1 Introduction

The CreateGraphics package allows the developer to:

- Generate basic graphics
- Add event listeners
- Sound functionality for built-in mp3 files

Functionality is performed using the createjs javascript library available at:
`http://www.createjs.com/`.

2 Graphics

In order to work with the CreateGraphics package, you need to include the following line at the top of your Grace file:

```
import graphix as g
```

The graphics object needs to be created with the following command:

```
def graphics = g.create(width,height)
```

where width and height correspond to the desired graphics window width and height. For example:

```
def width = 300
def height = 300
def graphics = g.create(width, height)
```

3 Shapes

The following shape objects are available in CreateGraphics:Circle, Rectangle, Rounded Rectangle, PolyStar, Ellipse, Arc, Text, Line, Custom Shape, and Input Box. To draw one of these objects to the screen, pass a message to the graphics object:

- addCircle
- addRect
- addPolyStar
- addRoundRect
- addEllipse
- addArc
- addText
- addLine
- addCustomShape
- addInputBox

For instance, to add a circle to the window, you might do the following:

```
import graphix as g
def graphics = g.createGraphics(300, 300)
def circle = graphics.addCircle
circle.draw
```

Each shape has different parameters that are used to create it. These parameters have default values, so you don't need to set each one every time you create an shape.

3.1 Common Parameters

There are a few parameters that are common to each type of shape.

- **location** (Point): The x,y coordinates where the shape will be placed in the graphics window. Coordinates are expressed in Grace "Point" notation: x@y. Keep in mind that the origin is in the upper left corner of the window, so 10@10 will be 10 down and 10 right from the corner of the window.
- **color** (String): The color of the shape. Most basic colors can be set as "red", "blue", etc. However, you can also use 6-digit hex numbers such as "#CC3300" that corresponds to an HTML 5 hex colors. See `http://www.w3schools.com/tags/ref_colorpicker.asp` for more details.
- **fill** (Boolean): Whether or not you want to fill in the shape when it is drawn on the window.
- **visible** (Boolean): Whether or not you want the shape to be visible. This is true by default. Updating this value does not require for the shape to be re-drawn with the 'draw' method.

Chaining
In order to make the resulting code more compact, CreateGraphics has a number of "chaining" methods that you can use instead of setting parameters individually. The common ones are:

- on(location)
- colored(color)
- filled(fill)

This allows you to construct the object like this:

```
import graphix as g
def graphics = g.create(300, 300)
graphics.addCircle.colored("red").filled(true).draw
```

Other Methods
The following method is also available on all shape objects:

- **moveBy(x, y)**: This moves the shape relative to its current location. Both positive and negative numbers can be used.
- **contains(point)**: This determines whether or not a given point intersects with a shape object. Returns true or false.

3.2 Circle

Create: graphics.addCircle

Parameters:

- **radius** (Number): The length of the circle radius

Chaining Methods:

- **setRadius(Number)**

3.3 Rectangle

Create: graphics.addRect

Parameters:

- **width** (Number): Width of the rectangle
- **height**(Number): Height of the rectangle

Chaining Methods:

- **setWidth(Number)**
- **setHeight(Number)**

3.4 Rounded Rectangle

Create: graphics.addRoundRect

Parameters:

- **width** (Number): Width of the rectangle
- **height**(Number): Height of the rectangle
- **radius** (Number): Radius of the rounded corners

Chaining Methods:

- **setWidth(Number)**
- **setHeight(Number)**
- **setRadius(Number)**

3.5 PolyStar

Create: graphics.addPolyStar

Parameters:

- **size** (Number): Length of each side of the star
- **sides** (Number): Number of sides
- **pointSize** (Number): Size of the points

- **angle** (Number): Angle between the points

Chaining Methods:

- **setSize(Number)**
- **setSides(Number)**
- **setPointSize(Number)**
- **setAngle(Number)**

3.6 Ellipse

Create: graphics.addEllipse

Parameters:

- **width** (Number): Width of the ellipse
- **height**(Number): Height of the ellipse

Chaining Methods:

- **setWidth(Number)**
- **setHeight(Number)**

3.7 Arc

Create: graphics.addArc

Parameters:

- **radius** (Number): Radius of the arc
- **startAngle** (Number): Starting angle of the arc
- **endAngle**(Number): Ending angle of the arc

Chaining Methods:

- **setRadius(Number)**
- **setStartAngle(Number)**
- **setEndAngle(Number)**

3.8 Text

Create: graphics.addText

Parameters:

- **content** (String): The content of the string
- **font** (String): Size and font of the text (eg. ”12px Arial”)

Chaining Methods:

- **setContent(String)**
- **setFont(String)**

3.9 Line

Create: graphics.addLine

Parameters:

- **start** (Point): Location of the starting point of the line
- **end** (Point): Location of the ending point of the line

Chaining Methods: Parameters:

- **setStart(Point)**
- **setEnd(Point)**

3.10 Custom Shape

This shape consists of a set of points that you add in order to make a custom shape. Instead of configuring preset parameters, you just add points to the shape. **Create:** graphics.addCustomShape Methods:

- **addPoint** (Point): Add this point to shape the object

The addPoint method returns the object, that you can chain it together. For example:

```
import graphix as g
def graphics = g.createGraphics(300, 300)
graphics.addCustomShape.colored("red").addPoint(40@40).addPoint(0@40).addPoint(40@0).draw
```

3.11 Input Box

The input box allows you to create an input box that can be used to input strings. There are a number of parameters that can be adjusted with the input box, but, similar to the shapes objects, all parameters have default values and can be considered optional.

Parameters:

- **width** (Number): Width of the input box
- **height** (Number): Height of the input box
- **fontSize** (Number): Size of the font
- **fontFamily** (String): Name of font family (e.g. "Arial")
- **fontColor** (String): Color of the font used in input text
- **backgroundColor** (String): Color used in background of input box
- **borderColor** (String): Color used in border of input box
- **onSubmit** (Block): Code block to execute when return key is pressed

Chaining Methods:

- **setWidth** (Number)
- **setHeight** (Number)
- **setFontSize** (Number)
- **setFontFamily** (String)
- **setFontColor** (String)
- **setBackgroundColor** (String)
- **setBorderColor** (String)

Other Methods:

- **focus**: This will set the curser (focus) on the input box. This is automatically done when an input box is created, but the focus can only be set on one input box at a time, so either create the input box that should have the focus last, or set the focus manually after all input boxes have been created
- **draw**: Just like the shape objects, the "draw" method needs to be called to properly draw the input box.

4 Drawing a Shape

To draw a shape on the graphics window, first create it, then configure it, and then draw it. The following code creates the output down in Figure 1.

```
import graphix as g
def graphics = g.createGraphics(200, 200)
def circle = graphics.addCircle
circle.color := "red"
circle.radius := 20
circle.position := 30@30
circle.fill := true
circle.draw
```

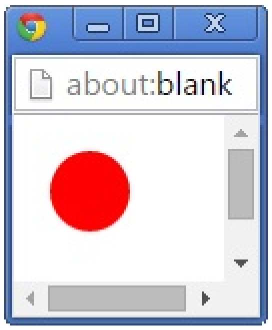

Figure 1: Creating a red circle

5 Adding a Click Handler

Adding a click handler to a shape defines a block of code that will be executed when the shape is clicked. For instance, let's say that we want the red circle to turn blue when it is clicked, and we want to add a message to the user. Then you would add something like this:

```
import graphix as g
def graphics = g.createGraphics(200, 200)
def circle = graphics.addCircle
circle.color := "red"
circle.onClick := {
  print("clicked circle")
  circle.color := "blue"
  circle.draw
}
circle.draw
```

The circle.draw line located in the click block is used to update the circle object after its color has been changed. The line circle.color := "blue" updates the variable inside the circle object, but a message to circle.draw needs to be sent in order for the graphics to actually be updated.

The following shape handlers are available:

- **onClick**
- **onMouseUp**
- **onMouseDown**
- **onPressMove**
- **onMouseOver**

There are also a few handlers for the graphics object for events on the window instead of a particular element in the window.

- **onStageMouseDown**: Triggers when the mouse is pressed on the stage
- **onStageMouseUp**: Triggers when the mouse is released on the stage
- **onStageMouseMove**: Triggers when the mouse is moved on the stage
- **onMouseExit**: Triggers when the mouse is moved from the stage

6 Timed Event

In some cases, you may want to add a delay before executing a block of code. The method **timedEvent(block, time)** can be called on the graphics object.

```
import graphix as g
def graphics = g.createGraphics(200, 200)
def circle = graphics.addCircle.filled(true)
var delayedBlock := {
  print("delayed click")
}
```

```
circle.onClick := {
 graphics.timedEvent(delayedBlock, 1000)
}
circle.draw
```

This will execute delayedBlock after 1000 milliseconds.

If you are calling the timed event repeatedly and want to clear it, use the **clearTimedEvent** method on the graphics object.

7 Adding sound

CreateGraphics supports basic sounds. All sounds are preloaded in the browser and cannot be customized at this time. To play a sound, just use the ”play” method of the graphics object. For example:

```
import graphix as g
def graphics = g.createGraphics(200, 200)
def circle = graphics.addCircle
circle.color := "red"
circle.click := {
 print("clicked circle")
 graphics.play("bicycle_bell")
 circle.draw
}
circle.draw
```

The following sounds are available: note1, note2, note3, note4, note5, note6, note7, note8, bicycle_bell, snap, whoosh, shutter.