Decent Sampler Preset Tuner

22 Oct 2024 by Frank Faruk Ceviz

frankfaruksampling.site

Simple Description

The Decent Sampler Preset Tuner is a Python script designed to adjust the tuning for specified musical notes within a Decent Sampler .dspreset file. It allows users to apply individual tuning adjustments to multiple notes, update the background image, and provides flexible file handling through drag-and-drop support or a file selection dialog. The script ensures the original image format is preserved unless a different extension is specified.

README

Decent Sampler Preset Tuner

Overview

The Decent Sampler Preset Tuner is a utility script that allows you to:

- **Adjust Tuning for Specific Notes**: Specify individual tuning adjustments for multiple musical notes within a Decent Sampler `.dspreset` file.
- **Update Background Image**: Change the background image used in the preset while preserving the original image format.
- **Flexible File Handling**: Easily select the `.dspreset` file via drag-and-drop (on supported systems) or through a graphical file selection dialog.

Features

- **Individual Note Tuning**: Apply different tuning values to specific notes.
- **Image Format Preservation**: Retain the original image file extension unless a new one is specified.

- **User-Friendly Interface**: Simple prompts guide you through the process.
- **Error Handling**: Provides clear messages for invalid inputs or errors.

Prerequisites

- **Python 3.x**: Ensure you have Python 3 installed on your system.
- **tkinter Library**: The script uses `tkinter` for the file dialog, which is included with most Python installations.

Installation

1. **Download the Script**
   Save the script as `decent_sampler_preset_tuner.py` on your computer.
2. **Verify Dependencies**
   - **tkinter**: This library is typically included with Python. If you encounter issues, you may need to install it or run the script in an environment where it's available.

Usage

Running the Script

Open a command prompt or terminal window, navigate to the directory containing the script, and run:

bash

Copy code

```
python decent_sampler_preset_tuner.py
```

Note for Windows Users: You can drag and drop the `.dspreset` file onto the script file to launch it with the file pre-selected.

Step-by-Step Instructions

1. **Select the `.dspreset` File**
    - If you didn't use drag-and-drop, a file dialog will appear.
    - Navigate to and select your Decent Sampler `.dspreset` file.
2. **Enter the New Background Image Name**
    - Prompt: `Enter the new image file name for bgImage (enter 0 to leave unchanged):`
    - **Options**:
        - **Change the Image**: Enter the new image file name (e.g., `new_background.png`).
            - If you omit the extension, the script uses the original image's extension.
        - **Keep Existing Image**: Enter `0` to leave the background image unchanged.
3. **Specify Music Notes and Tuning Adjustments**

Prompt:
python
Copy code

```
Enter the music notes and tuning adjustments in the format 'Note=TuningValue', separated by commas.

For example: G#=-0.51, A#=-0.25, C#=0.1

Your input:
```

    -
    - **Input Format**:
        - **Note-Tuning Pairs**: Enter pairs like `Note=TuningValue`.
        - **Separators**: Separate multiple pairs with commas.

Example:
shell
Copy code

```
G#=-0.49, A=-0.11, C#=0.05
```

    -
4. **Script Processing**
    - The script applies the specified tuning adjustments and updates the background image if provided.
    - Progress messages will be displayed in the console.

5. **Output**
    - The updated `.dspreset` file is saved in the same directory as the original file, with `_updated` appended to the filename.
        - **Example**: `my_instrument.dspreset` becomes `my_instrument_updated.dspreset`.

Confirmation message:
vbnet
Copy code
```
Tuning updated for notes: G#, A, C#

bgImage updated to: new_background.png

Updated file saved as:
/path/to/your/my_instrument_updated.dspreset
```

    -

Examples

Example 1: Adjusting Tuning for Multiple Notes

Input:
arduino
Copy code
```
Enter the new image file name for bgImage (enter 0 to
leave unchanged): 0
```

less
Copy code
```
Your input: G#=-0.49, A=-0.11
```

-
- **Result**:
    - Tuning adjustments are applied to samples containing notes `G#` and `A`.
    - Background image remains unchanged.

Example 2: Changing the Background Image and Tuning

Input:
arduino
Copy code
```
Enter the new image file name for bgImage (enter 0 to leave unchanged): new_bg_image
```

mathematica
Copy code
```
Your input: C#=0.1, D#=-0.2
```

-
- **Result**:
  - The `bgImage` attribute is updated to use `new_bg_image` with the original image's extension.
  - Tuning adjustments are applied to samples containing notes `C#` and `D#`.

Notes

- **Sample Name Format**:
  - The script assumes sample names follow a pattern like `name="..._<Note><Octave>_..."`.
  - It uses underscores (`_`) to accurately match notes in the sample names.
- **Supported Notes**:
  - The script supports both single-letter notes (e.g., `A`, `B`) and notes with sharps/flats (e.g., `G#`, `Ab`).
- **Tuning Values**:
  - Tuning values are strings and should represent the desired tuning adjustment (e.g., `-0.49`, `0.1`).

Error Handling

- **Invalid Note-Tuning Format**:
  - If you enter an invalid format, the script will display an error and exit.

Example:
python
Copy code
```
Invalid format for note 'G# -0.49'. Expected format is 'Note=TuningValue'.
```

- **No File Selected**:

If you cancel the file selection dialog, the script will exit with a message:
yaml
Copy code
```
No file selected. Exiting.
```

- **Exceptions**:
  - Any unexpected errors will be reported with details for troubleshooting.

Troubleshooting

- **tkinter Not Found**:
  - If you receive an error related to `tkinter`, ensure it's installed or use a Python environment that includes it.
- **Tuning Not Applied to Certain Notes**:
  - Ensure that the notes in your input match the format in the sample names.
  - For single-letter notes, the script uses underscores to accurately identify them.

License

This script is provided "as is" without warranty of any kind. Use it at your own risk. Feel free to modify and adapt it to suit your needs.

Contributing

Contributions are welcome! If you have suggestions for improvements or encounter any issues, please feel free to reach out.

Acknowledgements

- **Decent Sampler**: Decent Sampler is a free sampler plugin. This script is designed to assist users in customizing their Decent Sampler presets.

Thank you for using the Decent Sampler Preset Tuner! If you have any questions or need further assistance, feel free to reach out.

Copy and Paste the full code below

```
import sys

import os

import re

import tkinter as tk

from tkinter import filedialog


def select_file():

   # Check if a file path is provided via command-line
arguments (for drag-and-drop)

   if len(sys.argv) > 1:

       file_path = sys.argv[1]

   else:

       # Create a file dialog for the user to select the
file
```

```
        root = tk.Tk()
        root.withdraw()  # Hide the main window
        file_path = filedialog.askopenfilename(
            title="Select Decent Sampler .dspreset file",
            filetypes=[("Decent Sampler Preset",
"*.dspreset"), ("XML Files", "*.xml"), ("All Files",
"*.*")]
        )
    return file_path


def update_tuning(input_text, tuning_dict):
    for note, tuning_value in tuning_dict.items():
        # Escape any special regex characters in the note
        note_escaped = re.escape(note)


        # Build the note pattern using underscores to
delimit the note
        pattern =
rf'(<sample[^>]+name="[^"]*?_{note_escaped}(\d+)?_.*?"[^>]
*?)(\s*/?>)'


        # Function to add or update tuning
        def add_tuning(match):
            tag_content = match.group(1)
            closing = match.group(3)
            if 'tuning="' not in tag_content:
                # Insert tuning attribute before the
closing tag
```

```
                return f'{tag_content}
tuning="{tuning_value}"{closing}'

            else:

                # Update existing tuning value

                updated_tag = re.sub(r'tuning="[^"]*"',
f'tuning="{tuning_value}"', match.group(0))

                return updated_tag



        # Update the input text for the current note

        input_text = re.sub(pattern, add_tuning,
input_text)



    return input_text



def update_bgImage(input_text, new_image_name):

    # Regular expression to find bgImage="Images/..."
attribute

    pattern = r'(bgImage="Images/)([^"]*)(")'



    match = re.search(pattern, input_text)

    if match:

        original_image_name = match.group(2)

        original_extension =
os.path.splitext(original_image_name)[1]  # Includes the
dot, e.g., '.png'

    else:

        original_extension = '.png'  # Default to .png if
not found
```

```
    if new_image_name != '0':
        # If the user did not specify an extension, use the original extension
        if not os.path.splitext(new_image_name)[1]:
            new_image_name += original_extension

        # Function to perform replacement
        def replace_bg_image(match):
            return f'{match.group(1)}{new_image_name}{match.group(3)}'

        # Replace the bgImage attribute value with the new image name
        updated_text = re.sub(pattern, replace_bg_image, input_text)
    else:
        # Do not change the input text
        updated_text = input_text

    return updated_text


def main():
    # Select the file
    file_path = select_file()

    if not file_path:
        print("No file selected. Exiting.")
        sys.exit(1)
```

```
    # Ask for the new image file name
    new_image_name = input("Enter the new image file name
for bgImage (enter 0 to leave unchanged): ").strip()

    # Ask for the music notes and their tuning adjustments
    print("Enter the music notes and tuning adjustments in
the format 'Note=TuningValue', separated by commas.")
    print("For example: G#=-0.51, A#=-0.25, C#=0.1")
    notes_input = input("Your input: ")

    # Parse the input into a dictionary
    tuning_dict = {}
    notes_list = [note.strip() for note in
notes_input.split(',') if note.strip()]
    for note_pair in notes_list:
        if '=' in note_pair:
            note, tuning = note_pair.split('=')
            note = note.strip()
            tuning = tuning.strip()
            tuning_dict[note] = tuning
        else:
            print(f"Invalid format for note '{note_pair}'.
Expected format is 'Note=TuningValue'.")
            sys.exit(1)

    try:
        # Read the content of the file
```

```
        with open(file_path, "r", encoding='utf-8') as
file:
            content = file.read()


        # Update the content
        content = update_tuning(content, tuning_dict)
        content = update_bgImage(content, new_image_name)


        # Save the updated content back to a new file
        file_dir, file_name = os.path.split(file_path)
        file_base, file_ext = os.path.splitext(file_name)
        updated_file_name =
f"{file_base}_updated{file_ext}"
        updated_file_path = os.path.join(file_dir,
updated_file_name)


        with open(updated_file_path, "w", encoding='utf-8')
as file:
            file.write(content)


        print(f"\nTuning updated for notes: {',
'.join(tuning_dict.keys())}")
        if new_image_name != '0':
            print(f"bgImage updated to: {new_image_name}")
        else:
            print("bgImage unchanged.")
        print(f"Updated file saved as:
{updated_file_path}")
```

```
    except FileNotFoundError:
        print("File not found. Please check the file path and try again.")
    except Exception as e:
        print(f"An error occurred: {e}")


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