SPOK microservices

v 1.0

Generated by Doxygen 1.9.1

Chapter 1

SPOK microservices Documentation

1.1 Descripción Breve

SPOK es una aplicación web accesible, de código abierto, diseñada para el procesamiento de señales médicas, específicamente electroencefalografía (EEG). Acepta archivos en formatos `.abf`, `.edf` y `.mat`, ofreciendo herramientas similares a EEGLAB de manera gratuita. SPOK aprovecha la potencia de clústeres de Kubernetes para manejar procesos de alto costo computacional.

1.2 Características

- **Procesamiento de señales EEG**: Incluye funcionalidades como Convolución, Transformada rápida de Fourier (FFT), y Transformada Wavelet Continua (CWT).
- **Formatos soportados**: `.abf`, `.edf`, `.mat`.
- **Escalabilidad**: Basado en microservicios desplegados en Kubernetes.
- **Accesibilidad**: Compatible con navegadores como Chromium, Safari y Mozilla.
- **Código abierto y gratuito**: No se almacena información identificable del usuario.

1.3 Tabla de Contenidos

1.4 Uso

1.4.1 1. Subir un archivo EEG

Desde la interfaz web, puedes subir un archivo en formato `.abf`, `.edf`, o `.mat` para comenzar el procesamiento.

1.4.2 2. Seleccionar un análisis

Elige entre las diferentes herramientas de procesamiento disponibles, como la Transformada rápida de Fourier (FFT) o la Transformada Wavelet Continua (CWT).

1.4.3 3. Visualizar los resultados

Una vez procesada la señal, podrás visualizar los resultados en forma de gráficos interactivos y descargar los datos procesados.

1.5 Arquitectura

SPOC está compuesto por múltiples microservicios, cada uno encargado de realizar un tipo específico de procesamiento de señal. Estos microservicios incluyen:

- **FFT Service**: Realiza la transformada rápida de Fourier.
- **CWT Service**: Implementa la transformada wavelet continua.
- **Signals Averge** y otros microservicios especializados.

Cada microservicio está desplegado en un clúster de Kubernetes, lo que permite una escalabilidad y manejo eficiente de las cargas computacionales. La comunicación entre los microservicios se realiza utilizando gRPC.

Chapter 2

Arquitectura

2.1 Vista lógica

2.2 Vista de procesos

2.3 Vista de desarrollo

2.4 Vista de despliegue

Chapter 3

Configuración de Kubernetes para la aplicación SPOK

3.1 Requisitos

Para configurar Kubernetes y desplegar la aplicación SPOK, se necesitan las siguientes herramientas:

- **kubectl**: Herramienta de línea de comandos para interactuar con el clúster de Kubernetes. Para instalar kubectl, sigue las instrucciones en la `documentación oficial`.
- **istioctl**: Comando de línea para gestionar y configurar Istio. Sigue las instrucciones en la `documentación de instalación de Istio` para instalarlo.
- **Helm**: Administrador de paquetes para Kubernetes, que permite gestionar aplicaciones en el clúster. Puedes instalarlo siguiendo los pasos en la `documentación de Helm`.

3.2 Instalación de Istio en el clúster

Para habilitar Istio en el clúster, sigue estos pasos:

1. Instala Istio en el clúster usando el perfil `demo`:
   ```
   istioctl install --set profile=demo -y
   ```
2. Habilita la service mesh de Istio en el namespace `default` con el siguiente comando:
   ```
   kubectl label namespace default istio-injection=enabled --overwrite
   ```
3. Activa las opciones de observabilidad de Istio aplicando los complementos (addons) en el clúster. Esto se realiza en la carpeta donde se descargó Istio:
   ```
   kubectl apply -f samples/addons
   ```

3.3 Instalación de KEDA

Para habilitar el escalado automático con KEDA, sigue estos pasos usando Helm:

1. Agrega el repositorio de KEDA:
   ```
   helm repo add kedacore https://kedacore.github.io/charts
   ```
2. Actualiza el repositorio de Helm:
   ```
   helm repo update
   ```
3. Instala KEDA en el namespace `keda`:
   ```
   helm install keda kedacore/keda --namespace keda --create-namespace
   ```

Note

Para obtener más información sobre KEDA, consulta la `documentación oficial`.

3.4 Despliegue de la Aplicación

Para desplegar la aplicación SPOK en el clúster de Kubernetes, se utiliza **ArgoCD**. Sigue estos pasos para instalarlo:

1. Crea el namespace para ArgoCD:
   ```
   kubectl create ns argocd
   ```
2. Aplica la configuración de instalación de ArgoCD:
   ```
   kubectl apply -n argocd -f
    https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
   ```

ArgoCD permite la gestión de aplicaciones y el despliegue automatizado en el clúster de Kubernetes.

3.5 Configuración de Archivos

Para obtener y aplicar las configuraciones necesarias para Istio y ArgoCD, sigue estos pasos:

1. Clona el repositorio completo que contiene los archivos de configuración:
   ```
   git clone https://github.com/SPOC-PUJ/microservices.git
   ```
2. Aplica las configuraciones de Istio:
   ```
   kubectl apply -f microservices/istio-conf
   ```
3. Aplica las configuraciones de ArgoCD en el namespace `argocd`:
   ```
   kubectl apply -f microservices/Argo -n argocd
   ```

Estas configuraciones preparan el entorno para el despliegue y gestión de la aplicación SPOK en el clúster de Kubernetes.

Remarks

Si después deseas liberar espacio, puedes eliminar el repositorio clonado con el siguiente comando:
```
rm -rf microservices
```

Chapter 4

Microservicios

4.1 Wavelet transform

Microservicio CWT

4.2 Signals Average

Signals Average Microservice

4.3 Fourier trasnform

Fourier Transform Microservice

4.4 Inverser Fourier transform

Inverse Fourier Transform Microservice

4.5 Wavelet Descomposition

Fast Wavelet Transform

4.6 Moving averge filter

Moving Average Microservice

4.7 .ABF reader

ABF Reader

4.8 .Mat reader

MAT Reader

4.9 First difference

First Difference Microservice

4.10 Running Sum

Running Sum Microservice

Chapter 5

Module Index

5.1 Modules

Here is a list of all modules:

Chapter 6

Hierarchical Index

6.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 7

Class Index

7.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 8

Module Documentation

8.1 ABF Reader

This module implements a gRPC service to process ABF files.

Classes

- class ABFReaderServer.ABFServiceServicer

    A gRPC service class that handles ABF file processing.

Functions

- def ABFReaderServer.ABFServiceServicer.readAbf (self, request, context)

    Processes the ABF file received in the request and returns its data and sampling rate.
- def **ABFReaderServer.serve** ()

8.1.1 Detailed Description

This module implements a gRPC service to process ABF files.

It uses the pyABF library to read the file and extract the signal data and sample rate.

8.1.2 Function Documentation

8.1.2.1 readAbf()

```
def ABFReaderServer.ABFServiceServicer.readAbf (
              self,
              request,
              context )
```

Processes the ABF file received in the request and returns its data and sampling rate.

The method saves the received ABF file temporarily, processes it using pyABF, and extracts the signal data and sampling rate.

Parameters

request	The gRPC request containing the ABF file as bytes (file_content).
context	The gRPC context (used for error handling and metadata).

Returns

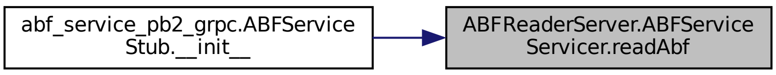
abf_service_pb2.ABFResponse The response containing the ABF data and sample rate.

Exceptions

IOError	Raised if an error occurs when reading or writing the file.

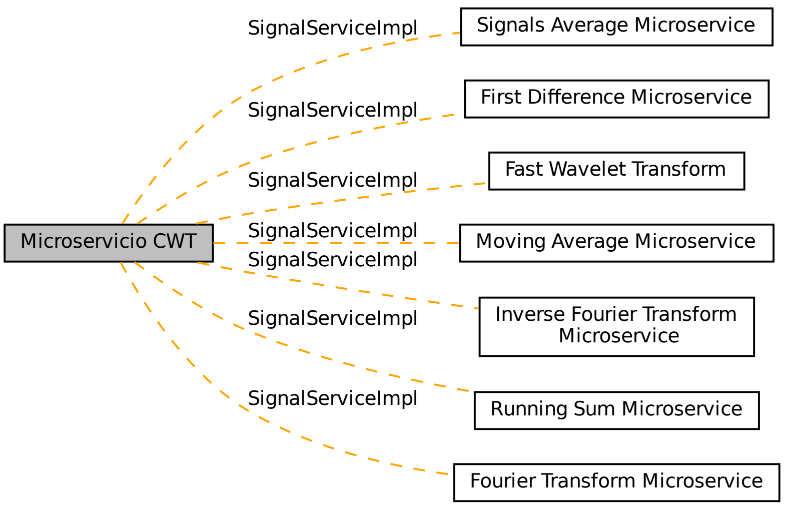
Reimplemented from abf_service_pb2_grpc.ABFServiceServicer.

Here is the caller graph for this function:


abf_service_pb2_grpc.ABFService Stub.__init__
ABFReaderServer.ABFService Servicer.readAbf

8.2 Microservicio CWT

Collaboration diagram for Microservicio CWT:


Microservicio CWT
SignalServiceImpl
Signals Average Microservice
First Difference Microservice
Fast Wavelet Transform
Moving Average Microservice
Inverse Fourier Transform Microservice
Running Sum Microservice
Fourier Transform Microservice

Classes

- class SignalServiceImpl

  Clase que implementa el servicio de CWT.

Functions

- Status SignalServiceImpl::ComputeCWT (ServerContext ∗context, const CWTRequest ∗request, CWTResponse ∗response) override

  Computes the Continuous Wavelet Transform (CWT) for a given signal.
- std::vector< double > SignalServiceImpl::GenerateLogScales (double start, double end, int numScales)

  Genera escalas logarítmicas.
- std::vector< Eigen::VectorXcd > SignalServiceImpl::CWTEigen (const Eigen::VectorXcd &signal, const std↩ ::vector< double > &scales)

  Calcula la CWT utilizando el Wavelet de Morlet.
- Eigen::VectorXcd SignalServiceImpl::MorletWavelet (int N, double scale, double f0=1.0, double fb=1.0)

  Genera el Wavelet de Morlet.
- Eigen::VectorXcd SignalServiceImpl::FFTconvolveEigen (const Eigen::VectorXcd &x, const Eigen::VectorXcd &h, bool shift)

  Realiza la convolución en el dominio de Fourier.
- Eigen::VectorXcd SignalServiceImpl::ZeroPadGivenSize (const Eigen::VectorXcd &a, int m)

  Zero-pads a vector to a specified size.
- void **RunServer** ()
- int **main** ()

8.2.1 Detailed Description

Este grupo contiene la implementación del microservicio para calcular la Transformada Wavelet Continua (CWT) utilizando gRPC.

8.2.2 Function Documentation

8.2.2.1 ComputeCWT()

```
Status SignalServiceImpl::ComputeCWT (
            ServerContext * context,
            const CWTRequest * request,
            CWTResponse * response )  [inline], [override]
```

Computes the Continuous Wavelet Transform (CWT) for a given signal.

This function receives a signal in the form of complex numbers, computes the CWT using a set of specified scales, and returns the coefficients.

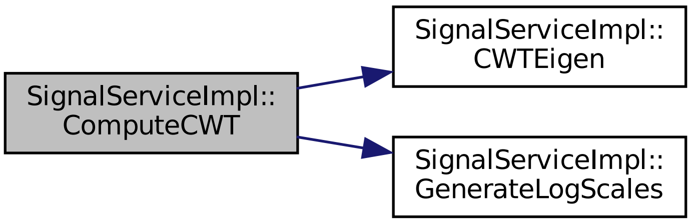
Parameters

context	Server context for handling requests.
request	The request containing the signal and scale parameters.
response	The response object where the CWT coefficients will be stored.

Returns

gRPC status indicating success or failure of the operation.

Here is the call graph for this function:



8.2.2.2 CWTEigen()

```
std::vector<Eigen::VectorXcd> SignalServiceImpl::CWTEigen (
            const Eigen::VectorXcd & signal,
            const std::vector< double > & scales )  [inline], [private]
```

Calcula la CWT utilizando el Wavelet de Morlet.

Esta función calcula la Transformada Wavelet Continua para la señal proporcionada utilizando un conjunto de escalas.

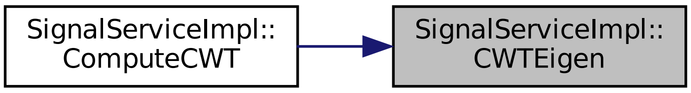
Parameters

signal	La señal de entrada representada como un vector de números complejos.
scales	Un vector de escalas a utilizar para el cálculo de la CWT.

Returns

Un vector de vectores complejos que representan los coeficientes de la CWT.

Here is the caller graph for this function:



8.2.2.3 FFTconvolveEigen()

```
Eigen::VectorXcd SignalServiceImpl::FFTconvolveEigen (
             const Eigen::VectorXcd & x,
             const Eigen::VectorXcd & h,
             bool shift )  [inline], [private]
```

Realiza la convolución en el dominio de Fourier.

Esta función utiliza la transformada de Fourier para realizar la convolución de dos señales en el dominio de Fourier.

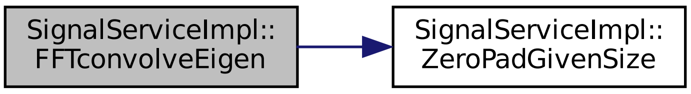
Parameters

x	La señal de entrada.
h	El kernel de convolución.
shift	Indica si se debe realizar un desplazamiento en el dominio de la frecuencia.

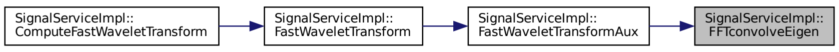
Returns

Resultado de la convolución.

Here is the call graph for this function:



Here is the caller graph for this function:



8.2.2.4 GenerateLogScales()

```
std::vector<double> SignalServiceImpl::GenerateLogScales (
            double start,
            double end,
            int numScales )  [inline], [private]
```

Genera escalas logarítmicas.

Esta función genera una serie de escalas logarítmicas entre un rango especificado.

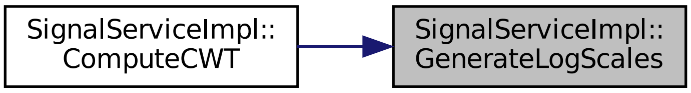
Parameters

start	Escala inicial.
end	Escala final.
numScales	Número de escalas a generar.

Returns

Vector de escalas generadas.

Here is the caller graph for this function:


SignalServiceImpl:: ComputeCWT
SignalServiceImpl:: GenerateLogScales

8.2.2.5 MorletWavelet()

```
Eigen::VectorXcd SignalServiceImpl::MorletWavelet (
             int N,
             double scale,
             double f0 = 1.0,
             double fb = 1.0 )  [inline], [private]
```

Genera el Wavelet de Morlet.

Esta función genera un vector que representa el Wavelet de Morlet para un tamaño dado y una escala especificada.

Parameters

N	Tamaño del wavelet.
scale	Escala del wavelet.
f0	Frecuencia central (por defecto 1.0).
fb	Ancho de banda (por defecto 1.0).

Returns

Vector de números complejos que representa el Wavelet de Morlet.

8.2.2.6 ZeroPadGivenSize()

```
Eigen::VectorXcd SignalServiceImpl::ZeroPadGivenSize (
             const Eigen::VectorXcd & a,
             int m )  [inline], [private]
```

Zero-pads a vector to a specified size.

Esta función rellena el vector de entrada con ceros para asegurar que su tamaño coincida con el tamaño objetivo especificado. Si el vector de entrada ya tiene el tamaño objetivo, se devuelve sin cambios.

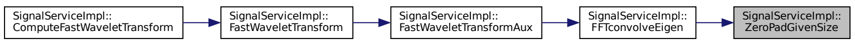
Parameters

a	El vector de entrada a rellenar.
m	El tamaño objetivo para el vector rellenado.

Returns

Un vector relleno con ceros de tamaño m.

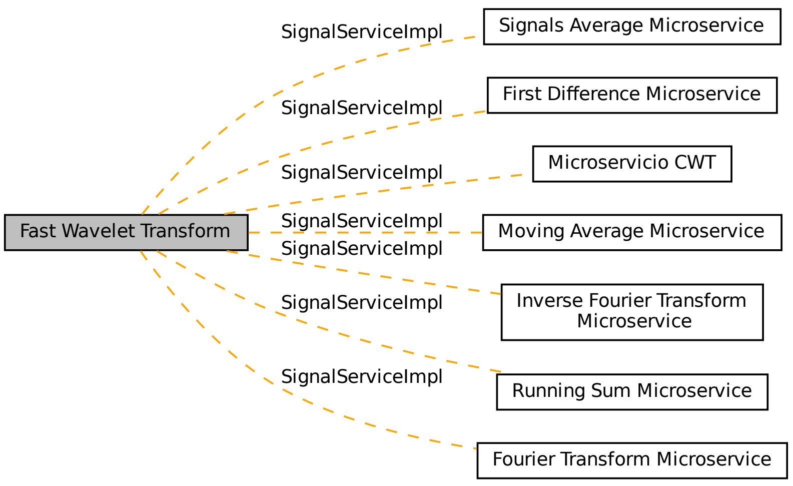
Here is the caller graph for this function:



8.3 Fast Wavelet Transform

This microservice computes the Fast Wavelet Transform (FWT) of complex signals.

Collaboration diagram for Fast Wavelet Transform:


Fast Wavelet Transform
SignalServiceImpl
Signals Average Microservice
First Difference Microservice
Microservicio CWT
Moving Average Microservice
Inverse Fourier Transform Microservice
Running Sum Microservice
Fourier Transform Microservice

Classes

- class SignalServiceImpl

  Clase que implementa el servicio de CWT.

Functions

- Status SignalServiceImpl::ComputeFastWaveletTransform (ServerContext ∗context, const FastWavelet↩ TransformRequest ∗request, FastWaveletTransformResponse ∗reply) override

  Computes the Fast Wavelet Transform of the provided signal.
- std::pair< std::vector< Eigen::VectorXcd >, std::vector< Eigen::VectorXcd > > SignalServiceImpl::FastWaveletTransform (const Eigen::VectorXcd &input, int DecLevel, std::string WaveName)

  Performs the Fast Wavelet Transform.
- std::pair< Eigen::VectorXcd, Eigen::VectorXcd > SignalServiceImpl::FastWaveletTransformAux (const Eigen::VectorXcd &input, Eigen::VectorXcd Lo_d, Eigen::VectorXcd Hi_d)

  Performs auxiliary computations for the Fast Wavelet Transform.
- Eigen::VectorXcd SignalServiceImpl::FFTconvolveEigen (const Eigen::VectorXcd &x, const Eigen::VectorXcd &h, bool shift=false)

  Computes the convolution of two signals using FFT.
- Eigen::VectorXcd SignalServiceImpl::ZeroPadGivenSize (const Eigen::VectorXcd &a, int m)

  Zero-pads a vector to the specified size.
- int **main** ()

8.3.1 Detailed Description

This microservice computes the Fast Wavelet Transform (FWT) of complex signals.

This module implements a gRPC service for computing the Fast Wavelet Transform (FWT) of a signal. It leverages Eigen for matrix operations and convolution.

8.3.2 Function Documentation

8.3.2.1 ComputeFastWaveletTransform()

```
Status SignalServiceImpl::ComputeFastWaveletTransform (
            ServerContext ∗ context,
            const FastWaveletTransformRequest ∗ request,
            FastWaveletTransformResponse ∗ reply )  [inline], [override]
```

Computes the Fast Wavelet Transform of the provided signal.

This function handles the gRPC request to compute the FWT, converting the incoming signal data into an appropriate format for processing. It allows users to specify parameters for the transformation, such as the wavelet type and the number of decomposition levels. Upon completion, it returns the computed approximation and detail coefficients.

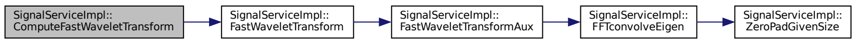
Parameters

context	gRPC server context, providing access to the server's state and functionalities.
request	Contains the signal data and parameters for the transform, including the wavelet name and desired decomposition levels.
reply	The response message containing the approximation and detail coefficients, structured as vectors for each decomposition level.

Returns

Status indicating success or failure of the operation, providing feedback on the processing outcome.

Here is the call graph for this function:


SignalServiceImpl:: ComputeFastWaveletTransform
SignalServiceImpl:: FastWaveletTransform
SignalServiceImpl:: FastWaveletTransformAux
SignalServiceImpl:: FFTconvolveEigen
SignalServiceImpl:: ZeroPadGivenSize

8.3.2.2 FastWaveletTransform()

```
std::pair< std::vector< Eigen::VectorXcd >, std::vector< Eigen::VectorXcd > > SignalService↩
Impl::FastWaveletTransform (
            const Eigen::VectorXcd & input,
            int DecLevel,
            std::string WaveName )  [inline], [private]
```

Performs the Fast Wavelet Transform.

This method executes the Fast Wavelet Transform on the provided input signal, allowing for multi-level decomposition. The user can specify the wavelet type, which determines the filter coefficients used during the transformation. The implementation adheres to the principles established by Mallat in 1988, which emphasizes computational efficiency in wavelet analysis.

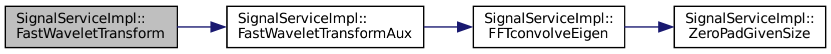
Parameters

input	The input signal as an Eigen::VectorXcd, representing complex signal data.
DecLevel	The number of decomposition levels to apply during the transformation, determining the granularity of the analysis.
WaveName	The name of the wavelet to be used, allowing flexibility in the choice of wavelet (e.g., Daubechies, Biorthogonal).

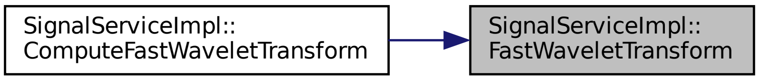
Returns

A pair containing vectors of approximations and details for each decomposition level, enabling further analysis or reconstruction of the signal.

Here is the call graph for this function:



Here is the caller graph for this function:



8.3.2.3 FastWaveletTransformAux()

```
std::pair<Eigen::VectorXcd,Eigen::VectorXcd> SignalServiceImpl::FastWaveletTransformAux (
            const Eigen::VectorXcd & input,
            Eigen::VectorXcd Lo_d,
            Eigen::VectorXcd Hi_d )  [inline], [private]
```

Performs auxiliary computations for the Fast Wavelet Transform.

This function applies convolution with specified low-pass and high-pass filters to the input signal, facilitating the decomposition process inherent in the wavelet transform. By utilizing filter coefficients corresponding to the selected wavelet, the function separates the signal into its approximation and detail components.

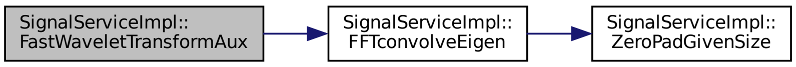
Parameters

input	The input signal, which is subject to convolution for wavelet decomposition.
Lo←_d	Low-pass filter coefficients for approximating the signal at each decomposition level.
Hi←_d	High-pass filter coefficients for extracting the detail components from the signal.

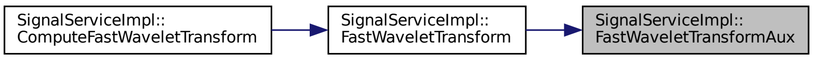
Returns

A pair of Eigen::VectorXcd representing the approximation and detail coefficients, allowing for further processing or analysis.

Here is the call graph for this function:



Here is the caller graph for this function:



8.3.2.4 FFTconvolveEigen()

```
Eigen::VectorXcd SignalServiceImpl::FFTconvolveEigen (
            const Eigen::VectorXcd & x,
            const Eigen::VectorXcd & h,
            bool shift = false )  [inline], [private]
```

Computes the convolution of two signals using FFT.

This function implements the convolution operation using the Fast Fourier Transform (FFT) for computational efficiency. By transforming both the input signal and the filter into the frequency domain, the convolution is computed as a simple multiplication, which is then transformed back to the time domain.

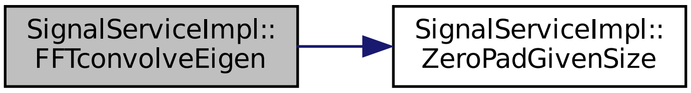
Parameters

x	The input signal, represented as an Eigen::VectorXcd, to which the convolution is applied.
h	The filter or kernel for convolution, which defines the characteristics of the transformation.
shift	Optional parameter for phase shifting, allowing for adjustments in the resulting signal phase as required for specific applications.

Returns

The result of the convolution as an Eigen::VectorXcd, representing the processed signal post-convolution.

Here is the call graph for this function:



8.3.2.5 ZeroPadGivenSize()

```
Eigen::VectorXcd SignalServiceImpl::ZeroPadGivenSize (
              const Eigen::VectorXcd & a,
              int m )  [inline], [private]
```

Zero-pads a vector to the specified size.

This function extends the input vector with zeros to reach a desired size. Zero-padding is crucial in signal processing, especially when preparing data for FFT-based operations, as it helps maintain consistent lengths for inputs and avoids boundary effects.

Parameters

a	The input vector to pad, which will be extended to the specified size.
m	The desired size for the padded vector, indicating how long the output vector should be.

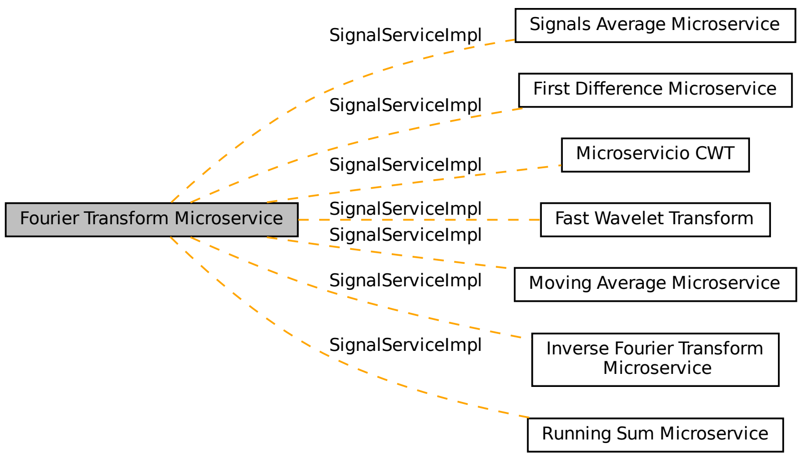
Returns

A zero-padded version of the input vector, ensuring it meets the specified size requirements.

8.4 Fourier Transform Microservice

This microservice computes the Fast Fourier Transform (FFT) of complex signals.

Collaboration diagram for Fourier Transform Microservice:


Fourier Transform Microservice
SignalServiceImpl
Signals Average Microservice
First Difference Microservice
Microservicio CWT
Fast Wavelet Transform
Moving Average Microservice
Inverse Fourier Transform Microservice
Running Sum Microservice

Classes

- class SignalServiceImpl

    Clase que implementa el servicio de CWT.

Functions

- Status SignalServiceImpl::ComputeFFT (ServerContext ∗context, const FFTRequest ∗request, FFTResponse ∗reply) override

    Computes the FFT of a signal.
- Eigen::VectorXcd SignalServiceImpl::FFTEigen (Eigen::VectorXcd &a)

    Performs the Fast Fourier Transform (FFT) on a complex signal.
- int **main** ()

8.4.1 Detailed Description

This microservice computes the Fast Fourier Transform (FFT) of complex signals.

The FFT microservice is implemented using gRPC and Eigen, handling complex-valued signals and applying the Fast Fourier Transform (FFT) to convert them from the time domain to the frequency domain.

8.4.2 Function Documentation

8.4.2.1 ComputeFFT()

```
Status SignalServiceImpl::ComputeFFT (
            ServerContext * context,
            const FFTRequest * request,
            FFTResponse * reply )  [inline], [override]
```

Computes the FFT of a signal.

This method processes the incoming request by extracting the complex signal from the gRPC request, converts it to an Eigen vector, applies the Fast Fourier Transform (FFT), and sends the result back in the response.

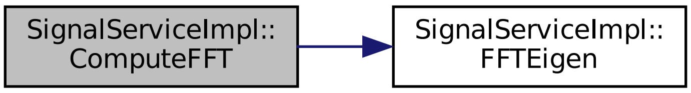
Parameters

context	The gRPC server context.
request	The request containing the input signal.
reply	The response containing the FFT-transformed signal.

Returns

A gRPC Status object indicating success or failure.

Here is the call graph for this function:



8.4.2.2 FFTEigen()

```
Eigen::VectorXcd SignalServiceImpl::FFTEigen (
            Eigen::VectorXcd & a )  [inline], [private]
```

Performs the Fast Fourier Transform (FFT) on a complex signal.

This function uses Eigen's FFT functionality to compute the forward FFT of the input complex signal.

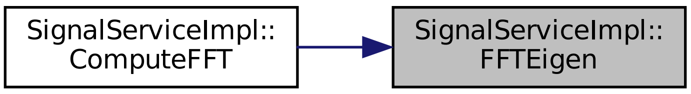
Parameters

a	The input signal as an Eigen::VectorXcd (complex vector).

Returns

The FFT-transformed signal as an Eigen::VectorXcd.

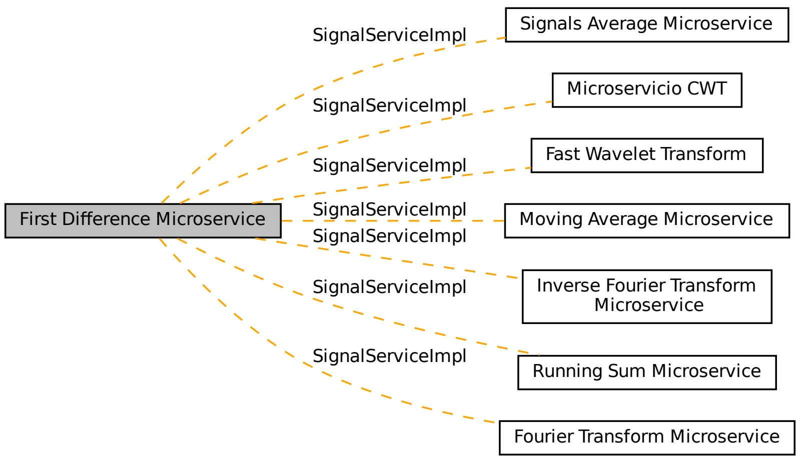
Here is the caller graph for this function:



8.5 First Difference Microservice

This microservice computes the first difference of a complex signal.

Collaboration diagram for First Difference Microservice:



Classes

- class SignalServiceImpl

    Clase que implementa el servicio de CWT.

Functions

- Status SignalServiceImpl::ComputeFirstDifference (ServerContext ∗context, const FirstDifferenceRequest ∗request, FirstDifferenceResponse ∗reply) override

    Computes the first difference of a signal.
- Eigen::VectorXcd SignalServiceImpl::FirstDifference (const Eigen::VectorXcd &input)

    Computes the first difference of a complex signal.
- int **main** ()

8.5.1 Detailed Description

This microservice computes the first difference of a complex signal.

The First Difference microservice is built using gRPC and Eigen. It processes complex-valued signals and computes the first difference between consecutive elements of the signal.

8.5.2 Function Documentation

8.5.2.1 ComputeFirstDifference()

```
Status SignalServiceImpl::ComputeFirstDifference (
            ServerContext * context,
            const FirstDifferenceRequest * request,
            FirstDifferenceResponse * reply )  [inline], [override]
```

Computes the first difference of a signal.

This method receives the input signal as a gRPC request, processes it, computes the first difference between consecutive elements of the complex signal, and sends the result in the response.

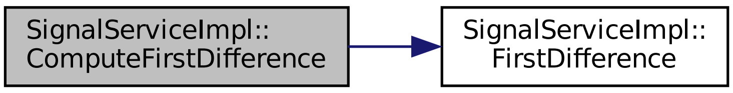
Parameters

context	The gRPC server context.
request	The request containing the input complex signal.
reply	The response containing the computed first difference of the signal.

Returns

A gRPC Status object indicating success or failure.

Here is the call graph for this function:



8.5.2.2 FirstDifference()

```
Eigen::VectorXcd SignalServiceImpl::FirstDifference (
            const Eigen::VectorXcd & input )  [inline], [private]
```

Computes the first difference of a complex signal.

This function calculates the first difference of an input signal, where each element in the result is the difference between consecutive elements of the input signal.

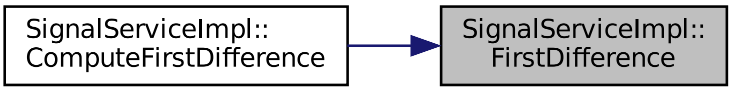
Parameters

input	The input signal as an Eigen::VectorXcd (complex vector).

Returns

The first difference of the signal as an Eigen::VectorXcd.

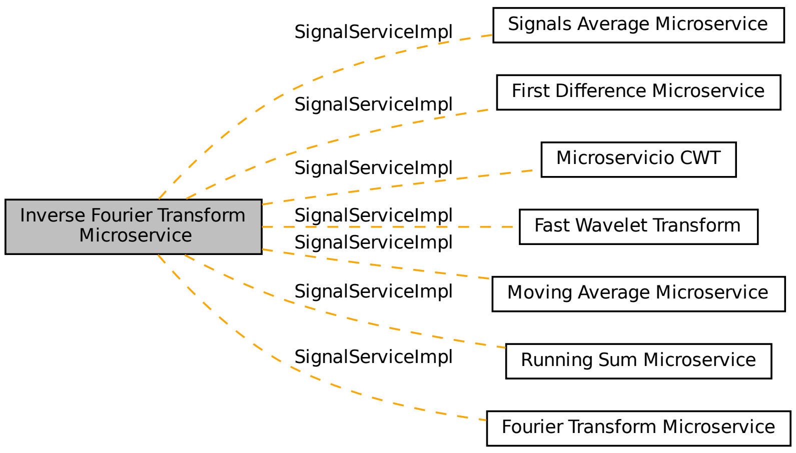
Here is the caller graph for this function:



8.6 Inverse Fourier Transform Microservice

This microservice computes the Inverse Fast Fourier Transform (IFFT) of complex signals.

Collaboration diagram for Inverse Fourier Transform Microservice:


Inverse Fourier Transform Microservice
SignalServiceImpl
Signals Average Microservice
First Difference Microservice
Microservicio CWT
Fast Wavelet Transform
Moving Average Microservice
Running Sum Microservice
Fourier Transform Microservice

Classes

- class SignalServiceImpl

    Clase que implementa el servicio de CWT.

Functions

- Status SignalServiceImpl::ComputeIFFT (ServerContext ∗context, const IFFTRequest ∗request, IFFTResponse ∗reply) override

    Computes the Inverse Fast Fourier Transform (IFFT) of a signal.
- Eigen::VectorXcd SignalServiceImpl::IFFTEigen (Eigen::VectorXcd &a)

    Performs the Inverse Fast Fourier Transform (IFFT) on a complex signal.
- int **main** ()

8.6.1 Detailed Description

This microservice computes the Inverse Fast Fourier Transform (IFFT) of complex signals.

The IFFT microservice is implemented using gRPC and Eigen, handling complex-valued signals and applying the Inverse Fast Fourier Transform (IFFT) to convert signals from the frequency domain back to the time domain.

8.6.2 Function Documentation

8.6.2.1 ComputeIFFT()

```
Status SignalServiceImpl::ComputeIFFT (
            ServerContext ∗ context,
            const IFFTRequest ∗ request,
            IFFTResponse ∗ reply ) [inline], [override]
```

Computes the Inverse Fast Fourier Transform (IFFT) of a signal.

This method processes the incoming request by extracting the complex signal from the gRPC request, converting it to an Eigen vector, applying the Inverse Fast Fourier Transform (IFFT), and sending the result back in the response.

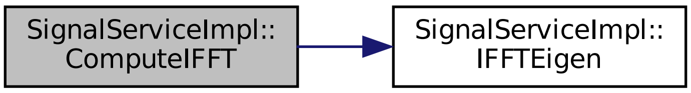
Parameters

context	The gRPC server context.
request	The request containing the input signal.
reply	The response containing the IFFT-transformed signal.

Returns

A gRPC Status object indicating success or failure.

Here is the call graph for this function:



8.6.2.2 IFFTEigen()

```
Eigen::VectorXcd SignalServiceImpl::IFFTEigen (
              Eigen::VectorXcd & a )  [inline], [private]
```

Performs the Inverse Fast Fourier Transform (IFFT) on a complex signal.

This function uses Eigen's FFT functionality to compute the inverse FFT of the input complex signal.

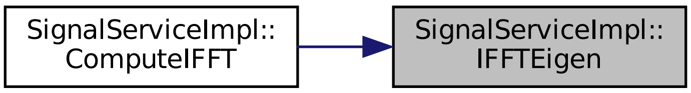
Parameters

a	The input signal as an Eigen::VectorXcd (complex vector).

Returns

The IFFT-transformed signal as an Eigen::VectorXcd.

Here is the caller graph for this function:



8.7 MAT Reader

This module implements a gRPC service to read MAT files and return the data for a specified field.

Classes

- class MatReaderServer.MatServiceServicer

  A gRPC service class that handles MAT file processing.

Functions

- def MatReaderServer.MatServiceServicer.readMat (self, request, context)

  Processes the MAT file received in the request and returns the data for the specified field.
- def **MatReaderServer.serve** ()

8.7.1 Detailed Description

This module implements a gRPC service to read MAT files and return the data for a specified field.

It uses the scipy.io library to read the MAT files.

8.7.2 Function Documentation

8.7.2.1 readMat()

```
def MatReaderServer.MatServiceServicer.readMat (
              self,
              request,
              context )
```

Processes the MAT file received in the request and returns the data for the specified field.

The method saves the received MAT file temporarily, processes it using scipy.io.loadmat, and extracts the matrix corresponding to the requested field. The matrix is then returned as a list of vectors.

Parameters

request	The gRPC request containing the MAT file (file_content) and the field name to be extracted.
context	The gRPC context (used for error handling and metadata).

Returns

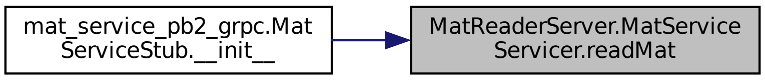
mat_service_pb2.MatResponse The response containing the extracted matrix or an error if the field is not found.

Exceptions

FileNotFoundError	Raised if the specified field is not found in the MAT file.

Reimplemented from mat_service_pb2_grpc.MatServiceServicer.

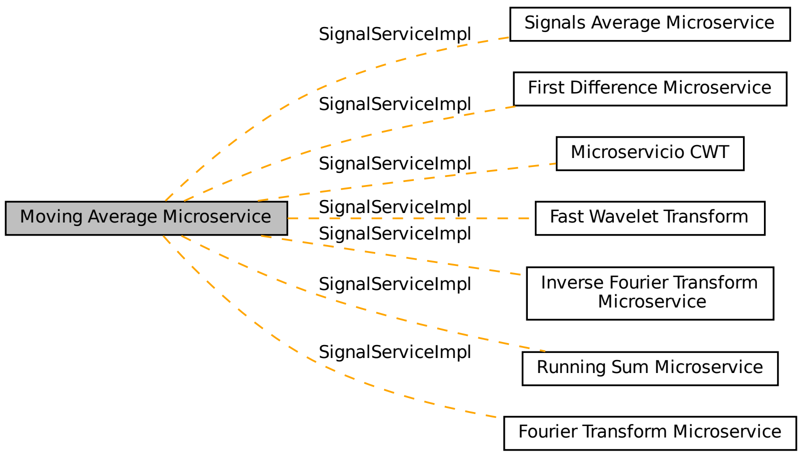
Here is the caller graph for this function:


mat_service_pb2_grpc.Mat ServiceStub.__init__
MatReaderServer.MatService Servicer.readMat

8.8 Moving Average Microservice

This microservice computes the moving average of a complex signal.

Collaboration diagram for Moving Average Microservice:


Moving Average Microservice
SignalServiceImpl
Signals Average Microservice
First Difference Microservice
Microservicio CWT
Fast Wavelet Transform
Inverse Fourier Transform Microservice
Running Sum Microservice
Fourier Transform Microservice

Classes

- class SignalServiceImpl

    Clase que implementa el servicio de CWT.

Functions

- Status SignalServiceImpl::ComputeMovingAverage (ServerContext ∗context, const MovingAverageRequest ∗request, MovingAverageResponse ∗reply) override

    Computes the moving average of a signal.
- Eigen::VectorXcd SignalServiceImpl::MovingAverage (const Eigen::VectorXcd &a, int window_size)

    Applies a moving average filter to a complex signal.
- int **main** ()

8.8.1 Detailed Description

This microservice computes the moving average of a complex signal.

The Moving Average microservice is implemented using gRPC and Eigen, handling complex-valued signals and applying a moving average filter with a specified window size.

8.8.2 Function Documentation

8.8.2.1 ComputeMovingAverage()

```
Status SignalServiceImpl::ComputeMovingAverage (
             ServerContext * context,
             const MovingAverageRequest * request,
             MovingAverageResponse * reply )  [inline], [override]
```

Computes the moving average of a signal.

This method takes the complex signal from the gRPC request, applies the moving average using the specified window size, and sends the result in the response.

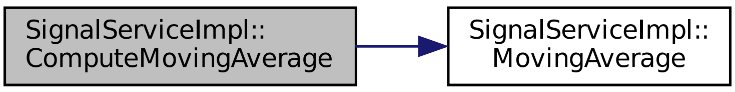
Parameters

context	The gRPC server context.
request	The request containing the input signal and window size.
reply	The response containing the filtered signal.

Returns

A gRPC Status object indicating success or failure.

Here is the call graph for this function:



8.8.2.2 MovingAverage()

```
Eigen::VectorXcd SignalServiceImpl::MovingAverage (
            const Eigen::VectorXcd & a,
            int window_size )  [inline], [private]
```

Applies a moving average filter to a complex signal.

This function calculates the moving average of the input complex signal using a sliding window and returns the filtered signal.

Parameters

a	The input signal as an Eigen::VectorXcd (complex vector).
window_size	The size of the moving window.

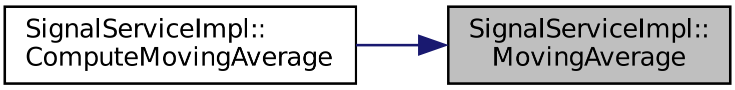
Returns

The filtered signal as an Eigen::VectorXcd.

Exceptions

std::invalid_argument	If the window size is less than 1 or greater than the signal size.

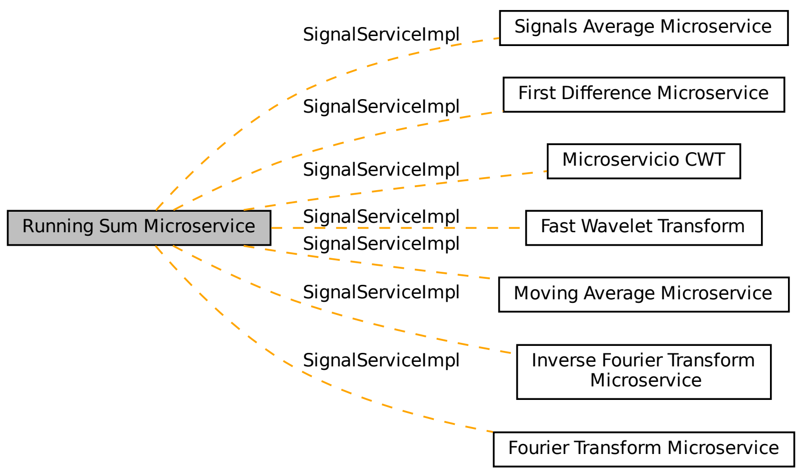
Here is the caller graph for this function:



8.9 Running Sum Microservice

This microservice computes the running sum of a complex signal.

Collaboration diagram for Running Sum Microservice:


Signals Average Microservice
First Difference Microservice
Microservicio CWT
Running Sum Microservice
Fast Wavelet Transform
Moving Average Microservice
Inverse Fourier Transform Microservice
Fourier Transform Microservice
SignalServiceImpl

Classes

- class SignalServiceImpl

  Clase que implementa el servicio de CWT.

Functions

- Status SignalServiceImpl::ComputeRuningSum (ServerContext ∗context, const RuningSumRequest ∗request, RuningSumResponse ∗reply) override

  Computes the running sum of a signal.
- Eigen::VectorXcd SignalServiceImpl::RuningSum (const Eigen::VectorXcd &Input)

  Calculates the running sum of a complex signal.
- int **main** ()

8.9.1 Detailed Description

This microservice computes the running sum of a complex signal.

The Running Sum microservice is implemented using gRPC and Eigen, processing complex-valued signals and calculating their cumulative sum.

8.9.2 Function Documentation

8.9.2.1 ComputeRuningSum()

```
Status SignalServiceImpl::ComputeRuningSum (
             ServerContext * context,
             const RuningSumRequest * request,
             RuningSumResponse * reply )  [inline], [override]
```

Computes the running sum of a signal.

This method receives the complex signal from the gRPC request, calculates the running sum of the input signal, and sends the result in the response.

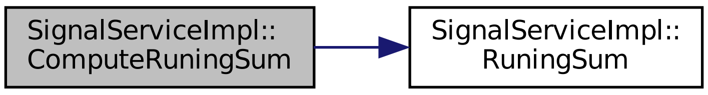
Parameters

context	The gRPC server context.
request	The request containing the input signal.
reply	The response containing the cumulative sum of the signal.

Returns

A gRPC Status object indicating success or failure.

Here is the call graph for this function:



8.9.2.2 RuningSum()

```
Eigen::VectorXcd SignalServiceImpl::RuningSum (
             const Eigen::VectorXcd & Input )  [inline], [private]
```

Calculates the running sum of a complex signal.

This function computes the cumulative sum of the input complex signal using Eigen's complex vector.

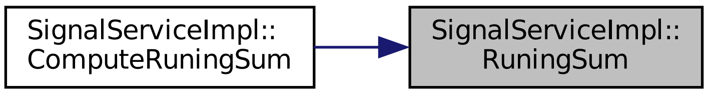
Parameters

Input	The input signal as an Eigen::VectorXcd (complex vector).

Returns

The running sum of the signal as an Eigen::VectorXcd.

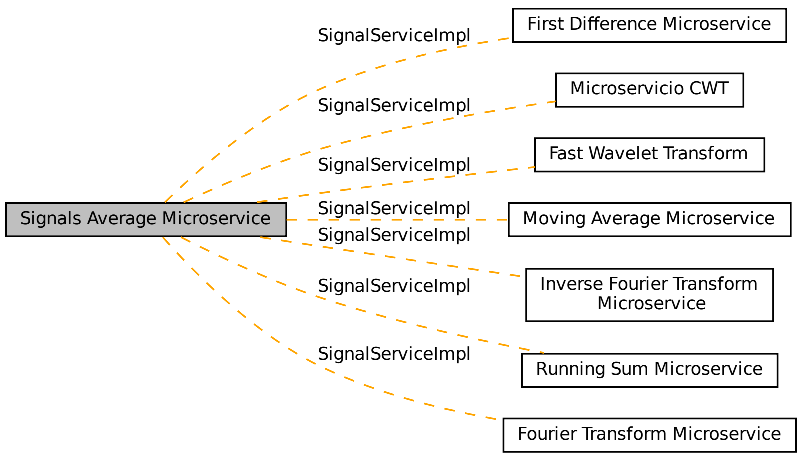
Here is the caller graph for this function:



8.10 Signals Average Microservice

This microservice computes the average of multiple complex signals.

Collaboration diagram for Signals Average Microservice:


Signals Average Microservice
SignalServiceImpl
First Difference Microservice
Microservicio CWT
Fast Wavelet Transform
Moving Average Microservice
Inverse Fourier Transform Microservice
Running Sum Microservice
Fourier Transform Microservice

Classes

- class SignalServiceImpl

  Clase que implementa el servicio de CWT.

Functions

- Status SignalServiceImpl::ComputeAverage (ServerContext ∗context, const AverageRequest ∗request, AverageResponse ∗reply) override

  Computes the average of multiple signals.
- Eigen::VectorXcd SignalServiceImpl::SignalsAverge (const std::vector< Eigen::VectorXcd > &Input)

  Computes the average of a vector of Eigen::VectorXcd signals.
- int **main** ()

8.10.1 Detailed Description

This microservice computes the average of multiple complex signals.

The Signals_Average microservice is implemented using gRPC and Eigen, allowing it to handle complex-valued signals and compute their averages efficiently. The service accepts input signals as complex vectors and returns the averaged signal.

8.10.2 Function Documentation

8.10.2.1 ComputeAverage()

```
Status SignalServiceImpl::ComputeAverage (
            ServerContext * context,
            const AverageRequest * request,
            AverageResponse * reply )  [inline], [override]
```

Computes the average of multiple signals.

This function receives a gRPC request containing multiple signals, processes each signal into an Eigen vector, calculates the average using the SignalsAverge method, and sends the averaged result back in the response.

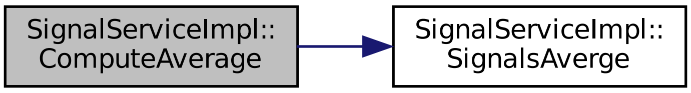
Parameters

context	The gRPC server context.
request	The request containing multiple signals.
reply	The response containing the averaged signal.

Returns

A gRPC Status object indicating success or failure.

Here is the call graph for this function:



8.10.2.2 SignalsAverge()

```
Eigen::VectorXcd SignalServiceImpl::SignalsAverge (
            const std::vector< Eigen::VectorXcd > & Input )  [inline], [private]
```

Computes the average of a vector of Eigen::VectorXcd signals.

This function takes a collection of complex-valued signals (Eigen vectors), ensures they are of the same size, and computes their element-wise average.

Parameters

Input	A vector of Eigen::VectorXcd containing the input signals.

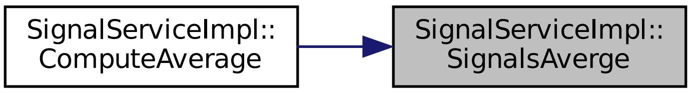
Returns

The averaged signal as an Eigen::VectorXcd.

Exceptions

std::runtime_error	if the input vectors have different sizes.

Here is the caller graph for this function:

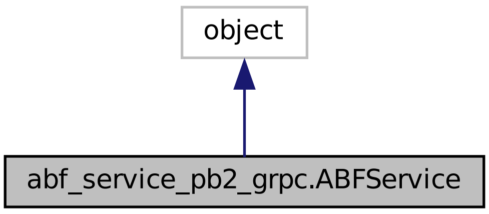


Chapter 9

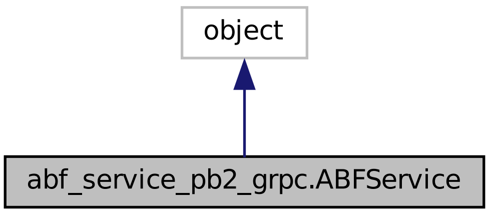
Class Documentation

9.1 abf_service_pb2_grpc.ABFService Class Reference

Inheritance diagram for abf_service_pb2_grpc.ABFService:



Collaboration diagram for abf_service_pb2_grpc.ABFService:



Static Public Member Functions

- def **readAbf** (request, target, options=(), channel_credentials=None, call_credentials=None, insecure=False, compression=None, wait_for_ready=None, timeout=None, metadata=None)

9.1.1 Detailed Description

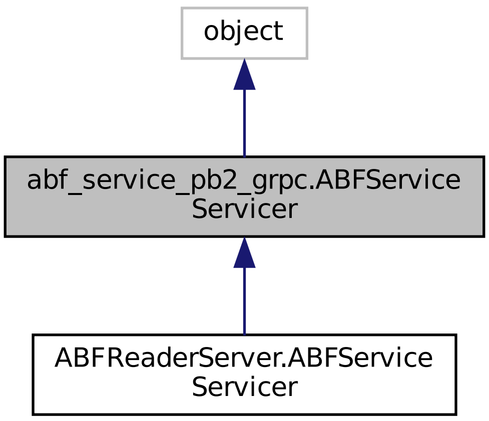
```
Missing associated documentation comment in .proto file.
```

The documentation for this class was generated from the following file:

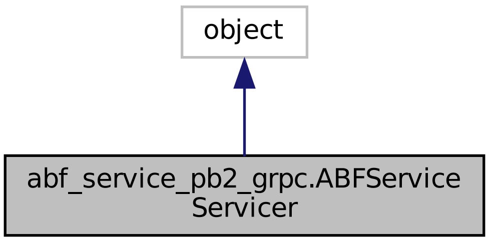
- abf_service_pb2_grpc.py

9.2 abf_service_pb2_grpc.ABFServiceServicer Class Reference

Inheritance diagram for abf_service_pb2_grpc.ABFServiceServicer:


object
abf_service_pb2_grpc.ABFService Servicer
ABFReaderServer.ABFService Servicer

Collaboration diagram for abf_service_pb2_grpc.ABFServiceServicer:



Public Member Functions

- def readAbf (self, request, context)

9.2.1 Detailed Description

```
Missing associated documentation comment in .proto file.
```

9.2.2 Member Function Documentation

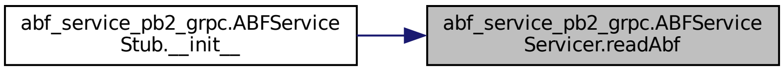
9.2.2.1 readAbf()

```
def abf_service_pb2_grpc.ABFServiceServicer.readAbf (
              self,
              request,
              context )
```

```
Missing associated documentation comment in .proto file.
```

Reimplemented in ABFReaderServer.ABFServiceServicer.

Here is the caller graph for this function:


abf_service_pb2_grpc.ABFService Stub.__init__
abf_service_pb2_grpc.ABFService Servicer.readAbf

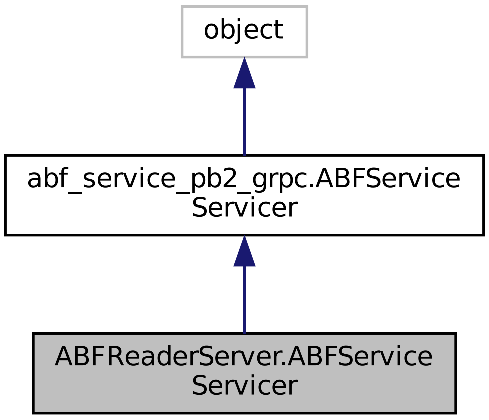
The documentation for this class was generated from the following file:

- abf_service_pb2_grpc.py

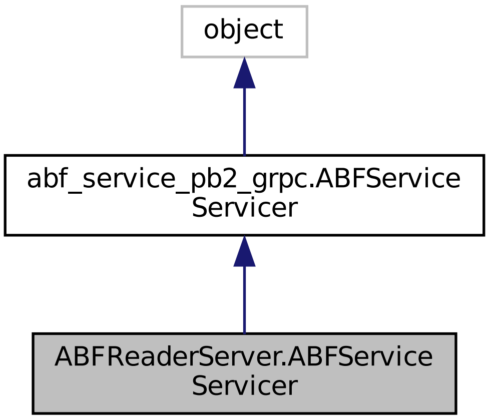
9.3 ABFReaderServer.ABFServiceServicer Class Reference

A gRPC service class that handles ABF file processing.

Inheritance diagram for ABFReaderServer.ABFServiceServicer:


object
abf_service_pb2_grpc.ABFService Servicer
ABFReaderServer.ABFService Servicer

Collaboration diagram for ABFReaderServer.ABFServiceServicer:


object
abf_service_pb2_grpc.ABFService Servicer
ABFReaderServer.ABFService Servicer

Public Member Functions

- def readAbf (self, request, context)

  Processes the ABF file received in the request and returns its data and sampling rate.

9.3.1 Detailed Description

A gRPC service class that handles ABF file processing.

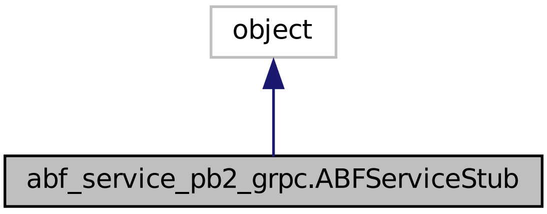
The class receives ABF files via gRPC, processes them using the pyABF library, and returns the extracted signal data and sample rate.

The documentation for this class was generated from the following file:

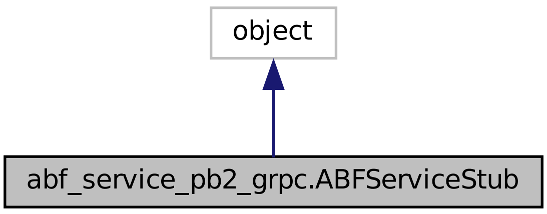
- ABFReaderServer.py

9.4 abf_service_pb2_grpc.ABFServiceStub Class Reference

Inheritance diagram for abf_service_pb2_grpc.ABFServiceStub:



Collaboration diagram for abf_service_pb2_grpc.ABFServiceStub:



Public Member Functions

- def __init__ (self, channel)

Public Attributes

- **readAbf**

9.4.1 Detailed Description

```
Missing associated documentation comment in .proto file.
```

9.4.2 Constructor & Destructor Documentation

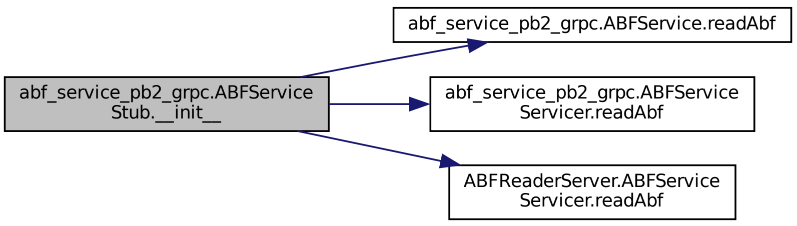
9.4.2.1 __init__()

```
def abf_service_pb2_grpc.ABFServiceStub.__init__ (
              self,
              channel )
```

```
Constructor.

Args:
    channel: A grpc.Channel.
```

Here is the call graph for this function:

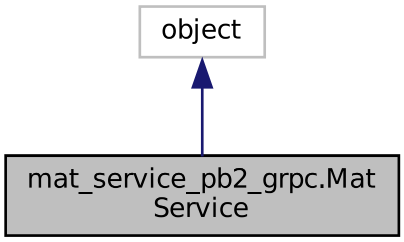

abf_service_pb2_grpc.ABFService.readAbf
abf_service_pb2_grpc.ABFService Stub.__init__
abf_service_pb2_grpc.ABFService Servicer.readAbf
ABFReaderServer.ABFService Servicer.readAbf

The documentation for this class was generated from the following file:

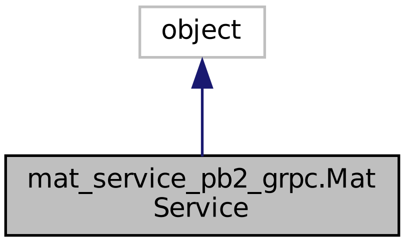
- abf_service_pb2_grpc.py

9.5 mat_service_pb2_grpc.MatService Class Reference

Inheritance diagram for mat_service_pb2_grpc.MatService:



Collaboration diagram for mat_service_pb2_grpc.MatService:



Static Public Member Functions

- def **readMat** (request, target, options=(), channel_credentials=None, call_credentials=None, insecure=False, compression=None, wait_for_ready=None, timeout=None, metadata=None)

9.5.1 Detailed Description

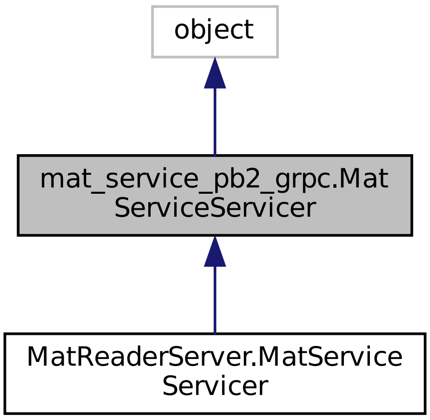
```
Missing associated documentation comment in .proto file.
```

The documentation for this class was generated from the following file:

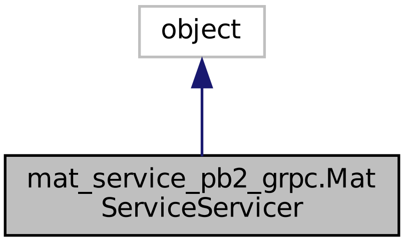
- mat_service_pb2_grpc.py

9.6 mat_service_pb2_grpc.MatServiceServicer Class Reference

Inheritance diagram for mat_service_pb2_grpc.MatServiceServicer:


object
mat_service_pb2_grpc.Mat ServiceServicer
MatReaderServer.MatService Servicer

Collaboration diagram for mat_service_pb2_grpc.MatServiceServicer:



Public Member Functions

- def readMat (self, request, context)

9.6.1 Detailed Description

```
Missing associated documentation comment in .proto file.
```

9.6.2 Member Function Documentation

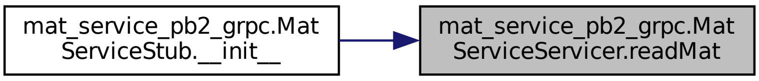
9.6.2.1 readMat()

```
def mat_service_pb2_grpc.MatServiceServicer.readMat (
              self,
              request,
              context )
```

```
Missing associated documentation comment in .proto file.
```

Reimplemented in MatReaderServer.MatServiceServicer.

Here is the caller graph for this function:


mat_service_pb2_grpc.Mat ServiceStub.__init__
mat_service_pb2_grpc.Mat ServiceServicer.readMat

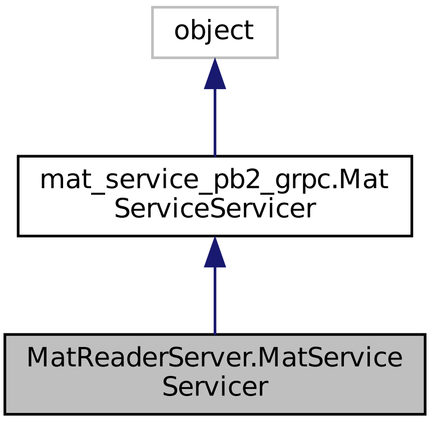
The documentation for this class was generated from the following file:

- mat_service_pb2_grpc.py

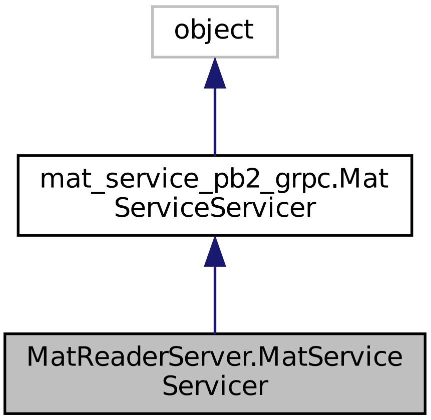
9.7 MatReaderServer.MatServiceServicer Class Reference

A gRPC service class that handles MAT file processing.

Inheritance diagram for MatReaderServer.MatServiceServicer:


object
mat_service_pb2_grpc.Mat ServiceServicer
MatReaderServer.MatService Servicer

Collaboration diagram for MatReaderServer.MatServiceServicer:


object
mat_service_pb2_grpc.Mat ServiceServicer
MatReaderServer.MatService Servicer

Public Member Functions

- def readMat (self, request, context)

    Processes the MAT file received in the request and returns the data for the specified field.

9.7.1 Detailed Description

A gRPC service class that handles MAT file processing.

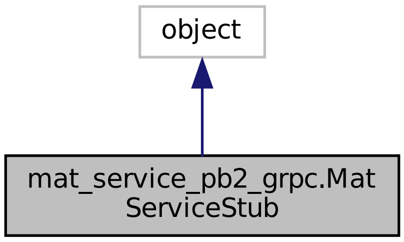
The class receives MAT files via gRPC, processes them using scipy.io, and returns the requested data field as a matrix of vectors.

The documentation for this class was generated from the following file:

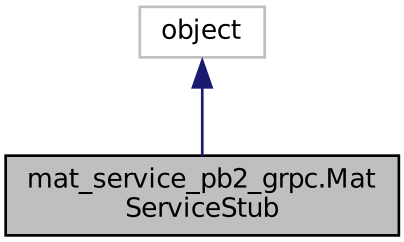
- MatReaderServer.py

9.8 mat_service_pb2_grpc.MatServiceStub Class Reference

Inheritance diagram for mat_service_pb2_grpc.MatServiceStub:



Collaboration diagram for mat_service_pb2_grpc.MatServiceStub:



Public Member Functions

- def __init__ (self, channel)

Public Attributes

- **readMat**

9.8.1 Detailed Description

```
Missing associated documentation comment in .proto file.
```

9.8.2 Constructor & Destructor Documentation

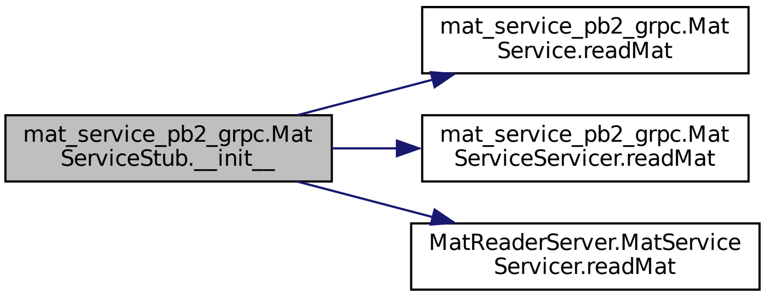
9.8.2.1 __init__()

```
def mat_service_pb2_grpc.MatServiceStub.__init__ (
              self,
              channel )
```

```
Constructor.

Args:
    channel: A grpc.Channel.
```

Here is the call graph for this function:



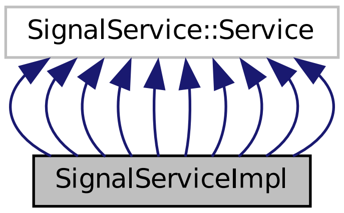
The documentation for this class was generated from the following file:

- mat_service_pb2_grpc.py

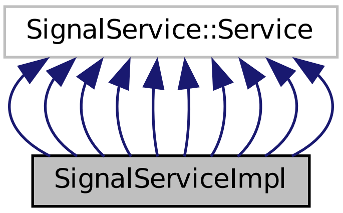
9.9 SignalServiceImpl Class Reference

Clase que implementa el servicio de CWT.

Inheritance diagram for SignalServiceImpl:



Collaboration diagram for SignalServiceImpl:



Public Member Functions

- Status ComputeCWT (ServerContext ∗context, const CWTRequest ∗request, CWTResponse ∗response) override

    Computes the Continuous Wavelet Transform (CWT) for a given signal.
- Status **ComputeFastWaveletHaar** (ServerContext ∗context, const FastWaveletTransformHaarRequest ∗request, FastWaveletTransformHaarResponse ∗reply) override
- Status ComputeFastWaveletTransform (ServerContext ∗context, const FastWaveletTransformRequest ∗request, FastWaveletTransformResponse ∗reply) override

    Computes the Fast Wavelet Transform of the provided signal.
- Status **ComputeFftConvolve** (ServerContext ∗context, const FftConvolveRequest ∗request, FftConvolve↩ Response ∗reply) override
- Status ComputeFFT (ServerContext ∗context, const FFTRequest ∗request, FFTResponse ∗reply) override

    Computes the FFT of a signal.
- Status ComputeFirstDifference (ServerContext ∗context, const FirstDifferenceRequest ∗request, First↩ DifferenceResponse ∗reply) override

    Computes the first difference of a signal.
- Status ComputeIFFT (ServerContext ∗context, const IFFTRequest ∗request, IFFTResponse ∗reply) override

    Computes the Inverse Fast Fourier Transform (IFFT) of a signal.
- Status ComputeMovingAverage (ServerContext ∗context, const MovingAverageRequest ∗request, Moving↩ AverageResponse ∗reply) override

    Computes the moving average of a signal.
- Status ComputeRuningSum (ServerContext ∗context, const RuningSumRequest ∗request, RuningSum↩ Response ∗reply) override

    Computes the running sum of a signal.
- Status ComputeAverage (ServerContext ∗context, const AverageRequest ∗request, AverageResponse ∗reply) override

    Computes the average of multiple signals.

Private Member Functions

- std::vector< double > GenerateLogScales (double start, double end, int numScales)

    Genera escalas logarítmicas.
- std::vector< Eigen::VectorXcd > CWTEigen (const Eigen::VectorXcd &signal, const std::vector< double > &scales)

Calcula la CWT utilizando el Wavelet de Morlet.

- Eigen::VectorXcd MorletWavelet (int N, double scale, double f0=1.0, double fb=1.0)

  Genera el Wavelet de Morlet.
- Eigen::VectorXcd FFTconvolveEigen (const Eigen::VectorXcd &x, const Eigen::VectorXcd &h, bool shift)

  Realiza la convolución en el dominio de Fourier.
- Eigen::VectorXcd ZeroPadGivenSize (const Eigen::VectorXcd &a, int m)

  Zero-pads a vector to a specified size.
- std::pair< Eigen::VectorXcd, Eigen::VectorXcd > **FastWaveletTransformHaar** (const Eigen::VectorXcd &input)
- std::pair< std::vector< Eigen::VectorXcd >, std::vector< Eigen::VectorXcd > > FastWaveletTransform (const Eigen::VectorXcd &input, int DecLevel, std::string WaveName)

  Performs the Fast Wavelet Transform.
- std::pair< Eigen::VectorXcd, Eigen::VectorXcd > FastWaveletTransformAux (const Eigen::VectorXcd &input, Eigen::VectorXcd Lo_d, Eigen::VectorXcd Hi_d)

  Performs auxiliary computations for the Fast Wavelet Transform.
- Eigen::VectorXcd FFTconvolveEigen (const Eigen::VectorXcd &x, const Eigen::VectorXcd &h, bool shift=false)

  Computes the convolution of two signals using FFT.
- Eigen::VectorXcd ZeroPadGivenSize (const Eigen::VectorXcd &a, int m)

  Zero-pads a vector to the specified size.
- Eigen::VectorXcd **FFTconvolveEigen** (const Eigen::VectorXcd &x, const Eigen::VectorXcd &h, bool shift)
- Eigen::VectorXcd **ZeroPadGivenSize** (const Eigen::VectorXcd &h, int size)
- Eigen::VectorXcd FFTEigen (Eigen::VectorXcd &a)

  Performs the Fast Fourier Transform (FFT) on a complex signal.
- Eigen::VectorXcd FirstDifference (const Eigen::VectorXcd &input)

  Computes the first difference of a complex signal.
- Eigen::VectorXcd IFFTEigen (Eigen::VectorXcd &a)

  Performs the Inverse Fast Fourier Transform (IFFT) on a complex signal.
- Eigen::VectorXcd MovingAverage (const Eigen::VectorXcd &a, int window_size)

  Applies a moving average filter to a complex signal.
- Eigen::VectorXcd RuningSum (const Eigen::VectorXcd &Input)

  Calculates the running sum of a complex signal.
- Eigen::VectorXcd SignalsAverge (const std::vector< Eigen::VectorXcd > &Input)

  Computes the average of a vector of Eigen::VectorXcd signals.

9.9.1 Detailed Description

Clase que implementa el servicio de CWT.

Implements the SignalService for computing average signals.

Implements the SignalService for computing the running sum of complex signals.

Implements the SignalService for computing the moving average of complex signals.

Implements the SignalService for computing the Inverse Fast Fourier Transform (IFFT).

Implements the SignalService for computing the first difference of complex signals.

Implements the SignalService for computing the Fast Fourier Transform (FFT).

Implements the SignalService for computing Fast wavelet transform (FWT).

Esta clase proporciona el servicio de gRPC para calcular la Transformada Wavelet Continua a partir de una señal de entrada.

This class provides the implementation of the SignalService, specifically the ComputeFastWaveletTransform method, which takes a complex signal as input, applies FWT using the Eigen library, and returns the result in the form of a vector of pairs of complex vector.

This class provides the implementation of the SignalService, specifically the ComputeFFT method, which takes a complex signal as input, applies FFT using the Eigen library, and returns the result in the form of a complex vector.

This class provides the implementation of the SignalService, specifically the `ComputeFirstDifference` method, which processes a signal, calculates its first difference (i.e., the difference between consecutive elements), and returns the result.

This class provides the implementation of the SignalService, specifically the ComputeIFFT method, which takes a complex signal as input, applies IFFT using the Eigen library, and returns the result in the form of a complex vector.

This class provides the implementation of the SignalService, specifically the ComputeMovingAverage method, which processes complex signals, applies a moving average filter, and returns the result.

This class provides the implementation of the SignalService, specifically the ComputeRuningSum method, which processes complex signals, calculates their running sum, and returns the result.

This class provides the implementation of the SignalService, specifically the ComputeAverage method, which takes multiple input signals, computes their average, and returns the result in the form of a complex vector.

The documentation for this class was generated from the following file:

- CWT/src/server.cc