3DPOLI COMPILER

Introduction

The purpose of 3DPoli Compiler is to define and preview structures. It has internal programming language, which is used for structure definition. Once the code is written, it must be compiled. After successful compilation the structure can be previewed in “3D Preview” tab. Also, the most important geometrical properties can be checked in “Structure Information” tab. Afterwards the compiled structure can be send to 3DPoli Fabrication part, which is responsible for hardware and fabrication.

For the moment 4 devices might be controlled with 3DPoli: Stages1 (3D), Stages2 (3D), Attenuator (1D, +power calibration), Shutter. Actual number of devices depends on your hardware configuration.

All commands can be found in Code Assistant group. Code Assistant works as a remainder of command syntax and might also decrease typing time.

3DPOLI PROGRAMMING LANGUAGE

Some facts

- The code is not case sensitive.
- Semicolon `;` is used to mark the end of command, but in most cases it can be ignored.
- There is only one type of variables in 3DPoli (8 bit floating point).
- Point `.` works as decimal separator and comma `,` - as a parameter separator.

Units

- Position [µm]
- Speed [µm/s]
- Time [ms]
- Power [mW]
- Angle [radians]

Comments

Comments are special symbols, which tell the compiler to ignore some part of the code. There are two types of comments in 3DPoli:

- Line comments: `//` (preferred) or `#` (for backwards compatibility). The code after these symbols will be ignored until end of the line.
- Block comments: `{ }` The code between these brackets will be ignored.

Parameters

There are the following types of parameters:

- ***Number*** – number, math expression, variable or function

- ***$Variable*** – variable name. All variables must have `$` symbol before the name. Variable name must start with a letter. For instance `$MyVariable`, `$x`, `$x5`.
- ***Procedure name –*** Procedure names must start with a letter, for instance `MyProc`.
- ***Axes*** – Combination of axes. There are 8 possible axes names in 3DPoli:
  - ❖ X, Y, Z – Axes of Stages1
  - ❖ A, B, C – Axes of Stages2
  - ❖ P – Attenuator axis
  - ❖ W – Power calibrated axis

  You can combine axes by writing them together. For instance `XZ`, `BC`. Order of axes names has no influence (`XYZ` = `YZX`).
- ***Stages –*** one of the following:
  - ❖ `XYZ` – Stages1
  - ❖ `ABC` – Stages2
  - ❖ `PW` or `P` – Attenuator
- ***"Text"*** – text parameters must be surrounded by `""` quotes. For the moment only STL command requires string parameter.

HARDWARE CONTROL COMMANDS

➢ LINE AND MOVE

Moves given axes. A and R denotes absolute and relative movements respectively.

```
MoveA(Axes, Pos1, [Pos2], [Pos3])
MoveR(Axes, Pos1, [Pos2], [Pos3])
LineA(Axes, Pos1, [Pos2], [Pos3])
LineR(Axes, Pos1, [Pos2], [Pos3])
```

Axes – list of axes with you want to move. Only axes of the same device can be moved in a single command. Number of axes determines number of position parameters.
Pos1, Pos2, Pos3 – new positions of axes for absolute or position increments for relative commands.

Movement velocities can be set with `setvel` (for line) and `setmovevel` (for move) commands. Historically Line commands were used as mark (writing structure) and Move as jump (shutter closed) commands. Nowadays there is no difference between these commands, however for code clarity it is still recommended to keep this practice. None of these commands affect the shutter, use `shutter` command for shutter control.
Note: By default only movements with opened shutter are displayed in 3D Preview. If you want to see all movements, press related buttons in Display tab.

➢ SHUTTER

Changes the state of the shutter.

```
Shutter(State)
```

State – if zero the shutter will be closed, otherwise – opened.

➢ HOME COMMANDS

```
SetHome(Axes)
ClearHome(Axes)
```

Set or clears Home position. After SetHome command, the current axis position is set as zero and all the following absolute positions will be calculated in respect to home position. ClearHome restores home position to zero position.

- **VELOCITY COMMANDS**
  Sets velocity of Stages. SetVel is for Line and SetMoveVel is for Move commands.

```
SetVel(Stages, Velocity)
SetMoveVel(Stages, Velocity)
```

Stages – specifies the affected stages.
Velocity – New velocity value.

- **PAUSE COMMAND**
  Adds pause of given duration.

```
Pause(Duration)
```

KEYWORDS

- **Define variable**

```
dvar($MyVar1, [$MyVar2, ...])
```

Defines variables and initializes their value to 0. You can not define two variables with the same name, unless they are defined in different procedures. Also, you can not use this command inside cycles or `if` condition.

- **Cycles**

For Cycle

```
for($CycleVar, From, To, By)
  //your code
end;
```

From value is assigned to Cycle Variable and the code inside for statement is looped. In each iteration ***By value*** is added to Cycle variable. Cycle finishes when Cycle Variable reaches ***To*** value. If ***To*** is smaller than ***From*** than ***By*** value is substracted. The sign of ***By*** value does not matter.

While Cycle

```
while(condition)
  //your code
end;
```

The code inside while cycle is looped until ***condition*** is not zero. At some point you must change ***condition*** to zero, otherwise it will be infinite. Note, that ussually condition is a logical sentence, like `while($x < 100)`.

- **Condition sentence**

```
if(condition)
  //your code
[else]
  //your code
end;
```

If ***condition*** is nonzero than code between ***if*** and ***else*** is executed. Otherwise the code between ***else*** and ***end*** is executed. ***Else*** statement is optional and can be ommited. In such case if ***condition*** is not satisfied the code inside ***if*** sentence is ignored.

- **Procedures**

```
procedure(Name, [$param1, $param2, ...])
  //your code
end;


RunProc(Name, [param1, param2, ...]);
```

Procedures are blocks of the code which are executed with `RunProc` command only. Every procedure must have unique name, unless they are defined in different procedures. Procedures inside procedures are allowed. Recursion algorhytms are also possible (procedure call to itself).

Parameter variables are optional. If defined, Runproc must be called with the same number of parameters.

If RunProc parameters are variables having a star `*` in front, procedures can assign new values to their calling parameters. For example:

```
Procedure(Add, $X, $Y, $Result)
  $Result := $X + $Y;
end;

dvar($Sum);

RunProc(Add, 2, 2, $Sum); //nothing happens, $Sum = 0
RunProc(Add, 2, 2, *$Sum);//$Sum = 4
```

FUNCTIONS

Functions – return numerical values and must be used inside parameters.

`Pi`	π constant (3.141...)
`E`	Exponent (2.718...)
`Sin(angle)`	Sine
`Cos(angle)`	Cosine
`Tan(angle)`	Tangent
`ASin(value)`	Arcsine
`ACos(value)`	Arccosine
`ATan(value)`	Arctangent
`ln(value)`	Natural logarhythm
`Sqrt(value)`	Square root
`Round(value)`	Round the value
`Trunc(value)`	Whole part of the value
`GetShutterState`	Returns shutter state (0 or 1)
`GetPos(axis)`	Returns position of axis
`GetHomePos(axis)`	Returns home position of axis

OPERATORS

`+`	addition	$x := 2 + 2;
`-`	substraction	$x := 2 - 2;
`*`	multiplication	$x := 2 * 2;
`/`	division	$x := 2 / 2;
`^`	power	$y := $x^2; $y := $x^0.25;
`=`	Is equal?	$y := $x = 100; if($y = 1);
`<>`	Is not equal?	if($x <> 100)
`>`	Is bigger?	if($x > 100)
`>=`	Is bigger or equal?	if($x >= 100)
`<`	Is smaller?	while($x < 100)
`<=`	Is smaller or equal?	while($x <= 100)
`not`	Logical NOT	$no := not $yes;
`or`	Logical OR	$yes := $maybe1 or $maybe2;
`and`	Logical AND	$yes := $maybe1 and $maybe2;
`xor`	Logical XOR	$yes := $maybe1 xor $maybe2;
`div`	Integer division	$NumCells := $width div $period
`mod`	Remainder	$one := 5 mod 2; //$one = 1
`:=`	Assigns new value to variable	$x := NewValue;

STL

Stl file is a standard litography file format. These files can be created with number of graphical programs. There are two types of stl files: binary and ascii. For the moment only binary stls are supported. Ascii stls can be converted to binary with third party software, like MeshLab.

`STL(Stages, "FileName", RefX, RefY, RefZ, SX, SY, SZ, RotX, RotY, RotZ)`

Stages – Sets stages for fabrication (stages1 or stages2).
"FileName" – STL file name
RefX, RefY, RefZ – reference point position. Use ***rMin***, ***rMid***, ***rMax*** constants to define reference point.
SX, SY, SZ – Sets the size or scaling of STL model. If value is positive than it defines model size in that direction. If value is negative than it defines scaling of the model in that direction in respect to (a) original model if all three parameters are negative, (b) other axes if at least one parameter is non negative.
RotX, RotY, RotZ – sets the angle of rotation about the X/Y/Z axis.

`SetSlicing(dZ, Partial?, PartialFrom, PartialTo)`

Sets slicing parameters.
dZ – Slicing step. Sign of dZ determines slicing direction. If positive, than model is sliced from minimum towards maximum and vice versa.
Partial? – tells if partial slicing should be used. If value is nonzero, model is sliced only in the range from ***PartialFrom*** to ***PartialTo***.

`SetHatching(Style, dXY, ConnectLines?, Optimization?)`

Sets hatching parameters.
Style – sets the way of hatching. Use one of the following values:

- hsNone – No hatching needed
- hsAutoXY – Hatching direction is chosen automatically with minimum number of shutter actions
- hsX – Hatch along X axis only
- hsY – Hatch along Y axis only
- hsXY – Every second slice is hatched along X and the next one – along Y axis
- hsXYbcc – Model is hatched according to bcc (body centered cubic) geometry
- hsXYfcc – Model is hatched according to fcc (face centered cubic) geometry

`SetContours(Style)`

Sets contours style.
Style – Use one of the following:

- csNone – No contours will be made
- csBeforeH – Contours are drawn before hatching
- csAfterH – Contours are drawn after hatching