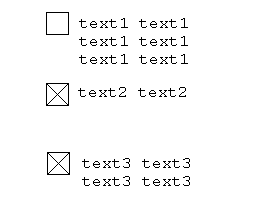
Creating a Skin MyGUI 3.0.0  
  
Part 2 - skin marked with a check (CheckBox)  
  
The last time we discussed the creation of a button. Box is a button too , but usually it beyond the basic four states , there are four that are shown when the flag stands property that it is marked (Checked):  
1. «Disabled» - button is disabled , the user does not have access to it.  
2 . «Normal» - normal state of the button .  
3 . «Highlighted» - button is highlighted , the user rolled the mouse button .  
4 . «Pushed» - button is pressed, the user clicked the mouse.  
5 . «Disabled\_checked» - button is disabled , the user does not have access to it (the button is marked .)  
6. «Normal\_checked» - normal state button ( marked ) .  
7. «Highlighted\_checked» - button is highlighted , the user rolled the mouse button (the button is marked .)  
8. «Pushed\_checked» - button is pressed, the user clicked the mouse button (the button is marked .)  
  
Imagine that when you create a check box , we need to consider that information explaining the meaning of the flag can be arranged in several rows. Here's a rough concept skin check boxes:



For the beginning we ask the artist to draw the texture with the image of the flag in all eight states. Rename (if necessary) texture «MyCheckBox.png» and copy the folder resources. That's what happened with me:



To start with the size. Our skin will consist of displays sabskina texture that will always be attached to the upper left corner and will not stretch . In the skins will attend the second sabskin displaying text. It will be located to the right of the first sabskina and stretch together with skin . Between the first and second sabskinom be short , for a better visual display of text.

So, open texture and outline of sufficient size for the skin. I turned 69 by 25 pixels . In this case, the width of the skin plays no role , but it is advisable to do it well enough visually apparent , it will facilitate the creation of skin and support in the editor :



Create a file «MyCheckBox.xml», copy it to the resources and describe it in our skin blank:

<?xml version="1.0" encoding="UTF-8"?>

<MyGUI type="Resource" version="1.1">

<Resource type="ResourceSkin" name="MyCheckBox"

size="69 25" texture="MyCheckBox.png">

</Resource>

</MyGUI>

Now add the first sabskin. Open texture in the editor and note the size and position of the first sabskina:



Sabskin first position 0 starts at the left, top 0 and has a width 27 and a height of 25 pixels. To sabskin always remained in the corner you must specify the alignment «Left Top»:

<Resource type="ResourceSkin" name="MyCheckBox"

size="69 25" texture="MyCheckBox.png">

**<BasisSkin type="SubSkin" offset="0 0 27 25" align="Left Top">**

**</BasisSkin>**

</Resource>

To make our skin was visible, create a condition named «normal» and specify the texture offset:

<Resource type="ResourceSkin" name="MyCheckBox"

size="69 25" texture="MyCheckBox.png">

<BasisSkin type="SubSkin" offset="0 0 27 25" align="Left Top">

**<State name="normal" offset="7 7 27 25"/>**

</BasisSkin>

</Resource>

In our case, the size and texture sabskina matches and is 27 by 25 pixels in the texture and the position is 7 and 7 top left pixel.  
  
Now add the code and try to create our button:

MyGUI::Gui::getInstance().load("MyCheckBox.xml");

MyGUI::Button\* button =

MyGUI::Gui::getInstance().createWidget<MyGUI::Button>(

"MyCheckBox",

MyGUI::IntCoord(30, 30, 130, 40),

MyGUI::Align::Default,

"Main");

If you did everything correctly then check should look like:



Our flag has not yet pressed and not marked and it has no text. It's time to add text sabskin. To begin, select the texture and note the position of the text in the skin:



I turned left and 30 pixels from the top 0, width 39 and height 25 pixels. I recall that the shift sabskinov considered relatively very skin and not the texture. Alignment will have sabskina «Stretch» ​​since we always want the text fills the skin. Color white point out «# FFFFFF». In order to correct the text shows in the properties of the skin shall indicate the default font «Default»:

<Resource type="ResourceSkin" name="MyCheckBox"

size="69 25" texture="MyCheckBox.png">

**<Property key="FontName" value="Default"/>**

<BasisSkin type="SubSkin" offset="0 0 27 25" align="Left Top">

<State name="normal" offset="7 7 27 25"/>

</BasisSkin>

**<BasisSkin type="SimpleText" offset="30 0 39 25" align="Stretch">**

**<State name="normal" colour="#FFFFFF"/>**

**</BasisSkin>**

</Resource>

Now add some text in the box and see what we got:

MyGUI::Gui::getInstance().load("MyCheckBox.xml");

MyGUI::Button\* button =

MyGUI::Gui::getInstance().createWidget<MyGUI::Button>(

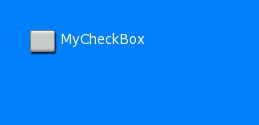
"MyCheckBox",

MyGUI::IntCoord(30, 30, 130, 40),

MyGUI::Align::Default,

"Main");

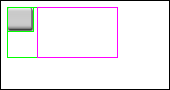
**button->setCaption("MyCheckBox");**



When stretching occurs widget and stretching the skin, but as we pointed sabskinu first position without stretching, and it does not stretch. Starting position skins:



After stretching the skin as follows:



Try to display the flag in a few lines:

MyGUI::Gui::getInstance().load("MyCheckBox.xml");

MyGUI::Button\* button =

MyGUI::Gui::getInstance().createWidget<MyGUI::Button>(

"MyCheckBox",

MyGUI::IntCoord(30, 30, 130, 40),

MyGUI::Align::Default,

"Main");

button->setCaption(**"MyCheckBox1\nMyCheckBox2"**);



Thus, the basis of our flag is ready, add it all our state and check their work. In our texture difference between the states is 30 pixels high, use this pattern and add the remaining states:

<Resource type="ResourceSkin" name="MyCheckBox"

size="69 25" texture="MyCheckBox.png">

<Property key="FontName" value="Default"/>

<BasisSkin type="SubSkin" offset="0 0 27 25" align="Left Top">

<State name="disabled" offset="7 **7** 27 25"/>

<State name="normal" offset="7 **37** 27 25"/>

<State name="highlighted" offset="7 **67** 27 25"/>

<State name="pushed" offset="7 **97** 27 25"/>

<State name="disabled\_checked" offset="7 **127** 27 25"/>

<State name="normal\_checked" offset="7 **157** 27 25"/>

<State name="highlighted\_checked" offset="7 **187** 27 25"/>

<State name="pushed\_checked" offset="7 **217** 27 25"/>

</BasisSkin>

<BasisSkin type="SimpleText" offset="30 0 39 25" align="Stretch">

<State name="disabled" colour="#FFFFFF"/>

<State name="normal" colour="#FFFFFF"/>

<State name="highlighted" colour="#FFFFFF"/>

<State name="pushed" colour="#FFFFFF"/>

<State name="disabled\_checked" colour="#FFFFFF"/>

<State name="normal\_checked" colour="#FFFFFF"/>

<State name="highlighted\_checked" colour="#FFFFFF"/>

<State name="pushed\_checked" colour="#FFFFFF"/>

</BasisSkin>

</Resource>

Now add the code in response to a button to change the state of our flag:

MyGUI::Gui::getInstance().load("MyCheckBox.xml");

MyGUI::Button\* button =

MyGUI::Gui::getInstance().createWidget<MyGUI::Button>(

"MyCheckBox",

MyGUI::IntCoord(30, 30, 130, 40),

MyGUI::Align::Default,

"Main");

button->setCaption("MyCheckBox1\nMyCheckBox2");

**button->eventMouseButtonClick = MyGUI::newDelegate(notifyMouseButtonClick);**

And the handler itself:

void notifyMouseButtonClick(MyGUI::Widget\* \_sender)

{

MyGUI::Button\* button = \_sender->castType<MyGUI::Button>();

button->setStateCheck(!button->getStateCheck());

}

Now run and check the work.



Congratulations, you have successfully created a skin for the celebrated flag!