# Descripción General

Este módulo contiene toda la interfaz visual que se utiliza en el sistema duetto Sound Lab. Su programación se basa principalmente en la utilización de las clases del framework PyQt para la manipulación de los eventos, ventanas, diálogos y funcionamiento general. La librería pyqtgraph es utilizada para la implementación de varios de los controles encargados de graficar las señales que integran el sistema así como para redefinir algunas clases de PyQt.

Módulo Graphic\_Interface

Documentación para el programador

# Sub-módulos

El módulo Graphic Interface publica los siguientes módulos:

* Dialogs🡪 Las ventanas de diálogo que se muestran en las distintas funcionalidades del sistema.
* UI Files 🡪Los ficheros .ui que se generan por el framework PyQt para la generación de ventanas.
* Widgets🡪 Los distintos controles visuales que se implementan o redefinen en la aplicación.
* Windows🡪 Las distintas ventanas que componen el sistema.

A continuación se profundizará en cada módulo.

# Módulo Dialogs

## **Descripción General**

Este módulo define todas las ventanas de diálogos que se utilizan en el sistema para el ingreso de parámetros a las funcionalidades.

## **Clases**

1. Change Volume🡪 parámetros para la modificación de volumen o amplitud.
2. Filter🡪 parámetros para el filtrado de la señal en el dominio de la frecuencia.
3. Insert Silence🡪 parámetros para insertar silencio.
4. New File🡪parámetros para la creación de una nueva señal.
5. ElemDetectSettingsDialog🡪parámetros para la detección de segmentos y medición.
6. EditcategoriesDialog 🡪edición de las opciones de clasificación. Permite editar las categorías y los valores de cada categoría de clasificación.

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

A excepción de ElemDetectSettingsDialog el resto de los diálogos solo son utilizados como simple vía de ingreso de parámetros y no poseen ninguna dificultad en su implementación.

## **Clase ElemDetectSettingsDialog**

Descripción

Este diálogo permite la definición de los parámetros que se utilizarán para la segmentación de la señal. Permite seleccionar los métodos de segmentación, los parámetros que serán medidos y los lugares en los que se realizaran las mediciones (para los parámetros que lo requieran). Consta de un control visual en el que se muestra la ejecución del algoritmo sobre una señal de menor tamaño y representativa de la que se está analizando.

***Variables***

1. Widget (QSignalDetectorWidget) 🡪 control en el que se muestra la ejecución instantánea del algoritmo seleccionado, a manera de previsualización, sobre la señal de menro tamaño representativa de la señal que se analiza.
2. ParamTree 🡪 Parameter tree con las opciones.
3. detectionSettings (DetectionSettings) 🡪 Objeto que contiene los tipos de segmentación (enum para distintos algoritmos) y las opciones de selección de umbral.

***Métodos***

1. ***detect 🡪*** método que ejecuta la detección sobre la señal de menor tamaño. Se ejecuta cuando ocurre algún cambio en los parámetros de detección
2. ***load\_Theme 🡪*** metodoencargado de cargar las características visuales que se han definido para el uso de la aplicación. Las mismas son las definidas en la ventana principal. Se implementa igual que en la ventana de segmentación.
3. ***changeDetectionMethod 🡪*** método que actualiza los parámetros de detección con los valores del parameter tree seleccionados por el usuario.
4. ***updateGraphsVisibility🡪*** actualiza la visualización en el control según el tipo de detección (en oscilograma o oscilograma y espectrograma).
5. ***toDB,updateThreshold,*** ***updateThresholdLine 🡪*** métodos que controlan el funcionamiento de la barra de umbral en el control de visualización.

***Invariantes, Reglas y Convenciones***

1. el método detect se invoca con cada cambio de las opciones de detección.
2. Las invariantes del control de visualización.

# Módulo UI Files

## **Descripción General**

En este módulo se almacenan los recursos (íconos, fotos) de la aplicación y los ficheros .ui de las ventanas del sistema. No se relaciona directamente con el resto de la aplicación y puede ser excluido del producto final. Solo se incluye como parte de este módulo de interfaz visual con finesorganizativos.

# Módulo Widgets

## **Descripción General**

En este módulo se ubican todos los controles visuales desarrollados como parte de la implementación del sistema. Se asume la programación de los controles como modular e independiente aunque existen dependencias entre ellos para su uso en el software. En principio cada control debe ser posible reutilizarlo y extenderlo para dotarlo de nuevas funcionalidades.

## **Clases**

1. HorizontalHistogramWidget🡪 extiende el control histogram de pyqtgraph para su visualización de forma horizontal.
2. HorizontalHistogramItem🡪item relacionado con el control anterior.
3. SpectrogramPlotWidget🡪 control que grafica el espectrograma. Se basa en reutilización de clases de pyqtgraph
4. SpecYAxis🡪 clase que se integra con el control anterior. Extiende la funcionalidad de los axis de pyqtgraph.
5. OscilogramPlotWidget🡪 control que grafica el oscilograma. Extiende la funcionalidad del control PlotWidget de pyqtgraph.
6. QSignalVisualizerWidget🡪 es el control más importante del sistema ya que integra los demás controles y brinda la mayoría de las funcionalidades. Este control permite la manipulación de una señal de audio. Utiliza dos controles internos OscilogramPlotWidget y SpectrogramPlotWidgetpara graficar oscilograma y espectrograma respectivamente. Es el encargado de sincronizar y coordinar los dos controles en el procesamiento de la señal. En un inicio la manipulación de los distintos gráficos se hizo en un solo control (cuando se utilizaba matplotlib) pero al trasladar la aplicación a pyqtgraph se hizo imprescindible separarlos.Gran parte del procesamiento realizado por la aplicación resulta en wrappers a métodos en este control. Debe ser posible utilizarlo independientemente y ser exportado para su uso por terceros. Se planea incluirlo en el API visual de duetto como la vía para la manipulación de señales. Debe poseer una amplia documentación.
7. QSignalDetectorWidget 🡪control que extiende las funcionalidades del QSignalVisualizerWidget parala segmentación y clasificación de elementos.
8. OscXAxis🡪clase que se integra con los controles anteriores. Axis del eje X
9. OscYAxis🡪 clase que se integra con los controles anteriores. Axis del eje Y
10. Tools🡪 enum que define las herramientas que los usuarios podrán utilizar en el sistema. Se utiliza en el control anterior y en los controles OscilogramPlotWidget y SpectrogramPlotWidget.
11. RectROI🡪 ?
12. UndoRedoManager,UndoRedoActions 🡪 clases para proveer al control principalde la funcionalidad de undo y redo para las acciones de edición sobre la señal.
13. EditCategoriesWidget🡪 Widget que permite visualizar una categoría de clasificación con sus valores. Es usado manualmente al invocar el diálogo de editar las categorías de clasificación. Se adiciona manualmente tantos widgets al diálogo como categorías existan. Se utiliza relacionado con una instancia de ClasificationData en SegmentationClasifficationWindow y en TwoDimensional..

## **Clase HorizontalHistogramItem**

Descripción

Esta clase es el elemento que se añade al HorizontalHistogramWidget. Hereda de HistogramLUTItem y es un widget que contiene el histograma y la barra de colores. Se sobrescriben los métodos init, paint y setHistogramRange.

***Variables***

***Métodos***

1. Paint 🡪 Se encarga de la correcta visualización del widget.
2. setHistogramRange🡪 Fija el rango en el eje X del histograma para evitar que este se ajuste automáticamente.

***Invariantes, Reglas y Convenciones***

## **Clase SpectrogramPlotWidget**

Descripción

Esta clase se encarga de visualizar el espectrograma y manejar las señales del mouse, así como el trabajo con las diferentes herramientas que pueden ser usadas en este. Hereda de GraphicsView. Usa las clases SpecYAxis, RectROI. Es usada por la clase QSignalVisualizerWidget.

***Variables***

1. viewBox🡪 contiene la imagen del espectrograma y la región de zoom. Permite cambiar la región del espectrograma que se está visualizando.
2. imageItem🡪 imagen del espectrograma.
3. xAxis🡪 eje X del espectrograma.
4. yAxis🡪 eje Y del espectrograma.
5. zoomRegion🡪 se usa para guardar unaregión del eje X del espectrograma, que es utilizada por las herramientas que lo necesiten.
6. pointerCursor🡪 cursor en forma de cruz.
7. rectangularCursor🡪 cursor en forma de rectángulo.
8. selectedTool🡪 indica la herramienta seleccionada.
9. rectRegion🡪 se usa para guardar la posición un rectángulo, que es usado por las herramientas que lo necesiten.
10. PIXELS\_OF\_CURSORS\_CHANGES🡪 constante que se usa para saber la cantidad mínima de pixeles que debe abarcar una región para no ser considerado una línea.

***Métodos***

1. load\_Theme🡪 obtiene del tema actual las propiedades correspondientes al espectrograma y efectúa los cambios en la visualización de este.
2. clearRectangularCursor🡪 elimina el cursor rectangular.
3. clearPointerCursor🡪 elimina el cursor con forma de cruz.
4. changeSelectedTool🡪 cambia la herramienta seleccionada actualmente.
5. showGrid🡪 según los parámetros que recibe muestra o no las líneas de los ejes X o Y de forma independiente.
6. on\_zoomRegionChanged 🡪 es llamado cada vez que la región guardada en self.zoomRegion cambia. Emite una señal para que el oscilograma se actualice en correspondencia.
7. mouseMoveEvent,mousePressEvent,mouseDoubleClickEvent, mouseReleaseEvent 🡪 estos métodos son llamados cuando los respectivos evento del mouse ocurren, se encargan de realizar la acción adecuada, dependiendo de la herramienta que esta seleccionada.
8. mouseInsideRectArea🡪 dice si las coordenadas que recibe como parámetro están dentro del rectángulo definido por self.rectRegion.
9. mouseInsideZoomArea🡪 dice si la coordenadaque recibe como parámetro está dentro de la región definida por self.zoomRegion.
10. fromClientToCanvas🡪 transforma una posición del eje X en el array de la señal en la coordenada X del pixel correspondiente en el widget.
11. fromCanvasToClient🡪 transforma la coordenada X de un pixel en el widget en la posición del eje X correspondiente en el array de la señal.
12. fromCanvasToClientY🡪 similar al anterior pero en el eje Y.
13. getFreqTimeAndIntensity🡪 devuelve el tiempo, la frecuencia y la intensidad que corresponden al pixel pasado como parámetro.
14. timeToStr🡪 recibe un tiempo en segundos y devuelve un string con ese tiempo en una unidad de medida más adecuada.

***Invariantes, Reglas y Convenciones***

Siempre que se quiera convertir entre coordenadas del array de la señal y pixeles de la imagen deben de usarse los métodos fromClientToCanvas, fromCanvasToClient y fromCanvasToClientY. ***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

## **Clase SpecYAxis**

Descripción

Esta clase se encarga de la visualización del eje “y” del espectrograma. Hereda de la clase AxisItem de pyqtgraph. En este eje se muestran las frecuencias del espectrograma en KHz. Usa la clase SpecgramSettings. Es usada por la clase SpectrogramPlotWidget.

***Variables***

1. parent (SpectrogramPlotWidget) 🡪widget que muestra este eje.

***Métodos***

1. tickStrings 🡪este método es de la clase base y se usa para saber que texto se le asigna a cada valor del eje.

***Invariantes, Reglas y Convenciones***

La variable parent debe estar asignada a un objeto con un método parent(), que devuelva un objeto con una variable specgramSettings de la clase SpecgramSettings para acceder a su campo freqs.

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

## **Clase OscilogramPlotWidget**

Descripción

Esta clase se encarga de visualizar el oscilograma y manejar las señales del mouse, así como el trabajo con las diferentes herramientas que pueden ser usadas en este. Hereda de PlotWidget. Usa las clases OscXAxis,OscYAxis, RectROI. Es usada por la clase QSignalVisualizerWidget.

***Variables***

1. parentQsignal🡪 Qsignalvisualizer que contiene a esta instancia.
2. axisXOsc🡪 eje X del oscilograma.
3. axisYOsc🡪 eje Y del oscilograma.
4. zoomRegion🡪 se usa para guardar unaregión del eje X del oscilograma, que es utilizada por las herramientas que lo necesiten.
5. pointerCursor🡪 cursor en forma de cruz.
6. rectangularCursor🡪 cursor en forma de rectángulo.
7. selectedTool🡪 indica la herramienta seleccionada.
8. rectRegion🡪 se usa para guardar la posición un rectángulo, que es usado por las herramientas que lo necesiten.
9. highligthedElement🡪es una seccion del oscilograma que se muestra resaltada con otro color. Se utiliza para resaltar la interaccion con el usuario en el QSignalDetector.
10. threshold🡪Permite establecer una línea de umbral horizontal que será mostrada visible o invisible a través del método setVisibleThreshold. Se utiliza en las funcionalidades del QSignalDetector.
11. decimalPlaces🡪constante que se usaal redondear.
12. PIXELS\_OF\_CURSORS\_CHANGES🡪 constante utilizada en la implementación de la visualización del cursor de zoom.representa la longitud mínima de pixeles que se considera para realizar una actualización de los límites del cursor y su visualización.

***Métodos***

1. load\_Theme🡪 obtiene del tema actual las propiedades correspondientes al oscilograma y efectúa los cambios en la visualización de este.
2. select\_region🡪 marca una región en el oscilograma.
3. setVisibleThreshold 🡪muestra u oculta a self.threshold
4. clearRectangularCursor🡪 elimina el cursor rectangular.
5. clearPointerCursor🡪 elimina el cursor con forma de cruz.
6. changeSelectedTool🡪 cambia la herramienta seleccionada actualmente.
7. setZoomRegionVisible🡪muestra u oculta la región guardada en self.zoomRegion.
8. on\_zoomRegionChanged🡪 es llamado cada vez que la región guardada en self.zoomRegion cambia.
9. mouseMoveEvent,mousePressEvent,mouseDoubleClickEvent, mouseReleaseEvent 🡪 estos métodos son llamados cuando los respectivos evento del mouse ocurren, se encargan de realizar la acción adecuada, dependiendo de la herramienta que esta seleccionada.
10. mouseInsideRectArea🡪 dice si las coordenadas que recibe como parámetro están dentro del rectángulo definido por self.rectRegion.
11. mouseInsideZoomArea🡪 dice si la coordenadaque recibe como parámetro está dentro de la región definida por self.zoomRegion.
12. fromClientToCanvas🡪 transforma una posición del eje X en el array de la señal en la coordenada X del pixel correspondiente en el widget.
13. fromCanvasToClient🡪 transforma la coordenada X de un pixel en el widget en la posición del eje X correspondiente en el array de la señal.
14. fromCanvasToClientY🡪 similar al anterior pero en el eje Y.
15. getAmplitudeTimeInfo🡪 devuelve el tiempo y la amplitud que corresponden al pixel pasado como parámetro.

***Invariantes, Reglas y Convenciones***

Siempre que se quiera convertir entre coordenadas del array de la señal y pixeles de la imagen deben de usarse los métodos fromClientToCanvas, fromCanvasToClient y fromCanvasToClientY.

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

## **Clase PowSpecPlotWidget**

Widget que implementa los procesamientos de una dimensión como son diferentes tipos de espectro de potencia.

***Variables***

1. GridX: variable booleana que indica si se pintan o no las celdas verticales del grafico.
2. GridY: variable booleana que indica si se pintan o no las celdas horizontales del grafico.
3. plotColor: color con que se pinta en el grafico.
4. backColor: color del fondo del grafico.
5. Freqs: lista que guarda la información de las frecuencias de acuerdo al procesamiento que se realice.
6. Pxx: variable que se guarda la matriz de información al computar el espectro de potencia de la señal.
7. Bins:variable que se guarda la lista de información temporal al computar el espectro de potencia de la señal.
8. Proc: Lista de instancias de OneDimensionalFunction que serian los procesamientos con que cuenta el widget.
9. Lastproc: esta variable hace referencia a la función que se encarga de calcular y pintar el ultimo procesamiento que el usuario ha realizado para que al cambiar el rango seleccionado por el cursor de zoom en la ventana principal , este método pueda ser llamado con el nuevo intervalo con el fin de actualizar el widget.
10. getInfo: Hace referencia a una función que devuelve una lista de valores con la información de un punto del grafico según el procesamiento que se haya ejecutado.
11. getStrPoint: Hace referencia a una función que devuelve una cadena con la información de un punto del grafico acuerdo al procesamiento ejecutado.
12. getStr:Hace referencia a una función que devuelve una cadena con la información de dos puntos del grafico de acuerdo al procesamiento ejecutado.
13. rangeX: tupla que guarda los índices de la primera y última muestra del intervalo seleccionado de la señal.
14. Data: guarda la información de la señal.
15. NFFTSpec: guarda el valor del tamaño de ventana del espectrograma en la ventana principal de la aplicación.
16. Bitdepth: guarda la profundidad de bit de la señal.
17. Fs: guarda la frecuencia de muestreo de la señal.
18. updateInterval:

***Métodos***

1. \_\_init\_\_(self, parent=None,\*\*kargs): En este método se inicializan las variables de la clase.
2. updateViewBox(self): Este método cambia el tamaño del viewBox del control (ojo se hace para obligar al control a repintar)
3. refresh(self): Este método llama al método al que esta haciendo referencia la variable “lastProc” para que calcule y pinte.
4. updateLast(self,range): actualiza el rango seleccionado en la variable “rangeX” y llama a a la función referenciada por la variable “lastProc”.
5. setData(self,data,range, bitdepth,rate, NFFTSpec, updateInterval): Este método actualiza con su entrada las variables data, rangeX, NFFTSpec, bitdepth, Fs y updateInterval. Este método se usa para inicializar estos valores en la clase PowerSpectrumWindow.
6. connectSignals(self,pTree): Este método recibe como entrada la referencia al ParameterTree de la ventana al cual estarán asociados los procesamientos del widget. Se recorre la lista de procesamientos “proc” para que cada instancia de los mismos llame a su método “connectMySignal” usando la referencia del parameter tree al cual se deben asociar.
7. getParamsList(self): Este método devuelve una lista de las opciones de cada procesamiento del widget.
8. loadTheme(self,Fs, window, plotColor,backColor, lines, maxY, minY, gridX, gridY): Este método actualiza las propiedades visuales del widget para pintar acorde al tema seleccionado anteriormente por el usuario en la ventana principal de la aplicación.

***Invariantes, Reglas y Convenciones***

1. Si se desea añadir un procesamiento al widget, se debe implementar una clase que herede de la clase OneDimensionalFunction y añadir una instancia de la misma a la lista que referencia la variable proc, inicializada en el método \_\_init\_\_().

## **Clase OneDimensionalFunction**

Esta clase presenta las funciones a implementar al añadir un nuevo procesamiento a la clase PowSpecPlotWidget.

***Variables***

1. myOptions: Diccionario en el que de acuerdo al procesamiento a implementar se definen las opciones que el mismo ofrece al usuario para mostrarlas en el ParameterTree. Este se define de acuerdo al que se usa en la librería pyqtgraph.
2. Widget: Esta variable guarda la referencia al widget para usarla para calcular y pintar.
3. Windows: Lista con las referencias a los tipos de ventana que brinda matplotlib.
4. pTree: Esta variable guarda una referencia al ParameterTree al que se mantienen conectadas sus opciones.

***Métodos***

1. \_\_init\_\_(self,widget): Este método inicializa las variables myOptions, widget y Windows.
2. connectMySignal(self,pTree): Este método recibe como parámetro la referencia al ParameterTree y esta la guarda en la variable pTree.
3. processing(self): En este método se realizan los cálculos del procesamiento y se pintan en el widget los resultados.
4. getInfo(self,x): Este método recibe una posición en el eje x del puntero según el procesamiento y devuelve una lista con la información referente a ese punto de acuerdo al procesamiento. Se utiliza para brindar la información al usuario en la barra de status
5. getStr(self,info0, info1): Este método recibe dos listas de información de dos puntos y devuelve como resultado una cadena con la información de los mismos y su relación.
6. getStrPoint(self, info): Este método recibe una lista de información de un puntos y devuelve como resultado una cadena con la información del mismo.

***Invariantes, Reglas y Convenciones***

1. Al heredar de esta clase en la función connectMySignal() se debe enlazar, de acuerdo a las opciones establecidas en las opciones del procesamiento, la función processing al ParameterTree.

## **Clase LogarithmicPowSpec**

Esta clase hereda de OneDimensionalFunction. La misma realiza el procesamiento de espectro de potencia logarítmico a la señal.

***Métodos***

1. processing(self): Este método aplica la transformada rápida de Fourier a todoel intervalo seleccionado de la señal, aplicando las configuraciones impuestas por el usuario en el ParameterTree ,y luego pinta en el widget el resultado llevado a escala de decibeles usando el tamaño de ventana del espectrograma de la ventana principal .Si el intervalo es más pequeño que el tamaño de ventana del espectrograma , se le añaden al mismo la cantidad de muestras para completarlo.

***Invariantes, Reglas y Convenciones***

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

## **Clase AveragePowSpec**

Esta clase hereda de OneDimensionalFunction. La misma realiza el procesamiento de espectro de potencia con promedio a la señal.

***Métodos***

1. processing(self): Este método realiza el cálculo del espectro de potencia promediado del intervalo seleccionado de la señal , haciendo uso de la función “psd” de la biblioteca matplotlib. Se utiliza el tamaño de ventana, solapamiento y ventana establecidas por el usuario en el ParameterTree. Luego pinta el resultado en el widget en escala de decibeles.

***Invariantes, Reglas y Convenciones***

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

## **Clase InstantaneousFrequencies**

Esta clase hereda de OneDimensionalFunction. La misma realiza el procesamiento de las frecuencias picos instantáneas de la señal.

***Variables***

***Métodos***

1. processing(self): Este método computa el espectrograma del intervalo seleccionado por el usuario. Usamos la función “mlab.specgram” de la biblioteca matplotlib. Posteriormente se calculan los máximos valores de frecuencia para cada de tiempo y se grafican los puntos resultantes en el widget.

***Invariantes, Reglas y Convenciones***

## **Clase QSignalVisualizerWidget**

Descripción

***Variables***

1. playing: pyqtSignal(int) 🡪signal que se emite cuando se está reproduciendo . Su parámetro indica la posición actual. Se usa para mostrar adecuadamente el cursor de reproducción. Quien la emite es en realidad el AudioSignal en el \_playCallback.
2. rangeChanged: pyqtSignal(int,int,int) 🡪signal que se emite cuando se cambia el rango de visualización, o sea cuando se hace zoom de cualquier tipo y cuando se mueve el rango visualizado (por ejemplo con el scroll bar). Sus parámetros son extremo izquierdo visualizado, extremo derecho visualizado y tamaño total de la señal, en ese orden (todos en coordenadas del oscilograma).
3. \_doRefresh: pyqtSignal(bool,bool,bool,bool) 🡪signal usada para desencadenar el refresh gráfico. El refresh fue dividido en dos métodos: uno donde se realizan varios cálculos, como computar el espectrograma (que puede ser llamado desde cualquier hilo) y otro donde se grafica (que necesita ejecutarse en el hilo principal de la aplicación). Esta señal permite al primero desencadenar la ejecución del segundo garantizando que esta se realice en el hilo principal.
4. rangeAmplitudeChanged:pyqtSignal(int, int)🡪signal emitida cuando se cambia el rango de amplitud visualizado en el oscilograma a través del cursor rectangular. Nota: si se cambia desde el panel de opciones no se emite esta señal. Los dos parámetros son las amplitudes mínima y máxima respectivamente, en porciento (de 0 a 100 para valores positivos de amplitud y de 0 a -100 para valores negativos)
5. rangeFrequencyChanged: pyqtSignal(int, int) 🡪signal emitida cuando se cambia el rango de frecuencias visualizado en el espectrograma a través del cursor rectangular. Nota: si se cambia desde el panel de opciones no se emite esta señal. Los dos parámetros son los índices de lasfrecuencias mínima y máxima respectivamente. Posible unificación de las dos señales.
6. histogram: HistogramLUTWidget🡪el widget del histograma del espectrograma
7. \_Z: ndarray 🡪matriz con la información del espectrograma. Se calcula en el método refresh y se grafica en el método \_refresh
8. osc\_gridx, osc\_gridy, spec\_gridx, spec\_gridy: bool 🡪indican si se deben pintar las líneas de grid correspondientes
9. osc\_color: QColor 🡪el color de la línea del oscilograma
10. axisXOsc: OscXAxis, axisYOsc: OscYAxis 🡪los ejes del oscilograma
11. axesOscilogram: DuettoPlotWidget 🡪widget donde se plotea el oscilograma. La clase DuettoPlotWidget hereda de la clase PlotWidget de pyqtgraph, por lo que se puede hacer todo lo que se hacía con esta
12. axesSpecgram: DuettoImageWidget 🡪widget donde se plotea el espectrograma. La clase DuettoImageWidget está hecha para imitar a la clase ImageWidget de pyqtgraph mientras brinda más libertad, pero no hereda de esta, por lo que para acceder a algunas opciones y funcionalidades es necesario acceder a algunos controles dentro de la clase
13. osc\_background, spec\_barckground: str 🡪colores de fondo del oscilograma y espectrograma, respectivamente
14. undoRedoManager: UndoRedoManager 🡪permite hacer undo y redo sobre las acciones de procesamiento y edición.
15. minYOsc, maxYOsc: int 🡪rango de valores de amplitud mostrados en el oscilograma en porciento (de 0 a 100 para valores positivos de amplitud y de 0 a -100 para valores negativos)
16. minYSpc, maxYSpc: int 🡪rango de valores de frecuencia mostrados en el espectrograma en porciento
17. lines: bool 🡪 True si en el oscilograma se deben unir los puntos por líneas, False si se desean mostrar los puntos por separado
18. envelopeCurve: PlotCurveItem 🡪la envolvente de la señal mostrada en el oscilograma
19. envelopeFactor: int o double 🡪factor por el cual multiplicar el array de envelope para mejor visualización.
20. lastRefreshPoint: bool 🡪permite saber si el último refresh se hizo mostrando puntos sin unir. Se usa para cuando se llama a refresh con partial=False, saber si es necesario llamar a plot pasándole toda la señal.
21. mainCursor: IntervalCursor 🡪 cursor que representa el intervalo visualizado. Nota: no es un cursor visual, sino una representación de un intervalo

***Propiedades***

1. visibleOscilogram, visibleSpectrogram 🡪permiten conocer y cambiar la visibilidad del oscilograma y espectrograma respectivamente

***Métodos***

1. setSelectedTool(*tool*: int) 🡪cambia la herramienta actual. Posibles valores de tool son: 0 (cursor de zoom, el por defecto), 1 (cursor rectangular), 2 (borrador circular), 3 (borrador rectangular), 4 (cursor de puntero) y 5 (umbral del oscilograma). Se recomienda usar la clase Tool que simula un enum
2. load\_Theme(*theme*: SerializedData) 🡪carga un tema, aplicando todas las configuraciones. Básicamente llama a los métodos load\_Theme del axesOscilogram y axesSpecgram y luego llama a refresh
3. signalName(): str 🡪devuelve el nombre de la señal abierta (en el caso de una señal de un fichero es el nombre del fichero)
4. createContextCursor(*actions*: list) 🡪configura el widget para usar su lista de QAction como menú contextual (el del click derecho) y añade a dicha lista las QAction contenidas en actions(Posible bug: este método no quita las QAction que podría haber, lo que podría ocasionar problemas si es llamado más de una vez)
5. applyFilterSpec(*indexF*: int, *indexT*: int, *FreqLow*: int, *FreqUp*: int) 🡪aplica un filtro que elimina una banda de frecuencias en un espacio de tiempo. indexF e indexT son índices en la matriz del espectrograma. Los parámetros son lo que su nombre indica
6. keyPressEvent(*QKeyEvent*: QKeyEvent) 🡪es llamado cuando el usuario presiona una tecla, y realiza la acción adecuada en dependencia de la tecla presionada (manipular el cursor de zoom con las teclas de izquierda, derecha, ctrl y shift; y reproducir y pausar con la tecla espacio)
7. changePlayStatus() 🡪reproduce o pausa según corresponda
8. play() 🡪reproduce la señal de audio y llama al método createPlayerLine para mostrar una línea con el progreso de la reproducción
9. switchPlayStatus() 🡪similar a changePlayStatus pero no considera que la señal puede estar en stop y no solo en play y pause(No veo la necesidad de este método, debería eliminarse)
10. stop() 🡪detiene la reproducción o grabación y elimina la línea de progreso. En el caso de que se estuviera grabando, regresa al estado previo a empezar la grabación (en términos de visibilidad de oscilograma y espectrograma y posibilidad de interacción) y llama a refresh
11. record() 🡪inicia la grabación de audio, muestra solo el oscilograma, deshabilita la interacción del usuario e inicia el timer que irá grabando
12. createPlayerLine(*value*: int) 🡪muestra una línea para señalar el progreso de la reproducción en la posición indicada por value (en coordenadas del oscilograma) e inicia un timer que irá actualizando dicha línea en función de la frecuencia de muestreo, la velocidad de reproducción y el tiempo de actualización (que por ahora es un valor fijo de 41 milisegundos)
13. removePlayerLine() 🡪quita la línea que muestra el progreso de la reproducción (no es eliminada por completo, simplemente quitada de los elementos mostrados en el oscilograma y espectrograma) y detiene el timer que la actualiza
14. pause() 🡪pausa la reproducción y detiene el timer que mueve la línea de progreso pero no la quita. Si se presiona play la reproducción continúa donde se quedó
15. notifyPlayingCursor(*frame*: int) 🡪actualiza la línea de progreso de la reproducción a la posición de frame (en coordenadas del oscilograma). Si la señal está en stop, quita la línea. Este es el método llamado por el timer en cada actualización, pasándole un valor incremental de frame
16. updatezoomcursor() 🡪actualiza la variable de instancia zoomCursor (representación lógica del cursor de zoom) a partir de la región seleccionada gráficamente
17. updateSpecZoomRegion(*a*: int, *b*: int) 🡪actualiza la posición del cursor de zoom del espectrograma a la región de a a b (en coordenadas del oscilograma) sin ocasionar que el espectrograma emita la señal correspondiente. Este es el método que se llama cuando el oscilograma emite la señal de que se cambió su cursor de zoom y permite actualizar el espectrograma sin que se genere una recursividad infinita
18. updateOscZoomRegion(*a*: int, *b*: int) 🡪análogo al método anterior. Permite actualizar la posición del cursor de zoom del oscilograma. a y b están en coordenadas del espectrograma
19. dropEvent(event: QDropEvent) 🡪está pensado para abrir un fichero que sea arrastrado hasta el widget, sobrescribiendo el evento drop. Lo que hace es copiar el fichero a la carpeta de la aplicación y abrir esta copia. Nunca llega a ejecutarse porque la ventana captura el evento de forma satisfactoria (No se está usando y su funcionamiento es dudoso, debería eliminarse)
20. deselectZoomRegion() 🡪pone el cursor de zoom en la región (0,0) y llama a refresh
21. \_oscRangeChanged(*window*: ?,*viewRange*: list) 🡪cambia el rango visualizado a partir de lo especificado en viewRange. Este parámetro debe ser una lista de dos elementos que son el extremo izquierdo y derecho del rango. Este método se hizo para ser llamado cuando se emita la señal de que el rango del oscilograma fue cambiado manualmente por el usuario. Como esto ya no ocurre (toda la interacción del usuario está controlada por nosotros), este método probablemente no se esté usando y debería eliminarse
22. \_specRangeChanged(b: ?) 🡪análogo al anterior pero toma la información del rango del espectrograma y no de un parámetro. Al igual que el anteriorprobablemente no se esté usando y debería eliminarse
23. \_from\_spec\_to\_osc(*coord*: int): int 🡪convierte coordenadas del espectrograma en su equivalente del oscilograma. Es el método que debería usarse siempre para realizar esta conversión
24. \_from\_osc\_to\_spec(*coord*: int): int 🡪convierte coordenadas del oscilograma en su equivalente del espectrograma. Es el método que debería usarse siempre para realizar esta conversión
25. zoomOut() 🡪hace zoom out manteniendo el centro y sin pasarse de los límites de la señal y llama a refresh (con dataChanged=False)
26. undo() 🡪deshaceuna acción, llamando al método undo del undoRedoManager
27. redo() 🡪rehace una acción, llamando al método redo del undoRedoManager
28. makeZoomRect(*specCoords*: bool) 🡪hace zoom al rectángulo seleccionado en el espectrograma o en el oscilograma en dependencia de si el valor de specCoords es True o False respectivamente. Primeramente se aplica el zoom temporal en ambos y luego el espectral o de amplitud en donde corresponda, llamando a refresh. Finalmente se emite una señal informando del cambio en el rango de frecuencias o de amplitudes
29. zoomIn() 🡪si está seleccionada la herramienta cursor rectangular, hace zoom al rectángulo seleccionado (temporal y espectral o de amplitud). En caso contrario hace zoom temporal manteniendo el centro
30. zoomNone() 🡪muestra la señal completa (solo hace zoom out temporal, no espectral ni de amplitud)
31. makeZoom(*\_min*: int, *\_max*: int, *specCoords*: bool) 🡪hace zoom temporal al rango especificado. specCoords determina si \_min y \_max están dados en coordenadas del espectrograma o del oscilograma (Las últimas 4 líneas no parecen estar haciendo nada, revisar)
32. changeRange(*left*: int, *right*: int, *emit*: bool, *updateOscillogram*: bool, *updateSpectrogram*: bool) 🡪método básico que es el que realmente realiza el zoom. Actualiza los valores de mainCursor, llama a refresh y emite una señal si no se le indica lo contrario. emit debe ser True si se desea emitir la señal, False en caso contrario. updateOscillogram y updateSpectrogram determinan qué debe actualizar el refresh
33. on\_newDataRecorded() 🡪método llamado por el timer para la grabación. Actualiza la señal con lo nuevo grabado y actualiza el oscilograma mostrando los últimos 3 segundos
34. refreshAxes() 🡪actualiza los ticks del eje x (tiempo) del oscilograma y hace que el espectrograma use los mismos valores
35. zoomY(*ymin*: ?,*ymax*: ?) 🡪?
36. computeSpecgramSettings(*overlap*: int) 🡪realiza el cálculo del espectrograma. Es posible especificarle un valor de overlap a través del parámetro overlap (en cantidad de puntos en la señal, no porciento); si este es None, se utilizará el de specgramSettings. El método toma un fragmento de la señal ligeramente mayor que el visualizdo en el oscilograma, calcula el espectrograma de dicho fragmento y guarda los resultados dentro de specgramSettings
37. refresh(*dataChanged*: bool, *updateOscillogram*: bool, *updateSpectrogram*: bool, *partial*: bool) 🡪actualiza el oscilograma y/o espectrograma. dataChanged permite saber si la señal ha cambiado desde el último refresh y de no ser así ahorrar algunos cálculos. updateOscillogram y updateSpectrogram determinan si es necesario actualizar el oscilograma y el espectrograma respectivamente. partial determina si se ploteará toda la señal en el oscilograma o solo el fragmento visible. La actualización está dividida en dos métodos; este realiza la mayoría de los cálculos y puede ejecutarse en cualquier hilo, mientras que el otro, llamado \_refresh, se encarga de la visualización y debe correr en el hilo principal de la aplicación. Este método, al final, se encarga de garantizar la ejecución del método \_refresh en el hilo principal de la aplicación usando una signal de PyQt
38. \_refresh(*dataChanged*: bool, *updateOscillogram*: bool, *updateSpectrogram*: bool, *partial*: bool)🡪como se había dicho, este método complementa al anterior, ocupándose de la visualización. Los parámetros tienen el mismo significado que en aquel.
39. updateSpectrogramColors() 🡪simula un movimiento en el histograma para actualizar la escala de colores mostrada ene le espectrograma
40. cursor\_zoom\_transform(*cursorIndex*: int): int 🡪 ?
41. clearZoomCursor() 🡪pone el cursor de zoom en el rango (0,0)
42. cut() 🡪corta el fragmento de señal seleccionado con el cursor de zoom (guardándolo en una simulación de clipboard) y llama a refresh
43. copy() 🡪 copia el fragmento de señal seleccionado con el cursor de zoom a una simulación de clipboard
44. paste() 🡪pega el fragmento de señal guardado en una simulación de clipboard en la posición del extremo izquierdo del cursor de zoom, y llama a refresh
45. reverse() 🡪invierte el fragmento de señal seleccionado con el cursor de zoom o la señal entera si no hay nada seleccionado y llama a refresh. Para ello llama al método signalProcessingAction, que ejecuta un grupo de acciones comunes para varios métodos de procesamiento de la señal
46. insertWhiteNoise(*ms*: int) 🡪 inserta ruido blanco de ms milisegundos en la posición del extremo izquierdo del cursor de zoom y actualiza
47. updateZoomAndMainCursorFromSignalChangingSizeProcessingAction(*ms\_added*: int) 🡪realiza un conjunto de acciones comunes para varios métodos en los cuales se cambia el tamaño de la señal. Ajusta el rango visualizado para adecuarse al cambio y llama a refresh
48. insertPinkNoise(*ms*: int, *type*: int, *Fc*: int, *Fl*: int, *Fu*: int) 🡪inserta ruido rosado de ms milisegundos en la posición del extremo izquierdo del cursor de zoom y actualiza
49. resampling(*samplingRate*: int) 🡪 hace resampling a la señal a la frecuencia de muestreo dada en el parámetro samplingRate
50. getIndexFromAndTo() 🡪determina los índices del rango a utilizar en varios métodos de procesamiento de la señal. Toma el rango seleccionado por el cursor de zoom y si no hay nada seleccionado toma la señal entera
51. signalProcessingAction(*delegate*: callable, \**args*) 🡪ejecuta el método de procesamiento de la señal dado por delegate, junto con un grupo de acciones comunes para ellos. Primeramente determina el rango, luego llama a delegate y finalmente llama a refresh
52. *…*aquí me faltan varios métodos de procesamiento de señal de los cuales realmente no hay mucho que decir…
53. openNew(*samplingRate*: int, *bitDepth*: int, *duration*: int, *whiteNoise*: bool)🡪llama al método open con parámetros específicos para crear una nueva señal
54. open(*filename*: str, *samplingRate*: int, *bitDepth*: int, *duration*: int, *whiteNoise*: bool)🡪si se proporciona un nombre de fichero en filename, abrirá dicho fichero y el resto de los parámetros son ignorados; en caso contrario creará una nueva señal con las características especificadas en los parámetros. En cualquier caso inicializará todo lo necesario y llamará a refresh, mostrando toda la señal (como si se le hubiera dado zoomNone)
55. save(*fname*: str) 🡪guarda la señal en un fichero wav
56. saveSelected(*fname*: str) 🡪genera una señal a partir del fragmento seleccionado y la guarda en un fichero wav
57. cursorsData(): bytearray 🡪devuelve la información de los cursores como un bytearray para ser almacenada en el fichero wav(No estoy seguro de esto)
58. loadUserData(*userData*: ?) 🡪carga la información guardada en el fichero wav y la transforma en cursores(No estoy seguro de esto)
59. SaveColorBar() 🡪guarda la barra de colores actual, mostrando un diálogo para que el usuario escoja nombre y ubicación

***Invariantes, Reglas y Convenciones***

Si se desea saber qué posición ocupa un punto del oscilograma en el espectrograma o viceversa siempre deben de usarse el método \_from\_spec\_to\_osc o \_from\_osc\_to\_spec según corresponda.

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

## **Clase OscXAxis**

Descripción

Esta clase se encarga de la visualización del eje “x” del oscilograma. Hereda de la clase AxisItem de pyqtgraph. En este eje se muestra el tiempo en segundos. Usa las clases IntervalCursor y SignalProcessor. Es usada por la clase QSignalVisualizerWidget.

***Variables***

1. parent (QSignalVisualizerWidget) 🡪 widget que muestra este eje.

*Métodos*

1. tickStrings 🡪este método es de la clase base y se usa para saber que texto se le asigna a cada valor del eje.
2. tickValues 🡪estemétodo es de la clase base y se usa para saber qué valores deben mostrarse en el eje.
3. tickSpacing🡪estemétodo es de la clase base y se usa para saber la diferencia entre dos valores consecutivos en el eje.

***Invariantes, Reglas y Convenciones***

La variable parent debe estar asignada a un objeto con una variable signalProcessor de tipo SignalProcessor para saber la frecuencia de muestreo. El tiempo es el número de la muestra entre la frecuencia de muestreo. El método refreshAxes de la clase QSignalVisualizerWidget se encarga de que este eje y el eje “x” del espectrograma sean el mismo. Se muestran 10 valores de tiempo excepto cuando la diferencia entre un valor y el siguiente sea menor de 10-4, mostrándose solo los valores que estén a esta distancia.

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

## **Clase OscYAxis**

Descripción

Esta clase se encarga de la visualización del eje “y” del oscilograma. Hereda de la clase AxisItem de pyqtgraph. En este eje se muestran la amplitud del oscilograma en porciento de la máxima posible. Usa la clase SignalProcessor. Es usada por la clase QSignalVisualizerWidget.

***Variables***

1. parent (QSignalVisualizerWidget) 🡪widget que muestra este eje.

***Métodos***

1. tickStrings 🡪este método es de la clase base y se usa para saber que texto se le asigna a cada valor del eje.

***Invariantes, Reglas y Convenciones***

La variable parent debe estar asignada a un objeto con una variable signalProcessor de tipo SignalProcessor para saber la máxima amplitud posible.

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

## **Clase RectROI**

Descripción

Esta clase se usa como cursor rectangular en el oscilograma y en el espectrograma. Al añadirla a un widget automáticamente se encarga de dibujar un rectángulo en la posición que se desee. Esta clase hereda de la clase ROI de pyqtgraph, sólo el constructor fue sobrescrito. Hereda lo métodos setPos y setSize que permiten ajustar su posición y tamaño. Es usada por la clase SpectrogramPlotWidget.

***Variables***

***Métodos***

***Invariantes, Reglas y Convenciones***

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

## **Clase UndoRedoManager**

**Descripción**

Esta clase manipula las acciones de undo y redo en el sistema. Es una estructura de datos en forma de pila que no elimina los elementos al extraerlos. Dado que las acciones realizadas conviene almacenarlas para poder deshacerlas y este funcionamiento es propio de una pila (las últimas acciones se deshacen primero) también es conveniente no eliminarlas para poder rehacerlas. Entonces es necesaria una estructura en forma de pila que pueda recorrerse en ambos sentidos. Esta clase implementa este manipulador de acciones mediante una lista y un puntero para la posición de la última acción realizada. Para su utilización en el sistema se asume que el control principal posee una instancia de esta clase y que antes de cada acción de edición sobre la señal se le adiciona la correspondiente acción para undo y redo.

***Variables***

1. actionList (list)🡪La lista en la que se almacenan las acciones.
2. actionIndex (int)🡪Elíndice de la lista que apunta a la última acción realizada.
3. actionExec (signal) 🡪La señal que se emite cuando se realiza una acción de undo o redo

***Métodos***

1. undo()🡪Deshace (si existen acciones por deshacer) la última acción de edición realizada sobre la señal. Si se ejecuta la acción se emite la señal actionExec.
2. redo()🡪Rehace (si existen acciones por rehacer) la última acción de edición realizada sobre la señal. Si se ejecuta la acción se emite la señal actionExec.
3. clearActions()🡪Borra las acciones que se han almacenado en la estructura.
4. addAction(UndoRedoAction)🡪Adiciona una nueva acción.
5. count()🡪Devuelve la cantidad de acciones que están almacenadas en la estructura.

***Invariantes, Reglas y Convenciones***

1. count() == 0 si no hay acciones adicionadas o se ha invocado a clearActions
2. count == cantidad de elem ¡= None en la lista actionList
3. actionList tiene tantos elem ¡= None como acciones en la estructura.
4. actionList tiene mayor o igual capacidad que la cantidad de acciones (count())
5. actionIndex apunta a la acción realizada anteriormente.
6. actionIndex == -1 si count == 0
7. si se hace redo y actionIndex< count -1 se incrementa en 1 actionIndex y se emite la señal actionExec.
8. si se hace undo y actionIndex > 0 se disminuye en 1 actionIndex y se emite la señal actionExec.
9. Count = count +1 luego de addAction

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

## **Clase UndoRedoAction**

Descripción

Es una clase auxiliar para utilizarla en conjunto con la anterior.Contiene dos variables callables o dos métodos undo y redo. Las distintas acciones de undo y redo heredan de esta clase y redefinen

***Variables***

1. undo (callable) 🡪método que se invoca cuando se hace undo a esta acción
2. redo(callable) 🡪método que se invoca cuando se hace redo a esta acción

***Métodos***

***Invariantes, Reglas y Convenciones***

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

# Módulo Windows

## **Descripción General**

En este módulo se incluyen todas las ventanas que se utilizan en el sistema. Las ventanas de procesamiento están directamente vinculadas con los controles del módulo anterior y se relacionan con ellos para brindar las funcionalidades.

## **Clases**

1. DuettoSoundLabWindow🡪 ventana principal de la aplicación. Procesamiento y edición de señales.
2. PowerSpectrumWindow🡪 ventana de visualización del power spectrum. Es utilizada en una de las funcionalidades brindadas por la ventana anterior.
3. TwoDimensionalWindow🡪 ventana de visualización de procesamientos bidimensionales.
4. SegmentationAndClasificationWindow🡪 Ventana que permite segmentar y clasificar la señal.
5. SerializedData🡪 estructura para hacer persistente a disco las opciones visuales del sistema para una futura utilización.
6. DuettoListParameterItem🡪 clase para la ordenación de los parámetros de algunas opciones visuales en la ventana DuettoSoundLabWindow

## **Clase DuettoSoundLabWindow**

Descripción

***Señales***

1. dropchanged

***Variables***

1. hist:Instancia de HorizontalHistogramWidget.
2. pow\_overlap:almacena el solapamiento(overlap) que se utilizara para calcular el espectro de potencia.
3. pow\_spec\_backg:almacena el color de fondo (background color) que se utilizara para pintar el espectro de potencia.
4. pow\_spec\_plotColor:almacena el color de la función (plot color) que se utilizara para pintar el espectro de potencia.
5. pow\_spec\_gridx:Variable booleana que especifica si el usuario desea pintar cuadriculas correspondiente solo al eje x en este caso para el espectro de potencia.
6. pow\_spec\_gridy:Variable booleana que especifica si el usuario desea pintar cuadriculas correspondiente solo al eje y en este caso para el espectro de potencia.
7. pow\_spec\_lines:Variable booleana que especifica si el usuario quiere graficar el espectro de potencia con interpolación entre los puntos calculados o no del espectro de potencia.
8. pow\_spec\_maxY:Almacena el valor de máxima intensidad a mostrar en el grafico del espectro de potencia.
9. pow\_spec\_minY:Almacena el valor de mínima intensidad a mostrar en el grafico del espectro de potencia.
10. window\_pow: Especifica el tipo de ventana que se quiere aplicar a la función que se tendrá en cuenta para el cálculo del espectro de potencia.
11. pow\_spec\_windows:Lista que almacena las referencias a las ventanas de espectro de potencia que estén activas durante la ejecución de la aplicación.
12. window\_spec:Especificael tipo de ventana que se quiere aplicar a la función.
13. statusbar:Instancia de la barra de estado de la ventana principal de la aplicación.
14. lastopen:Nombredel último fichero de audio que fue abierto en la aplicación.
15. ParamTree:Instancia del parámetro de tipo grupo a partir del cual se crea la instancia del árbol de configuraciones.
16. parameterTree:instancia del árbol de configuraciones de la ventana principal.

***Métodos***

1. \_\_init\_\_ (self, parent=None):Inicializa las variables y el estado de los controles de la ventana principal y se carga el tema por defecto de la aplicación.
2. changeFrequency (self, min, max):Actualiza en las configuraciones el rango de la frecuencia mínima y máxima que se está mostrando en el espectrograma. Se usa cuando se aplica la herramienta del cursor rectangular.
3. changeAmplitude(self, min, max):Actualiza en las configuraciones el rango de la amplitud mínima y máxima que se está mostrando en el oscilograma. Se usa cuando se aplica la herramienta del cursor rectangular.
4. folderFiles(self,folder):Se utiliza para obtener la lista de todos los ficheros de audio de la carpeta donde está ubicado el abierto en la aplicación.
5. updateStatusBar(self,line):Actualiza el texto de la barra de estado de la ventana principal con el del parámetro line.
6. on\_load(self):Se ejecuta al cargar la ventana principal, carga una señal inicial de la aplicación y carga un dialogo para que el usuario escoja un fichero a abrir si lo desea.
7. change(self, param, changes):Actualiza los cambios de las variables y los controles de la aplicación cuando el usuario hace una modificación en las opciones de configuración.
8. updateRegionTheme(self):Actualiza los valores del umbral (dB) en las opciones de configuracióndel espectrograma cuando el usuario lo modifica en el HistogramWidget.
9. SerializeTheme(self,filename):Guarda la configuración del estado actual del tema que está usando la aplicación.
10. updateMyTheme(self,data):Se usa al cargar un tema de un fichero .dth para actualizar la aplicación a la configuración del mismo.
11. DeSerializeTheme(self,filename): Carga en memoria la configuración de un tema guardado en un fichero de extensión .dth.
12. dragEnterEvent(self, event):Se llama al método “acceptProposedAction” de event y emite la señal “dropchanged ” de la ventana.
13. dragMoveEvent(self, event):Se llama al método “acceptProposedAction” de event.
14. dropEvent(self, event): Este evento captura la dirección del fichero arrastrado a la ventana y abre el mismo de ser posible.
15. dragLeaveEvent(self, event):Llama al método “accept()” de event.
16. on\_actionLoad\_Theme\_triggered (self):Abre un dialogo para que el usuario escoja un fichero de tema de Duetto (.dth) para usarlo en la aplicación.
17. on\_actionChangePlayStatus\_triggered(self): Llama a la función changePlayStatus() del widget.
18. on\_actionSave\_theme\_triggered (self):Abre un dialogo para que el usuario escoja un nombre para el fichero de tema de Duetto (.dth) que va a guardar para usarlo en la aplicación.
19. on\_actionSegmentation\_And\_Clasification\_triggered (self):Crea una nueva instancia de la ventana de segmentación y clasificación.
20. on\_actionZoom\_Cursor\_triggered (self):Actualiza en los controles de graficar la herramienta seleccionada, llama al método setSelectedTool de la variable widget en este caso el cursor de zoom.
21. on\_actionRectangular\_Cursor\_triggered (self):Actualiza en los controles de graficar la herramienta seleccionada, llama al método setSelectedTool de la variable widget, en este caso el cursor rectangular.
22. on\_actionRectangular\_Eraser\_triggered(self):Actualiza en los controles de graficar la herramienta seleccionada, llama al método setSelectedTool de la variable widget, en este caso el borrador rectangular.
23. on\_actionPointer\_Cursor\_triggered (self):Actualiza en los controles de graficar la herramienta seleccionada, llama al método setSelectedTool de la variable widget, en este caso el cursor puntero.
24. on\_actionResampling\_triggered(self):Crea una instancia de InsertSilenceDialog y con la configuración que el usuario escoja llama al método resampling dela variable widget.
25. on\_actionCut\_triggered(self):Llama al método cut () de la variable widget.
26. on\_actionPositive\_Values\_triggered(self):
27. on\_actionChange\_Sign\_triggered(self):
28. on\_actionNegative\_Values\_triggered(self):
29. on\_actionCopy\_triggered(self):Llama al método copy () de la variable widget.
30. on\_actionPaste\_triggered(self):Llama al método paste () de la variable widget.
31. on\_actionUndo\_triggered(self):Llama al método undo () de la variable widget.
32. on\_actionRedo\_triggered(self):Llama al método redo () de la variable widget.
33. on\_actionSmart\_Scale\_triggered(self):Llama a la función scale() del widget con la información obtenida en el dialogo.
34. on\_actionInsert\_Silence\_triggered(self):Inserta un segmento del tamaño especificado en el dialogo en la ubicación del cursor en la señal. Llama a la función insertSilence() del widget.
35. on\_actionGenerate\_Pink\_Noise\_triggered(self):Inserta un segmento de ruido rosado del tamaño especificado en el dialogo en la ubicación del cursor en la señal. Llama a la función insertPinkNoise () del widget.
36. on\_actionGenerate\_White\_Noise\_triggered(self):Inserta un segmento de ruido blanco del tamaño especificado en el dialogo en la ubicación del cursor en la señal. Llama a la función insertWhiteNoise () del widget.
37. filter\_helper(self):Brinda un nuevo dialogo al usuario del cual se obtiene el tipo de filtro que quiere aplicar el usuario. Se usa en la función on\_actionFilter\_triggered(self).
38. on\_actionFilter\_triggered(self):Llama a la función filter del widget con la información obtenida de apicar la función filter\_helper(self) de la ventana.
39. on\_actionSilence\_triggered(self): Llama a la función silence() del widget.
40. on\_actionNormalize\_triggered(self):Llama a la función normalize() del widget.
41. on\_actionFull\_Screen\_triggered(self):Pone la ventana en modo fullscreen ocultando la barra usando la función showFullScreen(), o en modo normal con la función showNormal() ambas de la ventana.
42. on\_action\_Reverse\_triggered(self):Deshace la ultima modificación o procesamiento hecho por el usuario.
43. updatePowSpecWin(self):Por cada ventana de power spectrum que este activa en ese momento, se llama al método updatePowSpectrumInterval () de la ventana.
44. on\_actionZoomIn\_triggered(self):Llama al método zoomIn () de la variable widget.
45. on\_actionZoom\_out\_triggered(self):Llama al método zoomOut () de la variable widget.
46. on\_actionZoom\_out\_entire\_file\_triggered(self):Llama al método zoomNone () de la variable widget.
47. on\_actionSettings\_triggered(self):Muestra u oculta el menú de opciones de configuración de la ventana principal.
48. on\_actionPower\_Spectrum\_triggered(self):Crea una nueva instancia de PowSpecWindow y manda a calcular el espectro de potencia para el intervalo de la señal seleccionado con la configuración establecida en el menú.
49. on\_actionExit\_triggered(self):Llama a close () de la ventana.
50. closeEvent(self,event): Se ejecuta al cerrar la ventana, llama a \_save () y luego cierra la ventana.
51. \_save(self,event = None):Abre un dialogo para preguntar si el usuario desea guardar los cambios de la señal antes de salir de la aplicación.
52. on\_actionNew\_triggered(self):Llama al método openNew () de la variable widget.
53. on\_actionOpen\_triggered(self):Llama al método \_open () de la ventana.
54. on\_actionClose\_triggered(self):Llama al método \_save () y luego a on\_load () de la ventana.
55. \_open(self,f=''):Llama al método open () de la variable widget.
56. on\_actionFile\_Up\_triggered(self):Llama al fichero de audio que le sigue al actual en su carpeta.
57. on\_actionFile\_Down\_triggered(self):Llama al fichero de audio anterior al actual en su carpeta.
58. on\_actionSave\_triggered(self):Llamaal método save () de la variable widget.
59. on\_actionSave\_selected\_interval\_as\_triggered(self):Llama al método saveSelected () de la variable widget.
60. on\_actionPlay\_Sound\_triggered(self):Llama al método play () de la variable widget.
61. on\_actionStop\_Sound\_triggered(self):Llama al método stop () de la variable widget.
62. on\_actionRecord\_triggered(self):Llama al método record () de la variable widget.
63. on\_actionPause\_Sound\_triggered(self):Llama al método pause () de la variable widget.
64. on\_actionCombined\_triggered(self):Llamaal método refresh () de la variable widget con la opción de que muestre el oscilograma y el espectrograma.
65. on\_actionSpectogram\_triggered(self):Llama al método refresh () de la variable widget con la opción de que muestre el espectrograma.
66. on\_actionOscilogram\_triggered(self):Llama al método refresh () de la variable widget con la opción de que muestre el oscilograma.
67. on\_actionOsc\_Image\_triggered(self):Salva un fichero de la imagen del oscilograma.Llama al método saveImage ().
68. on\_actionCombined\_Image\_triggered(self):Salva un fichero de la imagen del oscilograma y el espectrograma. Llama al método saveImage ().
69. on\_actionFull\_Screen\_triggered(self):Llama al método showFullScreen () o showNormal () según el estado en que se encuentre la ventana.
70. on\_actionSpecgram\_Image\_triggered(self):Salva un fichero de la imagen del espectrograma. Llama al método saveImage ().
71. saveImage(self,widget,text=""):Salva la imagen especificada en formato .jpg.
72. on\_widget\_rangeChanged(self, left, right, total):Actualiza el estado de la barra de desplazamiento cuando cambia el rango de visibilidad en el widget.
73. on\_horizontalScrollBar\_valueChanged(self, value):Llama al método changeRange () de la variable widget. Se ejecuta cuando el usuario mueve la barra de desplazamiento de la ventana.
74. on\_g\_triggered(self, action):Llama al métodostop () de la variable widget y cambia el valor de su campo playerSpeed. Se ejecuta cuando el usuario cambia la velocidad de reproducción.

***Invariantes, Reglas y Convenciones***

## **Clase PowerSpectrumWindow**

Descripción

***Variables***

1. parameterTree

***Métodos***

1. \_\_init\_\_(self,parent=None,plotColor="FFF",backColor="FFF",gridx=True,gridy=True,minY=-50,maxY=5,lines=True):Seinicializa el widget, el parmeter tree y las se cargan los aspectos visuales de la ventana.
2. windowSettingsChange(self, param, changes)🡪Este método actualiza las variables del widget cuando el usuario cambia alguna configuración de las opciones de procesamientos del ParameterTree.
3. on\_actionOneDimFunctSettings\_triggered(self)🡪Visualiza u oculta el parameter Tree (opciones de procesamientos).
4. updateStatusBar (self, message)🡪Este método está conectado a la señal del widget PointerChanged que se lanza cuando la posición de la herramienta puntero cuando este se mueve.
5. updatePowSpectrumInterval(self,data):Este métodose llama cuando la ventana está activa y cambia el intervalo seleccionado en la principal y este entonces llama a la función plot () pasándole como parámetro el nuevo intervalo de la señal seleccionado al que se le quisiera hacer la transformación.

***Invariantes, Reglas y Convenciones***

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

## **Clase TwoDimensionalWindow**

**Descripción**

Esta ventana es utilizada en la ventana de segmentación y clasificación. Luego de segmentar y extraer parámetros. La ventana permite representar y graficar bidimensionalmente dos de las magnitudes medidas. Se utiliza un scatter plot para el gráfico de los elementos. La ventana permite analizar los segmentos que se detectaron en la ventana **SegmentationAndClasificationWindow.** Mediante esta ventana se puedenvisualizar los parámetros medidos en cada segmento. Contiene un Parameter Tree en el que se insertan las opciones que brinda la ventana. Allí se puede encontrar las dos listas de parámetros (cada una con las mediciones realizadas) en las cuales se pueden seleccionar cual parámetro se graficará en cada eje (X e Y). Se pueden modificar de la misma manera algunas opciones visuales como la figura de los elementos bidimensionales (cuadrados, rombos etc), el color, la fuente para el texto de los ejes etc.

***Variables***

1. elementSelected (signal) 🡪Esta señal es lanzada cuando se selecciona un elemento bidimensional del gráfico. Es la primera parte de la comunicación entre la ventana de segmentación y esta ventana. Permite avisar cuando se selecciona un elemento desde esta ventana.
2. elementsClasification (signal) 🡪Esta señal es lanzada cuando se han seleccionado múltiples objetos del gráfico y se les han asignado categorías dentro de la clasificación. Es la implementación de la clasificación manual.
3. scatter\_plot(ScatterPLot) 🡪Es el gráfico que se muestra. Está formado por tantos elementos como hayan sido detectados en la ventana de segmentación.
4. classificationData(ClasificationData) 🡪Es la instancia de clasificación. En ella se almacena las categorías, los valores de cada categoría, así como los vectores de entrenamiento que son comunes a los métodos de clasificación. Es utilizada para la segmentación manual en esta ventana.
5. Columns(string[]) 🡪Los nombres de los parámetros medidos.
6. previousSelectedElement(int) 🡪El índice del último elemento seleccionado. Es utilizado para resaltar con otro color el elemento seleccionado así como para reorganizar visualmente la ventana de segmentación con el objetivo de hacerlo visible en la tabla de parámetros y en el control visual de oscilograma y espectrograma. Cuando se selecciona mediante click un elemento este valor es actualizado y es lanzada la señalelementSelected
7. data(array[,]) 🡪Array de (cantidad de segmentos)x(cantidad de parámetros medidos) con la información de los parámetros.

***Métodos***

1. plot🡪refresca el gráfico bidimensional con los valores contenidos en data. Genera un elemento por cada fila en data y lo posiciona en el grafico según las coordenadas seleccionadas por el usuario. Las coordenadas son los valores de los parámetros medidos que han sido seleccionados para cada eje.
2. on\_actionSaveGraphImage\_triggered🡪Permite exportar la imagen del grafico a disco para su futuro uso.
3. on\_actionMark\_Selected\_Elements\_As\_triggered🡪Luego de seleccionar varios elementos visuales en la ventana(mediante dibujo de un rectángulo. Haciendo click para fijar la posición inicial y haciendo drag para fijar las dimensiones alto y ancho) este método procede a abrir el diálogo de clasificación manual para que el usuario fije los valores de cada categoría para todos los elementos que se hayan seleccionado.
4. on\_actionHide\_Show\_Settings\_triggered 🡪Permite ocultar y volver a mostrar el Parameter tree con las opciones. Es conveniente si el usuario quiere observar con mayor comodidad el gráfico.
5. createParameterTreeOptionsMétodo privado que crea el parameter tree con los datos de las mediciones.
6. loadData (columns(string[]), data (array[,])) 🡪Método que actualiza los valores de la ventana porque hubo algún cambio en las mediciones o en los segmentos detectados.
7. load\_Theme🡪Carga con las opciones visuales de la aplicación en esta nueva ventana
8. selectElement (index(int)) 🡪Permite seleccionar el elemento que se encuentra en la posición index. Es la segunda parte de la comunicación entre la ventana de segmentación y esta.
9. deselectElement🡪El complemento del método anterior para revertir la selección

***Invariantes, Reglas y Convenciones***

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

## **Clase SegmentationAndClasificationWindow**

***Descripción***

En esta ventana se realiza la segmentación, clasificación y extracción de parámetros sobre la señal. Brinda las funcionalidades de la ventana principal que no modifican la señal. Contiene un control QSignalDetectorWidget para la visualización de la señal. Este control extiende las funcionalidades del SignalVisualizer para la segmentación y visualización de segmentos. La medición de parámetros puede ser observada mediante una ventana flotante (dock) que muestra una tabla con las mediciones de cada parámetro seleccionado en cada elemento detectado. Interactúa con el dialogo de detección.

***Variables***

1. widget QSignal DetectorWidget 🡪 control que permite la detección y segmentación de la señal. Extiende de QSignalVisualizer.
2. rejectSignal bool 🡪 bolean que establece si la señal con que se pretendía abrir la ventana fue aceptada o no. La ventana se diseñó para permitir el procesamiento de señales menores de 1 min por cuestiones de eficiencia (sobre todo en el cálculo del espectrograma y los algoritmos).
3. parameterTable\_rowcolor\_odd color, parameterTable\_rowcolor\_even color🡪 colores para alternar en la table de mediciones.
4. spectralMeasurementLocation🡪 posición de las mediciones espectrales. Object.
5. detectionSettings dict🡪 parámetros de los algoritmos de segmentación.
6. ParamTree🡪 árbol con las opciones de parámetros que serán ofrecidas en el dialogo de detección. Permite una mayor independencia a la hora de incorporar un nuevo parámetro.
7. twodimensionalGraphs🡪 array con las ventanas de procesamiento 2Dim abiertas.
8. measuredParameters = np.array([[], []])🡪parámetros medidos en la tabla de mediciones. Se utiliza para mantener un control en memoria de las mediciones. La tabla es un espejo visual de este array. Es una lista de listas con la información
9. classificationData🡪Classification data. Contiene los elementos de clasificación que se utilizarán.
10. elementsClasificationTableData🡪almacena la información de clasificación que está presente en la table de mediciones. Se almacena separada para poder responder a eliminaciones de categorías y de valores sin afectar la tabla ni recalcular las mediciones. #tiene la forma de una lista con [["category name","category value"]] para cada elemento. Ejemplo con dos elementos y dos categorías [[["Specie","Cartacuba"],["Location","Cuba"]], [["Specie","Sinsonte"],["Location","Camaguey"]]]

***Métodos***

1. on\_actionTwo\_Dimensional\_Graphs🡪hace aparecer una ventana para el análisis bidimensional de los parámetros medidos. Deben haberse detectado elementos previamente mediante el dialogo de detección y haberse medido parámetros.
2. on\_actionZoom\_Cursor, on\_actionPointer\_Cursor, on\_actionRectangular\_Cursor, on\_actionRectangular\_Eraser🡪Manipulan el cambio de herramientas en el control QsignalDetectorWidget.
3. updateStatusBar🡪actualiza la barra de status bar con información proveniente de los controles administrados por el SignalDetector. La información proviene de las herramientas que se utilizan.
4. on\_widget\_range, on\_horizontalScrollBar\_valueChanged🡪 actualizan el estado de la ventana al ocurrir un cambio en la barra de scroll o en el rango de la señal que se visualiza.
5. on\_actionFull\_Screen\_🡪cambia la visualización de la ventana a full screen si no se encuentra en ese tipo de visualización. De lo contrario la devuelve a su estado normal.
6. load\_Theme🡪 carga los datos de configuración visual del tema de la aplicación hacia la ventana. Delega en el signalDetector.
7. on\_actionOsgram\_Image,on\_actionSpecgram\_Image, on\_actionCombined\_Image 🡪 salvan a disco la imagen visualizada en los controles oscilograma, espectrograma y ambos respectivamente. Se apoyan en el método save image.
8. saveImage(widget,text (string) )🡪 método que guarda en disco la imagen del widget visual que recibe como parámetro. Realiza un screenshot del mismo y lo guarda como jpg. El widget debe ser visible en el momento de la toma del screenshot. El texto text se adiciona al nombre de la foto como nombre por defecto.
9. on\_actionPlay\_Sound, on\_actionStop\_Sound, on\_actionPause\_Sound🡪 acciones que delegan en el SignalDetector las funcionalidades de reproduccion de la señal.
10. on\_actionOscilogram ,on\_actionSpectogram , on\_actionCombined🡪 acciones que muestran solo el gráfico del oscilograma, solo el del espectrograma o ambos respectivamente. Delegan en el SignalDetector.
11. on\_actionZoomIn,on\_actionZoomOut,on\_actionZoom\_out\_entire\_file🡪 acciones que realizan operaciones de zoom en la señal. Zoom in, Zoom out y Zoom None (no hacer zoom y mostrar la señal completa) respectivamente. Delegan en el SignalDetector.
12. on\_actionExit,closeEvent🡪 métodos que controlan la salida de la ventana. Verifican cambios por guardar y liberan recursos.
13. on\_actionView\_Parameters, on\_actionElements\_Peaks, on\_actionTemporal\_Elements, on\_actionTemporal\_Numbers, on\_actionSpectral\_Numbers,on\_actionSpectral\_Figures, on\_actionTemporal\_Figures, on\_actionSpectral\_Elements🡪 acciones que modifican la visibilidad de los componentes visuales en los elementos detectados. Permiten cambiar la visibilidad de : la tabla de parámetros medidos (dock window), la frecuencia pico de cada elemento (espectrograma), todos los componentes de los elementos temporales, los números de los elementos temporales, los números de los elementos espectrales, las figuras de los elementos espectrales, las figuras de los elementos temporales, y todos los componentes de los elementos espectrales respectivamente.
14. actionDelete\_Selected\_Elements🡪 elimina los elementos que se encuentren seleccionados bajo en cursor de zoom. Actualiza el estado de la tabla de mediciones y los controles visuales así como en las ventanas de procesamiento bidimensional que estén abiertas.
15. on\_actionDeselect\_Elements\_triggered🡪 remueve la selección realizada sobre los elementos detectados. Quita el cursor de zoom si está visible y el elemento seleccionado actual que se muestra con un color resaltado. Actualiza el estado de la tabla de mediciones y los controles visuales así como en las ventanas de procesamiento bidimensional que estén abiertas.
16. startBatchProcess🡪 comienza el procesamiento en batch en hilo de ejecución distinto. El procesamiento en batch puede ser de segmentación o de separación del fichero en señales de menor duración. En el primer caso se utilizan las opciones de segmentación establecidas en el diálogo de segmentación y en el segundo caso se establece la duración de los ficheros mediante el numeric up down que se encuentra en el tabulador de batch.
17. Batch🡪 método que realiza el procesamiento en batch
18. on\_actionMeditions\_triggered(name="",table = None)🡪 accion que salva las mediciones realizadas en un fichero Excel (.xls). Utiliza la librería de python xlwt para este propósito. Name es el nombre del fichero en el que se va a guardar la información. Si no se ofrece un valor se provee uno por defecto mediante un fileSaveDialog. Table es un objeto en el que se ha almacenado la información con estructura de tabla que se quiere guardar. Se utiliza tanto para el procesamiento en batch como para el procesamiento manual.
19. getSpectralData🡪 calcula los parámetros espectrales necesarios para procesar una señal. Se utiliza en el procesamiento en batch.
20. selectInputFolder, selectOutputFolder🡪 métodos para seleccionar las carpetas de entrada y salida de los datos procesados en batch respectivamente.
21. Writedata(ws,table)🡪 metodo que escribe en un objeto (ws) de tipo xlwt.Workbook la información de la tabla table. Se utiliza para almacenar la información de las mediciones en un fichero Excel tanto en el procesamiento manual como en batch.
22. getSettings🡪 metodo que obtiene las opciones del dialogo de deteccion. Por esta via se actualiza cuales parametros serán medidos, los lugares en que se realizan las mediciones (para los parametros que lo requieran) y el metodo de segmentación utilizado.
23. on\_actionDetection\_triggered🡪 realiza la segmentación y detección de elementos en la señal. Invoca al diálogo de segmentación para obtener las opciones de segmentación del usuario y ejecuta el método seleccionado. Luego se realiza la medición de cada uno de los parámetros que fueron escogidos en el dialogo y se actualiza la tabla de parámetros medidos.
24. updateDetectionProgressBar🡪 actualiza la barra de progreso cuando se está realizando una tarea alta consumidora de tiempo como la segmentación y medición de parámetros.
25. elementSelectedInTable(row, column )🡪método que establece la comunicación para resaltar un elemento con la selección del usuario. Este método actualiza el elemento seleccionado en la tabla de mediciones cuando se selecciona una fila, en el oscilograma (visualmente mediante selectElement en SignalDetector) cuandos se realiza un click sobre algún elemento y las ventanas de procesamiento bidimensional cuando se selecciona un elemento. La comunicación se establece en todas las direcciones mediante la suscripción de este método a las señales que lanzan cada uno de sus integrantes (cellPressed en la tabla, elementClicked en el oscilograma y elementSelected en las ventanas 2dim).
26. Classification and threshold handling information…

***Invariantes, Reglas y Convenciones***

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

## **Clase SerializedData**

Descripción

Esta clase es la utilizada para serializar el tema visual utilizado en la aplicación. Contiene las variables que modifican visualmente la aplicación. Se serializa y recupera mediante el módulo pickle de la librería estándar.

***Variables***

1. osc\_background🡪color de fondo
2. osc\_plot🡪 color de linea en el oscilograma
3. osc\_GridX🡪líneas de grid coordenadas en el eje X del oscilograma
4. osc\_GridY🡪líneas de grid coordenadas en el eje Y del oscilograma
5. spec\_GridX🡪líneas de grid coordenadas en el eje X del espectrograma
6. spec\_GridY🡪líneas de grid coordenadas en el eje Y del espectrograma
7. minYOsc🡪 Mínimo valor (% de los valores posibles que puede alcanzar la señal en función de la profundidad de bits) para visualizar en el eje Y del oscilograma.
8. maxYOsc🡪Máximo valor (% de los valores posibles que puede alcanzar la señal en función de la profundidad de bits) para visualizar en el eje Y del oscilograma.
9. minYSpec🡪 Mínimo valor (en Khz según las frecuencias que aparecen representadas en la señal producto de la frecuencia de muestreo) para visualizar en el eje Y del espectrograma.
10. maxYSpec🡪Máximo valor (en Khz según las frecuencias que aparecen representadas en la señal producto de la frecuencia de muestreo) para visualizar en el eje Y del espectrograma.
11. pow\_Back🡪 color de fondo de la ventana de procesamiento unidimensional. Posiblemente deprecated.
12. pow\_Plot🡪 color de las líneas del gráfico de la ventana de procesamiento unidimensional. Posiblemente deprecated.
13. pow\_GridX🡪 grid en el eje X para la ventana de procesamiento unidimensional. Posiblemente deprecated.
14. pow\_GridY🡪 grid en el eje Y para la ventana de procesamiento unidimensional. Posiblemente deprecated.
15. spec\_background🡪Color de fondo para el gráfico de espectrograma.
16. colorBarState🡪 Barra de color para visualizar el espectrograma.
17. histRange🡪 Rango de umbral
18. endColor🡪 Opciones de color para uso futuro
19. startColor🡪 Opciones de color para uso futuro
20. quart1Color🡪 Opciones de color para uso futuro
21. centerColor🡪 Opciones de color para uso futuro
22. quart2Color🡪 Opciones de color para uso futuro

***Métodos***

1. …

***Invariantes, Reglas y Convenciones***

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

## **Clase DuettoListParameterItem**

Descripción

Esta clase almacena los valores de un ListParameter. Es usada para mostrar algunas opciones visuales en un control de tipo combobox, con un orden determinado. Hereda de la clase WidgetParameterItem. Es usada por la clase DuettoSoundLabWindow.

***Variables***

1. targetValue🡪valor del ítem que esta seleccionado actualmente.
2. values 🡪lista de valores posibles de este control. Es usada para mantener el orden de los valores.
3. valuesDict🡪diccionario con los valores posibles.

***Métodos***

1. makeWidget🡪crea el combobox de este control, le enlaza los métodos y propiedades correspondientes.
2. value🡪devuelve el valor del ítem seleccionado actualmente. El combobox de este control usa este método.
3. setValue🡪cambia qué valor está seleccionado actualmente.El combobox de este control usa este método.
4. limitsChanged🡪este método es usado para cambiar los valores posibles del control, se encarga de que todo quede estable después de los cambios.

***Invariantes, Reglas y Convenciones***

Los valores que se le pasan a esta clase al inicio deben ser una lista de tuplas de dos elementos, donde cada tupla tiene como primer elemento un string con el nombre del valor (debe ser único) y como segundo elemento el valor.

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