**ZSDraw**

The **ZSDraw** subsystem is a C++ class library for creating drawings consisting of two-dimensional graphical objects. The individual objects can exchange data with each other in simulation mode and change their states during runtime.

**Inhaltsverzeichnis**

[1. Introduction 3](#_Toc159264795)

[2. Image Size and Pixel Coordinates 4](#_Toc159264796)

[2.1. Pixel Drawings 4](#_Toc159264797)

[2.1.1. QRectF 4](#_Toc159264798)

[2.1.2. QLineF 4](#_Toc159264799)

[2.2. Metric Drawings 6](#_Toc159264800)

[3. Graphical Objects (Class CGraphObj) 9](#_Toc159264801)

[3.1. Coordinate Systems 9](#_Toc159264802)

[3.1.1. Items positioned without parent group on scene 9](#_Toc159264803)

[3.1.2. Items positioned in a group 12](#_Toc159264804)

[3.2. Transformations 15](#_Toc159264805)

[3.3. Resizing Groups 15](#_Toc159264806)

[3.3.1. Mouse Events 21](#_Toc159264807)

# Introduction

# Image Size and Object Coordinates

## Pixel Drawings

5

9

0

5

9

Image.width = 10 px

Image.height = 10 px

Rect.topLeft

Rect.topRight

Invisible

Line

Invisible

Line

Rect.bottomRight

Rect.bottomLeft

Rect.width = 10 px

Rect.height = 10 px

Line.p1 = (3, 4)

Line.p2 = (7, 6)

Line.dx = 4

Line.dy = 2

Line.length = 4.47213..

Internally the QGraphics system uses QPointF, QLineF, QRectF and QSizeF to store the coordinates of the graphics items.

### Rectangles

When creating a rectangle with TopLeft = (0.0/0.0) and Width = 10.0 px and Height = 10.0 px, the “right” and “bottom” methods both return 10.0. If the rectangle is drawn onto an image with both width and height set to 10.0 px, the left and bottom border lines will not become visible as shown in the image above.

### Lines

The start and end points of a line are always on whole number multiple pixel positions. If the line is neither a horizontal or vertical line, the length and angle of the line got to be calculated by:

Length = sqrt(dx² + dy²)

Angle = arctan(dy/dx)

Very unlikely the length or the angle of an oblique line will become a whole number multiple as shown in the figure above. This implies that the length or angle of a QLineF object in a pixels drawing (image) cannot be set by the user.

## Metric Drawings

### Y Axis Scale Orientation Top Down

5

10 px

10 px

5

9

Image.Width = 10 mm

Image.Width = 10 px

ScreenResolution = 1.0 px / mm

0.0

9

5.0

10.0 mm

0

0.0

5.0

10.0 mm

Line0.width/px = 9

Line0

Line1

Line2

Line3

Line0.p1/px (0/0)

Line0.p2/px (9/0)

Line0.p1/mm (0/0)

Line0.p2/mm (10/10)

Line0.width/mm = 10.0

If the drawing is setup to use metric dimensions with a width of 10 mm and a height of 10 mm, the lines Line1 and Line2 must become visible. On the screen lines can only be drawn on whole number pixel positions but not between two pixels. To ensure that Line1 and Line2 become visible, one pixel must be truncated from the available pixel range to calculate the position in metric unit. On the screen pixel positions will be rounded to a whole number.

xScaleRange/mm = Image.Width/mm = 10.0 mm

xRange/px = Image.Width/px = ScreenResolution/(mm/px) \* Image.Width/mm = 10 px

Transformation from metric positions into pixel positions:

x/px = xMin/px + ((xRange/px – 1 px) / xScaleRange/mm) \* x/mm = 0 px + (9 px / 10.0 mm) \* x/mm

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x/mm | 0.0 | 1.0 | 2.0 | 3.0 | 4.0 | 5.0 | 6.0 | 7.0 | 8.0 | 9.0 | 10.0 |
| x/px | 0.0 | 0.9 | 1.8 | 2.7 | 3.6 | 4.5 | 5.4 | 6.3 | 7.2 | 8.1 | 9.0 |
| xScreenPos/px | 0 | 1 | 2 | 3 | 4 | 5 | 5 | 6 | 7 | 8 | 9 |

Transformation from pixel positions into metric positions:

x/mm = xMin/mm + (xScaleRange/mm / (xRange/px - 1 px)) \* x/px = 0 mm + (10.0 mm / 9 px) \* x/px

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| xScreenPos/px | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| x/mm | 0.0 | 1.11 | 2.22 | 3.33 | 4.44 | 5.56 | 6.67 | 7.78 | 8.89 | 10.0 |

When using metric units, one pixel will be automatically added to the pixel range of the scene to draw lines at pixel positions.

This complies with the geometrical definition of a line to be infinitely thin and a point, which has no size. But to make a line and a point visible, you need at least one pixel.

When drawing a horizontal line, its “width” is 0 mm but one pixel is needed to make the line visible on the screen.

When drawing a point, its width and height is 0 mm but also one pixel is needed to make the point visible.

Image.Width = 10 mm

Image.Width = 10.0 mm \* 3.5 px/mm + 1 px = 36 px

ScreenResolution = 3.5 px/mm

0

5

10

15

20

25

30

350

0

5

10

15

20

25

30

350

0.0

5.0

10.0 mm

0.0

5.0

10.0 mm

Line0

Line1

Line3

Line2

Line0.width/px = 35

Line0.p1/px (0/0)

Line0.p2/px (35/0)

Line0.p1/mm (0/0)

Line0.p2/mm (10/0)

Line0.width/mm = 10.0

YScaleAxisOrientation = TopDown

Line4

Line4.p1

Line4.p2

#### Points

Transformation from metric positions into pixel positions:

**xScaleRange/mm = Image.Width/mm = 10.0 mm**

**xRange/px = Image.Width/px = ScreenResolution/(mm/px) \* Image.Width/mm + 1 px = 36 px**

x/px = xMin/px + ((xRange/px – 1 px) / xScaleRange/mm) \* x/mm = 0 px + (35 px / 10.0 mm) \* x/mm

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x/mm | 0.0 | 1.0 | 2.0 | 3.0 | 4.0 | 5.0 | 6.0 | 7.0 | 8.0 | 9.0 | 10.0 |
| x/px | 0.0 | 3.5 | 7.0 | 10.5 | 14.0 | 17.5 | 21.0 | 24.5 | 28.0 | 31.5 | 35.0 |
| xScreenPos/px | 0 | 4 | 7 | 11 | 14 | 17 | 21 | 25 | 28 | 32 | 35 |

Transformation from pixel positions into metric positions:

x/mm = xMin/mm + (xScaleRange/mm / (xRange/px - 1 px)) \* x/px = 0 mm + (10.0 mm / 35 px) \* x/px

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| xScreenPos/px | 0 | 5 | 10 | 15 | 20 | 25 | 30 | 35 |
| x/mm | 0.0 | 1.43 | 2.86 | 4.29 | 5.71 | 7.14 | 8.57 | 10.0 |

#### Sizes (Width, Height, dx, dy)

**xScaleRange/mm = Image.Width/mm = 10.0 mm**

**xRange/px = Image.Width/px = ScreenResolution/(mm/px) \* Image.Width/mm + 1 px = 36 px**

Transformation from metric sizes into pixel sizes:

dx/px = (xRange/px / xScaleRange/mm) \* x/mm = (36 px / 10.0 mm) \* x/mm

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| dx/mm | 0.0 | 1.0 | 2.0 | 3.0 | 4.0 | 5.0 | 6.0 | 7.0 | 8.0 | 9.0 | 10.0 |
| dx/px | 0.0 | 3.6 | 7.2 | 10.8 | 14.4 | 18.0 | 21.6 | 25.2 | 28.8 | 32.4 | 36.0 |

Transformation from pixel sizes into metric sizes:

dx/mm = (xScaleRange/mm / xRange/px) \* dx/px = (10.0 mm / 36 px) \* x/px

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| dx/px | 0 | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 36 |
| dx/mm | 0.0 | 1.39 | 2.78 | 4.17 | 5.56 | 7.14 | 8.33 | 9.72 | 10.0 |

**xScaleRange/mm = Image.Width/mm = 100.0 mm**

**xRange/px = Image.Width/px = ScreenResolution/(mm/px) \* Image.Width/mm + 1 px = 351 px**

Transformation from metric sizes into pixel sizes:

dx/px = (xRange/px / xScaleRange/mm) \* x/mm = (351 px / 100.0 mm) \* x/mm

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| dx/mm | 0.0 | 10.0 | 20.0 | 30.0 | 40.0 | 50.0 | 60.0 | 70.0 | 80.0 | 90.0 | 100.0 |
| dx/px | 0.0 | 35.1 | 70.2 | 105.3 | 140.4 | 175.6 | 210.6 | 245.7 | 280.8 | 315.9 | 351.0 |

Transformation from pixel sizes into metric sizes:

dx/mm = (xScaleRange/mm / xRange/px) \* dx/px = (100.0 mm / 351 px) \* x/px

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| dx/px | 0 | 1 | 10 | 50 | 100 | 150 | 200 | 250 | 300 | 350 | 351 |
| dx/mm | 0.0 | 0.28 | 2.85 | 14.25 | 28.49 | 42.74 | 56.98 | 71.23 | 85.47 | 99.72 | 100.0 |

#### Lines (Points and Sizes)

**xScaleRange/mm = Image.Width/mm = 100.0 mm**

**xRange/px = Image.Width/px = ScreenResolution/(mm/px) \* Image.Width/mm + 1 px = 351 px**

**Transformation from metric positions and sizes into pixel points and sizes:**

x/px = xMin/px + ((xRange/px – 1 px) / xScaleRange/mm) \* x/mm = 0 px + (350 px / 100.0 mm) \* x/mm

dx/px = (xRange/px / xScaleRange/mm) \* x/mm = (351 px / 100.0 mm) \* x/mm

**Transformation from pixel points and sizes into metric positions and sizes:**

x/mm = xMin/mm + (xScaleRange/mm / (xRange/px - 1 px)) \* x/px = 0 mm + (100.0 mm / 350 px) \* x/px

dx/mm = (xScaleRange/mm / xRange/px) \* dx/px = (100.0 mm / 351 px) \* x/px

**Line 0**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | x/px | y/px | x/mm | y/mm |
| P1 | 0 | 0 | 0.0 | 0.0 |
| P2 | 350 | 0 | 100.0 | 100.0 |
| Center | 175 | 0 | 50.0 | 50.0 |
| distance | 351 | 0 | 100.0 | 0.0 |
| Length | 351 | | 100.0 | |
| Angle | 0.0 ° | | | |

**Line 4**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | x/px | y/px | x/mm | y/mm |
| P1 | 100 | 100 | 28.6 | 28.6 |
| P2 | 250 | 250 | 71.4 | 71.4 |
| Center | 175 | 175 | 50.0 | 50.0 |
| distance | 150 | 150 | 42.8 | 42.8 |
| Length | 212 | | 60.5 | |
| Angle | 315° | | | |

#### Rectangles (Points and Sizes)

### Y Axis Scale Orientation Bottom Up

DrawingSize.ImageWidth = 10 mm

DrawingSize.ImageWidth = 10.0 mm \* 3.5 px/mm = 35 px

ScreenResolution = 3.5 px/mm

0

5

10

15

20

25

30

350

0

5

10

15

20

25

30

350

0.0

5.0

10.0 mm

0.0

5.0

10.0 mm

Line2

Line1

Line3

Line0

Line0.width/px = 35

Line0.p1/px (0/35)

Line0.p2/px (35/35)

Line0.p1/mm (0/0)

Line0.p2/mm (10/0)

Line0.width/mm = 10.0

DrawingSize.YScaleAxisOrientation = BottomUp

SceneRect.Width = DrawingSize.ImageWidth + 1 = 36 px

Transformation from metric positions into pixel positions:

ScaleRange/mm = DrawingSize.ImageWidth/mm = 10.0 mm

Range/px = DrawingSize.ImageWidth/px = ScreenResolution/(mm/px) \* DrawingSize.ImageWidth/mm + 1 px = 36 px

x/px = xMin/px + ((xRange/px – 1 px) / xScaleRange/mm) \* x/mm = 0 px + (35 px / 10.0 mm) \* x/mm

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x/mm | 0.0 | 1.0 | 2.0 | 3.0 | 4.0 | 5.0 | 6.0 | 7.0 | 8.0 | 9.0 | 10.0 |
| x/px | 0.0 | 3.5 | 7.0 | 10.5 | 14.0 | 17.5 | 21.0 | 24.5 | 28.0 | 31.5 | 35.0 |
| xScreenPos/px | 0 | 4 | 7 | 11 | 14 | 17 | 21 | 25 | 28 | 32 | 35 |

y/px = yMin/px - ((yRange/px – 1 px) / yScaleRange/mm) \* y/mm = 35 px - (35 px / 10.0 mm) \* y/mm

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| y/mm | 0.0 | 1.0 | 2.0 | 3.0 | 4.0 | 5.0 | 6.0 | 7.0 | 8.0 | 9.0 | 10.0 |
| y/px | 35.0 | 31.5 | 28.0 | 24.5 | 21.0 | 17.5 | 14.0 | 10.5 | 7.0 | 3.5 | 0.0 |
| yScreenPos/px | 35 | 32 | 28 | 25 | 21 | 18 | 14 | 11 | 7 | 4 | 0 |

Transformation from pixel positions into metric positions:

x/mm = xMin/mm + (xScaleRange/mm / (xRange/px - 1 px)) \* x/px = 0 mm + (10.0 mm / 35 px) \* x/px

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| xScreenPos/px | 0 | 5 | 10 | 15 | 20 | 25 | 30 | 35 |
| x/mm | 0.0 | 1.43 | 2.86 | 4.29 | 5.71 | 7.14 | 8.57 | 10.0 |

y/px = yMin/px - ((yRange/px – 1 px) / yScaleRange/mm) \* y/mm

yMin/px - y/px = ((yRange/px – 1 px) / yScaleRange/mm) \* y/mm

y/mm = (yMin/px - y/px) \* yScaleRange/mm / (yRange/px – 1 px) = (35 px – y/px) \* (10.0 mm / 35 px)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| yScreenPos/px | 0 | 5 | 10 | 15 | 20 | 25 | 30 | 35 |
| y/mm | 10.0 | 8.57 | 7.14 | 5.71 | 4.29 | 2.86 | 1.43 | 0.0 |

# Graphical Objects (Class CGraphObj)

## Coordinate Systems

Items live in their own local coordinate system. The item’s shape points (and it’s bounding rectangle) are defined relative to the item’s coordinate system with the coordinate origin point at **(0/0)**. Their coordinates are usually centered around its center point **(0, 0)**, and this is also the center for all transformations.

An item is positioned within the scene or as a child of a group. The position of the item is given in parent coordinates. If the item does not have a group as a parent, it’s parent is the scene.

Internally the objects calculate and keep their shape points in pixel coordinates.

To edit the coordinates of an object, the coordinates must be provided to the user in the unit specified for the drawing, either in pixels or a metric unit (mm, m, etc.).

When modifying the shape of an object the coordinates are set in parent coordinates and provided in the unit of the drawing scene (which may be either pixels or a metric unit). For metric units the Y coordinates are also set depending on the orientation of the Y scale axis which may be either from top to bottom or bottom to top. Those “external” coordinate values must be converted into the local coordinate system of the objects.

Vice versa if the coordinates got to be provided to the user the internal pixel coordinates must be transformed to become relative to the top left corner or bottom left corner, if the drawing scene is setup to use a metric unit and the Y scale orientation is from bottom to top.

For the pixel unit, the coordinates must always be specified relative to the upper left corner of the parent.

If the drawing is in a metric unit (mm, m, etc.), the coordinates are either specified relative to the upper left or lower left corner depending on whether TopDown or BottomUp was selected for the Y scale alignment.

### Items positioned without parent group on scene

**Scene** and **local** coordinates of items (checkmark and small rectangle) if the items don’t belong to a group and are positioned on the scene (unit of drawing in pixels, YScale aligned TopDown). The coordinates provided to the user of the checkmark and rectangle items are the **blue** scene coordinates.

Pixel Drawing, Y Scale Top Down

**x**

**y**

**(0/0)**

**Local Coordinates**

**Checkmark**

**250**

**350**

**Checkmark and Rectangle Coordinates**

**Provided to the User**

**(-50/-50)**

**(50/50)**

**350**

**250**

**(5/5)**

**(-5/-5)**

**0**

**0**

**0**

**0**

**Local Coordinates**

**Rectangle**

**320**

**230**

**270**

**280**

Metrics Drawing, Y Scale Bottom Up

**x**

**y**

**(0/0)**

**300**

**300**

**Checkmark and Rectangle Coordinates**

**Provided to the User**

**400**

**350**

**600**

**250**

**200**

**350**

**250**

**280**

**270**

**325**

**320**

**330**

### Items positioned in a group

**Scene, group** and **local** coordinates of items (checkmark and small rectangle) if the items belong to a group. The group is positioned on the scene (unit of drawing in pixels, YScale aligned TopDown). The coordinates provided to the user of the checkmark and rectangle items are the **red** group coordinates relative to top left corner of the groups bounding rectangle. The coordinates provided to the user for the group are the **blue** scene coordinates. The local coordinates of the checkmark and rectangle remain the same and are not shown again.

Pixel Drawing, Y Scale Top Down

**x**

**y**

**(0/0)**

**Group Coordinates**

**Provided to the User**

**0**

**0**

**250**

**350**

**350**

**250**

**100**

**100**

**70**

**80**

**20**

**30**

**Checkmark and Rectangle**

**Coordinates**

**Provided to the User**

**Internally used**

**Group Coordinates**

**0**

**0**

**-25**

**-50**

**50**

**50**

**-50**

**25**

!! But internally the origin of the parents (groups) coordinate system is the center of the parents bounding rectangle. When positioning the items within the group for drawing operations on the graphics scene the coordinates are defined in internal group coordinates !!

Metrics Drawing, Y Scale Bottom Up

Assuming 1px = 1mm (X-Scale with orientation Left to Right: x/mm = x/px)

**x/mm = x/px**

**y/mm**

**(0/0)**

**300**

**300**

**Coordinates of group object  
with scene as parent  
in metric unit  
provided to the user**

**400**

**350**

**600**

**250**

**200**

**350**

**250**

**280**

**270**

**325**

**320**

**330**

**Coordinates of objects  
with group as parent  
in metric unit  
provided to the user**

**(0/0)**

**100**

**100**

**50**

**50**

**30**

**20**

**75**

**70**

**80**

**Coordinates of group object  
with scene as parent in pixels**

**300**

**200**

**250**

**0**

**350**

**400**

**100**

**y/px**

**500**

**500**

**100**

**600**

## Transformations

Transformations (scaling, rotating, shearing, moving) are used, if not the shape points are modified directly but for example the bounding rectangle is resized or the object is rotated or sheared.

Transformations are also used if a group is resized, rotated or sheared. The group will apply its geometry change to its children.

The transformation matrix is applied to the original coordinates of the item. This should avoid rounding errors. When rotating an object the rotation angle for example is applied to the original coordinates of the item. When resizing a group the group will apply its scale factor to the children.

## Resizing Groups

When resizing a group all children of the group should be resized and positioned so that they keep their original relative positions and sizes within the group.

**x**

**y**

**(0/0)**

**0**

**0**

**250**

**350**

**350**

**250**

*100*

*100*

*70*

*80*

*20*

**40**

**Checkmark and Rectangle**

**Coordinates**

**Provided to the User**

**Internally used**

**Group Coordinates**

***0***

*0*

***-25***

*-50*

***50***

*50*

*-50*

*25*

**450**

**450**

**200**

**200**

**Group Coordinates**

**Provided to the User**

**140**

**160**

**60**

**-100**

**0**

**100**

**0**

**100**

**-100**

**-50**

**150**

By moving the bottom right corner of the group from (350/350) to (450/450) the following transformations have been applied to the group which again must be applied by the group to its children:

|  |  |
| --- | --- |
| **Group** | **Apply to Children** |
| Width scaled by 2.0. | Scale width by 2.0.  Move X position by factor 2.0. |
| Height scaled by 2.0. | Scale height by 2.0.  Move Y position by factor 2.0. |

To apply transformations to a graphical object the method “setGroupTransformations” is provided where all transformations may be passed at once as QVariants. If a transformation method should not be applied, an invalid QVariant value is passed.

X‘ = m11\*x + m21\*y + m31

Y‘ = m22\*y + m12\*x + m32

x = (1/m11)\*x’ – (m21/m11)\*y - m31/m11

y = (1/m22)\*y’ – (m12/m22)\*x - m32/m22

m11

m12

m13

m21

m22

m23

m31

dx

m33

m32

dy

ConnectionPoint5

ConnectionPoint6

ConnectionPoint7

ConnectionPoint8

In

Out1

Out2

Out3

CnctLine

ConnectionLine2

ConnectionLine3

ConnectionLine4

ConnectionLine5

Frame

Switch1

ConnectionPoint6

ConnectionPoint7

ConnectionPoint8

ConnectionLine3

ConnectionLine4

ConnectionLine5

Rect1

Group1

ConnectionLine1

ConnectionPoint5

ConnectionLine2

ConnectionPoint1

ConnectionPoint2

ConnectionPoint3

ConnectionPoint4

An item may be moved and rotated within the diagram scene’s coordinate system. Changing the size of an item does not change the scale factors but the real size of the item.

Transformations from the item’s coordinate system to the scene’s coordinate system is processed in two steps – first moving the object by changing it’s scene position and afterwards rotation the object around it’s bounding rectangles center point. To rotate the item three steps are needed – translating the item by the center point, rotating the item by the rotation angle and translating the item again back by the center point.

1. Move (GraphicsItem.setPos)

(0/0)

x

y

scenePos.x

scenePos.y

(0/0)

x

y

ptRotOrigin

1. Rotation

transform.translate( -ptRotOrigin )

(0/0)

x

y

(0/0)

scenePos.x

scenePos.y

transform.rotate( rotAngle\_deg )

(0/0)

x

y

(0/0)

scenePos.x

scenePos.y

transform.translate( ptRotOrigin )

(0/0)

x

y

(0/0)

scenePos.x

scenePos.y

ptRotOrigin

### Mouse Events

#### Resizing Bounding Rectangle

The graphics item receives mouse press, mouse move and mouse release events. The mouse position is provided in scene coordinates, relative to the parent object (if any) and in item coordinates.

(0/0)

x

y

(0/0)

Item.

scenePos.x

Item

scenePos.y

mouseEv.scenePos

We use the mouse position in item coordinates to resize the bounding rectangle of the item. The shape points will be adjusted correspondingly within the item’s coordinate system. On pressing the mouse the current size, the rotation point and the shape points are temporarily stored.

As long as the item receives mouse move events the item’s bounding rectangle and shape points will be adjusted relative to the coordinates captured while pressing the mouse. The transformation values (“setPos”, “ptRotOrigin”, “rotAngle\_deg”) will not be changed (but newly applied for each move event).

##### Calculating transformation values “straight forward” (wrong results)

**Calculate new Size**

(0/0)

x

y

Item.

scenePos.x

Item.

scenePos.y

mouseEv.pos

1. Calculating Bounding Rectangle and Shape Points on Move Events

(0/0)

x

y

Item.

scenePos.x

Item.

scenePos.y

mouseEv.pos

ptRotOrigin

1. Move (GraphicsItem.setPos)

(0/0)

x

y

Item.

scenePos.x

Item.

scenePos.y

(0/0)

ptRotOrigin

1. Rotation

transform.translate( -ptRotOrigin )

(0/0)

x

y

Item.

scenePos.x

Item.

scenePos.y

(0/0)

ptRotOrigin

transform.rotate( rotAngle\_deg )

(0/0)

x

y

Item.

scenePos.x

Item.

scenePos.y

(0/0)

ptRotOrigin

transform.translate( -ptRotOrigin )

(0/0)

x

y

Item.

scenePos.x

Item.

scenePos.y

mouseEv.scenePos

(0/0)

ptRotOrigin

1. On releasing the mouse the new transformation values will be adjusted.

(0/0)

x

y

Item.

scenePos.x

Item.

scenePos.y

mouseEv.scenePos

(0/0)

ptRotOrigin

The origin point for rotating the item (ptRotOrigin) and the scene position (Item.ScenePos) got to be newly calculated and adjusted. The new rotation origin point is simply the center point of the bounding rectangle of the item (in item coordinates). To get the new scene position:

* 1. the old scene position will be rotated using the previous rotation point as captured on pressing the mouse and
  2. the rotated scene position will then be rotated using the newly calculated rotation point.