPACCOUNT | STRING | Partner's depo account |
| 18 | PRICE | DOUBLE | Price |
| 19 | QUANTITY | DOUBLE | Quantity |
| 20 | VALUE | DOUBLE | Volume |
| 21 | STATUS | STRING | Status. Valid values: <ul style="list-style-type: none"> – FILLED – executed; – NOT FILLED – not executed; – INCLUDE IN REPORT – items included in the report |
| 22 | ACCRUEDINT | DOUBLE | Coupon yield |



| No. | Parameter | Type | Description |
|-----|----------------------|--------|--|
| 23 | PRICE1 | DOUBLE | Price of the first part of REPO |
| 24 | PRICE2 | DOUBLE | Buyback price |
| 25 | REPORTTRADENO | DOUBLE | Number of the trade for the first part of REPO |
| 26 | REPORATE | DOUBLE | REPO rate (%) |
| 27 | SETTLE_CODE | STRING | Settlement code |
| 28 | REPORT_NUM | DOUBLE | Report |
| 29 | CPREPORT_NUM | DOUBLE | Partner's report |
| 30 | TS_COMISSION | DOUBLE | Trading system commission |
| 31 | BALANCE | DOUBLE | Balance |
| 32 | SETTLETIME | STRING | Settlement time |
| 33 | AMMOUNT | DOUBLE | Amount of liability |
| 34 | REPOVALUE | DOUBLE | REPO value |
| 35 | REPOTERM | DOUBLE | REPO term |
| 36 | REPO2VALUE | DOUBLE | REPO buyback value |
| 37 | RETURN_VALUE | DOUBLE | REPO return value |
| 38 | DISCOUNT | DOUBLE | Discount (%) |
| 39 | LOWER_DISCOUNT | DOUBLE | Lower discount (%) |
| 40 | UPPER_DISCOUNT | DOUBLE | Upper discount (%) |
| 41 | BLOCK_SECURITIES | STRING | Block instruments. Valid values: Yes or No |
| 42 | URGENCY_FLAG | STRING | Fill. Valid values: Yes or No |
| 43 | TRADE_TYPE | STRING | Type. Valid values: <ul style="list-style-type: none"> _ Negotiated deal; _ First part of REPO deal; _ Second part of REPO deal; _ Compensation payment |
| 44 | TRADE_OPERATION_TYPE | STRING | Direction: Deposit or Withdraw |
| 45 | EXPECTED_DISCOUNT | DOUBLE | Discount after depositing (%) |
| 46 | EXPECTED_QUANTITY | DOUBLE | Quantity after depositing |
| 47 | EXPECTED_REPOVALUE | DOUBLE | REPO value after depositing |
| 48 | EXPECTED_REPO2VALUE | DOUBLE | Buyback value after depositing |



| No. | Parameter | Type | Description |
|-----|------------------------|--------|--|
| 49 | EXPECTED_RETURN_VALUE | DOUBLE | Return value after depositing |
| 50 | REPORT_TRADE_DATE | DOUBLE | Trade date |
| 51 | STATE_OF_CLEARING | STRING | Clearing status. Valid values: <ul style="list-style-type: none"> _ Processed; _ Not processed; _ In processing |
| 52 | TYPE_OF_CLEARING | STRING | Clearing type. Valid values: <ul style="list-style-type: none"> _ Not set; _ Simple; _ Multilateral |
| 53 | REPORT_COMISSION | DOUBLE | Report fee |
| 54 | COUPON_PAYMENT | DOUBLE | Coupon payment |
| 55 | COUPON_PAYMENT_DATE | DOUBLE | Coupon payment date |
| 56 | PRINCIPAL_PAYMENT | DOUBLE | Payment of principal |
| 57 | PRINCIPAL_PAYMENT_DATE | DOUBLE | Principal payment date |
| 58 | SETTLE_CURRENCY | STRING | Settlement currency |

12. Description of parameters from the **Table of Negotiated Trade Orders / Reports** returned by GET_ITEM:

| No. | Parameter | Type | Description |
|-----|------------|--------|-----------------------------|
| 1 | NUMBER | DOUBLE | Number |
| 2 | DATE | STRING | Date |
| 3 | TIME | STRING | Order entry time |
| 4 | CLASS | STRING | Class |
| 5 | SECCODE | STRING | Instrument code |
| 6 | SECURITY | STRING | Short name of an instrument |
| 7 | USERID | STRING | Trader ID |
| 8 | FIRMID | STRING | Dealer ID |
| 9 | FIRMNAME | STRING | Trader's firm ID |
| 10 | ACCOUNT | STRING | Depo account |
| 11 | CPFIRMNAME | STRING | Partner's name |



| No. | Parameter | Type | Description |
|-----|-------------|--------|--|
| 12 | CPFIRMID | STRING | Partner's ID |
| 13 | CPACCOUNT | STRING | Partner's depo account |
| 14 | QUANTITY | DOUBLE | Quantity |
| 15 | VALUE | DOUBLE | Value |
| 16 | COMISSION | DOUBLE | Commission |
| 17 | DIRECTION | STRING | Direction: Sent or Received |
| 18 | STATUS | STRING | Status: Awaiting execution, killed or filled |
| 19 | REPORT_TYPE | STRING | Report type: EXECUTION or CANCEL EXECUTION |
| 20 | REPORT_KIND | STRING | Type of report |

13.Description of parameters from the **Participant's cash positions table** returned by GET_ITEM:

| No. | Parameter | Type | Description |
|-----|---------------|--------|------------------------|
| 1 | FIRMID | STRING | Firm |
| 2 | CURRCODE | STRING | Currency |
| 3 | TAG | STRING | Position code |
| 4 | DESCRIPTION | STRING | Description |
| 5 | OPENBAL | DOUBLE | Opening balance |
| 6 | CURRENTPOS | DOUBLE | Current position |
| 7 | PLANNEDPOS | DOUBLE | Planned position |
| 8 | LIMIT1 | DOUBLE | External limit |
| 9 | ORDERBUY | DOUBLE | Buy value (in orders) |
| 10 | ORDERSELL | DOUBLE | Sell value (in orders) |
| 11 | NETOBLIGATION | DOUBLE | Net liability |
| 12 | PLANNEDBAL | DOUBLE | Check position |
| 13 | BANK_ACC_ID | STRING | Account ID |

14.Description of parameters from the **Participant's positions in instruments table** returned by GET_ITEM:



| No. | Parameter | Type | Description |
|-----|----------------|--------|--------------------|
| 1 | FIRMID | STRING | Firm |
| 2 | SEC_SHORT_NAME | STRING | Instrument name |
| 3 | SECCODE | STRING | Instrument code |
| 4 | OPENBAL | DOUBLE | Opening |
| 5 | CURRENTPOS | DOUBLE | Current |
| 6 | PLANNEDPOSBUY | DOUBLE | Planned buy value |
| 7 | PLANNEDPOSSELL | DOUBLE | Planned sell value |
| 8 | USQTYB | DOUBLE | Quantity bought |
| 9 | USQTYS | DOUBLE | Quantity sold |

15. Description of parameters from the **Participant's positions on trading accounts table** returned by GET_ITEM:

| No. | Parameter | Type | Description |
|-----|----------------|--------|--------------------|
| 1 | SECCODE | STRING | Instrument code |
| 2 | SEC_SHORT_NAME | STRING | Instrument name |
| 3 | FIRMID | STRING | Firm ID |
| 4 | TRDACCID | STRING | Trading account |
| 5 | DEPACCID | STRING | Depo account |
| 6 | OPENBAL | DOUBLE | Opening balance |
| 7 | CURRENTPOS | DOUBLE | Current position |
| 8 | PLANNEDPOSBUY | DOUBLE | Planned buy value |
| 9 | PLANNEDPOSSELL | DOUBLE | Planned sell value |
| 10 | PLANBAL | DOUBLE | Check position |
| 11 | USQTYB | DOUBLE | Quantity bought |
| 12 | USQTYS | DOUBLE | Quantity sold |
| 13 | PLANNED | DOUBLE | Planned position |



8.10 Functions for accessing a list of available parameters

8.10.1 GET_CLASSES_LIST

This function is used to obtain a list of class codes received from the server during the current session. The list of class codes is comma "," separated.

GET_CLASSES_LIST ()

Example:

```
`  
ClassesList = GET_CLASSES_LIST ()  
`
```

The list of available classes, e.g., TQBR, TQBS, TQNL, TQOB, TQOS, TQNO, is assigned to the variable ClassesList.

8.10.2 GET_CLASS_SECURITIES

This function is used to obtain a list of instrument codes for the list of classes set by the list of class codes. The list of instruments' codes is comma "," delimited.

GET_CLASS_SECURITIES (STRING)

Example:

```
`  
SecuritiesList = GET_CLASS_SECURITIES ("TQBR,GKO")  
`
```

The list of codes for all available instruments for classes A1 Shares and GKO is assigned to the variable SecuritiesList.

8.10.3 GET_SECURITY_INFO

This function retrieves information about an instrument with a specific code ("sec_code") from a particular class ("class_code"). If "class_code" is set to <space>, the function performs a search in all classes until the first item is found.

MAP GET_SECURITY_INFO (STRING class_code, STRING sec_code)

Parameters:

| No. | Parameter | Type | Description |
|-----|-----------|------|-------------|
|-----|-----------|------|-------------|



| No. | Parameter | Type | Description |
|-----|------------|--------|--|
| 1 | CODE | STRING | Instrument code |
| 2 | NAME | STRING | Instrument name |
| 3 | SHORT_NAME | STRING | Short name |
| 4 | CLASS_CODE | STRING | Class code |
| 5 | CLASS_NAME | STRING | Class name |
| 6 | FACE_VALUE | DOUBLE | Face value |
| 7 | FACE_UNIT | STRING | Face value currency code |
| 8 | SCALE | DOUBLE | Number of digits after the decimal point |
| 9 | MAT_DATE | STRING | Expiry date |
| 10 | LOT_SIZE | DOUBLE | Lot size |

Example:

```
\
SecInfo = GET_SECURITY_INFO("", "YUKO")
Lot = GET_VALUE (SecInfo, "LOT_SIZE")
\
```

The variable "SecInfo" is assigned the parameter values for the YUKO instrument. The variable "Lot" contains the number of instruments in one lot of YUKO.

8.11 Functions for handling programmable tables

This set of functions can only be used to handle the table "OWN" created when the program is calculated. This table is available not only when using the standard functions GET_ITEM and GET_NUMBER_OF, but also allows for specific modifications using the functions described below.

8.11.1 ADD_ITEM

This function inserts a line labeled "Index" into the "OWN" table. An associative array "table_string" is used for the initialization of table columns with values. "Table_string" should contain elements with keys equal to the column names.

```
ADD_ITEM (DOUBLE Index, MAP table_string)
```

8.11.2 MODIFY_ITEM

This function modifies an existing line labeled "Index" using the array "table_string".



MODIFY_ITEM (DOUBLE Index, MAP table_string)

8.11.3 DELETE_ITEM

This function is used to delete the line labeled "Index".

DELETE_ITEM (DOUBLE Index)

8.11.4 DELETE_ALL_ITEMS

This function is used to completely clear the "OWN" table.

DELETE_ALL_ITEMS()

Parameters:

| № | Parameter | Type | Description |
|---|--------------|--------|--|
| 1 | INDEX | DOUBLE | Index referring to a collection, which must contain an integer value |
| 2 | TABLE_STRING | MAP | Array containing the values for the columns of the modified row |

Example:

```
\n
st=CREATE_MAP()
st=SET_VALUE(st, "Value",10)
ADD_ITEM(1,st)
st=GET_ITEM ("OWN",1)
value=GET_VALUE(st,"Value")
DELETE_ALL_ITEMS()
\
```

This example is only available for a table containing a single column with the heading "Value". First, an array containing an element with the value "10" and the key "Value" is created. Then, a line with an index of "1" and the value for the "Value" column set to "10" is created in the table. Subsequently, the line with the index "1" is read back into the array, and the value of the element with the "Value" key is included in the variable "value". Finally, all table lines are deleted with the command "DELETE_ALL_ITEMS()".

8.11.5 SET_ROW_COLOR

This function assigns a color to the specified row of the table. This feature applies the language from the previous version and is retained for the purpose of compatibility.

SET_ROW_COLOR (STRING client_code, STRING background_color,
STRING selected_background_color)



Parameters:

| Nº | Parameter | Type | Description |
|----|---------------------------|--------|--|
| 1 | CLIENT_CODE | STRING | Client code. Once this parameter is set in "ROWNAME", the currently calculated client is highlighted |
| 2 | BACKGROUND_COLOR | STRING | Background color of the highlighted row |
| 3 | SELECTED_BACKGROUND_COLOR | STRING | Color of highlighting |

The background / highlighting color is set by the macro RGB(<red>, <green>, <blue>). If, for example, the string "RGB(255, 0, 0)" is transferred to the function as the color parameter, the color of highlighting is red. The background color is set using the string "DEFAULT_COLOR".

Example:

```
\nSET_ROW_COLOR (ROWNAME, "RGB(0,255,0)", "DEFAULT_COLOR")\n
```

Here, the background color for the currently calculated client is set to green, while the color of highlighting is set by default.

8.11.6 SET_ROW_COLOR_EX

This function assigns the background and font color to the specified row of the table.

SET_ROW_COLOR_EX (DOUBLE row_number, STRING background_color, STRING selected_background_color, STRING font_color, STRING selected_font_color)

Example:

| Nº | Parameter | Type | Description |
|----|---------------------------|--------|---|
| 1 | ROW_NUMBER | DOUBLE | Number of the row to be highlighted |
| 2 | BACKGROUND_COLOR | STRING | Background color of the row |
| 3 | SELECTED_BACKGROUND_COLOR | STRING | Background color of the row selected using the cursor |
| 4 | FONT_COLOR | STRING | Basic color of the font in a row |
| 5 | SELECTED_FONT_COLOR | STRING | Font color of the row selected using the cursor |

The background / highlighting color is set by the macro "RGB(<red>, <green>, <blue>)". If, for example, the string "RGB(255, 0, 0)" is transferred to the function as the color parameter, the color of highlighting is red. The background color is set using the string "DEFAULT_COLOR".



Example:

```
\nSET_ROW_COLOR_EX (10, "DEFAULT_COLOR", "DEFAULT_COLOR", "RGB(0,255,0)",\n"RGB(0,0,255)")\n
```

The background color of the font for the 10th row of the table is set to green. When the row is highlighted by the cursor, the font color becomes blue.

8.12 Functions for getting values from the quotes table

8.12.1 GET_PARAM

This function is used to obtain exchange information parameters. Using this function, it is possible to retrieve data from the Quotes Table for preset class and instrument codes.

GET_PARAM (STRING classcode_list, STRING seccode, STRING param_name)

Parameters:

| Nº | Parameter | Type | Description |
|----|----------------|--------|---|
| 1 | CLASSCODE_LIST | STRING | The list of instrument class codes separated by commas in which the instrument is searched. If, for example, the string "TQBR,TQBS,TQNL" is transferred to the function, the instrument is searched for in the A1 Shares, A2 Shares and B Shares classes. |
| 2 | SECCODE | STRING | Instrument code |
| 3 | PARAM_NAME | STRING | Parameter ID |

Example:

```
\nLastPrice = GET_PARAM ("TQBR", "HYDR", "last")\n
```

The variable "LastPrice" is assigned the value of the last trade's price in RusHydro shares for class A1 Shares of MOEX.



8.12.2 GET_PARAM_EX

This function is used to obtain the values of all exchange information parameters from the Quotes Table. Using this function, it is possible to retrieve any value from the Quotes Table for preset class and instrument codes.

```
MAP GET_PARAM_EX (STRING classcode, STRING seccode, STRING param_name)
```

If the command `USE_CASE_SENSITIVE_CONSTANTS` (see [8.3.2](#)) is used in the program code, then the values of parameters "classcode", and "seccode" should be specified in register corresponding to Quotes table register, and the value of parameter "param.name" should be specified in upper case.



Parameters:

| No. | Parameter | Type | Description |
|-----|------------|--------|-------------------------------------|
| 1 | CLASSCODE | STRING | Class code (for example, EQBR) |
| 2 | SECCODE | STRING | Instrument code (for example, LKOH) |
| 3 | PARAM_NAME | STRING | Parameter ID (for example, PRICE) |

The MAP has the following structure:

| No. | Parameter | Type | Description |
|-----|-------------|--------|--|
| 1 | RESULT | DOUBLE | The result of completing an operation, where 0 represents an error and 1 represents a found parameter |
| 2 | PARAM_TYPE | DOUBLE | The type of parameter data used in the Quotes Table. The list of valid values includes: <ul style="list-style-type: none">– 1 – DOUBLE;– 2 – LONG;– 3 – CHAR;– 4 – enumerated type;– 5 – time;– 6 – date |
| 3 | PARAM_VALUE | DOUBLE | Parameter value. For param_type = 3, the value equals '0'. For enumerated types, the value is equal to the ordinal value of the enumeration. |
| 4 | PARAM_IMAGE | STRING | The string value of a parameter similar to its representation in the table. The string representation includes separators of the digit positions and those between the integer and the fractional part. For enumerated types, the relevant string values are derived |

Example:

```
\nParam=GET_PARAM_EX("TQBR", "RTKM", "WAPRICE")\nWAPRICE=GET_VALUE(PARMAP, "PARAM_VALUE")
```

The variable "WAPRICE" is assigned the value of the volume-weighted average price for ordinary Rostelecom shares of class A1 Shares of MOEX.



8.12.3 Function parameter values

The list of possible instruments class codes ("classcode_list") are as follows:

| Class code | Name |
|-------------------|--|
| TQBR | MOEX SM: T+ A1 Shares, pays and RDR |
| TQBS | MOEX SM: T+ A2 Shares and pays |
| TQNL | MOEX SM: T+ B Shares and pays |
| TQOB | MOEX SM: T+ A1 Bonds |
| TQOS | MOEX SM: T+ A2 Bonds |
| TQNO | MOEX SM: T+ B Bonds |
| PSEQ | Negotiated deal mode: A1 Shares and pays |
| PSES | Negotiated deal mode: A2 Shares and pays |
| PSNL | Negotiated deal mode: B Shares and pays |
| PSOB | Negotiated deal mode: A1 Shares |
| PSNO | Negotiated deal mode: B Bonds |
| PSAU | MOEX SM: NDM: Initial placement (bonds) |
| AUCT | MOEX SM: Auction (Shares) |
| MAIN | MOEX Government securities |
| MAIC | Government securities: closing period |
| BOBR | OBR (Bank of Russia Bonds) |
| FUOP | MOEX Futures |

| Class code | Name |
|-------------------|---|
| GAZP | Gazprom Bonds |
| INDX | MOEX SM: Indices |
| GTS | RTS: SGK |
| GAZ | RTS: Gazprom shares |
| QUADRO | RTS-SGK (Trading in currency) |
| RTS10 | RTS10 |
| RTSIDX | RTS indices |
| RTSIND | RTS (Indices) |
| SES2 | Government securities: Large-scale lots |
| SPBFUT | Futures on FORTS |
| SPBOPT | Options on FORTS |
| SPBSPT | St. Petersburg SPOT |
| SPBGKO | MKO |
| SPBCEX | GGKO on the SPCEX |
| SPCGKO | Test system on the SPCEX |
| EQBREMU | Top-tier shares (Emulator) |
| USDRUB | FORTS USD/RUB exchange rate |

List of possible parameter identifiers:

| No. | Parameter | Type | Description |
|------------|------------------|-------------|--------------------|
| 1 | STATUS | STRING | Status |



| No. | Parameter | Type | Description |
|-----|---------------|---------|---|
| 2 | LOTSIZE | NUMERIC | Lot size |
| 3 | BID | NUMERIC | Highest bid price |
| 4 | BIDDEPTH | NUMERIC | Bid volume at the best price / bid depth |
| 5 | BIDDEPTHHT | NUMERIC | Total bid |
| 6 | NUMBIDS | NUMERIC | Number of bids |
| 7 | OFFER | NUMERIC | Lowest offer price |
| 8 | OFFERDEPTH | NUMERIC | Offer volume at the best price / offer depth |
| 9 | OFFERDEPTHHT | NUMERIC | Total offer |
| 10 | NUMOFFERS | NUMERIC | Number of offers |
| 11 | OPEN | NUMERIC | Opening price |
| 12 | HIGH | NUMERIC | Highest trade price |
| 13 | LOW | NUMERIC | Lowest trade price |
| 14 | LAST | NUMERIC | Last trad price |
| 15 | CHANGE | NUMERIC | Price difference between the last and the preceding session |
| 16 | QTY | NUMERIC | Quantity of instruments in the last trade |
| 17 | TIME | STRING | Time of the last trade |
| 18 | VOLTODAY | NUMERIC | Volume of instruments in anonymous trades |
| 19 | VALTODAY | NUMERIC | Value in cash |
| 20 | TRADINGSTATUS | STRING | Trading session status |
| 21 | VALUE | NUMERIC | Cash value of the last trade |
| 22 | WAPRICE | NUMERIC | Volume-weighted average price |
| 23 | HIGHBID | NUMERIC | Highest bid price for today |
| 24 | LOWOFFER | NUMERIC | Lowest offer price for today |
| 25 | NUMTRADES | NUMERIC | Number of trades for today |
| 26 | PREVPRICE | NUMERIC | Closing price |
| 27 | PREVWAPRICE | NUMERIC | Previous price |
| 28 | CLOSEPRICE | NUMERIC | Closing period price |
| 29 | LASTCHANGE | NUMERIC | Percentage change from the closing time |



| No. | Parameter | Type | Description |
|-----|-----------------------|---------|--|
| 30 | DIVIDENDDATE | NUMERIC | Dividend date |
| 31 | PRIMARYDIST | STRING | Placement / primary distribution |
| 32 | ACCRUEDINT | NUMERIC | Accrued coupon income |
| 33 | YIELD | NUMERIC | Yield from the last trade |
| 34 | COUPONVALUE | NUMERIC | Coupon value |
| 35 | YIELDATPREVWAPRI | NUMERIC | Yield according to the previous estimate |
| 36 | YIELDATWAPRICE | NUMERIC | Estimated yield |
| 37 | PRICEMINUSPREVWAPRICE | NUMERIC | Difference between the last estimate and the previous one |
| 38 | CLOSEYIELD | NUMERIC | Yield at closure |
| 39 | CURRENTVALUE | NUMERIC | Current value of MOEX indices |
| 40 | LASTVALUE | NUMERIC | Value of MOEX indices on the last day's closing |
| 41 | LASTTOPREVSTLPRC | NUMERIC | Price difference between the last and the preceding session |
| 42 | PREVSETTLPRICE | NUMERIC | Previous settlement price |
| 43 | PRICEMVTLIMIT | NUMERIC | Price movement limit |
| 44 | PRICEMVTLIMITT1 | NUMERIC | Price movement limit T1 |
| 45 | MAXOUTVOLUME | NUMERIC | Maximum volume / limit of active orders (in contracts) |
| 46 | PRICEMAX | NUMERIC | Maximum possible price |
| 47 | PRICEMIN | NUMERIC | Minimum possible price |
| 48 | NEGVALTODAY | NUMERIC | Value of negotiated trades in cash |
| 49 | NEGNUMTRADES | NUMERIC | Number of negotiated trades for today |
| 50 | NUMCONTRACTS | NUMERIC | Number of open positions |
| 51 | CLOSETIME | STRING | Closing time of previous trading session (for RTS indices) |
| 52 | OPENVAL | NUMERIC | RTS index value at the time of opening |
| 53 | CHNGOPEN | NUMERIC | Difference between the current RTS index and that at the time of opening |
| 54 | CHNGCLOSE | NUMERIC | Difference between the current RTS index and that at the time of closing |
| 55 | BUYDEPO | NUMERIC | Seller's collateral |
| 56 | SELLDEPO | NUMERIC | Buyer's collateral |



| No. | Parameter | Type | Description |
|-----|------------------|---------|---|
| 57 | CHANGETIME | STRING | Time of the last change |
| 58 | SELLPROFIT | NUMERIC | Profit from sales |
| 59 | BUYPROFIT | NUMERIC | Profit from buys |
| 60 | TRADECHANGE | NUMERIC | Price difference between the last trade and the previous one (Moscow Exchange derivatives market, St. Petersburg Stock Exchange, SPCEX) |
| 61 | FACEVALUE | NUMERIC | Face value (for SPCEX instruments) |
| 62 | MARKETPRICE | NUMERIC | Yesterday's market price |
| 63 | MARKETPRICETODAY | NUMERIC | Today's market price |
| 64 | NEXTCOUPON | NUMERIC | Date of coupon yield payment |
| 65 | BUYBACKPRICE | NUMERIC | Buyback price |
| 66 | BUYBACKDATE | NUMERIC | Buyback date |
| 67 | ISSUESIZE | NUMERIC | Issue size |
| 68 | PREVDATE | NUMERIC | Date of previous trading day |
| 69 | DURATION | NUMERIC | Duration |
| 70 | LOPENPRICE | NUMERIC | Official opening price |
| 71 | LCURRENTPRICE | NUMERIC | Official current price |
| 72 | LCLOSEPRICE | NUMERIC | Official closing price |
| 73 | QUOTEBASIS | STRING | Quote basis / price type |
| 74 | PREVADMITTEDQUOT | NUMERIC | Admitted quote for the previous day |
| 75 | LASTBID | NUMERIC | Best bid price at closing |
| 76 | LASTOFFER | NUMERIC | Best offer price at closing |
| 77 | PREVLEGALCLOSEPR | NUMERIC | Closing price from the previous day |
| 78 | COUPONPERIOD | NUMERIC | Coupon period |
| 79 | MARKETPRICE2 | NUMERIC | Market price 2 |
| 80 | ADMITTEDQUOTE | NUMERIC | Admitted quote |
| 81 | BGOP | NUMERIC | BGO for covered positions |
| 82 | BGONP | NUMERIC | BGO for uncovered positions |
| 83 | STRIKE | NUMERIC | Strike price |
| 84 | STEPPRICET | NUMERIC | Price step value |



| No. | Parameter | Type | Description |
|-----|-----------------|---------|---|
| 85 | STEPPRICE | NUMERIC | Price step value (for new Moscow Exchange derivatives market contracts) |
| 86 | SETTLEPRICE | NUMERIC | Settlement price |
| 87 | OPTIONTYPE | STRING | Option type |
| 88 | OPTIONBASE | STRING | Underlying asset |
| 89 | VOLATILITY | NUMERIC | Option volatility |
| 90 | THEORPRICE | NUMERIC | Theoretical price |
| 91 | PERCENTRATE | NUMERIC | Aggregated rate |
| 92 | ISPERCENT | STRING | Futures price type |
| 93 | CLSTATE | STRING | Clearing status |
| 94 | CLPRICE | NUMERIC | Last clearing quote |
| 95 | STARTTIME | STRING | Main session starting time |
| 96 | ENDTIME | STRING | Main session closing time |
| 97 | EVNSTARTTIME | STRING | Evening session starting time |
| 98 | EVNENDTIME | STRING | Evening session ending time |
| 99 | MONSTARTTIME | STRING | Morning session starting time |
| 100 | MONENDTIME | STRING | Morning session ending time |
| 101 | CURSTEPPRICE | STRING | Price step currency |
| 102 | REALVMPRICE | NUMERIC | Current market quote |
| 103 | MARG | STRING | Provided with margin |
| 104 | EXPDATE | NUMERIC | Instrument expiration date |
| 105 | CROSSRATE | NUMERIC | Cross rate |
| 106 | BASEPRICE | NUMERIC | Underlying rate |
| 107 | HIGHVAL | NUMERIC | Maximum value (RTSIND) |
| 108 | LOWVAL | NUMERIC | Minimum value (RTSIND) |
| 109 | ICHANGE | NUMERIC | Change (RTSIND) |
| 110 | IOPEN | NUMERIC | Opening value (RTSIND) |
| 111 | PCHANGE | NUMERIC | Percentage change (RTSIND) |
| 112 | OPENPERIODPRICE | NUMERIC | Price during the pre-trading period |



| No. | Parameter | Type | Description |
|-----|------------------------|---------|---|
| 113 | MIN_CURR_LAST | NUMERIC | Minimum current price |
| 114 | SETTLECODE | STRING | Default settlement code |
| 115 | STEPPRICECL | DOUBLE | Clearing price step value |
| 116 | STEPPRICEPRCL | DOUBLE | Price step value for intermediate clearing |
| 117 | MIN_CURR_LAST_TI | STRING | Time of changes in the minimum current price |
| 118 | PREVLOTSIZE | NUMERIC | Previous value for the lot size |
| 119 | LOTSIZECHANGEDAT | NUMERIC | Date of the last changes to the lot size |
| 120 | AUCTPRICE | NUMERIC | Post-trading auction price |
| 121 | CLOSING_AUCTION_VOLUME | NUMERIC | Volume of trades for the post-trading auction |

List of identifiers of additional parameters available for the function GET_PARAM_EX:

| No. | Parameter | Type | Description |
|-----|------------------|--------|-----------------------------------|
| 1 | LONGNAME | STRING | Full name of an instrument |
| 2 | SHORTNAME | STRING | Short name of an instrument |
| 3 | CODE | STRING | Instrument code |
| 4 | CLASSNAME | STRING | Class name |
| 5 | CLASS_CODE | STRING | Class code |
| 6 | TRADE_DATE_CODE | DOUBLE | Trading date |
| 7 | MAT_DATE | DOUBLE | Expiry date |
| 8 | DAYS_TO_MAT_DATE | DOUBLE | Number of days to the expiry date |
| 9 | SEC_FACE_VALUE | DOUBLE | Face value of an instrument |
| 10 | SEC_FACE_UNIT | STRING | Face value currency unit |
| 11 | SEC_SCALE | DOUBLE | Price accuracy |
| 12 | SEC_PRICE_STEP | DOUBLE | Minimum price step |
| 13 | SECTYPE | STRING | Instrument type |



8.13 Functions for retrieving values from the Level II Quotes table

8.13.1 GET_QUOTES_II_LEVEL_DATA

This function is used to retrieve the values for instrument quotes. Using this function, it is possible to obtain data from the Level II Quotes Table for preset class and instrument codes.

```
MAP GET_QUOTES_II_LEVEL_DATA (STRING ClassCode, STRING SecCode)
```

This function retrieves a "MAP" that has the following structure:

| No. | Parameter | Type | Description |
|-----|-------------|------------|------------------------|
| 1 | BID_COUNT | DOUBLE | Number of bid quotes |
| 2 | OFFER_COUNT | DOUBLE | Number of offer quotes |
| 3 | BID | COLLECTION | Bid / buying quotes |
| 4 | OFFER | COLLECTION | Offer / selling quotes |

"BID" and "OFFER" collections have the following structure:

| No. | Parameter | Type | Description |
|-----|-----------|--------|-------------------|
| 1 | PRICE | DOUBLE | Offer / bid price |
| 2 | QUANTITY | DOUBLE | Quantity in lots |

8.14 Functions for retrieving values from the Positions in instruments table

These functions are used to obtain the table values for a preset client code, firm code, instrument code, and depo account.

8.14.1 DEPO_OPEN_BALANCE

This function returns the value for the "Opening balance for instruments".

```
DEPO_OPEN_BALANCE (STRING client_code, STRING firmid, STRING seccode,  
STRING account)
```

8.14.2 DEPO_OPEN_LIMIT

This function returns the value for the "Opening limit for instruments".



DEPO_OPEN_LIMIT (STRING client_code, STRING firmid, STRING seccode, STRING account)

8.14.3 DEPO_CURRENT_BALANCE

This function returns the value for the "Current balance for instruments".

DEPO_CURRENT_BALANCE (STRING client_code, STRING firmid, STRING seccode, STRING account)

8.14.4 DEPO_CURRENT_LIMIT

This function returns the value for the "Current limit for instruments".

DEPO_CURRENT_LIMIT (STRING client_code, STRING firmid, STRING seccode, STRING account)

8.14.5 DEPO_LIMIT_AVAILABLE

This function returns the value for the available limit for instruments.

DEPO_LIMIT_AVAILABLE (STRING client_code, STRING firmid, STRING seccode, STRING account)

8.14.6 DEPO_LIMIT_LOCKED

This function returns the value for the "Number of locked instruments".

DEPO_LIMIT_LOCKED (STRING client_code, STRING firmid, STRING seccode, STRING account)

8.14.7 DEPO_LIMIT_LOCKED_BUY

This function returns the value for the "Number of lots of instruments locked for buying".

DEPO_LIMIT_LOCKED_BUY (STRING client_code, STRING firmid, STRING seccode, STRING account)

8.14.8 DEPO_LIMIT_LOCKED_BUY_VALUE

This function returns the value for the "Value of instruments locked for buying".

DEPO_LIMIT_LOCKED_BUY_VALUE (STRING client_code, STRING firmid, STRING seccode, STRING account)

Parameters:

| No. | Parameter | Type | Description |
|-----|-------------|--------|-----------------|
| 1 | client_code | STRING | Client code |
| 2 | firmid | STRING | Firm ID |
| 3 | seccode | STRING | Instrument code |
| 4 | account | STRING | Depo account |



| The parameter “account” is case sensitive (upper / lower case characters).



Example:

```
\nClDepoOB = DEPO_OPEN_BALANCE ("1075", "NC0080000000", "HYDR", "L01-00000F00")\nClDepoOL = DEPO_OPEN_LIMIT ("1075", "NC0080000000", "HYDR", "L01-00000F00")\nClDepoCB = DEPO_CURRENT_BALANCE ("1075", "NC0080000000", "HYDR", "L01-00000F00")\nClDepoCL = DEPO_CURRENT_LIMIT ("1075", "NC0080000000", "HYDR", "L01-00000F00")\nClDepoAV = DEPO_LIMIT_AVAILABLE ("1075", "NC0080000000", "HYDR", "L01-00000F00")\nClDepoLCK = DEPO_LIMIT_LOCKED ("1075", "NC0080000000", "HYDR", "L01-00000F00")\nClDepoLCKBuy = DEPO_LIMIT_LOCKED_BUY ("1075", "NC0080000000", "HYDR", "L01-00000F00")\nClDepoLCKBuyValue = DEPO_LIMIT_LOCKED_BUY_VALUE ("1075", "NC0080000000", "HYDR", "L01-00000F00")\n\
```

The example shows the assignment of values for the variables from the Positions in Instruments Table (ordinary shares of "RusHydro") for the client code "1075":

- The variable "ClDepoOB" is set to the value of the opening balance;
- The variable "ClDepoOL" is set to the value of the opening limit;
- The variable "ClDepoCB" is set to the value of the current balance;
- The variable "ClDepoCL" is set to the value of the current limit;
- The variable "ClDepoAV" is set to the value of the number of available instruments;
- The variable "ClDepoLCK" is set to the value of the number of locked instruments;
- The variable "ClDepoLCKBuy" is set to the value of the number of lots locked for buying;
- The variable "ClDepoLCKBuyValue" is set to the value of instruments locked for buying.

8.15 Functions for retrieving values from the Cash positions table

These functions are used to retrieve table values for a preset client code, firm code, position code, and currency code.

8.15.1 MONEY_OPEN_BALANCE

This function returns the value for "Opening cash balance".

```
MONEY_OPEN_BALANCE (STRING client_code, STRING firmid, STRING tag,  
STRING curr_code)
```

8.15.2 MONEY_OPEN_LIMIT

This function returns the value for the "Opening cash limit".

```
MONEY_OPEN_LIMIT (STRING client_code, STRING firmid, STRING tag,  
STRING curr_code)
```



8.15.3 MONEY_CURRENT_BALANCE

This function returns the value for the "Current cash balance".

```
MONEY_CURRENT_BALANCE (STRING client_code, STRING firmid, STRING tag, STRING curr_code)
```

8.15.4 MONEY_CURRENT_LIMIT

This function returns the value for the "Current cash limit".

```
MONEY_CURRENT_LIMIT (STRING client_code, STRING firmid, STRING tag, STRING curr_code)
```

8.15.5 MONEY_LIMIT_AVAILABLE

This function returns the value for the "Available cash limit".

```
MONEY_LIMIT_AVAILABLE (STRING client_code, STRING firmid, STRING tag, STRING curr_code)
```

8.15.6 MONEY_LIMIT_LOCKED

This function returns the value for the "Value of locked cash".

```
MONEY_LIMIT_LOCKED (STRING client_code, STRING firmid, STRING tag, STRING curr_code)
```

Example:

```
\nClMoneyOB = MONEY_OPEN_BALANCE ("1075", "NC0080000000", "EQTV", "SUR")\nClMoneyOL = MONEY_OPEN_LIMIT ("1075", "NC0080000000", "EQTV", "SUR")\nClMoneyCB = MONEY_CURRENT_BALANCE ("1075", "NC0080000000", "EQTV", "SUR")\nClMoneyCL = MONEY_CURRENT_LIMIT ("1075", "NC0080000000", "EQTV", "SUR")\nClMoneyAV = MONEY_LIMIT_AVAILABLE ("1075", "NC0080000000", "EQTV", "SUR")\nClMoneyLCK = MONEY_LIMIT_LOCKED ("1075", "NC0080000000", "EQTV", "SUR")\n\
```

The example shows the assignment of values for variables from the Cash positions table for the MOEX Stock Market for the client "1075":

- The variable "ClMoneyOB" is set to the value of the opening cash balance;
- The variable "ClMoneyOL" is set to the value of the opening cash limit;
- The variable "ClMoneyCB" is set to the value of the current cash balance;
- The variable "ClMoneyCL" is set to the value of the current cash limit;
- The variable "ClMoneyAV" is set to the value of the amount of available cash;
- The variable "ClMoneyLCK" is set to the value of amount of locked cash.



8.16 Functions for the calculation of margin positions

These functions are used to obtain the values of margin positions for a preset client code, firm code, class code, depo account, and parameter for the price at which the cost is calculated (for example, "OPEN" is used for the opening price, "LAST" is used for the last trade price, etc.).

8.16.1 SHORT_VALUE

This function returns the "Value of all short positions".

```
SHORT_VALUE (STRING client_code, STRING firmid, STRING seccode,  
             STRING class_code, STRING account, STRING price_param_code)
```

8.16.2 LONG_VALUE

This function returns the "Value of all long positions".

```
LONG_VALUE (STRING client_code, STRING firmid, STRING seccode, STRING class_code,  
            STRING account, STRING price_param_code)
```

Parameters:

| No. | Parameter | Type | Description |
|-----|------------------|--------|----------------------|
| 1 | CLIENT_CODE | STRING | Client code |
| 2 | FIRMID | STRING | Firm ID |
| 3 | SECCODE | STRING | Instrument code |
| 4 | CLASS_CODE | STRING | Class code |
| 5 | *ACCOUNT | STRING | Depo account |
| 6 | PRICE_PARAM_CODE | STRING | Price parameter code |

(*) This parameter is case sensitive (upper / lower case characters).

Example:

```
\nClShortsValue = SHORT_VALUE ("1075", "NC0080000000", "HYDR", "TQBR", "L01-00000F00",  
                             "LAST")\nClLongsValue = LONG_VALUE ("1075", "NC0080000000", "HYDRR", "TQBR", "L01-00000F00",  
                           "OPEN")\n\
```



The variable "CIShortsValue" is set to the value of all short positions for the client code "1075" with respect to the "RusHydro" instrument from class A1 Shares for the account L01-00000F00 adjusted to the last trade price.

The variable "CILongsValue" is set to the value of all long positions for the client code "1075" with respect to the "RusHydro" instrument from class A1 Shares for the account L01-00000F00 adjusted to the opening price.

8.17 Functions for retrieving values from the Client Portfolio and Buy / Sell tables

These functions are used to obtain values from the tables mentioned above. The table values are calculated on the QUIK client workstation at the intervals specified in the settings (menu item **System / Settings / General settings...**, tab Trading / Client Portfolio, checkbox "Refresh portfolio each ... seconds").

8.17.1 GET_CLIENT_MARGINAL_PORTFOLIO_INFO

This function retrieves an associative array (MAP) with the parameters of the Client Portfolio table corresponding to the trading participant's ID ("firmid") and the client code ("client_code").

MAP GET_CLIENT_MARGINAL_PORTFOLIO_INFO (STRING firmid, STRING client_code)

Parameters:

| No. | Parameter | Type | Description | |
|-----|-------------|----------------|---|-----------------|
| 1 | IS_LEVERAGE | STRING (12) | Attribute of monitoring positions type. Valid values include: <ul style="list-style-type: none"> MLim: scheme of monitoring a position "by leverage" is used, the leverage is calculated based on the Incoming limit value; MP: scheme of monitoring a position "by leverage" is used when the leverage is expressly stated; Mpos: positions monitoring scheme "open position limit" is used; <blank>: positions monitoring scheme "by limit" is used | Client type |
| 2 | IN_ASSETS | DOUBLE | Value of the client's equity before the trading session begins | Opening assets |
| 3 | LEVERAGE | DOUBLE | Leverage. If not set explicitly, this is calculated as the ratio of the opening limit to opening assets. | Leverage |
| 4 | OPEN_LIMIT | DOUBLE | Maximum value of borrowed assets before the beginning of the trading session | Opening limit |
| 5 | VAL_SHORT | DOUBLE | Value of short positions, which is always negative | Short positions |
| 6 | VAL_LONG | DOUBLE | Value of long positions | Long positions |



| No. | Parameter | Type | Description | |
|-----|-------------------|--------|--|---|
| 7 | VAL_LONG_MARGIN | DOUBLE | Value of long positions for margin instruments accepted as collateral | Long positions for margin instruments |
| 8 | VAL_LONG_ASSET | DOUBLE | Value of long positions for non-margin instruments accepted as collateral | Long positions for non-margin instruments |
| 9 | ASSETS | DOUBLE | Value of the client's equity with reference to current positions and prices | Portfolio value |
| 10 | CUR_LEVERAGE | DOUBLE | Current leverage | Current leverage |
| 11 | MARGIN | DOUBLE | Margin as a percentage | Margin |
| 12 | LIM_ALL | DOUBLE | Current maximum value of borrowed assets | Current limit |
| 13 | AV_LIM_ALL | DOUBLE | Value of borrowed assets available for the further opening of positions | Available current limit |
| 14 | LOCKED_BUY | DOUBLE | Value of assets in buy orders | Locked buying |
| 15 | LOCKED_BUY_MARGIN | DOUBLE | Value of assets in orders to buy instruments accepted as collateral | Locked buying of margin instruments |
| 16 | LOCKED_BUY_ASSET | DOUBLE | Value of assets in orders to buy non-margin instruments accepted as collateral | Locked buying of collateral |
| 17 | LOCKED_SELL | DOUBLE | Value of assets in orders to sell margin instruments | Locked selling |
| 18 | LOCKED_VALUE_COEF | DOUBLE | Value of assets in orders to buy non-margin instruments | Locked buying of non-margin instruments |
| 19 | IN_ALL_ASSETS | DOUBLE | Value of all client positions adjusted to the closing prices from the preceding trading session including positions for non-margin instruments | Opening assets |
| 20 | ALL_ASSETS | DOUBLE | Current value of all client positions | Current assets |
| 21 | PROFIT_LOSS | DOUBLE | Magnitude of change in the value of all client positions | Profit / loss |
| 22 | RATE_CHANGE | DOUBLE | Relative change in the value of all client positions | Rate of change |
| 23 | LIM_BUY | DOUBLE | Value of cash assets available for buying margin instruments | Limit to buy |
| 24 | LIM_SELL | DOUBLE | Value of margin instruments available for selling | Limit to sell |



| No. | Parameter | Type | Description | |
|-----|--------------------|--------|--|---|
| 25 | LIM_NON_MARGIN | DOUBLE | Value of cash assets available for buying non-margin instruments | Limit to buy non-margin instruments |
| 26 | LIM_BUY_ASSET | DOUBLE | Value of cash assets available for buying instruments admitted as collateral | Limit to buy instruments for collateral |
| 27 | VAL_SHORT_NET | DOUBLE | *Value of short positions. Not used in the calculation of the discount rate | Short (net) |
| 28 | VAL_LONG_NET | DOUBLE | *Value of long positions. Not used in the calculation of the discount rate | Long (net) |
| 29 | TOTAL_MONEY_BAL | DOUBLE | Total cash balance for all positions not including assets blocked under the fulfillment of liabilities expressed in the selected settlement currency calculation | Total money balance |
| 30 | TOTAL_LOCKED_MONEY | DOUBLE | Total amount of blocked assets in all of a client's cash positions recalculated at the server into the settlement currency via exchange cross rates | Total locked money |
| 31 | HAIRCUTS | DOUBLE | Total discounts on the value of long (only for collateral instruments) and short instrument positions, 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 | DOUBLE | Total amount of cash balances, values of long collateral instrument positions, and values of short positions without discount factors, without instrument value netting within the scope of the unified instrument position, and without the correlation between instruments | Assets without HC |
| 33 | STATUS_COEF | DOUBLE | The ratio of the total discounts to the current assets excluding discounts | Status coefficient |
| 34 | VARMARGIN | DOUBLE | Current variation margin for a client's positions for all instruments | Variation margin |
| 35 | GO_FOR_POSITIONS | DOUBLE | The amount of cash assets paid for all open positions on the futures market | Current clear positions |
| 36 | GO_FOR_ORDERS | DOUBLE | Value of assets for futures market orders | Current clear orders |
| 37 | RATE_FUTURES | DOUBLE | Ratio of the portfolio liquidation value to collateral in the futures market | Assets / Current clear positions |



| No. | Parameter | Type | Description | |
|-----|----------------|--------|--|------------------------|
| 38 | IS_QUAL_CLIENT | STRING | Attribute of a "qualified" client, which is permitted credit through borrowed assets with a leverage of 1:3. Valid values are: HighRisk – qualified or <empty> – no. | HighRisk |
| 39 | IS_FUTURES | STRING | Client account in Moscow Exchange derivatives market if there is a joint position; otherwise, the field is left empty | Futures trade account |
| 40 | CURR_TAG | STRING | Actual current calculation parameters for the specified row in the format "<Currency> - <Trading session ID>". Example: "SUR-EQTV" | Parameters calculation |

(*) For detailed information about discount factors, see Section 7 of the Dealer Library settings Administrator's manual.

Example:

```
\
GET_CLIENT_MARGINAL_PORTFOLIO_INFO ("NC0080000000", "1")
\
```

8.17.2 GET_CLIENT_MARGINAL_PORTFOLIO_INFO_EX

This function returns an associative array (MAP) containing the parameters of the Client Portfolio Table corresponding to a trader's ID ("firmid"), client code ("client_code"), and position on date ("limit_kind").

```
MAP GET_CLIENT_MARGINAL_PORTFOLIO_INFO_EX (STRING firmid,
STRING client_code, DOUBLE limit_kind)
```

Valid values of parameter 'limit_kind':

- __ 0 – position on date T0;
- __ 1 – position on date T1;
- __ 2 – position on date T2;
- __ 365 – position on date Tx.

For a description of the returned parameters, see [8.17.1](#).



The following parameters are returned additionally:

| No | Parameter | Type | Description |
|----|------------------|--------|---|
| 1 | INIT_MARGIN | DOUBLE | Value of the initial margin. The field is filled for MD clients |
| 2 | MIN_MARGIN | DOUBLE | Value of the minimum margin. The field is filled for MD clients |
| 3 | CORRECTED_MARGIN | DOUBLE | Value of the corrected margin. The field is filled for MD clients |
| 4 | CLIENT_TYPE | DOUBLE | Client type |
| 5 | PORTFOLIO_VALUE | DOUBLE | Portfolio value. For MD clients the value for rows with the maximum position on date is returned |
| 6 | RCV1 | DOUBLE | Risk coverage value 1. It is calculated as the difference between the Portfolio value and the Init. margin parameters. For MD and MD+ clients |
| 7 | RCV2 | DOUBLE | Risk coverage value 2. It is calculated as the difference between the Portfolio value and the Min. margin parameters. For MD and MD+ clients |

Example:

```
\nGET_CLIENT_MARGINAL_PORTFOLIO_INFO_EX ("NC0080000000", "1", "0")\n
```

8.17.3 GET_CLIENT_MARGINAL_BUY_SELL_INFO

This function returns an associative array (MAP) with the parameters of the Buy / Sell Table referring to the opportunity to buy or sell a specific instrument ("sec_code") from a particular class ("class_code") identified by client ("client_code"), trading firm ("firmid"), and a preset price ("price"). If the price is set to "0", the best bid / offer values are used.

```
MAP GET_CLIENT_MARGINAL_BUY_SELL_INFO (STRING firmid, STRING client_code,\nSTRING class_code, STRING sec_code, DOUBLE price)
```

Parameters:

| No. | Parameter | Type | Description |
|-----|-----------|------|-------------|
|-----|-----------|------|-------------|



| No. | Parameter | Type | Description |
|-----|--------------------|--------|---|
| 1 | IS_MARGIN_SEC | DOUBLE | Marginity identifier of the instrument. Valid values include 1 for margin or 0 for non-margin. For MD clients the field is not filled |
| 2 | IS_ASSET_SEC | DOUBLE | List of instruments belonging to an instrument which are accepted as collateral. Valid values include 1 for accepted as collateral and 0 for not accepted as collateral. For MD clients the field is not filled |
| 3 | BALANCE | DOUBLE | Current position of the instrument in lots |
| 4 | CAN_BUY | DOUBLE | * Estimated number of lots available for buying at a specific price |
| 5 | CAN_SELL | DOUBLE | * Estimated number of lots available for selling at a specific price |
| 6 | POSITION_VALUATION | DOUBLE | Position value in cash for an instrument at bid / offer prices |
| 7 | VALUE | DOUBLE | Estimated value of the position at the last trade price |
| 8 | OPEN_VALUE | DOUBLE | Estimated value of the client's position calculated from the closing price during the preceding trading session |
| 9 | LIM_LONG | DOUBLE | The maximum limit of the position for a specific instrument accepted as collateral for long positions |
| 10 | LONG_COEF | DOUBLE | The discount factor applied to long positions for a specific instrument |
| 11 | LIM_SHORT | DOUBLE | The maximum limit of short positions for a specific instrument |
| 12 | SHORT_COEF | DOUBLE | The discount factor applied to short positions for a specific instrument |
| 13 | VALUE_COEF | DOUBLE | Estimated value of a position at the last trade price with discount factors applied |
| 14 | OPEN_VALUE_COEF | DOUBLE | Estimated value of the client's position calculated using discount factors to the closing price from the preceding trading session |
| 15 | SHARE | DOUBLE | Percentage ratio of the position value for a specific instrument to the value of all of the client's assets calculated at current prices |
| 16 | SHORT_WA_PRICE | DOUBLE | Volume-weighted average price for short instrument positions |
| 17 | LONG_WA_PRICE | DOUBLE | Volume-weighted average price for long instrument positions |
| 18 | PROFIT_LOSS | DOUBLE | The difference between the volume-weighted average purchase price of instruments and their market price |
| 19 | SPREAD_HC | DOUBLE | The correlation coefficient between instruments |



| No. | Parameter | Type | Description |
|-----|------------------|--------|---|
| 20 | CAN_BUY_OWN | DOUBLE | Maximum possible number of shares in orders for purchase for an instrument in a specific class in a client's asset portfolio based on the best offer price |
| 21 | CAN_SELL_OWN | DOUBLE | Maximum possible number of shares in orders for sale for an instrument in a specific class in a client's asset portfolio based on the best bid price |
| 22 | IS_REST_SHORT_SE | DOUBLE | Attribute of an instrument allowed to be sold on borrowed funds. For MD clients the field is not filled. Valid values: _ 1 – allowed; _ 0 – not allowed |

(*) Depending on the QUIK server settings, the value may be expressed in lots or pieces. Specify the unit of measure with the broker.

Example:

```
\n
GET_CLIENT_MARGINAL_BUY_SELL_INFO ("NC0080000000", "1", "TQBR", "HYDR", 0)
\
```

8.17.4 GET_CLIENT_MARGINAL_BUY_SELL_INFO_EX

This function returns an associative array (MAP) with the parameters from the Buy / Sell Table referring to the opportunity to buy or sell a specific instrument ("sec_code") from a particular class ("class_code") identified by client ("client_code"), trading firm ("firmid"), and a preset price ("price"). If the price is set to "0", the best bid / offer values are used.

```
MAP GET_CLIENT_MARGINAL_BUY_SELL_INFO_EX (STRING firmid,
STRING client_code, STRING class_code, STRING sec_code, DOUBLE price)
```

The description of the returned parameters is found in sub-section 8.17.3.



The following additional parameters are also returned:

| No. | Parameter | Type | Description |
|-----|----------------------|--------|---|
| 1 | LIMIT_KIND | DOUBLE | Position on date. Valid values: <ul style="list-style-type: none">– 0 – T0;– 1 – T1;– 2 – T2 |
| 2 | D_LONG | DOUBLE | Effective initial discount for a long position. Parameter is filled for MD type clients |
| 3 | D_MIN_LONG | DOUBLE | Effective minimal discount for a long position. Parameter is filled for MD type clients |
| 4 | D_SHORT | DOUBLE | Effective initial discount for a short position. Parameter is filled for MD type clients |
| 5 | D_MIN_SHORT | DOUBLE | Effective minimum discount for a short position. Parameter is filled for MD type clients |
| 6 | CLIENT_TYPE | DOUBLE | Client type |
| 7 | IS_LONG_ALLO WED | DOUBLE | Attribute of an instrument allowed to be bought on borrowed funds. For MD clients the field is not filled. Valid values: <ul style="list-style-type: none">– 1 – allowed;– 0 – not allowed |
| 8 | IS_SHORT_ALL OWED | DOUBLE | Attribute of an instrument allowed to be sold on borrowed funds. For MD clients the field is not filled. Valid values: <ul style="list-style-type: none">– 1 – allowed;– 0 – not allowed |

Example:

```
\nGET_CLIENT_MARGINAL_BUY_SELL_INFO_EX ("NC0080000000", "1", "TQBR", "HYDR", 0)\n
```

8.18 File handling functions

This function is used to handle text files, and can be used to keep a program activity log, for example. File names can contain a description of the file path (for example, C:/QUIK/log/new.log).

8.18.1 CLEAR_FILE

This function clears the specified file.

```
MAP CLEAR_FILE (STRING target_file)
```



This function returns an associative array (MAP) containing the following parameters:

| No. | Parameter | Type | Description |
|-----|-------------|--------|---|
| 1 | RESULT | DOUBLE | Result of performing an operation. Valid values are 1 for successful execution and 0 for an error |
| 2 | DESCRIPTION | STRING | Operating system diagnostics in case of error |

8.18.2 WRITE

This function writes a string ("string_to_write") at the end of the "target_file".

```
MAP WRITE (STRING target_file, STRING string_to_write)
```

This function returns an associative array (MAP) containing the following parameters:

| No. | Parameter | Type | Description |
|-----|-------------|--------|---|
| 1 | RESULT | DOUBLE | Result of performing an operation. Valid values are 1 for successful execution and 0 for an error |
| 2 | DESCRIPTION | STRING | Operating system diagnostics in the event of an error |

8.18.3 WRITELN

This function writes the string "string_to_write" at the end of the "target_file" followed by a carriage return.

```
MAP WRITELN (STRING target_file, STRING string_to_write)
```

This function returns an associative array (MAP) containing the following parameters:

| No. | Parameter | Type | Description |
|-----|-------------|--------|---|
| 1 | RESULT | DOUBLE | Result of performing an operation. Valid values are 1 for successful execution and 0 for an error |
| 2 | DESCRIPTION | STRING | Operating system diagnostics in the event of an error |

Example:

```
\nCLEAR_FILE ("new.log")\nWRITE ("new.log","Hello, ")\nWRITELN ("new.log","world")\n\
```



8.18.4 GET_FILE_LEN

This function returns the number of strings in the "target_file". If such a file is not available, it returns "-1".

DOUBLE GET_FILE_LEN (STRING target_file)

8.18.5 READ_LINE

This function reads the "target_file" and returns a string numbered as "line". The string length must not exceed 1000 characters; longer strings will be truncated when read.

STRING READ_LINE (STRING target_file, DOUBLE line, DOUBLE error)

Here, "error" is the result of performing the operation. If the value is "0", the read was successful, "1" refers to an error, and "2" means that the end of the file was reached.

Example:

```
\nWRITELN ("new.log","Hello, world")\nmsg = READ_LINE ("new.log", GET_FILE_LEN("new.log"), error)\nMESSAGE (msg,1)\n\
```

8.19 String handling functions

8.19.1 LEN

This function returns the length of the string variable "value".

DOUBLE LEN (STRING value)

8.19.2 TRIM

This function deletes any spaces at the end of a line.

STRING TRIM (STRING value)

This function returns the string without empty characters at the end of the line.

8.19.3 SUBSTR

This function returns a sub-string from the string "value" beginning from the character having the number "start" and length "len".

STRING SUBSTR (STRING value, DOUBLE start, DOUBLE len)



8.19.4 FIND

This function searches for the entry of a sub-string into a preset string.

DOUBLE FIND (STRING value, DOUBLE start, STRING sub)

This function returns the position of the first entry of the sub-string ("sub") into the string ("value") starting from the position "start". If a sub-string is not found, the function returns the result "-1".

Example:

```
\
stroka="anymessage"
stroka2=SUBSTR(stroka, FIND(stroka, 1, "message"), LEN("message"))
MESSAGE (stroka2,2)
\
```

8.20 Chart handling functions

8.20.1 GET_CANDLE

This function is used to access the "candle" data in a chart and the values for the technical analysis indicators.

MAP GET_CANDLE (STRING class_code, STRING sec_code, STRING parameter_name, STRING interval, STRING graph_type, DOUBLE Date, DOUBLE Time)

This function returns an associative array (MAP) containing information about prices at a specific point in time ["Date","Time"] for the chart created for an instrument with a specific code ("sec_code") from a particular class ("class_code") and time interval ("interval"). If "class_code" is set to <space>, the function scans for "sec_code" in all classes until finding the first instance.

This type of chart is identified by the following codes:

| Chart Type | Code |
|------------------|------|
| PRICE | 1 |
| VOLUME | 2 |
| MOVING AVERAGE | 3 |
| PRICE OSCILLATOR | 4 |
| MACD | 5 |
| STANDARD DEV | 6 |
| BOLLINGER LINES | 7 |

| Chart Type | Code |
|---------------|------|
| STOCHASTIC | 8 |
| RSI | 9 |
| PARABOLIC SAR | 10 |
| SROC | 11 |
| MOM | 12 |
| ROC | 13 |
| MFI | 16 |



| Chart Type | Code |
|--------------------|------|
| WILLIAMS %R | 17 |
| ENVELOPS | 18 |
| VOLUME OSCILLATOR | 19 |
| BALANCE VOLUME | 20 |
| CUM AD | 21 |
| CHAIKIN OSCILLATOR | 22 |

| Chart Type | Code |
|----------------------------|------|
| CUM WAD | 23 |
| ELDER FI | 24 |
| ELDER RAY | 25 |
| VERTICAL HORIZONTAL FILTER | 26 |
| CHAIKIN VOLATILITY | 27 |

The value of the interval can be set using the following numeric values:

| Interval Length | Value |
|-----------------|-------|
| Month | month |
| | -3 |
| Week | week |
| | -2 |
| Day | day |
| | -1 |
| Tick | 0 |
| 1 minute | 1 |
| 2 minutes | 2 |
| 3 minutes | 3 |

| Interval Length | Value |
|-----------------|-------|
| 4 minutes | 4 |
| 5 minutes | 5 |
| 6 minutes | 6 |
| 10 minutes | 10 |
| 15 minutes | 15 |
| 20 minutes | 20 |
| 30 minutes | 30 |
| 60 minutes | 60 |
| 2 hours | 120 |
| 4 hours | 240 |

The value for "parameter_name" must correspond to one value of the parameter name from the table in the current parameters. For the list of these, see [8.12](#). If "parameter_name" is set to <space>, the search is carried out using data from the Time and Sales Table.

The value for "Date" must adhere to the format "YYYYMMDD". For example, 20050527 refers to May 27, 2005. The value for "Time" must adhere to the format "HHMMSS". For example, 163500 refers to 16:35:00 (or 4:35 pm).



This function returns an associative array (MAP) containing the following parameters:

| No. | Parameter | Type | Description |
|-----|-----------|--------|--|
| 1 | OPEN | DOUBLE | Opening price (the first transaction) for a specific time interval |
| 2 | CLOSE | DOUBLE | Closing price (the last transaction) within the interval |
| 3 | HIGH | DOUBLE | Highest trading price within the interval |
| 4 | LOW | DOUBLE | Lowest trading price within the interval |
| 5 | VOLUME | DOUBLE | Total volume of transactions within the interval |

Example:

```
\nmsg = GET_CANDLE("TQBR", "HYDR", "", "5", "PRICE", 20101130, 103500)\nMESSAGE(msg, 2)\n\
```

8.20.2 GET_CANDLE_EX

In some instances, when, for example, a technical analysis indicator consists of several lines, the use of "GET_CANDLE" is impossible. In order to access such indicators using QPILE, the function "GET_CANDLE_EX" is used.

MAP GET_CANDLE_EX (STRING Tag, DOUBLE Date, DOUBLE Time)

This function returns an associative array (MAP) containing data for a chart having a string identifier tag for the "Date" and "Time" points. Thus, in order to address chart data, a preliminary chart should be composed and assigned a unique string identifier (ID Tag). The ID Tag is set in the Chart settings dialogue box under the Advanced tab (see Chapter 4, "Working With Graphs", 4.2.4).

When using this function, the value for "Date" must adhere to the format "YYYYMMDD". For example, 20050527 refers to May 27, 2005. The value for "Time" must adhere to the format "HHMMSS". For example, 163500 refers to 16:35:00 (or 4:35 pm). In addition, this function rounds the parameter "Time" to the nearest lower unit of time within the chart interval. For example, if the chart interval is set to 5 minutes and the value for "Time" is set to 163700, the function rounds the value for "Time" to 163500; if the interval is set to 60 minutes, the function rounds 163700 to 160000.



The associative array returned by the function contains the following fields:

| No. | Parameter | Type | Description |
|-----|-----------|--------|--|
| 1 | COUNT | DOUBLE | Number of lines in the indicator |
| 2 | TIME | DOUBLE | Precise time of the candle |
| 3 | LINES | DOUBLE | Collection of lines where each element contains an associative array (MAP) |

Each element of the collection contains an associative array (MAP) having the following parameters:

| No. | Parameter | Type | Description |
|-----|-----------|--------|---|
| 1 | NAME | STRING | Name of the line (from the legend) |
| 2 | OPEN | DOUBLE | Opening price within the time interval |
| 3 | CLOSE | DOUBLE | Closing price within the time interval |
| 4 | HIGH | DOUBLE | Highest price within the time interval |
| 5 | LOW | DOUBLE | Lowest price within the time interval |
| 6 | VOLUME | DOUBLE | Total volume of transactions within the time interval |

Example:

```
slice = Get_Candle_Ex ("all", 20070511, 170000)
LineCount = Get_Value (slice,"COUNT")
time = Get_Value (slice,"TIME")
lines = Get_Value (slice,"LINES")
FOR lineID FROM 0 TO LineCount-1
line = Get_Collection_Item (lines, lineID)
open = Get_Value (line,"OPEN")
line_name = Get_Value (line,"NAME")
volume = Get_Value (line,"VOLUME")
END FOR
```

8.21 Order handling functions

These functions are used to create orders and send them to the trading system.



8.21.1 SEND_TRANSACTION

This function sends an order with the parameters shown in the array "trans_params" and, then, waits for a response from the trading system within the period "wait_timeout_for_replay" (no less than 5 seconds). Array elements "trans_params" are filled according to the rules for creating a string to import transactions from a file (for further details, see Chapter 6, "Working With Other Programs", sub-section 6.11).

MAP SEND_TRANSACTION (DOUBLE wait_timeout_for_replay, MAP trans_params)

Transactions for the withdrawal of groups of orders are not supported:

- KILL_ALL_ORDERS - withdraws all orders from the trading system;
- KILL_ALL_STOP_ORDERS - withdraws all stop orders;
- KILL_ALL_NEG_DEALS - withdraws all orders for OTC and REPO trades.

This function returns an associative array (MAP) containing the results of processing an order:

| No. | Parameter | Type | Description |
|-----|--------------|--------|--|
| 1 | RESULT | DOUBLE | Result of performing an operation. Valid values are 1 for successful execution and 0 for an error |
| 2 | RESULT_EX | STRING | Advanced diagnostics of an operation. This can have the values corresponding to the field "STATUS" in the ".tro" file when importing transactions (see Chapter 6, "Working With Other Programs", 6.11.4) |
| 3 | ORDER_NUMBER | STRING | Registration number of the order in the trading system |
| 4 | DESCRIPTION | STRING | Text comment response from the QUIK server or trading system |

Example:

```
\
new_global("trans_params", "")
new_global("trans_result", "")
trans_params = CREATE_MAP ()
trans_params = set_value (trans_params, "TRANS_ID", "333")
trans_params = set_value (trans_params, "ACTION", "NEW_ORDER")
trans_params = set_value (trans_params, "CLASSCODE", "TQBR")
trans_params = set_value (trans_params, "SECCODE", "HYDR")
trans_params = set_value (trans_params, "ACCOUNT", "L01-00000F00")
trans_params = set_value (trans_params, "OPERATION", "B")
trans_params = set_value (trans_params, "PRICE", "7.561")
trans_params = set_value (trans_params, "QUANTITY", "1")
trans_params = set_value (trans_params, "CLIENT_CODE", "")
trans_params = set_value (trans_params, "TYPE", "L")
trans_result = SEND_TRANSACTION (30, trans_params)
```




```
WRITELN ("qpile_trans.log", get_value (curr_datetime, "DATETIME") & ": " & "Result:
" & get_value (trans_result, "RESULT") & ", Result_ex: " & get_value (trans_result,
"RESULT_EX") & ", OrderNum: " & get_value (trans_result, "ORDER_NUMBER") & ",
Description: " & get_value (trans_result, "DESCRIPTION"))
\
```

8.22 Label handling functions

These functions are used to create labels and assign them in a chart.

8.22.1 ADD_LABEL

This function adds a label with preset parameters.

```
DOUBLE    ADD_LABEL (STRING tag, MAP label params)
```

A label is added to the window displaying the chart with the identifier "tag". If the addition has been successfully completed, the function returns the tag identifier or (-1) when the added tag fails.

The MAP array format contains the following label parameters:

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



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

Example:

```
\
label_params=create_map()
label_params=set_value(label_params,"TEXT", "Lable text")
label_params=set_value(label_params,"IMAGE_PATH", "image.bmp")
label_params=set_value(label_params,"ALIGNMENT", "LEFT")
label_params=set_value(label_params,"YVALUE", 2000)
label_params=set_value(label_params,"DATE", 20080616)
label_params=set_value(label_params,"TIME", 220000)
label_params=set_value(label_params,"R", 0)
label_params=set_value(label_params,"G", 0)
label_params=set_value(label_params,"B", 200)
label_params=set_value(label_params,"TRANSPARENCY", 10)
label_params=set_value(label_params,"FONT_FACE_NAME", "Tahoma")
label_params=set_value(label_params,"FONT_HEIGHT", 12)
label_params=set_value(label_params,"HINT", "Hint")
id=ADD_LABEL("ALL", label_params)
,
```

The result is represented as follows: a label with the parameters contained in the label_params array is added to the chart with the identifier "ALL".

8.22.2 DELETE_LABEL

This function deletes a label with preset parameters.

DOUBLE DELETE_LABEL(STRING tag, DOUBLE id)

Set to a preset identifier, this function deletes a label from the chart with the text identifier "tag". If the deletion is successful, the function returns the value 1; if it fails, it returns the value 0.



Example:

```
\
err=DELETE_LABEL("ALL", id)
,
```

This function results in the deletion of a label in the chart with the identifier "ALL".

8.22.3 DELETE_ALL_LABELS

This command deletes all labels in a selected chart.

DOUBLE DELETE_ALL_LABELS(String tag)

It deletes all labels set in the chart showing a diagram with the identifier tag. If this chart shows diagrams with different identifiers, the command will result in deleting labels from all diagrams, not only those with a specific value.

Example:

```
\
err=DELETE_ALL_LABELS("ALL")
,
```

This function results in the deletion of all labels in the chart.

8.22.4 GET_LABEL_PARAMS

This instruction enables one to obtain the label parameters.

MAP GET_LABEL_PARAMS(String tag, Double id)

This function returns the parameters of a label with a preset identifier. If a label with the specified identifier does not exist, then an empty MAP is returned.

- "tag" is the identifier for each indicator in the chart in which the mark is located;
- "id" is the code or serial number of the marker and starts with 1.

Example:

```
\
new_params=GET_LABEL_PARAMS("ALL", id)
,
```

This function enables one to obtain label data. If such a label does not exist, empty values are returned.



8.22.5 SET_LABEL_PARAMS

This function sets the parameters for a label with a preset identifier.

DOUBLE SET_LABEL_PARAMS(String tag, DOUBLE id, MAP new label params)

This function enables one to set new parameters for a label. If the parameter renewal is successful, the function returns a 1; if it fails, it returns a 0.

Example:

```
\nerr=SET_LABEL_PARAMS("ALL", id, label_params)\n'
```

This function results in the replacement of existing parameters with those preset by the function.

8.23 Service Functions

8.23.1 GET_TRADE_DATE

This function retrieves the date of the current trading session.

MAP GET_TRADE_DATE ()

This function returns an associative array (MAP) containing the following parameters:

| No. | Parameter | Type | Description |
|-----|-----------|--------|---|
| 1 | DATE | STRING | Trading date represented as a <DD.MM.YYYY> string |
| 2 | YEAR | DOUBLE | Year |
| 3 | MONTH | DOUBLE | Month |
| 4 | DAY | DOUBLE | Day |

Example:

```
\nwriteln(log_file_name, get_value(GET_TRADE_DATE(), "Date"))\n'
```

It writes into a file the following string:

```
02.06.2004
```



8.23.2 GET_DATETIME

This function returns the current date and time.

MAP GET_DATETIME ()

This function returns an associative array (MAP) containing the following parameters:

| No. | Parameter | Type | Description |
|-----|-----------|--------|---|
| 1 | DATETIME | STRING | Trading date represented as a <DD.MM.YYYY HH:MM:SS.sss> string where "sss" stands for milliseconds |
| 2 | YEAR | DOUBLE | Year |
| 3 | MONTH | DOUBLE | Month |
| 4 | DAY | DOUBLE | Day |
| 5 | DAYOFWEEK | DOUBLE | Serial number for the days in the week, where 0 is Sunday, 1 is Monday, 2 is Tuesday, 3 is Wednesday, 4 is Thursday, 5 is Friday, and 6 is Saturday |
| 6 | HOUR | DOUBLE | Hour |
| 7 | MIN | DOUBLE | Minute |
| 8 | SEC | DOUBLE | Second |
| 9 | MILLISEC | DOUBLE | Millisecond |

Example:

```
\nwriteln(log_file_name, get_value(GET_DATETIME(), "Datetime"))\n
```

It writes into a file the following string:

```
02.06.2004 16:57:34.460
```

8.23.3 APPLY_SCALE

This function returns a string with a number obtained by approximating the number "without_scale" to the size "scale".

STRING APPLY_SCALE (DOUBLE without_scale, DOUBLE scale)



8.23.4 IS_CONNECTED

This function is used to determine the status of the connection between the client terminal and the server. If the client terminal is connected, it returns the value 1; if the connection fails, it returns the value 0.

IS_CONNECTED ()

8.23.5 GET_INFO_PARAM

This function returns the values parameters in the information window (see menu **System / About program / Information window...**).

STRING GET_INFO_PARAM (STRING param_name)

The parameter "param_name" can have the values shown in the table below.

| Parameter Value | Description |
|-----------------|--------------------------|
| VERSION | Program version |
| TRADEDATE | Trading date |
| SERVERTIME | Server time |
| LASTRECORDTIME | Last record time |
| NUMRECORDS | Number of records |
| LASTRECORD | Last record |
| LATERECORD | Late record |
| CONNECTION | Connection |
| IPADDRESS | IP address of the server |
| IPPORT | Server port |
| IPCOMMENT | Connection description |
| SERVER | Server description |
| SESSIONID | Session identifier |
| USER | User |
| USERID | User ID |
| ORG | Organization |
| MEMORY | Memory used |
| LOCALTIME | Current time |
| CONNECTIONTIME | Connection time |

| Parameter Value | Description |
|------------------|---|
| MESSAGESENT | Messages sent |
| ALLSENT | Total number of bytes sent |
| BYTESENT | Useful bytes sent |
| BYTESPERSECSSENT | Bytes sent per second |
| MESSAGESRECV | Messages received |
| BYTESRECV | Useful bytes received |
| ALLRECV | Total number of bytes received |
| BYTESPERSECRECV | Bytes received per second |
| AVGSENT | Average rate of transfer |
| AVGRECV | Average receive rate |
| LASTPINGTIME | Time of last ping |
| LASTPINGDURATION | Delay of data during exchange with the server |
| AVGPINGDURATION | Average delay of data |
| MAXPINGTIME | Maximum time of delay |



| Parameter Value | Description |
|-----------------|-------------------------|
| MAXPINGDURATION | Maximum data delay time |

8.23.6 BREAKPOINT()

This function is used to interrupt the mode of calculation and display of the "Debug" window in which the user can view the further operation of the program.

BREAKPOINT()

This instruction does not contain any parameters and results in stopping the program execution, highlighting the next function in red, and calling up the "Debug" window in which the user can view the execution of the script code. This command can be included in the code as many times if necessary.

8.24 QPILE program debugging

The "Debug" window is used to check the execution of the script code stepwise, and allows for the tracing of code execution for programs written in QPILE. The window can be launched as described below:

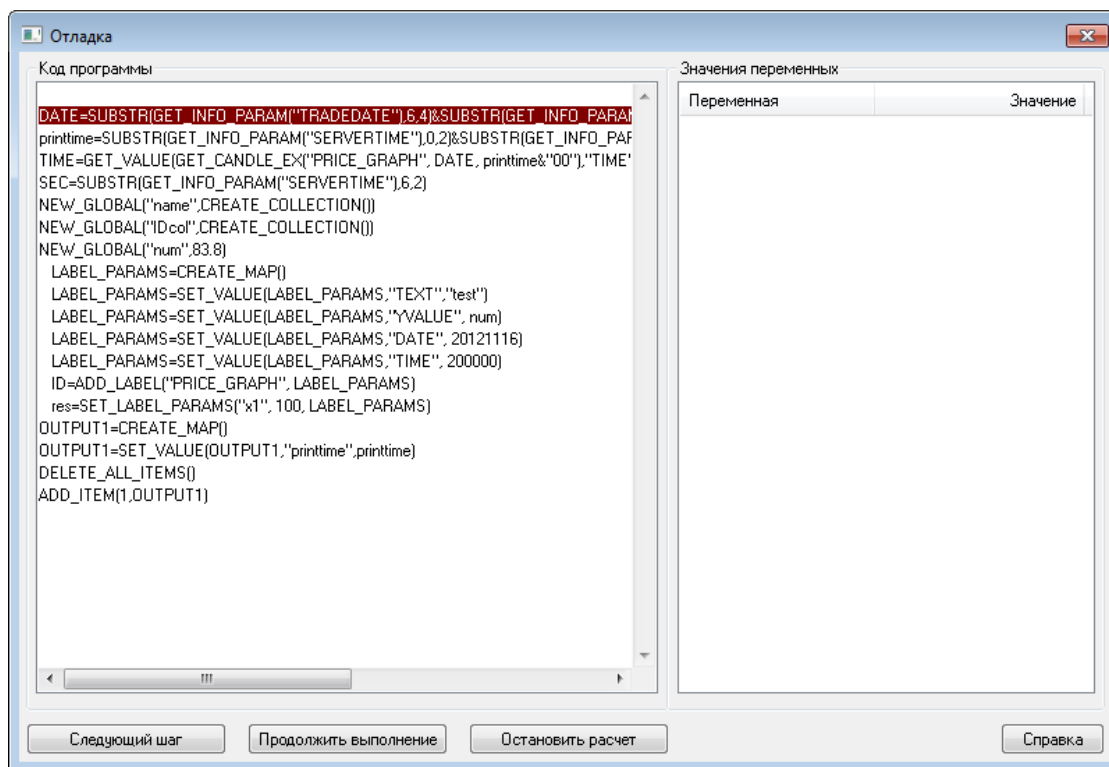
- From the QPILE table, select an instruction from the "Start calculation in debug mode" shortcut menu;
- From the program code, use the breakpoint() function.

The window contains two large fields:

- "Program code" displays the QPILE language code;
- "Values of variables" shows the variables and their values as two columns.

Once the window is launched using the breakpoint() command, the next line in the program is highlighted in red.





The "Debug" window contains the buttons listed below:

- "Next step" executes one operation in the code;
- "Continue execution" proceeds with executing the program until the "Stop calculation" button is depressed, another breakpoint() command is found, or the end of program code is reached;
- "Stop execution" stops the program on the last executed statement.

The following functions are available:

- Hit F5 to proceed with execution of the program;
- Hit SHIFT+F5 to stop the program debug and close the window;
- Hit F10 to move to the next line.



APPENDIX 1. QPILE command syntax

```
Program:
Statement_List

Statement_List:
Statement "\n"
Statement_List "\n" Statement

Statement:
NAME=Expression
IFOperator
FOROperator
FUNCDescr
CONTINUE      //skips the execution of statements until the end of the current
Statement_List
BREAK //starts the execution of the statement following the current Statement_List
RETURN //exit from the current control block (body of a function or entire program)

IFOperator:
"IF" Condition "\n"
Statement_List
"ELSE" "\n"
Statement_List
"END IF"

Condition
Condition "OR" Condition
Condition "AND" Condition
"("Condition") "
PrimaryCondition

PrimaryCondition
Expression "==" Expression
Expression "=" Expression //with the same semantics as "=="
Expression ">=" Expression
Expression "<=" Expression
Expression ">"Expression
Expression "<"Expression
Expression "!=" Expression
Expression "<>" Expression //with the same semantics as "!="

FOROperator:
"FOR" NAME "IN" ArgList "\n"
Statement_List
"END FOR"
```



```

"FOR" NAME "FROM" Expression "TO" Expression "\n"
    Statement_List
"END FOR"

ArgList
NAME    // in this case the variable NAME should contain a value of ArgList1 type
ArgList1

ArgList1:
Expression
ArgList "," Expression

FUNCDescr
"FUNC" NAME "(" FargList ")" "\n"
Statement_List
"END FUNC"

FargList
NAME
FargList "," NAME

Expression:
Expression "+" Term
Expression "-" Term
Expression "&" Term //concatenation of strings
Term

Term:
Term "/" Primary
Term "*" Primary
Primary

Primary:
NUMBER
STRINGNAME    //value for the variable NAME
"-" Primary
"(" Expression ")"
FunctionCall

FunctionCall
FNAME "(" ArgList1 ")"

NUMBER:
Digits
Digits "." DigitsSTRING:    //defined in a standard way
NAME:    //defined in a standard way

```



```
Keywords = {IF, ELSE, FOR, IN, TO, FROM, AND, OR, RESULT, FUNC, END FUNC, END FOR,
END IF, CONTINUE, BREAK, RETURN}
```

APPENDIX 2. Recommendations for writing programs in QPILE

1. Functions for handling structural variables such as "SET_VALUE", "ADD_COLLECTION_ITEM" and "REMOVE_COLLECTION_ITEM" return a modified value for the collection or array. Since all parameters are transferred to the function according to their values, the use of these functions as procedures results in the loss of any changes made to them.

Example:

```
clientscol=INSERT_COLLECTION_ITEM(clientscol,0,initmap)
'correct

INSERT_COLLECTION_ITEM(clientscol,0,initmap)
'incorrect because, in this case, clientscol will contain the same value after
calling the function as before calling it
```

2. Values returned by these functions, even those representing a real number, may appear as strings. In this case, the addition of "0" should be used to convert the result into a numeric value. For example, to properly initialize a variable, formulate "v=GET_VALUE()" as "v=0+GET_VALUE()". If this recommendation is followed, a real value for the variable is guaranteed.

Similarly, linking an operation with an empty string as "v=""&GET_VALUE()" can be used to convert a real value into a string.

3. Arbitrary user types can be formed by combining collections and associative arrays. If, for example, a list of structures of the type:

```
c=struct{
openbal:double
closebal:double
clientcode:string}
```

is required, it can be represented as a collection of associative arrays, each of them having three keys: "OPENBAL", "CLOSEBAL", and "CLIENTCODE". The code for initializing such a structure is given below:



```

initmap=CREATE_MAP()
initmap=SET_VALUE(initmap,"OPENBAL",0)
initmap=SET_VALUE(initmap,"CLOSEBAL",0)
initmap=SET_VALUE(initmap,"CLIENTCODE","")
clientscol=CREATE_COLLECTION()
FOR i FROM 0 TO 10
clientscol=INSERT_COLLECTION_ITEM(clientscol,0,initmap)
END FOR

```

After this, to access the field "OPENBAL" for the 5th client, write:

```

openbal = GET_VALUE(GET_COLLECTION_ITEM(clientscol,5),"OPENBAL")

```

If instead of a collection of such records an array with a key which is the client code is used, the values for the client's structure can be accessed without specifying its index in the array, i.e., using only the client code:

```

clientsmap=CREATE_MAP()
FOR i FROM 0 TO 10
clientsmap=SET_VALUE(clientsmap,"Q" & i, initmap)
END FOR

```

and the subsequent retrieval of the value for "OPENBAL" with client code "Q5":

```

openbal = GET_VALUE(GET_VALUE(clientsmap,"Q5"),"OPENBAL")

```

4. "MODIFY_ITEM" does not execute any action if the specified string is still missing from the table "OWN". Therefore, it is necessary to first check the availability of that string.

When creating a user table for which each iteration includes changes, the following code will be useful:

```

new_global("first_time_flag",0)

if first_time_flag==0
add_item(1, SAMPLE)
first_time_flag=1
else
modify_item(1, SAMPLE)
end if

```



When first launched, string number 1 is created which contains the values for fields from the variable "SAMPLE" previously calculated. In subsequent iterations, string number 1 is modified.

