

Table of Contents

Introduction.....	5
Base Concepts	7
Trees	7
Comments.....	11
Configuration	12
Identifiers and DNS.....	12
Hidden variables	12
Public and private variables.....	13
Libraries	14
Keywords	15
CPPL utility.....	17
WPPL utility	19
Service Commands	22
help	22
version	22
cls	22
shell.....	22
init.....	22
code	22
showcode.....	23
readcode.....	23
fdreadcode.....	24
createpprcode.....	24
display	24
displaynodes	25
dstree.....	25
datanames	26
suspend and resume.....	26

debugppl	27
log	27
exit	27
Special Commands	28
import	28
importlist	28
eval.....	29
length	29
isexist	30
isdigit	30
isinteger	30
isalldigits	30
isallinteger	31
iseven, isodd	31
del	31
calc	32
getbykey	33
getbyvalue	33
set	33
getvalue	34
getname.....	34
Nodes and Leaves.....	35
createnode.....	35
copynode	35
getnodes	36
getleaves.....	38
Arithmetic operators.....	40
Logical operators	40
Variables and Arrays.....	42
var	42
const	42
array.....	43
realloc	44

Storage	46
storage	46
sinit	47
sget	47
sset.....	48
swrite	48
sinfo	49
ssetrow	49
Backup & Recovery.....	50
savedata.....	50
readdata.....	52
Control Flow	53
if, else.....	53
switch, case, default	55
loop,do.....	57
for	58
break	60
continue.....	60
Input and Output.....	61
write.....	61
writearray	62
readline.....	64
Functions	65
function.....	66
return	71
funclist	71
Additional functionalities	72
Math	72
String.....	74
Directory	77
ArrayList.....	79
Queue	83
Stack.....	84

Dictionary.....	86
Convert	87
Excel	89
File.....	91
Random.....	92
Vector	94
Matrix	95
MN_Numerics.....	96
Structure of User's DLL.....	98
Examples of code	100
References.....	105

Introduction

PPL is the **Parenthesis Programming Language**, in which all elements (statements, parameters, blocks) are enclosed in parentheses. PPL includes a preprocessor to simplify the writing programs and reduce the number of parentheses.

There are some PPL languages (see [References](#)), and this language is not the latest with such abbreviation. The only thing that unites all these languages is the abbreviation.

PPL was developed with Microsoft Developer Studio ,C#, without using any third party packages.

The main PPL feature – extensibility, using functionalities of C# and adding user's libraries by means of creating DLLs in accordance with [template](#), described in this tutorial.

PPL supports 2 modes:

ppl (base) mode, which syntax is similar to language LISP, math and logical expressions in prefix notation (**ppl expression**).

Examples:

```
var (x [0]);  
set(x) (+ (1) (2));  
if(==(x) (1)) ...
```

scr (preprocessor) mode, which syntax is similar to language C, math and logical expressions in infix notation (**scr expression**).

Examples:

```
var x = 0;  
set x = 1 + 2;  
if (x == 1)...
```

PPL includes 2 levels of parsing - code written in scr mode translates to ppl mode before executing, parser on each level creates syntax tree.

CPPL and WPPL utilities call PPL API functions, PPL API may be used in other user applications.

Mode scr or ppl is set depending on file extension is being executed or by means of the command code, mode scr makes coding easier as it does not require statements to be enclosed in parentheses. By default mode is **ppl**.

Preprocessor includes the following statements –

var, const, array, set

and following compound statements (blocks) –

function, for, if, else, switch, case, default.

All ppl mode statements except above mentioned may be also added to scr code in format ppl.

Data are saved as Unicode symbols, digital data convert to string,

Examples:

```
set x = 5.2; saved as "5.2"
```

Boolean values are saved as strings - "True" and "False":

```
set x = True;
```

Execution of the program in the language PPL is carried out by means of the utilities cppl.exe or wppl.exe, which control commands are listed in section [Keywords](#).

There are different statement formats for ppl mode and scr mode if a statement belongs to two modes.

Base Concepts

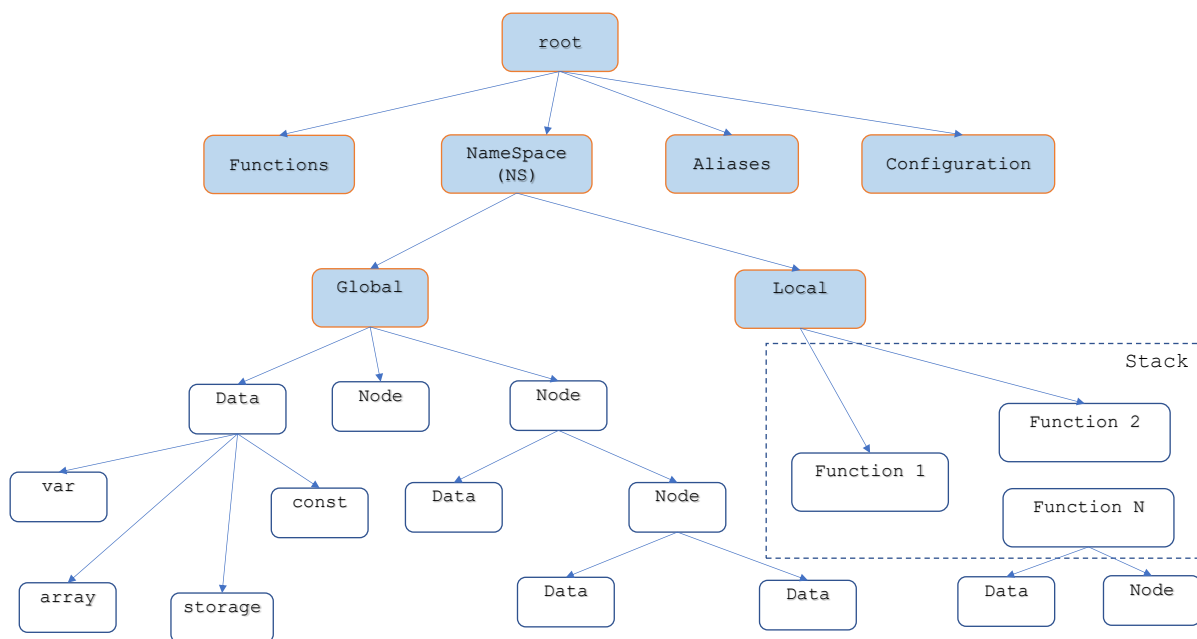
As it is customary in many programming language guides the first ppl program is:

```
write("Hello World!");  
without main function.
```

Trees

All information is stored in PPL as several kinds of Trees –

root, NS, Functions, Aliases, Configuration and may be displayed on Screen, saved and restored.



Blue nodes are created automatically when Cppl.exe loads or re-created by command **init**.

By default Tree **Functions** is filling from file Functions\mathlogic.ppl, defined as "**default_loaded_functions**" in file Configuration.data, it may be changed by user on other one, 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 **Aliases** is filling from file **Aliases.data**, to display its contents perform:

```
>display Aliases;
```

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

```
>display Configuration;
```

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.

```
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))
    )
);
=== results ===
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.

```
(Configuration
  (default_loaded_functions [Functions\mathlogic.ppl])
  (Code [ppl])
  (debugppl [no])
  (log [no])
  (stay_interactive [no])
  (max_number_hiddden_variables [10])
  (ReplaceMathLogicOperators [no])
  (OFD_port [11000])
  // (UserFunctions1 [Functions\*.ppl])
  // (UserFunctions2 [Functions\*.ppl])
  (UserImport1 [Directory])
  (UserImport2 [Math])
  // (UserImport3 [String])
)
```

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 used for special goals only (see [Hidden variables](#)).

Length of identifiers is not limited. Do not set keywords, their aliases 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, function DNS 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\""
```

Hidden variables

Hidden variables are used when it is necessary to pass an array member or storage member as a parameter to a function where its value will be changed.

As well hidden variables are used when index for accessing array members defined as expression (see example in [Arithmetic operators](#)).

These variables are created and deleted automatically and not visible on screen when command display is executed.

Hidden variables are created automatically for each function, their names start with "_f", hidden variables out of functions (in "main function") are created automatically also, their names start with "_main".

Max number of hidden variables is set in Configuration file as max_number_hidden_variables (by default = 5).

Names of hidden variables: _main0, _main1, ..., _main9, _f0, _f1, ..., _f9.

The example shows how to use hidden variables:

```
function
(
  func(arg)
  (
    (set(arg) ("Hello!"))
  )
);
>array(y[5]) (0);
>set(_main0) (y[0]);    // y[0].value => _main0
>func(_main0);
>set(y[0]) (_main0);    // changed _main0 => y[0].value
>write(y[0]);
```

Public and private variables

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

Examples:

```
>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))
  )
);
```

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

```
>var(x);
>var(x);    // re-creation
Error: [FuncCreateVariables] name [x] already exists
>del x;
>var(x);
```

Libraries

Default name of library is **Main**, it loads always when Cppl.exe or Wppl.exe starts. It is possible to set in file **Configuration.data** as "**UserImportN**" names of additional libraries. 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|Aliases|Functions|Local|node name]  
display NS.namespace.name]
```

Keywords

All keywords are divided into 9 groups and presented below:

Service Commands

help, version, cls, shell, init, code, showcode, readcode, fdreadcode, createppplcode, display, displaynodes, dn, dstree, datanames, suspend, resume, debugppl, log, exit

Special Commands

import, importlist, eval, length, calc, isexist, isdigits, isinteger, isalldigits, isallinteger, iseven, isodd, del, getbykey, getbyvalue, set, getvalue, getname

Nodes and Leaves

createnode, copynode, getnodes, getleaves

Variables and Arrays

var, const, array, realloc, empty

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

Input Output

write, writearray, readline

Functions

function, funclist, return

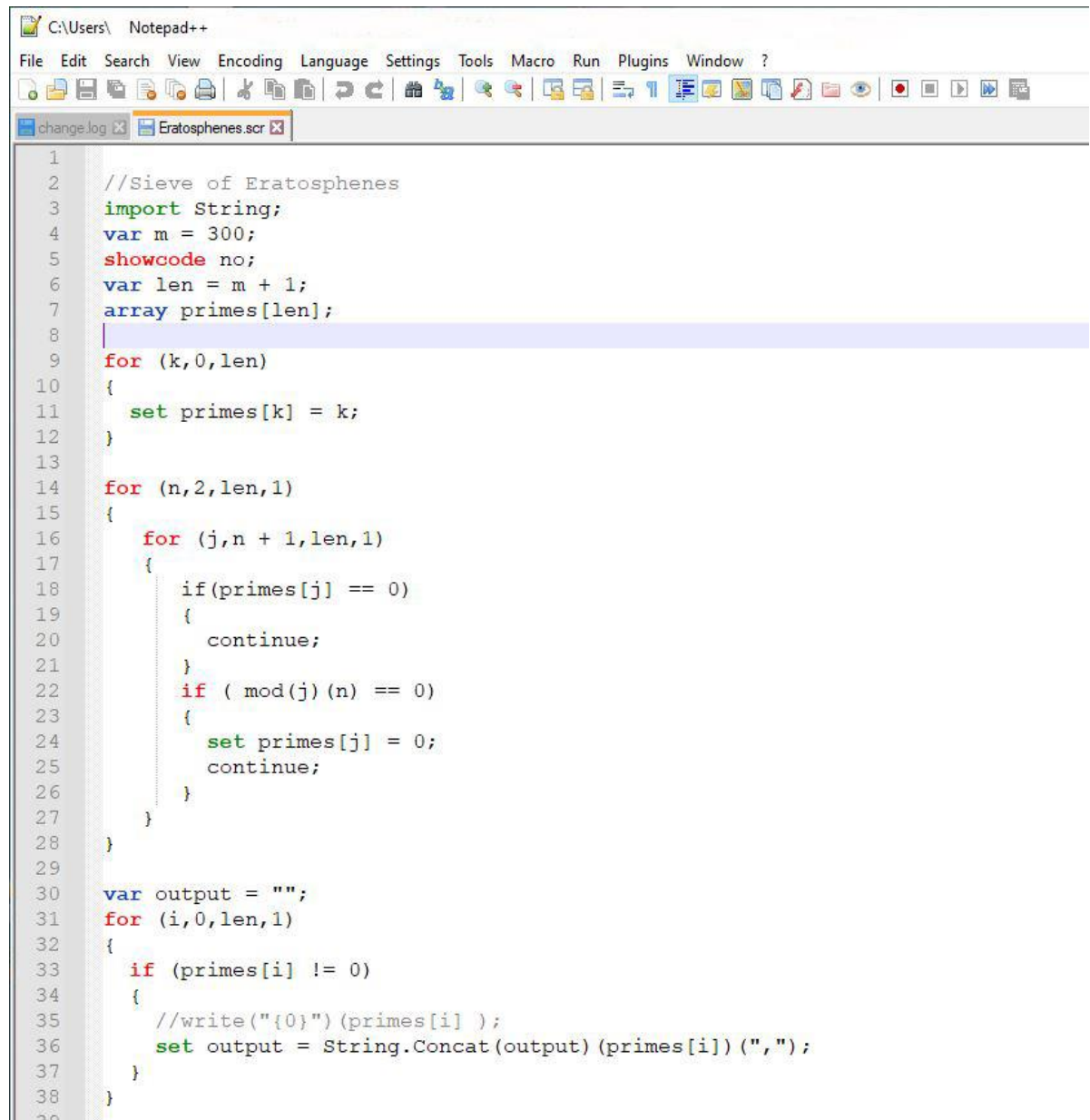
Special variables, constants and words

hidden variables: _main0, _main2,..., _f0,_f1,...

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

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

Highlighting keywords support is provided in Notepad++ :

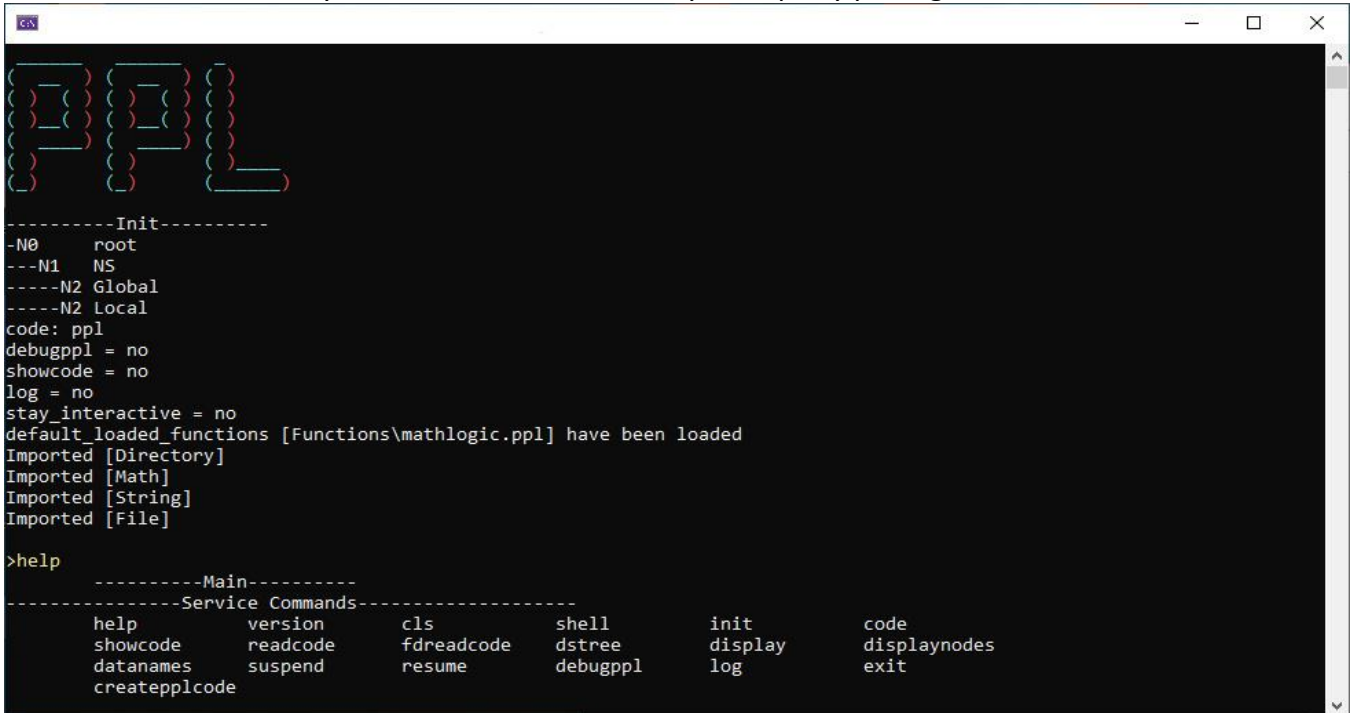


The image shows a Notepad++ window with the title bar 'C:\Users\ Notepad++'. The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, Plugins, Window, and ?. The toolbar contains various icons for file operations and editing. The active tab is 'Eratosphenes.scr'. The code is as follows:

```
1
2 //Sieve of Eratosphenes
3 import String;
4 var m = 300;
5 showcode no;
6 var len = m + 1;
7 array primes[len];
8
9 for (k,0,len)
10 {
11     set primes[k] = k;
12 }
13
14 for (n,2,len,1)
15 {
16     for (j,n + 1,len,1)
17     {
18         if(primes[j] == 0)
19         {
20             continue;
21         }
22         if ( mod(j)(n) == 0)
23         {
24             set primes[j] = 0;
25             continue;
26         }
27     }
28 }
29
30 var output = "";
31 for (i,0,len,1)
32 {
33     if (primes[i] != 0)
34     {
35         //write("{0}") (primes[i] );
36         set output = String.Concat(output) (primes[i]) (" ");
37     }
38 }
39
```


CPPL utility

Command-line REPL utility **cppl.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.



```
CPPL

-----Init-----
-N0      root
---N1     NS
-----N2 Global
-----N2 Local
code: ppl
debugppl = no
showcode = no
log = no
stay_interactive = no
default_loaded_functions [Functions\mathlogic.ppl] have been loaded
Imported [Directory]
Imported [Math]
Imported [String]
Imported [File]

>help
-----Main-----
-----Service Commands-----
      help      version      cls      shell      init      code
      showcode   readcode   fdreadcode  dstree    display  displaynodes
      datanames  suspend    resume     debugppl  log       exit
      createpplcode
```

These are following subdirectories and files used to work with cppl.exe

Subdirectories:

\Data

\Examples

\Functions

Files:

Aliases.data

Configuration.data

CPPL.exe, OFD.exe, Createulc.exe

There are 2 operating modes in accordance with cppl.exe arguments:

1. NonInteractive mode

Execute program in file with extension scr or ppl.

cppl.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.

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

Example:

File example.scr

```
var %1% = 0;
%2%;

cppl.exe example.scr x display;
```

2.Interactive mode

cppl.exe

Command input from standard input stream.

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

Examples:

```
>? Display;
      display [root|NS|Aliases]
>? d;
      display [root|NS|Aliases]
```

Prompt ">" appears on Screen before each command.

Examples:

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

In addition to commands required to work with scr/ppl programs, cppl.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 parameters:

help (?), **import**, **readcode** (rc), **showcode**, **createnode**, **isexist**, **display** (d), **del**, **code**, **getvalue**(get), **getname**, **debugppl**.

WPPL utility

WPPL.exe is also a PPL interpreter, its functionalities are liked cpp.exe. WPPL.exe is WPF Application, runs in interactive mode only.

The screenshot shows the WPPL application window. The top menu bar includes 'File' and 'Data'. Below it is a toolbar with buttons: 'init', 'cls', 'display', 'importlist', 'ppl' (with a dropdown arrow), 'showcode' (checked), 'debugppl' (unchecked), 'log' (unchecked), and 'help'. The main area is a blue console window. The top part shows command input: `>showcode;`, `>version`, `version 1.0.6`, `>help`, and `>`. The bottom part shows the help output, which is divided into sections: 'Init', 'Main', 'Service Commands', 'Special Commands', 'Nodes', and 'Variables and Arrays'.

```

-----Init-----
-N0   root
--N1  NS
----N2 Global
----N2 Local
code: ppl
debugppl = no
log = no
stay_interactive = no
Imported [Directory]
Imported [Math]
ShowCode=yes


-----Main-----
-----Service Commands-----
help      version  cls      shell      init      code
showcode  readcode  fdreadcode  dstree    display   createppl
datanames suspend  resume      debugppl  log        exit
-----Special Commands-----
import    importlist eval      length    exist      del
getbyname getbyvalue set       getvalue  getname
-----Nodes-----
createnode copynode
-----Variables and Arrays-----
var        const    array    realloc
  
```

Top part is used as input any PPL commands, down part is for results presentation. Also service commands may be performed by menu and wpf controls over top part. The following dialogs are used to perform service commands:

Clicking on **CreatePPL in Menu** opens the following dialog:


The screenshot shows the 'CreatePPLDlg' dialog box. It has two text input fields. The first field is labeled 'SCR filename:' and contains the path 'PL\PPL\PPL\WPPL\bin\Debug\netcoreapp3.1\Examples\rc.scr'. To its right is a 'Browse' button. The second field is labeled 'PPL filename:' and contains the path 'C:\Users\ok21\MyProjects.Work\C#2\PPL\PPL\PPL\WPPL\bin\'. To its right is a 'CreatePPL' button.

Clicking on **ReadCode in Menu** opens the following dialog:



The ReadCodeDlg dialog box has a title bar with a close button (X). It contains two text input fields: 'Filename:' with the text '2\PPL\PPL\PPL\WPPL\bin\Debug\netcoreapp3.1\Examples\rc.scr' and 'args:' which is empty. To the right of the 'Filename:' field is a 'Browse' button. To the right of the 'args:' field is a 'ReadCode' button.

Clicking on **ReadData in Menu** opens the following dialog:



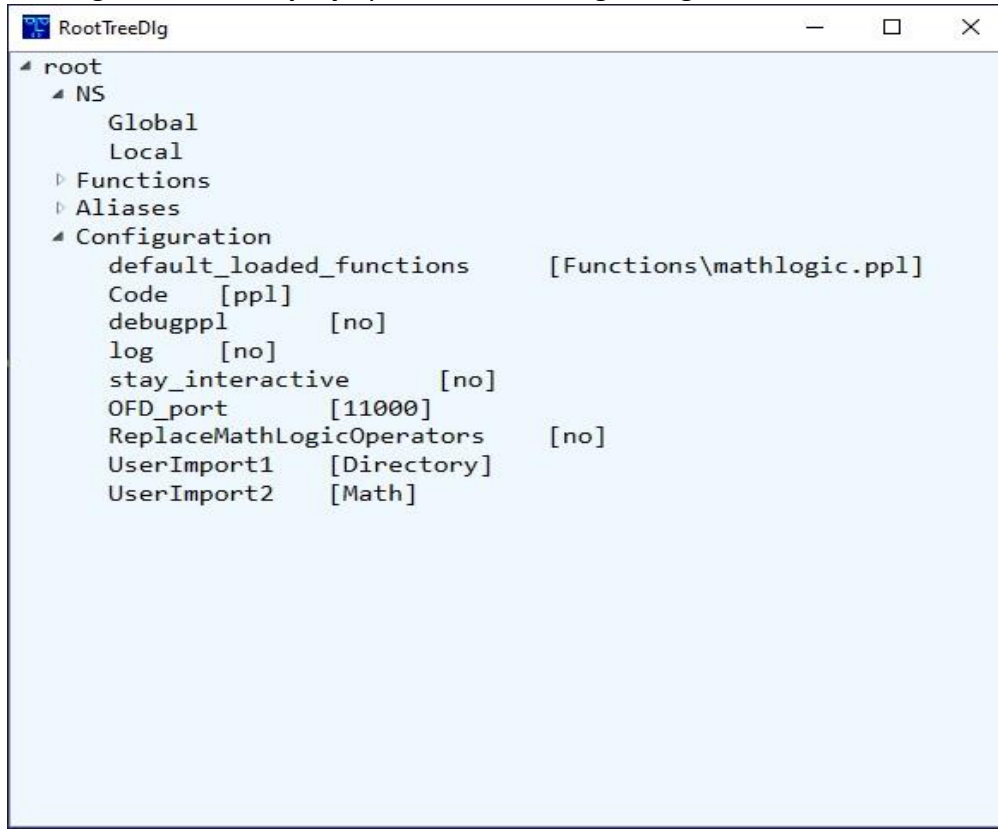
The ReadDataDlg dialog box has a title bar with minimize, maximize, and close buttons. It contains two text input fields: 'Filename:' with the text '\PPL\PPL\PPL\WPPL\bin\Debug\netcoreapp3.1\Data\Colors.data' and 'Node:' which is empty. To the right of the 'Filename:' field is a 'Browse' button. To the right of the 'Node:' field is a 'ReadData' button.

Clicking on **SaveData in Menu** opens the following dialog:



The SaveDataDlg dialog box has a title bar with a close button (X). It contains two text input fields: 'Filename:' with the text 'Data\' and a dropdown menu showing '.data', and 'Node:' with the text 'Global'. To the right of the 'Node:' field is a 'SaveData' button.

Clicking on button **display** opens the following dialog:



History of commands is supported by buttons **PgUp** and **Pgdn**.

WPPL.exe is an example of using the PPL API, which can be easily used in the user's application.

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** | ? [<library name>] | keyword

Examples:

```
>? Code;  
    Sets mode for Console input or displays on Screen  
code ppl | scr  
>? "display"    // only display in quotes  
    display | d [root|NS|Aliases|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 are saved and can be displayed by commands **write** or by **debugppl yes**

Format: **shell (command with parameters[#command with parameters])**

Examples:

```
>write(shell (cd:\));  
>debugppl 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

Reads file with code in format scr or ppl.

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. The file being called can also include readcode commands.

Example

```
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% = 0;
%2%;
>rc example.scr x display;
3. reading script with rc 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;
Results:
main script
first script
third script
return from first script
second script
return from main script
```

fdreadcode

Like readcode with using OpenFileDialog to select file. This command loads ofd.exe and sets connection with **cppl.exe** via UDP protocol, port defined in file **Configuration.data** as **OFD_port**.

Format: **fdreadcode|fdrc**

createpplcode

Creates file in format ppl from file in format scr.

Format: **createpplcode|cpc (file.scr)(file.ppl)**

Examples

```
>createppl (ttt.scr) (ttt.ppl);
```

display

Displays nodes(N) and leaves(L), alias – d.

Format:

display | d [root | Global | Functions | Aliases | Configuration | Local | path_node.name]

default: Global

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

Displays nodes(N) only, alias – d.

Format:

displaynodes | dn [root | Global | Functions | Aliases | Configuration | Local | path_node.name]

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 of the latest operation with nodes(N) and leaves(L) of code.
dstree displays all function tree within function.

Format: **dstree()** | **dstree**

Examples:

```
>code scr;
>var x = (2+3)*3;
result = 15
>dstree;
-----Syntax Tree-----
-N0      root      [0]
---N1    *
-----N2 +
-----L0      2
-----L1      3
-----L0 3
>d;
Warning: if code = 'scr' it is added ';' to end of command
-N1      NS
---N2    Global
-----L1 x      [15]
---N2    Local
```

datanames

Displays contents of [DNS](#). For Local use this command from function.

Format: **datanames [Local]**

Examples:

```
>var(x) ;
>createnode Node1;
>array(Node1.arr[5]);
>datanames;
-----Global_dns-----
      _main0   var
      _main1   var
      _main2   var
      _main3   var
      _main4   var
      _main5   var
      _main6   var
      _main7   var
      _main8   var
      _main9   var
      empty    const
      x
Node1   arr
>datanames Local;    // for using in functions
-----Local_dns-----
```

suspend and resume

Stops script to perform manually one or several commands,
for exit – **resume**

Format: **suspend**

Examples:

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

debugppl

Displays information about creation and deletion variables, results operations and duration or displays debugppl value on Screen.

Format: **debugppl [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
```

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 Cpp.exe (**exit**) or exit from script (**exit()**).

Special Commands

import

Loads Library, name of Library is name of DLL.

Format: **import <Library name>**

Examples:

```
>import Math;
```

importlist

Displays list of loaded Libraries

Format: **importlist**

Examples:

```
>importlist;
```

```
Main
```

```
Math
```

eval

Performs string in format ppl.

Format: **eval <ppl expression>[<result>]**

result:=var_name to save result

Examples

Ex. 1

```
> var (sum);
>var (x["+ (2) (3)"]);
>d;
-N1      NS
---N2    Global
-----L0 sum
-----L1 x      ["+ (2) (3)"]
---N2    Local
>eval(x)(sum)
>d
-N1      NS
---N2    Global
-----L0 sum      [5]
-----L1 x      ["+ (2) (3)"]
---N2    Local
```

Ex. 2

```
array y[] = {"+ (3) (5)", "+ (4) (6)"};
for(j,0,length(y))
{
    eval(y[j]);
};
```

Ex. 3

```
var(x);
eval("+ (1) (2)")(x);
write("{0} = {1}")(getname(x))(getvalue());
x = 3
```

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]);
>length(y);
result = 3
>var (x["Hello!"]);
>length(x);
result = 6
```

isexist

Determines whether var, array or storage with specified name exists or not, returns "True" or "False".

Format: **isexist(name) | isexist name**

name:= [NS.][namespace.][node.]name

Example:

```
>var (x);  
>isexist x;  
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) |**

Isdigits var name | member of array or storage | literal

Example:

```
>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) |**

Isinteger var name | member of array or storage | literal

Example:

```
>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) |**

Isalldigits member of array or storage

Example:

```
>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) |**
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)) |
Iseven var name | member of array or storage | literal)

Isodd(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**

To delete all Global contents: **del all**

fullname:= node path.name

node path:= node path | node

Example:

```
> createnode Node1;  
> var(Node1.x) ;  
> del Node1.x;  
  
> createnode Functions.Geo  
> del Functions.Geo
```

For re-run script, that creates array:

```
> if(==(isexist(y)) (True))  
(  
    (del y)  
);  
> array(y[5]) ;
```

calc

Calculates infix notation math. expression and writes result on screen.

Format: **calc math.expression**

Example:

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


getbykey

Gets value from array by name.

Format: **getbykey(name array)(name element)**

Example:

```
See example in readdata
>getbykey(Colors) (Black) ;
result = 0
```

getbyvalue

Gets name from array by value.

Format: **getbyvalue(name array)(value element)**

Example:

```
See example in readdata
>getbyvalue(Colors) (0) ;
result = Black
```

set

Sets value for variable and array element

Formats for **ppl code**:

1. **set(var)(value)**
 2. **set(array[index]) (value | ppl expression)**
 3. **set(array[index])(name)(value | ppl expression)**
- index:=value | ppl expression**

In format 3 command set is similar to KeyValueStruct and it is possible to use commands getbykey and getbyvalue for these array elements.

Formats for **scr code**:

1. **set var | array[index] = value | scr expression**
 2. **set var | array[index] = name,value | scr expression**
- index:=value | scr expression**

Command **set** checks whether index is out of bounds. For setting name and value command **set** checks whether name already exists in array.

Examples:

```
>code ppl:
>var (x) ;
>set(x) (0) ;

>array(y[3]) ;
>set(y[0]) (0) ;
>set(y[0]) (+ (1) (2)) ;
>set(y[1]) (one) (1) ;
>set(y[2]) (two) (2) ;
>set(y[0]) (y[1]) ;
```

```

>code scr:
>var x;
>set x = 1;
  >array[3];
>set y[x + 1] = 2 + 3;
>set y[0] = nul, 0;
>d;
-N1      NS
---N2    Global
-----N3 y      [Array 3]
-----L0      nul      [0]
-----L1      #
-----L2      #          [5]
-----L0 x      [1]
---N2      Local
>set y[0] = y[1];

```

getvalue

Returns value of single var|const or array element.

Format: **getvalue (var_name) | getvalue(array_name[index]) |**
getvalue var_name | getvalue array_name[index]

index:= value|ppl expression

Alias: get

Examples:

```

>array (y[3]) (999);
>write("getvalue(y[0]) = {0}") (getvalue (y[0]));
result: getvalue (y[0]) = 999

```

getname

Returns name of single var|const or array as string.

Format: **getname (name) | getname name**

Examples:

```

>var (x[ppl]);
>write("{0} = {1}") (getname x) (getvalue x);
x = ppl

```

Using operators getvalue and getname in function see examples\scr\func4.scr

Nodes and Leaves

createnode

Creates node in path, default path is "**Global**", it is possible to create nodes in Tree Global or in Tree Functions

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:

```
>rc tests\struct\personglob.ppl
>createnode Person;
>var(Person.Name);
>var(Person.Family);
>var(Person.DOB);
>var(Person.Gender);
>array(Person.cars[3]);
>copynode(Person)(Team);
Info [FuncCopyNode] Global node [Team] is created
>set(Team.Name)(Oscar);
>set(Team.Family)(Ko);
>set(Team.DOB)(2050);
>set(Team.Gender)(m);
>set(Team.cars[0])(Juke)(Nissan);
>set(Team.cars[1])(Qashqai)(Nissan);
>d
-N1      NS
---N2    Global
-----N3 Person  [Node]
-----L0      Name
-----L1      Family
-----L2      DOB
-----L3      Gender
```

```

-----N4      cars      [Array 3]
-----L0      #
-----L1      #
-----L2      #
-----N3 Team  [Person]
-----L0      Name      [Oscar]
-----L1      Family    [Ko]
-----L2      DOB       [2050]
-----L3      Gender    [m]
-----N1      cars      [Array 3]
-----L0      Juke       [Nissan]
-----L1      Qashqai    [Nissan]
-----L2      #
---N2      Local

```

getnodes

Creates ppl_array with fullnames of nodes till defined nesting. Processing results of commands getnodes and getleaves allows to find required information in hierarchical data dstructure.

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
---L12   #      Data12]
---L12   #      [Staff.Operations management.Managers.Personal
---L13   #      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:

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");
for(i,0,length(persons))
{
  if (isexist(property) == True)
  {
    del property;
  }
  getleaves(persons[i]) ("property");
  for(j,0,length(property))
  {
    set tmp = getname(property[j].name);
    if (tmp == "Name")
    {
      set name = property[j].value;
    }
    if (tmp == "Salary")
    {
      set salary = property[j].value;
      if (salary == 2000)
      {
        write("Name = {0,-15}\tSalary = {1}") (name) (salary);
      }
    }
  }
}
>rc data\mng2.scr
result:
Name = Charlotte      Salary = 2000
```

Name = Olivia	Salary = 2000
Name = Felix	Salary = 2000
Name = James	Salary = 2000
Name = Sophia	Salary = 2000

Arithmetic operators

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

and their aliases:

sum, sub, mul, div, pow, mod (see Aliases.data).

To use these aliases set **yes** for **ReplaceMathLogicOperators** in Configuration.data.

By default **ReplaceMathLogicOperators** = **no** to decrease processing time.

These are binary operators.

Do not confuse with functions names in **Mathlogic.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);
```

Example with Aliases:

```
> var z = (2 plus 3) mul 4;
```

If index for accessing array members defined as expression with arithmetic operators hidden variables are used:

```
>array (y[3]);
>var (x[1]);
>set (_main0) (+ (x) (1)); // _main0 - hidden var
>set (y[_main0]) (10); // set (y[2]) (10);
>d;
-N1      NS
---N2    Global
-----N3 y      [Array 3]
-----L0      #
-----L1      #
-----L2      #      [10]
-----L11     x      [1]
---N2      Local
```

Logical operators

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

xor only for ppl mode

and their aliases:

lt, le, gt, ge, eq, ne, and, or (see Aliases.data).

To use these aliases set **yes** for **ReplaceMathLogicOperators** in Configuration.data.

By default **ReplaceMathLogicOperators** = **no** to decrease processing time.

These are binary operators.

Do not confuse with functions names in **Mathlogic.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)
```

Variables and Arrays

var

Creates a single variable in Global or in Local function scope. It will be error if name already exists.

Format **ppl**:

var (name) | (name[init value]) [(name) | (name[init value])]...

name:= [node path]name

node path:= node. | node

init value:= value | ppl expression

ppl expression:=value | prefix notation expression

Examples:

```
>var (greeting["Hello"]);  
>var (x);  
>var (x) (y[1]) (z[+(2) (3)]);
```

Format **scr**:

var name | name = init value

name:= node path.name

node path:= node. | node

init value:= value | scr expression

scr expression:= value | infix notation expression

Examples:

```
>code scr;  
>var greeting = "Hello";  
>var x;  
>var z = 2 + 3;
```

const

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

Format **ppl**:

const (name[init value]) [(name[init value])]...

name:= [node.]name

init value:= value | ppl expression

ppl expression:=value | prefix notation expression

Example:

```
>const (x[0]) (y[1]) (z[+(2) (3)]);
```

Format **scr**:

const name = init value

name:= node path.name

node path:= node. | node
init value:= value | scr expression
scr expression:= value | infix notation expression
Examples:

```
>createnode N1;  
>code scr;  
>const greeting = "Hello";  
>const radian = 180 / Math.PI();  
result = 57.29577951308232
```

array

Creates single-dimensional array in Global or in Local function scope. It will be error if name already exists.

Format **ppl**:

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

Examples:

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

realloc

Changes length of array, all elements are saved in changed array.

Format: **realloc(array name)(new length)**

Examples:

```
>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
```

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

```
>rc examples\scr\testswrite.scr
>>storage(s)(3)(4)(5);
>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);
```

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
```

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.

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).

Format **ppl**:

Format: **storage (name)[(length dim1)][(length dim2)]...**

name:= node path.name

node path:= node. | node

length:= value | ppl expression

ppl expression:=value | prefix notation expression

Examples:

```
(1)
>storage (x); - variable
>d;
-N1      NS
---N1    Global
-----L0 x
```

```
(2)
>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
>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
-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

sinit

Init storage

Format: **sinit (name)(init value)**

init value:= value | ppl expression

Examples:

```
>storage (x) (2) (3) ;
>sinit (x) (0) ;
-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]
```

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]
>sget(stor) (0) (0) ;
```

sset

Sets value for element in storage

Format: **sset (name)(index1)(index2)(value)**

Examples:

```
>sset(stor) (0) ; // set value of single-variable = 0
```

```
// set value of single-dimensional array,stor[0]= 1
```

```
>sset(stor) (0) (1) ;
```

```
// set value of two-dimensional array,stor[0][0] = 1
```

```
>sset(stor) (0) (0) (1) ;
```

swrite

Displays elements values of storage

Format: **swrite(name) [(max_window_width = 100)]**

Examples:

```
>storage (x) (3) ;
```

```
>sinit(x) (0) ;
```

```
>swrite(x) (50) ;
```

	0	1	2
-----NS.Global-----			
[x]	0	0	0

```
>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


```

>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:

```

>sinfo(y);
result = Storage 1 5 // single-dimensional array length 5

```

After using realloc for storage Row it will be written:

```

>storage(s)(3)(5);
>realloc(s)(0)(10);
>sinfo(s);
result = Storage 2 reallocated

```

ssetrow

Sets value for elements of the lowest level.

Format: **ssetrow(name)(ind1)(ind2)(indN)... (elem1)(elem2)(elemM)...**

Examples:

```

// N = 2, M=3
>storage(y) (2) (3) ;
>ssetrow(y) (0) (1) (2) (3) ;
>ssetrow(y) (1) (4) (5) ;
>ssetrow(y) (1) (4) (5) (6) (7) (8) ;
Error: [FuncStorageSetRow] wrong format, number of parameters
[7] > [5]

```

```
>swrite(y);
-----NS.Global.y-----
[0]      1                      2                      3
[1]      4                      5
```

Backup & Recovery

savedata

Saves data from node to file with extension **.data** or **.json**

If node is root, all root contents will be saved.

Format: **savedata(filename.data | filename.json)[(node)]**

Default node: **NS.Global**

Examples:

1.

```
>createnode Node1;
>var(Node1.x[0])(Node1.y[1]);
>savedata (Examples\ppl\f3.json) (Global.Node1);
```

2.

```
>savedata(Data\Colors1.data) (Colors);
```

3.

```
>rc examples\ppl\json.ppl;
>createnode Node1;
>var(Node1.x[0])(Node1.y[1])(a[2])(b[3]);
>array(y[3])(0);
>set(y[0])(one)(1);
>set(y[1])(two)(2);
>set(y[2])(three)(3);
>var(c[true])(d[family]);
>array(cars)(Ford)(Nissan)(Renault);
>savedata (Examples\ppl\f3.json);
>shell(type f3.json)
{
  "Global":
  {
    "Node1": "Node"
    {
      "x":0,
      "y":1
    },
    "a":2,
    "b":3,
    "y":
    [
      1,
      2,
      3
    ],
```

```
    "c": "true",  
    "d": "family",  
    "cars":  
      [  
        "Ford",  
        "Nissan",  
        "Renault"  
      ]  
  }  
}
```

readdata

Reads data from file with extension **.data** to Aliases, Configuration and NS.Global node,**not to Local**.

Format: **readdata(filename.data)[(node)]**

Default node: **NS.Global**

Examples:

```
>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]
```

Control Flow

if, else

The meaning of the block "if-else" does not differ from the generally accepted.
About using statements terminator ";" in if see [Base Concepts](#).

Format **ppl**:

```
if (expression)
(
    (statement)
    (statement)
    [ (else
        (
            (statement)
            (statement)
        )
    ) ]
);
```

Here expression in prefix notation.

Statement in ppl mode.

Format **scr**:

```
if (expression)
{
    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))
    (write(TRUE))
    else
    (
        (write(false))
        (write(FALSE))
    )
)
);
true
TRUE
>write(end);
end
```

Example scr mode:

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

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.
Statement in ppl mode.

Format scr:

```
switch(expression)
{
    case <value>:
    case <value>:
        statement;
    break;
    case <value>:
        statement;
    break;
    default:
        statement;
    break;
}
```

Here expression in infix notation.
Statement in ppl or 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:

```
loop (iteration var) (begin) (end) [(increment)] or
loop () //infinity loop
(do
  (
    (statement)
    (statement)
    ...
  )
)
```

begin:= value|ppl expression

end:= value|ppl expression

increment:= value|ppl expression

By default increment = 1. Increment may positive or negative.

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](#)

for

Iteration block for scr mode only.

About using statements terminator ";" in for see [Base Concepts](#).

Format:

`for(iteration var, begin, end [, increment]) or`

`for() // infinity for`

```
{  
    statement;  
    statement;  
    ...
```

```
}
```

By default increment = 1. Increment may positive or negative.

Statement in ppl or scr mode.

Examples:

```
var x;  
for(i, 0, 10, 1)  
{  
    set x = i * 2;           // scr statement  
    write("x = {0}") (x);    // ppl statement  
}
```

Example with Infinity for:

```
showcode no;
// Euclidean algorithm
function gcd(x0,y0,z)
{
    // x0, y0 may be digital constants,not variables
    var x = x0;
    var y = y0;
    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)
        {
            set x = x - y; // or Sub(x) (x) (y) ;
        }
        if (x < y)
        {
            set y = y - x; // or Sub(y) (y) (x) ;
        }
        if (x == y)
        {
            set z = x;
            return;
        }
    }
}
var z = 0;
gcd(14144) (26163) (z) ;
write(gcd = {0}) (z) ;
del all;

//result
gcd = 17;
```

break

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

Example:

```
for(i, 0, 4, 1)
{
    if (i == 2)
    {
        write("true i = {0}") (i);
        break;
    }
};
```

continue

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

Example:

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

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 wrote in red color in cppl.exe or in wppl.exe.

Format:

write(value) | write(c# format)(value)(value)...

value:=value | ppl expression

Example:

```
>var (x[0]);
>write(x);
>write("{0}{1}") ("x=") (x); // like c# write("{0}{1}") ("x=",x);
//quote in string
>write("ppl\tlanguage"); // 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
```

writearray

Writes array contents to the standard output stream.

Format:

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

Examples:

1.

```
>rc examples\scr\writearray.scr
>function
(
  func  ()
  (
    (array (y ) (1) (2) (3))
    (writearray(y))
    (writearray(N.Y))
  )
);
>createnode N;
>array (N.Y ) (10) (20) (30);
>writearray(N.Y);
-----Array N.Y-----
[0]      10
[1]      20
[2]      30
>func();
-----Array y-----
[0]       1
[1]       2
[2]       3
-----Array N.Y-----
[0]      10
[1]      20
[2]      30
```

2.

```
>storage(x) (2) (3);
>sinit(x) (0);
// write the bottommost arrays in storage
>writearray(x.0.Row);
>writearray(x.1.Row);
```

3.

```
>storage(s) (2) (3);
>sinit(s) (0);
>d;
-N1      NS
---N2    Global
-----N3 s      [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]
---N2      Local
>set(s.0.Row[0]) (zero) (00)
>set(s.0.Row[1]) (one) (1)
>set(s.0.Row[2]) (two) (2)
>writearray(s.0.Row)
[0]      zero      00
[1]      one       1
[2]      two       2
```

readline

Reads the next line of characters from the standard input stream. Result will be passed to calling operator.

Format: **readline()**

Examples:

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


Functions

Standard functions library **mathlogic.ppl**, defined in file **Configuration.data** as **default_loaded_functions**, loads automatically or reloads when command **init** executes.

It includes the following functions:

```
Sum (result, n1, n2)
Sub (result, n1, n2)
Mult (result, n1, n2)
Div (result, n1, n2)
Pow (result, n1, n2)
PlusPlus (result)    // like c#: ++(var)
MinusMinus (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.

Examples:

```
>var (x[5]) ;
>Sum(x) (x) (2) ;    // =  set (x) (+ (x) (2) ) ;
>var (c[1]) ;
>PlusPlus (c)
>d;
-N1      NS
---N2    Global
-----L0 c      [2]
---N2    Local
>Minus2 (c)
>d;
-N1      NS
---N2    Global
-----L0 c      [1]
---N2    Local
```

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

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

function

Functions must be declared before called.

Functions are saved in Tree **Functions** or its nodes and in Tree **Global** or its nodes.

To pass value of array member to function without changes

use **getvalue(array_name[index])**.

To pass value of storage member to function without changes

use **sget (storage_name(ind1)(ind2)...)** .

To pass value of array member or storage member to function, where it will be changed, use hidden variables.

How to use hidden variables to call functions see [Hidden variables](#).

About using statements terminator ";" in function see [Base Concepts](#).

Format **ppl**:

```
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::= see [Identifiers and DNS](#)

Format **scr**:

```
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::= see [Identifiers and DNS](#)

Examples:

ppl mode

1.

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

2. passing array member to function unchanged

```
function
(
  func(arg1)
  (
    (write(arg1))
  )
);
var(x["Hello"]);
func(x);
array(y) ("Tom") ("Johny");
func(getvalue(y[0]));
```

3. passing array member to function for changing with hidden vars

```
function
(
  func(arg)
  (
    (set(arg) ("Hello!"))
  )
);
>array(y[5]);
>set(_main0) (y[0]);    // y[0].value => _main0
>func(_main0);
>set(y[0]) (_main0);    // changed _main0 => y[0].value
>write(y[0]);
```

scr mode (see examples\scr\func.scr)

```
function sum_arr (n,m)
{
  var tmp;
  for(i,0,length(n),1)
  {
    set tmp = n[i] + m[i];
    write( "[0] = {1}" )(i)( tmp);
  }
}
array x[] = {1,2,3,4,5};
array y[] = {2,3,4,5,6};
sum_arr(x)(y);
result:
[0] = 3
[1] = 5
[2] = 7
[3] = 9
[4] = 11
```

Functions can be declared as parameters
(see examples\scr\func3.scr).

Example:

```
4. function a(str)
{
  write(str);
}
function b(f)
{
  write("function b ...");
  eval(f);
}
function c(str)
{
  write(str);
}
array d[] = {"a(function a)","c(function c)"};
for (i,0,length(d))
{
  b(getvalue(d[i]));
}
result:
function b ...
function a
function b ...
function c
```

In the following example (see examples\scr\func4.scr) parameter index has default value = 0 and this parameter is omitted when the function is called (func(x), func(y)).

```
5.
function func (name, index[0])
{
  if ( String.Contains(name) ("Array") == "True")
  {
    var tmp;
    set tmp = getvalue(name[index]);
    write("{0} = {1}") (getname(name)) (getvalue(name[index]));
  }
  else
  {
    write("{0} = {1}") (getname(name)) (getvalue(name));
  }
}
}
import String;
array y[] = {1,2,3,4};
var x = 100;
func(x);
func(y);      // = func(y) (0);
func(y) (1);
//=====result=====
>rc examples\scr\func4.scr
>import String;
>array (y) (1) (2) (3) (4);
>var (x[100]);
>func(x);
x = 100
>func(y);
y = 1
>func(y) (1);
y = 2
```

Addition node before function name:

```
6. Call function from node of Tree Global
createnode New;
function
(
  New.func(name)      // public
  (
    (write(name))
  )
);
function
(
  New._hfunc(name)    // private
  (
    (write(name))
  )
);
New.func("Greetings!");
```

7. Call function from node of Tree Functions

```
createnode Functions.New;  
function  
(  
  Functions.New.func(name)    // public  
  (  
    (write(name))  
    (Functions.New._hfunc(name))  
  )  
);  
function  
(  
  Functions.New._hfunc(name)  // private  
  (  
    (write(name))  
  )  
);  
  
Functions.New.func("Greetings!");
```

8. Call function from Tree Function

```
function  
(  
  Sum(result) (n1) (n2)  
  (  
    (set(result) (+ (n1) (n2)))  
  )  
);  
var (x);  
Sum(x) (1) (2);
```

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

return

Exit from function or from script

.

Example:

```
function f()
{
    for(i, 0, 5, 1)
    {
        write(i);
        if (i == 3)
        {
            return;
        }
    };
};
f();
write("end of script");
```

funclist

Displays function names and their parameters from node Functions

Format:

funclist

Example:

>funclist;

```
-----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)
```

Additional functionalities

The following below-mentioned additional DLLs with C# functionalities are added and this list will be expanded.

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	

To get short help of 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 remainder in an output parameter: **Math.DivRem(Int64 n1)(Int64 n2)**

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
Insert	ToLower
Remove	ToUpper
Split	Trim
SplitCsv	

To get short help of 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)(count)

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)

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

String.ToCharArray(string)(node_of PPL_chars_array)

node_of PPL_chars_array is string in quotes or value of variable.

Example:

```
>Import String
>String.ToCharArray("qwerty") ("chars") ;
>writearray(chars)
-----Array chars-----
[0]      q
[1]      w
[2]      e
[3]      r
[4]      t
[5]      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 a 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.Split(string)("ppl_array_separators")("ppl_array_result") or

String.Split(string)("var_separator")("ppl_array_result")

Use comma instead of ','

 space instead of ' '

 tab instead of '\t'

Example:

> var text;

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

Returns a string array that contains the substrings in this instance that are delimited by separator of a specified string var,

If substring surrounded by quotes it may contain separator (see example\lib\splitcsv.scr):

String.SplitCsv(string)("var_separator")("ppl_array_result")

Use comma instead of ','

 space instead of ' '

 tab instead of '\t'

Do not use whitespace as separator:

Example:

```
>var text2;  
>set text2 = File.ReadAllText("Data\test2.csv");  
>String.SplitCsv(text2) ("comma") ("split_text2");
```

Returns string from File.ReadAllText without EndOfLine: **DeleteEndOfLine(string)**

Example:

```
>var text2;  
>set text2 = File.ReadAllText("Data\test2.csv");  
> set text2 = DeleteEndOfLine(text2);
```

Directory

Methods:

- GetFiles**
- GetDirectories**
- SetCurrentDirectory**
- GetCurrentDirectory**
- GetParent**
- CreateDirectory**
- Exists**
- Delete**

To get short help of every method in Directory.DLL:

> **Directory.help(method name);**

Returns the names of files (including their paths) in the specified directory:

Directory.GetFiles("node of PPL array")("path")

node of PPL array is string in quotes or value of variable.

Example:

```
1.
Directory.GetFiles("files") ( "c:\" );
or
var (x["files"]);
Directory.GetFiles(x) (path) ;
2.
>rc examples\lib\WriteFilesInDir.scr
function WriteFilesInDirectory (array,dir)
{
    Directory.GetFiles(array) (dir) ;
    writearray(array) ;
}
WriteFilesInDirectory ("files") ("c:\");
Result:
-----Array files-----
[0]      c:\DumpStack.log.tmp
[1]      c:\hiberfil.sys
[2]      c:\pagefile.sys
[3]      c:\swapfile.sys
```

Returns the names of directories (including their paths) in the specified directory:

Directory.GetDirectories("node of PPL array")("path")

node of PPL array is string in quotes or value of variable.

Example:

```
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")**

There following collections are supported: **ArrayList, Queue, Stack, Dictionary.**

ArrayList

Methods:

Create	ToArray	Count
Write	Reverse	Get
Add	Remove	Set
Clear	Insert	
Contains	IndexOf	
AddArray	Sort	

To get short help of every method in ArrayList.DLL:

> ArrayList.help(method name);

Creates ArrayList object: **ArrayList.Create(name)**

It is possible to create ArrayList repeatedly, in this case previous data removed.

Writes all array_list_names or all elements from the specified array_list to the standard output stream:

ArrayList.Write() or **ArrayList.Write(arrlist name)**

Adds a string to the end of the ArrayList: **ArrayList.Add(arrlist name)(string)**

To add empty string use keyword **empty**:

```
>ArrayList.Create(ar)
>ArrayList.Add(ar) (empty)
```

Adds node of PPL array to the end of the ArrayList:

ArrayList.AddArray("PPL array")(arrlist name)

Name of PPL array is **string in quotes** or value of variable with value = name of PPL array .

Removes all elements from the ArrayList: **ArrayList.Clear(arrlist name)**

Determines whether an element is in the ArrayList, returns "True" or "False":

ArrayList.Contains(arrlist name)(string)

Copies all elements from arrlist to new PPL array:

ArrayList.ToArray(arrlist name)("PPL_array")

Name of PPL array is **string in quotes** or value of variable with value = name of PPL array .

Error: If PPL array exists.

Reverses the order of the elements in the ArrayList: **ArrayList.Reverse(arrlist name)**

Removes the first occurrence of a specific object from the ArrayList:

ArrayList.Remove(arrlist name)(string)

Inserts an element into the ArrayList at the specified index:

ArrayList.Insert(name)(index)(element)

To insert empty string use keyword **empty**.

Returns the zero-based index of the first occurrence of a value in the ArrayList:

ArrayList.IndexOf(arrlist name)(value)

Sorts the elements in the ArrayList: **ArrayList.Sort(arrlist name)**

Returns the number of elements actually contained in ArrayList: **ArrayList.Count(arrlist name)**

The following example includes all ArrayList methods:

```
>rc Examples\lib\ArrayList.scr
  >import ArrayList;
  >ArrayList.Create("all");
  >createnode Private;
  >array(Private.src) (ONE) (TWO) (THREE);
  >var(x["Private.src"]);
  >ArrayList.AddArray(x) (all);
  >ArrayList.Write(all);
  >ArrayList.Add(all) (empty);
  >ArrayList.Add(all) (2two);
  >ArrayList.Add(all) (3three);
  >ArrayList.Add(all) (1one);
  >write("====Added objects====");
  >ArrayList.Write(all);
  >ArrayList.Remove(all) (1one);
  >ArrayList.Remove(all) (1one); // // error: 1one does not exist
  >write("====Removed objects====");
  >ArrayList.Write(all);
  >ArrayList.Reverse(all);
  >write("====Reverse====");
  >ArrayList.Write(all);
  >write("ArrayList.Contains 1one" = {0}) (ArrayList.Contains(all)
  (1one));
  >ArrayList.Insert(all) (2) (4four);
  >write("ArrayList.Contains 4four" = {0}) (ArrayList.Contains(all)
  (4four));
  >ArrayList.IndexOf(all) (3three);
  >ArrayList.Sort(all);
  >write("====Sort====");
  >ArrayList.Write(all);
  >ArrayList.ToArray(all) ("Private.dst_arr");
  >ArrayList.Clear(all);
  >d;
```



```
Result:
Imported [ArrayList]
all
    ONE
    TWO
    THREE
=====Added objects=====
all
    ONE
    TWO
    THREE

    2two
    3three
    lone
Warning: [ArrayList.FuncRemove] element [lone] does not exist
=====Removed objects=====
all
    ONE
    TWO
    THREE

    2two
    3three
=====Reverse=====
all
    3three
    2two

    THREE
    TWO
    ONE
ArrayList.Contains lone" = False
ArrayList.Contains 4four" = True
=====Sort=====
all

    2two
    3three
    4four
    ONE
    THREE
    TWO
-N1  NS
---N2 Global
-----N3  Private  [Node]
-----N4  src      [Array 3]
-----L0  #        [ONE]
-----L1  #        [TWO]
-----L2  #        [THREE]
-----N4  dst_arr  [Array 7]
-----L0  #
```

```
-----L1      #      [2two]
-----L2      #      [3three]
-----L3      #      [4four]
-----L4      #      [ONE]
-----L5      #      [THREE]
-----L6      #      [TWO]
-----L11     x      ["Private.src"]
---N2 Local
```

Returns value of ArrayList member by index:

ArrayList.Get(arrlist_name)(index)

Set value of ArrayList member by index:

ArrayList.Set(arrlist_name)(index)(value)

Example:

```
>import ArrayList
Imported [ArrayList]
>ArrayList.Create(x)
>ArrayList.Add(x) (qqq)
>ArrayList.Add(x) (zzz)
>ArrayList.Get(x) (0)
result = qqq
>ArrayList.Set(x) (0) (aaa)
>ArrayList.Get(x) (0)
result = aaa
```

Queue

Methods:

Create	Peek
Count	Clear
Write	Contains
Enqueue	AddArray
Dequeue	ToArray

To get short help of every method in Queue.DLL:

Queue.help(method name)

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

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

Copies all elements from Queue to the new PPL array:

Queue.ToArray(queue name) ("PPL array")

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

Stack

Methods:

Create	Peek
Count	Clear
Write	Contains
Push	AddArray
Pop	ToArray

To get short help of every method in Stack.DLL:

>Stack.help(method name)

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

Copies all elements from Stack to the new PPL array:

Stack.ToArray(stack name) ("PPL array")

Examples:

```
>import Stack
Imported [Stack]
>Stack.Create(s)
>Stack.Push(s) (one)
>Stack.Push(s) (two)
>Stack.Push(s) (three)
>debugpp1 yes
>Stack.Pop(s)
result = three
>Stack.Pop(s)
result = two
>Stack.Pop(s)
result = one
>Stack.Pop(s)
result = empty
```

Examples of code with Stack methods in **examples\lib\Stacks.ppl**

Dictionary

Methods:

Create	ContainsKey
Count	ContainsValue
Add	Remove
Write	AddArray
Clear	ToArray

To get short help of every method in Dictionary.DLL:

>Dictionary.help(method name)

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

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)

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

Removes all keys and values 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)

Determines whether the Dictionary contains a specific value, returns **True** or **False**:

Dictionary.ContainsValue(dictionary name)(value)

Adds PPL array to the Dictionary: **Dictionary.AddArray("PPL array")(dictionary name)**

Copies all elements from Dictionary to new PPL array:

Dictionary.ToArray(dictionary name) ("PPL array")

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

Convert

Methods:

```
StringToInt32Array
StringToHexArray
HexToBin
BinToHex
IntToHex
HexToInt
IntToBin
BinToInt
```

To get short help of every method in Convert.DLL:

>Convert.help(method name);

String characters convert to int32 array:

Convert.StringToInt32Array(string)("Int32 array")

String characters convert to hex array:

Convert.StringToHexArray(string)("Hex 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

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

-----Array Hex-----

```
[0]    31
[1]    32
[2]    33
[3]    34
[4]    35
```

Examples:

```
>debugppl 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 of 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();
fwrite(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

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)**

Example:

```
>File.ReadAllLines("examples\lib\split.txt")("x")
>d
-N1  NS
---N2 Global
-----N3 x   [Array 2]
-----L0  #   [1,2,3,4,5,6,7,8,9,10,]
-----L1  #   [11,12,13,14,15,16,17,18,19,20]
```

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)**

Random

Methods:

Create	NextDouble
Next	NextInt64
NextBytes	NextSingle

Creates Random object: **Random.Create(name)[(Seed)]**

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

Returns a non-negative random integer that is less than the specified maximum:
Random.Next(random_name) (maxValue)

Returns a random integer that is within a specified range:
Random.Next(random_name) (minValue)(maxValue)

Creates and fills the elements of a specified ppl_array with random numbers:
Random.NextBytes(random name)(ppl_array)(size of ppl_array)

Returns a random floating-point number that is greater than or equal to 0.0, and less than 1.0:
Random.NextDouble(random name)

Returns a non-negative random integer: **Random.NextInt64(random name)**

Returns a non-negative random integer that is less than the specified maximum:
Random.NextInt64(random name)(maxValue)

Returns a random integer that is within a specified range:
Random.NextInt64(random name)(minValue)(maxValue)

Returns a random floating-point number that is greater than or equal to 0.0, and less than 1.0:
Random.NextSingle(random name)

Examples:

```
>debugppl yes;  
>import Random  
Imported [Random]  
>Random.Create(r)  
>Random.Next(r)(0)(10)  
Result = 2  
>Random.NextBytes(r)(x)(5)  
>writearray(x)  
-----Array x-----  
[0] # 5  
[1] # 121  
[2] # 226  
[3] # 108  
[4] # 61
```

Vector

For using with library MathNet.Numerics and others.

Methods:

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>

CreateDouble("vector_name")(length)

like as Vector.Create("vector_name")(length)(double)

Add("vector_name")("ppl_array")

Write("vector_name")

WriteNames()

Delete("vector_name")

DeleteAll()

Matrix

For using with library MathNet.Numerics and others.

Methods:

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>

CreateDouble("matrix_name")(rows)(columns)

like as Matrix.Create("matrix_name")(length)(double)

Add("matrix_name")("ppl_array")

Write("matrix_name")

WriteNames()

Delete("matrix_name")

DeleteAll()

Example:

```
>import Matrix;
>Matrix.CreateDouble("A")(3)(3);
>array r1[] = {1,2,3};
>array r2[] = {4,5,6};
>array r3[] = {7,8,9};
>Matrix.AddRow("A")("r1");
>Matrix.AddRow("A")("r2");
>Matrix.AddRow("A")("r3");
>Matrix.Write("A");
>Matrix.Delete("A");

>import Vector;
>array v[] = {1,2,3,4,5};
>Vector.CreateDouble("V")(5);
>Vector.Add("V")("v");
>Vector.Write("V");
>Vector.Delete("V");
```

MN_Numerics

For using MathNet.Numerics.dll

Methods:

```
Matrix("matrix_name")(rows)(columns) // like as Matrix.CreateDouble
Vector("matrix_name")(rows)(columns) // like as Vector.CreateDouble
AddRowToMatrix("matrix_name")("ppl_array") // like as Matrix.AddRow
AddColumnToMatrix("matrix_name")("ppl_array") // like as Matrix.AddColumn
AddDataToVector("vector_name")("ppl_array") // like as Vector.Add
```

Linear Equation Systems:

See detailed information: <https://numerics.mathdotnet.com/LinearEquations.html>

```
Solve("matrix_name")("vector_name")("ppl_array_result")
```

```
DeleteAll() // delete all matrix and name
```

```
DeleteMatrix("matrix_name")
```

```
DeleteVector("vector_name")
```

For operations with vectors and matrices It is possible to use methods from

MN_Numerics.Matrix and MN_Numerics.Vector

or from

Matrix and Vector, **but not together.**

Example: (see examples\mnn\lesrow2.scr)

```
//linear equation systems
// AX = B
// Creation rows as ppl_arrays
// Creation matrix in Matrix
// Creation vector in Vector
import Matrix;
import Vector;
import MN_Numerics;
Matrix.CreateDouble("A")(3)(3);
array row1[] = {3,2,-1};
array row2[] = {2,-2,4};
array row3[] = {-1,0.5,-1};
Matrix.AddRow("A")("row1"); // fill matrix
Matrix.AddRow("A")("row2");
Matrix.AddRow("A")("row3");

Vector.CreateDouble("B")(3);
array vector[] = {1,-2,0}; // create vector as ppl_array
Vector.Add("B")("vector"); // fill vector
array X[length(vector)] = 0; // create result as ppl_array
MN_Numerics.Solve("A")("B")("X");
writearray(X);
Matrix.Delete("A");
```



```
Vector.Delete("B");  
del all;          // remove all arrays in ppl  
results:  
-----Array X-----  
[0] #           1  
[1] #          -1.9999999999999996  
[2] #          -1.9999999999999993
```

Constants: to get list of constants from MathNet.Numerics:
>MN_Numerics.help();

Examples:

```
>debugppl yes  
>import MN_Numerics;  
>MN_Numerics.Pi()  
result = 3.141592653589793  
>MN_Numerics.E()  
result = 2.718281828459045
```

Structure of User's DLL

Directory Template is the example for creation user's DLL, see Template.cs.

Example:

```
>import Template
>Template.sum(1) (2)
result = 3

>Template.help
help
sum
>Template.help(sum)
    Returns sum of two double-precision floating-point numbers:
    Template.sum(double d1) (double d2)
```

Add in Project Dependencies the project **PPL**

Utility createulc.exe creates code for User's DLL .

createulc.exe <name user DLL> [path]

Example:

```
createulc.exe MyLib
=====see result here=====
using System;
using System.Collections.Generic;
namespace PPLNS
{
    public class MyLib : AbstractClass
    {
        // ppl & help_dict in Abstract Class
        //public PPL ppl;
        //Dictionary<string, string> help_dict = new Dictionary<string,
            string>();
        public MyLib(PPL ppl)
        {
            this.ppl = ppl;
        }
        //=====
        public void AddToKeywordDictionary()
        {
            keyword_dict = new Dictionary<string, PPL.OperatorDelegate>();
            keyword_dict.Add("help", FuncHelp);
            keyword_dict.Add("keyword", FuncKeyword);
            // add here other methods & their keywords
            //...

            // add here short help
            //help_dict.Add("keyword", "short help lines, divided by EndOfLine");
```

```
//...
try
{
    if (ppl.ImportList.ContainsKey("MyLib") == false)
    {
        foreach (KeyValuePair<string, PPL.OperatorDelegate> pair in keyword_dict)
            ppl.processing.keyword_dict.Add("MyLib." + pair.Key, pair.Value);
        ppl.ImportList.Add("MyLib", this);
    }
}
catch (Exception io)
{
}
}
//=====
public bool FuncKeyword(List<string> parameters, ref string result, Composite node = null)
{
    try
    {
        //...
    }
    catch (Exception ex)
    {
        ppl.print("Error: ...");
        return false;
    }
    return true;
}
}
}
```

Examples of code

See Examples\scr\Eratosphenes.scr
mode scr

```
//Sieve of Eratosphenes
import String;
var n = 100;
var len = n + 1;
array primes[len];

for(i,0,len)
{
    set primes[i] = i;
}
for(i,2,len,1)
{
    for(j,i + 1,len,1)
    {
        if(primes[j] == 0)
        {
            continue;
        }

        if ( mod(j)(i) == 0)
        {
            set primes[j] = 0;
            continue;
        }
    }
}

var output = "";
for(i,0,len,1)
{
    if (primes[i] != 0)
    {
        //write("{0}") (primes[i] );
        set output = String.Concat(output) (primes[i]) (",");
    }
}

var index = length(output) - 1;

set output = String.Remove(output) (index) (1); // remove the
// latest ','
write("{0}") (output );
```

Code generated from scr mode ppl

```
//Sieve of Eratosphenes
import String;
var (n[100]);
var (len[ + (n) (1) ]);
array(primes[len]);
loop (i) (0) (len) (1)
(
  do
  (
    (set (primes[i]) (i))
  )
);
loop (i) (2) (len) (1)
(
  do
  (
    (
      loop (j) ( + (i) (1) ) (len) (1)
      (
        do
        (
          (
            if ( == (primes[j]) (0) )
            (
              (continue)
            )
          )
          (
            if ( == (mod(j) (i)) (0) )
            (
              (set (primes[j]) (0))
              (continue)
            )
          )
        )
      )
    )
  )
);
var (output[""]);
loop (i) (0) (len) (1)
(
  do
  (
    (
      if ( != (primes[i]) (0) )
      (
        //write("{0}") (primes[i] );
        (set (output) (String.Concat(output) (primes[i]) (",")))
      )
    )
  )
);
```

```
    )  
  )  
);  
var (index[ - (length(output)) (1) ]);  
set (output) (String.Remove(output) (index) (1));  
// remove the latest ', '  
write("{0}") (output );
```

The following example performs copying elements from two dimensional storage to one dimensional array
see examples\scr\copyto.scr

```
// copy row elements from first column to last column  
// prepare before call destination array  
function CopyRowElementsToArray(src,row,first_element,last_element,dst)  
{  
  write(src={0} row={1} first_element={2} last_element={3} dst={4})  
    (getname(src))(row)(first_element)(last_element) (getname(dst));  
  
  for(i, first_element, last_element + 1)  
  {  
    set dst[i] = sget(src)(row)(i);  
  }  
}  
//=====
```

```
// copy column elements from first row to last row  
// prepare before call destination array function  
CopyColumnElementsToArray(src,column,first_element,last_element,dst)  
{  
  write(src={0} column={1} first_element={2} last_element={3} dst={4})  
    (getname(src))(column)(first_element)(last_element) (getname(dst));  
  
  for(i, first_element, last_element + 1)  
  {  
    set dst[i] = sget(src)(i)(column);  
  }  
}  
//=====
```

```
import String;  
storage(src)(8)(8);  
var tmp = 0;  
for(i,0,8)  
{  
  for(j,0,8)  
  {  
    PlusPlus(tmp);  
    sset(src)(i)(j)(tmp);  
  }  
}
```

```
}
swrite(src);

array dst_row[6];
write("function CopyRowElementsToArray");
CopyRowElementsToArray(src)(1)(0)(5)(dst_row);

var output = "";
var index;

for(i,0,6)
{
    set output = String.Concat(output)(dst_row[i])(",");
};

set index = length(output) - 1;
set output = String.Remove(output)(index)(1); //remove the latest ','
write("{0}")(output );

set output = "";
array dst_column[8];
write("function CopyColumnElementsToArray");
CopyColumnElementsToArray(src)(7)(0)(7)(dst_column);

for(i,0,8)
{
    set output = String.Concat(output)(dst_column[i])(",");
};

set index = length(output) - 1;
set output = String.Remove(output)(index)(1); //remove the latest ','
write("{0}")(output );
```

```
>rc examples\scr\copyto.scr;
```

results:

	0	1	2	3	4	5	6	7
	-----NS.Global.src-----							
[0]	0	1	2	3	4	5	6	7
[1]	8	9	10	11	12	13	14	15
[2]	16	17	18	19	20	21	22	23
[3]	24	25	26	27	28	29	30	31
[4]	32	33	34	35	36	37	38	39
[5]	40	41	42	43	44	45	46	47
[6]	48	49	50	51	52	53	54	55
[7]	56	57	58	59	60	61	62	63

```
function CopyRowElementsToArray
```

```
src=src row=1 first_element =0 last_element=5 dst=dst_row
```

```
8,9,10,11,12,13
```

```
function CopyColumnElementsToArray
```

```
src=src column=7 first_element =0 last_element=7 dst=dst_column
```

```
7,15,23,31,39,47,55,63
```

Run file **examples.bat** with numerous examples of code.

References

1. Polymorphic Programming Language

https://en.wikipedia.org/wiki/Polymorphic_Programming_Language

1969 Thomas A. Standish

2. Prototypical Programming Language

<https://www.mathstat.dal.ca/~selinger/ppl/>

2000, Ari Lamstein and Peter Selinger

3. Practical Programming Language

<https://www.ppl-lang.dev/index.html>

4. Introducing Gen, a new PPL language by MIT

Probabilistic Programming Language

<https://becominghuman.ai/introducing-gen-a-new-ppl-language-by-mit-f77397eeff3>

Gen it is packet for Julia

2019, Alexandre Dall Alba

5. Piped Processing Language

<https://opendistro.github.io/for-elasticsearch-docs/docs/ppl/>