Project and Makefile Documentation

Introduction

This project uses a series of Makefile files to manage audio processing and signal simulation using Csound and audio processing tools like FAUST. Each Makefile is designed to automate different phases of the process from generating audio files to processing them with various configurations depending on the needs.

Project Structure

Main Makefile:

- **Description**: Coordinates the execution of secondary Makefiles for Csound processing and audio processing.
- **Variables**:
  - `CS` : Specifies the configuration to use (speeding up or slowing down).
  - `CSOUND_DIR` : Directory containing the Makefile for Csound processing.
  - `AUDIO_PROCESSING_DIR` : Directory containing the Makefile for audio processing using FAUST.
- **Main Targets**:
  - `all` : Executes the `csound_process` and `audio_process` targets.
  - `csound_process` : Executes the Makefile in the `$(CSOUND_DIR)` directory.
  - `audio_process` : Executes the Makefile in the `$(AUDIO_PROCESSING_DIR)` directory with the parameter `TIPO=$(CS)` .
  - `clean` : Cleans up temporary files generated by both processes.
  - `cleanAll` : Removes all output directories and `.wav` files in the `rec` folder.

Csound Processing Makefile (Makefile in `csound_$(CS)`):

- **Description**: Processes `.sco` files using a specific `.orc` file to generate `.wav` audio output.
- **Variables**:
  - `ORC_FILE` : Specifies the `.orc` file to use.
  - `SCO_FILES` : Finds all `.sco` files in the current directory.
  - `OUTPUT_PATH` : Path for the generated output files.
- **Main Targets**:
  - `all` : Creates the output directory and processes the `.sco` files.
  - `process_sco_files` : Processes each `.sco` file to generate the corresponding `.wav` files.
  - `modificaSco` : Runs a `Python` script to modify `.sco` files with `BPM` and `COUNTER` parameters.
  - `clean` : Removes all generated files in the output directory.

FAUST Processing Makefile (Makefile in `./simulazioniFaust`):

- **Description**: Executes audio simulations on .wav files using a specific FAUST executable.
- **Variables**:
  - MONDO: FAUST executable to use.
  - SRCDIR: Directory containing the `.wav` files to process.
  - OUTDIR: Output directory for processed files.

    - REAPER_PROJECT: Associated Reaper project for listening.

- **Main Targets**:
    - `all`: Processes the `.wav` files in `$(SRCDIR)` and `$(SRCDIRSTROF)` with the FAUST executable.
    - `open`: Removes generated files in the output directory.
    - `clean`: Removes the entire output directory.
    - `cleanAll`: Rimuove l'intera directory di output.

FAUST Generation Makefile (Makefile in `./faustDSP`):

- **Description**: Compiles and prepares the FAUST executable for audio file processing.
- **Variables**:
    - `FAUST`: Specifies the `.dsp` FAUST file to compile.
    - `MONDO`: Name of the generated FAUST executable file.

- **Main Targets**:
    - `all`: Compiles the `.dsp` file and moves it to the specified directory.
    - `process`: Executes the compilation and transfer of the FAUST executable.
    - `clean`: Removes intermediate files generated during the process.

How to Use the Project

Initial Setup:

- Ensure that the required tools (Csound, FAUST, Python) are installed.
- Configure the `CS` variable in the main `Makefile` to choose between speeding up and slowing down modes.

Running the Project:

- Run `make all` in the main Makefile to start the entire audio processing pipeline.
- To run specific phases of the process, use the appropriate targets (`csound_process`, `audio_process`).

Cleaning the Project:

- Use `make clean` in the main Makefile to remove temporary files.
- Use `make cleanAll` for a complete cleanup, including the removal of output files.

Conclusion

This modular structure efficiently manages various complex audio processing tasks, offering flexibility and control through Makefiles. Each of them is specifically designed for a phase of the process, allowing easy extension or modification of functionalities as needed.