Forest

Table of Contents

Introduction

Forest Overview

Forest is a console application developed in C++ using Microsoft Developer Studio, without relying on third-party packages. It operates on **PPL.DLL**, an implementation of the **Parenthesis Programming Language (PPL)**. Originally written in C#, the PPL code was translated into C++. PPL is an interpreted programming language in which all elements (statements, parameters, blocks) are enclosed in parentheses, resembling LISP syntax.

It includes a preprocessor to simplify code writing and reduce parentheses usage. PPL source code (**format: .scr**) is converted into an intermediate representation (**format: .ppl**) for immediate execution.

Key features of PPL
1. **Extensibility:**
- Combines C++ functionalities with custom user libraries by creating DLLs via the CodeGen.exe utility.
- Enables adding full PPL functionality to any user application.
2. **Modes:**
- **ppl (base) mode**: Syntax similar to LISP, using prefix notation for mathematical and logical expressions.
  Examples:
```
var (x [0]);
set(x) (+(1)(2));
set (x)( -(0)(+(3)(2)));  // infix notation: x = -(3 + 2);
if(==(x) (1)) …
```

- **scr (preprocessor) mode**: Syntax similar to C, using infix notation for mathematical and logical expressions.
  Examples:
```
var x = 0;
set x = 1 + 2;  // or x = 1 + 2;
if (x == 1)…
```

  Commands in format scr may be used on the left side of the expression(example).

3. **Parsing:**
  - Code written in scr mode is translated into ppl mode before execution. Each parser level creates a syntax tree.
  - Forest calls PPL API functions, which can also be used in other applications.

4. **Execution Modes:**
   - The operating mode (scr or ppl) is determined by the file extension or a command in the code itself.
   - scr mode simplifies coding by eliminating the need for enclosing statements in parentheses.
   - Default mode is set in **Configuration.Data**:   **(Code   [ppl])**

Built-In Statements and Compound Blocks

Preprocessor includes the **following statements** –
var, const, realloc,
storage statements,
array statements,
set, setkvp,cpc, sumdata,
savedata,readdata,array,
write#, writearray,call,
createstruct, insertstruct,
delegate, dlgtistant,dlgset,dlgtcall,callback,
setloopend,savedata,readdata

and following **compound statements (blocks)** –
definestruct,function, for, if, else, switch, case, default, finally,failure.

All ppl mode statements may be also added to scr code in format ppl if these statements do not have scr mode.

Data Handling

Unicode Storage: All data is saved as Unicode symbols, while digital data is converted into strings.
Examples:
```
set  x = 5.2; saved as string "5.2"
```
Boolean values are saved as strings - "**True**" and "**False**":
```
set x = True;
```

Script Execution Stages

Script execution involves:
   - Creating an array of statements (simple or compound).
   - Generating a syntax tree for each statement.
   - Traversing all nodes in the syntax tree and executing the associated procedures.
   - Repeating the process recursively for each statement in compound blocks.
   - Continuing the process for each statement in the array until completion.
   - The dstree command allows users to visualize syntax trees for each script statement.

Debugging Features

Forest includes commands for debugging:

- **dbg**: Enables debugging mode.
- **traceppl**: Traces code execution.
- **suspend and resume**: Temporarily pauses script execution and resumes it.
- **start and stop**: Measures the execution duration and outputs results in milliseconds.

Fundamentals

The code is executed in the order it is written without main function.
As is customary in many programming language guides the first ppl program is:
write("Hello World!");

Another example with using Console and String libraries:
File examples\console\colors.scr:

```
import String;
import Console;
var text = "Hello World!";
array colors[] = {Green,Red,Yellow,Cyan,Blue,Black,
                  Magenta,Gray,DarkRed,DarkGray,White,Red};

array ArrayChars;
call String.ToCharArray(text, getname(ArrayChars));

for(i,0,length(text))
{
    // function with 1 argument does not need "call"
    Console.ForegroundColor(colors[i]);
    Console.Write(ArrayChars[i]);
}
Console.DefaultColors();
write();    // to new line
```

Result:

The following 4 samples illustrate the possibilities of PPL:

```
1.
>rc examples\cowsay\cowsay1.scr
array cow[] =
{
"  _____",
"< P P L >",
" ------ ",
"        \     ^__^",
"         \   (oo)_____",
"            (__)\        )\/\",
"                ||----w |",
"                ||      ||"
};

Console.Clear();
for(i,0,length(cow))
{
    call Console.SetCursorPosition(0,i);
    call Console.Write(cow[i]);
}
call Console.SetCursorPosition(0,length(cow));
```

```
C:\ Forest              ×    +   ∨

 ------
< P P L >
 ------
        \     ^__^
         \   (oo)_____
            (__)\        )\/\
                ||----w |
                ||      ||
>|
```
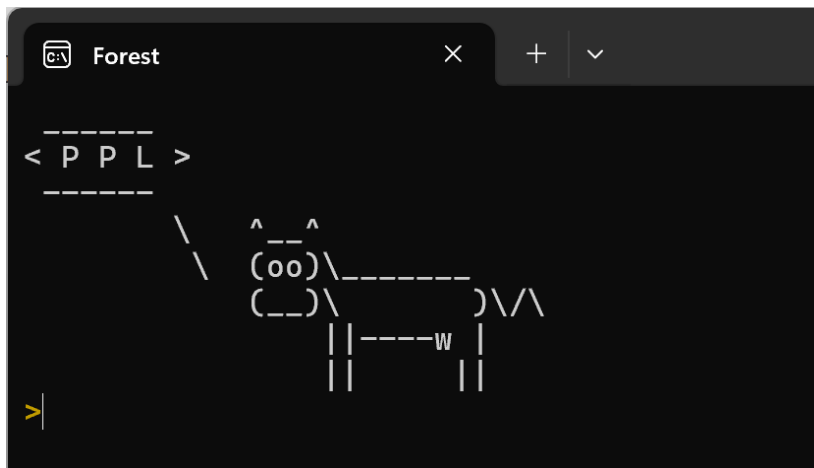
```
2.  Added loading of second file, function call, creation node and
array under this node
>rc examples\cowsay\cowsay2.scr

3. Added second operator 'for' to move cow
>rc examples\cowsay\cowsay3.scr

4. Added passing data as arguments when calling command readcode
>rc examples\cowsay\cowsay4.scr 1 1 10
```

Trees

Functions and data are stored in PPL as several kinds of Trees –
root, NS, Functions, Configuration and may be displayed on Screen, saved and restored.
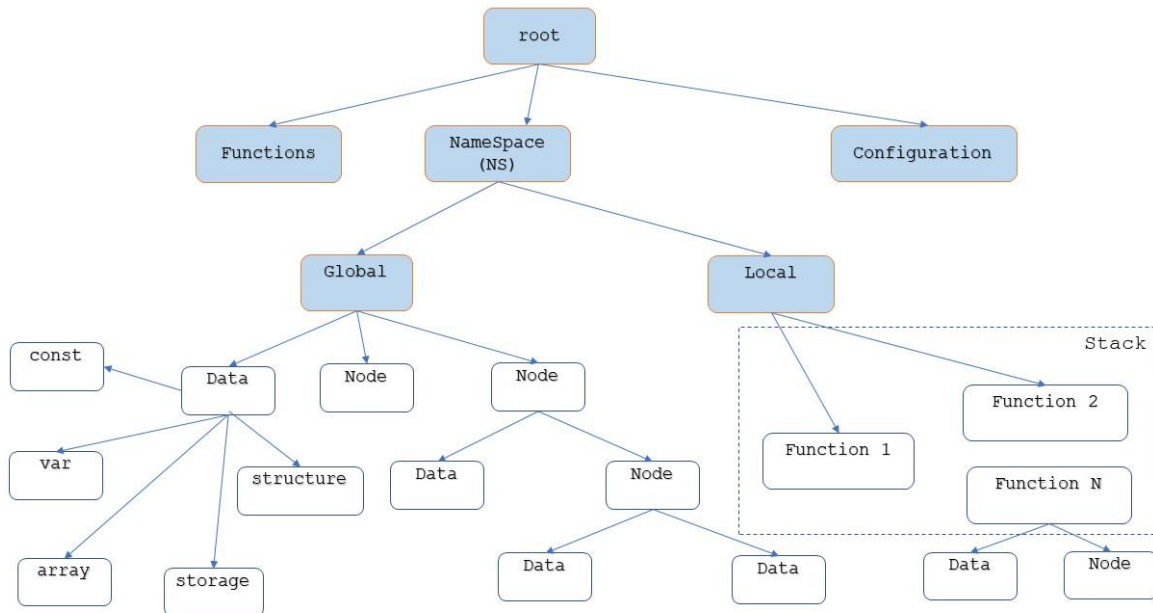
Blue nodes are created automatically when Forest.exe) loads or re-created when command
init is called.

By default Tree **Functions** is filling from file **Functions\ CommonFunctions.ppl**, defined as
"**default_loaded_functions**" in file **Configuration.data**. This may be changed by the user to
another file. To display its contents, perform:

```
>display Functions;
```

User may perform command **readcode (rc)** to read files with user's functions and add these
functions to Tree **Functions** or to node, created under Tree **Functions**.
Tree **Functions (or nodes under** Tree **Functions) saves only functions, not data (see examples
6-8 in function)**.
Examples:

```
>d Functions
-----Functions-----
-N2      Sum       [function]
---L0    result
---L1    n1
---L2    n2
---N1    #         [internal_block]
```

```
-----N2 set
-------L0        result
-------N3        +
---------L0      n1
---------L1      n2
```

Adding functions to node under Tree Functions:

```
>createnode Functions.Calc;
function
(
  Functions.Calc.Sum(result)(n1)(n2)
  (
    (set(result)(+(n1)  (n2)))
  )
);
>d Functions
-----Functions-----
-N2      Calc     [Node]
---N3    Sum      [function]
-----L0 result
-----L1 n1
-----L2 n2
-----N1 #          [internal_block]
-------N2          set
---------L0        result
---------N3        +
-----------L0      n1
-----------L1      n2
```

Tree **Configuration** is filling from file **Configuration.data**, to display its contents perform:

```
>display Configuration;
(default_loaded_functions   [Functions\CommonFunctions.ppl])
   //(UserFunctions1           [])
   //(UserFunctions2           [])
   //(UserFunctions3           [])
   //
   (Code                       [ppl])
   (debugppl                   [no])
   (delete_all_in_readcode     [yes])
   (log                        [no])
   (stay_interactive           [no])      // for Forest.exe
   (OFD_port                   [11000])   // for Forest.exe
   //(UserFunctions1            [Functions\printchar.ppl])
   (UserImport1                [Directory])
   (UserImport2                [Math])
   (UserImport3                [String])
   (UserImport4                [File])
   (UserImport5                [Console])

   //(UserImport6               [Convert])
   //(UserImport7               [ArrayList])
   //(UserImport8               [Stack])
   //(UserImport9               [Queue])
```

```
   //(UserImport10                [DataFrame])
   //(UserImport11                [Dictionary])
```

Any public variables (var, const, array, storage and node) are saved in Tree **Global** as common data for all functions and for code without functions(in "main function").

Public functions are available to functions from any node in Tree **Global**.

Private variables and functions are available to functions in the node, that they belong to only. Full name variables and functions include name of node.

Example:

```
createnode N1;
   function
   (
      N1.f()
      (
         (write("public function N1.f"))
         (N1._f())
      )
   );

   function
   (
      N1._f()     // private
      (
         (write("private function N1._f"))
         (write(N2.x))
         //(write(N2._x))     // Error: [GetValue] [N2._x] private
                                          object, no access
         //(N2._f())          // Error: [Traversal] [N2._f] private
                                          function, no access

      )
   );

createnode N2;
   var(N2.x["public var N2.x"]);
   var(N2._x["private var N2._x"]);
   function
   (
      N2._f()    // private
      (
         (write("private function N2._f"))
      )
   );

N1.f();
write(N2.x);
//write(N2._x);    // Error: [GetValue] [N2._x] private object, no
                         access
```

Variables for functions are created in Tree **Local**, to display its contents perform in function:
```
>display Local
```
When exiting a function, its variables are deleted.

For illustration difference between modes scr and ppl consider the following examples:

```
>rc Examples\scr\for.scr
=== scr code for preprocessor ===
var begin = 0;
var end = 3;
for(i,begin + 1,end + 1,1)
{
  write(i);
}
=== generated by preprocessor ppl code ===
>var (begin[0]);
>var (end[3]);
>loop (i)( + ( begin )( 1 ) )( + ( end )( 1 ) )( 1 )
(
  do
  (
    (write(i))
  )
);
Result:
1
2
3
```

Statement terminator ';' always follows after each type of statements in scr mode.
In ppl mode statement terminator ';' does not follow after statements within compound statements(blocks) – loop, switch,if,function.

Examples in ppl mode:
```
loop (i)(0)(3)(1)
(
    do
    (
      (write(hello))
      (write(world))
    )
);
```

Comments

Two kinds of commentaries are possible:
/*…*/ - for several lines of code
and
// - for one line of code or part of line.

Configuration

Configuration is defined in the file **Configuration.data**, meaning of its members is explained in this tutorial.

Identifiers and DNS

Names of nodes, variables, arrays, storage and functions contain any symbols, first symbol is any upper or lower case letter or any of the following symbols: **_$#** , but not a digit. Variables with first symbol **"_"** in name are hidden or private variables (see hidden variables).
Name **"all"** can not be used (see cmd del).
Length of identifiers is not limited. Do not set keywords and names of Libraries as identifiers. When data is created, its full name and saved address are added to **Data Names Structure (DNS)**. DNS creates separately for non-functions identifiers in Global and for each function in Local, DNS of function will be destroyed when exiting the function.
Symbolic values are enclosed in quotation marks, to include a quotation mark in a symbolic expression, precede it with backslash.
Example:
```
"123\"qwe" => "123"qwe"
```
Backslash before the last quote mark it is backslash, not quote mark.
```
"123\qwe\\" => "123\qwe\"
```

Public and private variables and functions

Variables, constants, arrays, storage and functions, whose names start with **underscore** are private, all other are public.
Examples:
1.

```
>code ppl;
>createnode N1;
>var(N1.x["public var N1.x"]);
>var(N1._x["private var N1._x"]);
function
(
   N1._f()      // private
   (
      (write("private function N1._f"))
      (write(N1.x))
      (write(N1._x))
   )
);
function
(
   N1.f()
   (
      (N1._f())
   )
);
N1.f();
```

Result:
```
code: ppl
private function N1._f
public var N1.x
private var N1._x
```

2.

```
>code scr;
>createnode N1;
>var N1.x = "public var N1.x";
>var N1._x = "private var N1._x";
function N1._f()      // private
{
    write("private function N1._f");
    write(N1.x);
    write(N1._x);
}
function N1.f()   // public
{
   N1._f();
}
N1.f();
```

Error occurs when re-creating a variable, it is possible to delete this variable and to create again:

```
>code scr;
>var x;
>varx;    // re-creation
Error: [FuncCreateVariables] name [x] already exists
>del x;
>var x;
```

Compound statements (blocks)

Compound statements include one or several statements enclosed in curly brackets:

```
if (x == 1)
{
    write("COMPOUND");
    write("STATEMENTS");
}
else
{
    write("compound");
    write("statements");
}
```

If compound statements **"for"**, **"if"** contain only one **not compound statement** in curly brackets it is possible to omit brackets:

```
if (x == 1)
    write("COMPOUND");
else
    write("statements");
```

```
if (x == 1)                          // right
{
    if (y == 2)
        write("right sample");
}
```

```
if (x == 1)                          // wrong
    if (y == 2)
    {
        write("wrong sample");
    }
```

Libraries

Default name of library is **Main**, it loads always when Forest.exe starts. It is possible to set in file **Configuration.data** as **"UserImportN"** names of additional libraries initialization loaded .
To display list of loaded libraries perform:

```
>importlist;
Main
Directory
Math
```

To display contents of any library perform:
<name of library>.help
or **help** for Main library

Example:

```
>Directory.help;
help
GetFiles
GetDirectories
SetCurrentDirectory
GetCurrentDirectory
```

To get short information about any library function perform:
<name of library>.help(function name)

```
>Math.help(Sinh)
     Returns the hyperbolic sine of the specified angle:
     Math.Sinh(double value)
```

```
For Main Library help  or ?:
>? d
        display | d [root|NS| Functions|Local|node name]
        display NS.namespace.name]
```

Keywords

Keyword formats are defined in this tutorial, all format are defined for **ppl mode** by default. Additionally defined format for **scr mode** for some keywords. All keywords are divided into 11 groups and presented below:

Service Commands

> **help, version, cls, shell, init, code, showcode, readcode, fdreadcode, createpplcode, display, displaynodes, dn, dstree, datanames, suspend, resume, debugppl, traceppl, recreate, log, exit, createcodeppl, sumdata, start, stop**

Special Commands

> **import, importlist, eval, length, calc, sleep,**
> **isexist, isdigits, isinteger, isalldigits, isallinteger, iseven, isodd,**
> **del, getbykey, getbyvalue, set, setkvp, getvalue, getname , gettoken,type**

Nodes and Leaves

> **createnode, copynode, getnodes, getleaves**

Structures

> **defunestruct,insertstruct,createstruct,dstruct,dd**

Variables and Arrays

> **var, const, array, realloc,**
> **array.push, array.pop, array.reverse, array.shift, array.remove,**
> **array.clear, array.unshift, array.insert, array.slice,**
> **array.sum, array.copy, array.min, array.max, array.average, array.first,**
> **array.last, array.concat**

Storage

> **storage, sinit, sget, sset, swrite, sinfo, ssetrow**

Backup and Recovery

> **savedata, readdata**

Control Flow

> **if, else, switch, case, default, loop, do, for, break, continue, setloopend.**

Input Output

> **write, writeline (write#), writearray, readline**

Functions

> **function, funclist, funcname, argc, getargname, call, return, getresult**

Delegates and callbacks
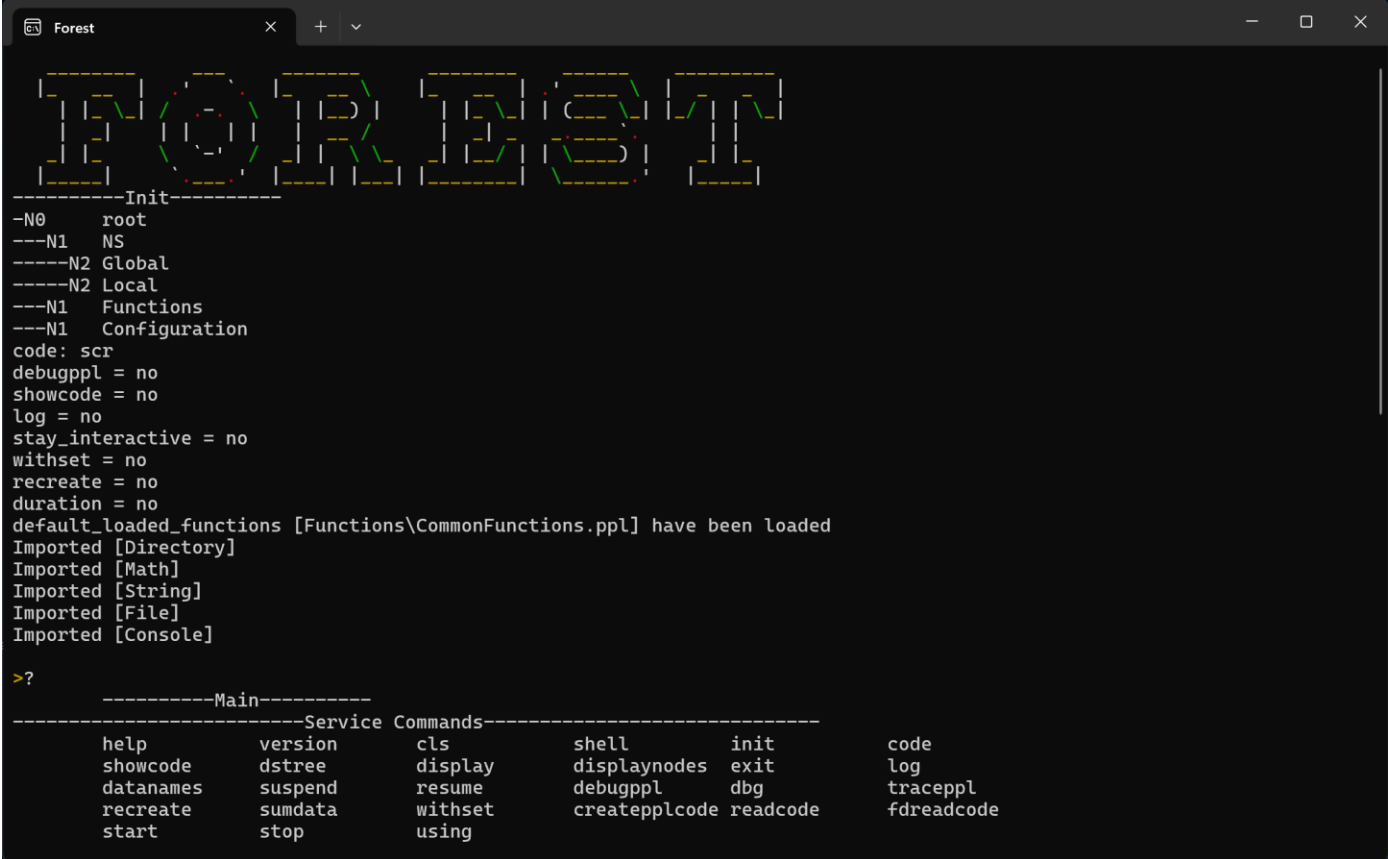
      delegate, dlgtinstance, dlgtset, dlgtcall, callback

Special variables, constants and words

      empty – see methods ArrayList.Add, Queue.Enqueue, Stack.Push

      tab,comma,space – see String.Split, String.Splitcsv.

Forest utility

Command-line REPL utility **Forest.exe** is a PPL interpreter which syntax and keywords are given in this tutorial. This utility is written in C++ without any third party packages.

These are following subdirectories and files used to work with Forest.exe
Subdirectories:
\Data
\Examples
\Functions
\JsonHelp
\CodeGen
Files:
Configuration.data
FOREST.exe,Assistant.exe, DLL's.

Set Screen Buffer Size and Window Size in Properties\Layout.

There are 2 operating modes in accordance with Forest.exe arguments:
1. **NonInteractive mode**

Execute program in file with extension scr or ppl.

Forest.exe file [arg1 arg2 …]

file := file.ppl|file.scr

If arguments are present, they override the variables 1, 2 and so on in the body of the called file. Number between two symbols $ is the serial number of argument.

An error occurs if arguments quantity less than max variable number.

Value of argument is literal, not command.

When value of **stay_interactive** in file **Configuration.data = "no"** Forest.exe finishes after program execution**,** when value of **stay_interactive** = **"yes"** Forest.exe does not finish and continues in interactive mode.

Example:

File example.scr

```
var $1$ = $2$;
>Forest.exe example.scr x 2;
```

2.Interactive mode

Forest.exe

Command input from standard input stream.

To get list of commands and their short explanation perform **help** (or ?).

Examples:

```
>? display;
        display | d [root|NS|Functions|Local|node name]
>? d;
        display | d [root|NS|Functions|Local|node name]
```

Prompt "**>**" appears on Screen before each command.

Examples:

```
>display;
-N1     NS
---N2   Global
```

In interactive mode **set** command must be present:

```
>var x; x=1;    // wrong
>var x; set x=1;    // right
>var x;             // right
>set x=1;
```

In addition to commands required to work with scr/ppl programs, Forest.exe allows you to execute all Windows commands and save the results. Command **shell** uses for that.

Examples:

```
>var (x);
>set(x)(shell(cd));      // output is saved in var x
>write(x);
```

The following often used commands and operators with one parameter may be used with or without parentheses around arguments:

help (?), import, readcode (rc),showcode, createnode, isexist, display (d), displaynodes(dn),del, code,debugppl (dbg).

PPL Assistant

PPL Assistant displays format commands in ppl and scr modes. As well this application displays format methods in additional libraries.
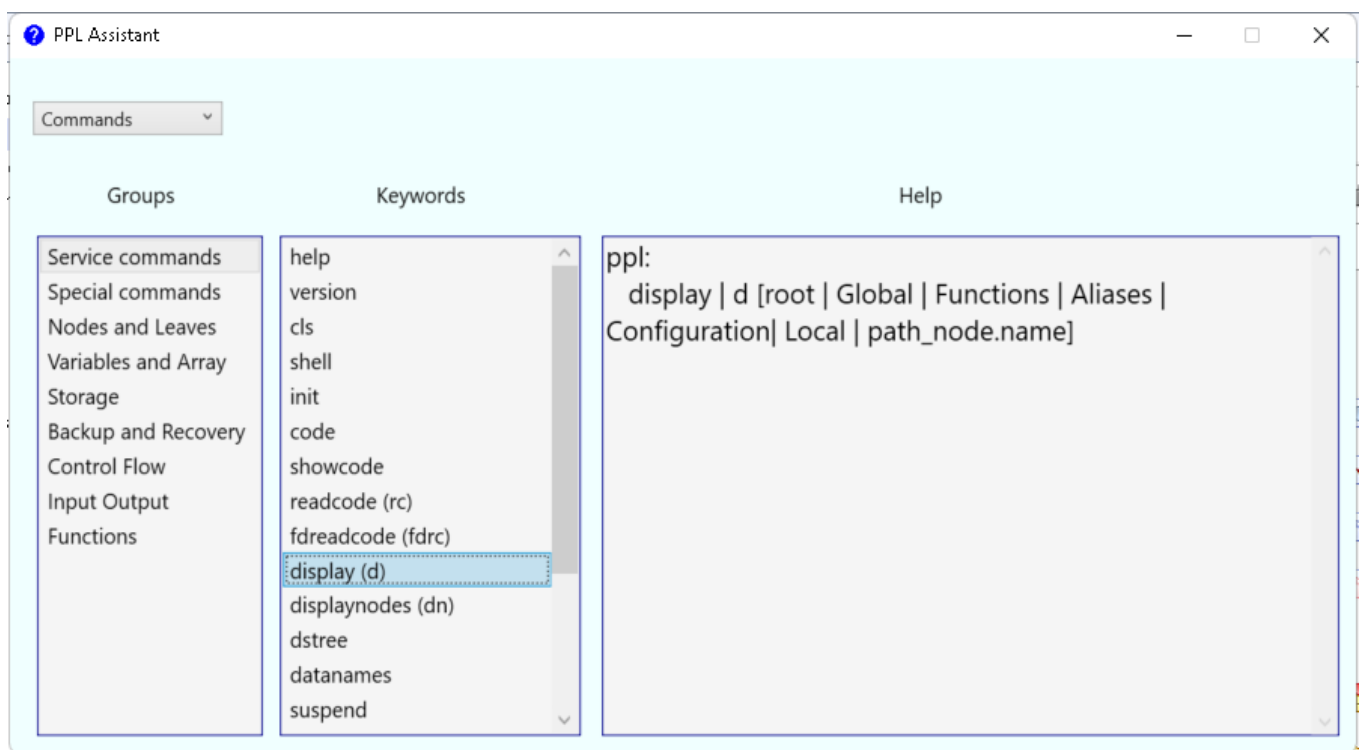
Service Commands

help

Displays keywords list for Library by <name> or
format of command from Main library.
Library must be loaded before (see <import>)
by default name = Main, this library is loaded automatically
Format: **help | ? keyword**

```
Forest                    X   +  v                                                                    —  ☐  X
>?
        ----------Main----------
--------------------------Service Commands----------------------------
    help          version       cls           shell         init          code
    showcode      dstree        display       displaynodes  exit          log
    datanames     suspend       resume        debugppl      dbg           traceppl
    recreate      sumdata       withset       createpplcode readcode      fdreadcode
    start         stop          using
--------------------------Special Commands----------------------------
    import        importlist    eval          length        type          del
    isexist       isdigits      isinteger     calc          sleep
    isalldigits   isallinteger  iseven        isodd         getname
    getbykey      getbyvalue    set           setkvp        getvalue
    gettoken
--------------------------Nodes and Leaves----------------------------
    createnode    copynode      getnodes      getleaves
--------------------------Structures----------------------------
    definestruct  createstruct  insertstruct  dstruct       functionstruct
    instancename
--------------------------Variables and Arrays----------------------------
    var           const         setconst      array         realloc
    array.push    array.pop     array.reverse array.shift   array.unshift
    array.remove  array.clear   array.insert  array.slice   array.sort
    array.sum     array.min     array.max     array.copy
    array.average array.first   array.last    array.concat
--------------------------Storage----------------------------
    storage       sinit         sget          sset
    swrite        sinfo         ssetrow
--------------------------Backup and Recovery----------------------------
    savedata      readdata
--------------------------Control Flow----------------------------
    if            else          switch        case          default
    loop          do            for           break         continue
    setloopend
--------------------------Input Output----------------------------
    write         write#        writeline     writearray    readline
--------------------------Functions----------------------------
    function      call          funclist      funcname      argc
    getargname    return        getresult
--------------------------Delegates and Callbacks----------------------------
    delegate      dlgtinstance  dlgtset       dlgtcall      callback

    to get short explanation of each command: help command
>|
```

Examples:
```
>? Code;
     Sets mode for Console input or displays on Screen
     code ppl | scr
>? "display"    // only display in quotes
     display | d [root|NS| Functions|Local|node name]
     display [NS.namespace.name]
```
Any other library has function help for display its contents.
<name of library>.help [(keyword)]
```
>Matrix.help (Rotate)
```

version

Display current version
Format: **version**

cls

Clears the Screen
Format: **cls**

shell

Executes Windows Console Commands, several commands are hash symbol separated.
Results of shell may be saved or displayed by commands **write, write#** or by
 debugppl yes
Format: **shell (command with parameters[#command with parameters])**

Examples:
```
>write(shell (cd:\));
>dbg yes;
>shell (dir /b tests#cd);
```

init

Deletes all data and functions and creates new root, use this command for
console input only.
Format: **init**

code

Sets mode for Console input or displays it on Screen.
Mode scr is more convenient for writing code with blocks and for using infix
expressions. But in other cases there is no difference.
Format: **code [ppl|scr]**
default - ppl
It is possible to set **code** in file **Configuration.data**.

showcode

Shows or hides on Screen ppl_code when command readcode is executed or displays
showcode value on Screen
Format: **showcode [yes|no]**
Default: no

Examples:
```
>showcode no;
```

readcode(rc)

Reads file with code in format scr or ppl.
At the end of the execution readcode the previous code (scr or ppl) will be set.
Format: **readcode | rc <file.scr|ppl> [arg1 arg2 ...]**
If arguments are present, they override the variables 1, 2 and so on in
the body of the called file.
Number between two symbols $ is the serial number of argument.
An error occurs if arguments quantity less than max variable number.
Arguments are literals, not commands.
If **delete_all_in_readcode = yes** in **Configuration.data** command
delete all is added automatically as first command when first command **readcode** is
called.
The file being called may also include readcode commands. Files called by command
readcode may be of different formats - .scr or .ppl.
If caller script and internal scripts are in the same path you may omit path for internals.
You may specify relatively path from Forest.exe or full path, including drive:

```
>rc path\file.scr|ppl
  or
>rc drive:\path\ file.scr|ppl
```

Examples
1.
```
> Directory.SetCurrentDirectory(examples\ppl);
>rc loop.ppl;   // or rc examples\ppl\loop.ppl
loop (i)(0)(3)(1)
(
   do
   (
      (write("PPL"))
      (write("ppl"))
   )
);
```
2. File example.scr
```
   var $1$ = $2$;
   //pass arguments via rc
   >rc example.scr x 2;
```
3. reading script with command readcode inside:
```
File main.scr:
   write("main script");
   rc examples\rc\first.scr;
   rc examples\rc\second.scr;
   write("return from main script");
File first.scr:
```

```
     write("first script");
     rc examples\rc\third.scr;
     write("return from first script");
File second.scr:
     write("second script");
File third.scr
     write("third script");

     > rc examples\rc\main.scr;
Result:
     main script
     first script
     third script
     return from first script
     second script
     return from main script
```

fdreadcode

Like readcode with using FileDialog to select file. This command loads ofd.exe and sets connection with **Forest.exe** via UDP protocol, port defined in file **Configuration.data** as **OFD_port**.
Format: **fdreadcode|fdrc**

createpplcode (cpc)

Creates file in format ppl from file (or files) in format scr.
Format **ppl**: **createpplcode|cpc (file.scr) [(file.ppl) [(all)]]**
Format **scr**: **createpplcode|cpc [file.scr file.ppl [all]]**
arg **all** – include all files, loaded by command **readcode.**

Examples
```
> code ppl;
>createpplcode (ttt.scr) (ttt.ppl);
>cpc (ttt.scr) (ttt.ppl);

>code scr;
>cpc ttt.scr;
```

display (d)

Displays nodes(N) and leaves(L) in Tree.
Format:
display | d [root | Global | Functions | Configuration| Local | path_node.name]
default: **Global**

Increase Screen Buffer Size.Height in FOREST.exe Properies\Layout in case large number of lines.

Examples:
```
 >array(y[2])(0);
 >d;
-N  NS
---N  Global
-----N  y
-------L0               [0]
-------L1               [0]
>d Global.y;
>d y;
>d Functions.Sum;
```

displaynodes (dn)

Displays nodes(N) only.
Format:
displaynodes | dn [root | Global | Functions | Configuration| Local | path_node.name]

Increase Screen Buffer Size.Height in FOREST.exe Properies\Layout in case large number of lines.

Example:
```
>dn Functions
-----Functions-----
-N3     Sum     [function]
---N1   #       [internal_block]
-----N2 set
```

```
-------N3          +
-N3      Sub      [function]
---N1    #        [internal_block]
-----N2 set
-------N3          -
```
…………………………………

dstree

Displays syntax tree per each statement in script and corresponding statement in format ppl.

dstree may be entered from script only, not from console in interactive mode.

dstree is for one-time running, it is needed to enter it each time before presentation.

Format: **dstree() | dstree**

Examples:
File **examples\if.scr**

```
dstree;
for(i,0,2)
{
   if(i == 0)
   {
      write#("i == 0");
      continue;
   }
   else
     write#("i != 0 = {0}",i);
}
>rc examples\if.scr;
--N0     root
----N1   loop
------L0            i
------L1            0
------L2            2
------L3            1
------N2            do
--------N3          #                   [internal_block]
----------N4     if
------------N5   ==
--------------L0            i
--------------L1            0
------------N5   #                   [internal_block]
--------------N6           write
----------------L0         "i == 0"
--------------N0           continue
----------N4     else
------------N5   #                   [internal_block]
--------------N6           write
----------------L0         "i != 0 = {0}"
----------------L1         i
code: scr
i == 0
i != 0 = 1
```

datanames

Displays contents of DNS.
Format: **datanames [Local]**

Examples:
```
>var(x);
>createnode Node1;
>array(Node1.arr[5]);
>var(Node1.y);
>datanames;
-----------Global_dns---------------
node              name              type
------------------------------------
                  empty             const
                  x
Node1             arr               array
Node1             y                 var

>datanames Local;      // for using in functions
```

suspend and resume

Stops script to perform manually one or several commands in **ppl-mode**,
continue script execution – **resume**
stop script – **exit** and double click
Format: **suspend**

Examples:
```
>Enter:
>d
-N1      NS
---N1    Global
>resume  // continue script execution
```

debugppl (dbg)

Displays information about creation and deletion variables, results operations and
duration or displays debugppl value on Screen.
Format: **debugppl | dbg [yes|no]**
It is possible to set **debugppl** in file **Configuration.data**.

Example:
```
>var(x)
>debugppl yes
>duration = 0.0015026
>del x
```

```
leaf [x] is deleted
>duration = 0.0054401
```

traceppl

Displays all commands and function names on screen during the execution of commands. By default – traceppl no.
Format: **traceppl [yes|no]**

Example:
>traceppl yes;

recreate

Permits recreation vars,arrays, storage and nodes. By default – recreate no.
Format: **recreate [yes|no]**

Example:
```
>recreate yes;
>code scr;
code: scr
>var x;
>var x;
>recreate no;
>var x;
Error: [FuncCreateVariables] name [x] already exists
```

log

Writes commands and results to logfile in directory **Log** or displays log value on Screen.
Format: **log [yes|no]**
It is possible to set **log** in file **Configuration.data**.
Opened logfile will be closed by command init or exit.

exit

Exit from Forest.exe **exit** or exit from script **exit()**.

sumdata

Defines argument type for summation. By default – **sumdata digit**.
Format:
sumdata [digit | string]

```
Example:
>code scr;
>sumdata digit;
>write#(1+2);
3
>sumdata string;
>write#(1+2);      // scr-format
12
>write(1+2);      // error: ppl-format
1+2
>write(+(1)(2);   // right: ppl-format
12


>sumdata string;
>var x = 1;
>call Sum(x,2);
>write#("x={0}",x);
x=12
```

It is possible to sum more than 2 items:
```
>sumdata string;
>var x;
>x = "a"+"\"" +"c" + 1;
a"c1
>sumdata digit;
>var y;
>y = 1+2+3+4;
10


sumdata digit;
function f(a,b,c,q)
{
   q = a + b + c;
}
var res;
call f(1,2,3,res);
write#("res={0}",res);

sumdata string;
call f(1,2,3,res);
write#("res={0}",res);
sumdata digit;
res=6
res=123
```

withset

Adds or no operator 'set' in code
Format:
withset [yes | no]
by default **no**.
It is possible to set in file **Configuration.data**.

start, stop

For measuring duration of a script or its part.
Format:
start | start()
stop | stop (text)

Example:
```
start;
array x[10];
for(i,0,length(x))
{
    x[i] = i;
}
writearray x row;
stop("test duration:");result:
-----Array x-----
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
test duration: 0.7918 msec
```

using

      Creates nickname for long name

Format **ppl**:

      using (short_name) (long_name)

Format **scr**:

      using short_name = long_name

Example:

```
>code scr
>using x = xxxxx
>var xxxxx
>x = 0
>d
-N2      NS
---N3    Global
-----L0 empty    (const)
-----L1 xxxxx    [0]
---N3    Local
>write#("x = {0}",x)
x = 0
>write#("xxxxx = {0}",x)
xxxxx = 0
```

Special Commands

import

Loads Library from current directory or from user directory.
Format:
import Library name | Directory\Library name

Examples:
```
>import Math;
>import DLL\Erato;    // (see examples\scr\erato.scr)
```

Importlist (il)

Displays list of loaded Libraries
Format: **importlist | il**

Examples:
```
>importlist;
Main
Math
```

eval

Performs string in format ppl.
Format: **eval ("ppl expression") | eval (variable_name)**

Examples:
```
>var x = "var(r); set(r)(/(180)(Math.PI())); write(r)";
>eval(x);
Result:57.29577951308232
>write(eval("+(1)(2)"))
Result:3
```

```
>var x = "array (y)(1)(2)(3);d;"
>eval(x)
-N2      NS
---N3    Global
-----L0        empty      (const)
-----L1        x [array (y)(1)(2)(3);d;]
-----N4 y                               [Array 3]
-------L0        #        [1]
-------L1        #        [2]
-------L2        #        [3]
```

```
>code scr
```

```
>array y[3]
>y[0] = "write(\"Hello World\")"
>eval(y[0])
Hello World

>code ppl
code: ppl
>storage(stor)(2)(2)
>sset(stor)(0)(0)("var (x[0];d)")
>eval(sget(stor)(0)(0))
-N2      NS
---N3    Global
-----L0        empty      (const)
-----N4 stor                              [Storage 2 2x2]
-------N5       0                              [Array element]
---------N6     Row                            [Array 2]
----------L0        #    [var (x[0];d)]
----------L1        #
-------N5       1                              [Array element]
---------N6     Row                            [Array 2]
----------L0        #
----------L1        #
-----L1         x [0]
---N3    Local
```

```
function f(x)
{
    write#("x = {0}",x);
}
var y = "f(\"Hello\");write(\"x36 is ended\")";
eval (y);
Result:
x = Hello
x36 is ended
```

length

Returns length of value for var | const or length array | storage
Format: **length (var | const name | array name|storage name)**

Examples:
```
>array y[3];
>write(length(y));
Result: 3
>var (x["Hello!"]);
>write(length(x));
Result: 6
>var c = "Hello";
>write(length(c));
Result: 5
```

isexist

Determines whether var, array or storage with specified name exists or not in Global or Local, returns "**True**" or "**False**".
Format: **isexist(name)**
name:= [NS.][namespace.][node.]name

Example:
```
1.
>code scr;
>dbg yes
>var x;
>isexist (x);
Result: True
2.
>createnode Functions.New;
>isexist (Functions.New);
Result: True
```

isdigits

Checks is value of var or member of array or storage digital, returns "**True**" or "**False**".
Format: **isdigits(var name | member of array or storage | literal)**

Example:
```
>code scr;
>dbg yes;
>var x = 1.1;
>isdigits (x);
Result: True
```

isinteger

Checks is value of var or member of array or storage integer, returns, returns "**True**" or "**False**".
Format: **isinteger(var name | member of array or storage | literal)**

Example:
```
>code scr;
>dbg yes;
>var x = 1;
>isinteger (x);
Result: True
```

Isalldigits

Checks is all members of array or storage digital, returns, returns "**True**" or
"**False**".
Format: **isalldigits(member of array or storage)**

```
Example:
>code scr;
>dbg yes;
>array x[] = {1,2,3};
>isalldigits (x);
Result: True
```

isallinteger

Checks is all members of array or storage integer, returns, returns "**True**" or
"**False**".
Format: **isallinteger(member of array or storage)**

```
Example:
>array(x)(1)(2)(3);
>isallinteger (x);
Result: True
```

iseven, isodd

Checks is integer value even or odd, returns "**True**" or "**False**".
Format:
iseven(var name | member of array or storage | literal)
isodd(var name | member of array or storage | literal)

del

Deletes any kinds of data from Global or Local Tree, also deletes nodes from Functions.
Format: **del (fullname) | del fullname**
To delete all Global contents**: del all,** so name "**all**" can not be used as any kind of variable names.
If **"delete_all_in_readcode" = yes** in **Configuration.data** command
delete all adds automatically as first command when command **readcode** is called (**not for readdata**). Otherwise all data will be saved in memory, if necessary add this command manually.
fullname:= node path.name
node path:= node path | node

Example:
```
>createnode Node1;
>var(Node1.x);
//the following line removes Node1 and Node1.x
>del Node1;
>createnode Functions.Geo
>del Functions.Geo
```

To re-run script, that creates array or use command **recreate yes**:
```
if(isexist(y) == True)
    del y;
array y[5];
```

If **"delete_all_in_readcode" = yes**

command "del all" added for external readcode only, not for internal, to save for using all varaibles, defined in external readcode.

Example:

```
File examples\x51.scr
var x = 0;
rc examples\x51.scr;
d;

File examples\x51.scr
array y;

>rc examples\x50.scr
Result:
code: scr
code: scr
-N2      NS
---N3    Global
-----L0        empty                    (const)
-----L1        x                        [0]
-----N4 y                               [Array 0]
---N3    Local
```

calc

Calculates infix notation math. expression and writes result on screen, may be used for ppl and scr modes, **but in interactive mode only**, not in .ppl or .scr files.
Limitation: do not use expression for calculation indexes to array elements.
Format: **calc math.expression**

Example:
```
>code ppl;
>var (x[1]);
>calc x + 2*Math.PI();
7.283185307179586
```

```
>code scr;
>array arr[] = {1,2,3};
>var x = 1;
>calc 1+ arr[0];
>calc 1+ arr[1+1];    // error: calculation  indexes
> var y = 1+1;
>calc 1 + arr[y];     // right
```

```
>calc Math.Sqrt(1+3) + 1;
```

sleep

Suspends the interpreter for the specified number of milliseconds
Format: **sleep(msec)**

Example:
```
>sleep(100);
```

getbykey (getk)

Gets value from array by name.
Format**: getbykey | getk (name array)(name element)**

Example:
```
See example in readdata
>getbykey(Colors)(Black);
Result: 0
```
if key is absent return **nan**.

getbyvalue (getv)

Gets name from array by value.
Format: **getbyvalue | getv (name array)(value element)**

Example:
```
See example in readdata
>getbyvalue(Colors)(0);
Result: Black
```
if value is absent return **nan**.

set

Sets value for variable and array element
Format **ppl**:
set (var_name | array_name [index]) (value | array_name [index])
index:=value| ppl expression
Format **scr**:
set var_name | array_name [index] = value | scr expression
index:=value| scr expression

Command **set** checks whether index is out of bounds.

Examples:
```
>code ppl:
>var (x);
>set(x)(+(1)(2));
>array(y[3]);
>set(y[+(1)(2)])(0);

>code scr:
>var x;
>set x = 1;
>array y[3];
>set y[x + 1] = 2 + 3;
>set y[0] = y[1];
```

```
>set y[3]=0;
Error: [SetElementArray] wrong index 3 array y out of of bounds
```

set may be omitted in **scr-mode**:
```
>x = 3;
>y[0] = 2+3;
```

```
>code scr:
>var x;
>set x = d;        // error, command "d" is running
>set x = "d";
>eval(x);          // this statement runs command "d"
```

To calculate indexes for access to array elements command set in scr-mode creates
 temporary variables and deletes them at the end:
file test.scr
```
array a[]  = {1,2,3,4,5,6,7};
set a[1+2] = a[2+2]+ 1;
```
>rc test.scr
The following ppl-code will be generated:
```
array (a)(1)(2)(3)(4)(5)(6)(7);
var (#0[ + (1)  (2) ]);
var (#1[ + (2)  (2) ]);
set (a[#0])(+(a[#1])(1));
del #0;
del #1;
```
It is possible to use logical and comparison operands:
```
>code scr;
>var x;
>set x = ((1==1) && (2==2)) && (3==3);
>write(x);
Result: True
```

```
>code scr
array x[3] = 0;
set x[1+1] = ((1==1) && (2==2)) && (3==3);
writearray x row;
```
The following ppl-code will be generated:
```
>array(x[3])(0);
>var (#0[ + (1) (1) ]);
>set (x[#0])( &&  ( &&  ( ==   (1) (1) ) ( ==   (2) (2) ) ( ==
                          (3) (3) )  ) );
>del #0;
>writearray (x ) ( row );
```

```
>var x;
>set x = 1==1? t:f;
>write(x);
```

The following ppl-code will be generated:

```
var (x);
if (== (1) (1) )
(
   (set (x)(t))
);
(
  else
  (
     (set (x)(f))
  )
);
write(x);
```

```
To set data in structure see (sample).
```

setkvp

Sets key and value array element
Format **ppl**:
setkvp(array_name [index])(key)(value | ppl expression)
index:=value| ppl expression
Format **scr**:
setkvp(array_name [index]) = key, value | scr expression)
index:=value| scr expression

Command **setkvp** checks whether index is out of bounds. For setting key and value
command **setkvp** checks whether key already exists in array.
To calculate indexes for access to array elements command setkvp in scr-mode creates
 temporary variables and deletes them at the end (**for non-interactive mode only**):

To get key and value it is possible by commands **getbykey** and **getbyvalue**.

```
>code ppl:
>array(y[3]);
>setkvp(y[0])(+(1)(2));
>setkvp(y[1])(one)(1);
>setkvp(y[2])(two)(2);

>code scr:
>var x = 1;
>array y[3];
>setkvp y[x + 1] = five,2 + 3;
>setkvp y[0] = null, 0;
```

setkvp may be omitted in **scr-mode**:

```
>y[0] = five,2 + 3;
>y[1] = Two,;
>y[1] = Two,"";
```

getvalue (get)

Returns value of single var|const or array element.
Error: when argument is literal or not existed variable.
Format: **getvalue | get (var_name) |**
            getvalue | get(array_name[index]
index:= value|ppl expression

Examples:
```
>array y[3] = 999;
>write#("getvalue(y[0]) = {0}",get(y[0]));
Result: getvalue (y[0]) = 999
```

```
To get data from structure see (sample).
```

getname

Returns name of single var|const | array | array element as string.
Error: when argument is literal or not existed variable.
Format: **getname (name)**

Examples:
1.
```
>var (x[ppl]);
>write("{0} = {1}")(getname(x))(getvalue(x);
x = ppl
```
2.
```
function f(array: arr)
{
   write#("argname={0}  name={1}",getargname(arr), getname(arr));
}
array y;
f(y);
Result: argname=y   name=arr

function f2(storage: stor)
{
   write#("argname={0} name={1}",getargname(stor),getname(stor));
}
storage s[2];
f2(s);
Result: argname=s   name=stor
```

gettoken

Returns token in accordance with its number in string, string contains tokens, separated by **"separator"**. Parts of string arounded by quotes are passed.
As well gettoken may return number of tokens.
If number >= max number of tokens cmd returns **"Exception"**.
Format: **gettoken (string)(separator)(number)**
　　　return: **item_value**
　or

　　　　gettoken (string)(separator)
　　　return: **number of tokens**

Example:
```
1.
>dbg yes;
>gettoken("Hello,World")(",")(0);
result = Hello
>gettoken("Hello,World")(",")(1);
result = World
>gettoken("Hello,World")(",")(2);
Result = Exception


2.
>code scr;
>var x = "\"Hello,World\",  PPL";
>dbg yes
>gettoken(x)(",")(0);
result = Hello,World
>gettoken(x)(",")(1);
result = PPL
3.
>code ppl;
>gettoken("\"Hello,World\",PPL")(",")(1);
result = PPL
4.
>gettoken("Hello,World")(",");
result = 2
```

If it is needed to use math.expression or result of operation call function **GetToken** from **CommonFunctions**:

```
4.
function GetToken(text,separator,index,result)
{
    result = gettoken(text)(separator)(index);
}

var name = "Nissan.Juke";
var count;
count = gettoken(name)(".");
write#("count={0}",count);
var token;

GetToken(name)(".")( -(count) (2) ) (token);   // ppl-format
  write#("token={0}",token);
call GetToken(name,".", count - 2 , token);     // scr-format
  write#("token={0}",token);
call GetToken(name,".",
     gettoken(name)(".") - 20 , token);         // scr-format
  write#("token={0}",token);
```

```
count=2;
token=Nissan
token=Nissan
token=Exception
```

type

Returns type of object (var, const, array, storage or struct).
Format: **type (name) | type name**

```
Example:
>var x;
>write(type(x));
var
```

Nodes and Leaves

createnode

Creates node in path, default path is "**Global**", it is possible to create nodes in Global, Local and Functions Trees. It will be error if name already exists (see recreate).

Format: **createnode(path.name) | createnode path.name**

Examples:

```
> createnode (Node)
> createnode Node.SubNode
>d
-N1      NS
---N1    Global
-----N2 Node
-------N3        SubNode


>createnode Functions.Geo
```

copynode

Copies one or more times node from path with new name, by default path is "Global"
Format:
copynode (src node)(dst node)[number of copies]
default number of copies: 1

Examples:
```
>code scr;
>createnode Person;
>var Person.Name;
>var Person.Family;
>var Person.Gender;
>array Person.cars[3];
>createnode Team;
>copynode(Person)(Team);
>Team.Name = Oscar;
>Team.Family = Ko;
>Team.Gender = m;
>setkvp Team.cars[0] = Juke,Nissan;
>setkvp Team.cars[1] = Qashqai,Nissan;
>d
-N1      NS
---N2    Global
-----N3 Person   [Node]
```

```
-------L0        Name
-------L1        Family
-------L3        Gender
-------N4        cars      [Array 3]
---------L0      #
---------L1      #
---------L2      #
-----N3 Team     [Person]
-------L0        Name      [Oscar]
-------L1        Family    [Ko]
-------L3        Gender    [m]
-------N1        cars      [Array 3]
---------L0      Juke      [Nissan]
---------L1      Qashqai [Nissan]
---------L2      #
---N2    Local
```

getnodes

Creates (or recreates if exists) ppl_array with fullnames of nodes till defined nesting. Processing results of commands getnodes and getleaves allows to find required information in hierarchical data dtructure.

Format:

getnodes (top node)[(nesting)](ppl_array)

Number of required nesting it is possible to get by command displaynode.

If (nesting) do not set node names under top_node will be saved in ppl_array.

For example there is file Data\Mng2.data

```
(Staff
  (Marketing
    (Managers
      (Personal Data1 [base]
        (Name [Benjamin])
        (Salary [6000])
        (Hobby
           (sport [tennis])
           (music [jazz])
        )
      )
    )
    (Clerks
      (Personal Data2 [base]
        (Name [Oliver])
        (Salary [4000])
       ……..
```

Read it:

```
>readdata (data\Mng2.data);
>d
-N2      NS
```

```
      ---N3   Global
      -----N4 Staff
      -------N5       Marketing
      ---------N6     Managers
      -----------N7   Personal Data1  [base]
      ------------L0 Name      [Benjamin]
      ------------L1 Salary  [6000]
      ------------N8 Hobby
      --------------L0        sport   [tennis]
      --------------L1        music   [jazz]
      ---------N6     Clerks
      -----------N7   Personal Data2  [base]
      ------------L0 Name      [Oliver]
      ------------L1 Salary  [4000]
```

or

```
      >dn Staff
      -----Variables  and arrays-----
      -N4     Staff
      ---N5   Marketing
      -----N6 Managers
      -------N7       Personal Data1  [base]
      ---------N8     Hobby
      -----N6 Clerks
      -------N7       Personal Data2  [base]
      -------N7       Personal Data3  [base]
      ---------N8     Hobby
      -------N7       Personal Data4  [base]
      -------N7       Personal Data5  [base]
      ---------N8     Hobby
```

Get fullnames of nodes till nesting 7 and save in ppl_array "persons":

```
>getnodes(Staff)(7)("persons");
>d persons
-----Variables  and arrays-----
-N4      persons [Array 21]
---L0   #       [Staff.Marketing.Managers.Personal Data1]
---L1   #       [Staff.Marketing.Clerks.Personal Data2]
---L2   #       [Staff.Marketing.Clerks.Personal Data3]
---L3   #       [Staff.Marketing.Clerks.Personal Data4]
---L4   #       [Staff.Marketing.Clerks.Personal Data5]
---L5   #       [Staff.Finance.Managers.Personal Data6]
---L6   #       [Staff.Finance.Managers.Personal Data7]
---L7   #       [Staff.Finance.Managers.Personal Data8]
---L8   #       [Staff.Finance.Clerks.Personal Data9]
---L9   #       [Staff.Finance.Clerks.Personal Data10]
---L10  #       [Staff.Finance.Clerks.Personal Data11]
---L11  #       [Staff.Operations management.Managers.Personal
                                              Data12]
---L12  #       [Staff.Operations management.Managers.Personal
                                              Data13]
---L13  #       [Staff.Operations management.Clerks.Personal Data14]
---L14  #       [Staff.Operations management.Clerks.Personal Data15]
```

```
---L15  #         [Staff.Operations management.Clerks.Personal Data16]
---L16  #         [Staff.Operations management.Clerks.Personal Data17]
---L17  #         [Staff.Operations management.Clerks.Personal Data18]
---L18  #         [Staff.Operations management.Clerks.Personal Data19]
---L19  #         [Staff.Human Resource.Managers.Personal Data20]
---L20  #         [Staff.Human Resource.Clerks.Personal Data21]
```

getleaves

Creates ppl_array whose elements have names and values of node. If ppl_array is exists it will be deleted and will be created new one.
Format:
getleaves(node)("ppl_array")

Example:
See previous example with command getnodes

```
>getleaves(Staff.Marketing.Managers.Personal Data1)("property")
>d property
-----Variables  and arrays-----
-N4     property         [Array 2]
---L0   "Name"  [Benjamin]
---L1   "Salary"         [6000]
```

Full code of file Data\mng2.scr to find persons with salary = 2000:

```
var tmp;
var salary;
var name;
readdata  data\Mng2.data;

getnodes(Staff)(7)("persons");    // create ppl_array persons
for(i,0,length(persons))
{
  // delete array "property" if exists
  //and create array "property"
  getleaves(persons[i])( "property");
  for(j,0,length(property))
  {
    tmp = getname(property[j].name);
    if (tmp == "Name")
      name = property[j].value;
    if (tmp == "Salary")
    {
      salary = property[j].value;
      if (salary == 2000)
      {
        write#("{0}",persons[i]);
        write#("\tName = {0,-15}\tSalary = {1}", name, salary);
      }
    }
  }
}
```

```
}
```

```
>rc data\mng2.scr
Result:
Staff.Marketing.Clerks.Personal Data4
        Name = Charlotte        Salary = 2000
Staff.Marketing.Clerks.Personal Data5
        Name = Olivia           Salary = 2000
Staff.Operations management.Clerks.Personal Data18
        Name = Felix            Salary = 2000
Staff.Operations management.Clerks.Personal Data19
        Name = James            Salary = 2000
Staff.Human Resource.Clerks.Personal Data21
        Name = Sophia           Salary = 2000
```

Structures

Preprocessor generates additional ppl-code for commands **definestruct, insertstruct and createstruct** so they used for non-interactive mode only (in scripts).

definestruct

> Creates named block statement, that contains one or several objects - vars, arrays, storages and insertstructs. Struct named block cannot be empty, functions may be located inside **definestruct** or behind.
> Variables and functions with first character **"_"** are private. (see example
> > Examples\struct\TestStruct14.scr).

> Format **scr**:
> **definestruct name**
> **{**
> > **array …;**
> > **var …;**
> > **storage…;**
> > **insertstruct …;**
> > **…**
>
> **}**

> Example:

```
definestruct Room
{
  array computers[3];
  array tables[3];
}
```

Generated code in ppl-mode

```
definestruct
(Room
  (
  (array (computers[3]) )
  (array (tables[3]) )
  )
);
```

insertstruct

Inserts defined structure, previously defined by definestruct. It is possible to insert several instances.
Format **scr**:
insertstruct object_name [[size]] as struct_name

Example:
```
insertstruct rooms[2] as Room;
```
Generated ppl_code:
```
(createnode (rooms) )
(copynode (rooms)(Room)(2))
```

createstruct

Creates one or several instances in accordance with structure type, previously defined by definestruct.
Command createstruct creates per each instance of struct node with name of instance, that contains all objects except of functions.
Additionally for all instances it is created one time node with name of struct, that contains struct functions only, not struct objects.

definestruct Car →

```
struct Car
var
  model, color, year;
functionstruct
  Init(){..}
functionstruct
  Print() {...}
```

createstruct Ford as Car   createstruct Nissan as Car   createstruct BMW as Car

```
Node Ford:
   var mode, color, year
```

```
Node Nissan:
   var mode, color, year
```

```
Node BMW:
   var mode, color, year
```

```
Node Car:
function  Init(){..}
function  Print() {...}
```
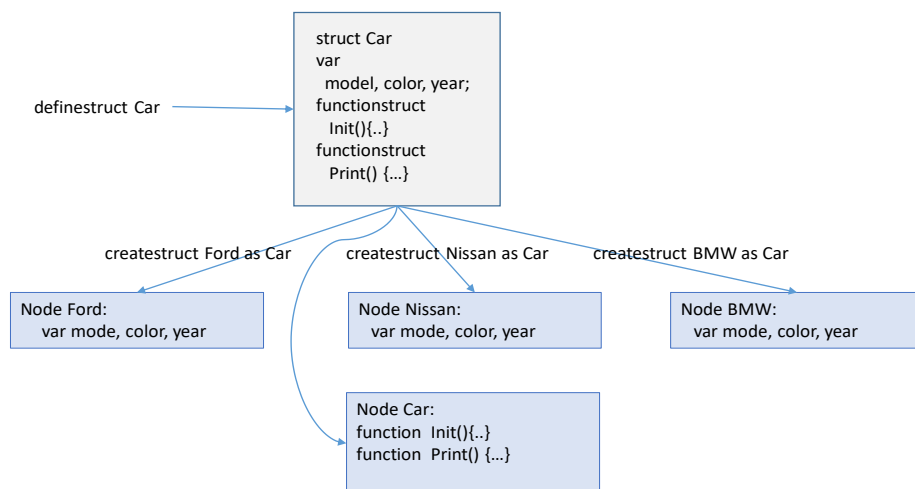
Format scr: **createstruct name[[size]] as struct_name**

Example: create struct array
```
createstruct Group[2] as Person;
```
Generated ppl_code:
```
createstruct (Group)(Person)(2);
```

See examples\struct\teststr4.scr
Tree of above-created object:
```
>displaynodes;
Result:
-N4      Group              [Person]
---N5    0                  [Node]
-----N6 rooms               [Rooms]
-------N7        0
---------N8      computers        [Array 3]
---------N8      tables           [Array 3]
-------N7        1
---------N8      computers        [Array 3]
---------N8      tables           [Array 3]
---N5    1                  [Node]
-----N6 rooms               [Rooms]
-------N7        0
---------N8      computers        [Array 3]
---------N8      tables           [Array 3]
-------N7        1
---------N8      computers        [Array 3]
---------N8      tables           [Array 3]
```

To set data for structure members it is possible by this manner (see examples above):
(see file Examples\Struct\TestStruct4.scr)
```
definestruct Room
{
  array computers[3];
  array tables[3];
}
definestruct Person
{
  var Name;
  var Family;
  insertstruct rooms[2] as Room;
}
createstruct person[2] as Person;
//person.0.Name  = "John";
// index as var
var y = 0;
person.y.Name  = "John";
```

```
person.y.Family = "Deere";
var x = person.y.Name;          // right;
var x1 = person.0.Name;         // right
var x2 = get(person.y.Name);    // right
write#("x = {0}",x);
write#("x1 = {0}",x1);
write#("x2 = {0}",x2);
person.y.rooms.y.computers[0] = "HPE Cray EX";
person.y.rooms.y.computers[0] = "Asus";
person.y.rooms.y.computers[1] = "Sony";
person.y.rooms.y.computers[2] = "HP";
person.y.rooms.y.tables[0] = "IKEA 70x140";
person.y.rooms.y.tables[1] = "IKEA 70x160";
person.y.rooms.y.tables[2] = "IKEA 70x200";
d Person;
d Room;
dd person;
```

instancename

Returns the current structure instance name for using in structure functions.
Format:
instancename() | instancename

Example:
```
>write(instancename);
```

functionstruct

Adds function to structure.
Format is the same as format of function plus structure name.

Format **ppl:**
functionstruct are located behind **definestruct** in this format:
functionstruct
(
   function_name
   (struct name)
   parameter_list
   (function body)
);
function_name | struct_name::= identifier
parameter_list::= parameter [parameter_list]
parameter::= (identifier) | (identifier[default value]) | empty
function body::= (statement1) [(statement2) (statementN)]
identifier::= [var] | [array] | [storage] |
       [struct <struct_name>] | [struct array <struct_name>]:<param_name>

Format **scr:**
functionstruct may be located inside **definestruct** or behind in this format:
For using inside the **definestruct**:
functionstruct function_name
   (parameter_list)
   {
      function body
   }
For using behind the **definestruct** add struct_name:
functionstruct   function_name
   (struct name,parameter_list)
   {
      function body
   }

function_name | struct_name::= identifier
parameter_list::= parameter, [parameter_list]
parameter::=
       identifier | identifier[default value] | identifier = default value | empty
function body::= statement1; [statement2; statement;]
identifier::= [var] | [array] | [storage]:<name>
By default parameter type is var.

See examples:
Examples\struct\TestStruct11.scr (functions of struct are located behind struct)
Examples\struct\TestStruct12.scr (functions of struct are located inside struct)

Any function, belonging to structure by command **funcstruct,** may be replaced behind
by command **function**.
Access to struct functions and any struct objects is provided via **"this"**.

See example in Examples\Struct\TestStruct10.scr:

```
function Print()
{
  write("Global Function");
}

definestruct Script
{
    var x;
    var y;
}
functionstruct Print(Script)
{
  write#("==={0}==={1}===",funcname,instancename);
  write#("this.x = {0} this.y = {1}",this.x,this.y);
}

functionstruct Foo(Script)
{
    write#("==={0}==={1}===",funcname,instancename);
    this.Print();
}

createstruct script as Script;
script.x = 100;
script.y = "PPL";
script.Foo();
script.Print();
Print();

function script.Print()
{
  write#("==={0}==={1}===",funcname,instancename);
  write#("Updated function Print   this.x = {0} this.y =
          {1}",this.x,this.y);
}
script.Print();
```

```
 //Result:
===Script.Foo===script===
===Script.Print===script===
    this.x = 100 this.y = PPL
===Script.Print===script===
    this.x = 100 this.y = PPL
Warning: [FuncCreateFunction] function [Script.Print] is updated
===Script.Print===script===
Updated function Print    this.x = 100 this.y = PPL
Print: Global Function
```

Example of using delegates for functions of structures:

```
definestruct Person
{
    var name;
    var family;
    var age;
    functionstruct Init(n,f,a)
    {
        this.name = n;
        this.family = f;
        this.age = a;
    }
}
function Print2()
{
    write("global function: Print2");
}
definestruct Employee
{
    var position;
    insertstruct person as Person;
    functionstruct Print()
    {
      write( "struct function: Employee.Print");
      write#("    person: name={0} family={1} age={2}",
         this.person.name,this.person.family,this.person.age);
      write#("    position={0}",this.position);
    }
}
createstruct employee as Employee;
employee.position = "Manager";
call employee.person.Init("Johnny","Walker",40);

delegate myDlgt ();
dlgtinstance instance myDlgt;
dlgtset instance Print2;
dlgtcall instance();
```

```
dlgtset instance employee.Print;
dlgtcall instance();
```

dstruct

Displays contents of structure types.
Format **ppl**:
dstruct [(struct_name) [(data)]];
Format **scr**:
dstruct [struct_name [data]];

dd

Displays contents of structure instance (as well as "display") without functions.
Format: **dd instance_name**

Example:
see examples\struct\testCar.scr.

```
definestruct Car
{
  var model;
  var color;
  var year;
}
functionstruct Print(Car)
{
   write#("==={0}===",InstanceName);
   write#("model = {0},   color = {1}, year =
       {2}",this.model,this.color,this.year);
}
functionstruct Init(Car,m,c,y)
{
  this.model = m;
  this.color = c;
  this.year  = y;
}
createstruct Ford as Car;
Ford.Init("Mustang")("Red")("1969");
createstruct Nissan as Car;
Nissan.Init("Qashqai")("White")("2023");
Ford.Print();
Nissan.Print();
//Result:
===Ford===
model = Mustang,   color = Red, year = 1969
===Nissan===
model = Qashqai,   color = White, year = 2023
```

```
>dd
-N2      NS
---N3    Global
-----L0 empty    (const)
-----N4 Ford              [Car]
-------L0        model    ["Mustang"]
-------L1        color    ["Red"]
-------L2        year     ["1969"]
-----N4 Nissan            [Car]
-------L0        model    ["Qashqai"]
-------L1        color    ["White"]
-------L2        year     ["2023"]
```

Arithmetic operators

+, -, *, /, ^, %, ++, --

These are binary operators.

Do not confuse with functions names in file **CommonFunctions.ppl**:

Sum, Sub, Mul, Div, Pow

Examples in ppl prefix notation:
```
+(x)(y)
*(+(x)(y))(-(z)(3))
```
Examples in scr infix notation:
```
>code scr;
> var z = x  + y;
> var z  = (x  + y) * (z - 3);
```

Logical operators

<, <=, >, >=, ==, !=, &&, ||, xor

xor only for ppl mode

These are binary operators.

Do not confuse with functions names in file **CommonFunctions.ppl**:

LT, LE, GT, GE, EQ, NE, AND, OR,XOR

Examples in ppl prefix notation:
```
==(x)(y)
&&(==(x)(y))(==(z)(3))
```
Examples in scr infix notation:
```
x == y
(x == y) && (z == 3)
```

Conditional ?: operator

Condition ? if_true: if_false;

```
set x = 3 < 4 ? 1 : 2;
or
x = 3 < 4 ? 1 : 2;
```

Variables and Arrays

var

Creates a single variable in Global or in Local function scope. It will be error if name
already exists (see recreate).
Format **ppl**:
var (name)| (name[initial value])
name:= [node path]name
node path:= node. | node
initial value:= value| ppl expression
ppl expression:=value | prefix notation expression

Examples:
```
>var (greeting["Hello"]);
>var (x);
>code ppl;
>array(z)(1)(2)(3);
>var (x[z[0]]);          or
>var (x[get(z[0])]);
>var (x[get(y[get([y0])])]);   // error: var (x[y[y0]]);
```

Format **scr**:
var name
var name1,name2,name3…
var name|name = initial value
var name1,name2,name3… = init_value

name:= node path.name
node path:= node. | node
initial value:= value| scr expression
scr expression:= value | infix notation expression

Command **var** in scr-mode creates temporary variables and deletes them at the end
to calculate indexes for access to array elements.

Examples:
```
>code scr;
>createnode N1;
>createnode N1.N2;
>var greeting = "Hello";
>var x;
>var N1.N2.z = 2 + 3;
>var x = z[0];
>var x = get(y[get(y[0])]);
>array arr[] = {1,2,3};
```

```
>var y = arr[0] + arr[1];
>var a,b,c = 0;
>var b = 2 >= 3;      // b = False


>var OneRadian = 180/Math.PI();
>var x = Math.Sin(30/OneRadian);
```
Do not use Math.Function in Math.Function:
```
>var x = Math.Sin(30/(180/Math.PI()))  // error
```

const

Creates a single constant variable in Global or in Local function scope. It will be error if name already exists (see recreate).

Format **ppl**:
const (name[initial value])
name:= [node.]name
initial value:= value| ppl expression
ppl expression:=value | prefix notation expression

Example:
```
>const (x[0])
>const (y[+(2)(3)]);
```

Format **scr**:
const name = initial value
const name1,name2,name3… = initial value

name:= node path.name
node path:= node. | node
initial value:= value| scr expression
scr expression:= value | infix notation expression

Examples:
```
>createnode N1;
>code scr;
>const N1.Greetings = "Hello"
>write(N1.Greetings)
Hello
>const radian = 180 / Math.PI();
>write(radian);
57.2958
```

setconst

Converts var with assigned value to const.
Format **ppl:**
    setconst (var_name)
Format **scr:**
    setconst var_name

Example:
```
>code scr;
>var x = 0;
>setconst x;
```

array

Creates single-dimensional array in Global or in Local function scope. It will be error if array with same name already exists (see recreate).

Format **ppl**:

> array(name [length]) [(initial value)]
> array(name)(1st item)(2nd item)…
> name:= node path.name
> node path:= node. | node
> length:= value| ppl expression
> initial value:= value| ppl expression
> item:= value| ppl expression
> ppl expression:=value | prefix notation expression

Examples:

```
>code ppl;
>var(x[10]);
>array(y[3]);
>array(y[/(x)(2)])(0);        // init by 0 all 5 elements
>array(y[x])(*(x)(3));        // init by 30 all 10 elements
>array(y)(1)(x)(+(1)(2));     // init 3 elements array = 1,10,3
```

Format **scr**:

> array name[length];
> array name [length] = initial value;
> array name [] = {1st item, 2nd item,…};
> name:= node path.name
> node path:= node. | node
> length:= value| scr expression
> initial value:= value| scr expression
> item:= value| scr expression
> scr expression:=value | infix notation expression

Examples:

```
>code scr;
>array y[3];
>array y[1+2] = 0;         // init by 0 all 3 elements
>array y[] = {1,2,1+2};    // init 3 elements array = 1,2,3
>var x = 1;
>array y[x+2];
```

To access an array element you need to calculate index as a separate variable:

```
>code scr;
>var x = 1;
>array y[] = {1,2,3,4,5};
var index = x + 1;
>write(y[index]);          // or write(y[2])
```

```
Index may be used as ndex as expression in command set
>set y[x + 1] = 100;
```

Creation array with **length = 0**:
```
array arr;   or
array arr[]; or
array arr[0];
```
In the following sample array with **length = 0** is created preliminary and reallocated in
 function **Directory.GetDirectories**   in accordance with real length:
```
>createnode Dir;
>array Dir.dir; // or array Dir.dir [0];
>call Directory.GetDirectories(getname(Dir.dir), "c:\\users\\");
```

The following commands in format **scr** are used without parentheses in simple
statement without equal sign.
They are used with parentheses in statements as arguments of other commands or in
statements with equal sign on the right side:

`array.`**push,**

`array.`**shift,**

`array.`**unshift,**

`array.`**remove,**

`array.`**insert,**

`array.`**slice,**

`array.`**concat,**

`array.`**clear.**

Example:
```
 >code scr;
>array y[] = {1,2,3,4,5};
>array.push y 10;                   // simple statement
>write#( array.push (y)(20));    // argument of command write#
7
>writearray y row
1,2,3,4,5,10,20
```

Writing arguments in parentheses implies ppl-mode and prefix notation.

Example:
```
>code scr;
>array y[] = {1,2,3,4,5};
>array.remove (y)(1+1);                 // wrong
>array.remove (y)(+(1)(1));             // right
```

See array commands examples in **examples\ArrayFunc\Samples.scr**.
Additional array service see in file **CommonFunctions.ppl**:
 CsvToArray, ArrayToCsv, ArrayIsExist, ArrayIndexOf, ArrayLastIndexOf, ArrayForEach.

realloc

Changes length of array, all elements are saved in changed array.
Format **ppl**:
    realloc(array_name)(new length)
    realloc(array_name)(new length) (init_value)
Format **scr**:
    realloc array_name[new length]
    realloc array_name[new length] = init_value
    realloc(array_name)(new length) [(init_value)]   (supported format ppl)

Example 1
```
>code scr;
>array y[5] = 0;
>realloc y[10];
>d;
-N1      NS
---N2    Global
-----N3 y          [Array 10]
-------L0          #          [0]
-------L1          #          [0]
-------L2          #          [0]
-------L3          #          [0]
-------L4          #          [0]
-------L5          #
-------L6          #
-------L7          #
-------L8          #
-------L9          #
---N2    Local
>realloc y[3];
>d;
-N1      NS
---N2    Global
-----N3 y          [Array 3]
-------L0          #          [0]
-------L1          #          [0]
-------L2          #          [0]
---N2    Local
```

If init_value is specified this value will be set in all elements of the new array. If init_value is not specified old values are saved in the new array.
Size of the new array can be equal zero, can be smaller or larger than the old one.

It is possible to use realloc for storage on Row level.

Example 2
```
>storage s[3,4,5];
>realloc s.0.0.Row[3];   // or realloc(s.0.0.Row)(3);
>ssetrow s[0,0] = {1,2,3};
>sinit s = 0;

>realloc s.0.1.Row[10];
>ssetrow s[0,1] = {1,2,3,4,5,6,7,8,9,10};

>realloc s.0.2.Row[15];
>ssetrow s[0,2] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
>realloc s.1.1.Row[10];

>ssetrow s[1,1] = {1,2,3,4,5,6,7,8,9,10};

>realloc s.2.1.Row[10];
>ssetrow s[2,1] = {1,2,3,4,5,6,7,8,9,10};
>swrite s;

Result:
      0   1   2   3   4   5   6   7   8   9   10  11  12  13  14

------------------------------------------NS.Global.s.0------------------------------------------
[0]   0   0   0
[1]   1   2   3   4   5   6   7   8   9   10
[2]   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
[3]   0   0   0   0   0
------------------------------------------NS.Global.s.1------------------------------------------
[0]   0   0   0   0   0
[1]   1   2   3   4   5   6   7   8   9   10
[2]   0   0   0   0   0
[3]   0   0   0   0   0
------------------------------------------NS.Global.s.2------------------------------------------
[0]   0   0   0   0   0
[1]   1   2   3   4   5   6   7   8   9   10
[2]   0   0   0   0   0
[3]   0   0   0   0   0
```

Example 3

```
>storage s[2,1];
>realloc s.0.Row[3];
>d s;
-----Variables  and arrays-----
-N4     s                 [Storage 2 *]
---N5   0                 [Array element]
-----N6 Row               [Array 3]
-------L1        #
-------L2        #
-------L0        #
---N5   1                 [Array element]
-----N6 Row               [Array 1]
-------L0        #

>code scr;
>storage s[2,3];
>realloc s.0.Row[0] = 0;

>storage s[2];
>realloc s.Row[3];
>d s;
-----Variables  and arrays-----
-N4     s                 [Storage 1 *]
---N5   Row               [Array 3]
-----L0 #
-----L1 #
-----L2 #
```

array.push

Adds a new item to an array as last, returns a new size of array.
Format **ppl: array.push (array_name)(item_value) |**
        or for key and value array (kvp)
        array.push (array_name)(item_name)(item_value)
Format **scr: array.push (array_name, item_value) |**
        array.push (array_name, item_name, item_value)

It is possible to use variables or array items as item name or item value.

Example:

```
>code scr;
>array y;
>array.push (y,1);
>array.push (y,1 + 2);
>array.push (y,1 + array.min());
>writearray y row;
-----Array y-----
1, 3, 2
>var len = array.push(y)(1 + 2);      // error, it is PPL-forlamt
>var len = array.push(y)(+(1)(2));   // right
or
>array.push(y,1+2);
>var len = length(y);
```

array.pop

Returns the latest item value and removes item**.**
For kvp-array returns item name and item value, separated by comma and removes item.
 If array is empty: returns string "**Empty**".
Format**: array.pop (array_name)**

Examples:
1.
```
>code scr
>array y[] = {1,2,3};
>var result = "";
>set result = array.pop(y);    // result = 3
>writearray y row;
Result:
-----Array y-----
1, 2
```
2.
```
>code scr
>array y[3];
>setkvp y[0] = one,1;
>setkvp y[1] = two,2;
>setkvp y[2] = three,3;
>var result = "";
>set result = array.pop(y);    // return three,3
>write#("name={0}value={1}",
    gettoken(result)(",")(0),gettoken(result)(",")(1));
name=three value=3
>writearray y row;
Result:
-----Array y-----
1, 2
```

array.reverse

Reverses items order in array, returns a size of array;
Format **ppl: array.reverse (array_name)**
Format **scr: array.reverse array_name | array.reverse (array_name)**

array.shift

Removes the first item of the array, returns a new size of array.
Format **ppl: array.shift (array_name)**
Format **scr: array.shift array_name | array.shift (array_name)**

Example:
```
>code scr;
>array y[] = {1,2,3,4,5};
>array.shift y;
>writearray y row
-----Array y-----
2,3,4,5
> write#(array.shift(y))
2
>writearray  y row
-----Array y-----
3, 4
```


array.remove

Removes item by index, returns a new size of array.
Format **ppl: array.remove (array_name)(index)**
Format **scr: array.remove array_name index |**
       array.remove (array_name, index) |
       array.remove (array_name) (index)

Example 1:
```
>array y[] = {1,2,3,4,5};
>write#("y.length = {0}",length(y))
y.length = 5
>write#("y.length = {0}",array.remove(y,1));
y.length = 4
>writearray y row
-----Array y-----
1, 3, 4, 5
```

It is possible to use variables or array items as array_name or index.
Example 2:

```
>array y[] = {1,2,3,4,5};
>var i = 0;
>var x = "y";
>array.remove(get(x),i);
>writearray y row;
-----Array y-----
2, 3, 4, 5
```

array.clear

Removes all items from array, returns 0.
Format **ppl: array.clear (array_name)**
Format **scr: array.clear array_name | array.clear (array_name)**

array.unshift

Adds a new item as first to an array, returns a new size of array.
Format **ppl: array.unshift (array_name)(item_value) |**
         array.unshift (array_name)(item_name)(item_value)
Format **scr: array.unshift (array_name, item_value) |**
         array.unshift (array_name, item_name, item_value)

It is possible to use variables or array items as index, item name or item value.

array.insert

Inserts item before item with index, returns a new size of array.
Format **ppl:**
     array.insert (array_name)(index)(item_value) |
     array.insert (array_name)(index)(item_name)(item_value)
Format **scr:**
     array.insert (array_name,index, item_value) |
     array.insert (array_name,index, item_name, item_value)

It is possible to use variables or array items as index, item name or item value:

```
>array y[] = {1,2,3,4,5}
>array r[] = {1,3}
>array.insert(y, r[1], r[0])
>writearray y row
-----Array y-----
1, 2, 3, 1, 4, 5
```

array.slice

Forms a slice [of specified length] out of the current array segment starting at the specified index and return new array length or **"Error"**.

Format **ppl**:

      array.slice (array_name)(index)| array.slice(array_name)(index)(length)

Format **scr**:

      array.slice (array_name, index)|
      array.slice (array_name, index, length)

It is possible to use variables or array items as index or length.

index starts from 0

Example:

```
>code scr;
>array y[] = {1,2,3,4,5};
>writearray y row;
-----Array y-----
1, 2, 3, 4, 5
>array.slice(y, 2);
>writearray y row;
-----Array y-----
3, 4, 5
>array.slice(y, 1, 2);
>writearray y row;
-----Array y-----
4, 5
```

array.sum

Returns the sum of all items in array. In case of error returns string **"Error"**.
Format: **array.sum (array_name)**

Example:
```
>code scr;
>dbg yes;
>array y[] = {1,2,s,4,5,a};
Info: [CreateArrayFormat1] Global array [y] is created
>array.sum(y);
Error: [FuncArraySum] not digital data array y[2] = s
Error: [FuncArraySum] not digital data array y[5] = a
result = Error;
>recreate yes;
>array y[] = {1,2,3,4,5};
Info: [CreateArrayFormat1] Global array [y] is created
>array.sum(y);
result = 15
```

array.copy

Copies src array to dst array. In case of error returns string **"Error"**.
Format ppl : **array.copy (src array)(dst array)**
Format scr : **array.copy src_array dst_array |**
             array.copy (src_array, dst_array) |
             array.copy (src_array) (dst_array)

Example:

```
>code scr;
>array src[] = {1,2,3,4,5};
>array dst;    // dst array is created before array.copy
>array.copy src dst;
          // or
          // array.copy (src, dst);
          // array.copy (src) (dst);
>dst[0] = 10;
>writearray dst row;
-----Array dst-----
10, 2, 3, 4, 5
>writearray src row;
-----Array src-----
1, 2, 3, 4, 5
```

array.min

Returns the minimum value in array. In case of error returns string **"Error"**.
Format: **array.min (array name)**
Example:

```
>code scr;
>dbg yes;
>array y[] = {10, 20, 5, 30};
>array.min(y);
Result = 5
```

array.max

Returns the maximum value in array. In case of error returns string **"Error"**.
Format: **array.max (array name)**

Example:

```
>code scr;
>dbg yes;
>array y[] = {10, 20, 5, 30};
>array.max(y);
Result = 30
```

array.average

Returns the mean value in array. In case of error returns string **"Error"**.
Format: **array. average (array name)**
Example:

```
>code scr;
>dbg yes;
>array y[] = {10, 20, 30, 40, 50};
>array.average(y);
Result = 30
```

array.first

Returns value of the first item in array. In case of error returns string **"Error"**.
Format: **array.first (array name)**
Example:

```
>code scr;
>dbg yes;
>array y[] = {100, 200, 300};
>array.first(y);
Result = 100
```

array.last

Returns value of the last item in array. In case of error returns string **"Error"**.
Format: **array.last (array name)**
Example:

```
>code scr;
>dbg yes;
>array y[] = {100, 200, 300};
>array.last(y);
Result = 300
```

array.concat

Concatenates several arrays to destination array, returns length of destination array
or **"Error"** if one of arrays is absent.
Format **ppl**: **array.concat (array1)(array2)(...arrayN) (dst array)**
Format **scr**: **array.concat array1 array2 ...arrayN   dst_array |**
        array.concat (array1, array2, ...arrayN, dst_array) |
        array.concat (array1)(array2)(...arrayN) (dst array)

Example:
1.

```
>code scr;
>array y[] = {1,2,3,4,5};
```

```
>array a[] = {10,20,30};
>array b[] = {100,200};
>array dst;
>array.concat y a b dst;    // simple statement
 //   or
 //>array.concat (y, a, b, dst);
 //>array.concat (y) (a) (b) (dst);

>writearray y row;
-----Array y-----
1, 2, 3, 4, 5
>writearray dst row;
-----Array dst-----
1, 2, 3, 4, 5, 10, 20, 30, 100, 200
```

2.
```
>code scr;
>array y[] = {1,2,3};
>array b[] = {4,5,6};
>array dst;
>var len = array.concat(y)(b)(dst);  // right side of statement
                                     // with equal sign

>writearray dst row;
-----Array dst-----
1, 2, 3, 4, 5, 6
>write#("length={0}",len);
length = 6
```

array.sort

Sort ppl_array in accordance with ascend or descend order.
Format **ppl**: **array.sort (array_name)(ascend | descend)**
Format **scr**: **array.sort (array_name, ascend | descend) |**
          array.sort (array_name)(ascend | descend)

Example:
```
>code scr;
>array y[] = {1,30,2};
>array.sort (y,ascend);
>writearray y row;
-----Array y-----
1, 2, 30
```

See array samples in examples\arrayfunc.

Storage

Service of multi-dimensional arrays is realized by storage operators in mode ppl (parameters with prefix expressions in parentheses, but may be used also in mode scr (see Examples 3).

storage

Creates single variable, single-dimensional or multi-dimensional array with dimension from 1 to N in Global or in Local function scope. It will be error if name already exists (see recreate).
Storage contains several levels of arrays,name of the topmost level is name of storage,name of the bottommost arrays in each level is **Row**. Names of intermediate levels are array index in level. To set different length arrays on Row level use realloc.
 (see examples\scr\testswrite.scr).
Size of the lowest level may be 0.
Format **ppl**:
storage (name) | storage (name) (length dim1) |
storage (name) (length dim1) (length dim2)…
name:= node path.name
node path:= node. | node
length:= value| ppl expression
ppl expression:=value | prefix notation expression

Format **scr**:
storage name | storage name [length dim1, length dim2 …] [= value]

Examples:
```
1.
>code scr;
>storage x;     // or in format ppl: storage (x)
>d;
-N1      NS
---N1    Global
-----L0 x
2.
>code scr;
>storage x[2]; // or    in format ppl:
            //   storage (x)(2); – single-dimensional array
>d;
-N1      NS
---N1    Global
-----N3 x        [Storage 1 2]
-------N4        Row      [Array 2]
---------L0      #
---------L1      #
[Storage 1 2] – dimension length
```

```
3.
>code ppl
// size calculation for ppl only
>storage (x)(+(2)(3)) – single -dimensional array [Storage 1 5]
or same result:

>code scr;
>var y = 2 + 3;
>storage x[y]; // single-dimensional array, length = 5
4.
>storage x[2,3]  - two-dimensional array
>d
Result:
-N1      NS
---N2    Global
-----N3 x         [Storage 2 2x3]
-------N4         0        [Array element]
---------N5       Row      [Array 3]
-----------L0     #
-----------L1     #
-----------L2     #
-------N4         1        [Array element]
---------N5       Row      [Array 3]
-----------L0     #
-----------L1     #
-----------L2     #
[Storage 2 2x3] - dimension length x length
```

```
5.
>storage x[3,4,5,100]; – four-dimensional array
6.
>storage x[0,2];  //error length=0 may by for the lowest
                        Level only
>storage x [2,0]; // right
```

```
7.
>storage s[2] = 0;
>d s;
-----Variables  and arrays-----
-N4      s                [Storage 1 2]
---N5    Row              [Array 2]
-----L0 #        [0]
-----L1 #        [0]
```

sinit

Init storage
Format **ppl**:
sinit (name)(initial value)
Format **scr**:
sinit name = initial value

initial value:= value| ppl expression

Examples:
```
>code scr;
>storage x[2,3];
>sinit x = 0;
>d;
Result:
-N1      NS
---N2    Global
-----N3 x          [Storage 2 2x3]
-------N4          0        [Array element]
---------N5        Row      [Array 3]
-----------L0      #        [0]
-----------L1      #        [0]
-----------L2      #        [0]
-------N4          1        [Array element]
---------N5        Row      [Array 3]
-----------L0      #        [0]
-----------L1      #        [0]
-----------L2      #        [0]
>writearray x.0.Row row;
Result:
0, 0, 0
```

sget

Gets value of element in storage
Format:
sget (name)(index1)(index2)...

Examples:
```
>sget(stor);       // get value of single-variable
>sget(stor)(0);  // get value of single-dimensional array,index=0

// get value of two-dimensional array,stor[0][0]
>storage stor[2,2];
>sget(stor)(0)(0);   // or stor.0.Row[0] for using on the right
                     // side of the expression
>var x = sget(stor)(0)(0);    // error
>var x = stor.0.Row[0];       // right
```

sset

Sets value for element in storage
Format **ppl**: **sset (name)(index1)(index2)(value)**
Format **scr**: **sset name [index1,index2] = value**

Examples:
```
>code scr;
>storage stor;
>sset stor = 0; // set value of single-variable = 0
                // or: set stor = 0;

// set value of single-dimensional array,stor[0]= 1
>storage stor[3];
>sset stor[0] = 1;  // or: set stor.Row[0] = 1;

// set value of two-dimensional array,stor[0][0] = 1
>storage stor[2,3];
>sset stor[0,0] = 1; // or: set stor.0.Row[0] = 1;
> var y = 1;
>sset stor[y,y] = 2; // it is possible to use var as index
>swrite stor 30
result:
        0         1         2

---------NS.Global.stor---------
[0]      1
[1]               2
```

swrite

Displays elements values of storage
Format **ppl**:
swrite(name)
swrite(name) (max_window_width)
Format **scr**:
swrite name
swrite name max_window_width

default max_window_width = 100
Limit for FOREST.exe max_window_width = `Console.WindowWidth = 120`

Examples:

```
>code scr;
>storage x[3];
>sinit x = 0;
>swrite x 50;
        0                   1                   2
-------------------NS.Global--------------------
[x]     0                   0                   0


>code scr;
storage s[5,3];
>sinit s = 0;
>swrite s;
        0                               1                               2
---------------------------------------------NS.Global.s--------------
[0]     0                               0                               0
[1]     0                               0                               0
[2]     0                               0                               0
[3]     0                               0                               0
[4]     0                               0                               0


>swrite s 30;
        0           1           2
---------NS.Global.s----------
[0]     0           0           0
[1]     0           0           0
[2]     0           0           0
[3]     0           0           0
[4]     0           0           0
```

```
>code scr;
>storage xxx[3,3,5];
>sinit xxx = 0;
>swrite xxx 40;
      0    1    2    3    4

------------NS.Global.xxx.0------------
[0]   0    0    0    0    0
[1]   0    0    0    0    0
[2]   0    0    0    0    0
------------NS.Global.xxx.1------------
[0]   0    0    0    0    0
[1]   0    0    0    0    0
[2]   0    0    0    0    0
------------NS.Global.xxx.2------------
[0]   0    0    0    0    0
[1]   0    0    0    0    0
[2]   0    0    0    0    0
```

sinfo

Displays length of each dimension in storage
Format: **sinfo(name)**

Examples:
```
>storage y[5];
>sinfo(y);
```
Result: Storage 1 5   // single-dimensional array length 5
After using realloc for storage Row it will be changed:
```
>code scr;
>storage s[3,5];
>realloc s.0.Row[8];
>sinfo(s);
```
Result: Storage 2 *
```
>d;
-N2    NS
---N3   Global
-----L0      empty              (const)
-----N4 s                    [Storage 2 *]
-------N5      0                  [Array element]
---------N6    Row                   [Array 8]
-----------L0     #              [0]
-----------L1     #              [0]
-----------L2     #              [0]
-----------L3     #              [0]
-----------L4     #              [0]
-----------L5     #
-----------L6     #
-----------L7     #
-------N5      1                  [Array element]
---------N6    Row                   [Array 5]
-----------L0     #              [0]
-----------L1     #              [0]
-----------L2     #              [0]
-----------L3     #              [0]
-----------L4     #              [0]
-------N5      2                  [Array element]
---------N6    Row                   [Array 5]
-----------L0     #              [0]
-----------L1     #              [0]
-----------L2     #              [0]
-----------L3     #              [0]
-----------L4     #              [0]
```

ssetrow

Sets value for elements of the lowest level.
Format **ppl**:
ssetrow(name)(ind1)(ind2)(indN)... (elem1)(elem2)(elemM)...
Format **scr**:
ssetrow name [ind1,ind2,indN...] = {elem1,elem2,elemM...}

Examples:
```
// N = 2, M=3
>code scr;
>storage y[2,3];
>ssetrow y[0]={1,2,3};
>ssetrow y[1]={4,5,6};
>swrite y 50;
        0               1               2

        -----------------NS.Global.y-------------------
[0]     1               2               3
[1]     4               5               6
```

Backup & Recovery

savedata | (sd)

Saves data from node to file with extension **.data**.
If node is root, all root contents will be saved.
Format **ppl**: **savedata | sd (filename.data) [(node)]**
Format **scr**: **savedata | sd filename.data [node]**

Default node: **NS.Global**

Example:
```
>code ppl;
>savedata(Data\Colors1.data)(Colors);
```

readdata (rd)

Reads data from file file with extension **.data** to Nodes Configuration and
NS.Global, **not to Local.**
Format **ppl**: **readdata | rd (filename.data)[(node)]**
        readdata | rd filename.data

Format **scr**: **readdata | rd filename.data [node]**
Default node: **NS.Global**

Examples:
```
>code scr
>readdata Data\Colors.data;
>d;
-N  NS
---N  Global
-----N  Colors
-------L0       Black           [0]
-------L1       Blue            [9]
-------L2       Cyan            [11]
-------L3       DarkBlue        [1]
-------L4       DarkCyan        [3]
-------L5       DarkGray        [8]
-------L6       DarkGreen       [2]
-------L7       DarkMagenta     [5]
-------L8       DarkRed         [4]
-------L9       DarkYellow      [6]
-------L10      Gray            [7]
-------L11      Green           [10]
-------L12      Magenta         [13]
-------L13      Red             [12]
-------L14      White           [15]
-------L15      Yellow          [14]
```
Each item in this array has key (Black, Blue,…) and value(0,9,…), use
getbykey and **getbyvalue**.

Control Flow

if, else

The meaning of the block "if-else" does not differ from the generally accepted. See Base Concepts about using statements terminator ";" in if.

Format **ppl**:
```
if(condition)
(
   (statement)
   (statement)
   [(else
       (
         (statement)
          (statement)
       )
    )]
 );
```
Here expression in prefix notation.
Statement in ppl mode.

Format **scr**:
```
if (condition)
{
    statement;
    statement;
  [else
   {
    statement;
    statement;
   }]
}
```
Here expression in infix notation.
Statement in ppl or scr mode.

Example ppl mode:

```
var(x[1]);
var(y[1]);
if (==(x)(y))
(
    (write(TRUE))
  (else
    (
        (write(FALSE))
    )
  )
);
TRUE
```

Examples in scr mode:

```
var x = 1;
var y = 1;
if ( x == y )
{
  write(TRUE);
}
else
{
  write(FALSE);
}
```

```
var x;
if(x == empty)
  write("x = empty");
```

If block in "if" or in "else" contains only one statement it is possible to omit { }:

```
code scr;
var x = 0;
if(x == 1)
  write(TRUE);
else
  write(FALSE);
Rsult: FALSE
```

switch, case, default

switch statement – for select one from several case blocks to be   executed.
About using statements terminator ";" in switch see Base Concepts.

Format **ppl**:
```
switch(expression)
(
    (case1) [(case2)…]
    (
       (statement)
       (statement)
         …
    )
    (caseN) …
     (
       (statement)
       (statement)
         …
     )
      …
     [ (default)
        (
            (statement)
            (statement)
      …
        )
      ]
 );
```
Here expression in prefix notation.
Statements in ppl mode.

Format **scr**:
```
switch(expression)
{
   case <value>:
   case <value>:
      statement;
      statement;
       ...
  break;
  case <value>:
      statement;
      statement;
        ...
   break;
  [
  default:
     statement;
     statement;
```

```
    ...
    break;
  ]
}
```

Here expression in infix notation.
Statement in r scr mode.

Examples:
Format **ppl**:
1.

```
switch(x)
(
   (1)(3)
   (
      (write("Cases 1 & 3"))
   )
);
```

2.

```
var (x[1]);
switch (x)
   (
   (1)(3)
   (
      (write("Case 1 & 3"))
   )
   (2)
   (
      (write("Case 2"))
   )
   (default)
   (
      (write("Default"))
   )
  );
Result:
   Case 1 & 3
```

3. Format scr

```
var x = 2;
switch(x)
{
   case 1: case 3:
      write("Case 1 & 3");
   break;
   case 2:
      write("Case 2");
   break;
   default:
      write("Default");
   break;
}
```

loop,do

Iteration block for ppl mode only.
About using statements terminator ";" in loop see Base Concepts.

Format **ppl**:
```
loop (iteration var)(begin)(end)[(step)]   or
 loop () //infinity loop
 (do
   (
     (statement)
     (statement)
       …
   )
 )
begin:= value|ppl expression
end:= value|ppl expression
step:= value|ppl expression
```
By default step = 1. Step may positive or negative.
Parameter end is set before entry to iteration block and may be changed by
setloopend inside iteration block.
Statement in ppl mode.

Examples:
```
loop (i)(0)(10)(1)       // or loop (i)(10)(0)(-1)
(do
   (
     (write("i = {0}")(i))
   )
);
```

See infinity example – greatest common factor (gcf) calculation in for

setloopend

Changes "end" in loop (iteration var)(begin)(**end**)[(step)] inside iteration block
Format **ppl**:
```
setloopend(end_value)
```

Format **scr**:
```
setloopend end_value
```

Example:
1.
```
array arr[10] = 0;
for(j,0,length(arr))
{
   write#("j = {0}    length = {1}",j,length(arr));
   array.remove(arr)(0);
   setloopend length(arr);
}
writearray arr row;
j = 0    length = 10
j = 1    length = 9
j = 2    length = 8
j = 3    length = 7
j = 4    length = 6
-----Array arr-----
0, 0, 0, 0, 0
```

```
2. (see examples\lib\IsItemExist.scr)

// delete all repeated items from array
array arr[] = {a,1,2,a,2,3,4,4};
writearray arr row;
for(i,0,length(arr))
{
   for(j,i + 1,length(arr) )
   {
      if(arr[i] == arr[j])
      {
         array.remove (arr,j);
         setloopend length(arr);
      }
      setloopend length(arr);
   }
}
writearray arr row;
-----Array arr-----
a, 1, 2, a, 2, 3, 4, 4
-----Array arr-----
a, 1, 2, 3, 4
```

for

Iteration block for **scr mode** only.
About using statements terminator ";" in **"for"** see Base Concepts.

Format **scr**:
```
for(iteration var, begin, end [, step])  or
for()  // infinity for
{
    statement;
    statement;
    …
}
```
By default **step = 1**. Step may positive or negative.

Example:
```
var x;
for(i, 0, 10)
{
    x = i * 2;
    write#("x = {0}",x);
}
```

If block in "for" contains only one statement it is possible to omit { }:
```
for(i, 0, 10, 1)
  write#("i={0}",i);
```

Example with **infinity for**:

```
function gcd(x,y,z)
{
   if(isinteger (x) == "False")
   {
     write#("not integer value x={0}",x);
     return;
   }
   if(isinteger (y) == "False")
   {
     write#("not integer value y={0}",y);
     return;
   }
   for()    // infinity for
   {
     if (x > y)
     {
        x = x - y;        // or Sub(x)(x)(y);
     }
     if (x < y)
     {
        y = y - x;        // or Sub(y)(y)(x);
     }
     if (x == y)
     {
         z = x;
         return;
     }
   }
}
var z = 0;
var x = 14144;
var y = 26163;
call gcd(x,y,z);
write#(gcd = {0},z);
Result: gcd = 17;
```

break

Exit from loop (ppl mode) or from for (scr mode) or end of case in switch block.

Example:
```
code scr;
for(i, 0, 4, 1)
{
   if (i == 2)
   {
     write#("i = {0} break",i);
     break;
   }
```

```
    }
```

continue

Continue executing in loop (ppl mode) or in for (scr mode).

```
Example 1:
code ppl;
loop(x)(0)(5)(1)
  (do
    (
      (if (==(x)(3))
        (
          (write("x = {0} continue") (x) )
          (continue)
        )
      (write("x={0}") (x) )
    )
);

Example 2:
code scr;
for(x,0,5,1)
{
   if(x == 3)
   {
      write("x = {0} continue;",x);
      continue;
   }
   write#("x = {0}",x);
}
```

Input and Output

write

Writes the string value to the standard output stream.
String interpolation ($**"x"**) is not supported. If string value contains **"Error:"** it
will be written in red color.
Format:
write(value) | write(c# format)(value)(value)...
value:=value | ppl expression with prefix notation

Example:
```
>var  x = 0;
>write(x);
>write("{0}{1}")("x=")(x); // like c# write("{0}{1}")("x=",x);
//quote in string
>write("ppl\"language");  // ppl"language
//tab in string
>write("ppl\tlanguage");  // ppl language
//newline in string
>write("ppl\r\nlanguage");  // ppl
                              //language
>write(12col);
Result: 12col
>write("Error: wrong name {0}")(12col);
Result: Error: wrong name 12col
```

This operator is used in ppl and scr mode.
```
>code scr;
>var x = 2*5;        // scr expression is calculated in var
>write("{0} {1}")( "x = " )(x);
>code ppl
>write("{0}")(*(2)(5));    // ppl expression
```

write#, writeline

Like as write, for mode scr only, each argument is not enclosed in parentheses.
Format **scr**:
write#(arg) | write#("c# format", arg1 , arg2,...)
writeline(arg) | writeline("c# format", arg1 , arg2,...)

arg:= <literal> | <scr expression with infix notation>

Examples:

```
>code scr;
>write#("{0} {1}", aaaa,1+3)
Result: aaaa 4
>write#(1+2);
Result: 3
>write(1+2);    // not same result as with write#
Result: 1+2
>write(+(1)(2));
Result: 3
>write#("1+2");
Result: 1+2
>write#("{0,-10}{1}\t{2}","Forest","Hello","World")
Result: Forest     Hello World
```

writearray

Writes array contents to the standard output stream. By default writearray writes array elements into the column. Writearray writes array elements into the row by second argument **"row"**.

Format **ppl**:

writearray ([node.]array_name) [(row)]

Format **scr**:

writearray [node.]array_name [row]

or

writearray ([node.]array_name [,row])

Examples:

1.

```
>code scr;
>array y[] = {1,2,3};
>writearray y;
-----Array y-----
[0]      1
[1]      2
[2]      3
>writearray y row
Result:
-----Array y-----
1, 2, 3
```

2.

```
>code scr;
>storage(x)(2)(3);
>sinit(x)(0);
// write the bottommost arrays in storage
>writearray (x.0.Row);
Result:
[0]      #        0
[1]      #        0
[2]      #        0
>writearray (x.1.Row, row);
Result:
0, 0, 0
```

3.

```
>code scr;
>array y[3];
>setkvp y[0] = one,1;
>setkvp y[1] = two,2;
>setkvp y[2] = three,3;
>writearray y row;
Result:
-----Array y-----
{one,1}, {two,2}, {three,3}
```

readline

Reads the next line of characters from the standard input stream. Result will be passed to calling operator.
Format: **readline()**

Examples:
```
>var x;
>x = readline());
>Enter:
>>Hello
>d;
Result:
-N1     NS
---N1   Global
-----L0 x       [Hello]
```

Functions

Function library **CommonFunctions.ppl**, defined in file **Configuration.data** as **default_loaded_functions**, loads automatically and reloads when command **init** executes. It includes 2 types of functions:
-Mathematical and Logical functions,
-Array services and other functions.

Mathematical and Logical functions:

> **Sum (result)(n1)[(n2)]**
> **Sub (result)(n1)[(n2)]**
> **Mult (result)(n1)[(n2)]**
> **Div (result)(n1)[(n2)]**
> **Pow (result, n1, n2)**
> **PlusPlus (result) | Plus1 (result)**   **// like c#: ++(var)**
> **MinusMinus (result) | Minus1 (result)**   **// like c#: --(var)**
> **LT (result, n1, n2)**
> **LTEQ (result, n1, n2)**
> **GT (result, n1, n2)**
> **GTEQ (result, n1, n2)**
> **EQ (result, n1, n2)**
> **NOTEQ (result, n1, n2)**
> **AND (result, n1, n2)**
> **OR (result, n1, n2)**
> **XOR (result, n1, n2)**

These functions replace using prefix notations. Result are returned in 1st parameter and does not passed to the next command:

```
>var (z[0]);
>var(x[1]);
>set(z)(Sum(x)(1));     // error
>Sum(z)(x)(1));         // right
```

Examples:

```
1.
>var(x[1]);
>var(z);
>Sum(z)(x)(1);  // set z = x + 1
>Sum(x)(2);      // set x = x + 2
>d
-N2      NS
---N3    Global
-----L4 x        [3]
-----L5 z        [2]
```

```
     2. This sample returns wrong result:
     >var(x[5]);
     >Sum(x)(x)(2);
     >d x
     -----Variables  and arrays-----
     -L4      x         [5]
     >Sum(x)(2);    // right, x = x + 2
```

```
     3.
     >code scr;
     >call Sum(x,2,3);
     >call Sum(x,2+3);
     >Sum(x)(+(2)(3));
```

User may create own functions file, like CommonFunctions.ppl, and set it in file **Configuration.data** as **UserFunctionsN** or load it:

```
>rc user_functions.ppl|scr;
```

Files, defined in **Configuration**, load their function only and do not execute any command, commands in files, loaded by command rc (readcode) are executed one after another.

Array services and other functions:

CsvToArray (var:str, array:arr) – to fill array from string data separated by comma
ArrayToCsv (array:arr, var:str) – copy to string separated by comma data from array
ArrayIsExist (result, array:arr, value) – return True/False if devined value exists in array
ArrayIndexOf (result, array:arr, value) – return the index of the first occurrence within array or -1.
ArrayLastIndexOf (result, array:arr, value) – return the index of the last occurrence within array or -1.

Example:
```
array my_array;
call CsvToArray("1,2,3,,1,5",my_array);
writearray my_array row;

var result;
call ArrayToCsv(my_array,result);
write#("result = {0}",result);

call ArrayIndexOf   (result, my_array, 3);
write#("index = {0}",result);

call ArrayLastIndexOf   (result, my_array, 1);
write#("index = {0}",result);

Result:
1, 2, 3, , 1, 5
result = 1,2,3,,1,5
index = 2
index = 4
```
See **examples\ArrayFunc\SamplesFunc.scr**.

ArrayForEach(array:arr, array:callback_name) – calls a callback function once for each array element

Example: (examples\arrayfunc\foreach.scr)

```
function sumfunc(array:arr,var:i)
{
    set result = result + arr[i];
}

var result = 0;
delegate d2 (array:arr,var i);
dlgtinstance instance d2;
dlgtset instance sumfunc;
call ArrayForEach({1,2,3,4,5},instance);
write#("result = {0}",result);
//result = 15
```

WindowSize()
width=120 height=30

function

Functions must be declared before being called.
Functions are saved in Tree **Functions** or in Tree **Global** for later call.
Functions return result via parameters, like a classic procedure, and via operator "return" (see Examples\ArrayFunc\mean.scr).
When function is called **by name** (in mode **ppl** or **scr**) each argument must be enclosed in parentheses.
When function is called by command **'call'** (in mode **scr** only) each argument not must be enclosed in parentheses, but separated by comma.
Function uses data created inside, passed from calling function and data from NS.Global.
Data created in function are deleted when function will be finished.

Limitations:
1. For passing array member use 2 arguments:
   array name,
   index array member or temporary variable:

```
>array y[] = {1,2,3};
>var x = 10;
>var tmp;
>tmp = y[0];
>call Sum(x,tmp);
>write#("x={0}",x);
x=11
```

2. If argument value will be changed in function this argument can be used only one time when function is called(see example **Sum_wrong_result**).

3. Do not create function in function:
   function f()   //error
   {
      function s();
   }

Format **ppl**:

Statement terminator ';' does not follow after statements within function, but each statement is surrounded by parentheses.
function
(
  name
  parameter_list
  (function body)
);
name::= identifier
parameter_list::= parameter [parameter_list]
parameter::= (identifier) | (identifier[default value]) | empty
function body::= (statement1) [(statement2) (statementN)]
identifier::= [var] | [array] | [storage] |
    [struct <struct_name>] | [struct array <struct_name>]:<param_name>

Format **scr**:

Statement terminator '**;**' always follows after each type of statements within function.
function   name
        (parameter_list)
     {
          function body
     }
name::= identifier
parameter_list::= parameter, [parameter_list]
parameter::=
      identifier | identifier[default value] | identifier = default value | empty
function body::= statement1; [statement2;   statement;]
identifier::= [var] | [array] | [storage]:<name>
By default parameter type is var.

Examples:

```
>code scr
1.function without parameters
Function func ()
{
   write#("func");
}

2. function with default parameter
function func (n,m[10])    //= func (var:n,var:m[10])   or
                                  //function func (n,m = 10)

{
   write#(funcname());
}

3.
function func (array: n)
{
  write#( funcname());
}
```

```
4.
code ppl;
function
(
  test2(n)
  (
     (write(n))
  )
);
function
(
 test()
 (
    (loop (i)(0)(5)(1)
     (do
      (
        (test2(i))
      )
     )
    )
 )
);
test();      // function call
```

```
5.
 code scr;    // (see   examples\scr\func.scr)
 function sum_arr(array:n,array:m)
 {
   for(i,0,length(n),1)
     write#("[{0}]   [{1}]" ,i, n[i] + m[i]);
 }
 d Functions.sum_arr;
 array x[] = {1,2,3,4,5};
 array y[] = {6,7,8,9,10};
 sum_arr(x)(y);   // function call
 // command "call" allows use arguments by this manner:
 // call sum_arr({1,2,3,4,5},{6,7,8,9,10});
 result:
 [0]   [7]
 [1]   [9]
 [2]   [11]
 [3]   [13]
 [4]   [15]
```

In the following example (see examples\scr\func4.scr) parameter index default
value = 0 and this parameter may be omitted when the function is called.
Array and member array index are passed as 2 arguments.

```
6.
code scr;
 function func (array:arr, var:index[0])
 {
   write#("{0}[{1}] = {2}",getargname(arr),index, arr[index]);
 }
 array y[] = {1,2,3,4};
 func(y);
 call func(y);     // same as previous line
 call func(y,0);
 call func(y,1);
 call func(y,2);
 =========result==========
 y[0] = 1
 y[0] = 1
 y[0] = 1
 y[1] = 2
 y[2] = 3
```

```
7.  Call function from node saved in Tree Global
code ppl;
createnode New;
function
(
     New.func(name)    // public
     (
       (write(name))
     )
);
function
(
     New._hfunc(name)  // private
     (
       (write(name))
     )
)
```

```
8. Call function from node created in Global
code scr;
createnode N;
function N.f()
{
  write("Global.N.f - function");
};
N.f();
dn;
Global.N.f - function
-N2     NS
---N3   Global
-----N4 N                [Node]
-------N5       f                [function]
---------N6     #                [internal_block]
----------N7   write

 // Call function from node created in Functions
createnode Functions.N;
function Functions.N.f()
{
  write("Functions.N.f - function");
};
Functions.N.f();
 Functions.N.f - function
```

```
9. Recursion
var tmp = 0;
function  rec(x)
{
    set tmp = tmp + 1;
    write#("tmp={0}",tmp);
    if (tmp == x)
      {
        return;
      }
    rec(x);
}
rec(5);

10. Function may be updated
function f()
{
  write("f");
}
f();
function f()
{
  write("f2");
}
f();

result:
f
Warning: [FuncCreateFunction] function [f] is updated
f2
```

```
11.  Passing structure as function parameter
definestruct Room
{
  var x;
  array y[3];
}
// function f(struct Toom:r) - wrong struct_name causes error
function f(struct Room:r)
{
  set r.x = 1;
  set r.y[0] = "A";
}
createstruct R as Room;
f(R);
```

```
12.   Passing structure array as function parameter
definestruct Room
{
  var x;
  array y[3];
}
function f(struct array Room:r)
{
  set r.0.x = 1;
  set r.0.y[0] = "A";
}
createstruct R[2] as Room;
f(R);
```

Examples with using public and private functions in Trees Functions and Global it is possible to find in directory Examples\Access.

```
13. Array as parameter (see Examples\ArrayFunc\mean.scr)
// return result via parameter
function mean1(array:arr,result)
{
   result = 0;
   for(i,0,length(arr))
       result = result + arr[i];
   result = result/ length(arr);
}
var result;
call mean1({1,2,3,4,5,6,7,8,9,10},result);
write#("mean1 = {0}",result);

//return result via operator 'return'
function mean2(array:arr)
{
   var result = 0;
   for(i,0,length(arr))
       result = result + arr[i];
   result = result/ length(arr);
   return result;
}
call mean2({1,2,3,4,5,6,7,8,9,10});
write#("mean2 = {0}",getresult);

// the following line is wrong, because  array.average is not
// function, it is ppl command
var result = array.average({1,2,3,4,5,6,7,8,9,10 });
```

```
14. function names - funcnames
function fora()
{
  write(funcname);
}
fora();

function defaulta()
{
  write(funcname);
}
defaulta();

function varx()
{
  write(funcname);
}
varx();
```

15. **see** examples\CallFunc\func21.scr

```
function f(x)
{
    write(x);
}
var c = "qqq";
f("v");    // literal
f(c);      // variable
f(v);      // error wrong argument name

>rc examples\CallFunc\func21.scr
result:
v
qqq
Error: [FuncExecFunction] function [f] argument [v] not found
```

call

Command ʼ**call**ʼ invokes a function in mode **scr**, it is possible to use expression in infix notation as arguments, do not need to enclose in parentheses each argument when function is called.
 Do not use ʼ**call**ʼ as function argument or on the right side of mathematical expression.

Format **scr**:
call function_name(arg1,arg2,…)

Examples:
```
1.
>code scr
>var x = 0;
>call Sum(x,4 + 1);      // right: 2nd arg in infix notation
>d x
-----Variables  and arrays-----
-L1      x         [5]

>call Sum(x,-(5)(2));   // wrong: 2nd arg in prefix notation
                        // right: Sum(x)(-(5)(2))

2.Call function without arguments or with one not-expression
argument by command 'call' and without 'call':
call func(123);    same as   func(123);

3. To get return of function, called by 'call',
   run 'dbg yes' before:

>dbg yes;
>call Math.PI();
result = 3.141592653589793
// or
>debugppl no;
>call Math.PI();
>write(getresult);
3.141592653589793

4.To get return from function, when function is used as
argument do not use 'call', use ppl_notation:
>write#( Math.PI() );
 result = 3.141592653589793

5.
>write#(call Math.PI());    // error: call on the right side
>set x = call Math.PI();    // error
>set x = Math.PI();         // right
3.141592653589793
```

6. command **"**call**"**allows to pass array by this manner:
call function_name({item1,item2,…},arg2,…)

Example: (see examples\CallFunc\funcarr2.scr and examples\CallFunc\funcarr2.ppl)
```
function SumArray(array:arr,var:result)
{
   for(i,0,length(arr))
      call Sum(result,arr[i]);
}

var result=0;
call SumArray({1,2,3,4,5},result);
write#("result = {0}",result);
```
instead of:
```
array arr[] = {1,2,3,4,5};
call SumArray(arr,result);
```

Preprocessor generates the following ppl-code:
```
function
(
  SumArray (array:arr)(var:result)
  (
    (set (result)(0))
    (
      loop (i)(0)(length(arr))(1)
      (
        do
        (
          (Sum(result)(result)(arr[i]))
        )
      )
    )
  )
);
var (result);
array (arg_array0)(1)(2)(3)(4)(5 );
SumArray (arg_array0 )( result );
del arg_array0;
write ("result = {0}")(result);
```

```
array split_array;
call String.Split("1 2 3 4 5",{" "},"split_array");  // error,
                      // because generated code without quotes
call String.Split("1 2 3 4 5",getname({" "}),"split_array");
// right
```

return

Returns from function or exit from script, passes result from called function.

Format **ppl: return [(result)]**
        result := value | statement in prefix notation

Format **scr: return [()result[)]**
        result := value| statement in infix notation

Examples:
1.
```
function f()
{
  for(i, 0, 5)    // step by default = 1
  {
     if (i == 2)
      return;
     write(i);
   }
}
f();
write("end of script");
Result:
0
1
```

2.
```
code scr;
function f()
{
  return 2 + 3;
}
f();
write(getresult);
Result: 5
```

getresult

Gets result of return passed from called function.

Format:
    getresult () | getresult

Example:
1. Return from function:
```
function sum(x,y)
{
    return x + y;
}
call sum(1,2);
write#("result = {0}",getresult);
result = 3
```

funclist

Displays loaded function names and their parameters from node Functions.
Format:
funclist| finclist()

Example:
```
>funclist;
Result:
-----Function List-----
Sum   (result, n1, n2)
Sub   (result, n1, n2)
Mult   (result, n1, n2)
Div   (result, n1, n2)
Pow   (result, n1, n2)
PlusPlus   (result)
MinusMinus   (result)
LT   (result, n1, n2)
LTEQ   (result, n1, n2)
GT   (result, n1, n2)
GTEQ   (result, n1, n2)
EQ   (result, n1, n2)
NOTEQ   (result, n1, n2)
AND   (result, n1, n2)
OR   (result, n1, n2)
XOR   (result, n1, n2)
```

funcname

Returns the current function name.
Format:
funcname() | funcname

Examples:
```
>write(funcname);
main
```

```
function MyFunction()
{
   write(funcname);
}
MyFunction();
Result: MyFunction
```

argc

Returns number of arguments
Format:
argc() | argc

Example:
```
function Sum (result,n1,n2 = "")
{
    write#("argc = {0}",argc());
    if (argc() == 2)
      result = result + n1;
    else
      result = n1 + n2;
}
>code scr;
>var x = 0;
>call Sum(x,1);
>write#("x = {0}",x);
Result:
argc = 2
x = 1
```

getargname

Returns argument name (or argument value if it is literal) by parameter name
Format:
getargname (parameter_name)

Example:
```
function a(n)
{
  write#("param name=[{0}] arg name=[{1}]",
       getname(n),getargname(n));
}
var i = 0;
a(i);
Result: param name=[n]  arg name=[i]
```

finally and failure blocks

finally block are called at the end of the successful completion of the script or functions.
failure block are called in case of emergency termination of the script or functions.
These blocks are added to the body of script or functions as additional nodes and have access
to all variables in their scope.

Example:
```
finally  { write("script finally block"); }
failure  { write("script failure block"); }

function foo(var: x)
{
  finally
  {
    var str1 = "function finally block";
    //writ(str1);
    //Error: [Traversal] [foo] [finally] wrong cmd or function
    //   name [writ]
    write(str1);
  }

  failure
  {
    var str2 = "function failure block";
    write(str2);
  }

  if(isdigits (x) == True)
  {
     write("digits");
     return;
  }
  else
  {
     write("not digits");
     return;
  }
}
foo("a");

Result:
not digits
function finally block
script finally block
```

Delegates and callbacks

There are 4 operators for using delegates:
delegate – creation delegate
dlgtinstance – creation delegate instance
dlgtset - setting the function to delegate instance
dlgtcall – call function by delegate instance

delegate

delegate is created as an array whose elements define the method parameters**.**
Prefix "**delegate_**" **is added to delegate name** for internal using.
Format **ppl**:
delegate (<delegate name>) (param1) (param2)...
param:= <var_name> | var:<var_name> | array:<array_name> |
        storage:<array_name> | struct [array] <struct_name>: < var_name >
Format **scr**:
delegate <delegate name> (param1, param2,…)

Example (**scr-mode**):
```
>delegate  MyDelegate (var:x,array:arr);
Generated ppl code:
>d
-N2     NS
---N3   Global
-----L0       empty                      (const)
-----N4 delegate_MyDelegate              [Array 2]
-------L0       #                              [var:x]
-------L1       #                              [array:arr]
```

dlgtinstance

dlgtinstance is created as an array with 2 elements, first is delegate name, second is empty and will be set by dlgtset. Delegate parameters types must be matched types of function parameters . Prefix **"dlgtinstance_"** is added to delegate instance name for internal using.

Format **ppl**:

dlgtinstance (<delegate instance name>)(<delegate name>)

Format **scr**:

dlgtinstance <delegate instance name> <delegate name>

Example(**scr-mode**):
```
>delegate d2 (var:x,array:arr);
>dlgtinstance instance d2;
>d;
-N2      NS
---N3    Global
-----L0 empty    (const)
-----N4 delegate_d2      [Array 2]
-------L0        #       [var:x]
-------L1        #       [array:arr]
-----N4 dlgtinstance_instance    [Array 2]
-------L0        #       ["delegate_d2"]
-------L1        #
```

dlgtset

dlgtset sets function name as second element in **dlgtinstance** array.
Format **ppl**:
dlgtset (delegate instance name)(function name)
Format **scr**:
dlgtset delegate instance name function name

Example (**scr-mode**):
```
// see previous example with delegate and dlgtinstance
function f1(var:x,array:arr)
{
    write#("x = [{0}]",x);
    writearray arr;
}
>dlgtset instance f1;
```

dlgtcall

dlgtcall calls function defined in dlgtset.
Format **ppl**:
dlgtcall(delegate instance name)(arg1)(arg2)(arg3)...
Format **scr**:
dlgtcall delegate instance name(arg1,arg2,arg3,...)

Example:
```
// see previous examples with delegate, dlgtinstance and dlgtset
>var z = "qqq";
>array y[] = {1,2,3};
>dlgtcall instance (z,y);
Result:
x = [qqq]
-----Array arr-----
[0]     1
[1]     2
[2]     3
```

See samples with delegates - Examples\delegates*.scr

callback

callback invokes synchronous callback method.

Format **ppl**:

callback (callback name)(arg1)(arg2)(arg3)…

Format **scr**:

callback callback name (arg1,arg2,arg3,…)

Example: (see Examples\delegates\callback.scr)

```
function cb1(var:n)
{
   write#("====={0}=====",funcname);
   write#("n = {0}",n);}

function cb2(var:n)
{
   write#("====={0}=====",funcname);
   write#("n = {0}",n);}

function f(array:x,var:str)
{
   callback x(str);
}

delegate d2 (var:n);
dlgtinstance instance d2;
dlgtset instance cb1;
call f(instance,"PPL");
dlgtset instance cb2;
call f(instance,"PPL");
Result:
function cb1 PPL
function cb2 PPL
```

Error Diagnostics

PPL Preprocessor locates the error in scr-mode:
Examples:

```
// non-interactive mode
File ErrorQM.scr:
var x;
x = 3 < 4 ? 1  2;    // right: x = 3 < 4 ? 1 : 2;
>rc ErrorQM.scr
Error:[ProcessingQuestionMark] file:[examples\x23.scr] line:
 [2] omitted ':' [set x = 3 < 4 ? 1  2;]

// interactive mode
>code scr;
>var x;
>x = 3 < 4 ? 1  2;
Error:  [ProcessingQuestionMark] omitted ':' [
set x = 3 < 4 ? 1 2;]
```

```
//File ErrorVar.scr
var x 2;        // right: var x = 2;
>rc examples\Error.scr
//Error:  [TFuncVar] file: [examples\x23.scr] line: [1] wrong
format cmd 'var' [var x 2;]
```

Additional functionalities

The following below-mentioned additional DLLs with C++ functionalities are added and this list will be expanded. Custom DLLs use functions from the C++ STL and internal PPL functions. There are two types of methods called from additional DLLs:
- methods that return result, this result may be used in the next operation, method arguments in prefix notation (**ppl-mode**)
  for example:
  ```
  var result = Math.Max(10)(*(2)(10));
  ```
- methods that not return result, for example:
  ```
  ArrayList.Remove(arrlist, item1);
  ```
  Methods of second type of may be called by command 'call' in **scr-mode** and method arguments in infix_notation:
  ```
  call ArrayList.Remove(arrlist, 1+2);
  ```

In the following sample it is created wrapper for calling method that returns result (examples\lib\char.scr):

```
function Wrapper_GetChar(result,text,index)
{
 result  =  String.Char(text)(index);
}

var char;
var text = "Hello";
for(i,0,length(text))
{
   call Wrapper_GetChar(char,text, i);
   write#("{0} {1}", i, char);
}
```
See how to [create wrappers by ULC](#).

To get list of methods of additional loaded DLLs:
<DLLname>.help

For using user's library it is needed to set it in Configuration.data or to add by command **import** in program.

Use Application ULC.exe([Structure of User's DLL](#)) to create code for additional DLLs. Arguments of additional DLLs use **prefix math.notation, only Math library uses infix math. notation in scr-mode, this feature is added to preprocessor**:
set x = Math.Sqrt(1+3) + Math.Max(2)(3*5);

Math

Methods:

Max	**E**	**PI**
Min	**Exp**	
BigMul	**Floor**	
Sqrt	**Log**	
Round	**Log10**	
Abs	**Pow**	
Acos	**Sign**	
Asin	**Sin**	
Atan	**Tan**	
Atan2	**Truncate**	
Ceiling	**Tanh**	
Cos	**Cosh**	
DivRem	**Sinh**	

Limitation: Do not use Math.Function in Math.Function:

```
var x = Math.Sin(30/(180/Math.PI())) // error
```

To get short help for every method in Math.DLL:
>Math.help[(method name)];

Returns the larger of two double-precision floating-point numbers:
Math.Max(double d1)(double d2)

Returns the smaller of two double-precision floating-point numbers:
Math.Min(double d1)(double d2)

Produces the full product of two 32-bit numbers:
Math.BigMul(Int32 n1)(Int32 n2)

Returns the square root of a specified number: **Math.Sqrt(double d1)**

Rounds a double-precision floating-point value to a specified number:
Math.Round (double value)[(Int32 digits)]

Returns the absolute value of a double-precision floating-point number:
Math.Abs(double value)

Returns the angle whose cosine is the specified number**: Math.Acos(double d)**
Returns the angle whose sine is the specified number: **Math.Asin(double d)**

Returns the angle whose tangent is the specified number**: Math..Atan(double d)**

Returns the angle whose tangent is the quotient of two specified numbers:
Math.Atan2(double d1)(double d2)

Returns the smallest integral value greater than or equal to the specified number:
Math.Ceiling(double d)

Returns the cosine of the specified angle: **Math.Cos(double d)**

Returns the quotient and remainder, separated by '; ' as result:
Math.DivRem(Int64 n1)(Int64 n2)
Example:
```
write#("result: {0}",Math(9)(2));
result: 4;1
```

Represents the ratio of the circumference of a circle to its diameter: **Math.PI()**

Represents the natural logarithmic base: **Math.E()**

Returns e raised to the specified power: **Math.Exp(double value)**

Returns the largest integral value less than or equal to the specified number:
Math.Floor(double value)

Returns the logarithm of a specified number: **Math.Log(double value)**

Returns the base 10 logarithm of a specified number: **Math.Log10(double value)**

Returns a specified number raised to the specified power:
Math.Pow(double value)(double power)

Returns an integer that indicates the sign of a double-precision floating-point number:
Math.Sign(double value)

Returns the sine of the specified angle: **Math.Sin(double value)**

Returns the tangent of the specified angle: **Math.Tan(double value)**

Calculates the integral part of a number: **Math.Truncate(double value)**

Returns the hyperbolic tangent of the specified angle: **Math.Tanh(double value)**

Returns the hyperbolic cosine of the specified angle: **Math.Cosh(double value)**

Returns the hyperbolic sine of the specified angle: **Math.Sinh(double value)**

String

Methods:

Compare	**Replace**
Concat	**DeleteEndOfLine**
Contains	**StartsWith**
Format	**Substring**
IndexOf	**ToCharArray**
LastIndexOf	**ToLower**
Insert	**ToUpper**
Remove	**Trim**
Split	**Char**
SplitCsv	

To get short help for every method in String.DLL:
>String.help[(method name)];

Returns signed int as string: **String.Compare(stringA)(stringB)**

Returns concatenation of several strings: **String.Concat(string1)(string2)**...

Returns true|false: **String.Contains(string)(specified substring)**

Converts the value of objects to string based on the formats specified and returns result:
String.Format(format)(string1)(string2)...
Example:

```
String.Format("{0} {1}")("qwe")("zxc")
Result: qwe zxc
```

Returns a new string in which a specified number of characters from the current string are deleted:
String.Remove(string)(startIndex)(number of deleted symbols)
Example:
>rc examples\lib\StringRemove.scr

```
import String;
array primes = {1,2,3,5,7};
var output = "";
for(i,0,length(primes),1)
{
  set output = String.Concat(output)(primes[i])(",");
}
var index = length(output) - 1;
set output = String.Remove(output)(index)(1);  //remove the
 latest ','
write(output);
Result:1,2,3,5,7
```

Returns a new string in which all occurrences of a specified Unicode character or string in the current string are replaced with another specified Unicode character or string:

String.Replace(string)(old value)(new value)

Determines whether this string instance starts with the specified character:
Returns **True|False**:

String.StartsWith(string)(value)

Retrieves a substring from this instance. The substring starts at a specified character position and has a specified length:

String.Substring(string)(startIndex)[(length)]

Examples:

```
>var r = String.Substring("123asd")(0)(3);
result: r = 123
>var r = String.Substring("123asd")(3);
result: r = asd
```

Writes the characters in this instance to a Unicode character array:

String.ToCharArray(string)("ppl_chars_array")

node_of PPL_chars_array is string in quotes or value of variable.

```
Example:
>array chars;
>String.ToCharArray("qwerty")("chars"));
// or  String.ToCharArray("qwerty")(getname(chars));
>writearray chars row;
Result:
-----Array chars-----
q,w,e,r,t,y
```

Returns a copy of this string converted to lowercase: **String.ToLower(string)**

Returns a copy of this string converted to uppercase: **String.ToUpper(string)**

Returns a new string in which all leading and trailing occurrences of a set of specified characters from the current string are removed:

String.Trim(string)[(trim chars string)]

```
>String.Trim(" abcde")(" ae");
Result: bcd
```

Returns one character from string: **String.Char(string)(index)**

Returns the reallocated string array that contains the substrings in this instance that are delimited by elements of a specified string array or in special string var, string array must be created before with size = 0:

String.Split(string)("ppl_array_separators")(getname(ppl_array_result))   or
String.Split(string)("var_separator")(getname(ppl_array_result))
It is possible to use   comma instead of ','
        space instead of ' '
        tab instead of '\t'
String.Split does not skip separators between quotes.

Example:
> var text;
```
>set text = File.ReadAllText("Data\test.csv");
>array separator[] = {comma,tab,space};
>array split_text_array;
>String.Split(text)("separator")(getname(split_text_array));
```

Returns the reallocated string array that contains the substrings in this instance that are delimited by separator of a specified string var, string array must be created before with size = 0 (array <name>;).
If value with separator is surrounded by quotes it doesn't split.
(see example\lib\splitcsv.scr):

array ppl_array_result;
String.SplitCsv(string)("var_separator")("ppl_array_result")
It is possible to use   comma instead of ','
                    space instead of ' '
                    tab instead of '\t'

String.SplitCsv skips separators between quotes.
Example:
Returns string from File.ReadAllText without EndOfLine: **DeleteEndOfLine(string)**
```
>rc Examples\Lib\FilesplitCsv.scr

var text = File.ReadAllText("examples\lib\splitcsv.txt");
array splitcsv_text;   // array with results
call String.SplitCsv(text, ",", "splitcsv_text");
// or
// call String.SplitCsv(text, comma, "splitcsv_text");
writearray splitcsv_text row;
call File.WriteAllText("text","examples\lib\splitcsv_copy.txt");
```

Reports the zero-based index of the first occurrence of the specified string in this instance:

String.IndexOf (string)(value)[(start_index)][(count)]

Reports the zero-based index of the last occurrence of the specified string in this instance:

String.LastIndexOf (string)(value)[(start_index)][(count)]

Example   (see examples\lib\IsUniqueSymbol.scr)

```
function IsUnique(str,result)
{
  var char;
  for(i,0,length(str))
  {
   set char = String.Char(str)(i);
   if( String.LastIndexOf (str)(char) != i)
   {
     set result = "False";
     return;
   }
  }
  set result = "True";
}

var result;
call IsUnique("abcdef",result);
write(result);
call IsUnique("1234567",result);
write(result);
call IsUnique("abcABC",result);
write(result);
call IsUnique("abcadef",result);
write(result);
Result:
True
True
True
False
```

Returns a new string in which a specified string is inserted at a specified index position in this instance:

String.Insert (string)(start index)(string to insert)

Directory

Methods:
> **GetFiles**
> **GetDirectories**
> **SetCurrentDirectory**
> **GetCurrentDirectory**
> **GetParent**
> **CreateDirectory**
> **Exists**
> **Delete**

To get short help for every method in Directory.DLL:
> Directory.help[(method name)];

Writes the names of files (including their paths) in the specified directory to node_of_PPL_array, created before with size = 0:
array node_of_PPL_array;
Directory.GetFiles("node_of_PPL_array")("path")
node of PPL array is string in quotes or value of variable or
getname(node_of_PPL_array).
Example:

```
1.
>array files;
>Directory.GetFiles("files") ( "c:\" );
or
var (x["files"]);
>Directory.GetFiles(x)(path);

2.
     >rc examples\lib\WriteFilesInDir.scr

function WriteFilesInDirectory (array:arr,dir)
{
  array arr;
  Directory.GetFiles(arr)(dir);
  Writearray arr;
}
WriteFilesInDirectory ("files")("c:\");
```

```
Result:
-----Arr files-----
[0]     c:\DumpStack.log.tmp
[1]     c:\hiberfil.sys
[2]     c:\pagefile.sys
[3]     c:\swapfile.sys
```

Writes the names of directories (including their paths) in the specified directory to **node_of_PPL_array**, created before with size = 0:

array node_of_PPL_array;
Directory.GetDirectories("node_of_PPL_array")("path")

node of PPL array is string in quotes or value of variable or getname(**node_of_PPL_array**).

Example:
```
array dir;
Directory.GetDirectories("dir")("c:\Users");
or
var (x["dir"]);
```
Sets the current working directory to the specified directory:
Directory.SetCurrentDirectory("path")

Gets the current working directory: **Directory.GetCurrentDirectory()**

Returns parent fullname: **Directory.GetParent("path")**

Returns CreationTime: **Directory.CreateDirectory("path")**

Returns **True** or **False** : **Directory.Exists("path")**

Deletes the specified directory and any subdirectories and files in the directory
Returns **True** or **False**: **Directory.Delete("path")**

Queue

      Methods:

            Create   **Peek**
            Count   **Clear**
            Write   **Contains**
            Enqueue   **AddArray**
            Dequeue   **ToArray**   **Delete**

To get short help for every method in Queue.DLL:
Queue. help[(method name)]

Creates Queue object: **Queue.Create(queue name)**

Returns the number of elements actually contained in Queue: **Queue.Count(queue name)**

Writes queue names or all elements from the specified queue to the standard output stream:
Queue.Write() or Queue.Write(name)

Adds an object to the end of the **Queue: Queue.Enqueue(queue name)(string)**
To add empty string use keyword empty.

Removes and returns the object at the beginning of the Queue:
Queue.Dequeue(queue name)

Returns the object at the beginning of the Queue without removing it:
Queue.Peek(queue name)

Removes all objects from the Queue: **Queue.Clear(queue name)**

Determines whether an element is in the Queue, returns "True" or "False":
Queue.Contains(queue name)(string)

Adds PPL array to the **Queue: Queue.AddArray("PPL array") (queue name)**

Writes all elements from Queue to the PPL array created before with size = 0:
array "ppl_array";
Queue.ToArray(queue name) ("ppl_array")

Delete all Queue objects: **Queue.Delete();**

Examples:

Note the difference between value and "value" (also for **Stack** and **Dictionary**)

```
>import Queue
Imported [Queue]
>var x = 0;
>array r[] = {1,2,3};
>Queue.Create(Q)
>call Queue.Enqueue(Q,x)
>call Queue.Enqueue(Q,"x")
>call Queue.Enqueue(Q,r)
>call Queue.Enqueue(Q,"r")
>Queue.Write(Q)
===Q===
        0
        x
        Array 3
        r
```

It is possible to save in **Queue** (also in **Stack** and **Dictionary**) lines of code in ppl-format to be performed using **eval**:

```
>import Queue
Imported [Queue]
>Queue.Create(Q)
>var y = 1
>Queue.Enqueue(Q)("var (x[0]);Sum(x)(y)(2);d x;")
>Queue.Write(Q)
===Q===
        var (x[0]);Sum(x)(y)(2);d x;
>eval(Queue.Dequeue(Q))
-----Variables  and arrays-----
-L0      x                            [3]
```

Using delegates with **Queue** (also in **Stack** and **Dictionary**)
File examples\delegates\dlgt6.scr

```
function f1(var:x,array:arr)
{
   write#("x = [{0}]",x);
   writearray arr row;
}
delegate  dd (var:n,array);
dlgtinstance instance dd;

var z = "qqq";
array y[] = {1,2,3};

import Queue;
Queue.Create(Q);
Queue.Enqueue(Q)(f1);
dlgtset instance Queue.Dequeue(Q);
dlgtcall instance(z,y);

result:
code: scr
x = [qqq]
-----Array arr-----
1, 2, 3
```

See samples of code with Queue methods in **examples\lib\Queue.scr**

Stack

Methods:

Create	**Peek**	
Count	**Clear**	
Write	**Contains**	
Push	**AddArray**	
Pop	**ToArray**	**Delete**

To get short help for every method in Stack.DLL:
>Stack.help[(method name)]

Creates Stack object: **Stack.Create(stack name)**

Returns the number of elements actually contained in Stack: **Stack.Count(stack name)**

Writes stack names or all elements from the specified stack to the standard output stream:
Stack.Write() or Stack.Write(stack name)

Inserts an object at the top of the stack: **Stack.Push(stack name)(string)**
To insert empty string use keyword **empty**.

Removes and returns the object at the top of the Stack:
Stack.Pop(stack name)

Returns the object at the top of the Stack without removing it:
Stack.Peek(stack name)

Removes all objects from the Stack: **Stack.Clear(stack name)**

Determines whether an element is in the Stack, returns "**True**" or "**False**":
Stack.Contains(stack name)(string)

Adds PPL array to the Stack: **Stack.AddArray) ("PPL array")(stack name)**

Writes all elements from Stack to the PPL array created before with size = 0:
array "ppl_array";
Stack.ToArray(stack name) ("ppl_array")

Delete all Stack objects: **Stack.Delete();**

Examples:
```
>dbg yes;
>import Stack
Imported [Stack]
>Stack.Create(s)
>Stack.Push(s)(one)
>Stack.Push(s)(two)
>Stack.Push(s)(three)
>debug yes
>Stack.Pop(s)
result = three
>Stack.Pop(s)
result = two
>Stack.Pop(s)
result = one
>Stack.Pop(s)
result = empty
```

See samples of code with Stack methods in **examples\lib\Stacks.scr**

Dictionary

This library uses functions from **std::unordered_map**.
Methods:

Create	**Clear**
Delete	**Write**
Count	**ContainsKey**
Add	**Remove**
GetValue	**AddArray**
SetValue	**ToArray**

To get short help for every method in Dictionary.DLL:
>**Dictionary.help[(method name)]**

Creates Dictionary object: **Dictionary.Create(dictionary name)**

Deletes all Dictionary objects: **Dictionary.Delete()**
or deletes one object from Dictionary: **Dictionary.Delete(dictionary_name)**

Example:
```
>import Dictionary;
>Dictionary.Create(dict1);
>Dictionary.Create(dict2);
>Dictionary.Create(dict3);
>Dictionary.Delete(dict1);    // delete dict1
>Dictionary.Delete();         // delete dict2 and dict3
```

Returns the number of elements actually contained in Dictionary:
Dictionary.**Count(dictionary name)**

Adds the specified key and value to the Dictionary:
Dictionary.**Add(dictionary name)(key)(value)**

Dictionary.**GetValue**
return **True** or **False**: **Dictionary.GetValue(dictionary name)(key)(var name for result)**

Dictionary.SetValue
return **True or False: Dictionary.SetValue(dictionary name)(key)(value)**

Dictionary.Write
Writes dictionary names only: **Dictionary.Write()**
or
all elements from the specified Dictionary to the standard output stream:
Dictionary.Write(dictionary name)

Removes all objects from the Dictionary: **Dictionary.Clear(dictionary name)**

Determines whether the Dictionary contains the specified key, returns **True** or **False**:
Dictionary.ContainsKey(dictionary name)(key)

Removes the value with the specified key from the Dictionary:
Dictionary.Remove(dictionary name)(value)
Adds PPL array to the Dictionary: **Dictionary.AddArray("PPL array")(dictionary name)**

```
>import Dictionary
Imported [Dictionary]
>Dictionary.Create(D)
>array y[3]
>setkvp y[0] = one,1
>setkvp y[1] = two,2
>setkvp y[2] = three,3
>Dictionary.AddArray("y")(D)
>Dictionary.Write(D)
Dictionary.Write [D]
        one     1
        two     2
        three   3
```

Write all elements from Dictionary to new PPL array created before with size = 0:
array "ppl_array";
Dictionary.ToArray(dictionary name) ("ppl_array")

Examples of code with Dictionary methods in **examples\lib\Dictionary.scr**

Convert

Methods:
StringToInt32Array
StringToHexArray
HexToBin
BinToHex
IntToHex
HexToInt
IntToBin
BinToInt

To get short help for every method in Convert.DLL:
>Convert.help[(method name)];

String chararacters converts to int32 array created before with size = 0 and reallocated in
Convert.StringToInt32Array with size of string_chararacters:
Convert.StringToInt32Array(string_chararacters)("Int32 "ppl_array"")

String chararacters converts to hex array created before with size = 0 and reallocated in
Convert.StringToHexArray with size of string_chararacters:
Convert.StringToHexArray(string_chararacters)("Hex "ppl_array"")

All below mentioned methods convert data in accordance with method name and return:
Returns string bin: **Convert.HexToBin(string with hex value)**
Returns string hex: **Convert.BinToHex(string with bin value)**
Returns string hex: **Convert.IntToHex(string with Int32 value)**
Returns string Int32: **Convert.HexToInt(string with hex value)**
Returns string bin: **Convert.IntToBin(string with Int32 value)**
Returns string Int32: **Convert.BinToInt(string with bin value)**

Examples:
See Examples\lib\Convert.scr

```
>array Int32;
>Convert.StringToInt32Array("12345")("Int32");
Info [CreateArrayFormat2] Global array [Int32] is created
>writearray Int32;
Result:
-----Array Int32-----
[0]     49
[1]     50
[2]     51
[3]     52
[4]     53
>array Hex;
>Convert.StringToHexArray("12345")("Hex");
```

```
>writearray Hex;
Result:
-----Array Hex-----
[0]      31
[1]      32
[2]      33
[3]      34
[4]      35
```

Examples:
```
>dbg yes
>Convert.HexToBin(16);
result = 10110
>Convert.BinToHex(1111111)
result = 7F
>Convert.IntToHex(256)
result = 100
```

Excel

The following methods may be used for reading from XLSX files to two-dimensional storage or writing from two-dimensional storage to XLSX files.

Methods:
Open
Close
Read
CreateWorkBook
Write
SaveAs

To get short help for every method in Excel.DLL:
> Excel.help[(method name)];

Opens XLSX file for reading:
Excel.Open(filename.xlsx)

Closes XLSX file after reading or writing:
Excel.Close()

Reads opened XLSX to storage, size of storage must be enough to save Excel cells:
Excel.Read("sheet")("left top")("right down")("storage")
Example:
"left top": "A1"
"right down": "H10"

Creates workbook for writing:
Excel.CreateWorkBook()

Writes storage to Excel cells, quantity of cells must be enough to save storage:
Excel.Write("sheet")("left top")("right down")("storage")

Saves created XLSX file after writing:
Excel.SaveAs(filename.xlsx)

Examples:
see file Examples\Excel\test.scr
```
import Excel;
Excel.Open("$1$\examples\Excel\example.xlsx");
Excel.Read("Sheet1")("A1")("H10")("Example_XLSX");
Excel.Close();
swrite(Example_XLSX);
```

```
Excel.CreateWorkBook();
Excel.Write("Sheet1")("A1")("H10")("Example_XLSX");
Excel.SaveAs("$1$\examples\Excel\example2.xlsx");
Excel.Close();
```

>rc examples\excel\test.scr c:\path
Parameter **c:\path** overrides the variable **1** in file test.scr.

File

Methods:

ReadAllText	**ReadAllLines**
WriteAllText	**WriteAllLines**
Exists	**Delete**

To get short help for every method in **File.DLL**:
>File.help[(method name)];

Returns all contents of text file:**File.ReadAllText(filename)**

Creates a new file, write the contents to the file, and then closes the file:
File.WriteAllText(var_ppl)(filename)

Determines whether the specified file exists, returns True or False: **File.Exists(filename)**

Returns string array with lines of text file:**File.ReadAllLines(filename)("ppl_array")**

Creates a new file, writes one or more strings to the file, and then closes the file:
File.WriteAllLines(""ppl_array"")(filename)

Deletes the specified file: **File.Delete(filename)**
Example:
>rc examples\lib\File.scr

```
    var text;

    //examples\lib\split.txt
    //1,2,3,4,5,6,7,8,9,10,
    //11,12,13,14,15,16,17,18,19,20

    text = File.ReadAllText("examples\lib\split.txt");
    write(text);
    call File.WriteAllText(text,"examples\lib\Copy.txt");
    var b = File.Exists("examples\lib\Copy.txt");
    write(b);
    if (b == "True")
       File.Delete("examples\lib\Copy.txt");
    b = File.Exists("examples\lib\Copy.txt");
    write(b);

    array arr[] = {"One","Two","Three"};
    call File.WriteAllLines(getname(arr),"examples\lib\Copy2.txt");
    array arr2;
    call File.ReadAllLines("examples\lib\copy2.txt",getname(arr2));
    writearray arr2;
```

```
call File.Delete("examples\lib\copy2.txt");
```

```
Result:
1,2,3,4,5,6,7,8,9,10,
11,12,13,14,15,16,17,18,19,20
True
False
-----Array arr2-----
[0]      <"One"
[1]      <"Two"
[2]      <"Three"
```

Random

The Random linbrary use the Mersenne Twister pseudorandom number generator (PRNG).

Methods:
Create	**Delete**
Next	**Write**
NextToArray	

To get short help for every method in **Random.DLL**:
>Random.help[(method name)];

Creates Random object: **Random.Create(random_name)(type)(from)(to)**
type:= int | real

Returns a random integer: **Random.Next(random_name)**

Creates random numbers and writes them to the of a specified **"ppl_array"** created before with size = 0:
Random.NextToArray(random name)(""ppl_array"")(quantity of random elements)

Deletes created random_name or all random_names:
Random.Delete(random_name) | Random.Delete()

Writes random_names: **Random.Write()**

Example:
File examples\lib\random.scr

```
import Random;
call Random.Create(intR,int,1,10);
write(Random.Next(intR));
write(Random.Next(intR));
write(Random.Next(intR));

array int_arr;
call Random.NextToArray(intR,"int_arr",10);
writearray int_arr row;

call Random.Create(dblR,real,1,10);
write(Random.Next(dblR));
write(Random.Next(dblR));
write(Random.Next(dblR));

array double_arr;
call Random.NextToArray(dblR,"double_arr",10);
```

```
writearray double_arr row;

write("===Random.Delete===[dblR]");
Random.Delete(dblR);
Random.Write();

write("===Random.Delete===[all]");
Random.Delete();
Random.Write();
Result:
Imported [Random]
9
2
10
-----Array int_arr-----
9, 2, 10, 10, 3, 7, 4, 1, 6, 3
2.219293
8.515077
9.719810
-----Array double_arr-----
2.98931, 3.7735, 5.92499, 2.69544, 9.93593, 9.96815, 9.70925,
7.53255, 9.82999, 1.98876
===Random.Delete===[dblR]
0       intR    int
===Random.Delete===[all]
```

Console

For using in scr or ppl files, not for console input.
      Methods:

ForegroundColor	**Beep**
BackgroundColor	**Clear**
ForegroundPromptColor	**SetCursorPosition**
DefaultColors	**GetCursorPosition**
Write	
WindowWidth	**WindowHeight**

To get short help for every method in **Console.DLL**:
>Console.help[(method name)];

Sets the foreground color of the console:
      Console.ForegroundColor(color)
Sets the background color of the console:
      Console.BackgroundColor(color)
Sets the prompt foreground color of the console**:**
      Console.ForegroundPromptColor(color)
Sets the default foreground, background and ForegroundPromptColor color of the console:
      Console.DefaultColors()

Writes the text representation of the specified value or values to the standard output stream: **Console.Write [(format)](string)(string)**
Plays the sound of a beep through the console speaker:
      Console.Beep (frequency)(duration)
      frequency - 37 to 32767 hertz
      duration - msec
Clears the console buffer and corresponding console window of display information:
      Console.Clear()
Sets the position of the cursor:
      Console.SetCursorPosition(left column cursor position)(top row cursor position)
Gets the position of the cursor**:**
      Console.GetCursorPosition ()
Returns '**left column cursor position, top row cursor position'**

Only for Administrator mode:
Run function **WindowSize()   (see file Functions\CommonFunctions.ppl)**
>WindowSize()
width=120 height=30
Get/Set **WindowWidth([value])   and WindowHeight([value])**

```
>Console.WindowHeight(20)           // set
>write(Console.WindowHeight())   // get
20
```

See examples in examples\Console.

DateTime and TimeSpan
See **detailed e**xplanations in https:
//learn.microsoft.com/en-us/dotnet/api/system.datetime.subtract?view=net-8.0

Methods:

Clear	**AddMicroseconds**
Warning	**AddMilliseconds**
CreateDateTime	**AddMinutes**
GetDateTime	**AddMonths**
CreateTimeSpan	**AddSeconds**
GetTimeSpan	**AddTicks**
Subtract	**AddYears**
Add	**Compare**
AddDays	**DaysInMonth**
AddHours	**IsLeapYear**

To get short help for every method in DateTime.DLL:
>DateTime.help[(method name)];

See samples in **Examples\DateTime\dt1.scr**
\dt2.scr

Clear
Removes all DateTime and Timespan instances.
DateTime.Clear()

Warning
Sets to display or not warning when DateTime or TimeSpan instance is updated.
DateTime.Warning(yes | no)

CreateDateTime
Creates named instance of the DateTime.
DateTime.CreateDateTime(name)
DateTime.CreateDateTime(name)(Now | UtcNow | Today)
DateTime.CreateDateTime(name)(year)(month)(day)
DateTime.CreateDateTime(name)(year)(month)(day)(hour)(min)(sec)
DateTime.CreateDateTime(name)(year)(month)(day) (hour)(min)(sec)(msec)

GetDateTime
Returns instance of DateTimeProperties structure,
DateTimeProperties structure defined in file: Functions\Commonfunctions.scr.
DateTime.GetDateTime(DateTime instance name)
(DateTimeProperties instance name)

Example:

```
import DateTime;
rc Functions\DateTimeProperties.scr;
call DateTime.CreateDateTime(dt1,2024,1,1);
createstruct dtProperties as DateTimeProperties;
call DateTime.GetDateTime(dt1,  dtProperties);
dtProperties.WriteDateTimeProperties();
```

```
=====DateTimeProperties dtProperties=====
Date: 1/1/2024 12:00:00 AM
Day: 1
DayOfWeek: Monday
DayOfYear: 1
Hour: 0
Kind: Unspecified
Microsecond: 0
Millisecond: 500
Minute: 0
Month: 1
Nanosecond: 0
Second: 0
Ticks: 638396640005000000
TimeOfDay: 00:00:00.5000000
```

CreateTimeSpan

Creates named instance of the TimeSpan.

DateTime.CreateTimeSpan(instance Timespan name)(ticks)
DateTime.CreateTimeSpan(instance Timespan name)(hours)(minutes)(seconds)

DateTime.CreateTimeSpan
(instance Timespan name) (days)(hours)(minutes)(seconds)

DateTime.CreateTimeSpan
(instance Timespan name) (days) (hours)(minutes)(seconds)(msec)

DateTime.CreateTimeSpan
(instance Timespan name) (days) (hours)(minutes)(seconds)(msec)(mksec)

GetTimeSpan

Returns instance of TimeSpan.

DateTime.GetTimeSpan(instance Timespan name)(hours)(minutes)(seconds)

Example:

```
import DateTime;
call DateTime.CreateTimeSpan(ts1,5,3,1);
write#("DateTime.CreateTimeSpan {0}",getresult);
write#("DateTime.GetTimeSpan {0}",DateTime.GetTimeSpan(ts1));
DateTime.CreateTimeSpan 05:03:01
```

> **DateTime.GetTimeSpan 05:03:01**

Subtract

Returns the value that results from subtracting the specified time or duration from the value of this instance.

DateTime.Subtract(name_src DateTime) (name TimeSpan) (name_result DateTime)

Example:
```
call DateTime.CreateDateTime(dt,2024,1,9);
write#("DateTime.CreateDateTime {0}",getresult);

call DateTime.CreateTimeSpan (ts,8,0,0,0);  //name,days, hours, minutes,seconds

call DateTime.CreateDateTime(dtResult);
call DateTime.Subtract(dt,ts,dtRes);
write#("DateTime.Subtract {0}",getresult);
```
DateTime.CreateDateTime 1/9/2024 12:00:00 AM
DateTime.Subtract 1/1/2024 12:00:00 AM

Add

Returns a new DateTime instance that adds the value of the specified TimeSpan to the value of this instance.

DateTime.Add(name_src DateTime) (name TimeSpan) (name_result DateTime)

AddDays

Returns a new DateTime that adds the specified number of days to the value of this instance.

DateTime.AddDays(name_src DateTime) (days) (name_result DateTime)

AddHours

Returns a new DateTime that adds the specified number of hours to the value of this instance.

DateTime.AddHours(name_src DateTime) (hours) (name_result DateTime)

AddMicroseconds

Returns a new DateTime that adds the specified number of microseconds to the value of this instance.

DateTime.AddMicroseconds(name_src DateTime)
(microseconds) (name_result DateTime)

AddMilliseconds

Returns a new DateTime that adds the specified number of milliseconds to the value of this instance.

DateTime.AddMilliseconds(name_src DateTime)
(microseconds) (name_result DateTime)

AddMinutes

Returns a new DateTime that adds the specified number of minutes to the value of this instance.

DateTime.AddMinutes(name_src DateTime)(minutes) (name_result DateTime)

AddMonths

Returns a new DateTime that adds the specified number of months to the value of this instance.

DateTime.AddMonths(name_src DateTime)(months) (name_result DateTime)

AddSeconds

Returns a new DateTime that adds the specified number of seconds to the value of this instance.

DateTime.AddMinutes(name_src DateTime)(seconds) (name_result DateTime)

AddTicks

Returns a new DateTime that adds the specified number of ticks to the value of this instance.

DateTime.AddMinutes(name_src DateTime)(ticks) (name_result DateTime)

AddYears

Returns a new DateTime that adds the specified number of years to the value of this instance.

DateTime.AddMinutes(name_src DateTime)(years) (name_result DateTime)

Compare

Compares two instances of DateTime and returns an integer that indicates whether the first instance is earlier than, the same as, or later than the second instance.

DateTime.Compare(name1 DateTime)(name2 DateTime)

DaysInMonth

Returns the number of days in the specified month and year.

DateTime.DaysInMonth(year)(month)

IsLeapYear

Returns an indication whether the specified year is a leap year.

DateTime.IsLeapYear(year)

Vector

Using Vector and Matrix libraries significantly increases program performance.
Methods:

Vector.Create ("vector_name")(length)(type)

type:= double|float|decimal|bool|int|uint|long|ulong|string

types in accordance with value types in

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/built-in-types

Vector.Get(vector_name)(index)

Vector.Set(vector_name)(index)(value)

Vector.Add("vector_name")(""ppl_array"")

Write to line - "row"
Write not nullable data to column - "col"
Write all data to column - "col0"

Vector.Write("vector_name")[("row"|"col"|"col0")]

Vector.WriteNames()

Vector.Delete("vector_name")

Vector.DeleteAll()

Vector.Sort ("vector_name")(order)

order:= ascend | descend

To get short help for every method in **Vector.DLL**:
>Vector.help[(method name)];

Examples:
```
>import Vector;
>array v[] = {1,2,3,4,5};
>call Vector.Create ("V",5,int);
>call Vector.Add("V",getname(v));
> call Vector.Write("V");
 Result:
=====vector V=====
1       2       3       4       5
>call Vector.Set("V",0,0);
> call Vector.Write("V",col);
Result:
=====vector V=====
[1]       2
[2]       3
[3]       4
[4]       5
>call Vector.Delete("V");
```

See examples **in examples\MatrixVector.**

Matrix

Methods:
- **Matrix.Create ("matrix_name")(rows)(columns)(type)**
- **type:= double|float|decimal|bool|int|uint|long|ulong|string**
- **types in accordance wth value types in**
  - https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/built-in-types
- **Matrix.Get("matrix_name")(index row)(index_column)**
- **Matrix.Set("matrix_name")(index row)(index_column)(value)**
- **Matrix.AddArrayToRow ("matrix_name")(row)(""ppl_array"")**
- **Matrix.AddArrayToColumn ("matrix_name")(column)(""ppl_array"")**
- **Matrix.Write to line - "row"**
- **Matrix.Write not nullable data to column - "col"**
- **Write all data to column - "col0"**
- **Write("matrix_name")[("row"|"col"|"col0")]**
- **Matrix.WriteNames()**
- **Matrix.Delete("matrix_name")**
- **Matrix.DeleteAll()**
- **Matrix.Rotate("matrix_name")("cw" | "ccw")   - for square matrix only**
- **Matrix.Sort("matrix_name")(column index| order)**
- **order:= ascend | descend**

To get short help for every method in Matrix.DLL:
>Matrix.help[(method name)]

Example:
```
import Matrix;
call Matrix.Create("D",3,3,double);
array r11[] = {1,2,3};
array r12[] = {4,5,6};
array r13[] = {7,8,9};


call Matrix.AddArrayToRow("D",0,"r11");
call Matrix.AddArrayToRow("D",1,"r12");
call Matrix.AddArrayToRow("D",2,"r13");


// Rotation CCW
call Matrix.Write("D");
call Matrix.Rotate("D","ccw");
write("===ccw===");
Matrix.Write("D");

// Rotation CW
call Matrix.Rotate("D","cw");
```

```
write("===cw===");
Matrix.Write("D");
Matrix.Delete("D");
```

Result:

```
=====matrix D=====
[0]     1          2          3
[1]     4          5          6
[2]     7          8          9
===ccw===
=====matrix D=====
[0]     3          6          9
[1]     2          5          8
[2]     1          4          7
===cw===
=====matrix D=====
[0]     1          2          3
[1]     4          5          6
[2]     7          8          9
```

```
>rc Examples\MatrixVector.scr:

import Matrix;
call Matrix.Create("D",3,3,double);
array r11[] = {1,2,3};
array r12[] = {4,5,6};
array r13[] = {7,8,9};
call Matrix.AddArrayToColumn("D",0,getname(r11));
call Matrix.AddArrayToColumn("D",1,getname(r12));
call Matrix.AddArrayToColumn("D",2,getname(r13));
Matrix.Write("D");
call Matrix.Sort("D",0,descend);
Matrix.Write("D");
Matrix.DeleteAll();
```

Result:

```
code: scr
Imported [Matrix]
=====matrix D=====
[0]     1          4          7
[1]     2          5          8
[2]     3          6          9
=====matrix D=====
[0]     3          6          9
[1]     2          5          8
[2]     1          4          7
```

See examples in **examples\MatrixVector.**

DataFrame

DataFrame is a table with named columns, columns may be defined with different types,number of rows and columns is not limited.

Methods:

SetRow	**AddRowData**
SetColumn	**GetRowArray**
Write	**GetSliceRowArray**
Save	**GetColumnArray**
ReadFile	**GetSliceColumnArray**
Init	**SetCell**
List	**GetCell**
Create	**CellName**
InsertRows	**SetWidth**
AddRows	**SetType**
InsertColumns	**GetWidth**
AddColumns	**GetType**
RemoveRows	**SetPrintEmptyRows**
RemoveColumns	**GetPrintEmptyRows**
ClearColumns	**SetRowSelectedFrom**
SetWidthForAll	**GetRowSelectedFrom**
SetTypeForAll	**SetRowSelectedTo**
Sort	**GetRowSelectedTo**
Reverse	**SetReallocStep**
SelectRows	**GetReallocStep**
UnselectRows	**GetColumnNames**
GetRowsLength	
GetColumnLength	

To delete DataFrame – **del DataFrame_name**

To get short help for every method in **DataFrame.DLL**:
>**DataFrame.help[(method name)]**

DataFrame.Init – to add this call in first .scr file only, if project contains several .scr-files.
Format: **DataFrame.Init()**

DataFrame.Create - creates node df_name, node Settings and arrays per each column

Formats:
DataFrame.Create() - creates DataFrame "DF" with 26x26 columns with following names: A,B,C,… BA,BB,BC,…ZA,…ZZ and 1000 rows

DataFrame.Create(Name) - creates DataFrame "Name" with 26x26 columns with following names: A,B,C,… BA,BB,BC,…ZA,…ZZ
and 10 rows

DataFrame.Create(Name)(rows length) - creates DataFrame "Name" with 26 columns with following names: A,B,C,…Z and "rows length" rows

DataFrame.Create(Name)(rows length)(column1)(column2)…(columnN) - creates
 DataFrame "Name" with N columns and "rows length" rows

```
>import DataFrame
>DataFrame.Create(DF)(2)(A)(B)
>d
-N2      NS
---N3    Global
-----L0 empty    (const)
-----N4 DF              [DataFrame]
-------N5       Settings
---------L0     RowSelectedFrom
---------L1     RowSelectedTo
---------L2     PrintEmptyRows   [yes]
---------L3     RowsLength       [2]
---------L4     ReallocStep        [10]
---------L5     AType     [Text]
---------L6     AWidth    [12]
---------L7     BType     [Text]
---------L8     BWidth    [12]
-------N5       A                [Array 2]
---------L0     #
---------L1     #
-------N5       B                [Array 2]
---------L0     #
---------L1     #
```

DataFrame.Settings.RowSelectedFrom and **DataFrame. Settings.RowSelectedTo** sets selected rows and used in **DataFrame.Read**, **DataFrame.Save**, **DataFrame.Write.**

DataFrame.Settings.PrintEmptyRows sets to present or not empty rows and used in **DataFrame.Write**.

DataFrame. Settings.RowsLength is set automatically as number of rows.

DataFrame. Settings.ReallocStep sets number of allocated rows and used in **DataFrame Create** and **DataFrame AddRows**.

DataFrame. Settings.<column name>Type sets column type and used in **DataFrame**.**Read** and **DataFrame**.**Sort**.

DataFrame. Settings.<column name>Width sets column width and used in Write.

1. Example:


```
>import DataFrame
Imported [DataFrame]
>DataFrame.Create(AB)(10)
>DataFrame.Write(AB)

===============AB================

     A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R   S   T   U   V   W   X   Y   Z

0
1
2
3
4
5
6
7
8
9
>
```

2.   Example

```
>import DataFrame
>DataFrame.Create(MyDataFrame)(2)(One)(Two)(Three)
-----Variables  and arrays-----
-N4     MyDataFrame      [Node]
---N5   Settings         [Node]
---N5   One                 [Array 2]
---N5   Two                 [Array 2]
---N5   Three               [Array 2]
>DataFrame.Write(MyDataFrame)
Result:
     One            Two            Three
0
1
```

DataFrame.SetRow and **DataFrame.SetColumn** return **false** if **columnType = Number** and value is not digital

DataFrame.SetRow - sets values for each column for defined row from file

Format:
DataFrame.SetRow(df_name)(row index)("ppl_array")

DataFrame.SetColumn - sets values for each row for defined column from file

Format:
DataFrame.SetColumn(df_name)(column) ("ppl_array")

DataFrame.Write - displays DataFrame full contents or only contents of defined columns

Format:
DataFrame.Write[(df_name)] [(column)(column)…]

To display contents of specific rows call before **DataFrame.Write**:
DataFrame. RowSelectedFrom (df_name)(row index)
DataFrame. RowSelectedTo (df_name)(row index)
 or by method SelectRows.
DataFrame.SelectRows(df_name)(row index from)(row index to)

DataFrame.Save -saves full contents of DataFrame in file format **".csv"** or **".data"** or defined columns only in format **".csv"**

Format:
 DataFrame.Save(df_name)(filename.csv|.data)[(column)(column)]…
DataFrame.Save(df_name)(filename.data)

DataFrame.ReadFile - reads full contents of DataFrame from file in format **".csv"** or **" .data"** or contents of defined columns from file in format **".csv" only**

Format:
(1)   DataFrame.ReadFile(df_name) (filename.csv)[(column)(column)]…
or
(2)   DataFrame.ReadFile(filename.data)
Format (2) deletes existing data and creates new Dataframe or several DataFrames with names from file.data

Example:

```
File df.data
(DF [DataFrame]
 (Settings
    (RowSelectedFrom)
    (RowSelectedTo)
    (PrintEmptyRows [yes])
    (RowsLength [3])
    (ReallocIncrement [10])
    (NameType [Text])
    (NameWidth [12])
    (QuantityType [Text])
    (QuantityWidth [12])
  )
 (Name [Array 3]
   (# ["CARS"])
   (# ["TREES"])
   (# ["ROADS"])
 )
 (Quantity [Array 3]
   (# [20])
   (# [30])
   (# [40])
 )
)

>import DataFrame;
>call DataFrame.ReadFile(examples\DataFrame\df.data);
>call DataFrame.Write(DF);
```

Result:

```
     Name         Quantity

0    "CARS"       20
1    "TREES"      30
2    "ROADS"      40
```

DataFrame.InsertRows - inserts number of rows before index

Format:
DataFrame.InsertRows(df_name)(index)[(number of rows)]
By default number of added rows= 1

DataFrame.AddRows - adds number of empty rows at the end

Format:
DataFrame.AddRows([df_name])[(number of rows)]
By default df_name = DF
By default number of added rows: **DataFrame.Settingd.ReallocatedIncrement = 10**

DataFrame.InsertColumns - insert one or several named columns before defined column

Format:
DataFrame.InsertColumns(df_name))(defined column1)(column2)(column3)…

DataFrame AddColumns - adds number of named columns at the end

Format:
DataFrame.AddColumns(df_name)(column1)(column2)…

DataFrame RemoveRows – removes rows

Format:
DataFrame.RemoveRows(df_name) – removes all rows
DataFrame.RemoveRows(df_name)[(number_of_row) – removes 1 row
DataFrame.RemoveRows(df_name)(number_from)(*) - removes rows
                       between number_from to the end of rows
DataFrame.RemoveRows(df_name)(number_from)(number_to) - removes rows
                       between number_from – number_to

DataFrame.RemoveColumns - removes named columns and their Type and Width in **Settings**

Format: **DataFrame.RemoveColumns(df_name)[(column1)(column2)…]**

DataFrame.RemoveColumns(df_name) - removes all columns and their Type and Width in **Settings**:

DataFrame.ClearColumns - clears contents of all DataFrame or defined columns:
Format: **DataFrame.ClearColumns(df_name)[(column1)(column2)...]**

DataFrame.SetWidthForAll - sets same width for all columns to display DataFrame by **DataFrame.Write**

Format: **DataFrame.SetWidthForAll [(df_name)] (width)**
default name – DF
default with = 12

Example:
```
>DataFrame.SetWidthForAll(14)
```

DataFrame.SetTypeForAll - sets same type for all columns

Format: **DataFrame.SetTypeForAll [(df_name)] [(Text | Number)]**
default name – DF
default type = Text

Example:
```
>DataFrame.SetTypeForAll(Number)
```

DataFrame.Sort - sorts all contents by specified column in ascending or descending order

Format: **DataFrame.Sort(df_name)(ascend | descend)(column)**

Example:

```
File: examples\DataFrame\products.csv
Bagel,140,310,Medium
Buiscuit ,86,480,High
Jaffa cake,48,370,Med-High
Bread white,96,240,Medium
Bread wholemeal,88,220,LowMed
Chapatis,250,240,Medium
Cornflakes,130,300,Med-High

Program: examples\DataFrame\df8.scr
import DataFrame;
DataFrame.Create(Products)(0)(Bread&Cereals)(Size)(per100grams)
                          (energy);
set Products.Settings.Bread&CerealsWidth = 20;
DataFrame.ReadFile(Products)(examples\DataFrame\products.csv);
DataFrame.Write(Products);
DataFrame.Sort(Products)(ascend)(per100grams);
DataFrame.Write(Products);
```

results:

```
 Forest                    ×    +  ∨                          —   □   ×

>rc examples\dataframe\df8.scr
code: scr
Imported [DataFrame]
DataFrame [Products] added [7] rows

================Products================

     Bread&Cereals      Size        per100grams energy

0    Bagel              140         310          Medium
1    Buiscuit           86          480          High
2    Jaffa cake         48          370          Med-High
3    Bread white        96          240          Medium
4    Bread wholemeal    88          220          LowMed
5    Chapatis           250         240          Medium
6    Cornflakes         130         300          Med-High
=====DF Reverse=====

================Products================

     Bread&Cereals      Size        per100grams energy

0    Cornflakes         130         300          Med-High
1    Chapatis           250         240          Medium
2    Bread wholemeal    88          220          LowMed
3    Bread white        96          240          Medium
4    Jaffa cake         48          370          Med-High
5    Buiscuit           86          480          High
6    Bagel              140         310          Medium
=====DF Sorted order Ascending=====
```

See examples df8.scr and df9.scr(wrong results for digital data when Type = Text).

DataFrame.Reverse - reverses all DataFrame contents by columns

Format: **DataFrame.Reverse(df_name)**

DataFrame.SelectRows

Format:
Select one row only: **DataFrame.SelectRows(df_name)(select_from)**
Select rows select_from – to end: **DataFrame.SelectRows(df_name)(select_from)(*)**
Select rows between select_from – select_to:
DataFrame.SelectRows(df_name)(select_from)(select_to)

DataFrame.UnSelectRows

Format: **DataFrame.UnSelectRows(df_name)**

DataFrame.GetRowsLength – returns number of rows

Format: **DataFrame.GetRowLength(df_name)**

DataFrame.GetColumnsLength – returns number of columns

Format: **DataFrame. GetColumnsLength (df_name)**

DataFrame.AddRowData – adds row with data per columns at the end

Format: **DataFrame.AddRowData(df_name)("ppl_array")**

DataFrame.GetRowArray – copies data from defined row to ppl array

Format:
DataFrame.GetRowArray(df_name)(index row) ("ppl array")

DataFrame.GetSliceRowArray - copies dataframe data, defined by column,row_from and row_to to ppl array

Format:
To the end of rows:
DataFrame. GetSliceRowArray (df_name)(column)(row_from)("ppl array")
or
DataFrame. GetSliceRowArray (df_name)(column)(row_from)(row_to)("ppl array")

DataFrame.GetColumnArray - copies data from defined column to array

Format:
DataFrame.GetColumnArray(df_name)(column) ("ppl array")

DataFrame.GetSliceColumnArray – copies dataframe data, defined by row, column_from and column_to to array

Format:
To the end of columns:
DataFrame. GetSliceColumnArray (df_name)(row)(column_from)("ppl array")
or
DataFrame. GetSliceColumnArray (df_name)(row)(column_from)(column_to)("ppl array")
Example:
```
File: Examples\DataFrame\df22.scr
>rc examples\dataframe\df22.scr
```

```
import DataFrame;
call DataFrame.Create(Table,0,A,B,C,D,E);
call DataFrame.ReadFile(Table,examples\dataframe\Table.csv);
call DataFrame.Write(Table);

array sliced_arr;
write("Tests DataFrame.GetSliceColumnArray");
call DataFrame.GetSliceColumnArray(Table,0,B,"sliced_arr");
writearray sliced_arr row;
array.clear(sliced_arr);
call DataFrame.GetSliceColumnArray(Table,0,B,D,"sliced_arr");
writearray sliced_arr row;

write("Tests DataFrame.GetSliceRowArray");
array.clear(sliced_arr);
call DataFrame.GetSliceRowArray(Table,B,1,"sliced_arr");
writearray sliced_arr row;

array.clear(sliced_arr);
call DataFrame.GetSliceRowArray(Table,B,1,3,"sliced_arr");
writearray sliced_arr row;
Result:
code: scr
Imported [DataFrame]
DataFrame [Table] added [5] rows

A            B            C            D            E

0    1            2            3            4            5
1    10           20           30           40           50
2    100          200          300          400          500
```

```
3    1000         2000         3000         4000         5000
4    10000        20000        30000        40000        50000
Tests DataFrame.GetSliceColumnArray
-----Array sliced_arr-----
2, 3, 4, 5
-----Array sliced_arr-----
2, 3, 4
Tests DataFrame.GetSliceRowArray
-----Array sliced_arr-----
20, 200, 2000, 20000
-----Array sliced_arr-----
20, 200, 200
```

DataFrame. SetCell - sets value for cell ,defined by name or by column name and row index

Format:
DataFrame.SetCell(df_name)(column name)(index row)(value)
or
DataFrame.SetCell(df_name.cell_name)(value)

DataFrame.GetCell - returns value of cell, defined by name or by column name and row index

Format:
DataFrame.GetCell(df_name)(column name)(index row)
or
DataFrame.GetCell(df_name.cell_name)

DataFrame.CellName - sets name for defined cell

Format: **DataFrame.CellName(df_name)(column name)(index row)(cell name)**

DataFrame.SetWidth - sets Width value for defined column name in **Settings**

Format: **DataFrame. SetWidth (df_name)(column name)(value)**

DataFrame.SetType - sets Type value for defined column name in **Settings**

Format: **DataFrame. SetType (df_name)(column name)(Text | Number)**

DataFrame.GetWidth - returns Width for defined column name in **Settings**

Format: **DataFrame. GetWidth (df_name)(column name)**
Example:
```
>import DataFrame
>DataFrame.Create(DF)(2)(A)(B)
>DataFrame.SetWidth(DF)(A)(15)
>write(DataFrame.GetWidth(DF)(A))
result: 15
```

DataFrame.GetType - returns Type for defined column name in **Settings**

Format: **DataFrame. GetType(df_name)(column name)**
Example:
```
>import DataFrame
>DataFrame.SetType(DF)(A)(Number)
>write(DataFrame.GetType(DF)(A))
result: Number
```

DataFrame.SetPrintEmptyRows - sets PrintEmptyRows value in **Settings**

Format: **DataFrame.SetPrintEmptyRows (df_name)(value)**
value:= yes | no

DataFrame.GetPrintEmptyRows - returns PrintEmptyRows value in **Setting**

Format: **DataFrame.GetPrintEmptyRows (df_name)**
Example:
```
>import DataFrame;
>DataFrame.Create(DF)(2)(A)(B);
>DataFrame.SetPrintEmptyRows(DF)(yes);
>write(DataFrame.GetPrintEmptyRows(DF));
result: yes
```

DataFrame.SetRowSelectedFrom - sets RowSelectedFrom value in **Settings**

Format: **DataFrame. SetRowSelectedFrom (df_name)(index_row_from)**

DataFrame.GetRowSelectedFrom - returns RowSelectedFrom value in **Setting**

Format: **DataFrame. GetRowSelectedFrom (df_name)**

DataFrame.SetRowSelectedTo - sets RowSelectedTo value in **Settings**

Format: **DataFrame. SetRowSelectedTo (df_name)(index_row_to)**

DataFrame.**GetRowSelectedTo** – returns RowSelectedTo value in **Settings**

Format: **DataFrame. GetRowSelectedTo (df_name)**

Example:
```
>code scr;
>import DataFrame;
>call DataFrame.Create(DF,2,A,B);
>call DataFrame.SetRowSelectedFrom(DF,0);
>call DataFrame.SetRowSelectedTo(DF,1);
>write(DataFrame.GetRowSelectedFrom(DF));
result: 0
>write(DataFrame.GetRowSelectedTo(DF));
result: 1
```

DataFrame.**SetReallocStep** - sets ReallocStep value in **Settings**

Format: **DataFrame.SetReallocStep(df_name)(value)**

DataFrame.**GetReallocStep** - returns ReallocStep value in **Settings**

Format: **DataFrame.GetReallocStep(df_name)**
Example:
```
>import DataFrame;
>DataFrame.Create(DF)(2)(A)(B);
>DataFrame.SetReallocStep(DF)(100);
>write(DataFrame.GetReallocStep(DF));
result: 100
```

DataFrame.GetColumnNames – returns string with column names, separated by comma

Format:
DataFrame.GetColumnNames(df_name)
Example:
```
>import DataFrame;
>DataFrame.Init();
>call DataFrame.Create(Table,5,A,B,C,D,E);
>write(DataFrame.GetColumnNames(Table));
result:A,B,C,D,E
```

DataFrame.List – displays all created DataFrame names, their settings and cell names

Format: **DataFrame.List()**
Example:
```
>import DataFrame;
>DataFrame.Init();
>call DataFrame.Create(Table1,10,A,B,C,D,E);
>call DataFrame.CellName(Table1.A,0,A0);
>call DataFrame.CellName(Table1,B,0,B0);
>call DataFrame.SetWidthForAll(Table1,10);
>call DataFrame.List();
```
Result:
```
=====Table1=====
  Settings
        RowSelectedFrom =
        RowSelectedTo =
        PrintEmptyRows = yes
        RowsLength = 10
        ReallocIncrement = 10
        AType = Text
        AWidth = 10
        BType = Text
        BWidth = 10
        CType = Text
        CWidth = 10
        DType = Text
        DWidth = 10
        EType = Text
        EWidth = 10
  defined cell names in DataFrame [Table1]
        name              column     row
        A0                A          0
        B0                B          0
```

Examples of using DataFrame methods in directory: examples\DataFrame
df1.scr – df25.scr,AddressBook2.scr, AddressBook.scr

Structure of User's DLL

Utility **CodeGen.exe** generates files .h and .cpp in accordance with structure of User's DLL, described in configuration file.

Example:

Configuration file **Stack.cfg**

```
ProjectName = Stack
Methods = reate,Delete,Count,Write,Push,Pop,Peek,Clear,Contains,
          AddArray,ToArray
```

Run CodeGen.exe Stack.cfg

CodeGen.exe generates 2 files: Stack.cpp and Stack.h

Stack.cpp

```cpp
#include "pch.h"
#include "..\PPL\PPL.h"
#include "..\PPL\Component\Component.h"
#include "..\PPL\Processing\processing.h"
#include "Stack.h"

using namespace std;
namespace PPLNS
{
  static Stack* STACKInstance = nullptr;

  void Stack_CreateInstance(PPL* ppl)
  {
    STACKInstance = new Stack(ppl);
    STACKInstance->AddToKeywordDictionary();
  }

 Stack::Stack(PPL* ppl)
  {
    this->ppl = ppl;
    keyword_dict = new unordered_map<string, function<bool(const
vector<string>&, string&, Composite*)>>;
  }
  //=================================================
  void Stack::AddToKeywordDictionary()
  {
    //help_dict is created in BaseClass
    AddKeyword("help", BaseClass::FuncHelp);
    AddKeyword("Create", FuncCreate);
    AddKeyword("Delete", FuncDelete);
    AddKeyword("Count", FuncCount);
    AddKeyword("Write", FuncWrite);
    AddKeyword("Push", FuncPush);
    AddKeyword("Pop", FuncPop);
    AddKeyword("Peek", FuncPeek);
    AddKeyword("Clear", FuncClear);
```

```
    AddKeyword("Contains", FuncContains);
    AddKeyword("AddArray", FuncAddArray);
    AddKeyword("ToArray", FuncToArray);

    help_dict->insert({ "help", "\tStack.help([name])" });
    help_dict->insert({ "Create", "\t..." });
    help_dict->insert({ "Delete", "\t..." });
    help_dict->insert({ "Count", "\t..." });
  help_dict->insert({ "Write", "\t..." });
    help_dict->insert({ "Push", "\t..." });
    help_dict->insert({ "Pop", "\t..." });
    help_dict->insert({ "Peek", "\t..." });
    help_dict->insert({ "Clear", "\t..." });
    help_dict->insert({ "Contains", "\t..." });
    help_dict->insert({ "AddArray", "\t..." });
    help_dict->insert({ "ToArray", "\t..." });
    for (const auto pair : *keyword_dict)
    {
      string key = "Stack." + pair.first;
      ppl->processing->keyword_dict->insert({ key, pair.second });
    }
    ppl->ImportList.insert({ "Stack", this });
  }
  //========================================================
  bool Stack::FuncCreate(const vector<string>& parameters, string&
result, Composite* node) { return true; }
  bool Stack::FuncDelete(const vector<string>& parameters, string&
result, Composite* node) { return true; }
  bool Stack::FuncCount(const vector<string>& parameters, string&
result, Composite* node) { return true; }
  bool Stack::FuncWrite(const vector<string>& parameters, string&
result, Composite* node) { return true; }
 bool Stack::FuncPush(const vector<string>& parameters, string&
result, Composite* node) { return true; }
  bool Stack::FuncPop(const vector<string>& parameters, string&
result, Composite* node) { return true; }
  bool Stack::FuncPeek(const vector<string>& parameters, string&
result, Composite* node) { return true; }
  bool Stack::FuncClear(const vector<string>& parameters, string&
result, Composite* node) { return true; }
  bool Stack::FuncContains(const vector<string>& parameters, string&
result, Composite* node) { return true; }
  bool Stack::FuncAddArray(const vector<string>& parameters, string&
result, Composite* node) { return true; }
  bool Stack::FuncToArray(const vector<string>& parameters, string&
result, Composite* node) { return true; }
}
```

Stack.h

```cpp
/***************************************************************
*This code generated by Application CodeGen.exe                *
*for creation User DLL                                         *
*Add  Project Reference PPL.Lib                                *
*Add  pch.h from Project PPL                                   *
*Author: Oscar Kogosov, email: ok21@hotmail.com               *
*You must not remove this notice from this software.          *
***************************************************************/
#ifndef STACK_H
#define STACK_H

#ifdef STACK_EXPORTS
#define STACK_API __declspec(dllexport)
#else
#define STACK_API __declspec(dllimport)
#endif

namespace PPLNS
{
  class STACK_API Stack : public BaseClass
  {
  public:
    Stack(PPL* ppl);
    void AddToKeywordDictionary();
    bool FuncCreate(const vector<string>& parameters, string& result,
Composite* node);
    bool FuncDelete(const vector<string>& parameters, string& result,
Composite* node);
    bool FuncCount(const vector<string>& parameters, string& result,
Composite* node);
    bool FuncWrite(const vector<string>& parameters, string& result,
Composite* node);
    bool FuncPush(const vector<string>& parameters, string& result,
Composite* node);
    bool FuncPop(const vector<string>& parameters, string& result,
Composite* node);
    bool FuncPeek(const vector<string>& parameters, string& result,
Composite* node);
    bool FuncClear(const vector<string>& parameters, string& result,
Composite* node);
    bool FuncContains(const vector<string>& parameters, string&
result, Composite* node);
    bool FuncAddArray(const vector<string>& parameters, string&
result, Composite* node);
    bool FuncToArray(const vector<string>& parameters, string&
result, Composite* node);
  };
  extern "C" void STACK_API Stack_CreateInstance(PPL* ppl);
}
```

```
#endif
```

Add commentary in lines:
```
help_dict->insert({ "…", "\t..." });
```

Add code to each of methods in Stack.cpp –
```
        FuncCreate,
        FuncDelete,
        …….
```
Add to project file from CodeGen\x64\Debug\pch.h
Add to Project References the project **PPL.**
Add created DLL-file to directory Forest.

Error detection

Error detection in script files is carried out in three stages -
1. at the stage of reading files - checking the pairing of parentheses and square brackets, as well as the pairing of quotes;
2. at the pre-translation stage - checking the syntax of operators and commands, indicating name of PPL-module, the file name and line number;
3. at runtime stage.

Right code:
```
var x = 1;
write (x);
```

```
var x = 1;
write x);
Error: [TCheckNumberOfParentheses] not paired number of parentheses
       [write x)]
```

```
var x = 1;
awrite (x);
Error:  [TCreateCodeTree] file: [examples\err5.scr] line: [2] wrong
       cmd [awrite]
```

```
var x  1;
write (x);
Error:  [TFuncVar] file: [examples\err5.scr] line: [1] wrong format
       cmd 'var' [var x  1;]
```

Error detection in interactive mode:
```
>var x 1;
Error:  [TFuncVar] wrong format cmd 'var' [var x 1;]
Error: [HandlingServiceCommands] CodeTree = null
```

Examples of code

```
>rc examples\scr\Eratosthenes.scr
//Sieve of Eratosthenes
var m = 1000;
showcode no;
var len = m + 1;
array primes[len];

for (k,0,len)
 primes[k] = k;
// the fastest algorithm
start;
  var n = 2;
  var j = 0;

  for ()
  {
    if(n  * n >= len)
       break;

    if (primes[n] != 0)
    {
      j = n * n;
      for()
      {
        if(j >= len)
          break;
        primes[j] = 0;
        j = j + n;
      }
    }
    n = n + 1;
  }
stop("1. duration = ");
start;

var output = "";
for (i,0,len)
{
  if (primes[i] != 0)
    output = String.Concat(output)(primes[i])(",");

}

var index = length(output) - 1;

output = String.Remove(output)(index)(1);  // remove the latest ','
write#("{0}",output );
stop("2. duration = ");
```

Result:

```
     1. duration =  50.6454 msec
     1,2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,
     73,79,83,89,97,101,103,107,109,113,127,131,137,139,149,151,
     157,163,167,173,179,181,191,193,197,199,211,223,227,229,233,
     239,241,251,257,263,269,271,277,281,283,293,307,311,313,317,
     331,337,347,349,353,359,367,373,379,383,389,397,401,409,419,
     421,431,433,439,443,449,457,461,463,467,479,487,491,499,503,
     509,521,523,541,547,557,563,569,571,577,587,593,599,601,607,
     613,617,619,631,641,643,647,653,659,661,673,677,683,691,701,
     709,719,727,733,739,743,751,757,761,769,773,787,797,809,811,
     821,823,827,829,839,853,857,859,863,877,881,883,887,907,911,
     919,929,937,941,947,953,967,971,977,983,991,997
     2. duration =  18.3782 msec
```

Another faster solution when processing is in progress by C++ code:
>rc examples\scr\Eratosthenes.scr

```
import Erato;
start;
call Erato.Solve3(1000);
stop("duration = ");
```

Result:

```
1,2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,11
3,127,131,137,139,149,151,157,163,167,173,179,181,191,193,197,199,211,223,227,229,233,2
39,241,251,257,263,269,271,277,281,283,293,307,311,313,317,331,337,347,349,353,359,367,
373,379,383,389,397,401,409,419,421,431,433,439,443,449,457,461,463,467,479,487,491,49
9,503,509,521,523,541,547,557,563,569,571,577,587,593,599,601,607,613,617,619,631,641,6
43,647,653,659,661,673,677,683,691,701,709,719,727,733,739,743,751,757,761,769,773,787,
797,809,811,821,823,827,829,839,853,857,859,863,877,881,883,887,907,911,919,929,937,94
1,947,953,967,971,977,983,991,997
     duration =  0.3654 msec
```

Here C++ code:

```cpp
void Erato::Print(vector<int> v)
{
  std::stringstream ss;
  for (size_t i = 0; i < v.size(); ++i)
  {
    ss << v[i];
    if (i < v.size() - 1)
```

```cpp
      ss << ",";
    }
    std::cout << ss.str() << std::endl;
}
// the fastest algorithm
// Processing data without Vector & print results with erasing 0-
elements after processing
bool Erato::FuncSolve3(vector<string> parameters, string& result,
Composite* node)
{
  const std::string function_name = "Erato.Solve3";
  if (parameters.size() != 1)
  {
    cerr << "Error: [" << function_name << "] wrong format" << endl;
    return false;
  }
  try
  {
    int vector_length = stoi(parameters[0]);
    vector<int> v(vector_length);
    for (int i = 0; i < vector_length; i++) v[i] = i;

    for (int n = 2; n * n < vector_length; n++)
    {
      if (v[n] != 0)
      {
        // the fastest algorithm
        for (int j = n * n; j < vector_length; j += n)
        {
          v[j] = 0;
        }
      }
    }
    v.erase(remove(v.begin(), v.end(), 0), v.end());
    Print(v);
  }
  catch (const exception& ex)
  {
    cerr << "Error: [" << function_name << "] ... " << ex.what() <<
endl;
    return false;
  }

  return true;
}
```

Open console window:
cmd
and run:

Examples.bat
Callfunc.bat
Delegates.bat
Struct.bat
Dataframe.bat
ArrayFunc.bat
with numerous examples of code.

References

1. https://github.com/okogosov/PPL/blob/main/TutorialPPL.pdf