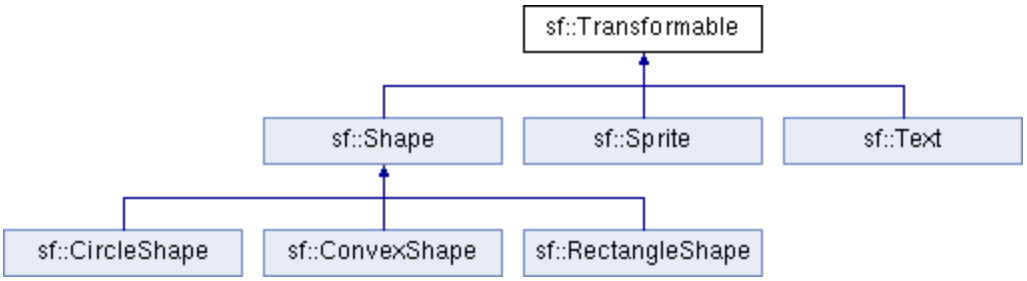
Transforming entities

All SFML classes (sprites, text, shapes) use the same interface for transformations sf::Transformable. This base class provides a simple API to move, rotate and scale your entities. It doesn't provide maximum flexibility, but instead defines an interface which is easy to understand and to use, and which covers all use cases.

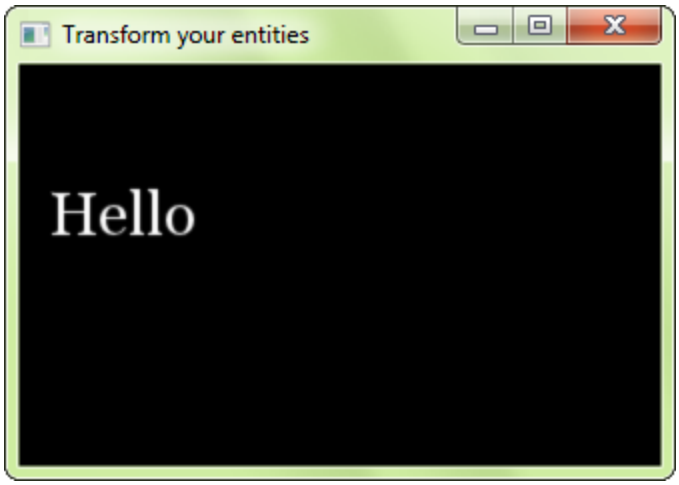
`sf::Transformable` (and all its derived classes) defines four properties: **position**, **rotation**, **scale** and **origin**. They all have their respective getters and setters. These transformation components are all independent of one another: If you want to change the orientation of the entity, you just have to set its rotation property, you don't have to care about the current position and scale.



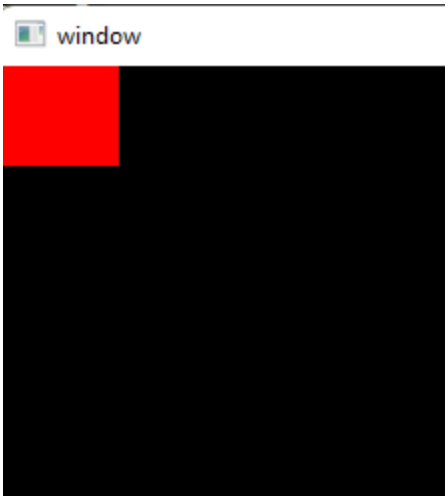
Position

The position property defines the postion of entity in 2D world.

```
// 'entity' can be a sf::Sprite, a sf::Text, a sf::Shape or any other transformable class

// set the absolute position of the entity
entity.setPosition(10.f, 50.f);

// move the entity relatively to its current position
entity.move(5.f, 5.f);

// retrieve the absolute position of the entity
sf::Vector2f position = entity.getPosition(); // = (15, 55)
```



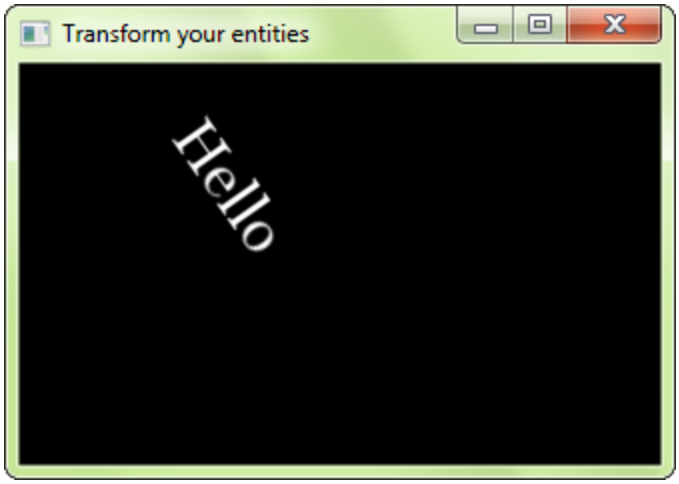
By default, entities are positioned relative to their top-left corner (0, 0).



Rotation

The rotation is the orientation of the entity in the 2D world. It is defined in **degrees**, in clockwise order (because the Y axis is pointing down in SFML).

```
// 'entity' can be a sf::Sprite, a sf::Text, a sf::Shape or any other transformable class

// set the absolute rotation of the entity
entity.setRotation(45.f);

// rotate the entity relatively to its current orientation
entity.rotate(10.f);

// retrieve the absolute rotation of the entity
float rotation = entity.getRotation(); // = 55
```



Scale

The scale factor allows the entity to be resized. The default scale is 1. Setting it to a value less than 1 makes the entity smaller, greater than 1 makes it bigger. Negative scale values are also allowed, so that you can mirror the entity.

```
// 'entity' can be a sf::Sprite, a sf::Text, a sf::Shape or any other transformable class

// set the absolute scale of the entity
entity.setScale(4.f, 1.6f);

// scale the entity relatively to its current scale
entity.scale(0.5f, 0.5f);

// retrieve the absolute scale of the entity
sf::Vector2f scale = entity.getScale(); // = (2, 0.8)
```

Origin

The origin is the center point of the three other transformations. The entity's position is the position of its origin, its rotation is performed around the origin, and the scale is applied relative to the origin as well. By default, it is the top-left corner of the window (point (0, 0)), but you can set it to the center of the window, or any other corner of the window for example.

To keep things simple, there's only a single origin for all three transformation components. This means that you can't position an entity relative to its top-left corner

while rotating it around its center for example. If you need to do such things, have a look at the next chapters.

```
// 'entity' can be a sf::Sprite, a sf::Text, a sf::Shape or any other transformable class

// set the origin of the entity
entity.setOrigin(10.f, 20.f);

// retrieve the origin of the entity
sf::Vector2f origin = entity.getOrigin(); // = (10, 20)
```

Note that changing the origin also changes where the entity is drawn on screen, even though its position property hasn't changed.

Centre the shapes

We will now see how to center the two most common shapes Circle and Rectangle

Centre the circle

```
sf::RenderWindow window(sf::VideoMode(600, 500), "window");

sf::CircleShape circle(100);
circle.setFillColor(sf::Color::White);

// center the circle
circle.setPosition
(
  (window.getSize().x / 2.f) - circle.getRadius()),
  (window.getSize().y / 2.f) - circle.getRadius())
);
```

Centre the rectangle

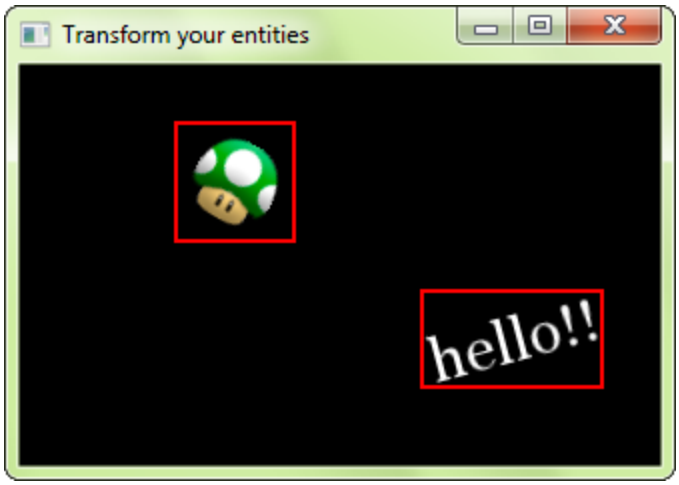
```
    sf::RenderWindow window(sf::VideoMode(600, 500), "window");

    sf::RectangleShape rectangle(sf::Vector2f(100, 50));
    rectangle.setFillColor(sf::Color::Red);

    // set the postion of a rectangle
    rectangle.setPosition(sf::Vector2f(window.getSize().x/2, window.getSize().y/2));
```

```
    // set the origin to the centre of the window
    rectangle.setOrigin(rectangle.getSize().x/2, rectangle.getSize().y/2);
```

Bounding Boxes

After transforming entities and drawing them, you might want to perform some computations using them e.g. checking for collisions.

SFML entities can give you their bounding box. The bounding box is the minimal rectangle that contains all points belonging to the entity, with sides aligned to the X and Y axes.




```
// get the bounding box of the entity
sf::FloatRect boundingBox = entity.getGlobalBounds();

// check collision with a point
sf::Vector2f point = ...;
if (boundingBox.contains(point))
{
    // collision!
}

// check collision with another box (like the bounding box of another entity)
sf::FloatRect otherBox = ...;
if (boundingBox.intersects(otherBox))
{
    // collision!
}
```

`getGlobalBounds`

The function is named `getGlobalBounds` because it returns the bounding box of the entity in the global coordinate system, i.e. after all of its transformations (position, rotation, scale) have been applied.

`getLocalBounds`

There's another function that returns the bounding box of the entity in its **local** coordinate system (before its transformations are applied) `getLocalBounds`. This function can be used to get the initial size of an entity, for example, or to perform more specific calculations.