

LDraw Script Syntax

CONTENTS

- [VARIABLES AND ARRAYS](#)
 - [INDIRECT ADDRESSES](#)
 - [COMMANDS](#)
 - [????.DAT](#)
 - [COND / CONDITIONAL - NOCOND / NOCONDITIONAL](#) (not supported yet)
 - [COPY](#)
 - [CROSS - NOCROSS](#)
 - [ECHO / EXTRUDE](#)
 - [ELSE](#)
 - [END](#)
 - [END IF / ENDIF](#)
 - [FILL](#)
 - [FOR-TO-STEP](#)
 - [GOSUB-RETURN](#)
 - [GOTO / GO TO](#)
 - [IF-THEN-ELSE](#)
 - [LET](#)
 - [LINE](#)
 - [MIRROR](#)
 - [NEXT](#)
 - [PIPE](#)
 - [PLOT](#)
 - [POINT](#)
 - [PRINT / ?](#)
 - [REM](#)
 - [RETURN](#)
 - [RING](#)
 - [SEGMENT / SEGM](#)
 - [WHILE-WEND](#)
 - [LDRAW LINES](#)
 - [CONSTANTS](#)
-

VARIABLES AND ARRAYS

A-Z No multi-lettered names allowed

X0-X99, Y0-Y99, Z0-Z99

P0-P99 represents a 3-D points.

P0 is X0 Y0 Z0

P1 is X1 Y1 Z1

...

P99 is X99 Y99 Z99

Note: P without number is just a variable, not representing X Y Z

Note 2: The function PLOT is not yet able to handle variables

INDIRECT ADDRESSES

Example:

```
p1 5 7 9
p2 6 8 10
p3 11 12 13
for i=1 to 3
  p(i+10) = z(i) y(i) x(i) * 1 1 -1
  2 24 p(i) p(i+10)
next i
```

makes:

```
2 24 5 7 9 9 7 -5
2 24 6 8 10 10 8 -6
2 24 11 12 13 13 12 -11
```

Note: The PRINT command is not yet able to handle indirect addressing

COMMANDS

***.DAT

Puts a .dat line in default alignment [and colour]

Examples:

```
stud.dat 80 0 40
```

makes:

```
1 16 80 0 40 1 0 0 0 1 0 0 0 1 STUD.DAT
```

```
3001.dat BLUE 0 0 0
```

makes:

```
1 1 0 0 0 1 0 0 0 1 0 0 0 1 3001.dat
```

COPY

Copies a 3d point

Example:

```
COPY P1 P2
( equals to X2=X1 Y2=Y1 Z2=Z1)
```

CROSS - NOCROSS

Determines whether a cross-line (2 24...) between an ECHOed POINT and its line echo should be drawn.

If CROSS is set outside LINE mode, it will be set globally.

If CROSS is set in LINE mode, it will be set locally, ie until LINE END.

If CROSS is set at the end of a POINT, it will only

Example:

```
ECHO 0 20 0
CROSS
LINE
  POINT 0 100 0
  POINT 0 200 0 NOCROSS
LINE END
```

makes:

```
2 24 0 100 0 0 120 0 - first point CROSSed
2 24 0 100 0 0 200 0 - original points LINEed
2 24 0 120 0 0 220 0 - ECHOed points LINEed
4 16 0 100 0 0 200 0 0 220 0 0 120 0 - space in between FILled
(second point not CROSSed)
```

Default value for CROSS is ON.

Setting CROSS mode resets COND mode.

ECHO / EXTRUDE

Turns 2-d lines and edges into 3-d

Examples:

```
ECHO 0 24 0
2 24 100 0 0 200 0 0
1 16 40 0 40 10 0 0 0 0 0 10 4-4edge.dat
ECHO OFF
```

makes:

```
2 24 100 0 0 200 0 0
2 24 100 24 0 200 24 0
4 16 100 0 0 100 24 0 200 24 0 200 0 0
1 16 40 0 40 10 0 0 0 0 0 10 4-4edge.dat
1 16 40 24 40 10 0 0 0 0 0 10 4-4edge.dat
1 16 40 0 40 10 0 0 0 24 0 0 10 4-4edge.dat
```

END (Optional)

Terminates script interpretation, disregarding succeeding lines.

END IF / ENDIF

Ends a multi-line IF-THEN[-ELSE] statement.

FILL - NOFILL

Determines whether the space between an ECHOed line and its echo should be filled.

If FILL is set outside LINE mode, it will be set globally.

If FILL is set in LINE mode, it will be set locally, ie until LINE END.

Default value for FILL is ON.

FOR-TO-STEP

Examples:

```
for i=1 to 15
for x=-200 to 200 step 20
for x=10 to 1 step -1
```

See also [NEXT](#)

GOSUB-RETURN

Syntax: GOSUB [:]<label>

Example:

```
for i=1 to 5
  gosub :hello
next i
end

:hello
  print 0 "Hello";
  gosub world
return

:WORLD
  print "World."
return
```

GOTO / GO TO

Syntax: GOTO [:]<label>

Example:

```
:DACAPO
PRINT "0 Don't try this at home!"
GOTO DACAPO
```

IF-THEN-ELSE

Examples:

```
if i>10 then i=1
if a=20 then goto :FINISH
```

Not allowed:

```
if i>10 then i=1 else i=i+1
```

(Why? ELSE must be used in multi-line statements, see below)

Corrected:

```
if i>10 then
  i=1
else
  i=i+1
end if
```

See also [END IF](#) , [GOTO](#)

LET

Examples:

```
LET A=0
b=a*2
b = 2*a
c=-.5/3
p15 = 100 24 60
```

Not allowed:

```
b = 2 * a
```

(Why? "2 * a" will be interpreted as three arguments)

LINE

Syntax: LINE [CROSS | NOCROSS] [FILL | NOFILL] [END | OFF]

Example:

```
LINE
POINT 10 20 30
POINT 40 50 60
70 80 90
LINE END
```

makes:

```
2 24 10 20 30 40 50 60
2 24 40 50 60 70 80 90
```

MIRROR

Syntax: MIRROR 0 | OFF | [+]X | [+]Y | [+]Z | -X | -Y | -Z
MIRROR 0 or MIRROR OFF

turns all mirroring off
MIRROR -X
 turns x mirroring off
MIRROR +X or MIRROR X
 turns x mirroring on

NEXT

Examples:

```
NEXT
```

```
Next i
```

(Index variable optional - anyway, it's ignored by compiler)

PLOT

Syntax: **PLOT @Pnn** [sectors totalsectors] x y z a b c d e f g h i

Plots out the coordinates of a circle or part of a circle, and puts the the table back into the .LDS source file.

Example:

```
PLOT @P20 4 16 0 0 0 10 0 0 0 0 10 0 0 0
```

changes the .LDS file to:

```
'PLOT @P20 4 16 0 0 0 10 0 0 0 0 10 0 0 0
```

```
P20 10 0 0
```

```
P21 9.239 3.827 0
```

```
P22 7.071 7.071 0
```

```
P23 3.827 9.239 0
```

```
P24 0 10 0
```

Note: The function PLOT is not yet able to handle variables

PIPE

Syntax: **PIPE** [sectors totalsectors] [colour]

Stitches circles or part of a circles into a "pipe".

Defaults: sectors=16, totalsectors=16, colour=16

See also [SEGMENT](#)

POINT

Syntax: **[POINT]** x y z

Example:

```
ECHO 0 10 0
```

```
POINT 1 2 3
```

makes (in CROSS mode):

```
2 24 1 2 3 1 12 3
```

Not allowed:

```
LINE
...
LINE OFF
ECHO 0 10 0
1 2 3
```

(Why? Writing POINT is optional only in LINE mode)

See also [LINE example](#)

PRINT or ?

Syntax: PRINT | ? ["<text>" | variable | constant] [;]

; will prevent line feed. **NOTE:** Semicolon is only allowed as the last character in an LDS PRINT command line.

Example:

```
P1 11 22 33.33
PRINT "0 TRANSLATE" p1
3001.dat RED 100 0 0
PRINT "0 TRANSLATE";
PRINT "OFF"
```

makes:

```
0 TRANSLATE 11 22 33.33
1 4 100 0 0 1 0 0 0 1 0 0 0 1 3001.dat
0 TRANSLATE OFF
```

which **LDLite 1.5 and newer - not LDraw!** - will interpret as:

```
1 4 111 22 33.33 1 0 0 0 1 0 0 0 1 3001.dat
```

NOTE: PRINT is currently not ECHOed or MIRRORed, maybe it should be?

REM or '

will be ignored by compiler.

RING

Syntax:

```
RING [segments total_segments] color1 color2 color3 x y z
ai ci di fi gi ii ao co do go io
```

NOTE: The command consists of two lines.

segments and total_segments are optional. If omitted, default values are 16/16.

total_segments values accepted: 16 for lo-res circles and 48 for hi-res circles.

color1 is color of inner disc,

color2 is color of ring,

color3 is color of outer n-disc.

color1 and/or color3 can be set on a negative value. Thus, disc will not be drawn.

Example:

You have the two following yellow areas,

```
1 14 100 0 0 7 0 0 0 0 0 0 5 2-4disc.dat
1 14 100 0 0 9 0 0 0 0 0 0 7 2-4ndis.dat
( "y-values"      ^      ^      ^      )
```

and want the space between them filled with black.

```
RING 8 16 14 0 14 100 0 0
7 0 0 0 0 5 9 0 0 0 0 7
```

NOTE: Since discs and rings are two-dimentional, I have omitted the "y-values".

Those are the 2nd, 5th, and 7th arguments in the rotations matrix and always equal to 0.

SEGMENT

Syntax: SEGM|SEGMENT x y z a b c d e f g h i

Plots a slice in the [PIPE](#) function

WHILE-WEND

Example: See [GOSUB example](#).

LDRAW LINES

0 Remarks and beta-commands will just be copied as-is

1-5 Variables will be replaced by their values.

CONSTANTS

BLACK = 0

BLUE = 1

GREEN = 2

DARKCYAN = 3

RED = 4

MAGENTA = 5

BROWN = 6

GRAY or GREY or LIGHTGRAY or LIGHTGREY = 7

DARKGRAY or DARKGREY = 8

LIGHTBLUE = 9

LIGHTGREEN = 10

CYAN or LIGHTCYAN = 11

LIGHTRED = 12

PINK = 13

YELLOW = 14

WHITE = 15
CLEARBLUE = 33
CLEARGREEN = 34
CLEARRED = 36
CLEARLIGHTBLUE = 41
CLEARYELLOW = 46
CLEAR or TRANSPARENT = 47
GOLD = 334
TAN or SAND = 382
CHROME or SILVER = 383
MINTGREEN = 431
ELECTRIC = 494
LIGHTYELLOW = 495

[Back to the LDS Page](#)

[Back to Tore's Home Page](#)