Table of Contents

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 2019,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 utility calls 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;
```

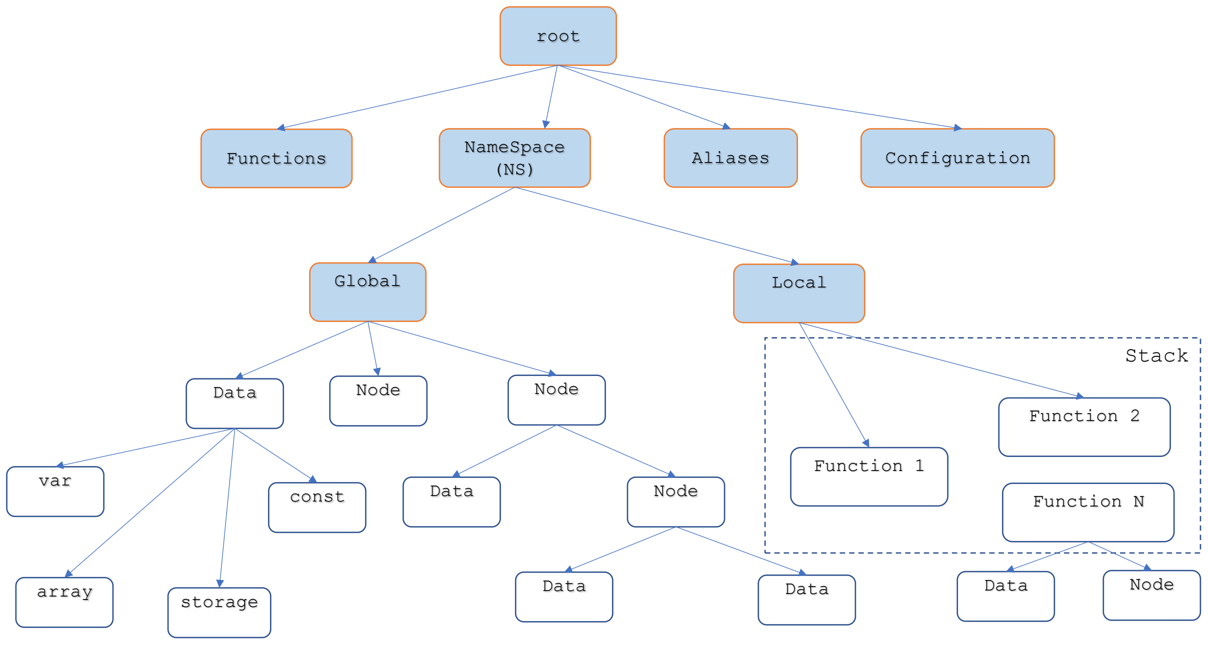
Execution of the program in the language PPL is carried out by means of the utility cppl.exe, which control commands are listed in section Keywords.
There are different statement formats for ppl mode and scr mode if a statement belongs to two modes.

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 –
Code, 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**.

Tree **Functions** are created 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 to read files with user's functions and add these functions to Tree Functions.

Tree **Aliases** are created from file Aliases.data, to display its contents perform:

```
>display Aliases;
```

Tree **Configuration** are created from file Configuration.data, to display its contents perform:

```
>display Configuration;
```

Any kinds of data (var, const, array, storage and node) are saved in Tree **Global** as common data for all functions and for code without functions. Error occurs on data re-creation, it is possible to delete data and to create again, but this can lead to undesirable consequences.

PPL uses Tree **Local** for saving data in functions, to display its contents perform in function:

```
>display Local
```

When exiting the function, local data of this function is destroyed.

To present Tree **Code** perform command dstree (Display Syntax Tree):

```
>code: scr
>var x = 2+3*(4+5);
result = 29
>dstree;
-----Syntax Tree-----
-N0     root    [0]
---N1   +
-----L0 2
-----N2 *
-------L0       3
-------N3       +
---------L0     4
---------L1     5
```

For illustration difference between mode scr and mode 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])
   (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. 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\"
```

Libraries

Default name of library is **Main**, it loads always when Cppl.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, cls, shell, init, code, showcode, readcode, fdreadcode, createppl, display, dstree, datanames, suspend, resume, debugppl, log, exit

Special Commands

import, importlist, eval, length, isexist, del, getbyname, getbyvalue, set, getvalue, getname

Nodes

createnode, copynode

Variables and Arrays

var, const, array, realloc

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

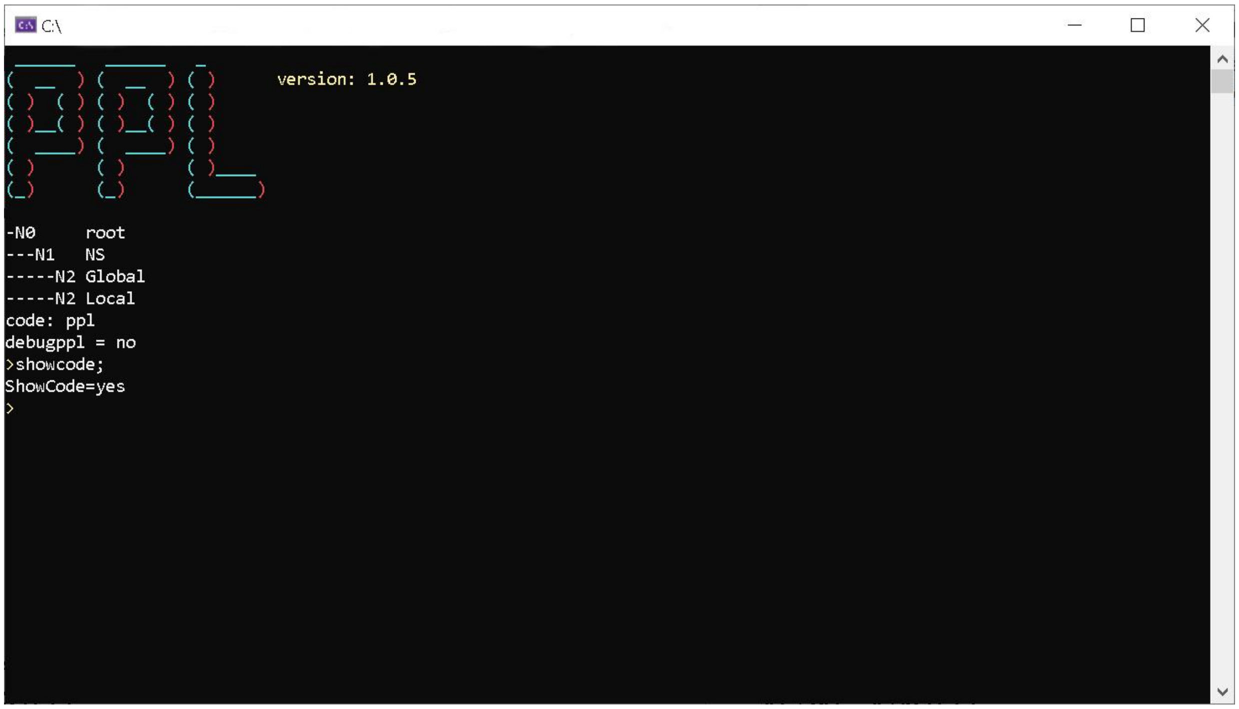
write, writearray, readline

Functions

function, funclist, return

CPPL utility

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



```
 _______  _______  _
(  __  ) (  __  ) ( )      version: 1.0.5
( )  ( ) ( )  ( ) ( )
( )__( ) ( )__( ) ( )
(  ____) (  ____) ( )
( )      ( )      ( )____
(_)      (_)      (______)

-N0     root
---N1   NS
-----N2 Global
-----N2 Local
code: ppl
debugppl = no
>showcode;
ShowCode=yes
>
```

There are following subdirectories and files use to work with cppl.exe
Subdirectories:
\Data
\Examples
\Functions
Files:
Aliases.data
Configuration.data
CPPL.exe
OFD.exe
Createulc.exe
CPPL.dll, Createulc.dll, ArrayList.dll, Directory.dll, String.dll, Math.dll, Convert.dll, Excel.dll, ExcelFW.dll, Template.dll

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 finishes and continues in interactive mode.

Example:
File sample.scr
```
var %1% = 0;
%2%;

cppl.exe sample.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 **>** writes 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.

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:

```
>help;
>? Code;
     Sets mode for Console input or displays on Screen
     code ppl | scr
```

Any other library has function help for display its contents.
<name of library>.help [keyword]

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
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.

Examples

```
1.
> Directory.SetCurrentDirectory(examples\ppl);
>rc loop.ppl;  // or rc examples\ppl\loop.ppl
loop (i)(0)(3)(1)
(
   do
   (
      (write(PPL))
       (write(ppl))
   )
 );

2.File sample.scr
var %1% = 0;
%2%;
>rc sample.scr x display;
```

fdreadcode

Like readcode with using FileDialog 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**

createppl

Creates file in format ppl from file in format scr.

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

Examples

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

display

Displays nodes(N) and leaves(L), alias – d.
Format:
display [root | Global | Functions | Aliases | Configuration| Local | path.node_name]
d [root | Global | Functions | Aliases | Configuration| Local | path.node_name]

default: Global
Examples:

```
 >array(y[3])(0);
  >d;
-N  NS
---N  Global
-----N  y
-------L0              [0]
-------L1              [0]
-------L2              [0]
```

dstree

Displays syntax tree of the latest operation with nodes(N) and leaves(L) of code.
Within function dstree displays all function tree.
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-----
        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. This operation allows to control the re-creation of the variable.
Format: **debugppl [yes|no]**
It is possible to set **debugppl** in file **Configuration.data**.
Example:

```
>var(x)
>debugppl yes
>var(x)
Info [FuncCreateVariables] Global var [x] is updated
>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 Cppl.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
Examples:

```
>createnode node;
>var (node.x);
>isexist node.x;
result=true
```

del

Deletes any kinds of data from Global or Local Tree
Format: **del fullname**
fullname:= node path.name
node path:= node path | node
Examples:

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

getbyname

Gets value from array by name.
Format**: getbyname(name array)(name element)**
Example:

```
See example in readdata
>getbyname(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 getbyname 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 is already exist in array.
Examples:
Format ppl:

```
>var (x);
>set(x)(0);

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

Format 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
```

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("get(y[0]) = {0}")(get(y[0]));
result: get(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

createnode

Creates node in path, default path is "Global"
Format: **createnode(path.name)**
Examples:

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

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
```

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;

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;
```

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;
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
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
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

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.
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)**

Examples:

```
>storage (x)(3);
>sinit(x)(0);
>swrite(x);
0           1           2
0           0           0
```

```
>storage(xx)(5)(3);
>sinit(xx)(00);
>swrite(xx);
   -------------------------------------------NS.Global.xx-------------------------------------------
[0]   00          00          00
[1]   00          00          00
[2]   00          00          00
[3]   00          00          00
[4]   00          00          00
```

```
>storage(xxx)(3)(3)(5)
>sinit(xxx)(0)
>swrite(xxx)
-----------------------------------------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**
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 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.
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
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[2]);
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.

Format:

```
loop (iteration var)(begin)(end)[(increment)]
(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))
  )
);
```

for

Iteration block for scr mode only.
Format:
```
for(iteration var, begin, end, increment)
{
   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
}
```

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.
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\"language");  // ppl"language
//tab in string
>write("ppl\tlanguage");  // ppl language
//newline in string
>write("ppl\r\nlanguage");  // ppl
                            //language
>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;
>write("{0} {1}")( "x = " )(x);
>code ppl
>write("{0}")(*(2)(5));
```

writearray

Writes array contents to the standard output stream.
Format:
writearray ([node.]array_name) | [node.]array_name
Example:

```
>rc examples\scr\whitearray.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
```

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

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

Function must be declared before called.
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

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

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:

```
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(get(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)).

```
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
```

return

Exit from function .

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

Now cppl utility includes 6 additional DLLs with C# functionalities and this list will be expanded.

Math

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 function in Math.DLL:
>Math.help(function 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.BugMul(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

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

To get short help of every function in String.DLL:
>String.help(function 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 digital string: **String.Remove(string)(value)**
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
```

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

Returns true|false: **String.StartsWith(string)(value)**

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

String.ToCharArray(string)(node_of PPL_chars_array)
node_of PPL_chars_array is string in quotes or value of variable.
If node_of PPL_chars_array exists its contents will be changed.

```
Example:
>Import String
>String.ToCharArray("qwerty")("chars");
>writearray(chars)
-----Array chars-----
[0]     q
[1]     w
[2]     e
[3]     r
[4]     t
[5]     y
```

String.ToLower(string)

String.ToUpper(string)

String.Trim(string)[(string)]

Directory

GetFiles
GetDirectories
SetCurrentDirectory
GetCurrentDirectory
GetParent
CreateDirectory
Exists
Delete

To get short help of every function in Directory.DLL:
> Directory.help(function name);

Directory.GetFiles("node of PPL array")("path")
node of PPL array is string in quotes or value of variable.
If node of PPL array exists its contents will be changed.

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
```

Directory.GetDirectories("node of PPL array")("path")
node of PPL array is string in quotes or value of variable.
If node of PPL array exists its contents will be changed.

Example:
```
Directory.GetDirectories("dir")("c:\Users");
or
var (x["dir"]);
```

Directory.GetDirectories (x)(path)

Directory.SetCurrentDirectory("path")

Directory.GetCurrentDirectory()

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

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

Returns true or false : **Directory.Exists("path")**

Returns true or false: **Directory.Delete("path")**

ArrayList

Create	**CopyTo**
Write	**Reverse**
Add	**Remove**
AddArray	**Insert**
Clear	**IndexOf**
Contains	**Sort**

Example:
Run all functions see Examples\lib\ArrayList.scr

```
>import ArrayList;
>createnode Private;
>ArrayList.Create("al1");
>array(Private.src)(ONE)(TWO)(THREE);
>ArrayList.AddArray("Private.src")("al1");
>ArrayList.Add(al1)(1one);
>ArrayList.Add(al1)(2two);
>ArrayList.Add(al1)(3three);
>ArrayList.Remove(al1)(1one);
>ArrayList.Write(al1);
al1
        ONE
        TWO
        THREE
        2two
        3three
>ArrayList.Reverse(al1);
>ArrayList.Write(al1);
al1
        3three
        2two
        THREE
        TWO
        ONE
>ArrayList.Insert(al1)(2)(4four);
>ArrayList.IndexOf(al1)(3three);
result = 0
>ArrayList.Sort(al1);
>ArrayList.CopyTo(al1)("Private.dst_arr");
>d;
-N1     NS
---N2   Global
-----N3 Private [Node]
-------N4       src     [Array 3]
---------L0     #       [ONE]
---------L1     #       [TWO]
---------L2     #       [THREE]
-------N4       dst_arr [Array 6]
---------L0     #       [2two]
---------L1     #       [3three]
---------L2     #       [4four]
---------L3     #       [ONE]
---------L4     #       [THREE]
---------L5     #       [TWO]
---N2   Local
>ArrayList.Clear(al1);
```

To get short help of every function in ArrayList.DLL:
> ArrayList.help(function name);

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

Writes all elements from the ArrayList to the standard output stream:
ArrayList.Write(arrlist name)

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

Adds node of PPL array to the end of the ArrayList:
ArrayList.AddArray(node of PPL array)(arrlist name)
node of PPL array is string in quotes or value of variable.
If node of PPL array exists its contents will be changed.

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 node of PPL array:
ArrayList.CopyTo(arrlist name)(node of PPL array)
node of PPL array is string in quotes or value of variable.
If node of PPL array exists its contents will be changed.

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)

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)**

Convert

StringToInt32Array
StringToHexArray
HexToBin
BinToHex
IntToHex
HexToInt
IntToBin
BinToInt

String chararacters convert to int32 array: **Convert.StringToInt32Array(string)("Int32 array")**
String chararacters convert to hex array: **Convert.StringToHexArray(string)("Hex array")**
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
```

All below mentioned functions convert data in accordance with function name and returns:
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:

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

Excel

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

```
Open
Close
Read
CreateWorkBook
Write
SaveAs
```

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")
Creates workbook for writing:
Excel.CreateWorkBook()
Writes storage to Excel cells, quantity of cells must be enough to save storage:
Excel.Write("sheet")("left top")("right down")("storage")
Saves created XLSX file after writing:
Excel.SaveAs(filename.xlsx)

Examples:
see file Examples\Excel\test.scr

```
import Excel;
Excel.Open("%1%\examples\Excel\example.xlsx");
Excel.Read("Sheet1")("A1")("H10")("Example_XLSX");
Excel.Close();
swrite(Example_XLSX);
Excel.CreateWorkBook();
Excel.Write("Sheet1")("A1")("H10")("Example_XLSX");
Excel.SaveAs("%1%\examples\Excel\example2.xlsx");
Excel.Close();
```

```
>rc examples\excel\test.scr c:\path
```

Parameter **c:\path** overrides the variable **%1%** in file test.scr.

Structure of User’s DLL

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

Example:

```
>import Template
>importlist()
Main
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 Dependensies the project **PPL**
Create with createulc.exe 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 with   EndOfLine");
      //...
      try
      {
        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,l ast_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
```

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/introduting-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/