Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Creación de una librería nativa en C# para el acceso de fichero WFDB

Autor: Renzo Roca Neyra

Tutor: Javier Albert Seguí

2019

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Creación de una librería nativa en C# para el acceso de fichero WFDB

Autor: Renzo Roca Neyra

Tutor: Javier Albert Seguí

TRIBUNAL:

Presidente: _____

Vocal 1º: _____

Vocal 2º: _____

CALIFICACIÓN: _____

FECHA: _____

Índice

Índice de imágenes

Índice de tablas

Resumen

El presente proyecto tiene como objetivo la construcción de una librería nativa en C# que permita la lectura e interpretación de ficheros que almacenan datos de señales biomédicas obtenidas por distintos dispositivos electrónicos los cuales se utilizan en distintas pruebas médicas.

Este proyecto surge ante la necesidad de mejorar el rendimiento de la librería WFDB nativa en C, tal como se había propuesto en el proyecto '*Desarrollo de una aplicación de escritorio para la visualización de bioseñales con WFDB*' de mi compañera Paula Madrilley Nieto.

Palabras clave

ECG, librería, C# , WFDB, señal

Abstract

The objective of this project is the construction of a native library in C # that allows the reading and interpretation of files that store biomedical signal data obtained by various electronic devices which are used in different medical tests.

This project arises from the need to improve the performance of the native WFDB library in C, as proposed in the project 'Development of a desktop application for the visualization of biosignals with WFDB' by my colleague Paula Madrilley Nieto.

Keywords

ECG, library, C #, WFDB, signal

1 Introducción

La implementación de la librería nativa en C# surge como mejora de rendimiento de librería WFDB nativa en C, utilizada en la aplicación ECG Viewer.

ECG Viewer es una aplicación de escritorio diseñada para entornos Windows, que sirve como herramienta de apoyo para el diagnóstico, la prevención y el tratamiento de las distintas enfermedades cardiovasculares, y que permite la lectura e interpretación de los resultados de los electrocardiogramas de larga duración a médicos y enfermeros de una manera fácil e intuitiva [1]. Esta aplicación realizada en C#, utiliza la librería WfdbCsharpWrapper para realizar las lecturas de los ficheros que almacenan grabaciones continuas de señales fisiológicas.

La librería WfdbCsharpWrapper es una biblioteca .NET escrita en C # que encapsula la biblioteca de interfaz nativa WFDB. Esta biblioteca de clases se ha desarrollado para permitir un acceso simplificado a las bases de datos de forma de onda (WFDB) desde cualquier lenguaje .NET mientras ofrece estilo y estándares de codificación .NET. La biblioteca envuelve todas las funciones C nativas y sus estructuras de datos correspondientes y permite hacer llamadas a la biblioteca nativa utilizando los estilos de codificación de procedimientos y OOP (Programación Orientada a Objetos). El acceso multiproceso a los datos de las señales no se admite debido a las limitaciones de la biblioteca nativa [2].

Debido a que ECG Viewer realiza llamadas a bajo nivel de la librería WFDB encapsulando las funciones y desencapsulando los resultados de dichas llamadas de funciones; el tiempo de respuesta es alto. Y este retraso se aprecia significativamente para ficheros de tamaño mayor de 5MB, los cuales terminan bloqueando la aplicación.

Para resolver este problema de rendimiento se ha optado por realizar una librería que este en el mismo lenguaje que la aplicación ECG Viewer.

Esta nueva librería inicialmente se pensó para realizar las operaciones básicas de lecturas y escritura de ficheros WFDB. Pero finalmente se decidió realizar únicamente las operaciones de lectura, debido a la dificultas encontrada en la escritura de los ficheros WFDB ya que demanda tener algún dispositivo electrónico que capture señales fisiológicas y esté conectado al ordenador. Además, la aplicación ECG Viewer solo necesita las operaciones de lectura.

Otros de los motivos por el cual se pensó realizar este proyecto son la falta de programas gratuitos para el manejo y análisis de los ficheros WFDB. Actualmente en el mercado estos

programas tienen un precio elevado dado que son comprados especialmente por hospitales y clínicas. Además, estos programas no son personalizable.

Por lo que esta nueva librería sería la base para implementar aplicaciones en .Net personalizables según las necesidades del usuario. Además de ser escalable ya que se puede ir implementando más funcionalidades a esta librería.

2 Base teórica

En este punto se va a describir todos los conceptos necesarios que hay que comprender para poder implementar la librería. Todos estos conceptos y sus relaciones fueron plasmados en el diseño de la nueva librería.

2.1 La librería WFDB nativa en C

La librería WFDB es un amplio conjunto de funciones (subrutinas) para leer y escribir ficheros en los formatos utilizados por las bases de datos Waveform, en el siguiente punto se explicará este tipo de base de datos. La librería fue desarrollada en 1990 y continúa actualizándose cada año en promedio, fue desarrollada por miembros perteneciente al proyecto "Recurso *de Investigación para Señales Fisiológicas Complejas"* o Physionet. Dicho proyecto se estableció en 1999 bajo los auspicios de los Institutos Nacionales de Salud (NIH). Las misiones originales y continuas de PhysioNet fueron conducir y catalizar la investigación y educación biomédica, en parte ofreciendo acceso gratuito a grandes colecciones de datos fisiológicos y clínicos y software de código abierto relacionado. [3]

La librería WFDB forma parte de un paquete de software denominado **PhysioToolkit**. Dicho paquete se utiliza para el procesamiento y análisis de señales fisiológicas, detección de eventos fisiológicamente significativos utilizando técnicas clásicas y métodos novedosos basados en física estadística y dinámica no lineal, visualización interactiva y caracterización de señales, creación de nuevas bases de datos, simulación de señales fisiológicas y de otro tipo, evaluación cuantitativa y comparación de métodos de análisis, y análisis de procesos no equilibrados y no estacionarios.

2.2 Bases de datos Waveform

Una base de datos Waveform es un conjunto de registros o grabaciones de señales fisiológicas capturadas en alta resolución y de forma continua. Cada base de datos Waveform se clasifica según su tipo de señal fisiológica y anotaciones. Entre los distintas señale fisiológicas tenemos los electrocardiogramas, electroencefalografía, electrogastrograma, respiración, presión arterial, temperatura corporal, etc.

Este tipo de base de datos suele ser utilizado mucho en clínicas y hospitales para almacenar las pruebas realizadas a los pacientes y juntar con sus propias bases de datos clínicas para fines médicos, estadísticos, científicos y aprendizaje automatizado.

PhysioNet ofrece distintas bases de datos Waveform en su componente **PhysioBank**. Es un extenso archivo de grabaciones digitales bien caracterizadas de señales fisiológicas, series de tiempo y datos relacionados para uso de la comunidad de investigación biomédica. Incluye colecciones de señales cardiopulmonares, neuronales y biomédicas de sujetos sanos y pacientes con una variedad de afecciones con importantes implicaciones para la salud pública, que incluyen muerte cardíaca súbita, insuficiencia cardíaca congestiva, epilepsia, trastornos de la marcha, apnea del sueño y envejecimiento. Estas colecciones incluyen datos de una amplia gama de estudios, desarrollados y aportados por miembros de la comunidad de investigación.

Algunos ejemplos de base de datos Waveform públicas son: MIT DB (Massachusetts Institute of Technology) , AHA DB (American Heart Association) y ESC DB (Europe Sociaty of Cardiology).

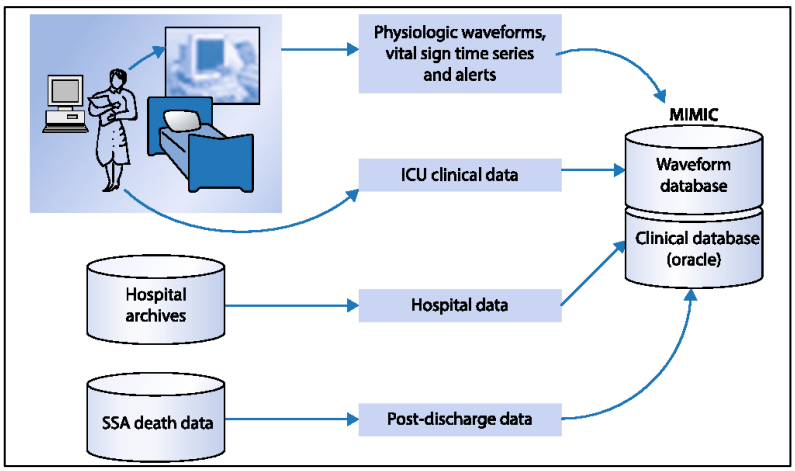


Figura 2-1Uso de un WaveForm database

2.3 Registro

Un registro de wfdb es un conjunto extensible de ficheros que pueden incluir ficheros de señal, ficheros de anotación y fichero de encabezado (fichero obligatorio). Estos ficheros no necesariamente tienen que estar en el mismo directorio o incluso dispositivo físico.

Estos registros, a diferencia de las bases de datos convencionales, tienen un tamaño considerable, normalmente mayor a 1MB.

Cada registro contiene una grabación continua de un solo paciente con cierta duración; por tanto, normalmente se accede a un solo registro y a partir de ese registro se realizan lecturas secuenciales.

Los registros se identifican a través del **nombre del registro,** y los nombres de los ficheros, comienza por el nombre del registro al cual pertenecen [4].

2.3.1 Señal

Es una secuencia finita de **muestras** enteras, que se obtiene al digitalizar una función continua a través de una determinada **frecuencia** de muestreo (número de muestras por segundo) y un **intervalo** de muestreo (tiempo entre muestras adyacentes).

2.3.1.1 Muestra

El valor entero de una **muestra** se interpreta como un voltaje, y las unidades de la muestra se denominan Unidades de convertidor de Analógico al Digital, o **ADU.** La relación entre ADU y la unidad física milivoltio, se denomina ganancia, que indica cuantos ADUs corresponde a un milivoltio.

Todas las señales de un registro dado generalmente tienen la misma frecuencia de muestreo, pero no necesariamente la misma ganancia.

El número de muestras que preceden a una muestra de una señal determinada es un atributo que se utiliza para medir el tiempo, denominado **número de muestra**. Por tanto, muestras con el mismo número de muestra se consideran, muestras simultáneas y el número de muestra de la primera muestra de cada señal es cero.

2.3.2 Anotación

Las anotaciones son etiquetas que apuntan a ubicaciones específicas dentro de un registro y describen eventos en esas ubicaciones. Por ejemplo, muchos de los registros que contienen señales de ECG tienen anotaciones que indican los tiempos de aparición y los tipos de cada latido cardíaco individual ("anotaciones latido a latido") [5].

Una anotación es una etiqueta, asociada a una muestra particular, que describe una característica de la señal en ese momento. La mayoría de las anotaciones son anotaciones QRS e indican el tipo QRS (normal, PVC, SVPB, etc.).

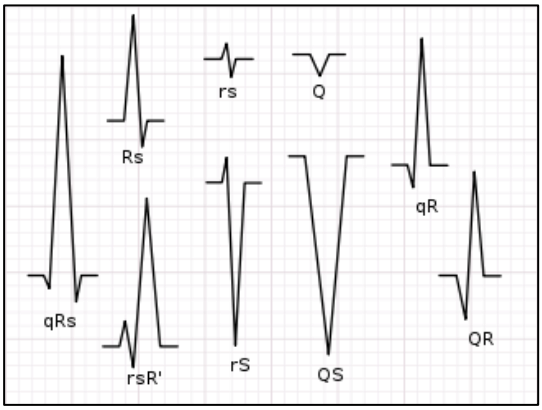


Figura 2-2 Complejo QRS, ejemplo de anotación

Por ejemplo, para MIT DB, algunas muestras (latido - complejo QRS) son descritas por una etiqueta llamada anotación. Normalmente un fichero de anotación para un registro del MIT DB contiene 2000 anotaciones de latidos y un menor número de anotaciones de ritmo y anotaciones de calidad de la señal.

Al igual que las muestras en las señales, las anotaciones se mantienen en el tiempo y el orden de las señales en los ficheros de anotaciones.

Dado un fichero de anotación, una muestra no puede estar asociada a más de 1 anotación en dicho fichero. Pero dicha muestra puede tener asociados a varios ficheros de anotaciones, distinguiéndose dichos ficheros por el nombre del **anotador**.

Normalmente el nombre del anotador es el nombre de la persona o programa que ha creado dicho fichero de anotación.

También se puede asociar una anotación a una sola señal, en vez de a cada muestra de la señal.

2.4 Tipo de ficheros estándar de WFDB

Tal como se ha visto en el apartado anterior, los registros son un conjunto de fichero (datos, cabeceras y anotaciones). Pero también existe un fichero de calibración, que contiene la información de las especificaciones de la calibración de la señal.

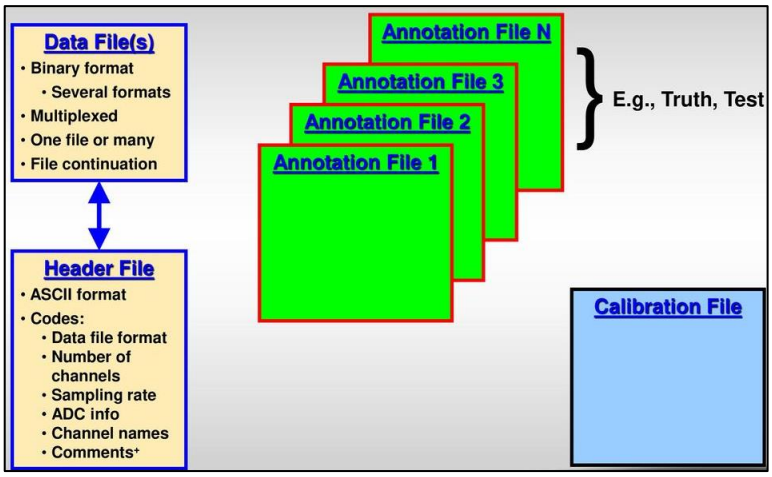


Figura 2-3 Estructura de un registro

2.4.1 Fichero de señal

Los ficheros de señales contienen las señales digitalizadas, son binarios y, por lo general, contienen amplitudes de 16 bits (formato 16), pares de amplitudes de 12 bits empaquetados en bits tripletes de bytes (formato 212). El nombre del fichero de señal suele tener el formato de ***'registro.dat'*** aunque no es obligatorio que contenga el nombre del registro ni la extensión dat.

Los ficheros de señal de WFDB existen en varios formatos. Cualquiera de estos formatos se puede utilizar para ficheros de señales multiplexados, en los que se almacenan **alternando** muestras de dos o más señales [6].

- **Formato 8**: Cada muestra se representa como una primera diferencia de 8 bits; es decir, para obtener el valor de muestra n, se suman los primeros n bytes del fichero de datos de muestra junto con el valor inicial del fichero de cabecera. Cuando se crean archivos de formato 8, las primeras diferencias que no pueden representarse en 8 bits se representan en su lugar por la mayor diferencia del signo apropiado (-128 o +127), y las diferencias subsiguientes se ajustan de modo que se obtenga la amplitud correcta tan rápido como sea posible. posible. Por lo tanto, puede haber pérdida de información si

las señales en otro de los formatos se convierten al formato 8. Las primeras diferencias almacenadas en archivos de formato 8 multiplexado siempre se determinan mediante la resta de muestras sucesivas de la misma señal (de lo contrario las señales con líneas de base) que difieren en 128 unidades o más no podrían representarse de esta manera).

- **Formato 16**: Cada muestra está representada por una amplitud de complemento de dos bits de 16 bits almacenada en primer lugar por byte menos significativo. Cualquier bit de orden superior no utilizado se extiende con signo desde el bit más significativo. Históricamente, el formato utilizado para las cintas de 9 pistas de distribución de bases de datos MIT-BIH y AHA era el formato 16, con la adición de un EOF lógico (octal 0100000) y un relleno de nulos después del EOF lógico.

- **Formato 24**: Cada muestra está representada por una amplitud de complemento de dos bits de 24 bits que almacena el byte menos significativo primero.

- **Formato 32**: Cada muestra está representada por una amplitud de complemento de dos bits de 32 bits almacenada en primer lugar por byte menos significativo.

- **Formato 61**: Cada muestra está representada por una amplitud de complemento de dos bits de 16 bits que almacena el byte más significativo primero.

- **Formato 80**: Cada muestra está representada por una amplitud de 8 bits en forma binaria de desplazamiento (es decir, se debe restar 128 de cada byte sin signo para obtener una amplitud de 8 bits con signo).

- **Formato 160**: Cada muestra está representada por una amplitud de 16 bits en forma binaria de desplazamiento (es decir, se deben restar 32,768 de cada par de bytes sin signo para obtener una amplitud de 16 bits con signo). En cuanto al formato 16, el byte menos significativo de cada par es el primero.

- **Formato 212**: Cada muestra está representada por una amplitud de complemento de dos de 12 bits.
    - La primera muestra se obtiene de los 12 bits menos significativos del primer par de bytes (primero se almacena el byte menos significativo).
    - La segunda muestra se forma a partir de los 4 bits restantes del primer par de bytes (que son los 4 bits altos de la muestra de 12 bits) y el siguiente byte (que contiene los 8 bits restantes de la segunda muestra).
    - El proceso se repite para cada par de muestras sucesivas.

7	6	5	4	3	2	1	0	11	10	9	8	11	10	9	8	7	6	5	4	3	2	1	0
1º BYTE								2º BYTE								3ºBYTE							

Tabla 1 Formato 212

- **Formato 310**: Cada muestra está representada por una amplitud de complemento de dos bits de 10 bits.
    - La primera muestra se obtiene a partir de los 11 bits menos significativos del primer par de bytes (primero el byte menos significativo almacenado), con el bit bajo descartado.
    - La segunda muestra proviene de los 11 bits menos significativos del segundo par de bytes, de la misma manera que el primero.
    - La tercera muestra se forma a partir de los 5 bits más significativos de cada uno de los primer2os dos pares de bytes (los del primer par de bytes son los bits menos significativos de la tercera muestra).
    - El bit no utilizado en cada par de bytes se establece en cero cuando se utiliza la biblioteca WFDB para escribir un archivo de señal de formato 310.
    - El proceso completo se repite para cada conjunto sucesivo de tres muestras.

1º M											3º M					2º M											3º M				
6	5	4	3	2	1	0	-	9	8	7	4	3	2	1	0	6	5	4	3	2	1	0	-	9	8	7	9	8	7	6	5
1º BYTE								2º BYTE								3º BYTE								4º BYTE						

Tabla 2 Formato 310

- **Formato 311**: Cada muestra está representada por una amplitud de complemento de dos bits de 10 bits. Tres muestras se empaquetan en un entero de 32 bits como en el formato 310, pero el diseño es diferente. Cada conjunto de cuatro bytes se almacena en orden little-endian (el byte menos significativo primero, el byte más significativo al final).
    - La primera muestra se obtiene de los 10 bits menos significativos del entero de 32 bits.
    - La segunda se obtiene de los siguientes 10 bits.
    - La tercera de los siguientes 10 bits y los dos bits más significativos no se utilizan (tenga en cuenta que estos bits son puesto a cero cuando use la biblioteca WFDB para escribir un archivo de señal de formato 311).
    - Este proceso se repite para cada conjunto sucesivo de tres muestras.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	-	-
1º BYTE								2º BYTE								3º BYTE								4º BYTE							

Tabla 3 Formato 311

2.4.2 Fichero de anotación

Los nombres de los ficheros de anotación tienen el formato de '***registro.anotador***, donde registro es el nombre del registro y anotador es el nombre del anotador'. Los denominados *'registro.atr* ' son referencia de ficheros de anotación. Los ficheros de anotación son binarios y contienen registros de longitud variable que promedian ligeramente más de 16 bits por anotación [7].

Se pueden leer ficheros de anotación en dos formatos.

- **Formato MIT:** Formato preferido, es compacto (con un promedio de poco más de dos bytes por anotación) y extensible, y se usa normalmente para los ficheros de anotación en línea. Cada anotación ocupa un número par de bytes.
    - El primer byte en cada par es el byte menos significativo.
    - Los seis bits más significativos (A) de cada par de bytes son el código de **tipo** de **anotación**.
    - Y los diez bits restantes (I) especifican el tiempo de la anotación, medido en intervalos de muestra de la anotación anterior (o desde el principio del registro para la primera anotación).
    - Los códigos de tipo están definidos para (0 < A <= ACMAX (49)), aunque existen varias otras posibilidades

- **Formato AHA**: El formato alternativo (formato de distribución AHA DB) utiliza 16 bytes por anotación, y normalmente se usa solo para el intercambio de archivos entre instituciones en cintas de 9 pistas. Dentro de cada bloque:
    - El primer byte no se usa, lo que puede ser muy útil para diferenciar del formato MIT.
    - El segundo byte contiene el código de anotación AHA
    - El tercer al sexto byte contiene el tiempo en PDP-11, formato de entero.
    - Los bytes séptimo y octavo contienen un número de serie de anotación.
    - En los archivos de anotación tomados directamente de las cintas de distribución de la base de datos AHA, los últimos ocho bytes de cada anotación no se utilizan.
    - El tiempo se da en milisegundos medidos desde el comienzo del segmento anotado del registro.
    - En los archivos de anotación en formato AHA generados por las aplicaciones de la biblioteca WFDB, los tiempos de anotación se dan en intervalos de muestra desde el comienzo del registro, y los últimos ocho bytes de cada anotación contienen el

subtipo de anotación MIT (en el noveno byte), el código de anotación MIT (en el décimo byte) y hasta seis caracteres ASCII (en los bytes restantes) utilizados para describir las anotaciones RHYTHM y NOTE.

2.4.3 Fichero de cabecera

Los ficheros de cabecera tienen nombres con el formato '***registro.hea***', donde registro es el nombre del registro. Para cada registro de la base de datos, existe un fichero de cabecera que especifica los nombres de los ficheros de señal asociados y sus atributos. Este fichero contiene texto ASCII orientado a línea y campos [8].

Las líneas tienen un tamaño máximo de 255 caracteres, incluido el salto de línea y los campos de cada línea están separados por espacio en blanco o tabulación.

Los ficheros de cabecera pueden contener las siguientes líneas:

- Línea de registro (línea obligatoria)
- Líneas de especificaciones de señal
- Líneas de especificación de segmento
- Líneas de comentario (precedida por '#')

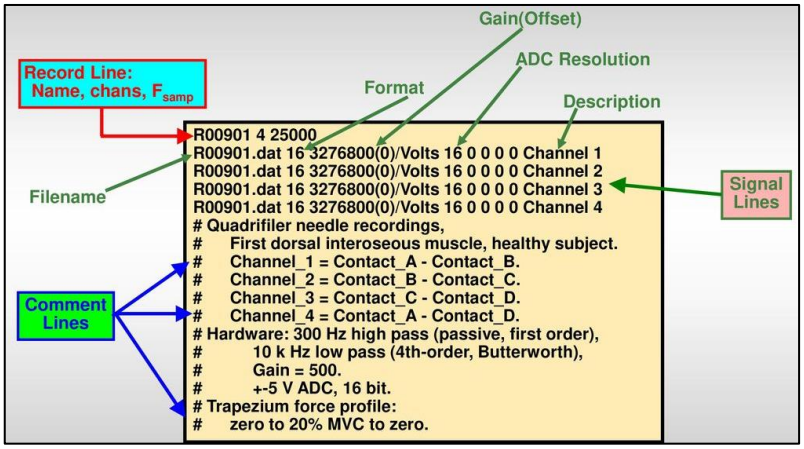


Figura 2-4 Estructura del fichero hea

2.4.3.1 Línea de registro

Los ficheros de cabecera contienen como mínimo una *línea de registro*, que especifica:

- **Nombre del registro (obligatorio):** Una cadena de caracteres que identifican el registro. El nombre del registro puede incluir únicamente letras, dígitos y guiones bajos ('_').

- **Número de segmentos**: Este campo, si está presente, no está separado por espacios en blanco del campo de nombre de registro; más bien, sigue un '/', que sirve como un separador de campo. Si el campo está presente, indica que el registro es un registro de varios segmentos (más de 1 segmento) y cuyo valor representa el número especificado de segmentos, además que el fichero de cabecera contiene líneas de especificación de segmento en lugar de líneas de especificación de señal. El número de segmentos debe ser mayor que cero. Un valor de 1 en este campo es legal, aunque es poco probable que sea útil**.**

- **Número de señales (obligatorio):** Tenga en cuenta que esto no es necesariamente igual al número de archivos de señal, ya que dos o más señales pueden compartir un archivo de señal. Este número no debe ser negativo; Sin embargo, un valor de cero es legal.
- **Frecuencia de muestreo:** Este número puede expresarse en cualquier formato legal para la entrada 3 de números de punto flotante (por lo tanto, '360', '360.', '360.0' y '3.6e2' son todos legales y equivalentes). La frecuencia de muestreo debe ser mayor que cero; si falta, se asume un valor de 250. Se mide en muestras por segundo por señal.
    - **Frecuencia del contador:** Este campo (un número de punto flotante, en el mismo formato que la frecuencia de muestreo) solo puede estar presente si la frecuencia de muestreo también está presente. No está separados por espacios en blanco del campo de frecuencia de muestreo; más bien, sigue un '/', que sirve como un separador de campo. Especifica la diferencia entre los valores de contador que están separados por un intervalo de un segundo. Normalmente, la frecuencia del contador puede derivarse de un contador de cinta analógica o de números de página en una grabación de gráfico. Si la frecuencia del contador está ausente o no es positiva, se supone que es igual a la frecuencia de muestreo.

    - **Contador base:** Este campo solo puede estar presente si la frecuencia del contador también está presente. No están separados por espacios en blanco del campo del contador de frecuencia; más bien, está rodeado de paréntesis, que lo delimitan. El valor del contador base es un número de punto flotante que especifica el valor del contador (número que sirve como referencia de tiempo, en un registro para el cual se define una frecuencia de contador)

correspondiente a la muestra 0. Si está ausente, el valor del contador base se toma como cero.

- **Número de muestras por señal:** Este campo solo puede estar presente si la frecuencia de muestreo también está presente. Si es cero o falta, el número de muestras no se especifica y la verificación de la suma de verificación de las señales está desactivada.
- **Tiempo base:** Este campo solo puede estar presente si el número de muestras también está presente. Indica la hora del día que corresponde al inicio del registro, en formato HH: MM: SS (con un reloj de 24 horas; por lo tanto, 13:05:00 o 13: 5: 0, representa 1:05 pm). Si este campo está ausente, las funciones de conversión de tiempo asumen un valor de 0: 0: 0, que corresponde a la medianoche.
- **Fecha base**: Este campo solo puede estar presente si el tiempo base también está presente. Contiene la fecha que corresponde al inicio del registro, en formato DD / MM / YYYY (por ejemplo, 25/4/1989 es el 25 de abril de 1989).

2.4.3.2 Línea de señal

Cada línea no vacía, sin comentarios, después de la línea de registro en un registro de un solo segmento contiene especificaciones para una señal, comenzando con la señal 0. Los ficheros de cabecera deben contener líneas de especificación de señal válidas para al menos tantas señales como se indicaron en el registro línea (la primera línea no vacía, sin comentarios en el fichero). Las líneas de especificación de señales adicionales no son leídas por las funciones de la biblioteca WFDB. De izquierda a derecha en cada línea, los campos son:

- **Nombre del fichero (obligatorio)**: El nombre del fichero en el que se guardan las muestras de la señal. Aunque el nombre del registro suele ser parte del nombre del fichero de señal, esta convención no es un requisito. Tenga en cuenta que varias señales pueden compartir el mismo fichero (es decir, pueden pertenecer al mismo grupo de señales); Sin embargo, todas las entradas para señales que comparten un archivo dado deben ser consecutivas. El nombre de archivo '-' se refiere a la entrada o salida estándar. La suma de las longitudes de los campos de nombre y descripción del está limitada a 80 caracteres.
- **Formato (obligatorio)**: Este campo es un número entero que especifica el formato de almacenamiento de la señal. Todas las señales en un grupo dado se almacenan en el mismo formato. Los formatos más comunes son el formato 8 (primeras diferencias de ocho bits) y el formato 16 (amplitudes de dieciséis bits). Los siguientes tres campos opcionales, si están presentes, están vinculados al campo de formato (es decir, no están

separados de él por espacios en blanco); pueden considerarse modificadores de formato, ya que describen con más detalle la codificación de muestras dentro del archivo de señal.

- **Muestras por Frame**: Si está presente, este campo sigue una 'x' que sirve como separador de campo. Normalmente, todas las señales en un registro dado se muestrean a la frecuencia de muestreo (base) como se especifica en la línea de registro; en este caso, el número de muestras por frame es 1 para todas las señales, y este campo se omite convencionalmente. Sin embargo, si la señal fue muestreada en algún múltiplo entero, *n*, de la frecuencia de muestreo base, cada **frame** (conjunto de muestras) contiene *n* muestras de la señal, y el valor especificado en este campo también es *n*. (Tenga en cuenta que los múltiplos no enteros de la frecuencia de muestreo base no son compatibles).
- **Skew**: Si está presente, este campo sigue un ':' que sirve como separador de campo. Idealmente, dentro de un registro dado, las muestras de diferentes señales con el mismo número de muestra son simultáneas (dentro de un intervalo de muestreo). Si este no es el caso (como, por ejemplo, cuando una grabación de cinta analógica multipista se digitaliza y el acimut de la cabeza de reproducción no coincide con el de la cabeza de grabación), el skew entre las señales a veces puede determinarse (por ejemplo, localizando características de forma de onda grabadas con relaciones de tiempo conocidas, como señales de calibración). Si se ha hecho esto, el campo de skew se puede insertar en el archivo de encabezado para indicar el número (positivo) de muestras de la señal que se considera que *preceden a la* muestra 0. Estas muestras, si las hay, se incluyen en la suma de comprobación.
- **Desplazamiento byte**: Si está presente, este campo sigue un '+' que sirve como separador de campo. Normalmente, los archivos de señal solo incluyen datos de muestra. Sin embargo, si un archivo de señal incluye un preámbulo, este campo especifica el desplazamiento en bytes desde el comienzo del fichero de señal a la muestra 0 (es decir, la longitud del preámbulo). Los datos dentro del preámbulo no se incluyen en la suma de comprobación de la señal. Tenga en cuenta que el desplazamiento de bytes debe ser el mismo para todas las señales dentro de un grupo determinado (use el campo skew para corregir el skew interseñal). Esta función se proporciona solo para simplificar la tarea de leer los archivos de señales no generados utilizando la biblioteca WFDB; La biblioteca WFDB no admite ningún medio para escribir dichos ficheros, y las

compensaciones de bytes se deben insertar en los ficheros de cabecera manualmente.

- **Ganancia**: Este campo es un número de punto flotante que especifica la diferencia en los valores de muestra que se observaría si se produjera un paso de una unidad física en la señal analógica original. Si la ganancia es cero o falta, esto indica que la amplitud de la señal no está calibrada; en tales casos, se puede asumir un valor de 200. Se mide en unidades ADC (Convertidor de analógico a digital) por unidad física.

    - **Línea base**: Este campo solo puede estar presente si la ganancia de ADC también está presente. No está separado por espacios en blanco del campo ganancia ADC; más bien, está rodeado de paréntesis, que lo delimitan. La línea base es un número entero que especifica el valor de muestra correspondiente a 0 unidades físicas. Si está ausente, la línea de base se considera igual a cero ADC. Tenga en cuenta que la línea de base no necesita ser un valor dentro del rango de ADC; por ejemplo, si el rango de entrada del ADC corresponde a 200-300 grados Kelvin, la línea de base es el valor (precisión extendida) que se asignaría a 0 grados Kelvin.
    - **Unidades**: Este campo solo puede estar presente si la ganancia de ADC también está presente. Sigue el campo de línea base si ese campo está presente, o el campo de ganancia si el campo de línea base está ausente. No está separado por espacios en blanco del campo anterior; más bien, sigue un '/', que sirve como un separador de campo. El campo de unidades es una cadena de caracteres sin espacios en blanco incrustados que especifica el tipo de unidad física. Si el campo de unidades está ausente, se puede suponer que la unidad física es de un milivoltio.

- **Resolución ADC**: Este campo solo puede estar presente si la ganancia de ADC también está presente. Especifica la resolución del convertidor analógico a digital utilizado para digitalizar la señal. Los ADCs típicos tienen resoluciones entre 8 y 16 bits. Si este campo falta o es cero, el valor predeterminado es 12 bits para señales de formato de amplitud, o 10 bits para señales de formato de diferencia (a menos que el campo de *formato* especifique un valor inferior).
- **ADC cero**: Este campo solo puede estar presente si la resolución ADC también está presente. Es un número entero que representa la amplitud (valor de muestra) que se observaría si la señal analógica presente en las entradas del ADC tuviera un nivel que

cayera exactamente en el centro del rango de entrada del ADC. Para un ADC bipolar, este valor suele ser cero, pero un ADC unipolar (offset binario) generalmente produce un valor distinto de cero en el centro de su rango. Junto con la resolución ADC, el contenido de este campo se puede utilizar para determinar el rango de valores de muestra posibles. Si falta este campo, se asume un valor de cero.

- **Valor inicial**: Este campo solo puede estar presente si el ADC cero también está presente. Especifica el valor de la muestra 0 en la señal, pero se usa solo si la señal se almacena en formato de diferencia. Si falta este campo, se asume un valor igual al ADC cero.
- **Suma de comprobación**: Este campo solo puede estar presente si el valor inicial también está presente. Es una suma de comprobación firmada de 16 bits de todas las *muestras* en la señal. (Por lo tanto, la suma de verificación es independiente del formato de almacenamiento). Si se lee todo el registro sin omitir muestras, y la línea de registro del encabezado especifica el número correcto de muestras por señal, este campo se compara con una suma de verificación calculada para verificar que el archivo de señal no ha sido corrompido. Se puede usar un valor de cero como un marcador de posición de campo si el número de muestras no está especificado.
- **Tamaño del bloque**: Este campo solo puede estar presente si la suma de comprobación está presente. Este campo es un entero y suele ser cero. Sin embargo, si la señal se almacena en un archivo que debe leerse en bloques de un tamaño específico, este campo especifica el tamaño del bloque en bytes. (En los sistemas UNIX, este es el caso solo para los archivos especiales de caracteres, correspondientes a ciertas cintas y archivos de disco sin procesar. Si es necesario, el tamaño de bloque puede aparecer como un número negativo para indicar que el archivo asociado carece de soporte de controlador de E/S.) Todas las señales que pertenecen al mismo grupo de señales tienen el mismo tamaño de bloque.
- **Descripción**: Este campo solo puede estar presente si el tamaño del bloque está presente. Cualquier texto entre el campo de tamaño de bloque y el final de la línea se toma como una descripción de la señal. Al crear nuevos registros, siga el estilo utilizado para documentar las señales en los ficheros de cabecera existentes. A diferencia de los otros campos en el fichero de cabecera, la descripción puede incluir espacios incrustados; tenga en cuenta que los espacios en blanco entre el tamaño de bloque y los campos de descripción no se consideran parte de la descripción, sin embargo. Si la

descripción no está presente, las funciones de biblioteca WFDB que leen ficheros de cabecera den una descripción de la forma 'registro *x*, de la señal *y*'.

2.4.3.3 Línea de segmento

Cada línea no vacía, sin comentarios, después de la línea de registro en el fichero de cabecera de **nivel superior** de un registro de múltiples segmentos contiene especificaciones para un segmento, comenzando con el segmento 0. (Las líneas de información no se pueden usar en el fichero de cabecera de nivel superior de un registro de varios segmentos.) Los archivos de cabecera de nivel superior deben contener líneas de especificación de segmento válidas para al menos tantos segmentos como se indicaron en la línea de registro. Las líneas de especificación de segmento extras son ignoradas.

Un **segmento** es simplemente un registro ordinario (segmento único), con su propio encabezado y archivos de señal. Al incluir segmentos en un registro de varios segmentos, las señales de WFDB pueden leer las señales dentro de ellos como si fueran señales continuas, comenzando con las del segmento 0 y continuando con las del segmento 1, sin necesidad de que las aplicaciones hagan nada. Especial para pasar de un segmento a otro. Las únicas restricciones son que los segmentos no pueden contener otros segmentos (deben ser registros de un solo segmento), las frecuencias de muestreo no deben cambiar de segmento a segmento, y la cantidad de muestras por señal debe definirse para cada segmento en la línea de registro del propio fichero de cabecera de nivel inferior del segmento.

Se definen dos tipos de registros multisegmentos. En un registro de diseño fijo, la disposición de las señales es constante en todos los segmentos, y la ganancia de la señal, la línea de base, las unidades, la resolución del ADC y el cero, y la descripción coinciden con las señales correspondientes en todos los segmentos (estas recomendaciones no son exigidas por la biblioteca WFDB, pero es probable que las aplicaciones existentes se comporten de forma impredecible si no se siguen).

Sin embargo, tenga en cuenta que no es necesario utilizar el mismo formato de almacenamiento de señal en todos los segmentos, y en algunos casos es posible que se ahorre mucho espacio al seleccionar un formato óptimo para cada segmento. Cada segmento de un registro de diseño fijo es un registro ordinario que contiene una o más muestras.

En un registro de diseño variable, la disposición de las señales puede variar, las señales pueden estar ausentes en algunos segmentos y las ganancias y líneas de base pueden cambiar entre los segmentos. Un registro de diseño variable se puede identificar por la presencia de un segmento de diseño, que debe ser el segmento 0 y debe tener una longitud de 0 muestras. El segmento de diseño no tiene archivos de señal asociados; su archivo de encabezado especifica la disposición deseada de las señales y sus ganancias y líneas de base. Los nombres de los archivos de señal en un encabezado de segmento de diseño se registran como '~'. Cuando se lee utilizando la biblioteca de WFDB versión 10.3.17 o posterior, las señales de un registro de diseño variable se reorganizan, se cambian y se vuelven a escalar según sea necesario para presentar las señales en la disposición y con las ganancias y líneas de base especificadas en la cabecera del segmento de diseño.

Cada línea de especificación de segmento contiene los siguientes campos, separados por espacios en blanco:

- **Nombre de registro (obligatorio)**: Una cadena de caracteres que identifica el registro de un solo segmento que comprende el segmento. Al igual que en la línea de registro, el nombre del registro puede incluir letras, dígitos y guiones bajos ('_') solamente.
- **Número de muestras por señal (obligatorio)**: Este número debe coincidir con el número especificado en el archivo de encabezado para el registro de un solo segmento que comprende el segmento.

Los registros de diseño variable pueden contener *segmentos nulos*, que pueden identificarse si el nombre del registro dado en la línea de especificación del segmento es '~'. El número de muestras por señal indica la longitud del segmento nulo; cuando se leen, estas muestras tienen el valor WFDB_INVALID_DATA. Los segmentos nulos no tienen fichero de cabecera o señales asociadas.

2.4.3.4 Línea de información/comentario

Las líneas de comentarios siguen a la última línea de especificación de señal en un fichero de cabecera. El contenido de estas líneas (excluyendo el carácter de comentario inicial '#') se conoce como 'cadenas de información'. No debe haber espacios en blanco antes del '#' inicial en ninguna línea.

Las líneas de información no están habilitadas para los registros multisegmentos.

2.4.4 Fichero de calibración

A diferencia de los ficheros de cabecera, señal y anotación, los archivos de calibración no están asociados con registros individuales. Solo se necesita un archivo de calibración si tiene registros que contengan señales que no sean ECG; en este caso, es probable que un solo archivo de calibración sea adecuado para usar con todos sus registros. Los archivos de calibración son archivos de texto, con líneas terminadas por pares ASCII de retorno de carro / avance de línea.

2.5 Fichero EDF

El formato de datos europeo (EDF) es un formato abierto ampliamente soportado para el intercambio de señales fisiológicas registradas, especialmente polisomnogramas (prueba usada en el estudio del sueño). Los archivos EDF encapsulan equivalentes funcionales de los archivos de encabezado y señal, y los archivos EDF + también pueden incluir flujos de anotación (almacenados como señales).

Los archivos EDF comienzan con un encabezado (texto) incrustado que contiene especificaciones de las señales y una cantidad limitada de información demográfica, seguido de las muestras binarias de las señales. Dentro de cada bloque de muestras, típicamente de un segundo a un minuto de duración, todas las muestras de la primera señal se almacenan **consecutivamente**, seguidas de todas las muestras de la segunda señal, etc.

El nombre del archivo tiene como extensión .edf, tanto para EDF y EDF +.

Un fichero edf contiene una grabación poligráfica digitalizada ininterrumpida y consta de un registro de encabezado seguido de registros de datos [9].

2.5.1 Registro de encabezado

El registro de encabezado de longitud variable identifica al paciente y especifica las características técnicas de las señales grabadas.

Los primeros 256 bytes en **ASCII** del registro de encabezado especifican los siguientes campos y subcampos (los cuales están separados por espacios en blanco, y estos no pueden contener dentro espacios en blanco, Si cualquiera de estos subcampos es desconocido, confidencial o no aplicable, se representará por una 'X'):

- **Versión:** Los 8 primeros bytes especifican la versión del formato de datos.
- **Identificación paciente:** Los siguientes 80 bytes identifican la información local del paciente. Constan de los siguientes subcampos:
    - **Código:** Código por el cual se conoce al paciente en la administración del hospital.
    - **Sexo**: Especificado en inglés, puede ser F (*Female*) o M (*Male*).
    - **Fecha de nacimiento:** En formato 'dd-MMM-aaaa' con las abreviaturas del mes en inglés.
    - **Nombre:** Nombre del paciente.
    - **Subcampos adicionales:** Puede existir más subcampos con los bytes restantes.

- **Identificación de la grabación:** Los siguientes 80 bytes identifican de forma local la grabación. Consta de los siguientes subcampos:
    - **Startdate:** Texto literal '*Startdate'* al inicio de la identificación de la grabación.
    - **Fecha de inicio:** En formato 'dd-MMM-aaaa' con el mes en inglés.
    - **Código de investigación**: El código de investigación de la administración del hospital, es decir, el número EEG(Electroencefalografía) o PSG(polisomnografía nocturna).
    - **Código del investigador:** Código que especifica el investigador o técnico responsable.
    - **Código del equipo:** Código que específica el equipo utilizado.
    - **Subcampos adicionales:** Puede existir más subcampos con los bytes restantes.

- **Fecha de inicio de la grabación:** Los siguientes 8 bytes identifican la fecha local en la ubicación del paciente cuando se inició la grabación en formato 'dd.MM.yy'.
- **Hora de inicio de la grabación:** Los siguientes 8 bytes identifican la hora local en la ubicación del paciente cuando se inició la grabación en formato 'HH.mm.ss'. La medianoche es 00:00:00.
- **Nº de bytes en el registro de encabezado:** Suma de los 256 bytes del inicio + (256 bytes x nº de señales).
- **Campo reservado:** Los siguientes 44 bytes deben son usado en EDF + para diferenciar entre una grabación continua o discontinua.
    - **Tipo grabación:**
        - 'EDF +C': Se tiene registros de datos contiguos, es decir, la hora de inicio de cada registro de datos coincide con la hora final del anterior registro de datos. L a hora final de cada registro de datos se calcula sumando la hora de inicio del anterior + la duración en segundo del registro.
        - 'EDF +D': Si tiene registro de datos discontinuos.

- **Nº de registro de datos**: Los siguientes 8 bytes representa el número de registros de datos, -1 si se desconoce en vez de 'X'.
- **Duración de un registro de datos**: Los siguientes 8 bytes representa la duración en segundos de un registro de datos. No se puede utilizar una coma (,) con carácter separador de decimal. Se recomienda que la duración de cada registro de datos sea un número entero de segundos.

- **Nº de señales (ns)**: Los siguientes 4 bytes representa el número de señales en cada registro de datos.

Los siguientes (256 x ns) bytes en **ASCII** del registro de encabezado especifican cada señal del registro de datos. Sus campos y subcampos son los siguientes:

- **Etiqueta**: Los siguientes (16 x ns) bytes especifica el tipo de señal (EEG, temperatura corporal, etc).
- **Tipo de transductor**: Los siguientes (80 x ns) bytes especifica el sensor aplicado, como "electrodo AgAgCl" o "termistor".
- **Dimensión física**: Los siguientes (8 x ns) bytes representa las unidades físicas en la que se miden las muestras (uV or degreeC).
- **Mínimo físico**: Los siguientes (8 x ns) bytes especifican el valor extremo inferior que pueden ocurrir en los registros de datos en unidades físicas.
- **Máximo físico**: Los siguientes (8 x ns) bytes especifican el valor extremo inferior que pueden ocurrir en los registros de datos en unidades físicas.
- **Mínimo digital**: Los siguientes (8 x ns) bytes especifican el valor extremo inferior que pueden ocurrir en los registros de datos en unidades digitales del ADC.
- **Máximo digital**: Los siguientes (8 x ns) bytes especifican el valor extremo superior que pueden ocurrir en los registros de datos en unidades digitales del ADC.
- **Prefiltración**: Los siguientes (80 x ns) bytes identifican la calibración de amplitud
- **Nº muestras en cada registro de datos**: Los siguientes (8 x ns) bytes identifican el número de muestra por cada señal, con este valor se puede obtener la frecuencia de muestreo a través de la duración de la señal.
- **Campo reservado**: Los siguientes (32 x ns) bytes.

De esta manera, el formato permite diferentes ganancias y frecuencias de muestreo para cada señal.

Los máximos y mínimos, tanto digitales como físicos, pueden especificar los siguientes caracterizas de las señales:

- Ganancia
- Línea base
- ADC cero

- ADC resolución

2.5.2 Registro de datos

- **Los registros de datos o frames** contienen épocas consecutivas de duración fija de la grabación poligráfica.
- Se recomienda que la duración de cada registro de datos sea un número entero de segundos.
- Se recomienda que su tamaño (número de bytes) no exceda los 61440.
- Solo si un registro de datos de 1s excede este límite de tamaño, se recomienda que la duración sea menor que 1s (por ejemplo, 0.01).
- Hay tantos registros de datos como se especifica en el registro de encabezado.
- Dentro de cada registro de datos, hay muestras de las *ns* señales, primero están las muestras de la señal 0, seguido las muestras de la señal 1 y así hasta la última señal.
- Cada valor de muestra se representa como un entero de 2 bytes en el formato de complemento a 2.

3 Descripción experimental

En este punto se describirá el diseño de la nueva librería (Clases, propiedades y métodos) basado en el punto anterior.

3.1 Diseño de las clases en C# de la nueva librería WFDB

Para el diseño de las clases nos basaremos en la librería WfdbCsharpWrapper.

Para este proyecto, se ha utilizado algunas clases de dicha librería añadiendo y quitando algunos campos y métodos:

- Clase Record
- Clase Signal
- Clase Annotation
- Clase AnnotationCode
- Clase Annotator

3.1.1 Clase Record

Representa un registro de una base de datos de tipo WFDB

Figura 3-1 Clase Record

3.1.1.1 Propiedades

- `string Name:`
  - Representa el nombre del registro.
  - Se obtiene del nombre del fichero de cabecera (*record.hea*) o el nombre del fichero edf (*record.edf)*.
  - Se utiliza para identificar un registro.

- `int NumberOfSignals:`
  - Representa el número de señales que tiene un registro.
  - Se obtiene de la línea de registro (primera línea) en el fichero de cabecera o en el registro de encabezado del fichero edf.

- Se utiliza para determinar cuántas líneas de señal se va a leer en el fichero de cabecera o calcular el número de bytes que se va a leer en el registro de encabezado del fichero edf.

- `int NumberOfSegments:`
  - Representa el número de segmentos de un registro. Para registros ordinarios, tiene valor 1 por defecto.
  - Se obtiene de la línea de registro en el fichero de cabecera.
  - Esta propiedad es solamente informativa ya que solo se ha implementado la lectura de registro con un único segmento y no registro multisegmentos.

- `int NumberOfSamplesPerSignal:`
  - Representa el número de muestras común para todas las señales del registro. Para registro con más de una frecuencia de muestreo, tiene el valor de la señal con máximo número de muestras.
  - Se obtiene de la línea de registro en el fichero de cabecera.
  - Se puede utilizar para calcular la duración del registro.
- `double SamplingFrequencyPerSignal:`
  - Representa la frecuencia de muestreo común para todas las señales del registro. Para registro con más de una frecuencia de muestreo, tiene el valor de la señal con la máxima frecuencia de muestreo.
  - Se obtiene de la línea de registro en el fichero de cabecera.
  - Se puede utilizar para identificar que muestra se ha obtenido en cualquier segundo; también para calcular el intervalo de muestreo.
  - Se mide en muestras por segundo.

- `double SamplingIntervalPerSignal:`
  - Representa el intervalo de muestreo común para todas las señales del registro. Es decir, el tiempo en segundos que transcurre entre cada muestra.
  - Se obtiene calculando la inversa de la frecuencia de muestreo.
  - Se utiliza para calcular en que segundo fue capturado cualquier muestra identificada por su número de muestra.
  - Se mide en segundos.

- `DateTime? StartDateTime:`
  - Representa la fecha y la hora en la que fue capturada la primera muestra del registro.
  - Se obtiene de la línea de registro en el fichero de cabecera o en el registro de datos del fichero edf.
  - Se utiliza para determina en el día y la hora exacta en la que fue capturada cualquier muestra identificada.

- `double DurationInSeconds:`
  - Representa la duración total del registro en segundos.
  - Se obtiene dividiendo el número de muestras común entre la frecuencia de muestreo común para registros ordinarios. Para registro con más de una frecuencia de muestreo, la duración se calcula multiplicando la duración de un frame (intervalo de frame) por el número de frames.
  - Esta propiedad es únicamente informativa.

- `double counterFrequency:`
  - Representa la frecuencia de contador del registro. Por defecto tiene el mismo valor que la frecuencia de muestreo para los registros ordinarios o la frecuencia de frame para los registros multifrecuencial.
  - Se obtiene de la línea de registro en el fichero de cabecera.
  - Esta propiedad es únicamente informativa.

- `double BaseCounter:`
  - Representa el contador base del registro.
  - Se obtiene de la línea de registro en el fichero de cabecera.
  - Esta propiedad es únicamente informativa.

- `int NumberOfFrames:`
  - Representa el número de frames (conjunto de muestras) de un registro. Para un registro ordinario, un frame contiene exactamente una muestra de cada señal, por lo que el número de frames es igual al número de muestras.

- Se obtiene del registro de encabezado del fichero edf (número de registros de datos).
- Se utiliza para calcular la duración del registro, número de muestras de una señal determinada.

- `double FrameFrequency:`
  - Representa la frecuencia de frame de un registro. Es decir, el número de frames por segundo. Para un registro ordinario, la frecuencia de frame es igual que la frecuencia de muestreo.
  - Se obtiene calculando la inversa del intervalo de frame o duración del frame en segundos.
  - Esta propiedad es únicamente informativa.

- `double FrameInterval:`
  - Representa el intervalo de frame, es decir el número de segundos que transcurre entre frames consecutivos o la duración de un frame en segundos. Para registros ordinarios, el intervalo de frame es igual que el intervalo de muestro.
  - Se obtiene directamente del registro de encabezado del fichero edf, como duración de registro de datos en segundos.
  - Esta propiedad es únicamente informativa.

- `List<string> Info:`
  - Representa información adicional asociada del registro.
  - Se obtiene de las líneas de información del fichero de encabezado o del registro de encabezado de los ficheros edf.
  - Esta propiedad es únicamente informativa.

- `List<Signal> Signals:`
  - Lista de instancias de señales del registro.

3.1.1.2 Métodos

- `public void AddSignal(Signal newSignal)`

  Añade una señal a la lista de señales del registro, estableciendo el número de muestras de señal común, la frecuencia de señal común y determinando el grupo al que pertenece la señal y el número de señal dentro del grupo al que pertenece.

- `public int SecondToIdSample(double second)`

  Determina el identificador de la muestra que fue capturada en un segundo determinado. El id de la muestra es calculado multiplicando el segundo por la frecuencia de muestreo.

- `public double IdSampleToSecond(int idSample)`

  Determina el segundo en el que fue capturada una muestra determinada. El segundo de la muestra es calculado multiplicando el id de la muestra por el intervalo de muestreo.

- `public static Record ReadHeaderFile(string recordName, Stream headerFile)`

  Lee un fichero de cabecera (.hea) de un registro determinado. Devuelve una instancia de la clase Record con las propiedades rellenadas tanto del registro como de las señales.

  Está método fue influenciado por la función de '*static int readheader(const char *record)*' de fichero '*signal.c*'.

- `public static Record RecordLineToRecord(string recordName,string recordLine)`

  Analiza la línea de registro de un fichero de cabecera (.hea) de un registro determinado. Devuelve una instancia de la clase Record con las propiedades rellenadas.

  Está método fue influenciado por la función de '*static int readheader(const char *record)*' de fichero '*signal.c*'.

3.1.2 Clase Signal

Representa una señal de un registro. Entre tus propiedades tenemos:

Figura 3-2 Clase Signal

3.1.2.1 Propiedades

- `string FileName:`
  - Representa el nombre fichero de datos de la señal.
  - Se obtiene de la línea de señal del fichero de cabecera o del nombre del fichero edf.
  - Se utiliza para leer las muestras de la señal e identificar a que grupo pertenece la señal.
- `int Group:`
  - Representa el grupo al que pertenece la señal. Señales con el mismo nombre de fichero de datos pertenecen al mismo grupo.
  - Se obtiene calculando cuantos ficheros de datos posee el registro.
  - Se utiliza para recorrer la lectura de los ficheros de datos en orden secuencial.
- `int Number:`
  - Representa el identificador dentro del grupo al que pertenece la señal.
  - Se obtiene calculando el número de señales que pertenece al mismo grupo.
  - Se utiliza para recorrer las señales dentro de un grupo.
- `string Description:`
  - Representa la descripción o etiqueta de la señal como el identificador del electrodo en el electrocardiograma.
  - Se obtiene de la línea de señal en el fichero de cabecera o del registro de encabezado en el fichero edf.
  - Se puede utilizar para identificar una señal.
- `int InitValue:`
  - Representa el valor inicial de la primera muestra. Por defecto tiene el mismo valor que adcZero.
  - Se obtiene de la línea de señal del fichero de cabecera.
  - Esta propiedad es únicamente informativa.

- `double Gain:`
  - Representa la ganancia de la señal. Por defecto tiene un valor de 200 ADU/milivoltio.
  - Se obtiene directamente de la línea de señal del fichero de cabecera o del registro de encabezado del fichero edf calculando: (`max_digital-min_digital)/(max_fisico –min_fisico`).
  - Se utiliza para obtener el valor en unidades física de una muestra de la señal.

- `int Baseline:`
  - `Represen el valor de la línea base de la señal.`
  - Se obtiene de la línea de señal del fichero de cabecera o del registro de encabezado del fichero edf calculando: max_digital – ganancia*max_fisico.
  - Se utiliza para obtener el valor en unidades física de una muestra de la señal.
- `SignalStorageFormat Format:`
  - Representa el formato en lo que están representados las muestras en el fichero de datos o registro de datos.
  - Se obtiene de la línea de señal del fichero de cabecera.
  - Se utiliza para leer las muestras.

- `string Units:`
  - Representa la unidad física en la que se miden las muestras.
  - Se obtiene de la línea de señal del fichero de cabecera o del registro de encabezado del fichero edf.
  - Esta propiedad es únicamente informativa.

- `int SamplesPerFrame:`
  - `Representa el número de muestras de un frame o registro de datos. Para registros ordinarios, el número de muestras por frame es 1.`
  - Se obtiene de la línea de señal del fichero de cabecera o del registro de encabezado del fichero edf.
  - Esta propiedad se utiliza para leer las muestras de una señal.
- `int BlockSize:`
  - `Representa el tamaño de bloque de memoria utilizada en el SSOO linux.`
  - Se obtiene de la línea de señal del fichero de cabecera.
  - Esta propiedad es únicamente informativa.
- `int AdcResolution:`
  - `Representa la resolución de convertidor de analógico al digital.`
  - Se obtiene directamente de la línea de señal del fichero de cabecera o calculando el número de bits necesarios para representa el rango digital (maxDigital – minDigital).
  - Esta propiedad es únicamente informativa.

- `int AdcZero:`
  - `Representa el valor cero de convertidor de digital al analógico.`
  - Se obtiene directamente de la línea de señal del fichero de cabecera o calculando la semisuma del máximo digital y el mínimo digital.
  - Esta propiedad es únicamente informativa.
- `int NumberOfSamples:`
  - `Representa el número de muestras capturada en la señal.`
  - Se obtiene directamente de la línea de señal del fichero de cabecera o calculando el producto: número de frames por número de muestras por frame.
  - Esta propiedad es únicamente informativa.
- `Int CheckSum:`
  - `Representa la suma de comprobación del registro.`
  - Se obtiene directamente de la línea de señal.
  - Esta propiedad es únicamente informativa.
- `int Skew:`
  - `Representa el` número de muestras de la señal que se considera que preceden a la primera muestra.
  - Se obtiene directamente de la línea de señal.
  - Esta propiedad es únicamente informativa.
- `int ByteOffset:`
  - `Representa el tamaño del preámbulo del fichero de datos.`
  - Se obtiene directamente de la línea de señal o del registro de encabezado del fichero edf.
  - Esta propiedad se utiliza para empezar a leer las muestras desde el desplazamiento.

- `double SamplingFrequency:`
  - `Representa el número de muestras capturadas por segundo de la señal.`
  - Se obtiene directamente de la frecuencia de muestreo común del registro al que pertenece o si es un registro multifrecuencial, se obtiene dividendo el número de muestras por frame de la señal entre el intervalo o duración del frame.
  - Se puede utilizar para identificar que muestra se ha obtenido en cualquier segundo; también para calcular el intervalo de muestreo.

- `double samplingInterval:`
  - `Representa el tiempo que transcurre entre muestras consecutivas de la señal.`
  - Se obtiene calculando la inversa de la frecuencia de mustreo.
  - Se utiliza para calcular en que segundo fue capturado cualquier muestra identificada por su número de muestra.

- `List<int> Samples:`
  - Lista de muestras de la señal.

3.1.2.2 Métodos

❖ `public double SampleToPhysicalUnits(int idSample)`

Convierte el valor digital (ADU) de una muestra dada en su valor físico (mV).

Esto se logra calculando: (valor de la muestra en adu – línea base) / ganancia

❖ `public double IdSampleToSecond(int idSample)`

Determina el segundo en el que fue capturada una muestra determinada. El segundo de la muestra es calculado multiplicando el id de la muestra por el intervalo de muestreo.

❖ `public int SecondToIdSample(double second)`

Determina el identificador de la muestra que fue capturada en un segundo determinado. El id de la muestra es calculado multiplicando el segundo por la frecuencia de muestreo.

❖ `public static Signal SignalLineToSignal(string signalLine)`

Analiza la línea de señal de un fichero de cabecera (.hea) de un registro determinado. Devuelve una instancia de la clase Signal con las propiedades rellenadas.

Está método fue influenciado por la función de '*static int readheader(const char *record)*' de fichero '*signal.c*'.

❖ `public static void ReadSignalFile(FileStream fileSignal, List<Signal> signalsGroup)`

Lee un fichero de señal (fileSignal) de un registro determinado. Devuelve una instancia

la lista de señales (signalsGroup) con las propiedades y muestras rellenadas.

- `private static void ReadSignalF16(FileStream fileSignal, List<Signal> signals)`

  Lee las muestras de todas las señales del fichero de datos (fileSignal) en el formato 16. Este tipo de método existe por cada formato existente.

3.1.3 Clase AnnotationCode

Representa los distintos tipos de anotaciones que puede tener una anotación, cada tipo es identificado por su código de anotación. Estos códigos son propiedades estáticas de solo lectura y están establecidos entre 0 y 49 incluidos. De los cuales los códigos del 42 al 48 están reservados para ser definidos por el usuario y el resto de códigos ya están definidos entre anotaciones de denotan un complejo QRS y los que no.

Cada código de anotación tiene sus características almacenadas en vectores de tamaño fijos (50): el vector descripción; mapeo 1; mapeo 2; mnemotécnico de electrocardiograma; posición de anotación; código AHA; etc. Estos vectores fueron obtenidos del fichero '*ecgmap.h*'.

CÓDIGO	TIPO	DESCRIPCIÓN	MNOTÉCNICO	MAP1	MAP2	AHA	QRS	POSICION
0	NotQrs			NotQrs	NotQrs	O	False	APUndef
1	Normal	Normal beat	N	Normal	Normal	N	True	APStd
2	Lbbb	Left bundle branch block beat	L	Normal	Normal	N	True	APStd
3	Rbbb	Right bundle branch block beat	R	Normal	Normal	N	True	APStd
4	Aberr	Aberrated atrial premature beat	a	Normal	Svpb	N	True	APStd

Tabla 4 Tipos de anotaciones

Figura 3-3 Clase AnnotationCode

3.1.3.1 Propiedades

- `byte Value`
  - Representa el código/tipo de la anotación.

- `string Description`
  - Representa la descripción del tipo de anotación. Esta propiedad está vacía para los códigos reservados ya que están pendientes de ser definidos por el usuario.
  - Se obtiene del vector `descriptions`.
- `bool IsAnnotation`
  - Representa a la bandera que indica si el código de anotación es válido o inválido.
  - Se obtiene comprobando si el código está comprendido entre 1 y 49. Ya que el código 0 se reserva para códigos vacíos.
- `bool IsQrs`
  - Representa a la bandera que indica si el código de anotación es un complejo QRS.
  - Se obtiene del vector isQrs.
- `AnnotationCode Map1`
  - Indica si las anotaciones que son de tipo QRS, son anotaciones:
    - Latidos normales.
    - Contracción ventricular prematura
    - Fusión de latido normal y ventricular
    - Latido no clasificado durante aprendizaje
  - Se obtiene del vector map1.

- `AnnotationCode Map2`
  - Indica si las anotaciones que son de tipo QRS, son anotaciones:
    - Latidos normales.
    - Contracción ventricular prematura
    - Fusión de latido normal y ventricular
    - Latido no clasificado durante aprendizaje
    - Latido supra ventricular prematuro o latido ectópico (auricular o nodal)
  - Se obtiene del vector map2.

- `AnnotationPos AnnotationPos`
  - Indica la posición adecuada de la anotación de tipo QRS:
    - Estándar
    - Alta
    - Baja
  - Se obtiene del vector annotationPos.

- `string EcgString:`
  - Representa un mnemotécnico de las anotaciones.
  - Se obtiene del vector `ecgMnemonicStrings.`

3.1.3.2 Métodos

➢ `public static AnnotationCode MapAhaToMit(char ahaCode)`

Convierte de un código de anotación AHA a un código de anotación MIT utilizando el vector mapAha2Mit.

ID_AHA	CODIGO_AHA	CODIGO_MIT	ID_MIT
0	E	Vesc	10
1	F	Fusion	6
2	G	NotQrs	0
3	H	NotQrs	0
4	I	NotQrs	0
5	J	NotQrs	0
6	K	NotQrs	0
7	L	NotQrs	0
8	M	NotQrs	0
9	N	Normal	1
10	O	Note	22
11	P	Pace	12
12	Q	Unknow	13
13	R	RonT	41
14	S	NotQrs	0
15	T	NotQrs	0
16	U	Noise	14
17	V	Pvc	5
18	W	NotQrs	0
19	X	NotQrs	0
20	Y	NotQrs	0
21	Z	NotQrs	0
22	[	Vfon	32
23	\	NotQrs	0
24	]	Vfoff	33

Tabla 5 Correspondencia de AHA código a MIT código

- `public char ToAha(int subCode)`

  Convierte del código de anotación MIT a un código de anotación AHA utilizando el vector mapAha2Mit. Solo se comprueba `subCode` si la anotación es de tipo ruido.

- `public static AnnotationCode ParseEcgString(string ecgString)`

  Convierte de un mnemotécnico en su código MIT utilizando el vector `mnemonicStrings`.

3.1.4 Clase Annotator

Representa un anotador, creador del fichero de anotaciones.

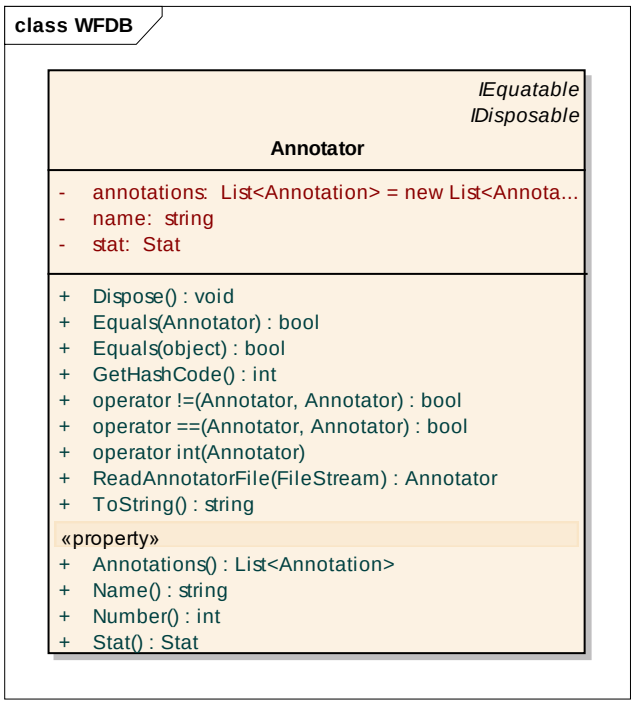


Figura 3-4 Clase Annotator

3.1.4.1 Propiedades

- `string Name`
  - Representa el nombre del anotador.
  - Se obtiene de la extensión del fichero de anotación.

- Se utiliza para identificar los ficheros de anotación.

- `Stat Stat`
  - Indica el tipo de acceso al fichero y el formato de este (MIT o AHA). Aunque para el proyecto solo se utiliza el acceso de lectura.
- `int Number`
  - Número de identificador del anotador.
- `List<Annotation> Annotations`
  - Lista de anotaciones del anotador.

3.1.4.2 Métodos

❖ `public static Annotator ReadAnnotatorFile(FileStream annotatorFile)`

Lee un fichero de anotación, en formato MIT o AHA. Devuelve una instancia de la clase Annotator con todas las anotaciones leídas. Este método está influenciado por las funciones del fichero *'annot.c'*:

- *'FINT annopen(char *record, WFDB_Anninfo *aiarray, unsigned int nann)'*
- *'FINT getann(WFDB_Annotator n, WFDB_Annotation *annot)'*

3.1.5 Clase Annotation

Representa una anotación realizada por el anotador sobre una muestra determinada.

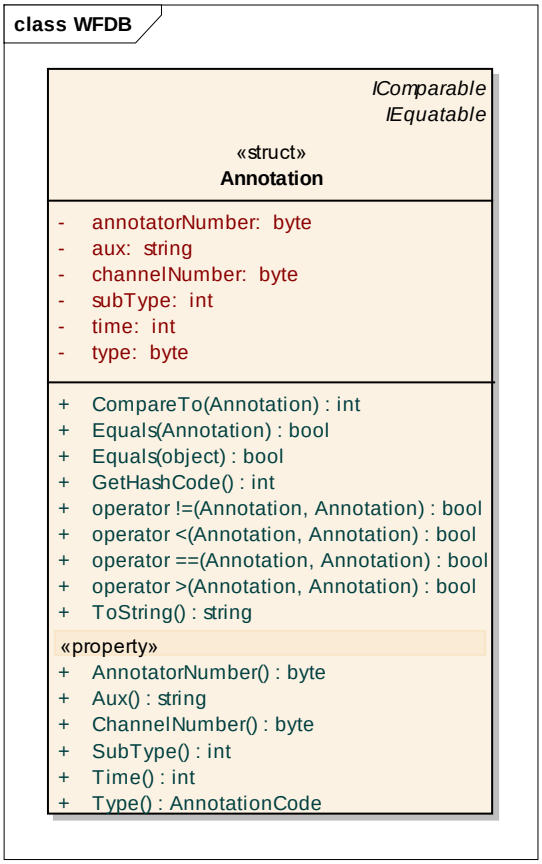


Figura 3-5 Clase Annotation

3.1.5.1 Propiedades

- `int Time`
  - Representa el tiempo en que fue capturada la anotación, este tiempo se corresponde al número de muestra de la anotación.
- `AnnotationCode Type`
  - Representa el tipo de anotación. Este valor entero está comprendido entre 1 y 48.
- `int SubType`
  - Representa al subtipo de la anotación. Esta propiedad es usada para las anotaciones de tipo Ruido y Artefacto para indicar las señales que son afectadas.
- `byte ChannelNumber`
  - Representa el número del canal o señal relacionada con la anotación.
- `byte AnnotatorNumber`
  - Representa el número de anotador de la anotación.
- `string Aux`
  - Representa la información relacionada con la anotación.

- Es utilizada para las anotaciones de cambio de ritmo, para indicar el nuevo ritmo y para las anotaciones de tipo nota, para almacenar texto el comentario

3.1.6 Clase InforRecordEDF

Representa a un registro con información en formato EDF. Esta clase se ha creado para poder manejar ficheros en formato EDF y convertido al formato estándar.

Todas las propiedades de InforRecordEDF son obtenidas del registro de encabezado del fichero edf.

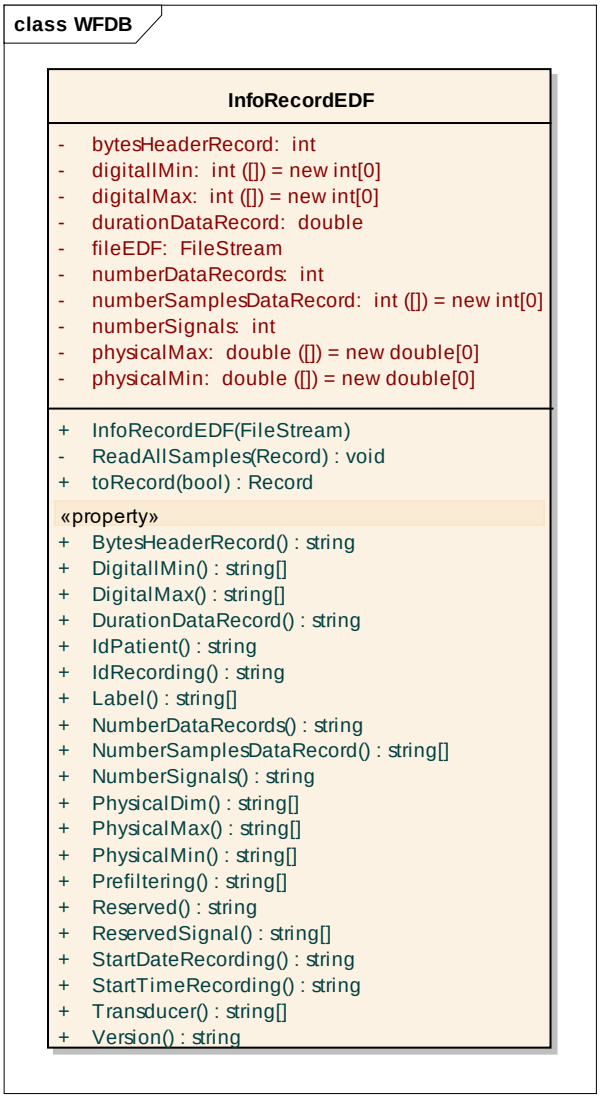


Figura 3-6 Clase InfoRecordEDF

3.1.6.1 Propiedades

- `string Version`
  - Representa la versión del formato EDF. Puede ser versión '0' o 'BIOSEMI'.
  - Se utiliza para identificar en que formato de representación binaria están almacenadas las muestras de las señales. Para versión '0' el formato de las muestras es de 16 bits en *little endian* y para la versión 'BIOSEMI' el formato de las muestras es de 24bits en *little endian*.

- `string IdPatient`
  - Representa la información local del paciente: Código, sexo, fecha de nacimiento, nombre del paciente, etc.
  - Esta propiedad es solamente informativa.

- `string IdRecording`
  - Representa la información local del registro: Fecha de inicio, código de investigación, código de investigador, código de equipo, etc.
  - Esta propiedad es solamente informativa
- `string StartDateRecording`
  - Representa la fecha de inicio de la grabación.
- `string StartTimeRecording`
  - Representa la hora de inicio de la grabación.
- `int bytesHeaderRecord`
  - Representa el tamaño en bytes que ocupa el registro de encabezado.
  - Este número es utilizado como el desplazamiento en bytes de las señales del registro.
- `string Reserved`
  - Representa algún otro campo personalizado del registro.
  - Se utiliza como información adicional del registro.
- `int numberDataRecords`
  - Representa el número de registro de encabezado.
  - Es utilizado como número de frames en el registro.
- `double durationDataRecord`
  - Representa la duración en segundos de un registro de encabezado.
  - Es utilizado cómo intervalo o duración de frame.
- `int numberSignals`

- Representa el número de señales del registro.

- `string[] Label`
  - Representa los tipos de señales del registro.
  - Se utiliza como descripción de las señales.
- `string[] Transducer`
  - Representa el sensor aplicado.
  - Se utiliza como información adicional del registro.
- `string[] PhysicalDim`
  - Representa las unidades físicas en las que se miden las muestras en las señales.
  - Se utiliza como unidades de las señales.

`double[] physicalMin`

  - Representa los valores físicos mínimos de las señales del registro.
  - Se utiliza para calcular otras propiedades de la señal.

- `double[] physicalMax`
  - Representa los valores físicos máximos de las señales del registro.
  - Se utiliza para calcular otras propiedades de la señal.

- `int[] digitallMin`
  - Representa los valores digitales mínimos de las señales del registro.
  - Se utiliza para calcular otras propiedades de la señal.

- `int[] digitalMax`
  - Representa los valores físicos mínimos de las señales del registro.
  - Se utiliza para calcular otras propiedades de la señal.

- `string[] Prefiltering`
  - Representa las calibraciones de amplitud de las señales del registro.
  - Se utiliza como información adicional del registro.

- `int[] numberSamplesDataRecord`
    - Representa los números de muestras que existen en un registro de datos o frame.
    - Se utiliza para como número de muestras por frame de cada señal.

- `string[] ReservedSignal`
    - Representa campos personalizados de las señales del registro.
    - Se utiliza como información adicional del registro.
-

3.1.6.2 Métodos

❖ `public InfoRecordEDF(FileStream fs)`

Constructor de la clase InfoRecordEDF, recibe el fichero edf y lee según las especificaciones del formato edf.

Este método está influenciado de la función del fichero *signal.c* :

- `static int edfparse(WFDB_FILE *ifile)`

❖ `public Record toRecord(bool readSamples=false)`

Convierte la información de registro EDF en una instancia de la clase Record. Si readSamples es true, lee los registros de datos o frames del fichero edf.

Este método está influenciado de la función del fichero *signal.c* :

- `static int edfparse(WFDB_FILE *ifile)`

❖ `private void ReadAllSamples(Record record)`

Lee todos los registros de datos del fichero edf y guarda las muestras leídas en el registro record, en cada señal correspondiente.

3.1.7 Clase ManagerWFDB

Representa la interfaz entre el usuario y los registros de tipo WFDB. Las funciones principales de la librería WFDB, como leer fichero de cabecera, señales, anotaciones, etc.

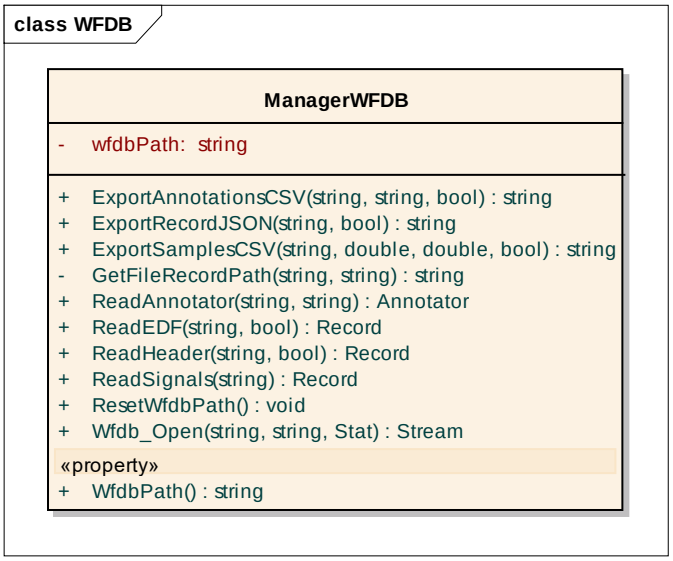


Figura 3-7 Clase ManagerWFDB

3.1.7.1 Propiedades

- `public static string WfdbPath`
  Representa las rutas de las carpetas raíces donde se va a buscar los ficheros de los registros. Si hay más de una ruta deben estar separadas con ';'. Se utiliza para saber en qué carpetas buscar los registros, y así evitar tener que pasarle la ruta absoluta de los registros. Por defecto tiene el valor de la variable de entorno WFDB.

3.1.7.2 Métodos

- ❖ `public static Stream Wfdb_Open(string recordName, string type, Stat mode)`
  Busca un fichero de un tipo determinado (extensión del fichero) en las carpetas especificada en la propiedad WfdbPath. Si encuentra el fichero, lo abre el fichero en el modo que indica el parámetro *mode*. Devolviendo un objeto de tipo *Stream*.

- ❖ `public static Record ReadHeader(string recordName, bool isEDF = false)`
  Busca y lee el fichero de cabecera dado el nombre del registro y el tipo de registro (EDF

o HEA). Devuelve un objetivo de tipo *Record*, con la información del registro y señales rellenadas.

- `public static Record ReadSignals(string recordName)`
  Buscará y lee los ficheros de señales o datos especificado en el fichero de cabecera. Devuelve un objetivo de tipo Record, con la información de los registros, señales y muestras leídas.

- `public static Record ReadEDF(string recordName, bool readSignals=true)`
  Lee un registro de tipo EDF, es decir lee el registro de encabezado y, por defecto, los registros de datos del fichero EDF. Devuelve un objetivo de tipo *Record* con toda la información del registro, señales y sus muestras.

- `public static Annotator ReadAnnotator(string recordName,string annotatorName)`
  Lee un fichero de anotación de un registro determinado dado el nombre de anotador (extensión del fichero de anotación). Devuelve un objeto de tipo *Annotator*, con toda la información obtenida del fichero de anotación.

- `public static string ExportSamplesCSV(string recordName,double secondInit=0, double secondEnd=-1,bool isEDF=false)`
  Exporta en un fichero *csv* las señales obtenidas de un registro a entre el segundo inicial y el segundo final, especificados como parámetro de entrada. Por defecto el registro a leer no es de tipo EDF y se exportan todas las muestras del registro, es decir a partir del segundo 0 y la duración total del registro.

- `public static string ExportAnnotationsCSV(string recordName, string annotatorName,bool readHeader=false)`
  Exporta en un fichero *csv* las anotaciones obtenidas del fichero de anotación de un registro. Esta función puede obtener el día y la hora en la que se ha realizado una anotación si el parámetro leer cabecera está a true. Este parámetro por defecto está a false.

- `public static string ExportRecordJSON(string recordName,bool isEDF=false)`
  Exporta en un fichero JSON toda la información obtenida del fichero de cabecera o registro de encabezado un registro. E s decir las propiedas de un objeto de la clase *Record.*

3.2 Estructura de la solución de la librería

Para el desarrollo de la librería se ha creado el proyecto en .Net en Visual Studio llamado LIBRERÍA_WFDB que consta de dos proyectos, un proyecto para el desarrollo de la librería, y otro para las pruebas unitarias de los métodos de la librería.

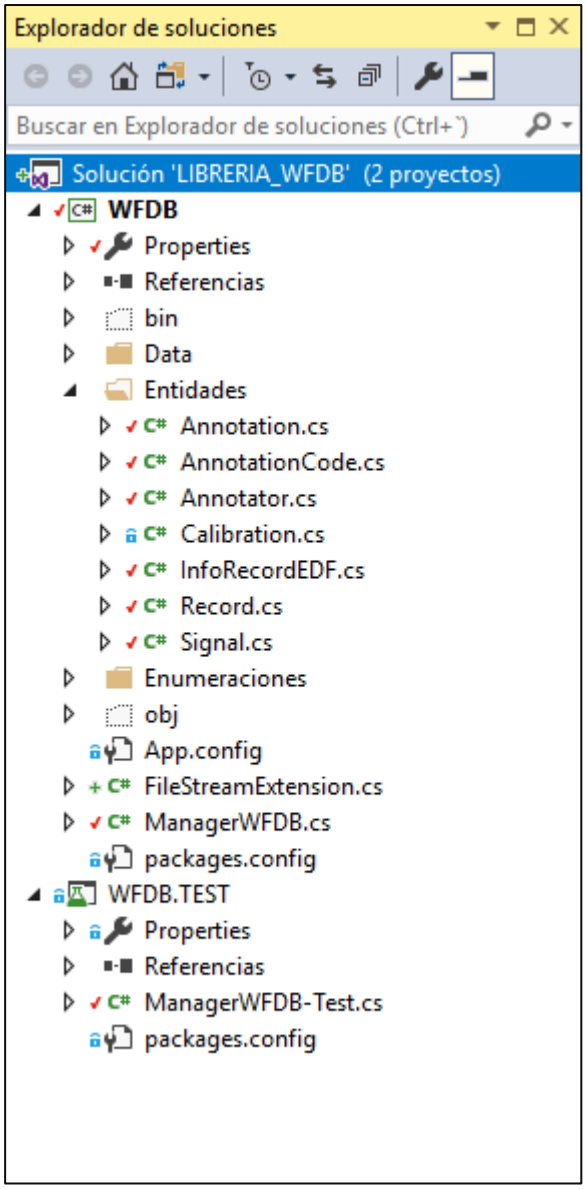


Figura 3-8 Estructura de la solución LIBRERIA_WFDB en Visual Studio

3.2.1 Proyecto WFDB

En este proyecto se ha desarrollado las clases anteriormente mencionadas en el diseño. Se ha organizado de la siguiente manera:

- Data: Carpeta donde se almacena ejemplos de registros perteneciente a distintas bases de datos, utilizados para las pruebas. Cada registro contiene su

fichero de cabecera, fichero de señal y algunos casos ficheros de anotación. Además, contiene los ficheros de salida de los métodos de exportación como los CSV y JSON.

- Entidades: Carpeta donde se almacena las clases basadas entidades de WFDB (Record, Signal, Annotation, etc).
- Enumeraciones: Carpeta donde se almacena los enum o enumerables del proyecto como los formatos de las señales, posiciones de las anotaciones.
- FileStreamExtension.cs: Extensión de la clase FileStream para leer las muestras de las señales en distintos formatos.
- ManagerWFDB.cs: Clase ya explicada en el diseño, interfaz de las entidades con el usuario.

3.2.2 Proyecto WFDB.TEST

Proyecto para los test unitarios del proyecto WFDB, consta únicamente de las pruebas de los métodos de la clase ManagerWFDB.

La nomenclatura utilizada para los nombres de los métodos de test unitaria es la siguiente:

Nombre del método a probar, seguido del contexto del método y por último el resultado esperado. Por ejemplo, para probar el método ReadSignals de la clase ManagerWFDB, cuando existe el fichero de señal, se espera que devuelva las señales con las muestras. Por lo que un nombre valido sería ReadSignals_whenFileSignalExist_returnSignalWithSamples.

Cada método de tiene tres partes:

1. Arrange: La primera consta de la organización para la prueba unitaria, donde se declara los objetos necesarios para llamar al método de prueba.
2. Act: La segunda parte es la llamada al método que se quiere comprobar su correcto funcionamiento.
3. Assert: La última parte son las comprobaciones del resultado obtenido del método anteriormente llamado son correcto. Comprobar que sean los resultados esperados.
   Si pasa todas las comprobaciones la prueba unitaria se da como valida.

```
[TestMethod]
public void ReadSignals_whenFileSignalExist_returnSignalsWithSamples()
{
    //Arrange
    Record result;


    //Act
    result = ManagerWFDB.ReadSignals("01911");

    //Assert
    Assert.AreEqual(30, result.Signals.ElementAt(0).Samples[184]);
    Assert.AreEqual(-13, result.Signals.ElementAt(1).Samples[128]);
}
```

Este es un ejemplo del método test anteriormente mencionado, donde se lee las señales del registro 01911 y se espera que la muestra número 184 tenga valor 30 y la número 128 tenga valor -13.

3.3 Desarrollo de métodos importantes de la librería

Entre los métodos mencionados en el diseño de la librería, lo más importante a destacar son los siguientes:

3.3.1 Método de la lectura de la línea de registro del fichero de cabecera

El método estático *RecordLineToRecord* perteneciente a la clase *Record*, realiza la lectura de la línea de registro del fichero de cabecera según las especificaciones mencionadas en la base teórica. Este método se considera importante porque es el primer paso para la lectura de cualquier registro.

Recibo la línea de especificación y la analiza token a token comprando que se ajuste al formato correcto.

Primero comprueba que la línea de registro comience por el nombre del registro, en caso contrario saltaría una excepción.

```
Public static Record RecordLineToRecord(string recordName,string recordLine)
{
 Record registro = new Record(recordName);
 if (!recordLine.StartsWith(recordName))
 throw new Exception(String.Format("RecordLineToRecord: record name in
record {0} header is incorrect", recordName));
```

Después se procede a dividir la cadena de texto por espacios en blancos, obteniendo un array de tokens. Si el número de tokens menor que el esperado se lanza excepción

```
string[] tokens = recordLine.Trim().Split(' ').Where(s =>
!string.IsNullOrWhiteSpace(s)).ToArray();

if (tokens.Length < 2)
 throw new Exception(String.Format("RecordLineToRecord:obsolete format in
record {0} header", recordName));
```

Una vez dividido en tokens, se procede a recorrer el array procesando cada token de forma determinada con un switch según la posición que ocupe en el array.

```
for (int i = 0; i < tokens.Length; i++){
 switch (i)
```

```
{
 case 0:
  if (tokens[i].Contains('/'))
    registro.NumberOfSegments = int.Parse(tokens[i].Split('/').Last());
  else
   registro.NumberOfSegments = 1;
  break;

 case 1:
   registro.NumberOfSignals = int.Parse(tokens[i]);
   break;

  case 2:
   registro.SamplingFrequencyPerSignal=
double.Parse(tokens[i].Split('/').First());

registro.CounterFrequency = registro.SamplingFrequencyPerSignal;

if (tokens[i].Contains('/'))
{
 registro.CounterFrequency = double.Parse(tokens[i].Split('/', '(')[1]);
   if (tokens[i].Contains('('))
  {
     registro.BaseCounter = double.Parse(tokens[i].Split('(', ')')[1]);
  }
}
  break;


  case 3:
   registro.NumberOfSamplesPerSignal = int.Parse(tokens[i]);
   break;


  case 4:
   DateTime dt;
   if             (!DateTime.TryParseExact(tokens[i],             "HH:mm:ss",
CultureInfo.InvariantCulture, DateTimeStyles.None, out dt))
                            if (!DateTime.TryParseExact(tokens[i], "hh:mm:ss",
CultureInfo.InvariantCulture, DateTimeStyles.None, out dt))
```

```
                                if          (!DateTime.TryParseExact(tokens[i],
"H:m:s", CultureInfo.InvariantCulture, DateTimeStyles.None, out dt))
                                    if      (!DateTime.TryParseExact(tokens[i],
"h:m:s", CultureInfo.InvariantCulture, DateTimeStyles.None, out dt))
                                        dt = new DateTime();

                        registro.StartDateTime = new DateTime(
                              1,
                              1,
                              1,
                              dt.Hour,
                              dt.Minute,
                              dt.Second
                              );
                        break;




case 5:
 if             (!DateTime.TryParseExact(tokens[i],              "dd/MM/yyyy",
CultureInfo.InvariantCulture, DateTimeStyles.None, out dt))
                            if (!DateTime.TryParseExact(tokens[i], "dd/MM/yy",
CultureInfo.InvariantCulture, DateTimeStyles.None, out dt))
                                if          (!DateTime.TryParseExact(tokens[i],
"d/M/y", CultureInfo.InvariantCulture, DateTimeStyles.None, out dt))
                                    dt = new DateTime();
                        registro.StartDateTime = new DateTime(
                            dt.Year,
                            dt.Month,
                            dt.Day,
                            registro.StartDateTime.Value.Hour,
                            registro.StartDateTime.Value.Minute,
                            registro.StartDateTime.Value.Second
                            );
                        break;
                    default:
                        break;
                }
}
```

Por ejemplo, el primer token además de contener el nombre del registro puede contener el número de segmentos en caso de ser un registro multisegmentos.

El segundo token contiene el número de señales del registro, y si sucesivamente con el resto de los tokens si existen. Tras recorrer los tokens de la línea de registros se devolvería el objeto registro con sus propiedades correspondientes establecidas.

3.4 Pruebas de la nueva librería WFDB

Para realizar las pruebas de los métodos que proporciona la clase *ManagerWFDB,* se utilizará los registros de las bases de datos que nos proporciona el repositorio de PhysioBank y comparar los resultados obtenido con la aplicación PhysioBank ATM.

PhysioBank es un archivo grande y creciente de datos fisiológicos [10] que contiene más de 90000 registros, o más de 4 terabytes de señales fisiológicas digitalizadas, organizadas en más de 80 bases de datos.

PhysioBank's Automated Teller Machine es una instalación de autoservicio para explorar PhysioBank usando su navegador web. Actualmente, su caja de herramientas incluye software que puede mostrar formas de onda anotadas, series de tiempo e histogramas de intervalos RR, convertir archivos de señales WFDB a archivos de texto, CSV, EDF [11] .

Para las pruebas su utilizará un registro de las siguientes bases de datos:

1. ECG-ID https://physionet.org/physiobank/database/ecgiddb/
2. SLEEP-EDF https://physionet.org/physiobank/database/sleep-edfx/

3.4.1 Prueba ECG-ID

La base de datos ECG-ID contiene 310 registros de ECG, obtenidos de 90 personas. Cada registro contiene [12]:

- Cable de ECG I, grabado durante 20 segundos, digitalizado a 500 Hz con resolución de 12 bits en un rango nominal de ± 10 mV.
- 10 pulsaciones anotadas (anotaciones de picos de onda R y T no auditados de un detector automático)
- Información (en el archivo .hea para el registro) que contiene edad, sexo y fecha de registro.

Cada registro incluye 2 señales:

1. Señal 0: ECG I (señal sin procesar con ruido)
2. Señal 1: ECG I filtrado (señal filtrada sin ruido)

Para esta prueba, se realizará la lectura de los ficheros del primer registro (*rec_1*) de la primera persona.

3.4.1.1 *Lectura del fichero de cabecera*

Se ha realizado la prueba utilizando el método de exportación del fichero de cabecera en JSON. Comparando el JSON obtenido con el resultado obtenido en aplicación PhysioBank ATM.

```
[TestMethod]
 public void ReadHeader_ECGID()
 {
     string jsonPath;
     jsonPath = ManagerWFDB.ExportRecordJSON("rec_1");
     Assert.IsTrue(File.Exists(jsonPath));
 }
```

Como se puede observar en las siguientes imágenes el resultado obtenido por la aplicación son iguales que los obtenidos por el método `ExportRecordJSON`. Además, el JSON obtenido tiene más información del registro.

PhysioBank ATM	ManagerWFDB
Record ecgiddb/Person_01/rec_1	`"Name": "rec_1",`

Notes
=====
Age: 25
Sex: male
ECG date: 07.12.2004
=====
	`"Info": [`
	`"",`
	`"Age: 25",`
	`"Sex: male",`
	`"ECG date: 07.12.2004"`
	`],`

Starting time: not specified	`"StartDateTime": null,`
Length: 0:20.000 (10000 sample intervals)	`"NumberOfSamplesPerSignal": 10000,` `"DurationInSeconds": 20.0,`
Sampling frequency: 500 Hz	`"SamplingFrequencyPerSignal": 500.0,`
2 signals	`"NumberOfSignals": 2,`
Group 0, Signal 0:	`"Group": 0,` `"Number": 0,`
File: rec_1.dat	`"FileName": "rec_1.dat"`
Description: ECG I	`"Description": "ECG I",`
Gain: 200 adu/mV	`"Gain": 200.0,`
Initial value: -17	`"InitValue": -17,`
Storage format: 16	`"Format": 16,`
I/O: can be unbuffered	
ADC resolution: 12 bits	`"AdcResolution": 12,`
ADC zero: 0	`"AdcZero": 0,`
Baseline: 0	`"Baseline": 0,`
Checksum: 17532	`"CheckSum": 17532,`
Group 0, Signal 1:	`"Group": 0,` `"Number": 1,`
File: rec_1.dat	`"FileName": "rec_1.dat"`
Description: ECG I filtered	`"Description": "ECG I filtered",`
Gain: 200 adu/mV	`"Gain": 200.0,` `"Units": "mV",`

Initial value: -23	"InitValue": -23,
Storage format: 16	"Format": 16,
I/O: can be unbuffered	
ADC resolution: 12 bits	"AdcResolution": 12,
ADC zero: 0	"AdcZero": 0,
Baseline: 0	"Baseline": 0,
Checksum: 2004	"CheckSum": 2004,

Tabla 6 Comparación Lectura del fichero de cabecera de rec_1

Ilustración 3-1 Resultados de la lectura del fichero de cabecera de rec_1

3.4.1.2 *Lectura del fichero de señal*

Se ha realizado la prueba utilizando el método de exportación del fichero de señal en CSV. Se va a comparar las primeras 10 muestras y las últimas 10 del registro con el resultado obtenido con la aplicación web.

```
[TestMethod]
public void ReadSignals_ECGID()
{
    string csvPath;
    csvPath = ManagerWFDB.ExportSamplesCSV("rec_1");
    Assert.IsTrue(File.Exists(csvPath));
}
```

	A	B	C	D	E
1	Tiempo	Hora	Fecha	ECG I(mV)	ECG I filtered(mV)
2	00:00:00.000	0:00:00	01/01/0001	-0,085	-0,115
3	00:00:00.002	0:00:00	01/01/0001	-0,08	-0,115
4	00:00:00.004	0:00:00	01/01/0001	-0,07	-0,12
5	00:00:00.006	0:00:00	01/01/0001	-0,075	-0,12
6	00:00:00.008	0:00:00	01/01/0001	-0,095	-0,12
7	00:00:00.010	0:00:00	01/01/0001	-0,09	-0,12
8	00:00:00.012	0:00:00	01/01/0001	-0,1	-0,12
9	00:00:00.014	0:00:00	01/01/0001	-0,1	-0,115
10	00:00:00.016	0:00:00	01/01/0001	-0,1	-0,115
11	00:00:00.018	0:00:00	01/01/0001	-0,085	-0,115

```
Elapsed time   ECG I iltered
hh:mm:ss.mmm     (mV)     (mV)
     0:00.000   -0.085   -0.115
     0:00.002   -0.080   -0.115
     0:00.004   -0.070   -0.120
     0:00.006   -0.075   -0.120
     0:00.008   -0.095   -0.120
     0:00.010   -0.090   -0.120
     0:00.012   -0.100   -0.120
     0:00.014   -0.100   -0.115
     0:00.016   -0.100   -0.115
     0:00.018   -0.085   -0.115
```

9992	00:00:19.980	0:00:19	01/01/0001	-0,105	-0,03
9993	00:00:19.982	0:00:19	01/01/0001	-0,105	-0,03
9994	00:00:19.984	0:00:19	01/01/0001	-0,095	-0,03
9995	00:00:19.986	0:00:19	01/01/0001	-0,09	-0,03
9996	00:00:19.988	0:00:19	01/01/0001	-0,07	-0,03
9997	00:00:19.990	0:00:19	01/01/0001	-0,07	-0,03
9998	00:00:19.992	0:00:19	01/01/0001	-0,055	-0,035
9999	00:00:19.994	0:00:19	01/01/0001	-0,06	-0,035
10000	00:00:19.996	0:00:19	01/01/0001	-0,065	-0,035
10001	00:00:19.998	0:00:19	01/01/0001	-0,08	-0,035
10002					

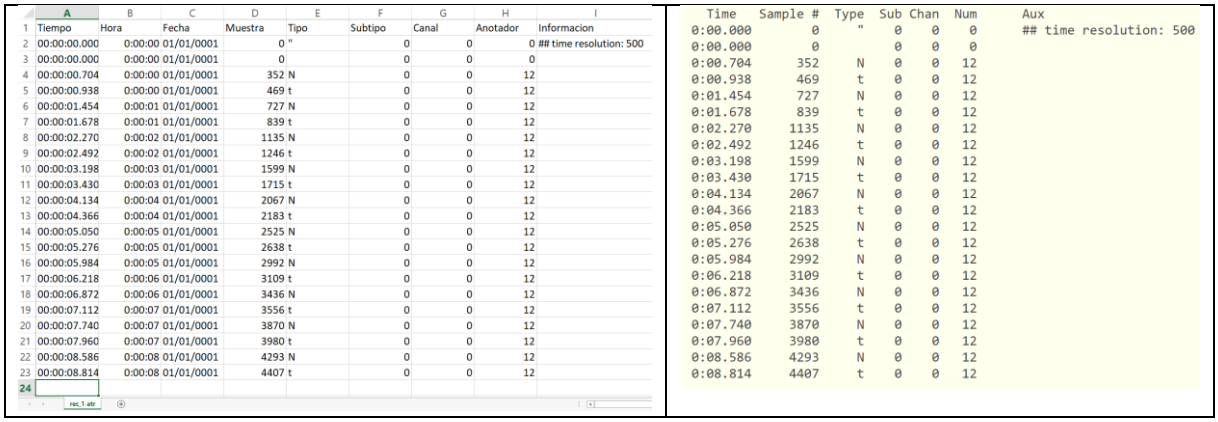
```
 0:19.980   -0.105   -0.030
 0:19.982   -0.095   -0.030
 0:19.984   -0.090   -0.030
 0:19.986   -0.070   -0.030
 0:19.988   -0.070   -0.030
 0:19.990   -0.055   -0.035
 0:19.992   -0.060   -0.035
 0:19.994   -0.065   -0.035
 0:19.996   -0.080   -0.035
 0:19.998   -0.080   -0.035
```

Como se puede apreciar los valores de las muestras y el tiempo en la que ha sido capturado son iguales. Cabe comentar que en el csv obtenido del método tiene una columna con el año 01/01/0001 y la hora 00:00 debido a que no está establecido la fecha de inicio ni la hora de inicio del registro en el fichero de cabecera.

3.4.1.3 *Lectura del fichero de anotaciones en formato MIT*

Se ha realizado la prueba utilizando el método de exportación del fichero de anotación (rec_1.atr) en CSV. Se va a comparar las anotaciones del csv con las anotaciones obtenido de la aplicación web.

```
[TestMethod]
public void ReadAnnotation_ECGID()
{
    string csvPath;
    csvPath = ManagerWFDB.ExportAnnotationsCSV("rec_1","atr",true);
    Assert.IsTrue(File.Exists(csvPath));
}
```



	A	B	C	D	E	F	G	H	I
1	Tiempo	Hora	Fecha	Muestra	Tipo	Subtipo	Canal	Anotador	Informacion
2	00:00:00.000	0:00:00	01/01/0001	0	"	0	0	0	## time resolution: 500
3	00:00:00.000	0:00:00	01/01/0001	0		0	0	0	
4	00:00:00.704	0:00:00	01/01/0001	352	N	0	0	12	
5	00:00:00.938	0:00:00	01/01/0001	469	t	0	0	12	
6	00:00:01.454	0:00:01	01/01/0001	727	N	0	0	12	
7	00:00:01.678	0:00:01	01/01/0001	839	t	0	0	12	
8	00:00:02.270	0:00:02	01/01/0001	1135	N	0	0	12	
9	00:00:02.492	0:00:02	01/01/0001	1246	t	0	0	12	
10	00:00:03.198	0:00:03	01/01/0001	1599	N	0	0	12	
11	00:00:03.430	0:00:03	01/01/0001	1715	t	0	0	12	
12	00:00:04.134	0:00:04	01/01/0001	2067	N	0	0	12	
13	00:00:04.366	0:00:04	01/01/0001	2183	t	0	0	12	
14	00:00:05.050	0:00:05	01/01/0001	2525	N	0	0	12	
15	00:00:05.276	0:00:05	01/01/0001	2638	t	0	0	12	
16	00:00:05.984	0:00:05	01/01/0001	2992	N	0	0	12	
17	00:00:06.218	0:00:06	01/01/0001	3109	t	0	0	12	
18	00:00:06.872	0:00:06	01/01/0001	3436	N	0	0	12	
19	00:00:07.112	0:00:07	01/01/0001	3556	t	0	0	12	
20	00:00:07.740	0:00:07	01/01/0001	3870	N	0	0	12	
21	00:00:07.960	0:00:07	01/01/0001	3980	t	0	0	12	
22	00:00:08.586	0:00:08	01/01/0001	4293	N	0	0	12	
23	00:00:08.814	0:00:08	01/01/0001	4407	t	0	0	12	
24									

Time	Sample #	Type	Sub	Chan	Num	Aux
0:00.000	0	"	0	0	0	## time resolution: 500
0:00.000	0		0	0	0	
0:00.704	352	N	0	0	12	
0:00.938	469	t	0	0	12	
0:01.454	727	N	0	0	12	
0:01.678	839	t	0	0	12	
0:02.270	1135	N	0	0	12	
0:02.492	1246	t	0	0	12	
0:03.198	1599	N	0	0	12	
0:03.430	1715	t	0	0	12	
0:04.134	2067	N	0	0	12	
0:04.366	2183	t	0	0	12	
0:05.050	2525	N	0	0	12	
0:05.276	2638	t	0	0	12	
0:05.984	2992	N	0	0	12	
0:06.218	3109	t	0	0	12	
0:06.872	3436	N	0	0	12	
0:07.112	3556	t	0	0	12	
0:07.740	3870	N	0	0	12	
0:07.960	3980	t	0	0	12	
0:08.586	4293	N	0	0	12	
0:08.814	4407	t	0	0	12	

Como se puede observar, los resultados de las anotaciones son iguales.

3.4.2 Prueba SLEEP-EDF

La base de datos sleep-edf contiene 197 grabaciones de polisomnografía de sueño de una noche entera, que contienen EEG (Electroencefalografía), EOG (electrooculografía), EMG (electromiografía) del mentón y marcadores de eventos. [13]

Para esta prueba se realizará la lectura del fichero en formato EDF del registro *SC4001E0-PSG.edf*. No se realizará la prueba de las anotaciones en formato de EDF debido a que no está funcionalidad no está programada ya que la librería original tampoco tiene esta funcionalidad.

3.4.2.1 Lectura de los registros de encabezado del fichero edf

Se ha realizado la prueba utilizando el método de exportación del fichero de cabecera en JSON (poniendo la variable de entrada isEDF a true). Comparando el JSON obtenido con el resultado obtenido en aplicación PhysioBank ATM.

```
    [TestMethod]
        public void ReadHeader_SLEEPEDF()
        {
            string jsonPath;
            jsonPath = ManagerWFDB.ExportRecordJSON("SC4001E0-PSG", isEDF: true);
            Assert.IsTrue(File.Exists(jsonPath));
        }
```

PhysioBank ATM	ManagerWFDB
Record sleep-edfx/sleep-cassette/SC4001E0-PSG.edf	"Name": "rec_1",
	"Info": ["Version EDF:0", "IdPatient:X F X Female_33yr", "IdRecording:Startdate 24-APR-1989 X X X", "Reserved:",

	"Transducer:Ag-AgCl electrodes;Ag-AgCl electrodes;Ag-AgCl electrodes;Oral-nasal thermistors;Ag-AgCl electrodes;Rectal thermistor;Marker button", "PhysicalMin:-192;-197;-1009;-2048;-5;34;-2047", "PhysicalMax:192;196;1009;2047;5;40;2048", "DigitallMin:-2048;-2048;-2048;-2048;-2500;-2849;-2047", "DigitalMax:2047;2047;2047;2047;2500;2731;2048", "Prefiltering:HP:0.5Hz LP:100Hz [enhanced cassette BW];HP:0.5Hz LP:100Hz [enhanced cassette BW];HP:0.5Hz LP:100Hz [enhanced cassette BW];HP:0.03Hz LP:0.9Hz;HP:16Hz Rectification LP:0.7Hz;;Hold during 2 seconds", "ReservedSignal:;;;;;;"],
Starting time: [16:13:00.000 24/04/1989]	"StartDateTime": "1989-04-24T16:13:00",
Length: 22:05:00.000 (2650 sample intervals)	"NumberOfFrames": 2650, "DurationInSeconds": 79500.0,
Sampling frequency: 0.0333333333333 Hz	"FrameFrequency": 0.033333333333333333,
7 signals	"NumberOfSignals": 7,
Group 0, Signal 0:	"Group": 0, "Number": 0,
File: SC4001E0-PSG.edf	"FileName": "SC4001E0-PSG.edf",
Description: EEG Fpz-Cz	"Description": "EEG Fpz-Cz",
Gain: 10.6640625 adu/uV	"Gain": 10.6640625, "Units": "uV",
Initial value: 0	"InitValue": 0,
Storage format: 16 (3000 samples per frame)	"Format": 16, "SamplesPerFrame": 3000,
I/O: can be unbuffered	
ADC resolution: 12 bits	"AdcResolution": 12,
ADC zero: 0	"AdcZero": 0,
Baseline: -1	"Baseline": -1,
Checksum: 0	"CheckSum": 0,
Group 0, Signal 1:	"Group": 0, "Number": 0,
File: SC4001E0-PSG.edf	"FileName": "SC4001E0-PSG.edf",
Description: EEG Pz-Oz	"Description": "EEG Pz-Oz",
Gain: 10.4198473282 adu/uV	"Gain": 10.6640625, "Units": "uV",
Initial value: 0	"InitValue": 0,
Storage format: 16 (3000 samples per frame)	"Format": 16, "SamplesPerFrame": 3000,
I/O: can be unbuffered	
ADC resolution: 12 bits	"AdcResolution": 12,
ADC zero: 0	"AdcZero": 0,
Baseline: 5	"Baseline": 5,

Checksum: 0	`"CheckSum": 0,`
Group 0, Signal 2:	`"Group": 0,` `"Number": 0,`
File: SC4001E0-PSG.edf	`"FileName": "SC4001E0-PSG.edf",`
Description: EOG horizontal	`"Description": "EOG horizontal",`
Gain: 2.02923686819 adu/uV	`"Gain": 2.0292368681863229,` `"Units": "uV",`
Initial value: 0	`"InitValue": 0,`
Storage format: 16 (3000 samples per frame)	`"Format": 16,` `"SamplesPerFrame": 3000,`
I/O: can be unbuffered	
ADC resolution: 12 bits	`"AdcResolution": 12,`
ADC zero: 0	`"AdcZero": 0,`
Baseline: 0	`"Baseline": 0,`
Checksum: 0	`"CheckSum": 0,`
Group 0, Signal 3:	`"Group": 0,` `"Number": 0,`
File: SC4001E0-PSG.edf	`"FileName": "SC4001E0-PSG.edf",`
Description: Resp oro-nasal	`"Description": "Resp oro-nasal",`
Gain: 1 adu/	`"Gain": 1.0,` `"Units": "",`
Initial value: 0	`"InitValue": 0,`
Storage format: 16 (30 samples per frame)	`"Format": 16,` `"SamplesPerFrame": 30,`
I/O: can be unbuffered	
ADC resolution: 12 bits	`"AdcResolution": 12,`
ADC zero: 0	`"AdcZero": 0,`
Baseline: 0	`"Baseline": 0,`
Checksum: 0	`"CheckSum": 0,`
Group 0, Signal 4:	`"Group": 0,` `"Number": 0,`
File: SC4001E0-PSG.edf	`"FileName": "SC4001E0-PSG.edf",`
Description: EMG submental	`"Description": "EMG submental",`
Gain: 500 adu/uV	`"Gain": 500.0,` `"Units": "uV",`
Initial value: 0	`"InitValue": 0,`
Storage format: 16 (30 samples per frame)	`"Format": 16,` `"SamplesPerFrame": 30,`
I/O: can be unbuffered	
ADC resolution: 13 bits	`"AdcResolution": 13,`
ADC zero: 0	`"AdcZero": 0,`
Baseline: 0	`"Baseline": 0,`
Checksum: 0	`"CheckSum": 0,`
Group 0, Signal 5:	`"Group": 0,` `"Number": 0,`
File: SC4001E0-PSG.edf	`"FileName": "SC4001E0-PSG.edf",`
Description: Temp rectal	`"Description": "Temp rectal",`
Gain: 930 adu/DegC	`"Gain": 930.0,` `"Units": "DegC",`
Initial value: -58	`"InitValue": -58,`

Storage format: 16 (30 samples per frame)	"Format": 16, "SamplesPerFrame": 30,
I/O: can be unbuffered	
ADC resolution: 13 bits	"AdcResolution": 13,
ADC zero: -58	"AdcZero": -58,
Baseline: -34469	"Baseline": -34469,
Checksum: 0	"CheckSum": 0,
Group 0, Signal 6:	"Group": 0, "Number": 0,
File: SC4001E0-PSG.edf	"FileName": "SC4001E0-PSG.edf",
Description: Event marker	"Description": "Event marker",
Gain: 1 adu/	"Gain": 1.0, "Units": "",
Initial value: 1	"InitValue": 0,
Storage format: 16 (30 samples per frame)	"Format": 16, "SamplesPerFrame": 30,
I/O: can be unbuffered	
ADC resolution: 12 bits	"AdcResolution": 12,
ADC zero: 1	"AdcZero": 1,
Baseline: 0	"Baseline": 0,
Checksum: 0	"CheckSum": 0,

Tabla 7 Resultado lectura de los registros de encabezado del fichero EDF

Como se puede observar, los atributos de las 7 señales obtenidas del fichero edf son iguales, y además el ManagerWFDB conserva toda la información adicional del registro de encabezado, a diferencia de PhysioBank ATM que ignora esta información.

También cabe señalar que para las señales 03 y 06. No tienen especificadas las unidades de medida y tanto la aplicación web como el ManagerWFDB, ponen una cadena vacía en las unidades. Esto se debe porque en el formato EDF no hay unidades de medidas por defecto, a diferencia de los ficheros estándar WFDB, en el cual me unidad de medida por defecto es mV.

3.4.2.2 Lectura de los registro de datos del fichero edf

Se ha realizado la prueba utilizando el método de exportación del fichero de señal en CSV. Debido a que la grabación es de 22 horas, lo cual son unas 7950000 muestras, la aplicación web solo puede obtener un máximo de 100000 muestra, por lo que se va a comparar las primeras 10 muestras.

ManagerWFDB

Tiempo	Hora	Fecha	EEG Fpz-Cz(uV)	EEG Pz-Oz(uV)	EOG horizontal(uV)	Resp oro-nasal()	EMG submental(uV)	Temp rectal(DegC)	Event marker()
00:00:00.000	16:13:00	24/04/1989	5,064	-2,495	16,262	-482	3,552	37,206	920
00:00:00.010	16:13:00	24/04/1989	-2,532	1,44	15,769	-482	3,552	37,206	920
00:00:00.020	16:13:00	24/04/1989	1,407	-4,127	9,363	-482	3,552	37,206	920
00:00:00.030	16:13:00	24/04/1989	-2,344	-2,687	1,478	-482	3,552	37,206	920
00:00:00.040	16:13:00	24/04/1989	-5,158	-0,768	7,392	-482	3,552	37,206	920
00:00:00.050	16:13:00	24/04/1989	-7,408	-1,44	2,464	-482	3,552	37,206	920
00:00:00.060	16:13:00	24/04/1989	-8,252	-0,96	0	-482	3,552	37,206	920
00:00:00.070	16:13:00	24/04/1989	-6,845	1,823	-0,986	-482	3,552	37,206	920
00:00:00.080	16:13:00	24/04/1989	-3,657	-0,864	0,986	-482	3,552	37,206	920
00:00:00.090	16:13:00	24/04/1989	-0,938	-0,864	4,928	-482	3,552	37,206	920

PhysioBank ATM

Time	Date	Fpz-Cz	G Pz-Oz	izontal	o-nasal	bmental	rectal	marker	
(hh:mm:ss.mmm	dd/mm/yyyy)	(uV)	(uV)	(uV)	()	(uV)	(DegC)	()	
[16:13:00.000	24/04/1989]	5.064	-2.495	16.262	-482.000		3.552	37.206	920.000
[16:13:00.010	24/04/1989]	-2.532	1.440	15.769	-482.000		3.552	37.206	920.000
[16:13:00.020	24/04/1989]	1.407	-4.127	9.363	-482.000		3.552	37.206	920.000
[16:13:00.030	24/04/1989]	-2.344	-2.687	1.478	-482.000		3.552	37.206	920.000
[16:13:00.040	24/04/1989]	-5.158	-0.768	7.392	-482.000		3.552	37.206	920.000
[16:13:00.050	24/04/1989]	-7.408	-1.440	2.464	-482.000		3.552	37.206	920.000
[16:13:00.060	24/04/1989]	-8.252	-0.960	0.000	-482.000		3.552	37.206	920.000
[16:13:00.070	24/04/1989]	-6.845	1.823	-0.986	-482.000		3.552	37.206	920.000
[16:13:00.080	24/04/1989]	-3.657	-0.864	0.986	-482.000		3.552	37.206	920.000
[16:13:00.090	24/04/1989]	-0.938	-0.864	4.928	-482.000		3.552	37.206	920.000

Los resultados de las muestras son iguales, la diferencia es que la aplicación web PhysioBank no ha sido capaz de leer todas las muestras a diferencia de la librería. La cual ha exportado un csv de 500Mb en un tiempo razonable de aproximadamente 1 minutos.

4 Problemas encontrados

Los problemas encontrados y las soluciones para el desarrollo del proyecto fueron los siguientes:

- **Dificultad en entender código de la librería WFDB**: En un principio se intentó analizar y comprender todo el código de las funciones en C que se quería traducir a C#, pero debido a que dicho código era muy extensos (muchas funciones que llaman a otras funciones) y no tenía buenas prácticas (nombres de variables no significativas, falta de comentarios, comentarios poco explicativos).
  Se optó como solución en analizar únicamente las entradas y las salidas de las funciones, analizar el qué es lo que hace y no el cómo lo hace.

- **Dificultad en cambiar de paradigma de programación:** Intentar cambiar el código del paradigma imperativo al paradigma orientado en objetos resulta complejo debido cambio de mentalidad a la hora de programar. En el primero no se tiene clases, mi encapsulamiento y utiliza mucho variables globales a lo largo de todo el proceso.
  Como solución se eligió tomar como base las clases ya creadas en la librería WfdbCsharpWrapper. La cual ya habia implementado este cambio de paradigma de la librería WFDB.
  En cuanto la implementación de los métodos, como en el problema anterior, se optó por replicar lo que hacen las funciones y no el como. Adaptándolo a la programación orientada a objetos e implementándolo dentro de la clase correspondiente.

- **Probar resultados de las funciones de la librería WFDB:** Dado a que los resultados de las funciones no se podrían probar directamente, ya que necesitaría depurar paso a paso el código en C. Lo cual demandaría mucho tiempo en buscar un compilador, instalador, configurar la librería en C con riesgo a que no funcione.
  Se decidió analizar el resultado final realizando las pruebas a través de aplicación web PhysioBank ATM.

5 Presupuesto

Para dar un valor económico al proyecto realizado, se calculará el presupuesto que se necesitaría para llevar a cabo el proyecto.

5.1 Presupuesto de ejecución material

5.1.1 Coste de material

Para este proyecto se ha utilizado un ordenador de gama media con Visual Studio Community instalado y con conexión a internet.

Material	**Precio**	**Duración**	**Uso**	**Total**
Ordenador portátil	550 €	3 años	3 meses	45,83 €
Internet	30 €	1 mes	3 meses	90,00 €
Coste total de material				135,83 €

5.1.2 Coste por tiempo de trabajo

El tiempo de trabajo estimado total es de 480 horas (3 meses) será realizado por un ingeniero informático profesional. La media salarial de un ingeniero informático es de 26.451€ brutos anuales [14]. Por lo que el salario por hora del ingeniero informático será 14€/hora aproximadamente.

Función	**Horas**	**Precio/hora**	**Total**
Análisis	280	14	3.920 €
Desarrollo	160	14	2.240 €
Pruebas	40	14	560 €
Coste total por tiempo de trabajo			6.720 €

5.1.3 Coste total de ejecución material

El coste total de ejecución es la suma del coste de material más el coste por tiempo de trabajo.

Coste total de material	135,83 €
Coste total por tiempo de trabajo	6.720 €
Coste total de ejecución material	6.855,83 €

5.2 Gastos generales y beneficio industrial

Se entiende como gastos generales a los costes de instalaciones u otros gastos adiciones para que el trabajador desempeñe su laboral. Para su cálculo se aplicará el 30% del coste total de ejecución material.

Coste total de ejecución material	6.855,83 €
Porcentaje	30%
Gastos generales y beneficio industrial	2.056,75 €

5.3 Presupuesto de ejecución por contrata

Para calcular el presupuesto de ejecución por contrata se suma el coste total por ejecución material y los gastos generales y beneficio industrial.

Coste total de ejecución material	6.855,83 €
Gastos generales y beneficio industrial	2.056,75 €
Presupuesto de ejecución por contrata	8.912,58 €

5.4 Presupuesto total

El presupuesto total será el presupuesto de ejecución por contrata aplicando el 21% del I.V.A.

Presupuesto de ejecución por contrata	8.912,58 €
21% de I.V.A	1.871,64 €
Presupuesto total	**10.784.22 €**

6 Mejoras futuras

Dado que este proyecto se basa en la creación inicial de una nueva librería, esta librería se le podría agregar tantas funcionalidades como se quiera o necesite. Entre ellas se propone las siguientes funcionalidades, algunas de la librería WFDB nativa en C:

- **Lectura de las señales en todos los formatos:** Debido a que en el repositorio utilizado en este proyecto, PhysioBank, no se ha encontrado ningún registro con las datos o muestras en formato de 8 bit, 160 bit, 311 bit ya que estos formatos no son muy utilizados actualmente. No se han programado estos formatos, por lo que estaría pendiente completar esta funcionalidad de la librería original de WFDB.

- **Lectura de los registros multisegmentos:** Al igual que la mejora anterior, no se ha encontrado ningún ejemplo de registro multisegmentos en el repositorio PhysionBank. Por lo que estaría pendiente la lectura de la línea de segmento del fichero de cabecera y el tratamiento del registro multisegmentos. Actualmente la librería solo obtiene el número de segmentos del registros y en caso de ser este mayor que 1, lanza la excepción de que no está implementado el tratamiento de registros multisegmentos.

- **Lectura del fichero de calibración:** Otra funcionalidad de la librería original WFDB y la cual tampoco se ha completado, ya que este fichero es meramente informativo y se encuentra codificado en ASCII y se puede leer directamente. Esta funcionalidad se podría implementar creando un método de lectura del fichero de calibración en la clase Calibration y agregándola como Propiedad de la Clase Record.

- **Lectura de las anotaciones en los ficheros en formato EDF o EDF +:** Aunque esta funcionalidad no está contemplado en la librería original WFDB, es importante ya que la gran mayoría de los registros en formato EDF posee anotaciones.

- **Convertir el fichero de señal de un formato a otro:** Pasar por ejemplo del formato 16 bit al formato 24 bit o al formato EDF. Es una de las funcionalidades que posee la librería nativa WFDB.

- **Convertir fichero CSV o JSON a fichero Wfdb en formato X:** Realizar el proceso inverso, con el csv o el json obtenido con la nueva librería, generar un fichero encabezado, señal, anotaciones y/o fichero en formato edf.

- **Mejorar el rendimiento de la lectura del fichero de señal o registro de datos:** La nueva librería realiza una lectura secuencial de las muestras, pero esta lectura podría ser mejorada si se hace en paralelo con concurrencia. Por ejemplo si el fichero de datos tiene *X* bytes, que se asigne *X*/*Y* bytes a cada *Y* hilos.

7 Conclusiones

7.1 Conclusiones del proyecto

El proyecto tenía que hacerse tarde o temprano, debido a que la librería hecha en C ha quedado desfasado ya que fue hecha en los 90 y ejecutarla hoy en día no aprovecharía los nuevos recursos de los ordenadores actuales. Además C es un lenguaje de bajo nivel, lo cual dificulta a los programadores entenderla y continuar añadiendo nuevas funcionalidades a la librería.

Se ha cubierto las funcionalidades más comunes y básicas de la librería como son las siguientes:

- Lectura de fichero de cabecera con excepción de las líneas de segmento (propuesto como futura mejora)
- Lectura del fichero de señal con la mayoría de los formatos comunes.
- Lectura completa del fichero de anotación.
- Lectura de fichero EDF y EDF + (sin anotaciones).
- Exportación de las muestras, anotaciones de las señales en formato CSV.
- Exportación de la información del registro en formato JSON.

Con estas funcionalidades se puede leer cualquier registro de cualquier base de datos del repositorio de PhysionBank y obtener los mismos resultados que su aplicación web. Las funcionalidades que no se han cubierto se debe a que son formatos antiguos y poco comunes, no hay ejemplos en el repositorio.

La nueva librería será base para futuros proyectos (añadiendo más funcionalidades, mejorando el rendimiento), lo cual permitirá algún día cercano, tener un software de análisis avanzados de señales fisiológicas como los que están en mercado.

Además servirá de mucho para programas estadísticos o máquinas de aprendizaje ya que gracias a que exporta la información de las muestras y anotaciones de las grabaciones en CSV y JSON, estos formatos son fáciles de analizar.

7.2 Conclusiones personales

Realizar este proyecto me ha enseñado mucho la importancia de tener buenas prácticas a la hora de programar y documentar. Tal como había comentado en el apartado de problemas encontrados, entender el código de una librería tan grande como la de WFDB, en cual ha estado actualizándose desde 1990 es complicado y más aún si hay malas prácticas dentro del código. Esa librería no está pensada para que otra persona la comprenda y la pueda modificar, sino para que sea únicamente utilizada por otras personas.

Asimismo, gracias a las pruebas unitarias se ganó mucho tiempo en probar los métodos ya que cualquier modificación que se hacía algún método no afectaba en nada a la prueba su prueba unitaria. Además, si se hacía algún cambio en parte del código y se querría probar que todo funcionaba correctamente, en vez de probarlo manualmente cada método, con un solo clic se podría comprobar que todos los métodos seguían funcionando correctamente pese a la modificación realizada. Por tanto, aunque parezca un trabajo demás desarrollar los métodos unitarios, este desarrollo a la larga ayuda mucho en las pruebas y mantenimiento o modificación del código.

También he aprendido a valorar las metodologías ágiles, ya que da buenos resultados ir alternando cíclicamente entre pequeñas fases de análisis, desarrollo y pruebas. A medida que se va analizando, se puede ir desarrollando y probando y así evitamos que se nos haga pesado estar mucho tiempo leyendo sin realizar nada.

Y en relación con el tema de la librería creada, me permitió comprender el funcionamiento de pruebas médicas de bioseñales. Los cuales gracias a los electrodos captura distintas señales en frecuencias de muestreo especificas durante un tiempo determinado. Si los dispositivos electrónicos detectan algún valor fuera de lo normal o de intereses, realizar una anotación de dicha muestra guardando el número de muestra la cual dio el valor de interés y con su etiqueta correspondiente.

Aunque aún queda algunos aspectos a mejorar para que la librería este completa, con el estado actual de la librería se puede leer la mayoría de los ficheros de bioseñales y puede ser aplicado a otros programas que necesiten de esta lectura.

Bibliografía

[1] P. M. Nieto, Desarrollo de una aplicación de escritorio para la visualización de bioseñales con WFDB, Universidad de Alcalá, 2018.

[2] «WfdbCsharpWrapper,» [En línea]. Available: https://github.com/BoutemineOualid/WfdbCsharpWrapper.

[3] «Physionet-About,» [En línea]. Available: https://alpha.physionet.org/about/.

[4] «WFDB Programmer's Guide: Record,» [En línea]. Available: https://physionet.org/physiotools/wpg/wpg_2.htm#Concepts-1.

[5] «PhysioBank Annotations,» [En línea]. Available: https://physionet.org/physiobank/annotations.shtml.

[6] «WFDB Applications Guide - Signal,» [En línea]. Available: https://physionet.org/physiotools/wag/signal-5.htm.

[7] «WFDB Applications Guide: Annot,» [En línea]. Available: https://physionet.org/physiotools/wag/annot-5.htm.

[8] «WFDB Applications Guide: Header,» [En línea]. Available: https://physionet.org/physiotools/wag/header-5.htm.

[9] «European Data Format,» [En línea]. Available: https://www.edfplus.info/specs/edf.html.

[10] «PhysioBank Databases,» [En línea]. Available: https://physionet.org/physiobank/database/.

[11] «PhysioBank ATM,» [En línea]. Available: https://physionet.org/cgi-bin/atm/ATM.

[12] «ECG-ID DATABASE,» [En línea]. Available: https://physionet.org/physiobank/database/ecgiddb/.

[13] «Sleep-EDF Database,» [En línea]. Available: https://physionet.org/content/sleep-edfx/1.0.0/.

[14] indeed, «Salarios para empleos de Ingeniero/a informático/a en España,» [En línea]. Available: https://www.indeed.es/salaries/Ingeniero/a-inform%C3%A1tico/a-Salaries.