Appendix 1

1 Appendix 1: Essential Core Code Listing

1.1 Scanner

```
class Token:
        def __init__(self,id,type,name,cid,cfid,ctid,comment):
                self.ID = id;
                self.type = type;
                self.name = name;
                self.channelID = cid;
                self.channelFromID = cfid;
                self.channelToID = ctid;
                self.comment = comment;

class Scanner:
        def __init__(self,total):
                self.__Tokens = [];
                self.__index = 0;
        def getTokens(blocks):
                while(True):
                        handle = __getNextHandle(blocks[self.__index]);
                        if(handle[0] == -1):
                                print("The block \""+handle[2]+"\" is of unknown type!");
                        if(handle[7]):
                                tempToken = Token(handle[0],handle[1],handle[2],...);
                                self.__Tokens.append(tempToken);
                        else:
                                break;
                return self.__Tokens;
```

Figure 1: The scanner is used for taking the Blocks generated by the GUI and turning them into tokens to be used by the parser.

1.2 Parser

```
class Tree:
        def __init__(self,curr):
                self.Curr = curr
                self.Next = None
                self.Prev = None
        def N(self,tmp):
                self.Next = tmp
        def P(self,tmp):
                self.Prev = tmp

class Parser:
        def __init__(self):
                self.hasMembers = False
                self.StartNode = None;
                self.current = self.StartNode;
                self.numNodes = 0;
        def __Start():
                self.StartNode = Tree();
                return self.StartNode
        def __addNode(address,argsList):
                self.numNodes += 1;
                self.hasMembers = True;
                Node = Tree();
                return Node;
        def __Term():
                return None;
        def getTree():
                return self.StatNode;
```

Figure 2: The parser is responsible for taking interpreting the Channel stack and building a tree for the generator.

1.3 Generator

```
def addBlock(self,name,II,args,comment):
        input = fileIO("python")
        newBlock = input.read_fil(name)
        for i in range (0,100):
                indent = ''
                for j in range(0,i+II):
                        indent += '\t'
                newBlock = newBlock.replace('#'+str(i)+'#',indent)
        for i in range(0, len(args)):
                word = args[i]
                word = word.replace('.','_')
                newBlock = newBlock.replace('<<ARG'+str(i)+'>>',word)
        newBlock = newBlock.replace('<<COM>>',comment)
        self.spaghetti += newBlock
```

Figure 3: The generator is responsible for interpreting the parsing tree created by the parser. It then translates the tree into a Python code file.

2 APPENDIX 1: ESSENTIAL GUI CODE LISTING

2.1 wysiwyg.kv - Block Menu

```
...
BoxLayout:
        height: .5
        orientation: 'vertical'
        MenuButton:
                text: "Variable"
                on_release: root.addBlock("variable")

        MenuButton:
                text: "Methods"
                on_release: dropdownMeth.open(self)

        MenuButton:
                text: "Class"
                on_release: root.addBlock("class")

        MenuButton:
                text: "Probe"
                on_release: root.addBlock("probe")

        MenuButton:
                text: "Output"
                on_release: root.addBlock("output")

        MenuButton:
                text: "Delete"
                on_release: root.delete_widgets()

...
```

Figure 4: The general layout of the buttons used in the Block Menu of our GUI. Each button has a label denoted by the ″text:″ attribute and a function that is called with the ″on_release:″ attribute.

2.2 wysiwyg.kv - Method Menu

```
...
ScrollView:
        DropDown:
                auto_width: False
                id: dropdownMeth
                on_parent: self.dismiss()
                size_hint_x: None
                size_hint_y: None
                width: 125
                CustButton:
                        text: 'if'
                        size: 80, 20
                        on_release: root.addBlock("method",3,"if")
                CustButton:
                        text: 'while'
                        size: 80, 20
                        on_release: root.addBlock("method",2,"while")
                CustButton:
                        text: 'for'
                        size: 80, 20
                        on_release: root.addBlock("method",3,"for")
                CustButton:
                        text: 'tf.abs'
                        size: 80, 20
                        on_release: root.addBlock("method",1,"tf.abs")
                CustButton:
                        text: 'tf.accumulate_n'
                        size: 100, 20
                        on_release: root.addBlock("method",1,"tf.accumulate_n")
                CustButton:
                        text: 'tf.acos'
                        size: 100, 20
                        on_release: root.addBlock("method",1,"tf.acos")
...
```

Figure 5: The general layout of the buttons used in the Method Menu of our GUI. This is displayed to the user in a scrollable dropdown menu when they click the ”Method” button in the Block Menu. Each button has a label denoted by the ”text:” attribute and a function that is called with the ”on_release:” attribute.

2.3 Builder Suite

```
<BuilderSuite@BoxLayout>:
    id: BuilderSuite
    size_hint_y: None
    padding: 3
    height: 40
    spacing: 1
    canvas:
        Color:
            rgba: .88, .88, .88, 1
        Rectangle:
            pos: self.pos
            size: self.size
    CustButton:
        text: "Save"
        size: 40, 40
        size_hint_x: None
    CustButton:
        text: "Play"
        size: 40, 40
        size_hint_x: None
    CustButton:
        text: "Stop"
        size: 40, 40
        size_hint_x: None
    CustButton:
        text: "Extract"
        size_hint_x: None
        size: 60, 40
        on_release: root.extract()

    CustButton:
        text: "Make Layer"
        size_hint_x: None
        size: 100, 40
        on_release: root.make_layer()

        CustButton:
        text: "Layers"
        size_hint_x: None
        size: 60, 40
        on_release: LayerDrop.open(self)

        TextInput:
                size_hint_x: 10
                id: outputpath
                hint_text: "Output Path"
                multiline: False
                on_text: root.updatePathname(self.text)
```

Figure 6: This portion of the GUI functions as a way for the user to save their current project, run the current project, stop running the current project, extract the python output file, create a layer from the current GUI, or use a previously made layer in their current project.