

PPL language

Table of Contents

Introduction.....	7
Base Concepts	9
Trees	11
Comments.....	15
Configuration	16
Identifiers and DNS.....	16
Public and private variables.....	16
Compound statements (blocks)	17
Libraries	18
Keywords	19
CPPL utility.....	21
WPPL utility	23
PPL Assistant	26
Service Commands	27
help	27
version	27
cls	28
shell.....	28
init.....	28
code	28
showcode.....	28
readcode	28
fdreadcode.....	30
createpplcode.....	30
display	30
displaynodes	30
dstree.....	32
datanames	33
suspend and resume.....	33
debugppl.....	33
traceppl.....	34

recreate	34
log	34
exit	34
sumdata	34
Special Commands	35
import	35
importlist	35
eval.....	35
length	36
isexist	36
isdigit	37
isinteger	37
isalldigits	38
isallinteger	38
iseven, isodd	38
del	39
calc	39
sleep.....	39
getbykey	40
getbyvalue	40
set	40
setkvp.....	41
getvalue	42
getname.....	42
gettoken.....	42
Nodes and Leaves.....	44
createnode.....	44
copynode	44
getnodes	45
getleaves.....	47
struct.....	48
CopyStruct	49
Arithmetic operators.....	50

Logical operators	50
Variables and Arrays.....	51
var	51
const	51
array.....	52
realloc	54
array.push	56
array.pop	56
array.reverse.....	56
array.shift.....	56
array.remove	56
array.clear	56
array.unshift	57
array.insert	57
Storage	58
storage	58
sinit	59
sget	59
sset.....	60
swrite	60
sinfo	61
ssetrow	61
Backup & Recovery.....	62
savedata.....	62
readdata.....	63
Control Flow	64
if, else.....	64
switch, case, default	66
loop,do.....	68
for	69
break	70
continue.....	71
Input and Output.....	72

write.....	72
write#, writeline	72
writearray	73
readline.....	74
Functions	75
function.....	80
call.....	85
return	87
funclist	87
funcname	88
argc	88
getargname.....	88
Delegates and callbacks	89
delegate	89
dltinstant.....	89
dltset	90
dltcall.....	90
callback	91
Additional functionalities	92
Math	93
String.....	95
Directory	99
Array	101
ArrayList.....	103
Queue	107
Stack.....	108
Dictionary.....	110
Convert	111
Excel	113
File.....	115
Random.....	116
Console	118
Vector	119

Matrix	120
MN_Numerics.....	121
DataFrame	123
Statistics.....	130
Structure of User's DLL.....	133
Examples of code	135
References.....	139

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, write#, call** and following compound statements (blocks) – **struct,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 will be converted into a 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

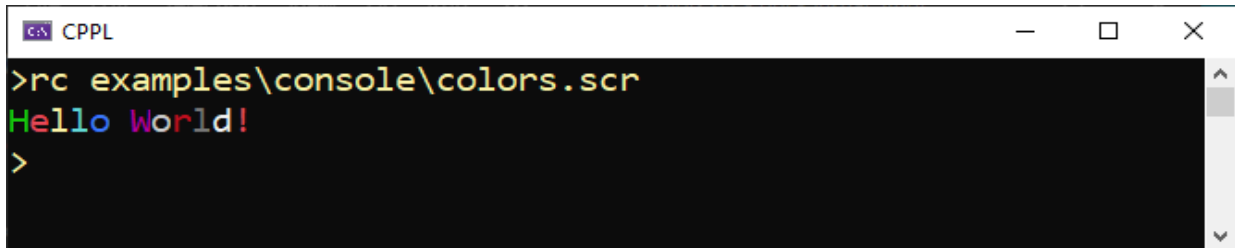
As is customary in many programming language guides the first ppl program is:

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

Another example with using Console and String libraries:

File examples\console\colors.scr:

```
var text = "Hello World!";  
array colors[] = {Green,Red,Yellow,Cyan,Blue,Black,  
                  Magenta,Gray,DarkRed,DarkGray,White,Red};  
  
array ArrayChars;  
call String.ToCharArray(text, getname(ArrayChars));  
  
for(i,0,length(text))  
{  
    // function with 1 argument not need operator call  
    Console.ForegroundColor(colors[i]);  
    Console.Write(ArrayChars[i]);  
};  
Console.DefaultColors();  
write();    // to new line
```



The screenshot shows a terminal window titled "CPPL". The prompt is ">rc examples\console\colors.scr". The output is "Hello World!" where each word is in a different color: "Hello" is green, "World" is red, and "!" is yellow. The prompt ">" is shown on the next line.

More 4 samples to illustrate the possibilities of PPL:

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

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



2. Added loading of second file, function call, creation node and array under this node

```
>rc examples\cowsay\cowsay2.scr
```

3. Added second operator 'for' to move cow

```
>rc examples\cowsay\cowsay3.scr
```

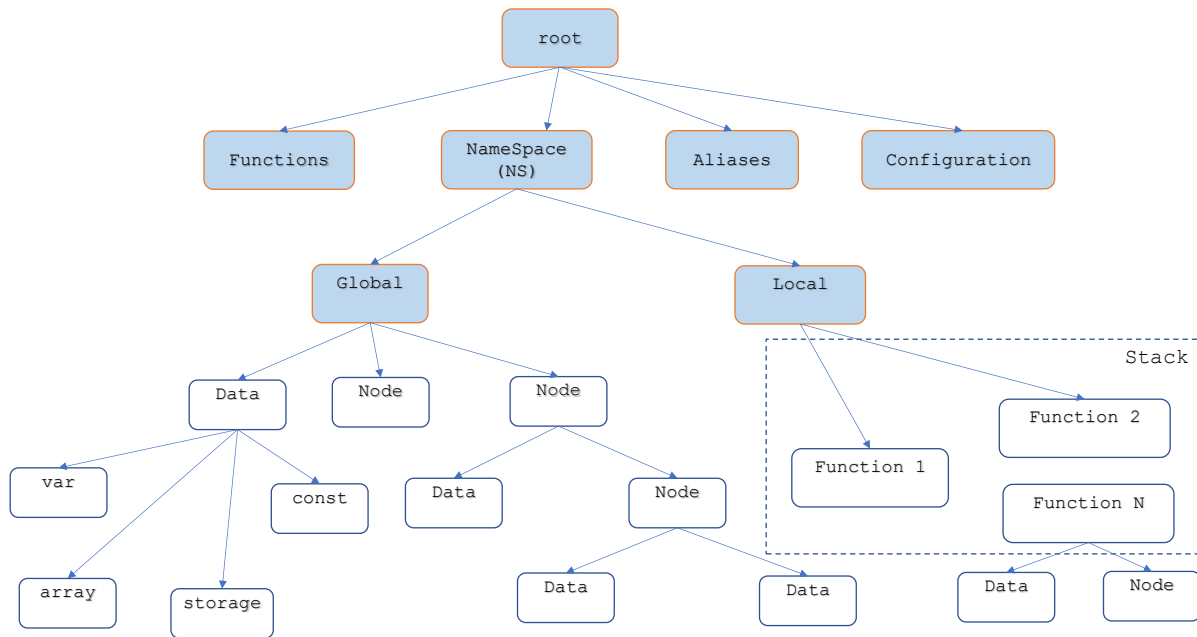
4. Added passing data as arguments when calling command readcode

```
>rc examples\cowsay\cowsay4.scr 1 1 10
```

Trees

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 Cpp.exe loads or re-created by command **init**.

By default Tree **Functions** is filling from file Functions\ **CommonFunctions.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.

Example:

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

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

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

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

Variables for functions are created in Tree **Local**, to display its contents perform in function:

>display Local

When exiting a function, its variables are deleted.

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

```
>rc Examples\scr\for.scr
=== scr code for preprocessor ===
var begin = 0;
var end = 3;
for(i,begin + 1,end + 1,1)
{
    write(i);
}
=== generated by preprocessor ppl code ===
>var (begin[0]);
>var (end[3]);
>loop (i) ( + ( begin ) ( 1 ) ) ( + ( end ) ( 1 ) ) ( 1 )
(
    do
    (
        (write(i))
    )
);
=== 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\ CommonFunctions .pp1])
  (Code [pp1])
  (debugpp1 [no])
  (delete all in readcode [yes])
  (log [no])
  (stay_interactive [no])
  (ReplaceMathLogicOperators [no])
  (OFD_port [11000])
  // (UserFunctions1 [Functions\*.pp1])
  // (UserFunctions2 [Functions\*.pp1])
  (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 hidden or private variables ([see hidden variables](#)).

String **"all"** can not be used as name([see cmd del](#)).

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\""
```

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);
```

Compound statements (blocks)

The following blocks **for**, **if**, **else** include one or several statements enclosed in curly brackets:

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

When only one statement (**not compound statement**) is in curly brackets it is possible to omit brackets:

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

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

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

Libraries

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

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

To display contents of any library perform:

<name of library>.help

or **help** for Main library

Example:

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

To get short information about any library function perform:

<name of library>.help(function name)

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

For Main Library help or ?:

```
>? d  
display | d [root|NS|Aliases|Functions|Local|node name]  
display NS.namespace.name]
```

Keywords

Keyword formats are defined in this tutorial, all format are defined for **ppl mode** by default. Additionally defined format for **scr mode** for some keywords. All keywords are divided into 9 groups and presented below:

Service Commands

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

Special Commands

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

Nodes and Leaves

createnode, copynode, getnodes, getleaves

Variables and Arrays

var, const, array, realloc, array.push, array.pop, array.reverse, array.shift, array.remove, array.clear, array.unshift, array.insert

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, writeline (write#), writearray, readline

Functions

function, funclist, funcname, argc, getargname, call, return

Delegates and callbacks

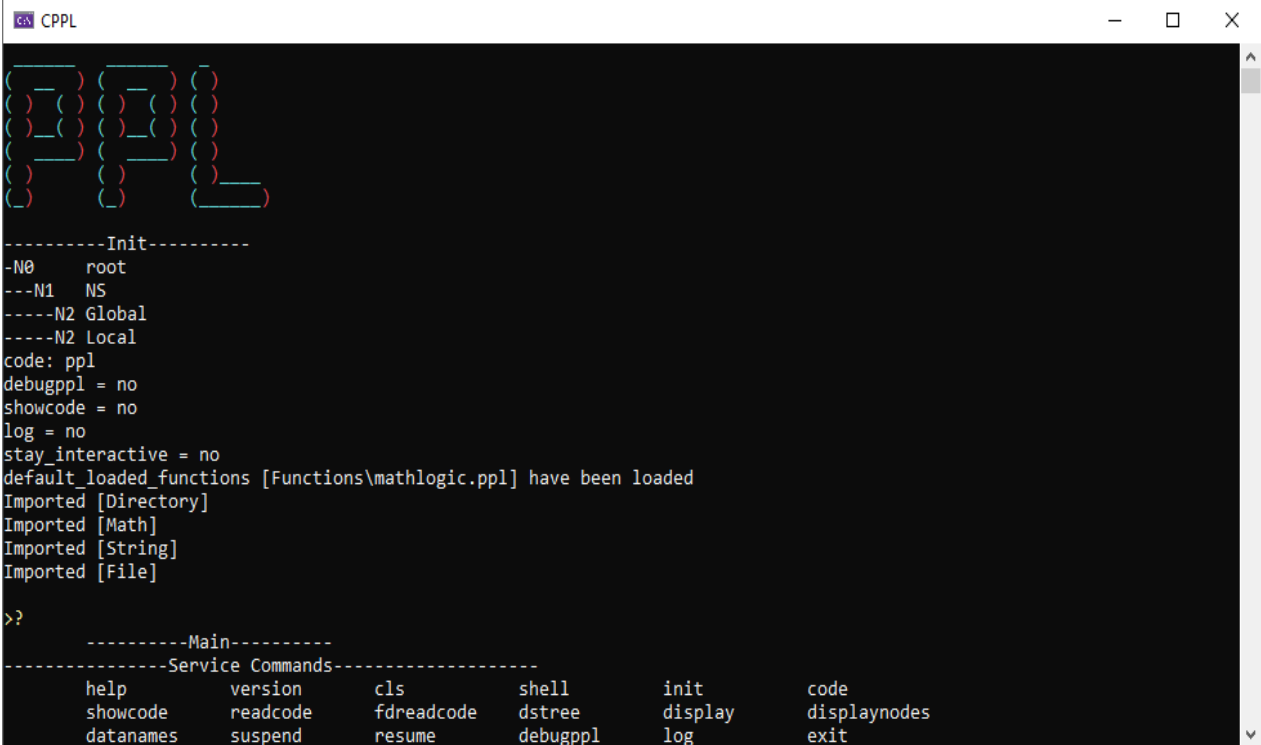
delegate, dlgtinstant, dlgtset, dlgtcall, callback

Special variables, constants and words

empty – see methods ArrayList.Add, Queue.Enqueue, Stack.Push
tab,comma,space – see String.Split, String.Splitcsv.

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]

>?

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

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,

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. Number between two symbols \$ is the serial number of argument.

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

Value of argument is literal, not command.

When value of **stay_interactive** in file **Configuration.data** = "no" 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 $1$ = $2$;  
>cppl.exe example.scr x 2;
```

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 arguments:

help (?), **import**, **readcode** (rc), **showcode**, **createnode**, **isexist**, **display** (d), **del**, **code**, **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'. Each section lists various commands and their functions.

```

>showcode;
>version
version 1.0.6
>help
>

-----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:



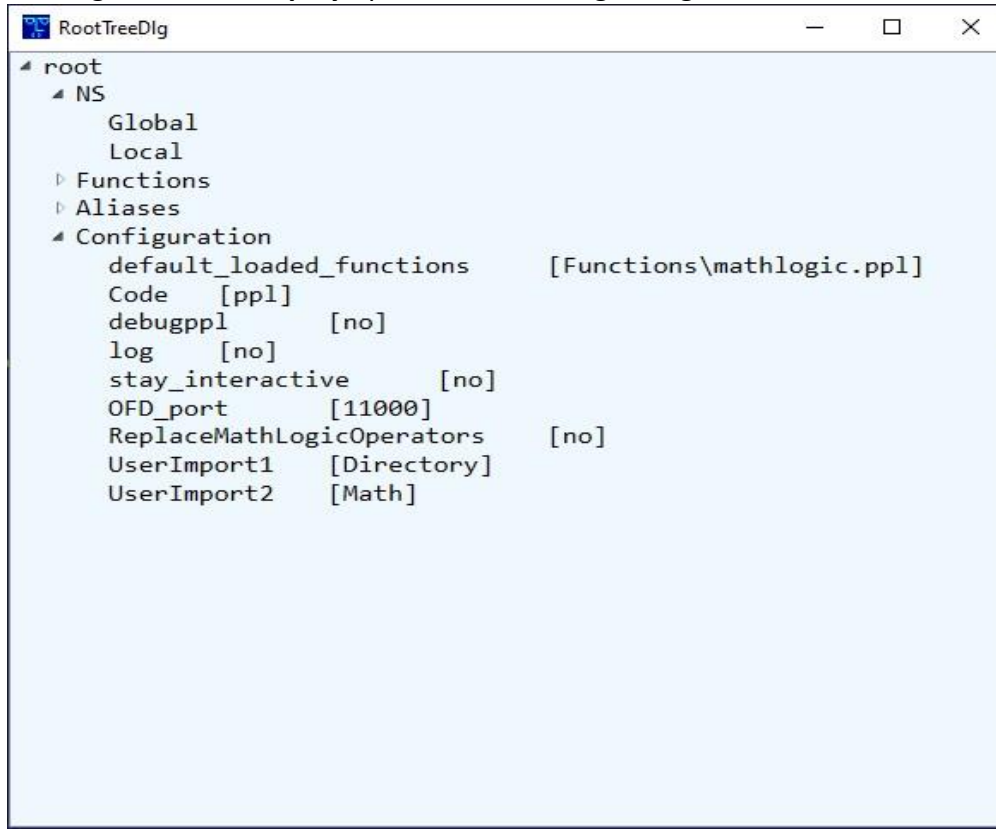
The ReadDataDlg dialog box has a title bar with minimize, maximize, and close buttons. It contains two text input fields: 'Filename:' with the text '\PPL\PPL\PPL\WPPL\bin\Debug\netcoreapp3.1\Data\Colors.data' and 'Node:' which is empty. To the right of the 'Filename:' field is a 'Browse' button. To the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



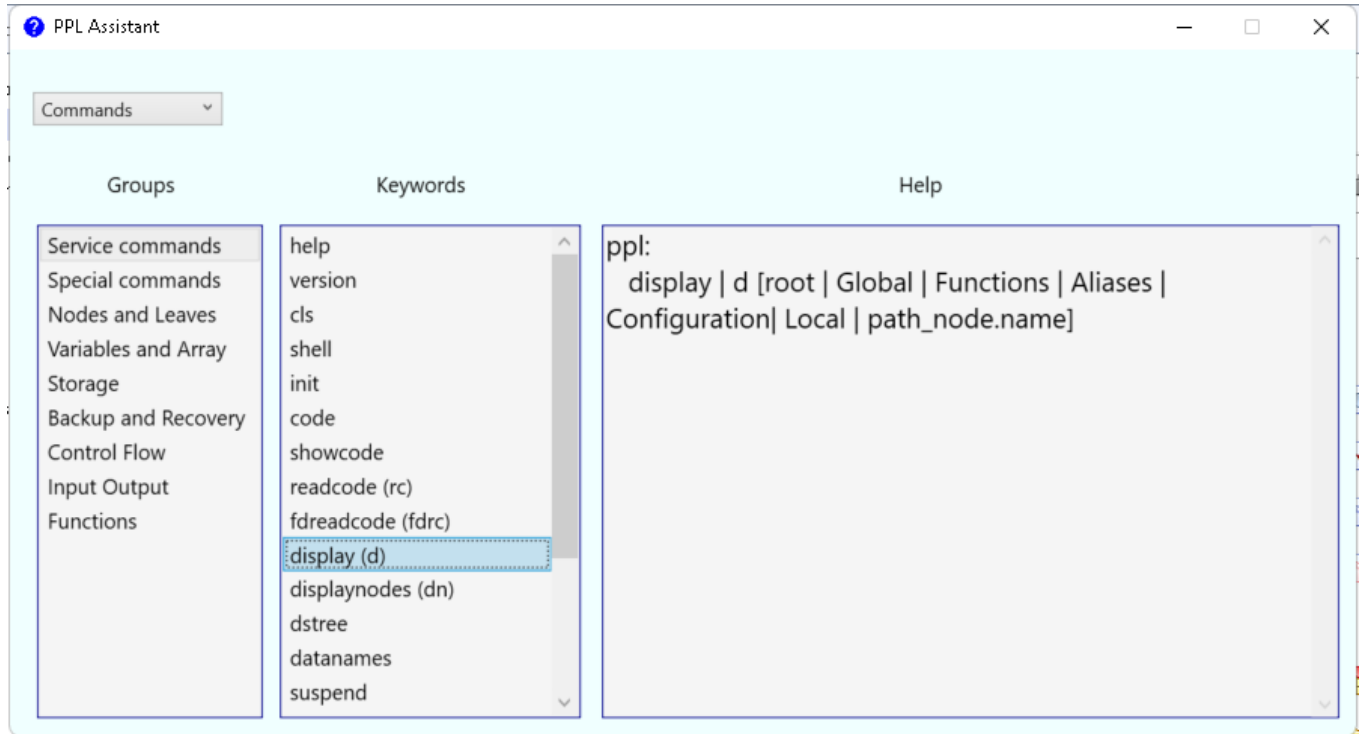
History of commands is supported by buttons **PgUp** and **Pgdn**.

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

PPL Assistant

PPL Assistant displays format commands in ppl and scr modes. As well this application displays format methods in [additional libraries](#).

Files JsonHelp*.json are generated by utility CreateUserLibCode.exe (version 1.0.15).



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

```

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

-----Special Commands-----
import    importlist  eval      length    del
isexist   isdigits   isinteger  calc      sleep
isalldigits isallinteger iseven    isodd
getbykey  getbyvalue set        getvalue  getname

-----Nodes and Leaves-----
createnode copynode  getnodes  getleaves

-----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      write#   writeline writearray readline

-----Functions-----
function   call     funclist  funcname  argc      return

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

Examples:

```

>? Code;
    Sets mode for Console input or displays on Screen
    code ppl | scr
>? "display" // only display in quotes
    display | d [root|NS|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.

At the end of the execution readcode the previous code will be set.

Format: **readcode | rc <file.scr|ppl> [arg1 arg2 ...]**

If arguments are present, they override the variables \$1\$, \$2\$ and so on in the body of the called file.

Number between two symbols \$ is the serial number of argument.

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

Arguments are literals, not commands.

If **delete_all_in_readcode = yes** in **Configuration.data** command

delete all is added automatically as first command when first command **readcode** is called.

The file being called can also include readcode commands. Files called by command **readcode** can be of different formats - .scr or .ppl.

If caller script and internal scripts are in the same path you can omit path for internals.

You can specify relative path from cppl.exe or full path, including drive:

```
>rc path\caller.scr
call internal script:
    rc internal.scr
or    rc path\internal.scr or rc drive:\path\internal.scr
```

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 $1$ = $2$;
>rc example.scr x 2;
```

3. reading script with command **readcode** inside:

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

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

```
File second.scr:
    write("second script");
```

```
File third.scr
    write("third script");
```

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

Results:

```
main script
first script
third script
return from first script
second script
return from main script
```

fdreadcode

Like readcode with using FileDialog to select file. This command loads ofd.exe and sets connection with **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

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

display

Displays nodes(N) and leaves(L) in Tree, 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 – dn.

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

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

Examples:

```
>rc examples\scr\func7.scr
```

```
function func(n,m)
{
    write#("{0} {1}", n , m);
    dstree;
}
func(aaa)(111);
```

Result:

```
aaa 111
-----Syntax Tree-----
-N3      func      [function]
---L0    n
---L1    m
---N4    #          [internal_block]
-----N5 write
-----L0          "{0} {1}"
-----L1          n
-----L2          m
-----N5 dstree
-----L0
```


datanames

Displays contents of [DNS](#).

Format: **datanames [Local]**

Examples:

```
>var(x) ;
>createnode Node1;
>array(Node1.arr[5]);
>datanames;
-----Global_dns-----
      _main0   var
      _main1   var
      _main2   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,

continue script execution – **resume**

stop script – **exit** and double click

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
```

traceppl

Displays all commands and function names on screen during the execution of commands. By default – traceppl no.

Format: **traceppl [yes|no]**

Example:

```
>traceppl yes;
```

recreate

Permits recreation vars, arrays, storage and nodes. By default – recreate no.

Format: **recreate [yes|no]**

Example:

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

log

Writes commands and results to logfile in directory **Log** or displays log value on Screen.

Format: **log [yes|no]**

It is possible to set **log** in file **Configuration.data**.

Opened logfile will be closed by command init or exit.

exit

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

sumdata

Defines argument type for summation. By default – **sumdata digit**.

Format:

sumdata [digit | string]

Example:

```
>code scr;  
>sumdata digit;  
>write#(1+2);  
3  
>sumdata string;  
>write#(1+2);  
12
```

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 in Global or Local, returns "True" or "False".

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

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

Example:

```
1.
>debugpp1 yes
>var (x);
>isexist x;
result=True
2.
> createnode Functions.New
> isexist Functions.New
result=True
```

isdigit

Checks is value of var or member of array or storage digital, returns "True" or "False".

Format: **isdigit(var name | member of array or storage | literal) |**

isdigit var name | member of array or storage | literal

Example:

```
>var (x[1.1]);
>isdigit 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 if all members of array or storage are 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 if all members of array or storage are 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 if integer value is 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**, so name “all” can not be used as any kind of variable names.

If **delete_all_in_readcode = yes** in **Configuration.data** command

delete all adds automatically as first command when command **readcode** is called. Otherwise all data will be saved in memory, if necessary add this command manually.

fullname:= node path.name

node path:= node path | node

Example:

```
>createnode Node1;
>var (Node1.x);
>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, may be used for ppl and scr modes, but in interactive mode only, not in .ppl or .scr files.

Format: **calc math.expression**

Example:

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

sleep

Suspends the interpreter for the specified number of milliseconds

Format: **sleep(msec)**

Example:

```
>sleep(100);
```

getbykey

Gets value from array by name.

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

Example:

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

if key is absent return **nan**.

getbyvalue

Gets name from array by value.

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

Example:

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

if value is absent return **nan**.

set

Sets value for variable and array element

Format **ppl**:

**set (var_name | array_name [index]) (value | array_name [index])
index:=value | ppl expression**

Format **scr**:

**set var_name | array_name [index] = value | scr expression
index:=value | scr expression**

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

Examples:

>code ppl:

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

>code scr:

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


To calculate indexes command **set** in **scr**-mode creates temporary variables and deletes them at the end:

```
>code scr;  
>array (a) (1) (2) (3) (4) (5) (6) (7) ;  
>set a[1+2] = a[2+2]+ 1;
```

Generated **ppl**-code:

```
array (a) (1) (2) (3) (4) (5) (6) (7) ;  
var (#0[ + (1) (2) ]);  
var (#1[ + (2) (2) ]);  
set (a[#0])(+(a[#1])(1));  
del #0;  
del #1;
```

setkvp

Sets key and value array element

Format for **ppl** code:

setkvp(array_name [index])(key)(value | ppl expression)

index:=value | ppl expression

Format for **scr** code:

setkvp(array_name [index]) = key, value | scr expression)

index:=value | scr expression

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

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

>code **ppl**:

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

>code **scr**:

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

getvalue

Returns value of single var|const or array element.

Error: when argument is literal or not existed variable.

Format: **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 | array | array element as string.

Error: when argument is literal or not existed variable.

Format: **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

gettoken

Returns token in accordance with its number in string, string contains tokens, separated by "separator". Parts of string surrounded by quotes are passed.

As well gettoken may return number of tokens.

If number >= max number of tokens cmd returns **"Exception"**.

Format: **gettoken (string)(separator)(number)**

return: **item_value**

gettoken (string)(separator)

return: **number of tokens**

Example:

```
1.  
>code ppl;  
>debugppl yes;  
>gettoken("Hello,World") (" ,") (0);  
result = Hello  
>gettoken("Hello,World") (" ,") (1);  
result = World  
>gettoken("Hello,World") (" ,") (2);  
result = Exception
```

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

Nodes and Leaves

createnode

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

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

Examples:

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

>createnode Functions.Geo
```

copynode

Copies one or more times node from path with new name, by default path is "Global"

Format:

copynode (src node) (dst node) [number of copies]

default number of copies: 1

Examples:

```
>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
                  Data12]
---L12     #      [Staff.Operations management.Managers.Personal
                  Data13]
---L13     #      [Staff.Operations management.Clerks.Personal Data14]
---L14     #      [Staff.Operations management.Clerks.Personal Data15]
---L15     #      [Staff.Operations management.Clerks.Personal Data16]

```

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

getleaves

Creates ppl_array whose elements have names and values of node:

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
```

struct

Struct is block statement for **scr mode** only

Format:

struct name

```
{
    var| array| |storage| struct name;
    var| array| |storage| struct name;
    .....
}
```

Example:

```
struct MyStruct
{
    var x = 0;
    struct N
    {
        array arr[] = {1,2,3};
        var w;
    }
}
```

The following ppl code will be generated:

```
createnode MyStruct;
var (MyStruct.x[0]);
createnode MyStruct.N;
array (MyStruct.N.arr[]) (1) (2) (3);
var (MyStruct.N.w);
```

```
>d;
-N2      NS
---N3    Global
-----L0 empty    (const)
-----N4 MyStruct    [Node]
-----L0      x      [0]
-----N5      N      [Node]
-----N6      arr     [Array 3]
-----L0      #      [1]
-----L1      #      [2]
-----L2      #      [3]
-----L0      w
```


CopyStruct

CopyStruct is a function (see **Functions\CommonFunctions.ppl**) to create copy of struct one (by default) or several times. Node with destination name must be created before.

```
CopyStruct(src) (dst) (n[1]);
{
    if ( argc == 2 )
        copynode (src) (dst) (1);
    else
        copynode (src) (dst) (n);
}
```

Example 1:

See creation MyStruct in example for **struct**:

```
>createnode MMM;          // create destination node
>call CopyStruct("MyStruct","MMM");
>d;
-----N4 MMM                [MyStruct]
-----L0      x              [0]
-----N5      N                [Node]
-----N2      arr              [Array 3]
-----L0      #              [1]
-----L1      #              [2]
-----L2      #              [3]
-----L0      w
```

Example 2:

```
>createnode AAA;
>call CopyStruct("MyStruct","AAA") (2);
>d;
-----N4 AAA                [MyStruct]
-----N5      0
-----L0      x              [0]
-----N1      N                [Node]
-----N2      arr              [Array 3]
-----L0      #              [1]
-----L1      #              [2]
-----L2      #              [3]
-----L0      w
-----N5      1
-----L0      x              [0]
-----N1      N                [Node]
-----N2      arr              [Array 3]
-----L0      #              [1]
-----L1      #              [2]
-----L2      #              [3]
-----L0      w
>set AAA.0.x = 2;          // assignment
```

Arithmetic operators

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

and their aliases:

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

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

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

These are binary operators.

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

Sum, Sub, Mul, Div, Pow

Examples in ppl prefix notation:

```
+ (x) (y)
* (+ (x) (y)) (- (z) (3))
```

Examples in scr infix notation:

```
>code scr;
> var z = x + y;
> var z = (x + y) * (z - 3);
```

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 file **Configuration.data**.

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

These are binary operators.

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

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

Examples in ppl prefix notation:

```
== (x) (y)
&& (== (x) (y)) (== (z) (3))
```

Examples in scr infix notation:

```
x == y
(x == y) && (z == 3)
```

Variables and Arrays

var

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

Format **ppl**:

var (name) | (name[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 (y[+(2)(3)]);
>array(z)(1)(2)(3);
>var (x[get(z[0])]) // instead of var (x[y[0]])
```

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;
>createnode N1;
>createnode N1.N2;
>var greeting = "Hello";
>var x;
>var N1.N2.z = 2 + 3;
```

const

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

Format **ppl**:

const (name[init value])

name:= [node.]name

init value:= value | ppl expression

ppl expression:=value | prefix notation expression

Example:

```
>const (x[0])  
>const (y[(+ (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 array with same name already exists (see [recreate](#)).

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];
```

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

```
>code scr;  
>var x = 1;  
>array y[] = {1,2,3,4,5};  
var index = x + 1;  
>write(y[index]);          // or write(y[2])  
Only operator set can use index as expression  
>set y[x + 1] = 100;
```

Creation array with **length = 0**:

```
array arr; or  
array arr[]; or  
array arr[0];
```

In the following sample array with **length = 0** is created preliminary and reallocated in function **Directory.GetDirectories** in accordance with real length:

```
> array Dir.dir; // or array Dir.dir [0];  
>call Directory.GetDirectories(getname(Dir.dir), "c:\\users\\");
```

realloc

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

Format: **realloc(array name)(new length) [(init_value)]**

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
```

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

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
```

The following array commands are to add, remove, clear, reverse items in array.

array.push

Adds a new item to an array as last, returns a new size of array.

Format **ppl: array.push (array_name)(item_value) |**
array.push (array_name)(item_name)(item_value)

array.pop

Returns the latest **item name and itemvalue, separated by comma**, and removes item from array or **Empty**(if array is empty) .

Format **ppl: array.pop (array_name)**

Example:

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

array.reverse

Reverses items order in array, returns a size of array;

Format **ppl: array.shift (array_name)**

array.shift

Removes the first item of the array, returns a new size of array.

Format **ppl: array.shift (array_name)**

array.remove

Removes item by index, returns a new size of array.

Format **ppl: array.remove (array_name)(index)**

array.clear

Removes all items from array, returns 0.

Format **ppl: array.clear (array_name)**

array.unshift

Adds a new item as first to an array, returns a new size of array.

Format **ppl: array.unshift (array_name)(item_value) |**
array.unshift (array_name)(item_name)(item_value)

array.insert

Inserts item before item with index, returns a new size of array.

Format **ppl: array.insert (array_name)(index)(item_value) |**
array.insert (array_name)(index)(item_name)(item_value)

See array commands examples in **examples\ArrayCmds\Samples.scr**.

Additional array service see in file **CommonFunctions.Data**.

Storage

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

storage

Creates single variable, single-dimensional or multi-dimensional array with dimension from 1 to N in Global or in Local function scope. It will be error if name already exists (see [recreate](#)).

Storage contains several levels of arrays, name of the topmost level is name of storage, name of the bottommost arrays in each level is **Row**. Names of intermediate levels are array index in level. To set different length arrays on Row level use realloc. (see examples\scr\testswrite.scr).

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)]**

Limit for CPPL.exe max_window_width = Console.WindowWidth = 120

NoLimit for WPPL.exe

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**.

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

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

Default node: **NS.Global**

Example:

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

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]
```

Each item in such array has key (Black, Blue,...) and value(0,9,...) , use [getbykey](#) and [getbyvalue](#).

Control Flow

if, else

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

Format **ppl**:

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

Here expression in prefix notation.

Statement in ppl mode.

Format **scr**:

```
if (condition)
{
    statement;
    statement;
    [else
    {
        statement;
        statement;
    } ]
}
```

Here expression in infix notation.

Statement in ppl or scr mode.

Example ppl mode:

```
var(x[1]);
var(y[1]);
if (==(x)(y))
(
    (write(true))
    (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

1. 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:

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

break

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

Example:

```
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 with prefix notation

Example:

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

>write(12col);
Result: 12col
>write("Error: wrong name {0}") (12col);
Result: Error: wrong name 12col
```

This operator is used in ppl and scr mode.

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

write#, writeline

Like as write, for mode scr only, each argument is not enclosed in parentheses.

Format:

write#(arg) | write#("c# format", arg1 , arg2,...)

writeline(arg) | writeline("c# format", arg1 , arg2,...)

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

Example:

```
>write#("{0} {1}", aaaa,1+3)
Result: aaaa 4
```


writearray

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

Format **ppl**:

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

Format **scr**:

writearray [node.]array_name [row]

or

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

Examples:

1.

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

2.

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

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

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

- Mathematical and Logical functions,
- Array services and other functions.

Mathematical and Logical functions:

```

Sum (result)(n1)[( n2)]
Sub (result)(n1)[( n2)]
Mult (result)(n1)[( n2)]
Div (result)(n1)[( n2)]
Pow (result, n1, n2)
PlusPlus (result)    // 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. Result are returned in 1st parameter and does not passed to the next command:

```

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

```

Examples:

```

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

```

2. This sample returns **wrong result**:

```
>var(x[5]);  
>Sum(x)(x)(2);  
>d x  
-----Variables and arrays-----  
-L4      x      [5]  
>Sum(x)(2);    // right, x = x + 2
```

3.

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

User may create own functions file, like CommonFunctions.ppl, and set it in file

Configuration.data as **UserFunctionsN** or load it:

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

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

Array services and other functions:

CopyStruct (src, dst, n) – create copy/copies of structure

Example:

See examples\struct\test.scr:

```
struct MyStruct
{
    var x = 0;
    struct N
    {
        array arr[] = {1,2,3};
        var w;
    }
}
createnode MMM;
call CopyStruct("MyStruct", "MMM");
createnode NNN;
call CopyStruct("MyStruct", "NNN", 2);
dn;
//results:
code: scr
-N2      NS
---N3    Global
-----N4 MyStruct      [Node]
-----N5      N          [Node]
-----N6      arr        [Array 3]
-----N4 MMM          [MyStruct]
-----N5      N          [Node]
-----N2      arr        [Array 3]
-----N4 NNN          [MyStruct]
-----N5      0
-----N1      N          [Node]
-----N2      arr        [Array 3]
-----N5      1
-----N1      N          [Node]
-----N2      arr        [Array 3]
```

CsvToArray (var:str, array:arr) – to fill array from string data separated by comma

ArrayToCsv (array:arr, var:str) – copy to string separated by comma data from array

ArrayIsExist (result, array:arr, value) – return True/False if devined value exists in array

ArrayIndexOf (result, array:arr, value) – return the index of the first occurrence within array or -1.

ArrayLastIndexOf (result, array:arr, value) – return the index of the last occurrence within array or -1.

Example:

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

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

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

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

//results:
1, 2, 3, , 1, 5
result = 1,2,3,,1,5
index = 2
index = 4
```

See `examples\ArrayFunc\SamplesFunc.scr`.

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

Example: (examples\arrayfunc\foreach.scr)

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

var result = 0;
delegate dd (array:arr,var i);
dltgtinstant instant dd;
dltgtset instant sumfunc;
call ArrayForEach({1,2,3,4,5},instant);
write#("result = {0}",result);
//result = 15
```

function

Functions must be declared before called.

Functions are saved in Tree **Functions** or in Tree **Global** for later call.

Functions return data via parameters, like a classic procedure.

When function is called **by name** (in mode **ppl** or **scr**) each argument must be enclosed in parentheses.

When function is called by command '**call**' (in mode **scr** only) each argument not must be enclosed in parentheses.

Limitations:

1. For passing array member use 2 arguments [\(sample\)](#):

array name, index array member
or temporary variable:

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

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

Format **ppl**:

Statement terminator ';' does not follow after statements within function.

function

```
(  
  name  
  parameter_list  
  ( function body )  
);  
name ::= identifier  
parameter_list ::= parameter [parameter_list]  
parameter ::= (identifier) | (identifier[default value]) | empty  
function body ::= (statement1) [(statement2) (statementN)]  
identifier ::= [var] | [array] :<name>
```


Format **scr**:

Statement terminator ';' always follows after each type of statements.

```
function   name
            (parameter_list)
            {
                function body
            }
name::= identifier
parameter_list::= parameter, [parameter_list]
parameter::=
    identifier | identifier[default value] | identifier = default value | empty
function body::= statement1 [ statement2 statementN ]
identifier::= [var] | [array] :<name>
By default parameter type is var.
```

Examples:

```
>code scr
1.Function func ()
{
    write#("func");
}
2. function func (n,m[10])    // = func (var:n,var:m[10])    or
                             //function func (n,m = 10)
{
    write#(funcname());
}
3.function func (array: n)
{  write#( funcname());  }
```

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

```
    )  
  )  
)  
);  
test();    // function call
```

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

In the following example (see examples\scr\func4.scr) parameter index default value = 0 and this parameter may be omitted when the function is called.

Array and member array index are passed as 2 arguments.

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

7. Call function from node saved in Tree Global

```
code ppl;
createnode New;
function
(
    New.func(name)      // public
    (
        (write(name))
    )
);
function
(
    New._hfunc(name)    // private
    (
        (write(name))
    )
)
New.func("Greetings!");
8. Call function from node saved in Tree Functions
code ppl;
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!");
```

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

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

call

Command 'call' calls function in mode **scr**, it is possible to use expression in infix notation as arguments, do not need to enclose in parentheses each argument when function is called.

Format:

call function_name(arg1,arg2,...)

Examples:

```
1.
>code scr
>var x = 0;
>call Sum(x,4 + 1);
>d x
-----Variables and arrays-----
-L1      x      [5]
```

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

2. To get return of function, called by 'call',
run 'debugppl yes' before:

```
>debugppl yes;
>call (Math.PI());
result = 3.141592653589793
```

3. For transfer return of function to the next command
do not use 'call'

```
>write#( Math.PI() );
result = 3.141592653589793
```

```
4.
>write#(call Math.PI()); // error
>set x = call Math.PI(); // error
>set x = Math.PI();
3.141592653589793
```

It is possible to pass array by this manner:

call function_name({item1,item2,...},arg2,...)

Example: (see examples\CallFunc\funcarr2.scr and examples\CallFunc\funcarr2.ppl)

```
function SumArray(array:arr,var:result)
{
    var x;      // this var to pass to function array element
    for(i,0,length(arr))
    {
        set x = arr[i];
        Sum(result)(x);
    }
}
```

```
var result=0;
call SumArray({1,2,3,4,5},result);
write#("result = {0}",result);
```

instead of:

```
array arr[] = {1,2,3,4,5};
call SumArray(arr,result);
```

Preprocessor generates the following ppl-code:

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

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| finclist()

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)
```

funcname

Returns the current function name.

Format:

funcname()

Example:

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

argc

Returns number of arguments

Format:

argc()

Example:

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

getargname

Returns argument name (or argument value if it is literal) by parameter name

Format:

getargname (parameter_name)

Example:

```
function a(n)  
{  
    write#("arg_name for param {0} =  
        {1}]",getname(n),getargname("n"));  
}  
var i = 0;  
a(i);  
//result:  
//arg_name for param n = [i]
```


Delegates and callbacks

There are 4 operators for using delegates:

delegate – creation delegate

dlgtinstant – creation delegate instant

dlgtset - setting the function to delegate instant

dlgtcall – call function by delegate instant

delegate

delegate is created as an array whose elements define the method parameters.

Prefix “**delegate_**” is added to delegate name for internal using.

Format **ppl**:

delegate | **dlgt** (<delegate name>) (param1) (param2)...

param:= var_name | var:<var_name> | array:<array_name>

Format **scr**:

delegate | **dlgt** <delegate name> (param1, param2,...)

Example (**scr-mode**):

```
>delegate MyDelegate (var:x,array:arr) ;
```

dlgtinstant

dlgtinstant is created as an array with 2 elements, first is delegate name, second is empty and will be set by dlgtset. Delegate parameters types must be matched types of function parameters . Prefix “**dlgtinstant_**” is added to delegate instant name for internal using.

Format **ppl**:

dlgtinstant (<delegate instant name>)(<delegate name>)

Format **scr**:

dlgtinstant <delegate instant name><delegate name>

Example(**scr-mode**):

```
>dlgt dd (var:x,array:arr) ;
>dlgtinstant instant dd;
>d;
-N2      NS
---N3    Global
-----L0 empty    (const)
-----N4 delegate_dd      [Array 2]
-----L0      #          [var:x]
-----L1      #          [array:arr]
-----N4 dlgtinstant_instant [Array 2]
-----L0      #          ["delegate_dd"]
-----L1      #
```

dlgtset

dlgtset sets function name as second element in **dlgtinstant** array.

Format **ppl**:

dlgtset (delegate instant name)(function name)

Format **scr**:

dlgtset delegate instant name function name

Example (**scr-mode**):

```
// see previous example with dlgt and dlgtinstant
function f1(var:x,array:arr)
{
    write#("x = [{0}]",x);
    writearray arr;
}
>dlgtset instant f1;
```

dlgtcall

dlgtcall calls function defined in dlgtset.

Format **ppl**:

dlgtcall(delegate instant name)(arg1)(arg2)(arg3)...

Format **scr**:

dlgtcall delegate instant name(arg1,arg2,arg3,...)

Example:

```
// see previous examples with dlgt, dlgtinstant and dlgtset
var z = "qqq";
array y[] = {1,2,3};
dlgtcall instant (z,y);
result:
x = [qqq]
-----Array arr-----
[0]      1
[1]      2
[2]      3
```

See the following samples with delegates:

Examples\callfunc\dlgt.scr

Examples\callfunc\dlgt2.scr

callback

callback invokes synchronous callback method.

Format **ppl**:

callback (callback name)(arg1)(arg2)(arg3)...

Format **scr**:

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

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

```
function cb1(var:n){write#("function cb1 {0}",n)};
function cb2(var:n) { write#("function cb2 {0}",n);}

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

delegate dd (var:n);
dlgtinstant instant dd;
dlgtset instant cb1;
call f(instant,"PPL");
dlgtset instant cb2;
call f(instant,"PPL");
result:
function cb1 PPL
function cb2 PPL
```

Additional functionalities

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

There are two types of methods called from additional DLLs:

- methods that return result, this result may be used in the next operation, for example:

```
var result = Math.Max(10) (20);
```

- methods that not return result, for example:

```
ArrayList.Remove(arrlist, item1);
```

Methods of second type of may be called by command 'call' in scr-format:

```
call ArrayList.Remove(arrlist, item1);
```

```
write#("ArrayList.Contains item1 = {0}", ArrayList.Contains(arrlist)( item1)); // right  
write#("ArrayList.Contains item1 = {0}", call ArrayList.Contains(arrlist)( item1)); // wrong
```

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

```
code scr;  
function GetChar(result,text,index)  
{  
    set result = String.Char(text) (index);  
}  
var char;  
var text = "Hello";  
for(i,0,length(text))  
{  
    call GetChar(char,text,i);  
    write(char);  
}
```

To get list of methods of additional loaded DLLs:

<DLLname>.help

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
LastIndexOf	ToLower
Insert	ToUpper
Remove	Trim
Split	Char
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)(number of deleted symbols)

Example:

>rc examples\lib\StringRemove.scr

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

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

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

Determines whether this string instance starts with the specified character:

Returns **True** | **False**: **String.StartsWith(string)(value)**

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

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

Writes 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:
>array chars;
>String.ToCharArray("qwerty") (getname(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 one character from string: **String.Char(string)(index)**

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

String.Split(string)("ppl_array_separators")(getname(ppl_array_result)) or
String.Split(string)("var_separator")(getname(ppl_array_result))

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};  
>array split_text_array;  
>String.Split(text) ("separator") (getname(split_text_array));
```

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

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

array ppl_array_result;

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:

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

>rc Examples\Lib\FilesplitCsv.scr

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

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

String.IndexOf (string)(value)(index)

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

String.LastIndexOf (string)(value)(index)

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

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

Directory

Methods:

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

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

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

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

array node_of_PPL_array;

Directory.GetFiles("node_of_PPL_array")("path")

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

Example:

1.

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

2.

```
>rc examples\lib\WriteFilesInDir.scr
```

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

Result:

-----Arr files-----

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

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

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

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

Example:

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

Sets the current working directory to the specified directory:

```
Directory.SetCurrentDirectory("path")
```

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

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

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

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

Deletes the specified directory and any subdirectories and files in the directory

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

Array

Methods:

Max	Min	Sum
Average	Sum2	Sub2
Mult2	Div2	Sort
Reverse	IndexOf	LastIndexOf

To get short help of every method in ArrayDLL: **Array.help[(method name)];**

Returns result: **Array.Max("ppl_array")**

Returns result: **Array.Min("ppl_array")**

Returns result: **Array.Sum("ppl_array")**

Returns result : **Array.Average("ppl_array")**

Math operations with 2 array, result is saved in **result_ppl_array**:

Array.Sum2("ppl_array1")("ppl_array2")("result_ppl_array")

Array.Sub2("ppl_array1")("ppl_array2")("result_ppl_array")

Array.Mult2("ppl_array1")("ppl_array2")("result_ppl_array")

Array.Div2("ppl_array1")("ppl_array2")("result_ppl_array")

Replaces source array: **Array.Sort("ppl_array") ("double" | "string")**

Replaces source array: **Array.Reverse("ppl_array")**

Returns result: **Array.IndexOf("ppl_array")(value)**

Returns result: **Array.LastIndexOf("ppl_array")(value)**

Example:

```
>code scr;
>import Array;
>array arr[] = {-4,4, -3,3,-2,2};
>Array.Sort("arr") ("double");
>writearray arr;
>Array.Reverse("arr");
>writearray arr;
=====result=====
-----Array arr----- double-----
[0]      -4
[1]      -3
[2]      -2
[3]       2
[4]       3
[5]       4
-----Array arr----- double-----
[0]       4
[1]       3
[2]       2
[3]      -2
[4]      -3
[5]      -4
```

```
>import Array;  
>code scr;  
>array a[] = {1,3,,5};  
>debugppl yes;  
>Array.Average(a);  
result = 3  
>recreate yes;  
>array a[] = {1,3,0,5};  
Info: [CreateArrayFormat1] Global array [a] is created  
>Array.Average(a);  
result = 2.25
```

See samples in Examples\lib\Array.scr.

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

ArrayList

Methods:

Create	ToArray	Count
Write	Reverse	Get
Add	Remove	Set
Clear	Insert	Delete
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**:

Example:

```
>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)

Writes all elements from arrlist to PPL array created before with size = 0 and reallocated in **ArrayList.ToArray** with size of arrlist name:

array "ppl_array";

ArrayList.ToArray(arrlist name)(getname("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(arrlist 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);
>array (N1.dst_arr);
>ArrayList.ToArray(all) (getname(N1.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
```

Delete all ArrayList objects:

ArrayList.Delete();

Queue

Methods:

Create	Peek	
Count	Clear	
Write	Contains	
Enqueue	AddArray	
Dequeue	ToArray	Delete

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

Queue. help[(method name)]

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

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

Writes queue names or all elements from the specified queue to the standard output stream:

Queue.Write() or **Queue.Write(name)**

Adds an object to the end of the Queue: **Queue.Enqueue(queue name)(string)**

To add empty string use keyword **empty**.

Removes and returns the object at the beginning of the Queue:

Queue.Dequeue(queue name)

Returns the object at the beginning of the Queue without removing it:

Queue.Peek(queue name)

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

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

Queue.Contains(queue name)(string)

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

Writes all elements from Queue to the PPL array created before with size = 0:

```
array "ppl_array";  
Queue.ToArray(queue name) ("ppl_array")
```

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

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

Stack

Methods:

Create	Peek	
Count	Clear	
Write	Contains	
Push	AddArray	
Pop	ToArray	Delete

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

>Stack.help[(method name)]

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

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

Writes stack names or all elements from the specified stack to the standard output stream:

Stack.Write() or **Stack.Write(stack name)**

Inserts an object at the top of the stack: **Stack.Push(stack name)(string)**

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

Removes and returns the object at the top of the Stack:

Stack.Pop(stack name)

Returns the object at the top of the Stack without removing it:

Stack.Peek(stack name)

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

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

Stack.Contains(stack name)(string)

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

Writes all elements from Stack to the PPL array created before with size = 0:

array "ppl_array";

Stack.ToArray(stack name) ("ppl_array")

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

Examples:

```
>import Stack
Imported [Stack]
>Stack.Create(s)
>Stack.Push(s) (one)
>Stack.Push(s) (two)
>Stack.Push(s) (three)
>debugppl 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	Delete

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

>**Dictionary.help[(method name)]**

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

Write all elements from Dictionary to new PPL array created before with size = 0:

array "ppl_array";

Dictionary.ToArray(dictionary name) ("ppl_array")

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

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 converts to int32 array created before with size = 0 and reallocated in **Convert.StringToInt32Array** with size of string_characters:

Convert.StringToInt32Array(string_characters)("Int32 "ppl_array")

String characters converts to hex array created before with size = 0 and reallocated in **Convert.StringToHexArray** with size of string_characters:

Convert.StringToHexArray(string_characters)("Hex "ppl_array")

All below mentioned methods convert data in accordance with method name and return:

Returns string bin: **Convert.HexToBin(string with hex value)**

Returns string hex: **Convert.BinToHex(string with bin value)**

Returns string hex: **Convert.IntToHex(string with Int32 value)**

Returns string Int32: **Convert.HexToInt(string with hex value)**

Returns string bin: **Convert.IntToBin(string with Int32 value)**

Returns string Int32: **Convert.BinToInt(string with bin value)**

Examples:

See Examples\lib\Convert.scr

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

```
>array Hex;  
>Convert.StringToHexArray("12345") ("Hex");  
>writearray Hex;  
-----Array Hex-----  
[0]      31  
[1]      32  
[2]      33  
[3]      34  
[4]      35
```

Examples:

```
>debugpp1 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();
swrite(Example_XLSX);
```

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

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

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

File

Methods:

ReadAllText	ReadAllLines
WriteAllText	WriteAllLines
Exists	Delete

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

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:

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

Console

Methods:

ForegroundColor	Beep
BackgroundColor	Clear
ForegroundPromptColor	SetCursorPosition
DefaultColors	GetCursorPosition
Write	

Sets the foreground color of the console:

Console.ForegroundColor(color)

Sets the background color of the console:

Console.BackgroundColor(color)

Sets the prompt foreground color of the console:

Console.ForegroundPromptColor(color)

Sets the default foreground, background and ForegroundPromptColor color of the console:

Console.DefaultColors()

Writes the text representation of the specified value or values to the standard output stream: **Console.Write [(format)](string)(string)**

Plays the sound of a beep through the console speaker:

Console.Beep (frequency)(duration)

frequency - 37 to 32767 hertz

duration - msec

Clears the console buffer and corresponding console window of display information:

Console.Clear()

Sets the position of the cursor:

Console.SetCursorPosition(left column cursor position)(top row cursor position)

Gets the position of the cursor:

Console.GetCursorPosition ()

Returns '**left column cursor position, top row cursor position**'

See examples in examples\Console.

Vector

For using with library MathNet.Numerics and others. Using Vector and Matrix libraries significantly increases program performance.

Methods:

```
Vector.Create ("vector_name")(length)(type)
type:= double|float|decimal|bool|int|uint|long|ulong|string
types in accordance with value types in
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/built-in-types
```

```
Vector.Get(vector_name)(index);
Vector.Set(vector_name)(index)(value);
Vector.Add("vector_name")( ""ppl_array"" )
```

Write to line - "row"

Write not nullable data to column - "col"

Write all data to column - "col0"

```
Vector.Write("vector_name")(["row"|"col"|"col0"])
```

```
Vector.WriteNames()
```

```
Vector.Delete("vector_name")
```

```
Vector.DeleteAll()
```

Examples:

```
>import Vector;
>array v[] = {1,2,3,4,5};
>call Vector.Create ("V",5,int);
>call Vector.Add("V",getname(v));
> call Vector.Write("V");
=====vector V=====
1  2  3  4  5
>call Vector.Set("V",0,0);
> call Vector.Write("V",col);
=====vector V=====
[1]  2
[2]  3
[3]  4
[4]  5
>call Vector.Delete("V");
```

See examples in **examples\MatrixVector**.

Matrix

For using with library MathNet.Numerics and others.

Methods:

```
Create ("matrix_name")(rows)(columns)(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
Get("matrix_name")(index_row)(index_column)
Set("matrix_name")(index_row)(index_column)(value)
AddArrayToRow ("matrix_name")(row)("")ppl_array""
AddArrayToColumn ("matrix_name")(column)("")ppl_array""
Write to line - "row"
Write not nullable data to column - "col"
Write all data to column - "col0"

Write("matrix_name")[("row" | "col" | "col0")]
WriteNames()
Delete("matrix_name")
DeleteAll()
Rotate("matrix_name")("cw" | "ccw") - for square matrix only
```

Example:

```
>import Matrix;
>call Matrix.Create ("A",3,3,float);
>array r1[] = {1,2,3};
>array r2[] = {4,5,6};
>array r3[] = {7,8,9};
>call Matrix.AddArrayToRow ("A",0, getname(r1));
>call Matrix.AddArrayToRow ("A",1, getname(r2));
>call Matrix.AddArrayToRow ("A",2, getname(r3));
>call Matrix.Write("A");
>call Matrix.Delete("A");
```

See examples in **examples\MatrixVector**.

MN_Numerics

For using MathNet.Numerics.dll

Methods:

```
Matrix("matrix_name")(rows)(columns)           // like as Matrix.CreateDouble
Vector("vector_name")(length)                   // 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_array"s
// Creation Matrix.matrix
// Creation Vector.vector
import Matrix;
import Vector;
import MN_Numerics;
Matrix.DeleteAll();
Vector.DeleteAll();
write("-----Creation Matrix.matrix & Vector.vector-----");
call Matrix.Create("A",3,3,double);
array row1[] = {3,2,-1};
array row2[] = {2,-2,4};
array row3[] = {-1,0.5,-1};
call Matrix.AddArrayToRow("A",0,getname(row1)); // fill matrix
call Matrix.AddArrayToRow("A",1,getname(row2));
call Matrix.AddArrayToRow("A",2,getname(row3));

call Vector.Create("B",3,double);
array vector[] = {1,-2,0}; // create vector as "ppl_array"
call Vector.Add("B",getname(vector)); // fill vector
```

```
array X[length(vector)] = 0; // create result as "ppl_array"
call MN_Numerics.Solve("A","B",getname(X));
writearray X;
Vector.Delete("B");
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
```

DataFrame

DataFrame is a table with named columns, columns may be defined with different types, number of rows and columns is not limited.

Methods:

Create	SetWidthForAll
SetRow	SetColumn
Write	ClearColumns
Save	Read
InsertRows	AddRows
InsertColumns	AddColumns
RemoveRows	RemoveColumns
Sort	Reverse
SelectRows	UnselectRows

Delete DataFrame - **del df_name**

df_name – DataFrame name

1. Example

```
>import DataFrame
Imported [DataFrame]
>DataFrame.Create(MyDataFrame)(2)(One)(Two)(Three)
-----Variables and arrays-----
-N4 MyDataFrame [Node]
---N5 Settings [Node]
---N5 One [Array 2]
---N5 Two [Array 2]
---N5 Three [Array 2]
>DataFrame.Write(MyDataFrame)
```

```
One Two Three
0
1
```

DataFrame.Create creates node <df_name>.Settings:

```
>d MyDataFrame.Settings
-----Variables and arrays-----
-N5  Settings    [Node]
---L0 RowSelectedFrom
---L1 RowSelectedTo
---L2 PrintEmptyRows [yes]
---L3 RowsLength    [8]
---L4 ReallocIncrement [10]
---L5 CityType      [Text]
---L6 CityWidth     [12]
---L7 PopulationType [Number]
---L8 PopulationWidth [12]
---L9 AreaType      [Text]
---L10 AreaWidth    [12]
---L11 ElevationType [Text]
---L12 ElevationWidth [12]
---L13 CodeType     [Text]
---L14 CodeWidth    [12]
---L15 DensityType  [Text]
---L16 DensityWidth [12]
```

RowSelectedFrom and **RowSelectedTo** used in the following methods:

Read,Save,Write

PrintEmptyRows used in Write.

RowsLength set automatically as number of rows.

ReallocIncrement used in Create and AddRows.

<column name>**Type** set in code and used in Read and Sort.

<column name>**Width** set in code and used in Write.

Creates Dataframe:

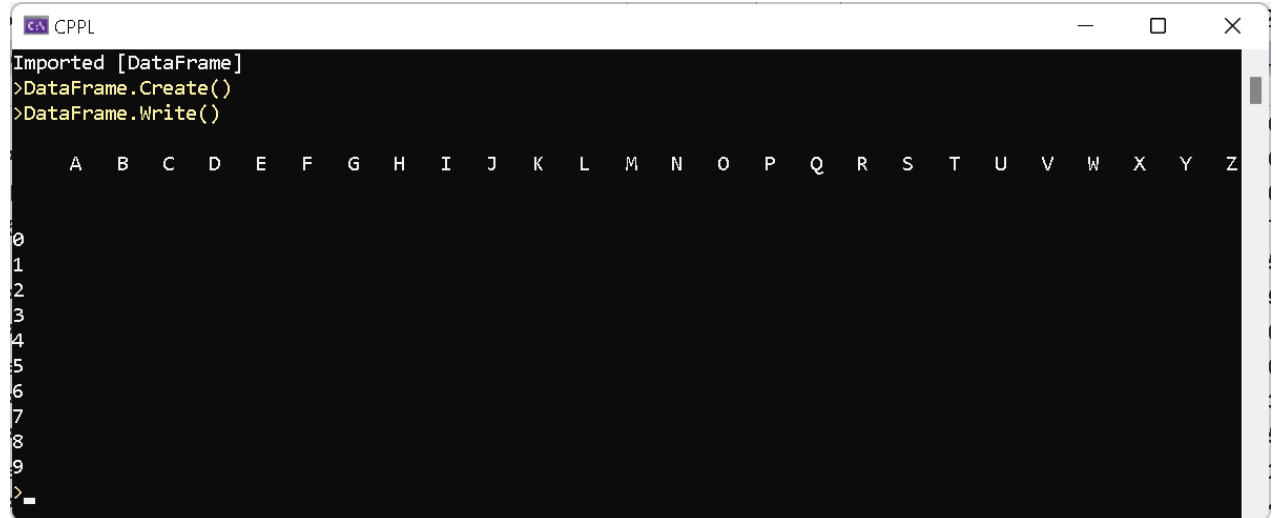
DataFrame.Create([df_name]) [(number rows)(column1)(column2)(column3)...

default df_name – DF

default number of rows – 10

default column names – from A to Z

Example:

A screenshot of a CPPL console window. The window title is "CPPL". The console output shows the following: "Imported [DataFrame]", ">DataFrame.Create()", ">DataFrame.Write()", followed by a table of 26 columns (A-Z) and 10 rows (0-9). The columns are labeled A through Z, and the rows are labeled 0 through 9. The table content is empty.

```
CPPL
Imported [DataFrame]
>DataFrame.Create()
>DataFrame.Write()

  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z
0
1
2
3
4
5
6
7
8
9
>
```

Sets same width for all columns to display DataFrame by DataFrame.Write:

DataFrame.SetWidthForAll([df_name])([width])

default name – DF

default with = 12

Example:

DataFrame.SetWidthForAll(14)

Sets value for each column for defined row:

DataFrame.SetRow(df_name)(row index)(value column1)(value column2)...

Sets value for each row for defined column:

DataFrame.SetColumn(df_name)(column)(value row1)(value row2)...

Displays DataFrame full contents or only contents of defined columns:

DataFrame.Write([df_name])([column)(column)...

To display contents of specific rows:

set <df_name>.Settings.RowSelectedFrom = value;

set <df_name>.Settings.RowSelectedTo = value | *;

"*" – to end

or by method SelectRows.

To set same width for all columns:

set <df_name>.Settings.<column name>Width = width

To enable or disable print empty rows:

```
set <df_name>.Settings.PrintEmptyRows = yes|no
```

Clears contents of all DataFrame or specified columns:

```
DataFrame.ClearColumns(df_name)[(column)(column)...
```

Save Dataframe full contents in format csv or [data](#) or contents of defined columns only in format .csv

```
DataFrame.Save(df_name)(filename.csv|.data)[(column)( column)]...
```

Read Dataframe full contents in format csv or [data](#) or contents of defined columns only in format .csv:

```
DataFrame.Read(df_name)(filename.csv|.data)[(column)( column)]...
```

To set type data for any column:

```
set <df_name>.Settings.<column name>Type = Text|Number
```

For each cell in column with type "Number" input data will be test, in case of error in cell contents will be written "***error**".

Inserts number of rows by index:

```
DataFrame.InsertRows(df_name)(index)[(number of rows)]
```

By default number of added rows= 1

Adds number of rows at the end:

```
DataFrame.AddRows([df_name])[ (number of rows)]
```

By default df_name = DF

By default number of added rows = 10

Insert one or several named columns before specified column:

```
DataFrame.InsertColumns(df_name) (specified column)(column)(column)...
```

Adds number of named columns at the end:

```
DataFrame.AddColumns(df_name)(column1)(column2)...
```

Removes rows:

RemoveRows(df_name) – remove all rows

RemoveRows(df_name)[(number_of_row)] – remove 1 row

RemoveRows(df_name)(number_from)(*) - remove rows
between number_from to end

RemoveRows(df_name)(number_from)(number_to) - remove rows
between number_from – number_end

Removes named columns and their Type and Width in Settings:

```
RemoveColumns(df_name)[(column1)(column2)...
```

Removes all columns and their Type and Width in Settings:

RemoveColumns(df_name)

Sorts all contents by specified column in ascending or descending order:

DataFrame.Sort(df_name)(ascend | descend)(column)

See examples df8.scr and df9.scr(wrong results for digital data when Type = Text).

Reverses all contents by specified column:

DataFrame.Reverse(df_name)(column)

Select one row only:

DataFrame.SelectRows(df_name)(select_from)

Select rows select_from – to end

DataFrame.SelectRows(df_name)(select_from)(*)

Select rows between select_from – select_to:

DataFrame.SelectRows(df_name)(select_from)(select_to)

Unselect rows:

DataFrame.UnSelectRows(df_name)

Examples:

```
Using data from DataFrame:
for(i,0,Length)
  write#("{0}    {1}",DF.City[i], DF.Area[i]);

write#("Average Elevation: {0}",
      Array.Average("DF.Elevation"));
```

Example:

File: examples\df\products.csv

Bagel,140,310,Medium

Biscuit ,86,480,High

Jaffa cake,48,370,Med-High

Bread white,96,240,Medium

Bread wholemeal,88,220,LowMed

Chapatis,250,240,Medium

Cornflakes,130,300,Med-High

Program: examples\df\df8.scr

```
import DataFrame;
```

```
DataFrame.Create(Products) (0) (Bread&Cereals) (Size) (per100grams)
                    (energy);
```

```
set Products.Settings.Bread&CerealsWidth = 20;
```

```
DataFrame.Read(Products) (examples\df\products.csv);
```

```
DataFrame.Write(Products);
```

```
DataFrame.Sort(Products) (ascend) (per100grams);
```

```
DataFrame.Write(Products);
```

```
>rc examples\df\df8.scr
code: scr
Info: [FuncImport] the library [DataFrame] is already loaded
DataFrame [Products] added [7] rows
-----Variables and arrays-----
-N4   Products      [Node]
---N5   Settings    [Node]
---N5   Bread&Cereals [Array 7]
---N5   Size        [Array 7]
---N5   per100grams  [Array 7]
---N5   energy      [Array 7]

      Bread&Cereals      Size      per100grams energy
0   Bagel                140        310        Medium
1   Biscuit              86         480         High
2   Jaffa cake           48         370        Med-High
3   Bread white          96         240        Medium
4   Bread wholemeal      88         220        LowMed
5   Chapatis             250        240        Medium
6   Cornflakes           130        300        Med-High

      Bread&Cereals      Size      per100grams energy
0   Bread wholemeal      88         220        LowMed
1   Bread white          96         240        Medium
2   Chapatis             250        240        Medium
3   Cornflakes           130        300        Med-High
4   Bagel                140        310        Medium
5   Jaffa cake           48         370        Med-High
6   Biscuit              86         480         High
>
```


Examples of using DataFrame methods in directory: **examples\df**

df1.scr - methods Read, Save

df2.scr - methods Read, RemoveColumns, Read, Save

df3.scr - methods AddColumns

df4.scr - methods DataFrame with default name DF

df5.scr - methods ClearColumns

df6.scr - methods RemoveRows, RemoveColumns

df7.scr - methods InsertRows, InsertColumns, SetRow, SetColumn, Sort, Reverse

df8.scr - methods Create, Read, Write, ascending and descending Sort, Reverse

df9.scr – methods Create, Read, Write, ascending Sort for Type = Text and Type Number

Statistics

For using MathNet.Numerics.dll

See details in -

<https://numerics.mathdotnet.com/api/MathNet.Numerics.Statistics/ArrayStatistics.htm>

Methods:

Covariance	FiveNumberSummary
GMean	HMean
Maximum	Mean
MeanStandardDeviation	Median
Minimum	OrderStatistic
Percentile	PopulationCovariance
PopulationStandardDeviation	PopulationVariance
QuantileCustom	Quantile
RanksInplace	RootMeanSquare
StandardDeviation	Variance

Formats:

```
Statistics.help([name])
Statistics.Covariance("\"sample1\")\"(\"sample2\")
Statistics.FiveNumberSummary(\"sample_name\")(\"result_array5\")
Statistics.GMean(\"array_ppl\")
Statistics.HMean(\"array_ppl\")
Statistics.Maximum(\"array_ppl\")
Statistics.Mean(\"array_ppl\")
Statistics.MeanStandardDeviation(\"array_ppl\")
Statistics.Median(\"array_ppl\")
Statistics.Minimum(\"array_ppl\")
Statistics.OrderStatistic(\"array_ppl\")(order)
Statistics.Percentile(\"array_ppl\")(selector)
Statistics.FuncCovariance(\"sample1\")(\"sample2\")
Statistics.PopulationStandardDeviation(\"array_ppl\")
Statistics.PopulationVariance(\"array_ppl\")
Statistics.QuantileCustom(\"array_ppl\")(tau)(definition)
Statistics.Quantile(\"array_ppl\")(tau)
Statistics.Ranks(\"array_ppl\")(\"rank_array_ppl\")(definition)
Statistics.RootMeanSquare(\"array_ppl\")
Statistics.StandardDeviation(\"array_ppl\")
Statistics.Variance(\"array_ppl\")
```

Example: (examples\Statistics\test.scr)

```
import Statistics;
recreate yes;
write#("Statistics Array Tests");

array sample1[] = {2.1,2.5,3.6,4};
array sample2[] = {8,10,12,14};
var covariance = Statistics.Covariance("sample1")("sample2");
write#("Covariance = {0}",covariance);

var population_covariance = Statistics.PopulationCovariance("sample1")("sample2");
write#("PopulationCovariance = {0}",population_covariance);

array arr[] = {5,9,3,1,7};
var gm = Statistics.GMean("arr");
var hm = Statistics.HMean("arr");
var msd = Statistics.MeanStandardDeviation("arr");

write#("GeometricMean = {0}",gm);
write#("HarmonicMean = {0}",hm);
write#("Mean, StandardDeviation = {0}",msd);

array arr[] = {3,2,5, 7, 6, 4, 6, 9,6, 8,7};
array FiveNumberSummary[5];
call Statistics.FiveNumberSummary("arr", "FiveNumberSummary");
writearray FiveNumberSummary;
write#("min = {0}",FiveNumberSummary[0]);
write#("low_quartile = {0}",FiveNumberSummary[1]);
write#("median = {0}",FiveNumberSummary[2]);
write#("upper_quartile = {0}",FiveNumberSummary[3]);
write#("max = {0}",FiveNumberSummary[4]);
write#("quartile_range = {0}",FiveNumberSummary[3] - FiveNumberSummary[1]);
write#();
var max = Statistics.Maximum("arr");
write#("max = {0}",max);

var min = Statistics.Minimum("arr");
write#("min = {0}",min);

var mean = Statistics.Mean("arr");
write#("mean = {0}",mean);

var order_statistic = Statistics.OrderStatistic("arr")(3);
write#("order_statistic = {0}",order_statistic);

var variance = Statistics.Variance("arr");
```

```
write#("variance = {0}",variance);

var population_variance = Statistics.PopulationVariance("arr");
write#("population_variance = {0}",population_variance);

var median = Statistics.Median("arr");
write#("median = {0}",median);

var percentile = Statistics.Percentile("arr")(50);
write#("percentile = {0}",percentile);

array arr[] = {5,9,3,1,7};
var standard_deviation = Statistics.StandardDeviation("arr");
write#("standard_deviation = {0}",standard_deviation);

var population_standard_deviation = Statistics.PopulationStandardDeviation("arr");
write#("population_standard_deviation = {0}",population_standard_deviation);

var quantile_customR1 = Statistics.QuantileCustom("arr")(0)(R1);
write#("quantile_customR1 = {0}",quantile_customR1);

var quantile_customR8 = Statistics.QuantileCustom("arr")(0)(R8);
write#("quantile_customR8 = {0}",quantile_customR8);

var quantile = Statistics.Quantile("arr")(0);
write#("quantile = {0}",quantile);

array ranks;
Statistics.Ranks("arr")(Average)("ranks");
writearray ranks;

var rms = Statistics.RootMeanSquare("arr");
write#("rms = {0}",rms);
```

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 **ulc.exe** creates code for User's DLL .

ulc.exe [config_file.json]

by default name of config file - ConfigULC.json

Examples:

1. creation code without classes:

```
ulc.exe ConfigULC.Array.json
```

File ConfigULC.Array.json:

```
{
  "name": "Array",
  "path": "",
  "functions":
  [
    "Max",
    "Min",
    "Sum",
    ...
  ]
}
```

Utility ulc.exe generates 2 files:

- **Array.cs** , this file must be added to user's dll project
- **Array.json** , this file must be added to directory **JsonHelp** (for Assistant.exe).

2. creation code with classes:

`ulc.exe ConfigULC.json`

File `ConfigULC.json`:

```
{
  "name": "Distributions",
  "path": "",
  "ulc_classes": [
    { "name": "Normal",
      "functions":
      [
        "Normal",
        "CDF",
        "Estimate",
        "InvCDF"
      ]
    },
    { "name": "StudentT",
      "functions":
      [
        "StudentT",
        "CDF",
        "InvCDF"
      ]
    },
    { "name": "ChiSquared",
      "functions":
      [
        "ChiSquared",
        "CDF",
        "InvCDF"
      ]
    }
  ]
}
```

Utility `ulc.exe` generates files:

- **Distributions.cs** - must be added to user's dll project
- **Normal.cs** - must be added to user's dll project
- **StudentT.cs** - must be added to user's dll project
- **ChiSquared.cs** - must be added to user's dll project
- **Distributions.json** - must be added to directory **JsonHelp** (for Assistant.exe).

Examples\scr\erato.scr

This samples significantly improves performance because all calculations are performed in Erato.DLL and data are saved by Vector.DLL.

```
// calculations in library Erato.dll
import Vector;
import Erato;
debugppl yes;
Vector.DeleteAll();

var m = 1000;
var len = m + 1;
array primes[len];
for (k,0,len)
    set primes[k] = k;
call Vector.Create("VPrimes",m + 1,int);
call Vector.Add("VPrimes",getname(primes));

call Erato.Solve("VPrimes");
var output = "";
var tmp;
for (i,0,len)
{
    set tmp = Vector.Get("VPrimes")(i);
    if (tmp != 0)
        set output = String.Concat(output)(tmp)(",");
}

var index = length(output) - 1;
set output = String.Remove(output)(index)(1);
writeline("{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");
call 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");
```

```
call 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
```

Open console window:

cmd

and run:

examples.bat

callfunc.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/>