# state_objects

March 30, 2024

## 1 objects for changing *state*

the pre-defined templates for objects in a FullControl ***design***, which describes changes to ***state***, are demonstrated here

after creating a FullControl ***design***, it is transformed into a ***result*** (e.g. gcode or a 3D plot preview). this notebook is focused on ***designs*** that are transformed into 'gcode' or 'plot' ***results*** - both types are already built into FullControl and have default settings. gcode can be formatted for a selection of printers and the 3D plot is implemented in plotly, but the range of printers is intended to be extended along with plotting software options

more details of the transformation methods are given in separate notebooks (gcode, plot)

some changes to ***state*** affect both ***results*** (e.g. x y z values of Points); some only affect the 'plot' ***result*** (e.g. color of Points); some only affect the 'gcode' ***result*** (e.g. fan speed).

when the design is being transformed into a gcode ***result***, design intentions that cannot be expressed by existing templates for ***state***-change objects can be defined using a ManualGcode object. information about adding new templates will be described in future documentation aimed at python developers

*<this document is a jupyter notebook - if they're new to you, check out how they work: link, link, link>*

*run all cells in this notebook in order (keep pressing shift+enter)*

[ ]:
```
import fullcontrol as fc
```

## 2 1. generic objects

the objects in this section are not specific to one ***result***-type, they may influence multiple different ***results***

### 2.1 Point

Point classes describe where the printhead should move to with x y z attributes

[ ]:
```
steps = []
steps.append(fc.Point(x=0,y=0,z=0))
steps.append(fc.Point(x=10,y=0,z=0))
steps.append(fc.Point(x=10,y=10,z=0))
```

```
print(fc.transform(steps, 'gcode'))
```

it is not necessary to define x, y and z - you can define only the coordinates that change - but the first point should have all of x y and z defined

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.Point(x=10))
steps.append(fc.Point(y=10))
print(fc.transform(steps, 'gcode'))
```

the color attribute of points can be defined for a 'plot' ***result***

color is described by a list [r, g, b] with values from 0 to 1

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0, color=[0,0,1]))
steps.append(fc.Point(x=50, y=5, color=[0,1,1]))
steps.append(fc.Point(x=0, y=10, color=[1,0,1]))
fc.transform(steps, 'plot', fc.PlotControls(color_type='manual'))
```

similar to the x y z point attributes, you only need to define the color attribute if it changes

to get an instant change in color, as opposed to a transition between two points, add a Point object with no change to x y z but a change to color

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0, color=[0,0,1]))
steps.append(fc.Point(x=25, y=5))
steps.append(fc.Point(x=0,y=10))
steps.append(fc.Point(y=30, color=[0,1,1])) # line plotted with a gradient␣
 ↪transition to this color
steps.append(fc.Point(x=25, y=40))
steps.append(fc.Point(x=0, y=50))
steps.append(fc.Point(color=[0,0,1])) # line plotted with an instant color␣
 ↪change
steps.append(fc.Point(y=60))
fc.transform(steps, 'plot', fc.PlotControls(color_type='manual'))
```

## 2.2 Fan

speed is set as a percent, which is converted to the range 0-255 in gcode

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.Point(x=5))
steps.append(fc.Fan(speed_percent=50))
steps.append(fc.Point(x=10))
steps.append(fc.Fan(speed_percent=0))
print(fc.transform(steps, 'gcode'))
```

### 2.3 Buildplate

the temperature of the buildplate can be set along with an instruction as to whether the printer should wait for the desired temperature to be reached before continuing

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.Point(x=5))
steps.append(fc.Buildplate(temp=80, wait=False))
steps.append(fc.Point(x=10))
steps.append(fc.Buildplate(temp=80, wait=True))
print(fc.transform(steps, 'gcode'))
```

### 2.4 Hotend

the temperature of the hotend is set along with an instruction to say whether the printer should wait for the desired temperature to be reached before continuing

the tool can also be stated for multitool printers

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.Point(x=5))
steps.append(fc.Hotend(temp=280, wait=True))
steps.append(fc.Point(x=10))
steps.append(fc.Hotend(temp=170, wait=False, tool=3))
print(fc.transform(steps, 'gcode'))
```

### 2.5 Printer

used to set print speed and travel speed (for non-extruding movements)

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.Point(x=5))
steps.append(fc.Point(x=10))
steps.append(fc.Printer(print_speed=500, travel_speed=2000))
steps.append(fc.Point(x=15))
steps.append(fc.Point(x=20))
steps.append(fc.GcodeComment(text="'F500' is not included in the gcode line␣
 ↪immediately above since the printer remembers it from the previous line"))
print(fc.transform(steps, 'gcode'))
```

### 2.6 ExtrusionGeometry

set the geometry of the extruded material, which is used to calculate the value of E in gcode

the 'area_model' attribute controls how the cross-sectional area of the extrudate is defined. it can be set to 'rectangle' (default), 'stadium', 'circle' or 'manual'

a 'stadium' is a rectangle with a semi-circle at each end

if 'area_model' == 'rectangle' or 'stadium', width and height must be defined

in some cases, cylindrical extrudates are expected, in which case area_model='circle' is logical and diameter must be defined

to manually set the cross-sectional, set area_model='manual' and set the 'area' attribute as desired

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.ExtrusionGeometry(area_model='rectangle', width=0.8, height=0.
 ↪2))
steps.append(fc.Point(x=10))
steps.append(fc.ExtrusionGeometry(width=0.4))
steps.append(fc.Point(x=20))
print(fc.transform(steps, 'gcode'))
```

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.Extruder(units='mm3'))  # set extrusion units to mm3 to make it␣
 ↪easier to manually calculate E values
steps.append(fc.ExtrusionGeometry(area_model='rectangle', width = 1, height = 0.
 ↪2))
steps.append(fc.Point(x=1))
steps.append(fc.GcodeComment(end_of_previous_line_text='     E = length (1 mm)␣
 ↪* width (1 mm) * height (0.2 mm) = 0.2 mm3'))
steps.append(fc.ExtrusionGeometry(width=0.5))
steps.append(fc.Point(x=2))
steps.append(fc.GcodeComment(end_of_previous_line_text='     width halved, but␣
 ↪length and height remained the same, so E halved'))
steps.append(fc.ExtrusionGeometry(area_model='circle', diameter=1))
steps.append(fc.Point(z=10))
steps.append(fc.GcodeComment(end_of_previous_line_text='     print a z-pillar.␣
 ↪area_model = circle. E = length (10 mm) * pi (3.14) * (d^2)/4 (1*1/4=0.25) =␣
 ↪7.85 mm3'))
steps.append(fc.ExtrusionGeometry(area_model='manual', area=2))
steps.append(fc.Point(z=20))
steps.append(fc.GcodeComment(end_of_previous_line_text='     area_model =␣
 ↪manual. E = length (10 mm) * area (2 mm2) = 20 mm3'))
print(fc.transform(steps, 'gcode'))
```

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0.5))
steps.extend([fc.ExtrusionGeometry(width=1, height=0.5), fc.
 ↪PlotAnnotation(label='W1 H0.5'), fc.Point(x=5)])
steps.extend(fc.travel_to(fc.Point(x=0, y=-2)))
steps.extend([fc.ExtrusionGeometry(area_model = 'circle', diameter = 0.5), fc.
 ↪PlotAnnotation(label='Circle dia 0.5'), fc.Point(x=5)])
steps.extend(fc.travel_to(fc.Point(x=0, y=-4)))
```

```
steps.extend([fc.ExtrusionGeometry(area_model = 'circle', diameter = 1), fc.
 ↪PlotAnnotation(label='Circle dia 1'), fc.Point(x=5)])
steps.extend(fc.travel_to(fc.Point(x=0, y=-6)))
steps.extend([fc.ExtrusionGeometry(area_model = 'manual', area = 0.5), fc.
 ↪PlotAnnotation(label='Manual area 0.5'), fc.Point(x=5)])
steps.extend(fc.travel_to(fc.Point(x=0, y=-8)))
steps.extend([fc.ExtrusionGeometry(area_model = 'manual', area = 1), fc.
 ↪PlotAnnotation(label='Manual area 1'), fc.Point(x=5)])
fc.transform(steps, 'plot', fc.PlotControls(style="tube",␣
 ↪color_type='print_sequence', tube_type='cylinders', tube_sides=8))
```

### 2.7 Extruder

the 'on' attribute of an Extruder object is used to turn extrusion on or off - it defaults to True if not set

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0.2))
steps.append(fc.Point(x=25))
steps.append(fc.Extruder(on=False))
steps.append(fc.Point(x=0, y=5))
steps.append(fc.Extruder(on=True))
steps.append(fc.Point(x=25))
fc.transform(steps, 'plot')
```

other potential attributes of an Extruder are - 'units': units of E in gcode ('mm' or 'mm3') - 'dia_feed': diameter of the feedstock filament (mm) - 'relative_gcode': E values in gcode are relative (see 'M83' in general gcode documentation) if set to True

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.Point(x=5))
steps.append(fc.Point(x=10))
steps.append(fc.Extruder(dia_feed=2.85))
steps.append(fc.GcodeComment(text='dia_feed changed from default 1.75 to 2.85␣
 ↪mm: E value changes'))
steps.append(fc.Point(x=15))
steps.append(fc.Point(x=20))
steps.append(fc.Extruder(units='mm3'))
steps.append(fc.GcodeComment(text='units changed from default mm to mm3: E␣
 ↪value changes'))
steps.append(fc.Point(x=25))
steps.append(fc.Point(x=30))
steps.append(fc.Extruder(relative_gcode=False))
steps.append(fc.GcodeComment(text='relative_gcode changed from default True to␣
 ↪False: E value increases incrementally'))
steps.append(fc.Point(x=35))
```

```
steps.append(fc.Point(x=40))
steps.append(fc.Point(x=45))
steps.append(fc.Point(x=50))
print(fc.transform(steps, 'gcode'))
```

### 2.8 StationaryExtrusion

extrude material from the nozzle without moving in XYZ

volume is defined in mm^3 - note the number for E in gcode will not be equal to volume for most printers - but FullControl does the unit conversion for you

speed is dependent on your printer but will most likely be - *units_of_E_for_your_printer* / minute

[ ]:
```
steps = []
steps.append(fc.Extruder(on=False))
steps.append(fc.Extruder(units='mm3'))
steps.append(fc.Point(x=10, y=10, z=2))
steps.append(fc.StationaryExtrusion(volume=5, speed=50))
steps.append(fc.Point(x=20, y=10, z=2))
steps.append(fc.StationaryExtrusion(volume=5, speed=100))
print(fc.transform(steps, 'gcode'))
```

### 2.9 PrinterCommand

to allow FullControl ***designs*** to create 'gcode' ***results*** for various printers, each printer has a list of commands that can be called to change ***state***

a few demo commands are automatically included for each Printer. the most likely to be used during design are: - "retract" = "G10 ; retract" - "unretract" = "G11 ; unretract"

each command has an 'id' which can be memorable and easy to include in a ***design*** than a ManualGcode object

to call a command, the PrinterCommand object is used

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.Point(x=10))
steps.append(fc.Extruder(on=False))
steps.append(fc.PrinterCommand(id='retract'))
steps.append(fc.Point(x=0, y=2))
steps.append(fc.Extruder(on=True))
steps.append(fc.PrinterCommand(id='unretract'))
steps.append(fc.Point(x=10))
print(fc.transform(steps, 'gcode'))
```

new commands can be added using the extend_commandlist attribute of a Printer object

information about changing the command list permanently will be provided in future documentation about adding new printers

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.Printer(new_command={'pause': 'M601 ; pause print'}))
steps.append(fc.Point(x=10))
steps.append(fc.PrinterCommand(id='pause'))
steps.append(fc.Point(x=20))
steps.append(fc.PrinterCommand(id='pause'))
steps.append(fc.Point(x=30))
steps.append(fc.PrinterCommand(id='pause'))
print(fc.transform(steps, 'gcode'))
```

# 3 2. *result*-specific objects

the objects in this section target a specific ***result***

this is done for convenience but could change in the future. E.g. an alternative approach for 'GcodeComment' and 'PlotAnnotation' (both described below) would be to have a single 'annotation' object that applies a comment to gcode and written text to a plot, but that approach isn't implemented in this release to allow the objects to be more easily tailored to the specific type of ***target***

the objects that target the 'plot' ***result*** are ignored when the ***design*** is transformed to a 'gcode' ***result*** and vice versa

## 3.1 GcodeComment

a comment can be added to the gcode as a new line or as an addition to the previous line of gcode

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.GcodeComment(text='comment as a new line of gcode'))
steps.append(fc.Point(x=10))
steps.append(fc.GcodeComment(end_of_previous_line_text='comment added to the␣
 ↪end of the previous line of gcode'))
print(fc.transform(steps, 'gcode'))
```

## 3.2 ManualGCode

if design intentions cannot be expressed by the above objects to control ***state***, the ManualGcode object can be used to insert gcode at any point in the ***design***

CAUTION: this is a bit of a hack… any changes to state implemented by manual gcode will not be tracked by FullControl when converting the ***design*** to a ***result*** (for both types - gcode and plot). - e.g. if manual gcode "G1 X0" was added, the ***state*** of the printer would not be updated by this command. Therefore the next line of gcode would potentially have an incorrect length calculated and, therefore, an incorrect E-value calculated. Similarly, the lines in a 'plot' ***result*** would not include the X0 position. - e.g. if manual gcode "G91; relative positioning" was added, FullControl would still output absolute x y z values for all subsequent Points

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.ManualGcode(text="G4 P2000 ; pause for 2 seconds"))
steps.append(fc.Point(x=10))
print(fc.transform(steps, 'gcode'))
```

### 3.3 PlotAnnotation

annotations can be added to plots

if the 'point' attribute is not supplied, the annotation appears at the current position of the printer when the annotation appears in the ***design*** (list of steps)

if the 'point' attribute is supplied, this is used to dictate the position of the annotation and means that it doesn't matter where they are defined in the list of steps

[ ]:
```
steps = []
steps.append(fc.Point(x=20, y=10, z=0.2))
steps.append(fc.PlotAnnotation(label="start point"))
steps.append(fc.Point(x=30, y=20))
steps.append(fc.Point(x=20, y=30))
steps.append(fc.Point(x=10, y=20))
steps.append(fc.PlotAnnotation(label="end point"))
fc.transform(steps, 'plot')
```

[ ]:
```
steps = []
steps.append(fc.Point(x=20, y=10, z=0.2))
steps.append(fc.Point(x=30, y=20))
steps.append(fc.Point(x=20, y=30))
steps.append(fc.Point(x=10, y=20))
steps.append(fc.PlotAnnotation(label="centre point", point=fc.Point(x=20, y=20,␣
↪z=0)))
steps.append(fc.PlotAnnotation(label="3 mm above the bed", point=fc.Point(x=20,␣
↪y=20, z=3)))
fc.transform(steps, 'plot')
```