Annex A: rhino_specific

This section explains the functions available in the 'rhino_specific' folder.

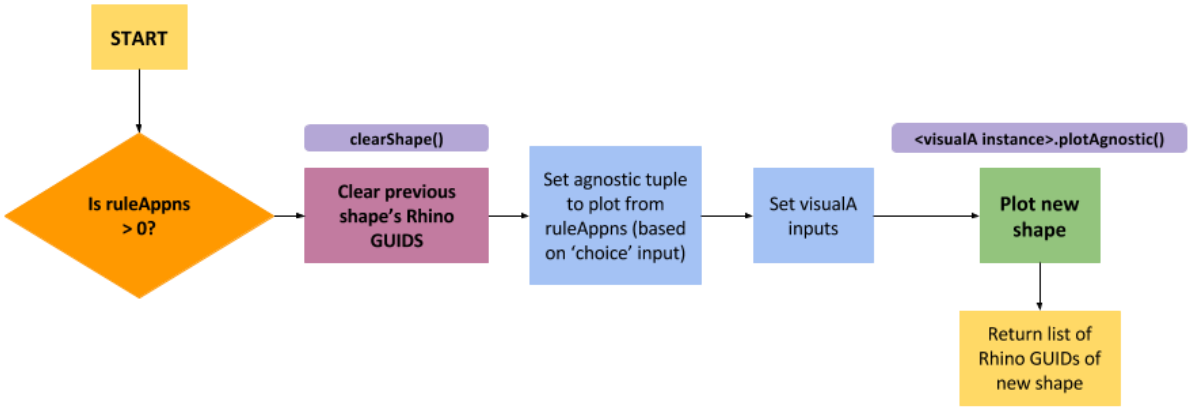
- 'draw_result' clears a shape and plots an agnostic tuple in the Rhino workspace.
- 'visualA' is used to plot individual agnostic tuples in the Rhino workspace (it is used in 'draw_result').
- 'rhino_specific_attributes' houses functions coded as static methods under the class 'rhino_methods' for adding attributes to shapes, clearing frames and entire shapes (as used in 'add_attribute.py').

The use of the class of functions 'rhino_methods' in this section is referred to by the import name 'rm', as it is imported as follows in the demo scripts:

```
from rhino_specific.rhino_specific_attributes import rhino_methods as rm
```

draw_result

This function takes as input the list of Rhino GUIDs corresponding to the main shape, the index number of the particular rule application inside 'ruleAppns' that is to be plotted, the name of the layer where the shape is to be plotted, and the reference point/insertion point of the frame block instance that is to house the new shape. The function returns a list of the Rhino GUIDs corresponding to the new shape.



'clearShape' is used to delete the Rhino GUIDs of the previous shape from the Rhino workspace. It takes as input a list of Rhino GUIDs.

```
rm.clearShape(shapeIds)
```

The new shape is assigned to the 'shapeNew' variable from 'ruleAppns' using the 'choice' input (an integer).

```
shapeNew = ruleAppns[choice][2]
```

The plotting function, ‘visualA’, is then initialized using the agnostic tuple in shapeNew, layerNameMain and refPointName.

```
vis1 = visualA(shapeNew, layerNameMain, refPointMain)
```

The ‘visualA’ instance is then used to plot the new shape, using the method ‘plotAgnostic()’. It returns the list of Rhino GUIDs corresponding to the shape just plotted.

```
shapeNewAgnosticIds = vis1.plotAgnostic()
```

This class is explained further in the next section.

visualA

The file ‘visualAgnostic’ houses the class of functions called ‘visualA’. It is a function class that plots an agnostic tuple in Rhino. An instance is initialized with the following inputs: the agnostic tuple, the layer the shape is to be plotted on, and the reference point/insertion point of the frame where the shape will be housed. The reference point can be either the block insertion point for a frame or any other 3D-coordinate location.

To plot a shape requires the following lines of code:

1. Setting up the variables inside the class instance.
   <name of visualA instance> = visualA(<shape name, agnostic form>, <layer name>, <reference point>)
2. Plotting the shape to Rhino.
   <name of visiualA instance>.plotAgnostic()

The following code snippet can be found in ‘draw_result.py’, where it plots the new shape after rule application.

```
# Initializing variables
vis1 = va(shapeNew, layerNameMain, refPointMain)

# Plotting shape
shapeNewAgnosticIds = vis1.plotAgnostic()
```

rhino_specific_attributes

The following functions are used for adding attributes to Rhino objects and changing objects and frame contents within the Rhino workspace. There are also methods for cleaning up frames, erasing shapes and removing attributes.

1. addLabel

This method adds labels to Rhino objects (line segments and points), which are visualized as text dots when the form is updated. To call the method:

```
rm.addLabel()
```

The user will be prompted to select the object they wish to put a label on. After selecting the object, the user is prompted to enter the label text for the object. If a polyline is selected, it adds the label to all lines in the polyline. For line segments and polylines, the GUID of the text dot attached to the line segment is stored in the User Text of the line object for later handling, i.e. deletion.

2. addDesc

This method adds descriptions to Rhino objects (line segments and points). It prompts for whether the description is to be read as a string of text or as a tuple. To call the method:

```
rm.addDesc()
```

The user will be prompted to select the object they wish to put a description on. After selecting the object, the function prompts for the treatment of the description (text string or tuple) and for the description itself.

3. addEnum

This method adds enumerative values to Rhino objects (line segments and points). To call the method:

```
rm.addEnum()
```

The user will be prompted to select the object they wish to put an enumerative value on. After selecting the object, the function prompts for the enumerative value. Note that the value of the input will be processed based on the matrix provided in the definition of 'enumSort' in the file 'sortTypes' (under sortalgi -> sortal_lib_api -> setup)

4. addWeight

This method adds width to lines, or grayscale color to points. It prompts the user on whether to accept a single object ('object') or several objects ('sObjects'). For lines, widths are from 0.0 to 2.0mm. For points, grayscale is from 0 (black) to 255 (white). The method can take either several objects (enter 'sObjects' when prompted) or a single object (enter 'object' when prompted) as input. To call the method:

```
rm.addWeight()
```

Afterwards, the user can select which objects to add the weight attribute to. To view lines with line widths, set the view in Rhino to Print Preview. However, note that color is not visible when Rhino is in Print Preview.

Of special note is that in Rhino, a special variant of 'weight' is used, called 'rWeight'. It is initialized in the 'sortTypes.py' file (under sortalgi -> sortal_lib_api -> setup). Instead of 'weight', 'rWeight' is the basis for creating 'weightSort' and 'colorSort'.

```
weightSort = primitiveSort('weight', rWeight, '2.0')
colorSort = primitiveSort('color', rWeight, '255')
```

The number string given as the final input above is the maximum value for that instance of primitiveSort based on rWeight. For example, if an individual based on weightSort is fed a number above this maximum value, then it rounds the stored value down to the maximum value.

5. attachLabel

This method attaches a pre-existing text dot to a line segment. It prompts for the line segment first, then for the text dot that will be assigned as a label to the line. To call the method:

```
rm.attachLabel()
```

The text of the text dot is also assigned to the User Text of the line as well, which means that in future handling, the line segment is considered as a line segment with a label attribute.

6. delLabel

This method removes the label from a line. It resets the User Text dictionary of the line segment for the key 'Label' to empty, so that in future handling, the line does not have a label attribute. This method also deletes the text dot 'attached' to the line, or the text dot that corresponds to the GUID stored in the User Text dictionary of the line. To call the method:

```
rm.delLabel()
```

7. addColor

This method adds a color to an object or several objects. It prompts first for the objects, then the RGB value of the color (i.e. entered as three numbers separated by commas for each value: '255,0,120' for example). To call the method:

```
rm.addColor()
```

Note that object color is not visible when viewing the workspace in Print Preview (which is used to view line segment widths).

8. delete

This method deletes all Rhino objects from the window, cleans up the sort and rule registers, and draws empty grammar frames. To call the method:

```
rm.delete()
```

9. clearFrame

This method prompts for the selection of a frame, and clears the frame of all Rhino objects inside it. To call the method:

```
rm.clearFrame()
```

10. clearShape

This method takes as input a list of GUIDs (or a list of lists of GUIDs) and deletes them from the Rhino workspace. To call the method.

```
rm.clearShape(<list of Rhino GUIDs>)
```