Making software with:

# JUCE

## Introduction to COMPONENTS - Part 1

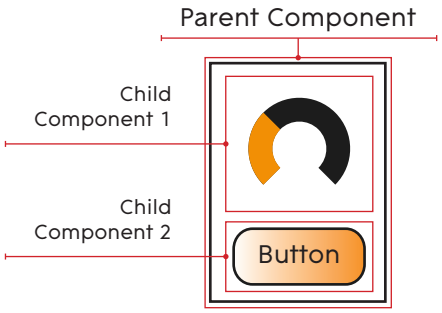
This tutorial is a relatively high level explanation of how graphical interfaces and interactive elements are constructed. We will introduce the Component class as the fundamental building block of graphical interfaces and explain the concepts of how to use it.

## What is a Component?

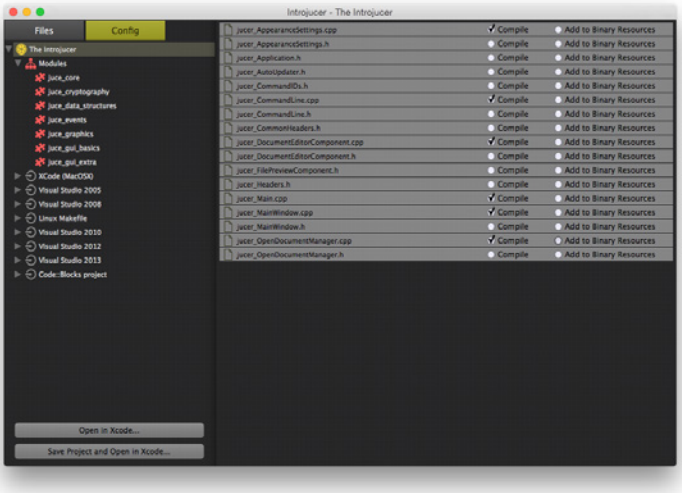
A Component is a graphical element in a JUCE application. It is used across JUCE to create everything from buttons and faders to advanced menus and even entirely new windows.

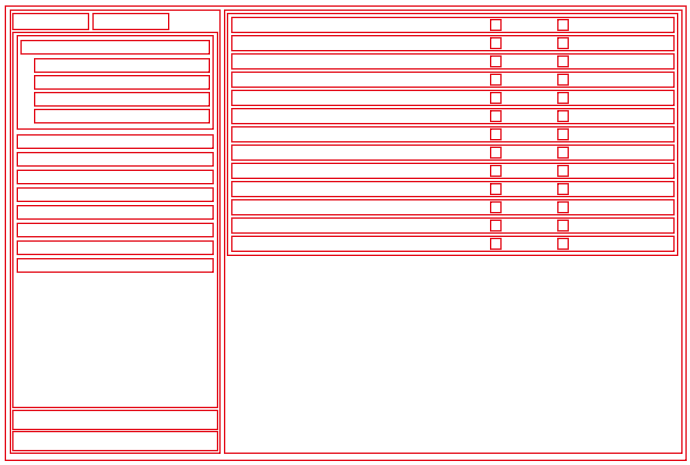
Components can be visible or invisible and often own smaller Components as members, these are called child Components.

For example you might have a panel with a slider and a button on it (shown to right). All of these elements inherit the Component class.




Components allow us to build complex, hierarchical and responsive interfaces for different devices and screen sizes, whilst keeping the code-base clean, modular and maintainable.





For example the Introjucer window can be broken down into a series of nested children Components (shown in red).

# The anatomy of the Component class:

The Component base class has a core set of methods and variables that we can use to quickly design and build interactive graphical elements. The full list of Component attributes can be found the in the Component documentation, here are just a few of the fundamental attributes. Every class you make that Inherits the Component base class will receive all of these features.


Every derived class from Component has:

| | | |
|---|---|---|
| **Name** | A String name to refer to the Component. | **`setName (String& newName);`**<br>**`String getName()`** |
| **Bounds** | A Rectangle<int> object that holds the size and position of the Component relative to its parent. | **`setBounds (Rectangle<int> newBounds);`**<br>**`Rectangle<int> getBounds();`**<br>**`Rectangle<int> getLocalBounds();`**<br>(localBounds is the bounds rectangle with top left (0, 0). |
| **Paintable canvas** | A canvas the size of the bounds that can be used to draw to the screen. | **`paint (Graphics& g) {}`**<br>Overridable function that allows you to draw custom graphics within the bounds. |
| **Mouse/touch interaction** | In built overrideable listeners to all different Mouse events and behaviours. | **`mouseDown(){}, mouseOver(){},`**<br>**`mouseDrag(){}, mouseMoved(){} etc;`**<br>Overrideable callback functions to handle mouse interactions with the Component. |
| **Enabled State** | An inbuilt bool switch that gives the Component and On/Off state. (disabling Components also disables all child Components). | **`setEnabled (bool shouldBeEnabled);`**<br>**`bool isEnabled()`** |
| **Visibility** | A bool that sets whether the Component is visible or not. | **`setVisible (bool shouldBeVisible);`**<br>**`bool isVisible()`** |
| **Parent Component** | The Component that contains this one. | **`Component* getParentComponent()`**<br>This returns a pointer to the parent Component if you need to traverse the Component hierarchy**.** |

And many more...

# Using Components: Resized and Paint

The creation and function of Components follows a sequence of events that allows us to efficiently arrange and draw them on screen. This is the general Component process.

Initialise Component

↓

Add and make visible to parent

↓

Resized

↓

Paint

Declare the Component as a member variable in the parent class. This could be the Main content Component or in another container Component class.

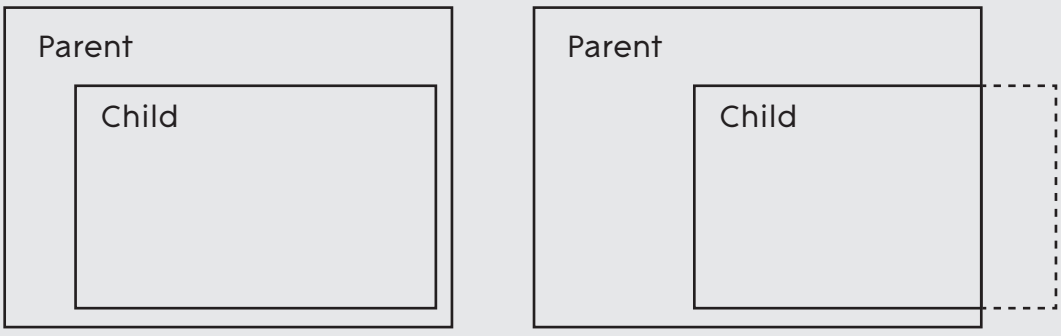
To activate and see the child Component within the parent we need to call `addAndMakeVisible (*childComponent)` within the constructor of the parent Component class.

This function is triggered whenever `setBounds (newBounds)` or `setSize (newSize)` are called on the component. The resized() function is overrideable and is where the bounds of any child Components should be set.

This overrideable function is where custom graphics can be designed and drawn to the screen. This is triggered once after the Component is added and made visible and can be called again at any time with `repaint()` to refresh or animate the contents of the Component.

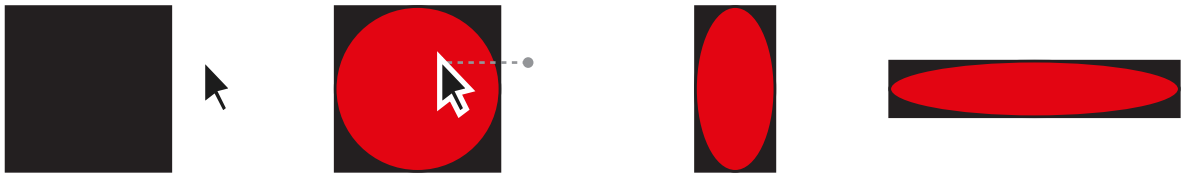
- When paint() or repaint() are called, all children Components are repainted too. This happens recursively through the hierarchy.
- Child Components' resized() function are only called if they are explicitly resized (usually with setBounds()) in the parents resized() method.
- Resized function calls happen immediately.
- Paint function calls happen asynchronously to optimise drawing to the screen at the best refresh rate.

NOTE: Child Components can be positioned to exceed the bounds of the Parent but everything outside the parent bounds will not be drawn. If you cant see your component make sure the bounds have been set properly in the parent's resized method.



Here is a prototypical class structure for a very basic new Component object. The Component draws a black rectangle with a red ellipse that toggles on and off with mouse enter events. The ellipse is drawn to the edges of the bounds rectangle.

NB: this is an inline class structure for the purposes of this tutorial.



```
class ToggleLightComponent  : public Component
{
public:
    ToggleLightComponent (String name = “light”)
            : Component (name),
              isOn (false)
    {
    }

    void paint (Graphics& g) override
    {
        g.fillAll (Colours::black);

        if (isOn)
        {
            g.setColour (Colours::red);
            g.fillEllipse (getLocalBounds().toFloat());
        }
    }

    void resized() override
    {
    }

    void mouseEnter (const MouseEvent&) override
    {
        isOn = !isOn;
        repaint();
    }

private:
    // member variables for the Component
    bool isOn;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (ToggleLightComponent)
};
```

Publicly Inherit the Component base class

Initialise the base class with the name argument (optional), and initialise the member variables.

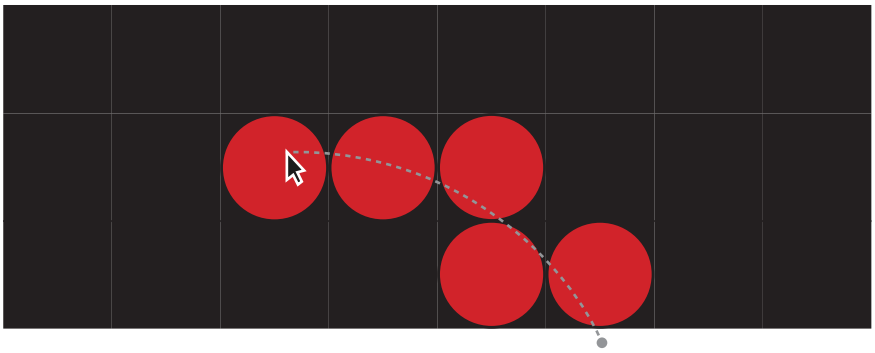
This is where we draw the graphics within our component. If the mouse is over we set the drawing colour to red and then fill an ellipse to the bounds of the Component.

This is called whenever the Component is resized. We can set the bounds of child Components relative to our new size here (in this case we have no child Components).

Mouse event callbacks handle the mouse behaviour,  we must call repaint() to draw the component again with the new state.

JUCE macro that catches memory leaks of our class

Now we are going to make a parent Component class to hold multiple ToggleLightComponents in a grid to create a spotty drawing canvas.



```
class ToggleLightGridComponent  : public Component
{
public:
    ToggleLightGridComponent (String name = "grid")
        : Component (name)
    {
        for (int i = 0; i < numX * numY; ++i)
        {
            addAndMakeVisible (toggleLights[i]);
        }
    }

    void resized() override
    {
        int stepX = getWidth() / numX;
        int stepY = getHeight() / numY;

        for (int x = 0; x < numX; ++x)
        {
            for (int y = 0; y < numY; ++y)
            {
                Rectangle<int> elementBounds (x * stepX, y * stepY, stepX, stepY);

                toggleLights [x + numX * y].setBounds (elementBounds);
            }
        }
    }

private:
    // member variables for the Component
    static const int numX = 20;
    static const int numY = 20;

    ToggleLightComponent toggleLights [numX * numY];

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (ToggleLightGridComponent)
};
```

We must add the child Components to this Component and make them visible (this is done here in a single function "addAndMakeVisible").

In this "resized" method we position our child Components in a grid relative to the width and height of this Component.

Array of our child ToggleLightComponents

Our Component should work however we need to add it to the Main Component (or a parent Component) to give it a size on the screen. We could call **`setSize(w, h)`** from the constructor of the Component however it is usually better to set the size of the Component in the **`resized()`** method of the parent Component (relative to the bounds of the parent). This structure allows you to create responsive and resizeable programs.

Try running the ComponentTutorialExample in the JUCE examples folder to see the whole structure of the program.