This document briefly describes the most important code processing steps.

1.**Parsing & Traversal in scr-mode (Pre-translator)**

Input for parsing is script code from which previously removed commentaries, it is looked at for the presence of commands installed in the list of pre-translator commands, then for each command a code is prepared that ends with ';' or surrounded with '{ .. }'.
Part of code prepared like that are sent to function, that is intended for this or that statement,
Each part of code begins with command from the list of pre-translator commands (exception see p.5).
If part of input script code, ends with ';' does not contain any of pre-translator commands such code is considered as code in ppl-mode and passed as is.
AST prepared in this way is input for traversal, finally result of traversal is input for parsing in ppl-mode (see p.2).
This result (code in ppl-mode) may be saved in file for analysis.

2.**Parsing & Traversal in ppl-mode**

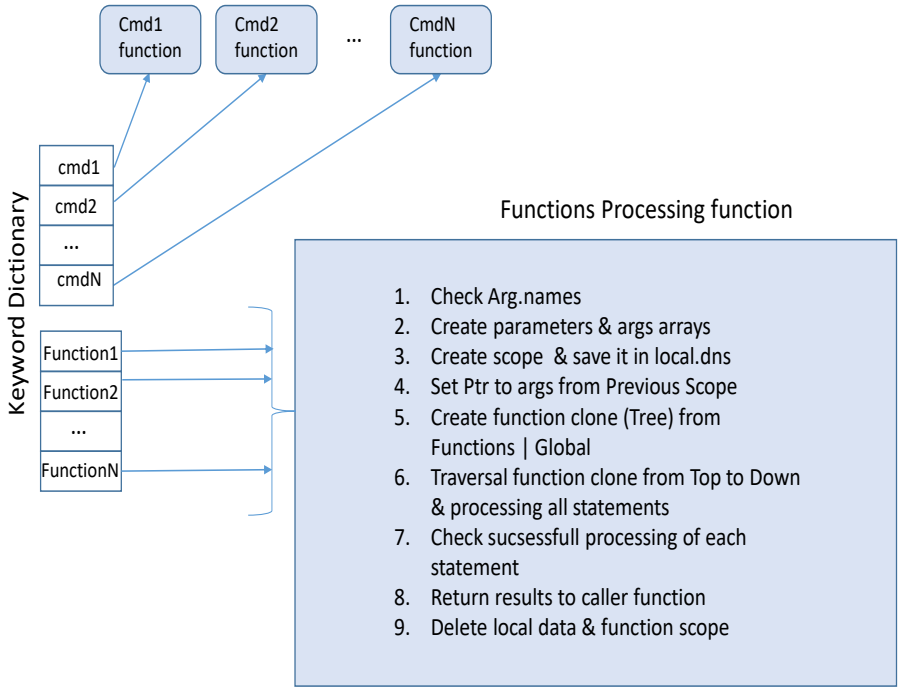
Script code is divided on statements (single or compound), each statement ends with ';'.
Each of these statements is input for parsing, which creates AST for th following traversal.
Traversal for Compound statements(**definestruct,function, for, if, else, switch, case, default**)
Is carried out from top AST to down.
Traversal for single statements is carried out from down AST to top, processing results are sent to stack of parent of statement tree and used as arguments to function, that is intended for this or that statement.

3. **Function creation**

There are following steps to create function from script and save it:

1. check function name on validity
2. check if function name exists in PrimaryOperatorList
3. create function clone of function tree
4. if function exists in Function | Global - delete
5. check function clone - parameters & function body
6. save function clone in Functions | Global
7. add function name to Keyword_Dictionary.

4. **Function processing Function**



5. **Operator set**

The PPL preprocessor converts scr-code to intermediate ppl-representation for immeadiate execution.
Format of statement for preprocessor is no different from normal:
operator <body><end symbol>
For example: var x = value;
Here: operator: var
body: x = 0
end symbol: ;

The set operator is a special case, this operator is used much more often than others, and in many programming languages it is implied but not written. Early versions of the PPL language required the mandatory inclusion of the set operator in the program code, its absence caused errors, so I decided to get rid of the set, that is, instead of
set x = 0;
to write
x = 0;

This process is carried out as follows:
The PPL preprocessor scans the code from left to right to "=" and inserts operators set or setkvp (set key value pair) in the appropriate places according to the following rules:
Symbol "=" appears in the following 2 groups of statements:
1. Conditional expressions: ==, <=, >=, !=
2. PPL operators: set, setkvp, array, sinit, ssetrow, sset,var,const, realloc.

Format of these operators is given below:

set name = value;
setkvp name = key,value;
array name[new length] = init_value
array name[] = { elem1,elem2,elemN…};
sinit name = init_value;
ssetrow name [ind1,ind2,indN…] = {elem1,elem2,elemN…};
sset name = init_value;
var name = value;
const name = value;
realloc array_name[new length] = init_value;

When symbol "= " is found, it checks whether it is an element of the first group, if yes, the search continues.
If not, it scans from right to left to the first end symbol (or to the beginning of the code) and this position is considered the beginning of the statement.

In this case, end symbol is a end symbol of both a simple sentence and a block one and has the following meanings: ; space tab { }).