

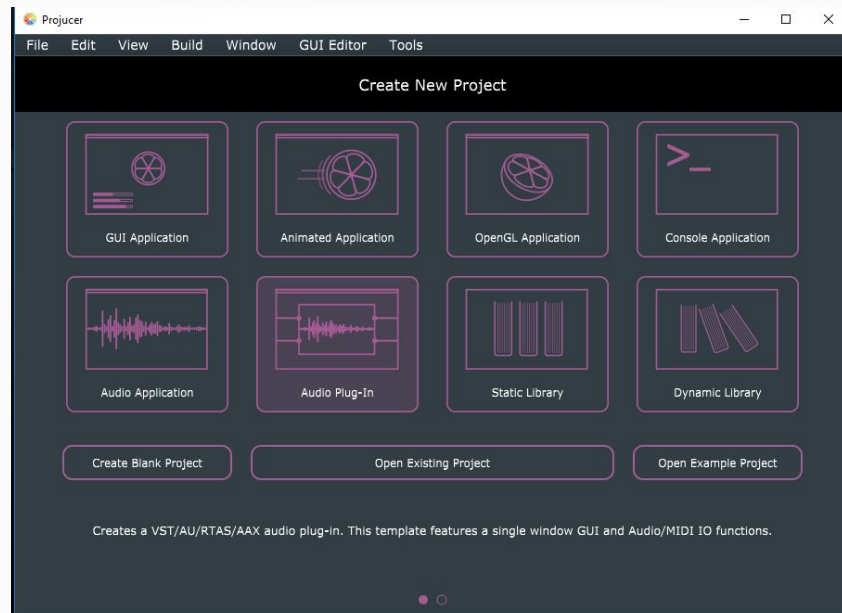
Adding a custom GUI

To your JUICE Project



Creating a new project

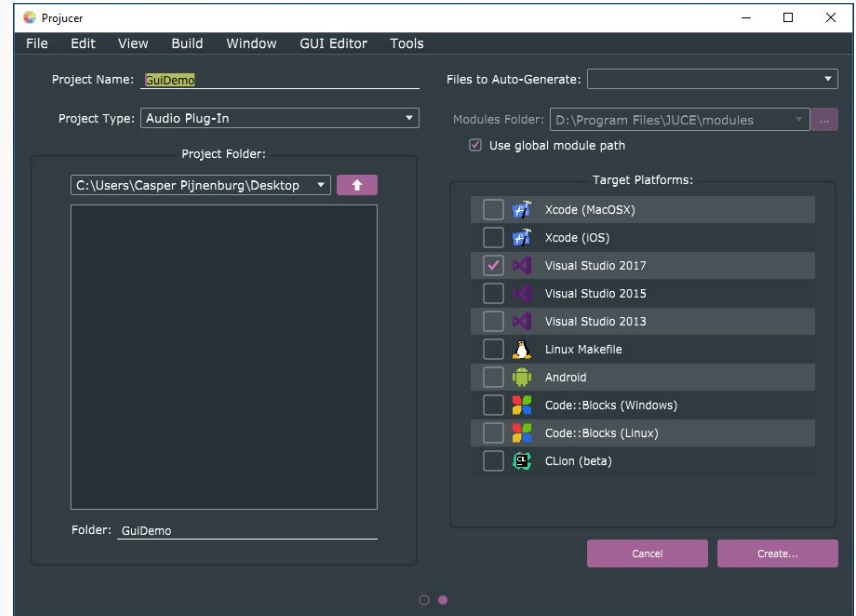
Select 'Audio Plug-in'



Creating a new project

Give it a name, in my case 'GuiDemo'

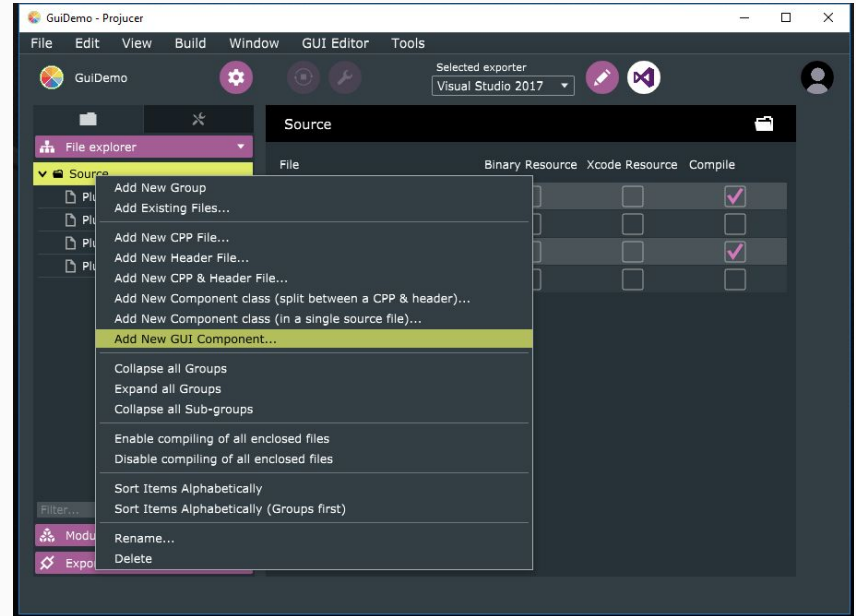
Click on Create...



Adding a new GUI component

Right-click on the Source folder

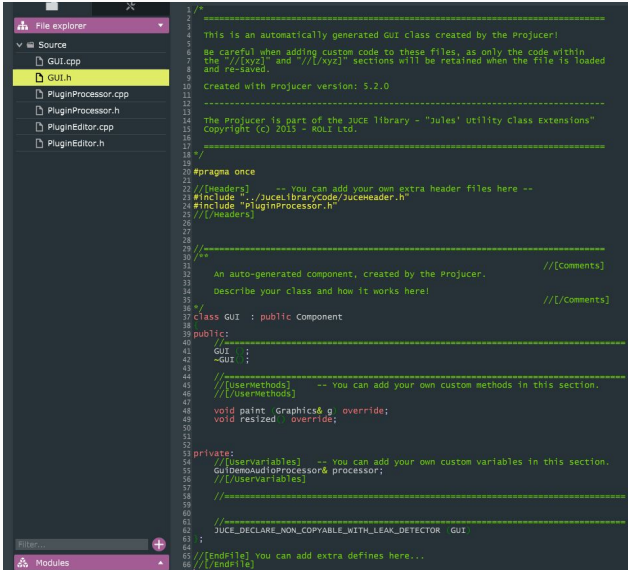
Select 'Add new GUI Component...'



Adding a new GUI component

Open GUI.h

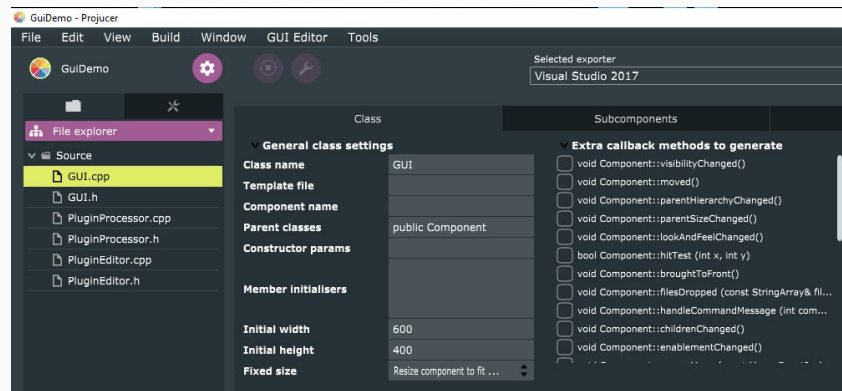
In the private section, add a reference to your GUI. 'GuiDemoAudioProcessor& processor;'



```
1  //
2  //=====
3  // This is an automatically generated GUI class created by the Projucer!
4  //
5  // Be careful when adding custom code to these files, as only the code within
6  // the //[[xyz]] and //[[/xyz]] sections will be retained when the file is loaded
7  // and re-saved.
8  //
9  // Created with Projucer version: 5.2.0
10 //=====
11 //
12 // The Projucer is part of the JUCE library - "Jules' utility Class Extensions"
13 // Copyright (c) 2013 - ROLI Ltd.
14 //=====
15 //
16 //[[headers]]
17 //-----
18 // You can add your own extra header files here --
19 //
20 #include "../juce_librarycode/jucemheader.h"
21 #include "PluginProcessor.h"
22 //[[/headers]]
23 //-----
24 //
25 //[[comments]]
26 //-----
27 // An auto-generated component, created by the Projucer.
28 // Describe your class and how it works here!
29 //[[/comments]]
30 //-----
31 //
32 // class GUI : public Component
33 //
34 public:
35     GUI() {}
36     ~GUI() {}
37
38     //[[userMethods]]
39     //-----
40     // You can add your own custom methods in this section.
41     //[[/userMethods]]
42
43     void paint(Graphics& g) override;
44     void resized() override;
45
46 private:
47     //[[userVariables]]
48     //-----
49     // You can add your own custom variables in this section.
50     GuiDemoAudioProcessor& processor;
51     //[[/userVariables]]
52     //-----
53
54     //=====
55     JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR GUI
56     //=====
57
58 //[[defines]]
59 //-----
60 // You can add extra defines here...
61 //[[/defines]]
```

Configuring the GUI component

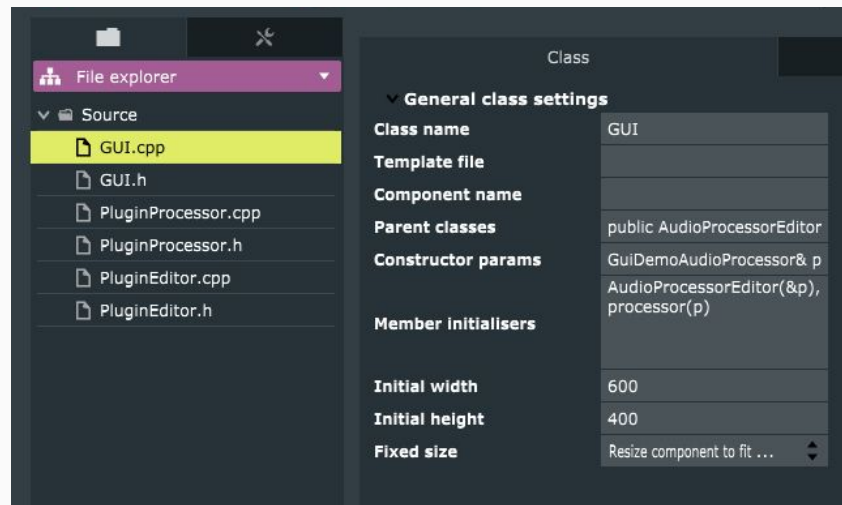
Open GUI.cpp



Configuring the GUI component

In Parent Class type: public
AudioProcessorEditor

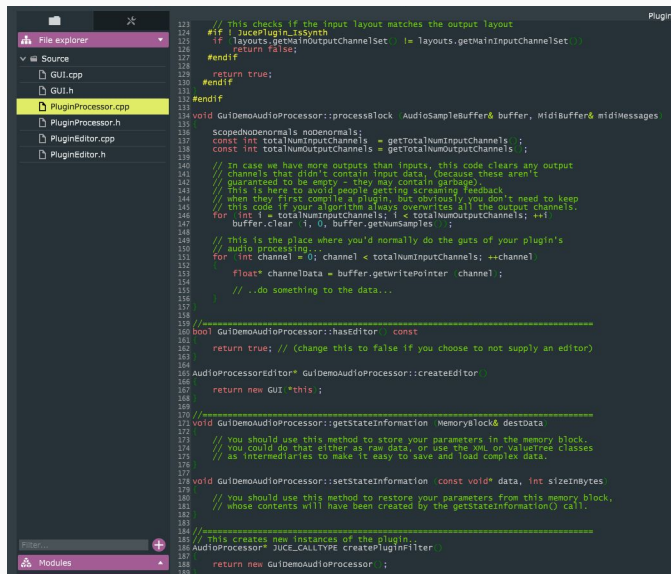
In Member Initialisers type
AudioProcessorEditor(&p), processor(p)



Configuring the Processor

In PluginProcessor.cpp, find the createEditor method.

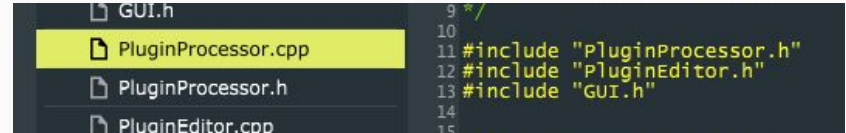
Instead of the default AudioProcessorEditor, return your own class, return your own. In my case this is return GUI(*this)



```
223 // This checks if the input layout matches the output layout
224 #if ! JUCE_PLUGIN_TSYNTH
225 if (layouts.getMainOutputChannelSet() != layouts.getMainInputChannelSet())
226     return false;
227 #endif
228 return true;
229 #endif
230
231 void GuiDemoAudioProcessor::processBlock (AudioSampleBuffer& buffer, MidiBuffer& midiMessages)
232 {
233     ScopedNoDenormals noDenormals;
234     const int totalNumInputChannels = getTotalNumInputChannels();
235     const int totalNumOutputChannels = getTotalNumOutputChannels();
236
237     // In case we have more outputs than inputs, this code clears any output
238     // channels that didn't contain input data, (because these aren't
239     // guaranteed to be empty - they may contain garbage)
240     // This is here to avoid people getting screaming feedback
241     // when they first compile a plugin, but obviously you don't need to keep
242     // this code in your plugin if it always generates all the output channels.
243     for (int i = totalNumInputChannels; i < totalNumOutputChannels; ++i)
244         buffer.clear(i, 0, buffer.getNumSamples());
245
246     // This is the place where you'd normally do the guts of your plugin's
247     // audio processing...
248     for (int channel = 0; channel < totalNumInputChannels; ++channel)
249     {
250         float* channelData = buffer.getWritePointer(channel);
251         // ..do something to the data...
252     }
253 }
254
255 bool GuiDemoAudioProcessor::hasEditor() const
256 {
257     return true; // (change this to false if you choose to not supply an editor)
258 }
259
260 AudioProcessorEditor* GuiDemoAudioProcessor::createEditor()
261 {
262     return new GUI(*this);
263 }
264
265 void GuiDemoAudioProcessor::getStateInformation (MemoryBlock& destData)
266 {
267     // You should use this method to store your parameters in the memory block.
268     // You could do that either as raw data, or use the Xml or ValueTree classes
269     // as intermediaries to make it easy to save and load complex data.
270 }
271
272 void GuiDemoAudioProcessor::setStateInformation (const void* data, int sizeInBytes)
273 {
274     // You should use this method to restore your parameters from this memory block,
275     // whose contents will have been created by the getStateInformation() call.
276 }
277
278 // This creates new instances of the plugin
279 AudioProcessor* JUCE_CALLTYPE createPluginFilter()
280 {
281     return new GuiDemoAudioProcessor();
282 }
```


Configuring the Processor

Include your header file, in my case GUI.h

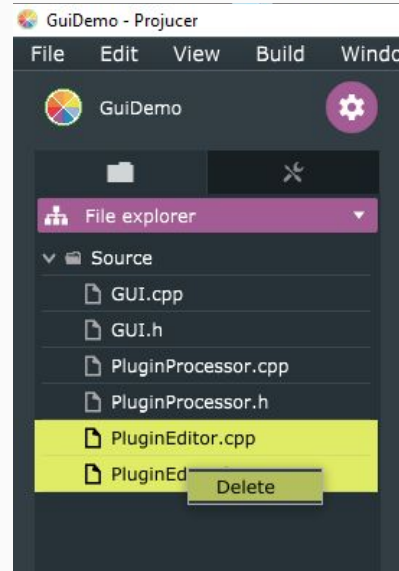


The screenshot shows a code editor interface. On the left, a file explorer sidebar lists four files: 'GUI.h', 'PluginProcessor.cpp' (highlighted in yellow), 'PluginProcessor.h', and 'PluginEditor.cpp'. On the right, the code editor displays the contents of 'PluginProcessor.cpp'. The code includes a comment on line 9, followed by three include statements on lines 11, 12, and 13: '#include "PluginProcessor.h"', '#include "PluginEditor.h"', and '#include "GUI.h"'. Line numbers 9 through 15 are visible on the left margin of the code editor.

```
9  /*  
10  
11  #include "PluginProcessor.h"  
12  #include "PluginEditor.h"  
13  #include "GUI.h"  
14  
15
```

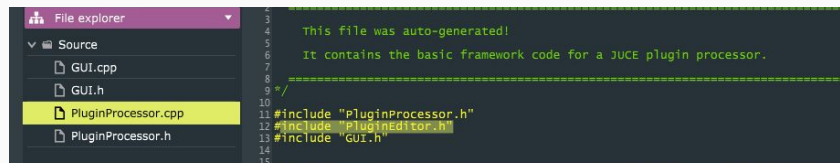
Cleaning the project

Remove the old GUI components.



Cleaning the project

Remove any reference to the old files. Otherwise your project will not compile!



The screenshot shows a code editor interface. On the left, a 'File explorer' panel is open, showing a 'Source' folder containing four files: 'GUI.cpp', 'GUI.h', 'PluginProcessor.cpp' (which is highlighted), and 'PluginProcessor.h'. On the right, the code editor displays the contents of 'PluginProcessor.cpp'. The code starts with a comment: 'This file was auto-generated! It contains the basic framework code for a JUCE plugin processor.' followed by a series of include directives: '#include "pluginProcessor.h"', '#include "pluginEditor.h"', and '#include "GUI.h"'. The line numbers 1 through 15 are visible on the left side of the code editor.

```
1  
2  
3  
4 This file was auto-generated!  
5 It contains the basic framework code for a JUCE plugin processor.  
6  
7  
8  
9  
10  
11 #include "pluginProcessor.h"  
12 #include "pluginEditor.h"  
13 #include "GUI.h"  
14  
15
```

Done!

Next up? Learning how to create user interfaces using the GUI editor.