# Descripción General

Este módulo contiene todo el procesamiento que se realiza con una señal de audio en el sistema duetto-Sound Lab. Es un módulo independiente del sistema por lo que puede ser extendido o reutilizado fácilmente. Lo componen varios sub módulos que han sido divididos en cuanto a las funcionalidades o áreas de procesamiento más importantes.

Módulo Duetto\_Core

Documentación para el programador

# Sub-módulos

El módulo Duetto\_Core publica los siguientes módulos:

* Audio Signals 🡪 Representación de las señales de audio.
* Clasification 🡪Métodos de clasificación.
* Segmentation 🡪 Métodos de segmentación
* SignalProcessors 🡪 Procesamiento de señales
* Cursors (deprecated) Manejo de cursores

A continuación se profundizará en cada módulo.

# Módulo Audio Signals

## **Descripción General**

Este módulo define una jerarquía de clases para representar las señales de audio.

## **Clases**

1. Audio Signal 🡪 Abstracción de una señal de audio.
2. FileAudioSignal 🡪Una señal de audio procedente del sistema de ficheros.
3. WavFileSignal 🡪 Una señal de audio de tipo wav.

***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

## **Clase Audio Signal**

Descripción

Esta clase representa la abstracción de una señal de audio. Encapsula las propiedades fundamentales de toda señal de audio. Se propone utilizarla como si fuera una clase abstracta y la base de la jerarquía de señales de audio.

***Variables***

1. data (array) //los datos de la señal
2. samplingRate (integer) // el sampling rate de grabación de la señal
3. bitDepth (integer) // la profundidad de bits en los datos de la señal
4. channels (integer) // la cantidad de canales que posee la señal (reservado para futuro uso)
5. playStatus (enum) //el estatus de reproducción en que se encuentra la señal. Puede tomar los siguientes valores [PLAYING, PAUSED, STOPPED, RECORDING]
6. playAudio (Object) // objeto utilizado para comunicarse con los dispositivos al reproducir y grabar
7. stream (Object) // objeto que establece un flujo de datos para la grabación y reproducción. Se utiliza estrechamente vinculado a playAudio
8. playSection (tuple3) // región que está siendo reproducida actualmente. Los valores son los índices en el array data (inicio,final,actual).
9. name (string) // el nombre de la señal

***Métodos***

1. openNew(samplingRate, duration(ms), bitDepth(integer), whiteNoise (bool) ) 🡪 genera una nueva señal de audio con las características brindadas en los parámetros.
2. resampling (samplingRate) 🡪 la señal es re muestreada al nuevo samplingRate.
3. getMaximumValueAllowed() y getMinimumValueAllowed() devuelven el máximo y minimo valor que puede contener el array data debido al bitDepth usado.
4. \_playCallback() y \_recordCallback() son dos métodos invocados por PyAudio al reproducir y Grabar respectivamente. Interaccionan con stream, playStatus, playSection.
5. generateWhiteNoise ( duration (s), begin\_at(integer) ) genera ruido blanco en la señal a partir del índice begin at por duration time en seg. begin at < data.size
6. play(startIndex (integer), endIndex(integer), speed(integer)) reproduce la señal dese el índice start hasta el end a una velocidad speed. speed está expresada en %, startIndex<= endIndex<=data.size
7. stop() detiene la señal si se estaba reproduciendo o detiene la grabación si se estaba grabando
8. pause() detiene la señal si se estaba reproduciendo.
9. record(speed (integer)) comienza la grabación de una señal con las mismas características que la actual (canales y bitDepth) pero a distinta velocidad de grabación que la actual en términos de SR. speed está expresado en %
10. toWav() devuelve una señal de audio de tipo WavAudioSignal. Este método se propone para mantener un mismo tipo de señal en el procesamiento y permitir varios tipos de señales en el sistema. El funcionamiento sería abrir la señal con la correspondiente clase de la jerarquía y luego transformarla a wav para trabajar en el sistema con un solo tipo de señales.

***Invariantes, Reglas y Convenciones***

1. playSection es una tupla de 3 valores (inicio,final,actual). inicio<= actual <= final. final <= data.size
2. playStatus = PLAYING luego de ejecutar play
3. playStatus = STOPPED luego de ejecutar stop
4. playStatus = PAUSED luego de ejecutar pause
5. playStatus = RECORDING luego de ejecutar record
6. stream = None si no se está grabando o reproduciendo

***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

## **Clase File Audio Signal**

Descripción

Hereda de AudioSignal. Representa todas las señales de audio provenientes de un archivo en el sistema de ficheros. Permite mayor modularidad en el código al factorizar las funcionalidades que se realizan sobre los distintos formatos de audio provenientes del sistema de ficheros. Permite incrementar el número de formatos de audio que manipula el sistema DUETTO con gran facilidad implementando clases derivadas que abran el formato deseado. Luego se ha de implementar la transformación a señal de tipo wav como se explicó anteriormente. Puede ser extendida su definición a las señales que se ubiquen con una url. Contiene los métodos polimórficos open y save. Se propone utilizarla como si fuera una clase abstracta.

***Variables***

1. path (string) // ruta en el Sistema de ficheros que se corresponde con la ubicación de la señal.

***Métodos***

1. open(string filepath) //abre una señal proveniente del fichero ubicado en la dirección filepath
2. save (string filepath) // guarda el fichero actual en la ruta indicada.

***Invariantes, Reglas y Convenciones***

***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

## **Clase Wav Audio Signal**

Descripción

Es la implementación de las señales de tipo wav que se encuentran en el sistema de ficheros. Hereda de FileAudioSignal.

***Variables***

1. ***userData (array)*** //los datos que se desean guardar dentro de la señal en disco para una próxima lectura. Puede usarse en un futuro en módulo pickle

***Métodos***

1. ***readUserChunk*** lee de un fichero abierto el chunk de datos almacenado por el sistema en un análisis previo.
2. ***open*** abre una señal de audio de tipo wav
3. ***save*** hace persistente a disco la señal analizada.
4. ***smallSignal*** devuelve una señal de tipo wav de pequeño tamaño (aprox 40 ms) que es representativa de la señal actual. Para señales menores de 100 ms se devuelve la señal completa.

***Invariantes, Reglas y Convenciones***

Las mismas que AudioSignal

# Módulo Clasification

## **Descripción General**

En este módulo se definen las vías de clasificación de señales de audio implementadas. Este módulo está muy relacionado con el de segmentación. Para realizar la clasificación se procedería a segmentar la señal de audio y a los elementos detectados se le calcularían los parámetros que determine el modelo de clasificación. Posteriormente se aplicaría la clasificación a los segmentos detectados.

No hay ningún método implementado y está reservado para uso futuro.

# Módulo Segmentation

## **Descripción General**

Este módulo se integra fuertemente con el de clasificación. El proceso de clasificación se propone realizarlo previa segmentación de la señal. Extracción de parámetros en los segmentos y posterior implementación de uno o varios modelos de clasificación. Todos estos procesamientos serán realizados por este módulo.

## **Clases**

La segmentación se propone realizarla mediante la definición de Febles Torres, Yasel (Tesis de Diploma) en cuanto a la segmentación integrando varios dominios de definición. Se proponen varias clases para la detección de elementos y la jerarquía de elementos.

***Directorios***

1. ***Detectors🡪*** Clases que realizan la detección de elementos.
2. ***Elements 🡪*** Definición de elementos.

Elements

Directorio que contiene la definición de los elementos.

## Clase Element

Representa la mínima pieza de información que conforma un segmento a clasificar. Es la clase base de los elementos. Interacciona con las clases detectoras de elementos y con los segmentos. Tiene un conjunto de variables para la representación visual de sus características.

Las variables visuales están divididas en categorías. Cada categoría contiene un array con listas [***visualcontrol, bool***] en las cuales se encuentran los controles visuales que deben ser mostrados (el texto o las figuras con las mediciones de parámetros por ejemplo) y el bool de su visibilidad. Esto permite que el control que utilice la clase pueda visualizarlo solo mediante la adicción de cada elemento visual al control que se utiliza para graficar. En particular esto se hace en el Signal Visualizer en el método drawElements.

Los herederos deben implementar el cálculo de los controles visuales. Esto permite también la granularidad en la visualización del elemento ya que se puede observar subconjuntos de sus controles visuales haciendo más personalizable la interacción con el usuario. Es fácilmente extensible a otra plataforma visual mediante la re implementación del cálculo de los controles para la nueva plataforma. Clase Abstracta.

***Variables***

1. ***signal (Signal)*** //La señal sobre la que se define el elemento. Debe ser una señal de Audio.
2. ***visible*** (bool) la visibilidad de este elemento.
3. ***visual\_text , visual\_locations, visual\_figures, visual\_peaksfreqs*** arreglos con los controles visuales para cada categoría.

***Métodos***

1. ***visualwidgets ()*** método que itera por los componentes visuales del elemento.

## Clase OneDimensionalElement

Representa los elementos de una dimensión. Define el cálculo de parámetros para elementos de una dimensión. Clase Abstracta.

***Variables***

1. ***indexFrom, indexTo (int)*** //Índices temporales de inicio y fin del elemento en la señal.
2. ***parameterDecimalPlaces*** constante entera para las cifras decimales en el cálculo de los parámetros.

## Clase TwoDimensionalElement

Representa los elementos de dos dimensiones. Define el cálculo de parámetros para elementos de dos dimensiones. Clase Abstracta.

***Variables***

1. ***matrix (int[,])*** //Matriz de representación del elemento.

## Clase OscilogramElement

Implementación de elemento de una dimensión con el oscilograma como PA. Esta clase se implementa en particular para el funcionamiento en Sound Lab. Contiene subelementos espectrales que son instancias de SpecgramElement. Contienen un grupo de funciones que calculan todos los parámetros temporales y espectrales (en los subelementos) que se necesiten en la aplicación.

Contiene un subelemento espectral como mínimo que ocupa toda la gama de frecuencias posible y es en el que se realizan los cálculos de parámetros. Las mediciones espectrales se realizan en las columnas de la matriz de representación de este elemento espectral en función del lugar de las mediciones que se solicite. Las mediciones espectrales pueden ser realizadas en una multitud de lugares que permite definir la estructura SpectralMeasurementLocation.

***Variables***

1. ***twoDimensionalElements*** (array)🡪 los elementos espectrales de dos dimensiones.
2. ***specgramSettings*** (struct SpecgramSettings) ***🡪*** las opciones para la confección del espectrograma en el que se definen los subelementos bidimensionales.
3. ***number*** int. El número de este elemento en la detección realizada. Es utilizado fundamentalmente para la visualización.
4. ***parameters*** dict. Se utiliza para la implementación del patrón memoize en la mayoría de las funciones que calculan parámetros.

***Métodos***

1. ***startTime, endTime, duration, distanceFromStartToMax, peekToPeek, rms***. Métodos que calculan parámetros temporales.
2. ***peakFreqAverage, maxFreqAverage, minFreqAverage, peakFreq, peakAmplitude, minFreq, maxFreq, bandwidth, peaksAbove.*** Métodos que calculan parámetros espectrales.
3. ***freq\_min\_max\_band\_peaksAbove,*** ***peak\_f\_a.*** Métodos auxiliares que calculan varios parámetros espectrales en una sola ejecución.
4. ***computeTwoDimensionalElements.*** Método que ejecuta la detección de subelementos espectrales.
5. ***spectralElements*** La cantidad de subelementos espectrales que contiene este elemento temporal.

## Clase SpecgramElement

Clase que representa los elementos espectrales. Contiene métodos que calculan parámetros de elementos espectrales. Implementa los controles visuales heredados de Element para la visualización.

Detectors

Es un directorio que contiene a todas las clases que realizan la detección. Aquí se encuentran los detectores de elementos ***ElementsDetectors*** y los ***FeatureExtractionDetectors(deprecated)***

## Clase ElementDetector

Es la clase base de los detectores de elementos. De ella heredan los tipos de detectores dimensionales que se implementen para cada dominio de representación de la señal.

***Variables***

1. ***elements (array)*** //los elementos que se detectaron.

***Métodos***

1. ***detect(signal s)*** método abstracto que realiza la detección de elementos en la señal s. Luego de ejecutar el método detect elements contiene los elementos detectados.
2. ***mergeIntervals(array of int tuple2 elements\_array, int distancefactor) elements\_array*** almacena las detecciones de elementos unidimensionales como tuplas de (int,int). El método mezcla las tuplas de ***elements\_array*** si la distancia entre su final y el comienzo de la próxima no es menor que el porcentaje ***distancefactor*** con relación Es usado frecuentemente en la detección de elementos de una dimensión.

## Clase OneDimensionalElementDetector

Realiza la detección de elementos de una dimensión utilizando un PA de una dimensión como el oscilograma. Hereda de ElementsDetector implementando el método detect que devolverá un array con elementos de una dimensión. Contiene definiciones de enums para el tipo de threshold automático y para el tipo de método de detección unidimensional que se utilizará.

## Clase TwoDimensionalElementDetector

Realiza la detección de elementos bidimensionales en PA de dos dimensiones como el espectrograma. Funciona de la misma manera que la clase de deteccion de elementos de una dimensión.

# Módulo SignalProcessors

## **Descripción General**

En este módulo se implementan la mayoría de las acciones de edición realizadas sobre una señal. Contiene un conjunto de clases que permiten realizar los procesamientos. Un procesador de señales es una clase que contiene métodos que reciben una señal de audio, un intervalo de la misma y realizan modificaciones sobre los valores de la señal. No modifican el tamaño de la misma sino sus valores.

## **Clases**

1. SignalProcessor 🡪 Clase base de la jerarquía de procesadores de señales.
2. FilterSignalProcessor🡪 Clase que implementa filtros en el dominio de la frecuencia sobre una señal.
3. EditionSignalProcessor 🡪 Procesador de señales de edición. Permite cortar, copiar y pegar fragmentos de una señal en otra. (deprecated, se debe usar el clipboard que provee PyQt)
4. CompensationSignalProcessor 🡪 Clase que realiza la compensación de las bocinas para reproducir una señal de audio con mayor fidelidad. (deprecated, no debe encontrarse como signal processor sino como parte de audio signal)
5. CommonSignalProcessor 🡪 Clase que implementa las operaciones comunes sobre una señal. Silencio, reverse etc.

## **Clase SignalProcessor**

Descripción

Esta clase es la base de la jerarquía de los procesadores de señales. Implementa funciones que modifican los valores de la señal. Interactúa con la clase AudioSignal.

***Variables***

1. signal 🡪la señal que se modificará.

***Métodos***

1. ***checkIndexes*** (indexFrom (integer), indexTo (integer)) 🡪 metodo auxiliary que verifica la factibilidad de los índices recibidos. Verifica que sean positivos y menores que la longitud de la señal.

***Invariantes, Reglas y Convenciones***

***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

## **Clase FilterSignalProcessor**

Descripción

Esta clase realiza el filtrado sobre la señal. Se pueden realizar filtros en el dominio del tiempo y en la frecuencia. Actualmente se encuentra implementado en el dominio de la frecuencia. Posee un método filter que es el que se debe invocar para realizar el filtrado variando sus parámetros para los distintos filtros.

***Variables***

***Métodos***

1. ***filter*** (indexFrom(int), indexTo(int) ,filterType (enum) ,Fc(int),Fl(int),Fu(int)) 🡪realiza un filtro en el intervalo de la señal [indexFrom: indexTo] el tipo de filtro se especifica con filterType que puede tomar los valores BAND\_PASS, BAND\_STOP, LOW\_PASS y HIGH\_PASS. Fc, Fl y Fu son las frecuencias de corte especificadas para los filtros.(cut,low y upper). En un futuro se debe implementar los filtros en el dominio del tiempo IIR, FIR.

***Invariantes, Reglas y Convenciones***

***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

## **Clase EditionSignalProcessor**

Descripción

Esta clase implementa las operaciones de edición (cortar, copiar, pegar) en una señal. Realiza las mismas utilizando una variable interna para almacena los valores. Debe ser removida y sustituida por la utilización del clipboard que provee PyQt.

***Variables***

1. Clipboard 🡪 el array con los valores de la señal

***Métodos***

1. ***cut*** (start\_index (integer), end\_index (integer)) 🡪 corta el fragmento de la señal delimitado por los índices***.***
2. ***copy*** (start\_index (integer), end\_index (integer)) 🡪 copia el fragmento de la señal delimitado por los índices***.***
3. ***paste*** (start\_index (integer)) 🡪 pega el contenido de la variable clipboard en la posición indicada en start\_index.

***Invariantes, Reglas y Convenciones***

La variable clipboard luego de ejecutar copy o cut contiene el fragmento de señal delimitado por los índices.

La señal luego de ejecutar cut no puede contener el fragmento que se almacena en clipboard y su tamaño se reduce en end\_index - start\_index.

Al ejecutar paste se cumple que la señal contiene la señal almacenada en clipboar en la posición indicada como parámetro y su tamaño crece en el size del clipboard.

***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

## **Clase CompensationSignalProcessor**

Descripción

Esta clase se encarga de realizar la compensación de las bocinas de forma que el sonido emitido sea idéntico al que fue grabado. Esto no sucede producto de la confección de las bocinas y los micrófonos comunes que potencian más determinadas frecuencias que otras. Esta en un lugar incorrecto del diseño ya que debe integrarse con la forma de reproducir el sonido y no como un procesamiento de la señal.

***Variables***

***Métodos***

***Invariantes, Reglas y Convenciones***

***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

## **Clase CommonSignalProcessor**

Descripción

Esta clase implementa las operaciones comunes con una señal.

***Variables***

***Métodos***

1. ***scale*** 🡪 modifica las amplitudes de la señal. Puede multiplicar la señal por un valor fijo (normalize) o por una función (creciente o decreciente y puede ser elegida de una lista sin, linear etc…)
2. ***setSilence***🡪 convierte en silencio una sección de la señal
3. ***reverse***🡪 invierte el orden de las muestras de un fragmento de la señal
4. ***absolute values***🡪 devuelve los valores absolutos de un fragmento de la señal
5. ***change sign***🡪 cambia el signo de un fragmento de la señal
6. ***insert silence*** 🡪 inserta silencio en la señal. Esta no es una operación puramente de signal processing pero se incluyo aquí.

***Invariantes, Reglas y Convenciones***

La señal se modifica correctamente después de aplicar cada método.

***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

# Módulo Cursors

## **Descripción General**

Propone una representación de los cursores lógicos sobre una señal. Cursores de intervalo, puntuales etc. Se pretendía que fuera un complemento a la detección de elementos y parámetros. Este módulo está obsoleto y será eliminado en futuras versiones.