Current Version: 0.1.2

Last Updated: August 13, 2025

Introduction

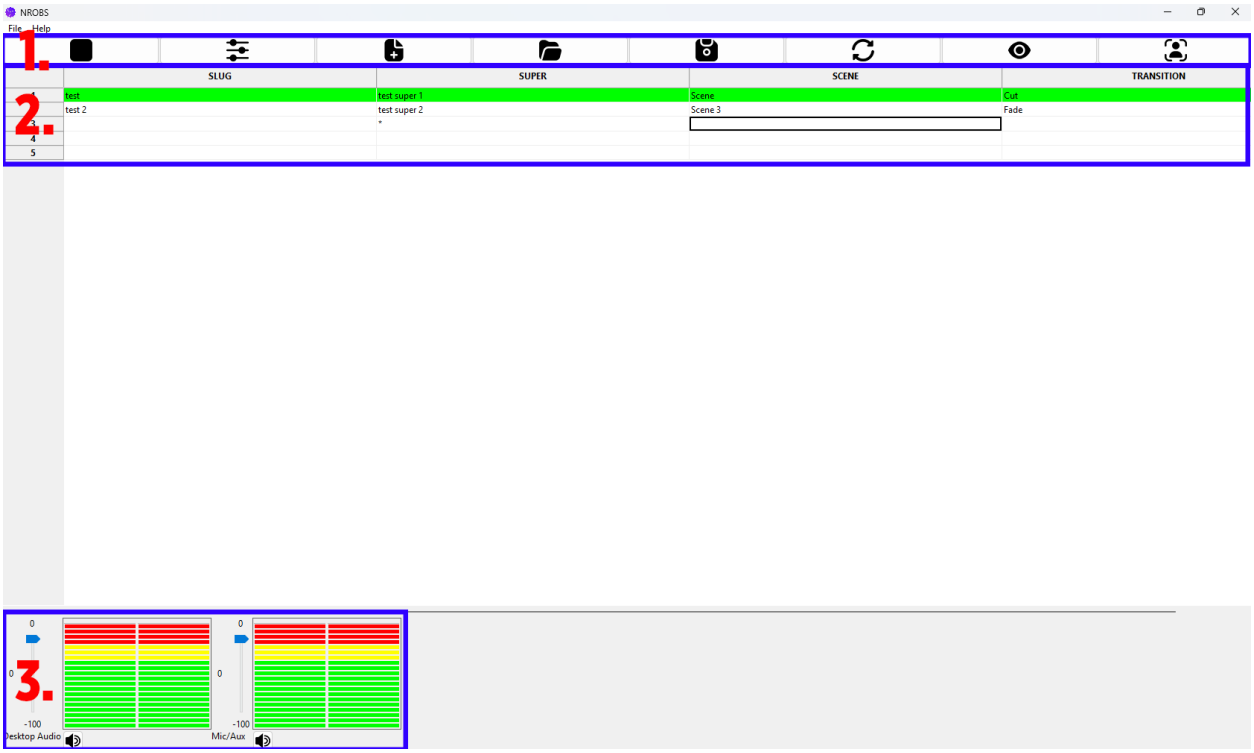
NROBS (Newsroom Open Broadcaster Software) is meant to bridge the gap between a professional broadcast automation environment and a livestream environment powered by OBS (Open Broadcaster Software). Users can create rundowns complete with slugs, super text, and scene/transition choices, then play through in order by pressing the spacebar. This document covers basic operation of NROBS as well as providing technical, backend information so that engineering staff might be able to troubleshoot, as well as the source code, which is also publicly available on Github. While this guide will cover basic functions and operations, best practices will be a matter of preference and circumstance.

Getting Started

Setup

On first launch, NROBS will prompt the user to enter the WebSocket connection information for their OBS instance. This can be accessed within OBS by clicking Tools, then WebSocket Server Settings. The WebSocket Server is built into current OBS builds by default and does not require any additional tools to be installed. Ensure that "Enable WebSocket server" is checked. You can click Show Connect Info to find all the information you'll need to connect.
IMPORTANT: NROBS REQUIRES OBS TO BE IN STUDIO MODE TO OPERATE.

Interface Overview



1. The Ribbon Menu: Various button controls for general operation. From left to right:

- Play/Stop: This button connects or disconnects NROBS from your OBS instance. When NROBS launches, it will connect by default so long as OBS is running, the WebSocket Server is enabled, and all stored connection data is correct. *The rundown will not play out unless NROBS is connected. Ensure the play symbol has changed to a stop symbol before starting a show.*
- Settings: Launches the settings window where you can adjust connection info including the IP of the machine with the OBS instance, the port (4455 is the default), the password, and the endpoint for automated super playout.
- New: Opens a new window to create a fresh rundown.
- Open: Open a saved rundown.
- Save: Save the current rundown.

- Refresh: Refreshes the available scenes and transitions in the event that they did not update automatically for any reason.
- Visibility Toggle: You can toggle elements in the current on-air scene on or off by clicking this button.
- Live Toggle: When this button is toggled on, pressing the spacebar will continue to advance the rundown even if NROBS isn't the program in focus.

2. The Rundown: This grid constitutes the rundown. A line highlighted in green is a scene that's in preview, while a red line indicates that a line is in program.

- Slug column: An optional column to slug a story if desired.
- Super column: An optional column to send super text to the automated super scene item. *If a super column is left blank, the super will retain its most recent text. You can clear the super completely by entering an asterisk in the super column.*
- Scene column: A dropdown menu populated with each scene available in the OBS instance. This is the scene that will be put into preview/program as the rundown is played out.
- Transition column: A dropdown menu populated with each available transition in the OBS instance. *If no transition is selected, a cut will be used by default.*

3. Audio Monitor: Active VU meters that display the post-fader output of available permanent sources in the OBS instance. Their levels can be adjusted on the fly via the sliders, or muted/unmuted via the button at the bottom.

Managing the Rundown

- **Populating the rundown:** The contents of cells can be edited just like any spreadsheet, with the exception of the Scene and Transition columns, which you need to double click to open the drop-down menu.
- **Adding lines:** There are two ways to add lines to a rundown. The primary way is to press ctrl + i. The secondary way is to right click on a row number, which will open a context menu that lets you add a line before or after the line you just clicked.
- **Removing lines:** To remove a line, right click on the row number and click Remove in the context menu that appears.
- **Reorder lines:** You can click and drag a row into a new spot to rearrange lines.
- **Prepare a line:** If you want to put a line into preview outside of the rundown order (for example, you wish to go back to a story from a few lines ago, or wish to skip a line), double click the row number.
- **Playout Mode:** A rundown must be put into play before it can be played out. NROBS will be put into playout mode by default when launched, so long as OBS is open, the WebSocket server is active, and the connection info is correct. If OBS is launched after NROBS, press the Play button (the first button in the ribbon menu) after OBS has launched, and ensure that the icon has changed to a Stop button before continuing. If the button does not change to a stop button, double check that the WebSocket server is active and all the connection info in Settings is correct.
- **Live Mode**: By clicking the last button on the ribbon menu, you can toggle on Live Mode. Live Mode will continue to advance the rundown when the spacebar is pressed even if NROBS isn't the focused application. This can be useful if you need to actively manage multiple applications while live but need to be able to advance the rundown regardless.

Managing a Scene

If desired, you can toggle items in the program scene on and off by clicking the eyeball icon in the ribbon menu. For example, if you wanted to lose the super on the fly, you would click on the eyeball, then click on the super scene item to hide it.

Managing Audio

The VU meters will show post-fader levels for permanent audio sources only. You can click and drag the sliders to the left of the meter to adjust the gain, or toggle the mute buttons at the bottom to mute/unmute. These meters and faders work in harmony with the faders in OBS, so you can adjust audio in whichever way is most convenient for your workflow. *Note: sources that are receiving no audio input will always display a maxed out VU meter.*

Managing Supers

Any text in the Super column will be sent to the super scene object, assuming an endpoint is provided. Supers are a browser object managed by a combination of PHP and Javascript, stylized by embedded CSS. If a Super column is left blank, the super will be left alone. If you enter an asterisk (*) in the column, it will clear the super and animate it off.

Super Overview

For automated supers, a server with PHP is required. There are three required items, plus a fonts folder containing any referenced OTF fonts:
- index.html
- send_super.php
- super.php
- super.txt
- /fonts

index.html
This is the page that will be added to the OBS instance as a browser scene object to display supers. It also contains the embedded CSS that defines the look of the super and the Javascript that fits the text to the super and animates the super in and out. You can point the browser object directly to this page or simply to the directory. *Make sure that the resolution is set to 1920x1080.*

send_super.php
This file receives the super text and puts it into super.txt. **This is the file that should be configured as your endpoint in NROBS.**

super.php
This file grabs the text from super.txt if it has changed and pushes it as a SSE to index.html. Essentially, this is what actually makes the super change at the appropriate time.

super.txt
A text file that contains the text to be displayed in the super. Managed by send_super.php and accessed by super.php.

Styling Supers

The styling of supers, meaning the overall look, color, and font, is handled entirely within the CSS embedded at the top of index.html. To customize a look of a super, edit the values appropriately.
Tip: It is possible to manage multiple styles of supers by having subfolders with different instances of index.html. Just make sure that each instance is pointed correctly to super.php and you can use the same endpoint for every style. For example, your standard super would be in the root folder, but you could have a morning show super or a sport show super style in ./morning and ./sports folders, setting those as the sources in OBS.

Backend Overview

NROBS uses the OBS WebSocket Server API via the Python package obsws_python. Detailed documentation for OBS WebSocket is available on Github[1], as well as documentation for obsws_python[2]. NROBS was written in Python 3.13.5 in the Anaconda[3] distribution of the Spyder IDE. All calls to OBS are handled via the WebSocket connection. Calls to a server for supers are handled via a POST request. For that reason, if there's a failure to connect to OBS or to update supers, those network connections should be investigated first.

The complete original project files for NROBS are available on Github[4] which would allow for the project to be edited or recompiled by anyone, though .zip files with compiled releases are available there as well. The following pages contain the current source code for the 0.1.2 build, which is current as of August 13, 2025.

Rundowns can be saved to locations of the users choosing. This generates a .json file which can be examined and edited by other applications, such as Notepad, if desired.

Connection and super endpoint data are saved as .json files in ./data/settings. In the event that connection data needs to be changed and cannot be done in NROBS for any reason, you can edit the .json files directly to accomplish this.

---

[1] https://github.com/obsproject/obs-websocket

[2] https://github.com/aatikturk/obsws-python

[3] https://www.anaconda.com/download

[4] https://github.com/tom-a-smith-citizen/OBS-Rundown

GUI

```
# -*- coding: utf-8 -*-
"""
NROBS
Created on Tue Jul  8 10:18:43 2025

@author: TOSmith
"""

import wx
import wx.grid as gridlib
import wx.lib.agw.hyperlink as hl
import wx.lib.agw.peakmeter as PM
from   wx.adv import SplashScreen as SplashScreen
import obsws_python as obs
import os
import json
import platform
import requests
from enum import IntEnum
from math import log
import webbrowser
import keyboard

class OBS(object):
    def __init__(self, parent, host, port, password):
        self.parent = parent
        self.host = host
        self.port = port
        self.password = password

    def connect(self,event):
        try:
            print(f"Connecting to {self.host}:{self.port}")
            self.cl =
obs.ReqClient(host=self.host,port=int(self.port),password=self.password,timeo
ut=3)
            self.cl_events =
obs.EventClient(host=self.host,port=int(self.port),password=self.password,tim
eout=3,subs=(obs.Subs.LOW_VOLUME | obs.Subs.INPUTVOLUMEMETERS))
            self.cl_events.callback.register(self.on_scene_list_changed)
            self.cl_events.callback.register(self.on_scene_transition_ended)
            self.parent.grid_panel.set_scene_choices()
            self.parent.grid_panel.set_transition_choices()
            has_audio_panel = hasattr(self.parent, 'mic_panel')
            if has_audio_panel:
                self.parent.mic_panel.build_faders()
            else:
                setattr(self.parent, 'mic_panel', AudioPanel(self.parent))
                self.parent.sizer.Add(self.parent.mic_panel,0,wx.EXPAND)
            self.start_event_listeners()
            self.parent.SetSizerAndFit(self.parent.sizer)
            self.parent.Layout()
        except Exception as e:
            print("Couldn't connect to OBS:",e)
```

```
    def start_event_listeners(self):
            self.cl_events.callback.register(self.on_input_volume_meters)
            self.cl_events.callback.register(self.on_input_volume_changed)

    def on_input_volume_meters(self,data):
        try:
            LEVELTYPE = IntEnum(
                "LEVELTYPE",
                "VU POSTFADER PREFADER",
                start=0,
                )
            def fget(x):
                return round(20 * log(x, 10), 1) if x > 0 else -200.0

            for device in data.inputs:
                name = device["inputName"]
                if device["inputLevelsMul"]:
                    left, right = device["inputLevelsMul"]
                    l = fget(left[LEVELTYPE.POSTFADER])
                    r = fget(right[LEVELTYPE.POSTFADER])
                    wx.CallAfter(self.parent.mic_panel.update_vu,name,l,r)
        except Exception as e:
            print("Problem handling VU meters:", e)

    def on_input_volume_changed(self,data):
        try:
            name = data.input_name
            dB = int(data.input_volume_db)
            fader = getattr(self.parent.mic_panel,f"{name}_fader")
            wx.CallAfter(fader.SetValue,dB)
        except Exception as e:
            print("Error dynamically adjusting fader:",e)

    def on_scene_list_changed(self, event):
        print("Scene list changed.")
        wx.CallAFter(self.parent.grid_panel.set_scene_choices)

    def on_scene_transition_ended(self, event):
        print("Transition finished.")
        for row in range(self.parent.grid_panel.grid.GetNumberRows()):
            color = self.parent.grid_panel.grid.GetCellBackgroundColour(row,
0)
            if color == wx.Colour(0, 255, 0):  # Green
                green_row = row
                break
        name = self.parent.grid_panel.grid.GetCellValue(green_row,2)
        transition = self.parent.grid_panel.grid.GetCellValue(green_row,3)
        if transition == "":
            print("Transition not set, using cut.")
            transition = "Cut"
        if name.strip() != "":
            self.cl.set_current_preview_scene(name)
            self.cl.set_current_scene_transition(transition)
            has_audio_panel = hasattr(self.parent, 'mic_panel')
            if has_audio_panel:
                wx.CallAfter(self.parent.mic_panel.build_faders)

    def get_scene_list(self):
```

```
        try:
            resp = self.cl.get_scene_list()
            scenes = [di.get("sceneName") for di in reversed(resp.scenes)]
            return scenes
        except Exception as e:
            print("Couldn't load scene list:", e)
            return []

    def get_transition_list(self):
        try:
            resp = self.cl.get_scene_transition_list()
            transitions = resp.transitions
            transitions = [di.get("transitionName") for di in
reversed(transitions)]
            return transitions
        except Exception as e:
            print("Couldn't load transition list:",e)
            return []

    def get_visible_items(self):
        resp = self.cl.get_current_program_scene()
        name = resp.scene_name
        items = self.cl.get_scene_item_list(name).scene_items
        output = {}
        for x in items:
            output[x['sourceName']] = {'id': x['sceneItemId'],
                                       'enabled': x['sceneItemEnabled']}
        return output

    def toggle_item(self, event, k, v, enabled):
        preview_scene = self.cl.get_current_preview_scene().scene_name

self.cl.set_current_preview_scene(self.cl.get_current_program_scene().scene_n
ame)

self.cl.set_scene_item_enabled(self.cl.get_current_program_scene().scene_name
,v,enabled)
        self.cl.trigger_studio_mode_transition()
        self.cl.set_current_preview_scene(preview_scene)
        self.parent.grid_panel.grid.SetFocus()

    def get_audio_inputs(self):
        audio_inputs = self.cl.get_input_list('wasapi_input_capture').inputs
        special_sources = self.cl.get_special_inputs()
        global_sources = special_sources.__dict__
        sources_output = {}
        for key, value in global_sources.items():
            if key != "attrs" and key[0:2] != "__":
                sources_output[key] = {'global': True,
                                       'name': value,
                                       'UUID': None}

        for x in audio_inputs:
                sources_output[x['inputUuid']] = {'global': False,
                                       'name': x['inputName'],
                                       'UUID': x['inputUuid']}

        return sources_output
```

```
    def get_audio_levels(self, sources_output: dict):
        source_and_level = {}
        for key, value in sources_output.items():
            if sources_output[key]['name'] is not None:
                level = self.cl.get_input_volume(sources_output[key]
['name']).input_volume_db
                muted = self.cl.get_input_mute(sources_output[key]
['name']).input_muted
                source_and_level[sources_output[key]['name']] = {'level':
level,
                                                                 'muted':
muted}
                print(f"{sources_output[key]['name']}: {level} dB")
        return source_and_level

    def adjust_level(self, event, name, fader):
        new_level = fader.GetValue()
        self.cl.set_input_volume(name=name,vol_db=new_level)

    def toggle_mute(self, name):
        self.cl.toggle_input_mute(name)

class GUI(wx.Frame):
    def __init__(self,title,obs_connection,super_endpoint):
        super().__init__(parent=None,title=title)
        splash = Splash()
        splash.CenterOnScreen(wx.BOTH)
        splash.Show(True)
        self.Bind(wx.EVT_CLOSE,self.on_close)
        self.SetIcon(wx.Icon("./data/icons/app.png",wx.BITMAP_TYPE_PNG))
        self.super_endpoint = super_endpoint
        self.obs_connection = obs_connection
        self.obs_conn =
OBS(self,obs_connection[0],obs_connection[1],obs_connection[2])
        self.ribbon_panel = Ribbon(self)
        self.grid_panel = Grid(self)
        self.build_menubar()
        self.sizer = wx.BoxSizer(wx.VERTICAL)
        self.sizer.Add(self.ribbon_panel,0,wx.ALL|wx.EXPAND)
        self.sizer.Add(self.grid_panel, 1, wx.ALL|wx.EXPAND|wx.CENTRE)
        self.SetSizerAndFit(self.sizer)
        self.SetInitialSize(self.GetBestSize())
        self.Bind(wx.EVT_SIZE,self.grid_panel.auto_resize_columns)
        self.Layout()
        self.Show()
        wx.CallAfter(self.ribbon_panel.on_play,wx.Event)

    def on_close(self, event):
        try:
            self.obs_conn.cl_events.disconnect()
        except Exception as e:
            print("Couldn't disconnect from OBS:",e)

        try:

self.obs_conn.cl_events.callback.deregister([self.obs_conn.on_input_volume_me
ters,self.obs_conn.on_input_volume_changed])
```

```
            self.obs_conn.cl.disconnect()
        except Exception as e:
            print("Couldn't deregister event listeners:",e)
        finally:
            keyboard.unhook_all()
            self.save_settings()
            self.Destroy()

    def save_settings(self):
        try:
            if not os.path.isdir("data/settings"):
                os.makedirs("data/settings")
            with open("data/settings/obs_settings.json", "w") as file:
                settings = {"host": self.obs_conn.host,
                            "port": self.obs_conn.port,
                            "password": self.obs_conn.password}
                json.dump(settings,file)
                print("Json saved.")
        except Exception as e:
            print("Error dumping json settings to file:", e)
        try:
            if not os.path.isdir("data/settings"):
                os.makedirs("data/settings")
            with open("data/settings/super_endpoint.json","w") as file:
                settings = {"endpoint": self.super_endpoint}
                json.dump(settings,file)
                print("Json saved.")
        except Exception as e:
            print("Couldn't save settings:",e)

    def build_menubar(self):
        menubar = wx.MenuBar()

        file = wx.Menu()
        new = file.Append(wx.ID_ANY,"New","New rundown.")
        self.Bind(wx.EVT_MENU, self.on_new, new)
        _open = file.Append(wx.ID_ANY,"Open","Open a rundown.")
        self.Bind(wx.EVT_MENU, self.ribbon_panel.on_open,_open)
        save = file.Append(wx.ID_ANY,"Save As","Save a rundown.")
        self.Bind(wx.EVT_MENU, self.ribbon_panel.on_save,save)
        settings = file.Append(wx.ID_ANY,"Settings","Change settings.")
        self.Bind(wx.EVT_MENU,self.ribbon_panel.on_settings,settings)
        _exit = file.Append(wx.ID_ANY,"Quit","Quit this program.")
        self.Bind(wx.EVT_MENU, self.on_close, _exit)
        menubar.Append(file,"File")

        _help = wx.Menu()
        about = _help.Append(wx.ID_ANY,"About","About this program.")
        self.Bind(wx.EVT_MENU,self.on_about,about)
        documentation = _help.Append(wx.ID_ANY,"Documentation","Open a PDF
with this program's documentation.")
        self.Bind(wx.EVT_MENU,self.on_documentation,documentation)
        menubar.Append(_help,"Help")

        self.SetMenuBar(menubar)

    def on_new(self, event):
```

```
        GUI("OBS Rundown",
(self.obs_connection[0],self.obs_connection[1],self.obs_connection[2]),self.s
uper_endpoint)

    def on_about(self, event):
        AboutFrame(self)

    def on_documentation(self,event):
        webbrowser.open("https://github.com/tom-a-smith-citizen/OBS-Rundown")

class Ribbon(wx.Panel):
    def __init__(self, parent):
        super().__init__(parent=parent)
        self.parent = parent
        self.sizer = wx.BoxSizer(wx.HORIZONTAL)
        self.is_playing = False
        self.live_mode = False
        self.load_bitmaps()
        self.SetSizer(self.sizer)
        self.Layout()

    def load_bitmaps(self):
        sys_appearance = wx.SystemSettings.GetAppearance()
        if sys_appearance.IsDark() and platform.system() != "Windows":
            self.directory = "./data/icons/dark"
        else:
            self.directory = "./data/icons/light"
        self.button_play = wx.BitmapButton(self,
bitmap=wx.Bitmap(os.path.join(self.directory,"play.png"),wx.BITMAP_TYPE_PNG))
        self.button_play.Bind(wx.EVT_BUTTON,self.on_play)
        self.button_play.SetToolTip("Play")
        self.sizer.Add(self.button_play,1,wx.ALL)
        self.button_settings = wx.BitmapButton(self,
bitmap=wx.Bitmap(os.path.join(self.directory,"settings-
sliders.png"),wx.BITMAP_TYPE_PNG))
        self.button_settings.Bind(wx.EVT_BUTTON,self.on_settings)
        self.button_settings.SetToolTip("Settings")
        self.sizer.Add(self.button_settings,1,wx.ALL)
        self.button_new = wx.BitmapButton(self,
bitmap=wx.Bitmap(os.path.join(self.directory,"add-
document.png"),wx.BITMAP_TYPE_PNG))
        self.button_new.Bind(wx.EVT_BUTTON,self.parent.on_new)
        self.button_new.SetToolTip("New")
        self.sizer.Add(self.button_new,1,wx.ALL)
        self.button_open = wx.BitmapButton(self,
bitmap=wx.Bitmap(os.path.join(self.directory,"open.png"),wx.BITMAP_TYPE_PNG))
        self.button_open.Bind(wx.EVT_BUTTON, self.on_open)
        self.button_open.SetToolTip("Open")
        self.sizer.Add(self.button_open,1,wx.ALL)
        self.button_save = wx.BitmapButton(self,
bitmap=wx.Bitmap(os.path.join(self.directory,"save.png"),wx.BITMAP_TYPE_PNG))
        self.button_save.Bind(wx.EVT_BUTTON,self.on_save)
        self.button_save.SetToolTip("Save")
        self.sizer.Add(self.button_save,1,wx.ALL)
        self.button_refresh = wx.BitmapButton(self,
bitmap=wx.Bitmap(os.path.join(self.directory,"refresh.png"),wx.BITMAP_TYPE_PN
G))
        self.button_refresh.Bind(wx.EVT_BUTTON, self.on_refresh)
```

```
        self.button_refresh.SetToolTip("Refresh Scenes/Transitions")
        self.sizer.Add(self.button_refresh,1,wx.ALL)
        self.button_visible = wx.BitmapButton(self,
bitmap=wx.Bitmap(os.path.join(self.directory,"eye.png"),wx.BITMAP_TYPE_PNG))
        self.button_visible.Bind(wx.EVT_BUTTON,self.on_visible)
        self.button_visible.SetToolTip("Toggle Scene Item Visibility")
        self.sizer.Add(self.button_visible,1,wx.ALL)
        if platform.system() == "Windows":
            self.button_live = wx.BitmapToggleButton(self,
wx.ID_ANY,wx.Bitmap(os.path.join(self.directory,"live.png"),wx.BITMAP_TYPE_PN
G))
            self.button_live.Bind(wx.EVT_TOGGLEBUTTON,self.on_live_toggle)
            self.button_live.SetToolTip("Live Mode")
            self.sizer.Add(self.button_live,1,wx.ALL)

    def on_live_toggle(self, event):
        state = event.GetEventObject().GetValue()
        self.live_mode = state
        self.parent.grid_panel.grid.SetFocus()

    def on_play(self,event):
        self.is_playing = not self.is_playing
        if self.is_playing:
            try:
                print("Now Playing...")
                self.parent.obs_conn.connect(wx.Event)
                if hasattr(self.parent.obs_conn, "cl"):

self.button_play.SetBitmap(wx.Bitmap(os.path.join(self.directory,"stop.png"),
wx.BITMAP_TYPE_PNG))
                    self.button_play.SetToolTip("Stop")
            except ConnectionRefusedError:
                print("Couldn't connect to OBS.")

self.button_play.SetBitmap(wx.Bitmap(os.path.join(self.directory,"play.png"),
wx.BITMAP_TYPE_PNG))
                self.button_play.SetToolTip("Play")
        else:
            print("Stopped.")
            try:
                self.parent.obs_conn.cl.disconnect()
            except Exception as e:
                print("Couldn't disconnect from OBS:",e)
            try:

self.parent.obs_conn.cl_events.callback.deregister([self.parent.obs_conn.on_i
nput_volume_meters,self.parent.obs_conn.on_input_volume_changed])
            except Exception as e:
                print("Couldn't deregister event listeners:",e)

self.button_play.SetBitmap(wx.Bitmap(os.path.join(self.directory,"play.png"),
wx.BITMAP_TYPE_PNG))
            self.button_play.SetToolTip("Play")
        self.parent.grid_panel.grid.SetFocus()

    def on_settings(self,event):
        SettingsUI(self.parent)
        self.parent.grid_panel.grid.SetFocus()
```

```
    def on_open(self, event):
        with wx.FileDialog(self, "Open rundown", wildcard="JSON files
(*.json)|*.json",defaultDir="./saved_rundowns",
                           style=wx.FD_OPEN | wx.FD_FILE_MUST_EXIST) as
fileDialog:
            if fileDialog.ShowModal() == wx.ID_CANCEL:
                return
            pathname = fileDialog.GetPath()
            try:
                self.parent.grid_panel.load_rundown(pathname)
            except IOError:
                wx.LogError(f"Cannot open file '{pathname}'.")
        self.parent.grid_panel.grid.SetFocus()

    def on_save(self,event):
        with wx.FileDialog(self, "Save rundown", wildcard="JSON files
(*.json)|*.json", defaultDir="./saved_rundowns",style=wx.FD_SAVE |
wx.FD_OVERWRITE_PROMPT) as fileDialog:
            if fileDialog.ShowModal() == wx.ID_CANCEL:
                return
            pathname = fileDialog.GetPath()
            try:
                self.parent.grid_panel.save_rundown(wx.Event, pathname)
            except IOError:
                wx.LogError("Cannot save current data in file '%s'." %
pathname)
        self.parent.grid_panel.grid.SetFocus()

    def on_refresh(self,event):
        self.parent.grid_panel.set_scene_choices()
        self.parent.grid_panel.set_transition_choices()
        self.parent.grid_panel.grid.SetFocus()

    def on_visible(self, event):
        button = event.GetEventObject()
        screen_pos = button.GetScreenPosition()
        button_size = button.GetSize()
        client_pos = self.ScreenToClient(screen_pos)
        menu_x = client_pos.x
        menu_y = client_pos.y + button_size.height
        items = self.parent.obs_conn.get_visible_items()
        self.PopupMenu(VisiblityPopupMenu(self, items), menu_x, menu_y)
        self.parent.grid_panel.grid.SetFocus()


class Grid(wx.Panel):
    def __init__(self,parent):
        super().__init__(parent=parent)
        self.parent = parent
        self.Bind(wx.EVT_KEY_DOWN, self.on_key_down)
        self.Bind(gridlib.EVT_GRID_LABEL_LEFT_DCLICK,self.on_double_click)
        self.Bind(gridlib.EVT_GRID_LABEL_RIGHT_CLICK,self.on_right_click)
        self.Bind(gridlib.EVT_GRID_CELL_CHANGED,self.auto_resize_columns)
        keyboard.hook_key("space",self.on_spacebar)
        self.init_gui()

    def on_spacebar(self,event):
```

```
        focus = wx.Window.FindFocus()
        if focus is None and self.parent.ribbon_panel.live_mode == True:
            self.advance_rundown()

    def init_gui(self):
        self.grid = gridlib.Grid(self)
        self.grid.SetInitialSize((500,100))
        self.grid.EnableDragRowMove(enable=True)
        self.grid.CreateGrid(1,4)
        self.grid.SetColLabelValue(0,"SLUG")
        self.grid.SetColLabelValue(1,"SUPER")
        self.grid.SetColLabelValue(2,"SCENE")
        self.grid.SetColLabelValue(3,"TRANSITION")
        self.set_scene_choices()
        self.set_transition_choices()
        sizer = wx.FlexGridSizer(1,1,1,1)
        sizer.AddGrowableCol(0,1)
        sizer.AddGrowableRow(0,1)
        sizer.Add(self.grid,1,wx.ALL|wx.EXPAND)
        self.SetSizer(sizer)
        self.highlight_row(0, wx.Colour(0, 255, 0))  # Green
        self.grid.SetFocus()

    def save_rundown(self, event, filename):
        rundown = {}
        for row in range(self.grid.GetNumberRows()):
            rundown[str(row)] = {
                'slug': self.grid.GetCellValue(row, 0),
                'super': self.grid.GetCellValue(row, 1),
                'scene': self.grid.GetCellValue(row, 2),
                'transition': self.grid.GetCellValue(row, 3),
                }
        if not os.path.isdir("./saved_rundowns"):
            os.makedirs("./saved_rundowns")
        with open(filename, "w") as file:
            json.dump(rundown, file, indent=2)
            print(f"Saved rundown to {filename}")

    def load_rundown(self, filename):
        try:
            with open(filename, "r") as file:
                rundown = json.load(file)

            self.grid.ClearGrid()
            existing_rows = self.grid.GetNumberRows()
            needed_rows = len(rundown)

            if needed_rows > existing_rows:
                self.grid.AppendRows(needed_rows - existing_rows)
            elif needed_rows < existing_rows:
                self.grid.DeleteRows(0, existing_rows - needed_rows)

            for row_str, data in rundown.items():
                row = int(row_str)
                self.grid.SetCellValue(row, 0, data.get('slug', ''))
                self.grid.SetCellValue(row, 1, data.get('super', ''))
                self.grid.SetCellValue(row, 2, data.get('scene', ''))
                self.grid.SetCellValue(row, 3, data.get('transition', ''))
```

```
            self.set_scene_choices()
            self.set_transition_choices()
            self.grid.ForceRefresh()
            print(f"Loaded rundown from {filename}")
        except Exception as e:
            wx.LogError(f"Could not load rundown from file '{filename}':
{e}")


    def auto_resize_columns(self, event=None):
        total_width = self.grid.GetSize().width
        col_count = self.grid.GetNumberCols()

        for col in range(col_count):
            self.grid.SetColSize(col, total_width // col_count)

        self.grid.ForceRefresh()

        if event:
            event.Skip()

    def highlight_row(self, row, color):
        for col in range(self.grid.GetNumberCols()):
            self.grid.SetCellBackgroundColour(row, col, color)

    def clear_all_highlights(self):
        for row in range(self.grid.GetNumberRows()):
            for col in range(self.grid.GetNumberCols()):
                self.grid.SetCellBackgroundColour(row, col, wx.WHITE)

    def add_row(self):
        row = self.grid.GetNumberRows()
        self.grid.AppendRows(1)
        self.grid.SetCellValue(row, 0, "")  # SLUG
        self.grid.SetCellValue(row, 1, "")  # SUPER
        self.grid.SetCellValue(row, 2, "")  # SCENE
        self.grid.SetCellValue(row, 3, "")  # TRANSITION
        self.set_scene_choices()
        self.set_transition_choices()
        self.grid.ForceRefresh()

    def set_scene_choices(self):
        if hasattr(self.parent, "obs_conn"):
            new_scene_choices = self.parent.obs_conn.get_scene_list()
            current_selections = {}
            for row in range(self.grid.GetNumberRows()):
                current_value = self.grid.GetCellValue(row, 2)
                current_selections[row] = current_value
            for row in range(self.grid.GetNumberRows()):
                editor =
wx.grid.GridCellChoiceEditor(choices=new_scene_choices, allowOthers=False)
                self.grid.SetCellEditor(row, 2, editor)
                self.grid.SetCellValue(row, 2, "")
                if current_selections[row] in new_scene_choices:
                    self.grid.SetCellValue(row, 2, current_selections[row])
            self.grid.ForceRefresh()

    def set_transition_choices(self):
```

```
        if hasattr(self.parent, "obs_conn"):
            new_transition_choices =
self.parent.obs_conn.get_transition_list()
            current_selections = {}
            for row in range(self.grid.GetNumberRows()):
                current_value = self.grid.GetCellValue(row, 3)
                current_selections[row] = current_value
            for row in range(self.grid.GetNumberRows()):
                editor =
wx.grid.GridCellChoiceEditor(choices=new_transition_choices,
allowOthers=False)
                self.grid.SetCellEditor(row, 3, editor)
                self.grid.SetCellValue(row, 3, "")
                if current_selections[row] in new_transition_choices:
                    self.grid.SetCellValue(row, 3, current_selections[row])
            self.grid.ForceRefresh()

    def on_double_click(self, event):
        row = event.GetRow()
        try:
            name = self.grid.GetCellValue(row,2)
            self.parent.obs_conn.cl.set_current_preview_scene(name)
        except Exception as e:
            print(e)
        for r in range(self.grid.GetNumberRows()):
            for c in range(self.grid.GetNumberCols()):
                self.grid.SetCellBackgroundColour(r, c, wx.WHITE)
        for col in range(self.grid.GetNumberCols()):
            self.grid.SetCellBackgroundColour(row, col, wx.GREEN)

        self.grid.ForceRefresh()
        self.grid.ClearSelection()

    def on_right_click(self,event):
        row = event.GetRow()
        x,y = event.GetPosition()
        self.PopupMenu(RowPopupMenu(self, row), x, y)

    def send_super_text(self,text):
        try:
            headers = {'Content-Type': 'application/x-www-form-urlencoded'}
            data = f"text={text}"

requests.post(self.parent.super_endpoint,headers=headers,data=data)
        except Exception as e:
            print("There was a problem sending super text:",e)

    def on_key_down(self, event):
        code = event.GetKeyCode()
        if event.ControlDown() and code == ord('I'):
            self.add_row()
            return
        if code == wx.WXK_SPACE:
            try:
                self.advance_rundown()
            except AttributeError:
                print("Couldn't advance the rundown because the OBS instance
does not exist.")
```

```
            except Exception as e:
                print("Unhandled exception advancing rundown:",e)
        else:
            event.Skip()

    def advance_rundown(self):
        red_row = None
        green_row = None
        for row in range(self.grid.GetNumberRows()):
            color = self.grid.GetCellBackgroundColour(row, 0)
            if color == wx.Colour(0, 255, 0):  # Green
                green_row = row
            elif color == wx.Colour(255, 0, 0):  # Red
                red_row = row
        name = self.grid.GetCellValue(green_row,2)
        transition = self.grid.GetCellValue(green_row,3)
        if transition.strip() == "":
            transition = "Cut"
        super_text = self.grid.GetCellValue(green_row,1)
        if name != "":
            self.parent.obs_conn.cl.set_current_preview_scene(name)
            self.parent.obs_conn.cl.set_current_scene_transition(transition)
        self.clear_all_highlights()
        if green_row is not None:
            self.highlight_row(green_row, wx.Colour(255, 0, 0))
            next_row = green_row + 1
            if next_row >= self.grid.GetNumberRows():
                next_row = 0
        if next_row < self.grid.GetNumberRows():
            self.highlight_row(next_row, wx.Colour(0, 255, 0))
        elif red_row is None:
            self.highlight_row(0, wx.Colour(0, 255, 0))
        self.grid.ForceRefresh()
        self.parent.obs_conn.cl.trigger_studio_mode_transition()
        if super_text.strip() != "":
            wx.CallAfter(self.send_super_text(super_text))

class AudioPanel(wx.Panel):
    def __init__(self, parent):
        super().__init__(parent=parent)
        self.parent = parent
        self.sizer = wx.FlexGridSizer(1,0,0,0)
        self.build_faders()
        self.SetSizerAndFit(self.sizer)
        self.Layout()

    def build_faders(self):
        sys_appearance = wx.SystemSettings.GetAppearance()
        if sys_appearance.IsDark() and platform.system() != "Windows":
            self.directory = "./data/icons/dark"
        else:
            self.directory = "./data/icons/light"
        self.sizer.Clear()
        self.sizer.Layout()
        inputs_list = self.parent.obs_conn.get_audio_inputs()
        source_and_level = self.parent.obs_conn.get_audio_levels(inputs_list)
        for key, value in source_and_level.items():
            sizer = wx.FlexGridSizer(0,2,1,1)
```

```
            fader = setattr(self, f'{key}_fader', wx.Slider(self,
value=int(value['level']), maxValue=0, minValue=-100,style=wx.SL_VERTICAL|
wx.SL_MIN_MAX_LABELS|wx.SL_INVERSE|wx.SL_VALUE_LABEL))
            fader = getattr(self,f'{key}_fader')
            peak_meter = setattr(self, f'{key}_vu', PM.PeakMeterCtrl(self, 0,
style=wx.SIMPLE_BORDER, agwStyle=PM.PM_VERTICAL))
            peak_meter = getattr(self, f'{key}_vu')
            peak_meter.SetMeterBands(2, 20)
            peak_meter.SetRangeValue(66.67,83.3,100)
            label = wx.StaticText(self,label=key)
            sizer.AddMany([(fader,1,wx.ALL|wx.EXPAND|wx.CENTRE),
                           (peak_meter,1,wx.ALL|wx.EXPAND|wx.CENTRE),
                           (label,1,wx.ALL|wx.EXPAND|wx.CENTRE)])
            fader.Bind(wx.EVT_SCROLL, lambda evt, name=key, fader=fader:
self.parent.obs_conn.adjust_level(evt, name, fader))
            if value['muted']:
                bitmap = os.path.join(self.directory,'volume-slash.png')
                name = "muted"
            else:
                bitmap = os.path.join(self.directory,'volume.png')
                name = "unmuted"
            button =
wx.BitmapButton(self,bitmap=wx.Bitmap(bitmap,wx.BITMAP_TYPE_PNG),name=name)
            button.Bind(wx.EVT_BUTTON, lambda evt, name=key:
self.toggle_mute(evt, name))
            sizer.Add(button,1,wx.CENTRE)
            self.sizer.Add(sizer,1,wx.ALL|wx.EXPAND)
        self.Layout()

    def convert_obs_db_to_peakmeter(self, obs_db_level, peakmeter_max=100):
        obs_db_level = max(-200, min(0, obs_db_level))
        if obs_db_level <= -60:
            return 0.0
        else:
            converted_level = ((obs_db_level + 60) / 60) * peakmeter_max
            return converted_level

    def update_vu(self, name, l, r):
        try:
            data =
[self.convert_obs_db_to_peakmeter(l),self.convert_obs_db_to_peakmeter(r)]
            peak_meter = getattr(self, f"{name}_vu")
            peak_meter.SetData(arrayValue=data, offset=0, size=len(data))
        except AttributeError:
            print(f"No peak meter exists for {name}")
        except Exception as e:
            print(f"Error updating VU for {name}: {e}")

    def toggle_mute(self, event, name):
        sys_appearance = wx.SystemSettings.GetAppearance()
        if sys_appearance.IsDark() and platform.system() != "Windows":
            self.directory = "./data/icons/dark"
        else:
            self.directory = "./data/icons/light"
        self.parent.obs_conn.toggle_mute(name)
        obj = event.GetEventObject()
        button_name = obj.GetName()
        if button_name == "unmuted":
```

```
            obj.SetName("muted")
            obj.SetBitmap(wx.Bitmap(os.path.join(self.directory,'volume-
slash.png'),wx.BITMAP_TYPE_PNG))
        elif button_name == "muted":
            obj.SetName("unmuted")

obj.SetBitmap(wx.Bitmap(os.path.join(self.directory,'volume.png'),wx.BITMAP_T
YPE_PNG))


class SettingsUI(wx.Frame):
    def __init__(self,parent):
        super().__init__(parent=parent, title="Settings")
        self.parent = parent
        self.SetIcon(wx.Icon('./data/icons/app.png',wx.BITMAP_TYPE_PNG))
        self.panel_main = wx.Panel(self)
        self.sizer_main = wx.FlexGridSizer(5,2,10,10)
        self.sizer_buttons = wx.BoxSizer(wx.HORIZONTAL)
        self.label_host = wx.StaticText(self.panel_main,label="Host")
        self.field_host = wx.TextCtrl(self.panel_main)
        self.field_host.SetValue(self.parent.obs_conn.host)
        self.label_port = wx.StaticText(self.panel_main,label="Port")
        self.field_port = wx.TextCtrl(self.panel_main)
        self.field_port.SetValue(str(self.parent.obs_conn.port))
        self.label_password = wx.StaticText(self.panel_main,label="Password")
        self.field_password =
wx.TextCtrl(self.panel_main,style=wx.TE_PASSWORD)
        self.field_password.SetValue(self.parent.obs_conn.password)
        self.label_endpoint = wx.StaticText(self.panel_main,label="Super
Endpoint")
        self.field_endpoint = wx.TextCtrl(self.panel_main)
        self.field_endpoint.SetValue(self.parent.super_endpoint)
        self.button_apply = wx.Button(self.panel_main,label="Apply")
        self.button_apply.Bind(wx.EVT_BUTTON,self.on_apply)
        self.button_cancel = wx.Button(self.panel_main,label="Cancel")
        self.button_cancel.Bind(wx.EVT_BUTTON,self.on_cancel)
        self.sizer_buttons.AddMany([(self.button_apply,1,wx.ALL),
                                    (self.button_cancel,1,wx.ALL)])
        self.sizer_main.AddMany([(self.label_host,1,wx.ALL|wx.EXPAND),
                                 (self.field_host,1,wx.ALL|wx.EXPAND),
                                 (self.label_port,1,wx.ALL|wx.EXPAND),
                                 (self.field_port,1,wx.ALL|wx.EXPAND),
                                 (self.label_password,1,wx.ALL|wx.EXPAND),
                                 (self.field_password,1,wx.ALL|wx.EXPAND),
                                 (self.label_endpoint,1,wx.ALL|wx.EXPAND),
                                 (self.field_endpoint,1,wx.ALL|wx.EXPAND),
                                 (self.sizer_buttons,1,wx.ALL)])
        self.panel_main.SetSizerAndFit(self.sizer_main)
        self.Layout()
        self.Show()

    def on_apply(self, event):
        host = self.field_host.GetValue()
        port = self.field_port.GetValue()
        password = self.field_password.GetValue()
        endpoint = self.field_endpoint.GetValue()
        if host == "" or port == "" or password == "" or endpoint == "":
```

```
            dlg = wx.MessageDialog(self,message="Fields cannot be left empty.
Make sure all fields are filled out and try again.",caption="Fields Cannot Be
Empty",style=wx.OK|wx.ICON_ERROR)
            dlg.ShowModal()
        else:
            self.parent.obs_conn.host = host
            self.parent.obs_conn.port = port
            self.parent.obs_conn.password = password
            self.parent.super_endpoint = endpoint
        self.parent.save_settings()
        self.Destroy()


    def on_cancel(self, event):
        self.Destroy()

class VisiblityPopupMenu(wx.Menu):
    def __init__(self, parent, items):
        super().__init__()
        self.parent = parent
        self.build_items(items)

    def build_items(self, items):
        for key, value in items.items():
            item = self.Append(wx.ID_ANY, key, kind=wx.ITEM_CHECK)
            item.Check(value['enabled'])
            self.Bind(
                wx.EVT_MENU,
                lambda evt, k=key, v=value['id'], enabled=not
value['enabled']: self.parent.parent.obs_conn.toggle_item(evt, k, v,
enabled),
                id=item.GetId()
            )

class RowPopupMenu(wx.Menu):
    def __init__(self, parent, row):
        super().__init__()
        self.parent = parent
        self.row = row
        self.init_ui()

    def init_ui(self):
        add_before = self.Append(wx.ID_ANY, "Add Row Before")
        self.Bind(wx.EVT_MENU, self.on_add_before, add_before)
        add_after = self.Append(wx.ID_ANY, "Add Row After")
        self.Bind(wx.EVT_MENU, self.on_add_after, add_after)
        remove = self.Append(wx.ID_ANY, "Remove")
        self.Bind(wx.EVT_MENU, self.on_remove, remove)

    def on_add_before(self, event):
        pos = self.row - 1
        self.parent.grid.InsertRows(pos=pos,numRows=1,updateLabels=True)
        wx.CallAfter(self.parent.grid.ForceRefresh)
        wx.CallAfter(self.parent.grid.set_scene_choices)
        wx.CallAfter(self.parent.grid.set_transition_choices)

    def on_add_after(self, event):
        pos = self.row + 1
```

```
        self.parent.grid.InsertRows(pos=pos,numRows=1,updateLabels=True)
        wx.CallAfter(self.parent.grid.ForceRefresh)
        wx.CallAfter(self.parent.parent.grid_panel.set_scene_choices)
        wx.CallAfter(self.parent.parent.grid_panel.set_transition_choices)

    def on_remove(self, event):
        self.parent.grid.DeleteRows(pos=self.row,numRows=1,updateLabels=True)
        wx.CallAfter(self.parent.grid.ForceRefresh)
        wx.CallAfter(self.parent.parent.grid_panel.set_scene_choices)
        wx.CallAfter(self.parent.parent.grid_panel.set_transition_choices)

class AboutFrame(wx.Frame):
    def __init__(self, parent):
        super().__init__(parent=parent)
        self.parent = parent
        self.SetTitle('NROBS - About')
        self.SetIcon(wx.Icon('./data/icons/app.png',wx.BITMAP_TYPE_PNG))
        self.panel_main = wx.Panel(self)
        self.sizer_main = wx.FlexGridSizer(6,1,10,10)
        self.font = wx.Font(12, wx.FONTFAMILY_MODERN, 0, 90, underline =
False, faceName ="Arial Bold")
        self.logo = wx.Image('./data/icons/app.png', wx.BITMAP_TYPE_PNG)
        self.logo.Rescale(300,300)
        self.bitmap_logo =
wx.StaticBitmap(self.panel_main,bitmap=self.logo.ConvertToBitmap())
        self.label_program_name = wx.StaticText(self.panel_main,
label="NROBS")
        self.label_program_name.SetFont(self.font)
        self.label_byline = wx.StaticText(self.panel_main, label="by Tom
Smith")
        self.label_email = hl.HyperLinkCtrl(self.panel_main,
label="tom@tomsmith.media",URL="mailto:tom@tomsmith.media")
        self.hl_icon_attribution =
hl.HyperLinkCtrl(self.panel_main,label="Uicons by Flaticon",URL="https://
www.flaticon.com/uicons")
        self.sizer_main.AddMany([(self.bitmap_logo,1,wx.ALL|wx.CENTER|
wx.ALIGN_CENTER),
                                 (self.label_program_name,1,wx.ALL|wx.CENTER|
wx.ALIGN_CENTER),
                                 (self.label_byline,1,wx.ALL|wx.CENTER|
wx.ALIGN_CENTER),
                                 (self.label_email,1,wx.ALL|wx.CENTER|
wx.ALIGN_CENTER),
                                 (self.hl_icon_attribution,1,wx.ALL|
wx.CENTER|wx.ALIGN_CENTER)])

        self.panel_main.SetSizerAndFit(self.sizer_main)
        self.SetInitialSize(self.GetBestSize())
        self.Layout()
        self.Show()

def load_obs_settings():
    if os.path.isfile("data/settings/obs_settings.json"):
        with open("data/settings/obs_settings.json","r") as file:
            settings = json.load(file)
            obs_connection =
(settings['host'],int(settings['port']),settings['password'])
            return obs_connection
```

```
    else:
        return None

def load_super_endpoint():
    if os.path.isfile("data/settings/super_endpoint.json"):
        with open("data/settings/super_endpoint.json","r") as file:
            settings = json.load(file)
            super_endpoint = settings['endpoint']
            return super_endpoint
    else:
        return "N/A"

class FirstBoot(wx.Frame):
    def __init__(self):
        super().__init__(parent=None,title="NROBS Setup")
        splash = Splash()
        splash.CenterOnScreen(wx.BOTH)
        splash.Show(True)
        self.SetIcon(wx.Icon('./data/icons/app.png',wx.BITMAP_TYPE_PNG))
        self.panel = wx.Panel(self)
        self.sizer_main = wx.BoxSizer(wx.VERTICAL)
        self.sizer_controls = wx.FlexGridSizer(0,2,10,10)
        self.init_ui()
        self.Layout()
        self.panel.SetSizerAndFit(self.sizer_main)
        self.Show()

    def init_ui(self):
        #Explainer Text
        self.text_explainer = wx.StaticText(self.panel,label="Before use you
must configure the software. Make sure you've got the WebSocket server
enabled in OBS by clicking Tools, WebSocket Server Settings, then enter the
relevant connection information here. If you are not using a super endpoint,
leave that field as 'None' or blank.")
        #Labels
        self.label_host = wx.StaticText(self.panel,label="Host")
        self.label_port = wx.StaticText(self.panel,label="Port")
        self.label_password = wx.StaticText(self.panel,label="Password")
        self.label_super_endpoint = wx.StaticText(self.panel,label="Super
Endpoint")
        #TextCtrls
        self.field_host = wx.TextCtrl(self.panel,name="host")
        self.field_port = wx.TextCtrl(self.panel,name="port")
        self.field_password =
wx.TextCtrl(self.panel,style=wx.TE_PASSWORD,name="password")
        self.field_super_endpoint = wx.TextCtrl(self.panel,name="endpoint")
        #Buttons
        self.btn_apply = wx.Button(self.panel,label="Apply")
        self.btn_apply.Bind(wx.EVT_BUTTON,self.on_apply)
        self.btn_quit = wx.Button(self.panel,label="Quit")
        self.btn_quit.Bind(wx.EVT_BUTTON,self.on_quit)
        #Add to Sizers
        self.sizer_controls.AddMany([(self.label_host,1,wx.ALL|wx.EXPAND),
                                     (self.field_host,1,wx.ALL|wx.EXPAND),
                                     (self.label_port,1,wx.ALL|wx.EXPAND),
                                     (self.field_port,1,wx.ALL|wx.EXPAND),
                                     (self.label_password,1,wx.ALL|
wx.EXPAND),
```

```
                                        (self.field_password,1,wx.ALL|
wx.EXPAND),
                                        (self.label_super_endpoint,1,wx.ALL|
wx.EXPAND),
                                        (self.field_super_endpoint,1,wx.ALL|
wx.EXPAND),
                                        (self.btn_apply,1,wx.ALL|wx.EXPAND),
                                        (self.btn_quit,1,wx.ALL|wx.EXPAND)])
        self.sizer_main.AddMany([(self.text_explainer,1,wx.ALL|wx.EXPAND),
                                 (self.sizer_controls,1,wx.ALL|wx.EXPAND)])

    def on_apply(self, event):
        obs_connection = {}
        super_endpoint = {}
        fields = [self.field_host, self.field_port, self.field_password]
        for field in fields:
            value = field.GetValue()
            name = field.GetName()
            if value == "":
                dlg = wx.MessageDialog(self,message="Fields cannot be left
empty. Make sure all fields are filled out and try again.",caption="Fields
Cannot Be Empty",style=wx.OK|wx.ICON_ERROR)
                dlg.ShowModal()
                break
            else:
                if name != "port":
                    obs_connection[name] = value
                else:
                    obs_connection[name] = int(value)
        valid_auth = self.test_connection(obs_connection)
        if valid_auth:
            self.write_obs_connection(obs_connection)
        super_endpoint['endpoint'] = self.field_super_endpoint.GetValue()
        self.write_super_endpoint(super_endpoint)
        self.Destroy()

    def test_connection(self,obs_connection):
        host = obs_connection['host']
        port = obs_connection['port']
        password = obs_connection['password']
        try:
            cl =
obs.ReqClient(host=host,port=port,password=password,timeout=3)
            ver = cl.get_version().obs_version
            if ver is not None:
                return True
        except Exception:
            return False
        return False

    def write_obs_connection(self, obs_connection):
        with open('./data/settings/obs_settings.json','w') as file:
            json.dump(obs_connection,file)

    def write_super_endpoint(self, super_endpoint):
        with open('./data/settings/super_endpoint.json','w') as file:
            json.dump(super_endpoint,file)
```

```
    def on_quit(self):
        self.Destroy()

class Splash(SplashScreen):
    def __init__(self,parent=None):
        bitmap = wx.Bitmap("./data/icons/splash.png",type=wx.BITMAP_TYPE_PNG)
        splash = wx.adv.SPLASH_CENTRE_ON_SCREEN | wx.adv.SPLASH_TIMEOUT
        duration = 3000
        super(Splash, self).__init__(bitmap=bitmap,
                                     splashStyle=splash,
                                     milliseconds=duration,
                                     parent=None,
                                     id=-1,
                                     pos=wx.DefaultPosition,
                                     size=wx.DefaultSize,
                                     style=wx.STAY_ON_TOP | wx.BORDER_NONE)
        self.Bind(wx.EVT_CLOSE, self.on_exit)

    def on_exit(self, event):
        event.Skip()
        self.Hide()

def main():
    obs_settings = load_obs_settings()
    endpoint = load_super_endpoint()
    if obs_settings is None:
        app=[]; app = wx.App(None)
        frame = FirstBoot()
        app.SetTopWindow(frame)
        app.MainLoop()
    obs_settings = load_obs_settings()
    if obs_settings is not None:
        app=[]; app = wx.App(None)
        frame = GUI("NROBS",obs_settings,endpoint)
        app.SetTopWindow(frame)
        app.MainLoop()
    else:
        main()

if __name__ == "__main__":
    main()
```

index.html

```
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>OBS Lower Third</title>
  <style>
    @font-face {
      font-family: 'GothamBlack';
      src: url('./fonts/Gotham XNarrow Black.otf') format('opentype');
    }

    body {
      margin: 0;
      padding: 0;
      background: transparent;
      overflow: hidden;
    }

    #lower-third {
      position: absolute;
      bottom: 115px;
      left: 60px;
      display: flex;
      justify-content: flex-start;
      pointer-events: none;
    }

.banner {
  position: relative; /* make it a positioning context for .blue-bar */
  font-family: 'GothamBlack', sans-serif;
  font-size: 2.5em;
  background-color: white;
  color: black;
  width: 1540px;
  height: 96px;
  padding: 0.03em 0.5em 0.03em 0.3em;
  box-shadow: none;
  display: inline-block;
  text-transform: uppercase;
  transform-origin: top;
  transform: scaleY(0);
  opacity: 0;
  transition: transform 1.2s ease-out, opacity 0.5s ease-out;
}

.blue-bar {
  position: absolute;
  bottom: 3px;
  left: 2px;
  right: 2px;
  height: 10px;
  background-color: #1d59c9;
}
    #lower-third.slide-in .banner {
```

```
      transform: scaleY(1);
      opacity: 1;
    }

    #lower-third.slide-out .banner {
      transform: translateY(100%);
      opacity: 0;
      transition: transform 0.6s ease-in, opacity 0.6s ease-in;
    }

    .hidden {
      display: none;
    }
  </style>
</head>
<body>
  <div id="lower-third" class="hidden">
    <div class="banner">
  <span id="text"></span>
  <div class="blue-bar"></div>
</div>
  </div>
<script src="https://unpkg.com/fitty/dist/fitty.min.js"></script>
<script>
  const evtSource = new EventSource("super.php");
  const lowerThird = document.getElementById("lower-third");
  const textSpan = document.getElementById("text");

  // Initialize fitty and save the instance
  const fittyInstance = fitty(textSpan, { minSize: 10, maxSize: 80 });

  evtSource.onmessage = function (event) {
    const data = event.data.trim();

    if (data === '*') {
      // Animate banner out
      lowerThird.classList.remove("slide-in");
      lowerThird.classList.add("slide-out");
    } else {
      // Animate out first
      lowerThird.classList.remove("slide-in");
      lowerThird.classList.add("slide-out");

      // Wait for animation to complete before updating text
      setTimeout(() => {
        textSpan.textContent = data;

        // Resize the text to fit the banner
        fittyInstance.fit();

        // Animate back in
        lowerThird.classList.remove("slide-out", "hidden");
        lowerThird.classList.add("slide-in");
      }, 650); // Match slide-out transition
    }
  };
</script>
```

```
</body>
</html>
```

send_super.php

```
<?php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $text = trim($_POST['text'] ?? '');
    file_put_contents('super.txt', $text);
    echo "Updated";
}
```

super.php

```
<?php
header('Content-Type: text/event-stream');
header('Cache-Control: no-cache');

$last = '';
while (true) {
    clearstatcache();
    $data = trim(@file_get_contents('super.txt'));
    if ($data !== $last) {
        echo "data: $data\n\n";
        ob_flush();
        flush();
        $last = $data;
    }
    sleep(1);
}
```