Table of Contents

Group 4 .NET FTP Server Library Documentation – HIOF, Spring 2025

Version 2.0.0

Release date: 27/5/2025

Introduction

This library implements an FTP server written in C#. It allows developers to integrate FTP functionality into their own applications, whether for storage, file transfer, or as part of backend systems. It is particularly useful for developers who want a customizable and extensible FTP solution without having to write low-level networking code themselves.

The library is intended for .NET developers working on server applications, system integration, or those who need to expose file access over a network using the FTP protocol.

Installation

Follow these steps to install and set up the library:

1. Clone or download the repository to your desired directory.
2. Open the solution in Visual Studio 2022 or later.
3. Ensure that .NET 8 SDK is installed.
4. Build the solution using 'Build Solution' (Ctrl+Shift+B).
5. Reference the 'Group4.FtpServer' project in your own application if using it as a library.
6. Start the FTP server by instantiating the FtpServer class and calling StartAsync(). (Check out basic usage examples).

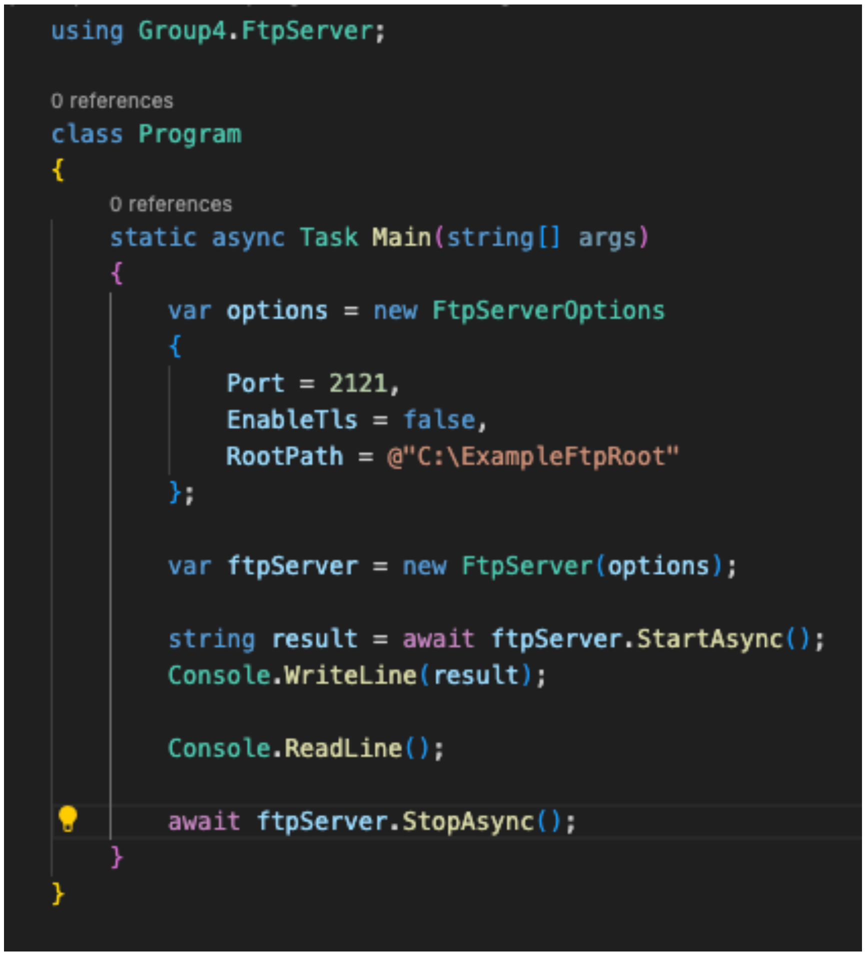
Overall structure of the FTP server library

- Networking layer – handling all TCP connections from clients.
- Session & Server layer – Manages the lifetime and the context of each FTP client session
- Command Handling layer - implements the FTP protocol commands as a collection of individual handlers.
- Authentication layer

- Storage layer – file system interactions are abstracted behind IBackendStorage interface, supporting multiple backends.
- Configuration – server options are managed via the FtpServerOptions class.

Basic usage examples

Example 1

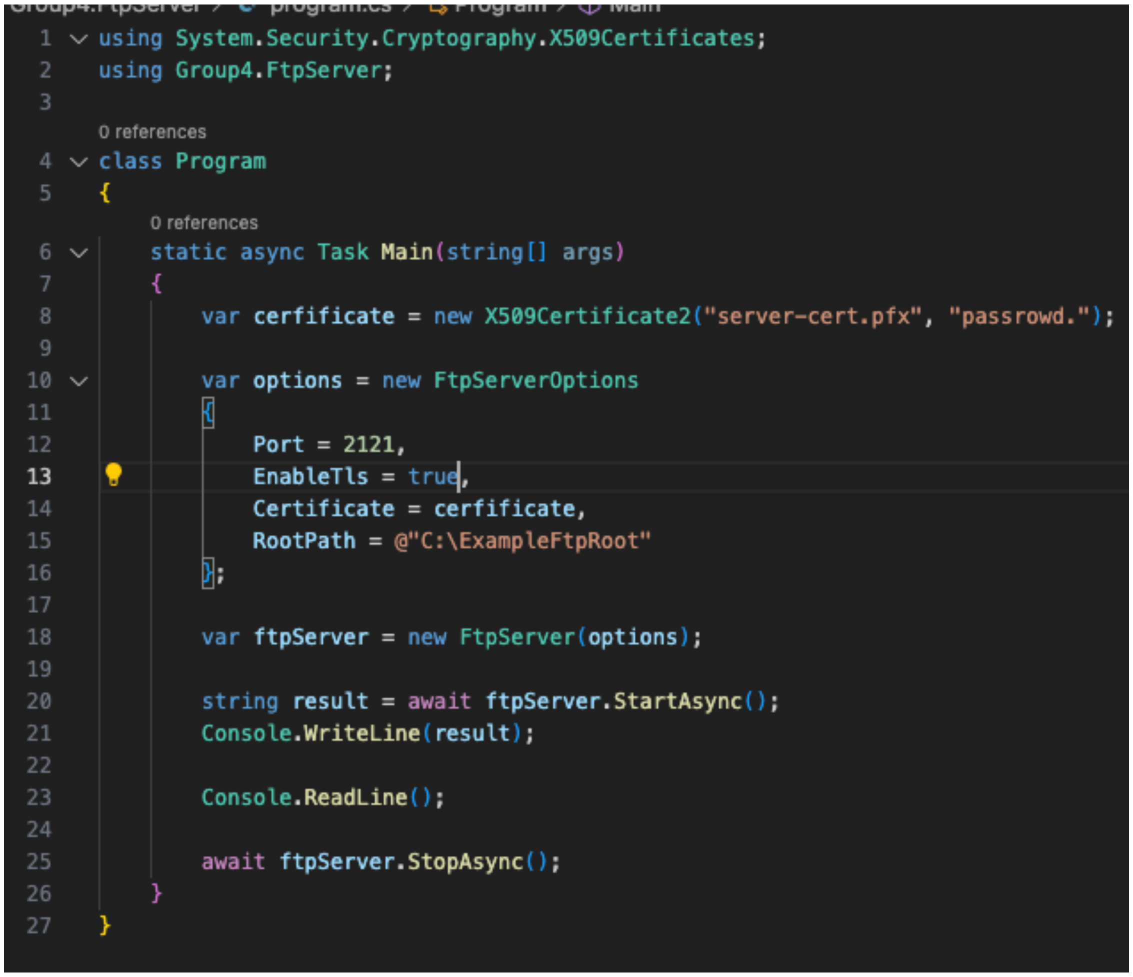


```
using Group4.FtpServer;

0 references
class Program
{
    0 references
    static async Task Main(string[] args)
    {
        var options = new FtpServerOptions
        {
            Port = 2121,
            EnableTls = false,
            RootPath = @"C:\ExampleFtpRoot"
        };

        var ftpServer = new FtpServer(options);

        string result = await ftpServer.StartAsync();
        Console.WriteLine(result);

        Console.ReadLine();

        await ftpServer.StopAsync();
    }
}
```

Summary:

This example demonstrates how to configure and start the FTP server using the FtpServerOptions and FtpServer classes. The server is started asynchronously, listens on port 2121, and uses a local folder (C:\ExampleFtpRoot) as the root for file storage. Once started, it prints a status message and waits for user input before shutting down.

Example 2 (with TLS)



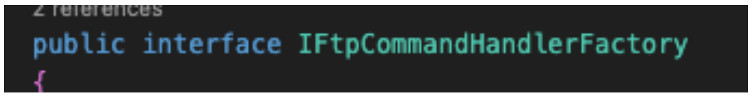
```
using System.Security.Cryptography.X509Certificates;
using Group4.FtpServer;

0 references
class Program
{
    0 references
    static async Task Main(string[] args)
    {
        var cerfificate = new X509Certificate2("server-cert.pfx", "passrowd.");

        var options = new FtpServerOptions
        {
            Port = 2121,
            EnableTls = true,
            Certificate = cerfificate,
            RootPath = @"C:\ExampleFtpRoot"
        };

        var ftpServer = new FtpServer(options);

        string result = await ftpServer.StartAsync();
        Console.WriteLine(result);

        Console.ReadLine();

        await ftpServer.StopAsync();
    }
}
```

Summary:

Much the same as in example 1, however, demonstration of using the server with TLS is shown here in example 2.

Command Handlers

IFTPCommandHandlerFactory

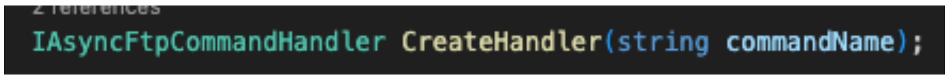


```
2 references
public interface IFtpCommandHandlerFactory
{
```

Summary:

The IFtpCommandHandlerFactory interface defines a factory contract for creating command handler instances that process FTP commands like USER, PASS, LIST, and others.

Methods



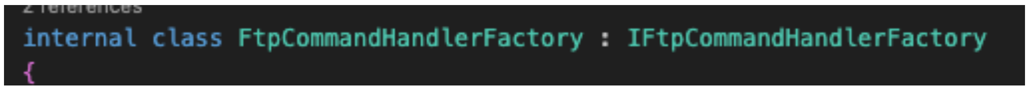
```
2 references
IAsyncFtpCommandHandler CreateHandler(string commandName);
```

Creates and returns an instance of the appropriate FTP command handler for the specified command.

Throws:

- ArgumentException if the command name is null or empth.
- NotSupportedException if the command is not supported by the factory.

FTPCommandHandlerFactory

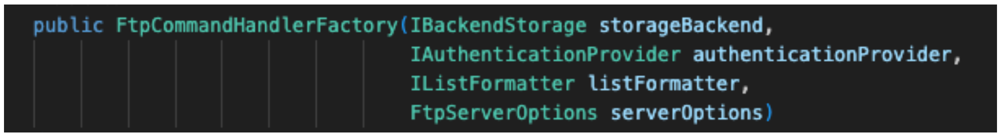


```
2 references
internal class FtpCommandHandlerFactory : IFtpCommandHandlerFactory
{
```

Summary:

The FtpCommandHandlerFactory class is the default implementation of the IFtpCommandHandlerFactory interface. It dynamically creates FTP command handler instances based on the command name received from the client. Each supported FTP command (e.g USER, PASS, LIST, STOR) is mapped to a specific handler class that contains the logic to process that command.

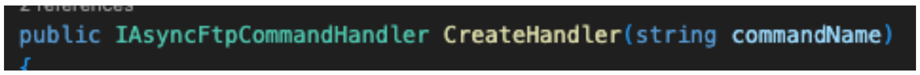
Constructor



```
public FtpCommandHandlerFactory(IBackendStorage storageBackend,
                                IAuthenticationProvider authenticationProvider,
                                IListFormatter listFormatter,
                                FtpServerOptions serverOptions)
```

Initializes the factory with all necessary dependencies required to construct command handlers.

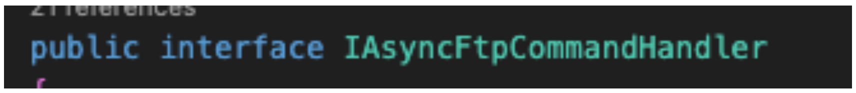
Methods



```
public IAsyncFtpCommandHandler CreateHandler(string commandName)
```

Creates an appropriate FTP command handler instance based on the input commandName. Internally calls a private method that matches the name against known command keywords.

IAsyncFtpCommandHandler

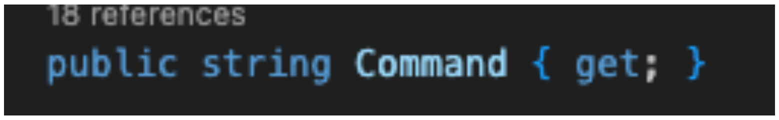


```
public interface IAsyncFtpCommandHandler
```

Summary:

The IAsyncFtpCommandHandler interface defines the contract for processing individual FTP commands asynchronously. Each handler is responsible for responding to a specific command (e.gUSER, LIST, RETR) issued by the client during an active FTP session.

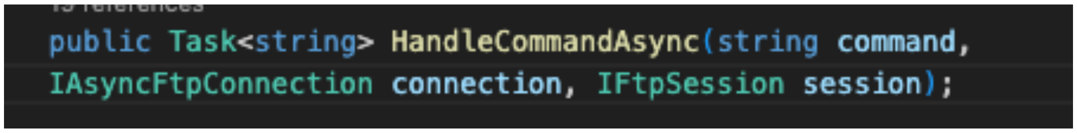
Properties



```
18 references
public string Command { get; }
```

Specifies the FTP command string that this handler is responsible for (e.g. USER, LIST). Used by the server or factory to match command strings to the correct handler.

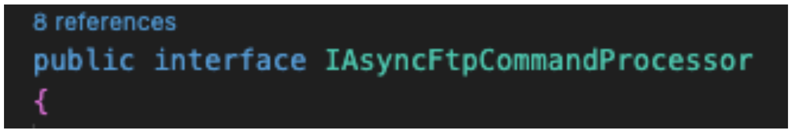
Methods



```
public Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session);
```

Processes the specified FTP command and returns a response string asynchronously.

IAsyncFtpCommandProcessor

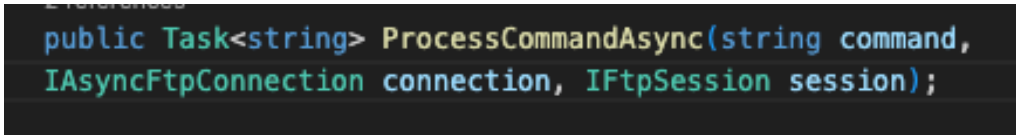


```
8 references
public interface IAsyncFtpCommandProcessor
{
```

Summary:

The IAsyncFtpCommandProcessor interface defines the contract for routing and processing FTP commands received from a client during an active session. Rather than executing command logic directly, this processor delegates the command to the appropriate handler

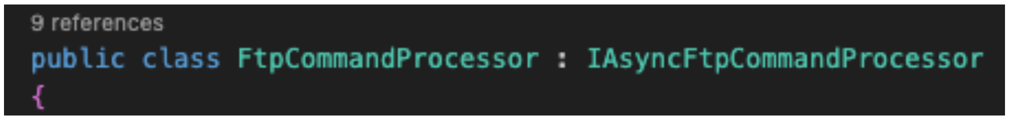
Methods



```
public Task<string> ProcessCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session);
```

Routes the given FTP command to the appropriate handler and returns the corresponding response asynchronously.

FtpCommandProcessor

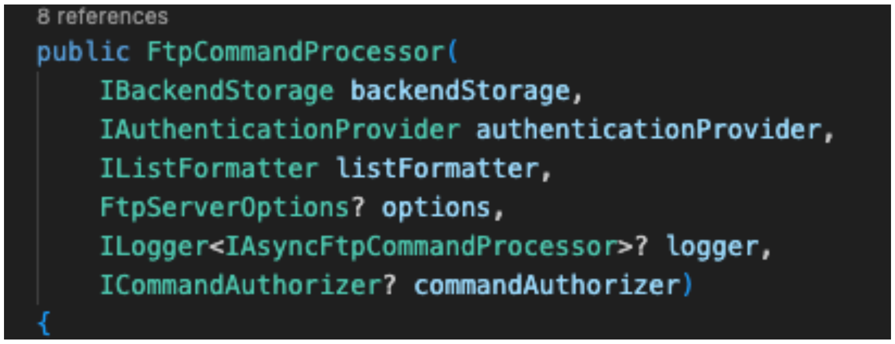


```
9 references
public class FtpCommandProcessor : IAsyncFtpCommandProcessor
{
```

Summary:

The FtpCommandProcessor class is the core dispatcher for handling FTP commands. It implements the IAsyncFtpCommandProcessor interface and delegates the actual command execution to the appropriate handler retrieved from a FtpCommandHandlerFactory. It supports flexible construction with or without logging and server configuration options.

Constructors

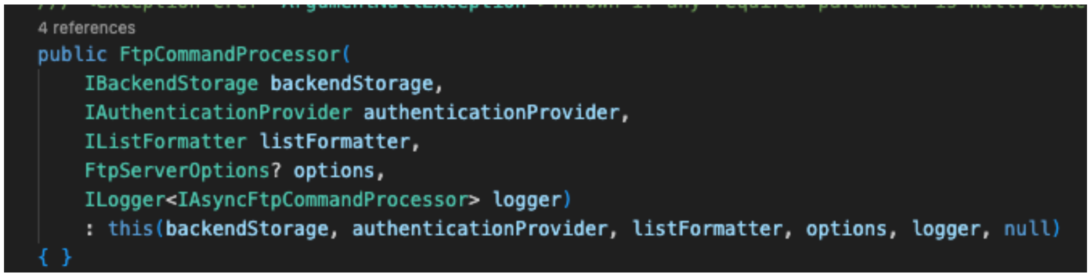


```
8 references
public FtpCommandProcessor(
    IBackendStorage backendStorage,
    IAuthenticationProvider authenticationProvider,
    IListFormatter listFormatter,
    FtpServerOptions? options,
    ILogger<IAsyncFtpCommandProcessor>? logger,
    ICommandAuthorizer? commandAuthorizer)
{
```

Initializes the command processor with all dependencies, including storage, authentication, directory listing formatter, server options, logger and command authorizer.

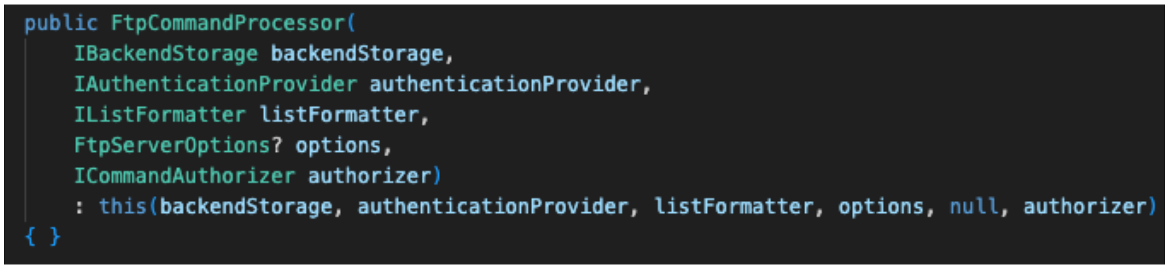
Throws:

- ArgumentNullException if either backendStorage, AuthenticationProvider or listFormatter is null.



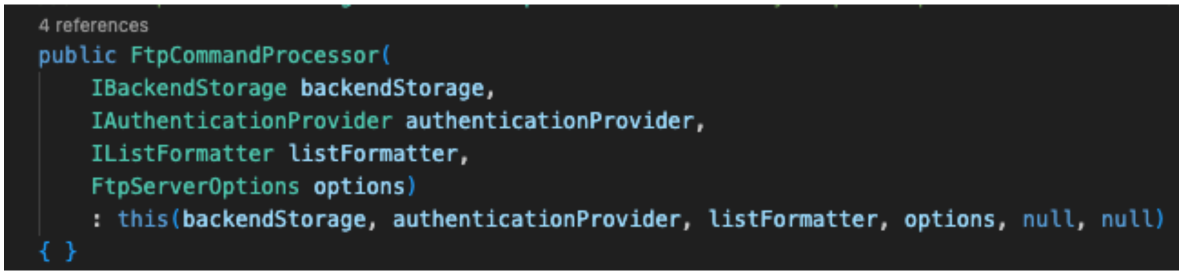
```
4 references
public FtpCommandProcessor(
    IBackendStorage backendStorage,
    IAuthenticationProvider authenticationProvider,
    IListFormatter listFormatter,
    FtpServerOptions? options,
    ILogger<IAsyncFtpCommandProcessor> logger)
    : this(backendStorage, authenticationProvider, listFormatter, options, logger, null)
{ }
```

Initializes a new instance of the FTPCommand Processor without an authorizer.



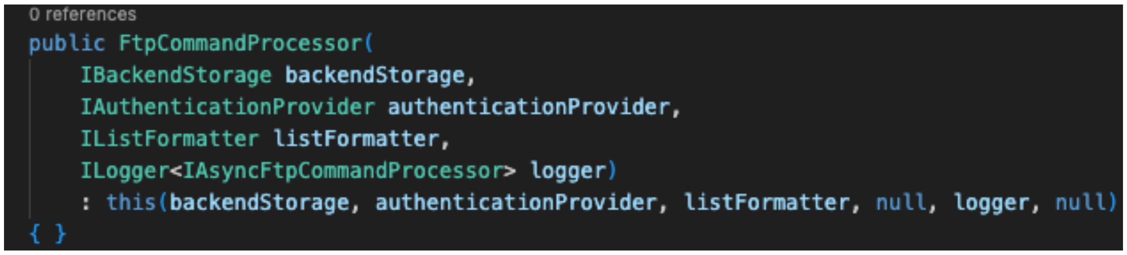
```
public FtpCommandProcessor(
    IBackendStorage backendStorage,
    IAuthenticationProvider authenticationProvider,
    IListFormatter listFormatter,
    FtpServerOptions? options,
    ICommandAuthorizer authorizer)
    : this(backendStorage, authenticationProvider, listFormatter, options, null, authorizer)
{ }
```

Initializes the processor without a logger.



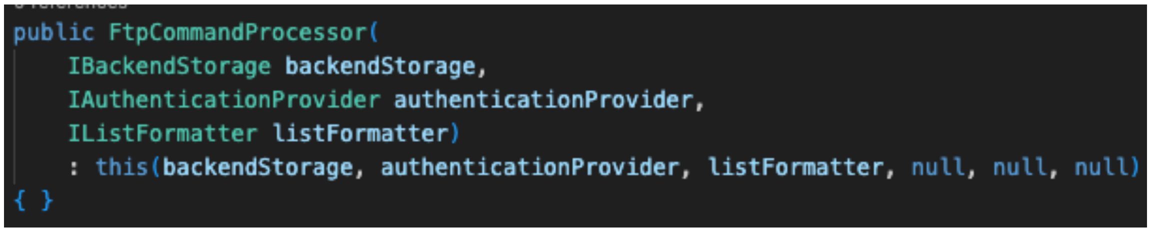
```
4 references
public FtpCommandProcessor(
    IBackendStorage backendStorage,
    IAuthenticationProvider authenticationProvider,
    IListFormatter listFormatter,
    FtpServerOptions options)
    : this(backendStorage, authenticationProvider, listFormatter, options, null, null)
{ }
```

Initializes the processor without a logger or authorizer.



```
0 references
public FtpCommandProcessor(
    IBackendStorage backendStorage,
    IAuthenticationProvider authenticationProvider,
    IListFormatter listFormatter,
    ILogger<IAsyncFtpCommandProcessor> logger)
    : this(backendStorage, authenticationProvider, listFormatter, null, logger, null)
{ }
```

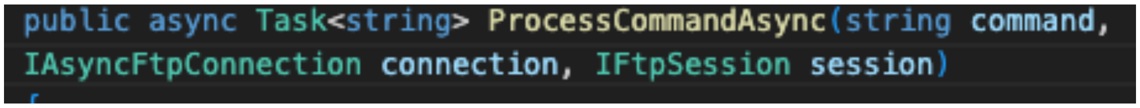
Initializes the processor with logger only, no options or authorizer.



```
public FtpCommandProcessor(
    IBackendStorage backendStorage,
    IAuthenticationProvider authenticationProvider,
    IListFormatter listFormatter)
    : this(backendStorage, authenticationProvider, listFormatter, null, null, null)
{ }
```

Initializes a new instance of the FTP command processor with only required dependencies. (No options, logger or authorizer).

Methods



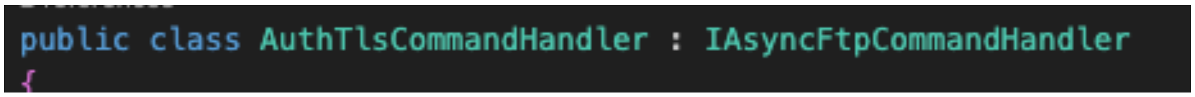
```
public async Task<string> ProcessCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
```

Receives a raw FTP command string from the client, determines the correct handler using the FtpCommandHandlerFactory, and delegates the execution to that handler.

Throws:

- ArgumentException if the command is null or empty.
- NotSupportedException if no handler matches the command name.

AuthTlsCommandHandler



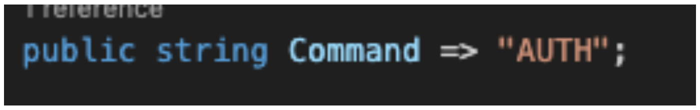
```
public class AuthTlsCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

The AuthTlsCommandHandler class handles the FTP AUTH TLS command, which initiates the process of upgrading an unencrypted FTP connection to a TLS-secured one. It checks whether TLS is enabled on the server, and then attempts to upgrade the current

TcpFtpConnection to use an encryption. This handler implements the IAsyncFtpCommandHandler interface and is triggered when the "AUTH" command is received.

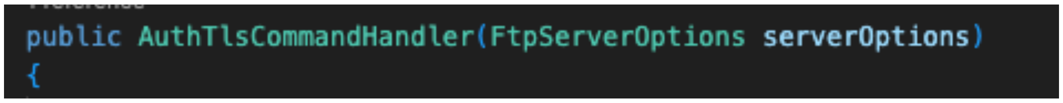
Properties



```
public string Command => "AUTH";
```

Specifies that this handler processes the "AUTH" command, used to initiate TLS negotiation in FTP.

Constructors



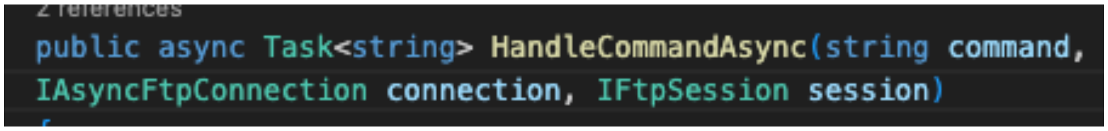
```
public AuthTlsCommandHandler(FtpServerOptions serverOptions)
{
```

Initializes the handler with the current server configuration, which is used to check whether TLS is enabled.

Throws:

- ArgumentNullException if server optinos is null.

Methods



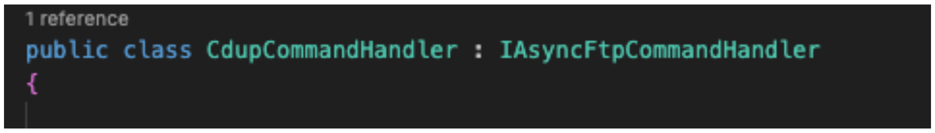
```
public async Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
```

Processes the AUTH TLS command to upgrade the current FTP connection to use TLS encryption.

Throws:

- InvalidOperationException if the connection does not support TLS upgrades.

CdupCommandHandler

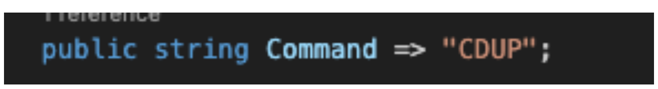


```
1 reference
public class CdupCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

The CdupCommandHandler class implements the FTP CDUP command, which moves the current working directory one level up in the directory.It checks for authentication, prevents navigation above the root directory, and updates the session’s working directory accordingly.

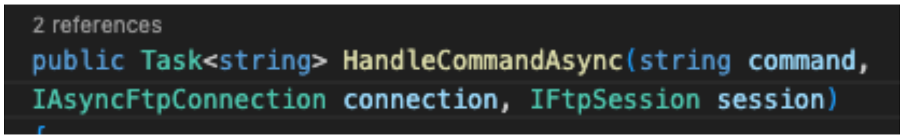
Properties



```
1 reference
public string Command => "CDUP";
```

Indicates that this handler is responsible for the "CDUP" command.

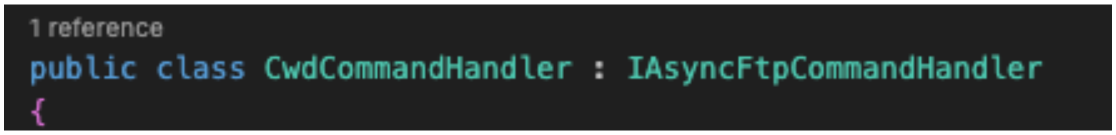
Methods



```
2 references
public Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
```

Handles the CDUP command by attempting to move up one level from the current directory.

CwdCommandHandler

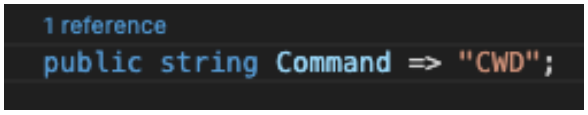


```
1 reference
public class CwdCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

The CwdCommandHandler class handles the FTP CWD (Change Working Directory) command, allowing the client to navigate into a different directory within their session. It updates the current directory if the command is valid.

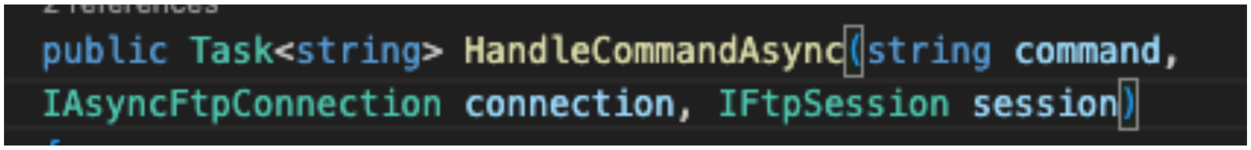
Properties



```
1 reference
public string Command => "CWD";
```

Indicates that this handler is responsible for processing the CWD FTP command.

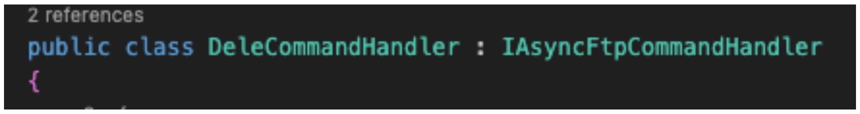
Methods



```
2 references
public Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
```

Processes the CWD command by changing the sessions current directory to the one specified by the client.

DeleCommandHandler

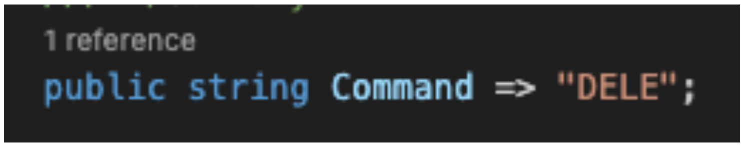


```
2 references
public class DeleCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

The DeleCommandHandler class handles the FTP DELE command, which instructs the server to delete a specific file from the storage backend. It validates the user authentication status, checks the syntax of the command, and attempts to remove the target file using the configured IBackendStorage implementation.

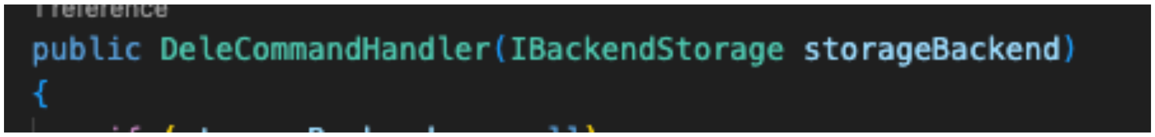
Properties



```
1 reference
public string Command => "DELE";
```

Indicates that this handler processes the DELE FTP command for file deletion.

Constructors



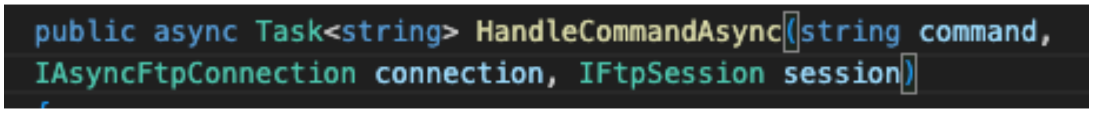
```
1 reference
public DeleCommandHandler(IBackendStorage storageBackend)
{
```

Initializes the handler with a storage backend that is responsible for performing file deletions.

Throws:

- ArgumentNullException if storageBackend is nulll.

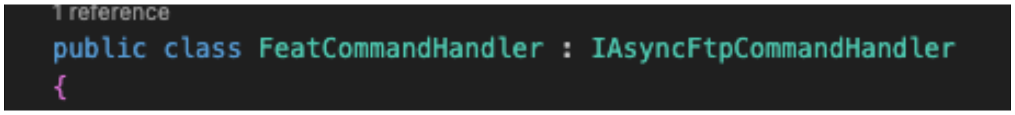
Methods



```
public async Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
```

Processes the DELE command by extracting the target file path, validating user state and input, and instructing the backend to delete the file.

FeatCommandHandler

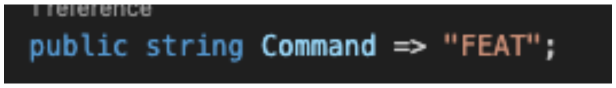


```
1 reference
public class FeatCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

The FeatCommandHandler class implements support for the FTP FEAT command, which asks the server to list all extended features it supports. This handler returns a fixed list of features.

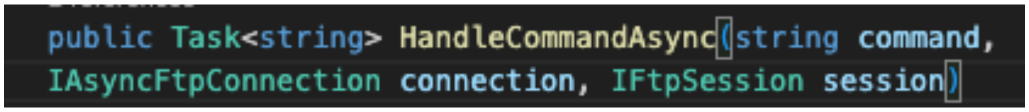
Properties



```
public string Command => "FEAT";
```

Specifies that this handler is responsible for processing the FEAT FTP command.

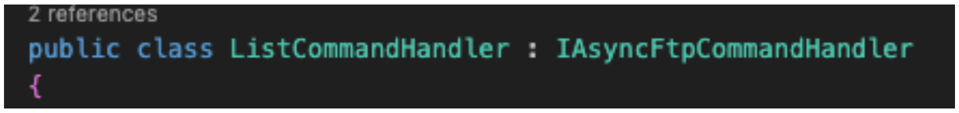
Methods



```
public Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
```

Returns a string listing the server’s supported FTP extensions.

ListCommandHandler

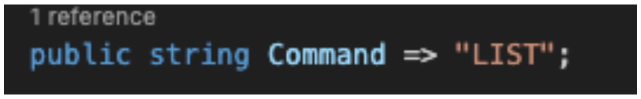


```
2 references
public class ListCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

The ListCommandHandler class implements the FTP LIST command, which retrieves a directory listing from the server and sends it back to client.

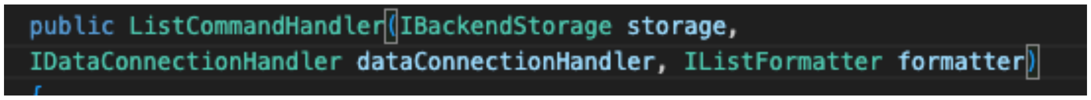
Properties



```
1 reference
public string Command => "LIST";
```

Specifies that this handler processes the LIST FTP command

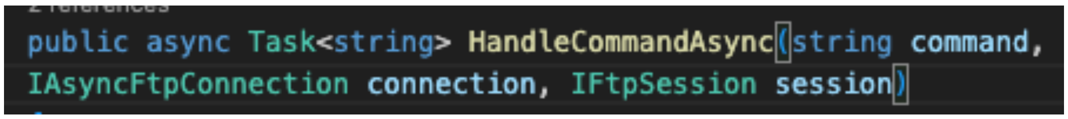
Constructor



```
public ListCommandHandler(IBackendStorage storage,
IDataConnectionHandler dataConnectionHandler, IListFormatter formatter)
```

Initializes the handler with a backend storage, a data connection handler and a list formatter for converting file metadata into string listings.

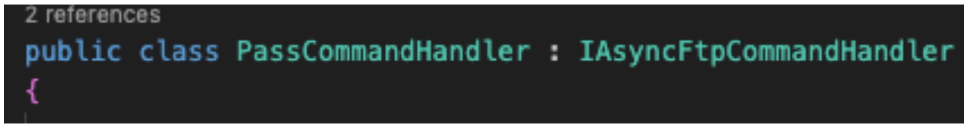
Methods



```
public async Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
```

Retrieves the contents of the current working directory and sends the listing to the client over a data connection.

PassCommandHandler

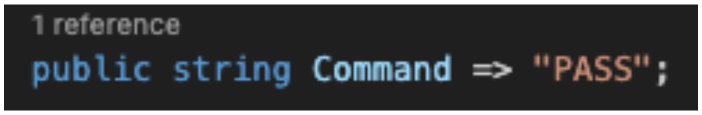


```
2 references
public class PassCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

The PassCommandHandler class implements the FTP PASS command, which is used to submit a password after the client has provided a username via the USER command. It authenticates the user using an IAuthenticationProvider and updates the session state accordingly.

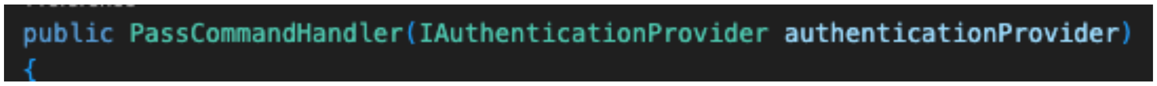
Properties



```
1 reference
public string Command => "PASS";
```

Indicates that this handler is responsible for processing the PASS command.

Constructor



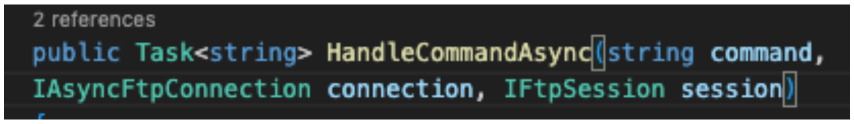
```
public PassCommandHandler(IAuthenticationProvider authenticationProvider)
{
```

Initializes the handler with the authentication provider that verifies user credentials.

Throws:

- ArgumentNullException if authenticationPRovider is null.

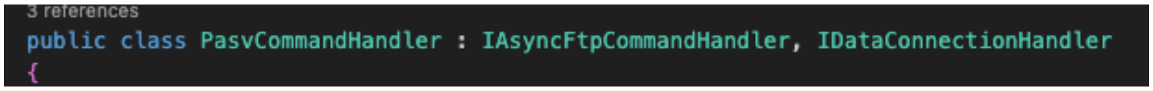
Methods



```
2 references
public Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
```

Processes the PASS command by verifying the client-provided password using the IAuthenticationProvider.

PasvCommandHandler

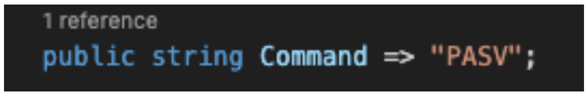


```
3 references
public class PasvCommandHandler : IAsyncFtpCommandHandler, IDataConnectionHandler
{
```

Summary:

The PasvCommandHandler class handles the FTP PASV command, which switches the server into passive mode for data transfers. In passive mode, the server listens for a client connection on a new data port and informs the client where to connect. This class also implements the IDataConnectionHandler interface, providing mechanism for accepting and closing the data connection.

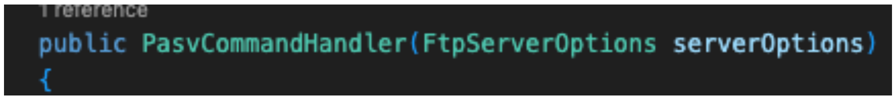
Properties



```
1 reference
public string Command => "PASV";
```

Specifies that this handler processes the "PASV" command.

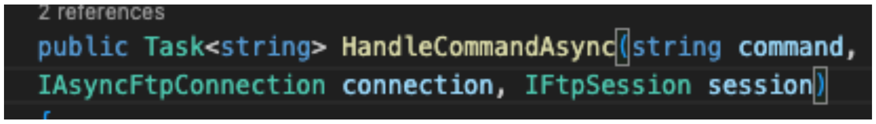
Constructor



```
1 reference
public PasvCommandHandler(FtpServerOptions serverOptions)
{
```

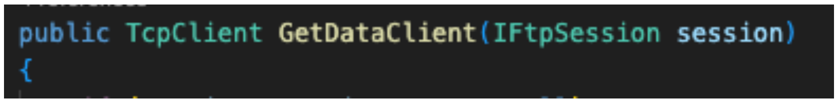
Initializes the handler with the server's configuration settings.

Methods



```
2 references
public Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
```

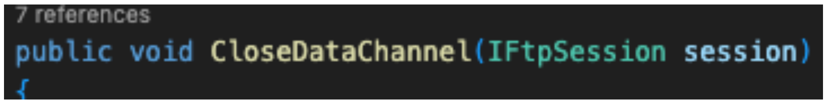
Processes the PASV command to establish a passive mode data connection.



```
public TcpClient GetDataClient(IFtpSession session)
{
```

Retrieves the TCP client for the passive data connection.

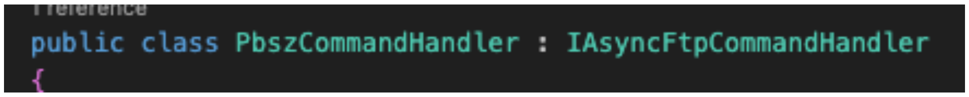
Throws: InvalidOperationException if passive mode has not been initialized.



```
7 references
public void CloseDataChannel(IFtpSession session)
{
```

Closes the passive data channel if it is open.

PbszCommandHandler

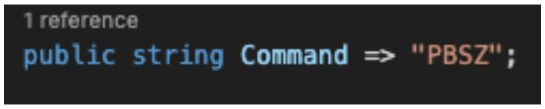


```
public class PbszCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

The PbszCommandHandler class handles the FTP PBSZ (Protection Buffer Size) command, which is used in FTPS (FTP over TLS) to set the buffer size for secure data transfers. This handler checks authentication status and responds witch a success message.

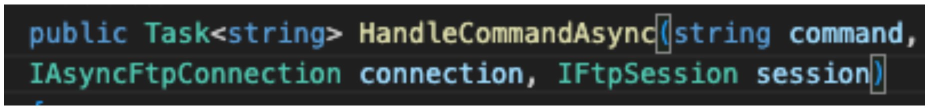
Properties



```
1 reference
public string Command => "PBSZ";
```

Specifies that this handler processes the PBSZ FTP command.

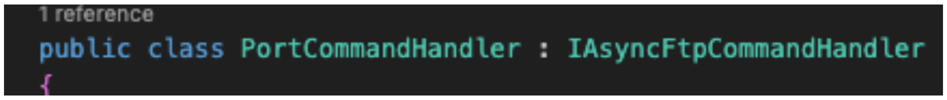
Methods



```
public Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
```

Processes the PBSZ command, returning a success response if the user is authenticated.

PortCommandHandler

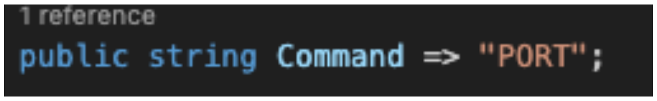


```
1 reference
public class PortCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

The PortCommandHandler class implements the FTP PORT command, which is used to initiate an active-mode data connection by providing the client's IP address and port.

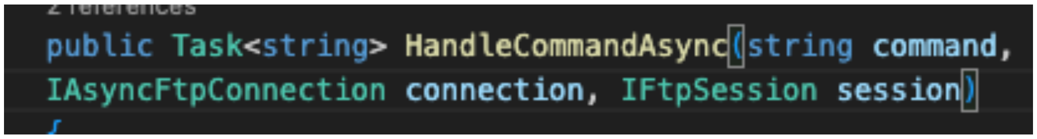
Properties



```
1 reference
public string Command => "PORT";
```

Indicates that this handler is responsible for processing the "PORT" FTP command.

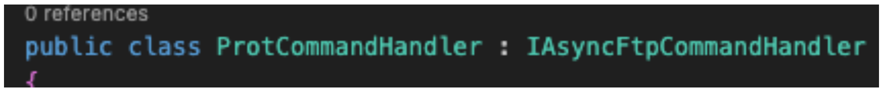
Methods



```
2 references
public Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
{
```

Processes the PORT command to set up an active mode data connection.

ProtCommandHandler

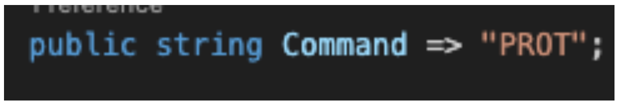


```
0 references
public class ProtCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

The ProtCommandHandler class handles the FTP PROT command, which specifies the desired level of protection for data transfers when using FTPS (FTP over TLS).

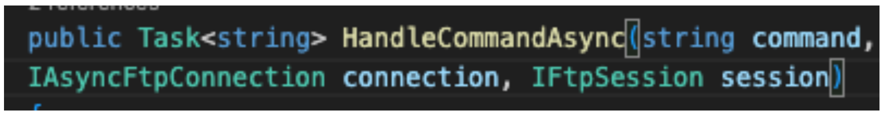
Properties



```
public string Command => "PROT";
```

specifies that this handler is responsible for processing the "PROT" FTP command.

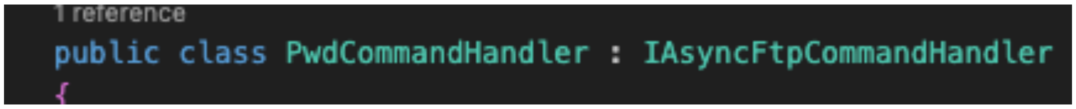
Methods



```
public Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
```

Processes the PROT command to set the data protection level.

PwdCommandHandler

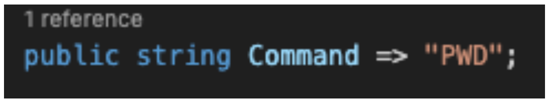


```
1 reference
public class PwdCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

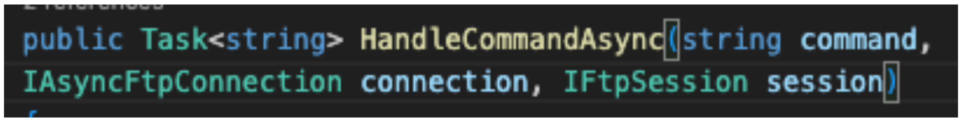
The PwdCommandHandler class implements the FTP PWD (Print Working Directory) command. It returns the client's current working directory as maintained in the session state. This command is used by clients to confirm their current location in the server's directory.

Properties



```
1 reference
public string Command => "PWD";
```

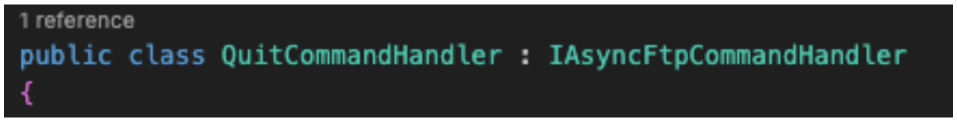
Methods



```
public Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
{
```

Processes the PWD command by returning the session's current working directory in the expected FTP format.

QuitCommandHandler

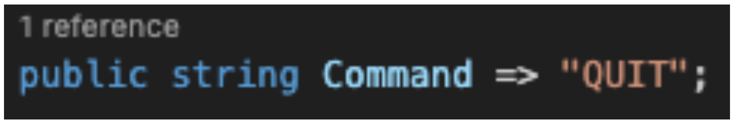


```
1 reference
public class QuitCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

The QuitCommandHandler class implements the FTP QUIT command, which signals the client’s intent to terminate the session. It sends a goodbye message to the client and closes the active connection, providing graceful shutdown of the FTP session.

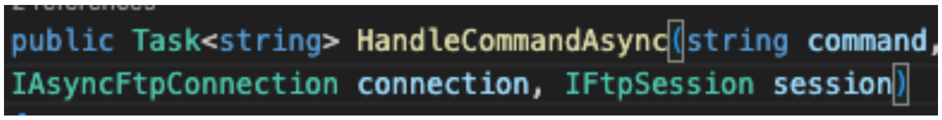
Properties



```
1 reference
public string Command => "QUIT";
```

Indicates that this handler processes the "QUIT" FTP command.

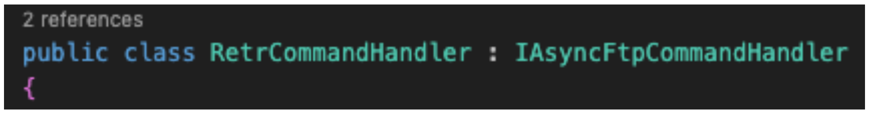
Methods



```
public Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
```

Handles the QUIT command by sending a goodbye response and initiating the connection shutdown.

RetrCommandHandler

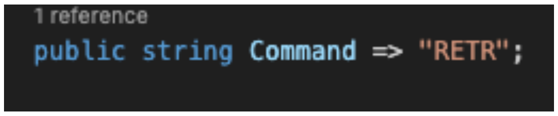


```
2 references
public class RetrCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

The RetrCommandHandler class handles the FTP RETR command, which allows a client to retrieve (download) a file from the server. It retrieves the requested file from the backend storage and transmits it over a data connection.

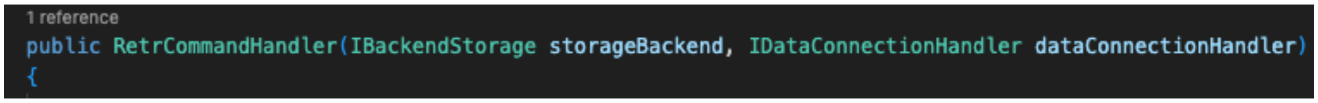
Properties



```
1 reference
public string Command => "RETR";
```

Specifies that this handler is responsible for the RETR FTP command.

Constructor



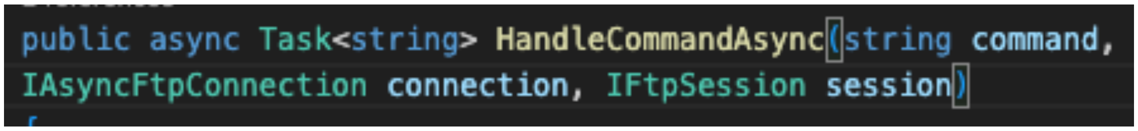
```
1 reference
public RetrCommandHandler(IBackendStorage storageBackend, IDataConnectionHandler dataConnectionHandler)
{
```

Initializes a new instance of the RetrCommandHandler class. Does so with IBackendStorage for accessing file contents and an IDataConnectionHandler for sending over the file.

Throws:

- ArgumentException if any dependency is null.

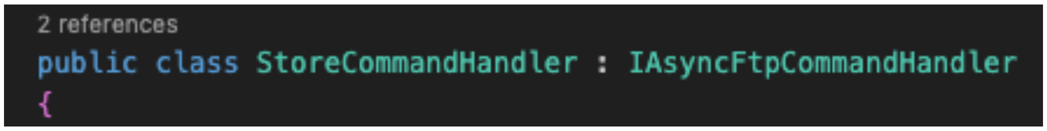
Methods



```
public async Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
```

Processes the RETR command to retrieve a file and send it over the data connection.

StoreCommandHandler

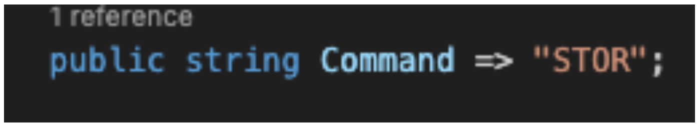


```
2 references
public class StoreCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

The StoreCommandHandler class handles the FTP STOR command, which allows a client to upload a file to the server.

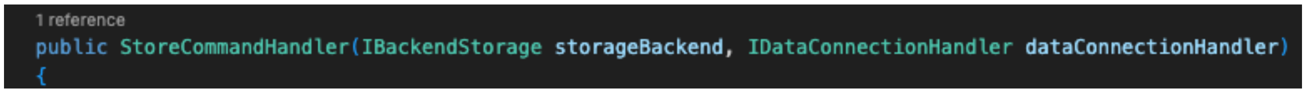
Properties



```
1 reference
public string Command => "STOR";
```

Specifying that the handler is responsible for processing the STOR command.

Constructor



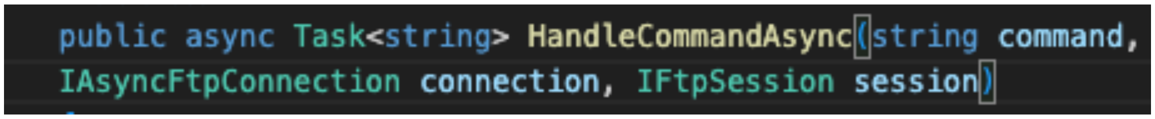
```
1 reference
public StoreCommandHandler(IBackendStorage storageBackend, IDataConnectionHandler dataConnectionHandler)
{
```

Initializes a new instance of the StoreCommandHandler class.

Throws:

- ArgumentNullException if either parameter is null.

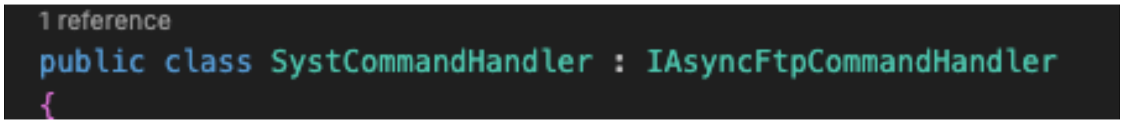
Methods



```
public async Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
```

Processes the STOR command by receiving a file over a data connection and writing it to the backend.

SystCommandHandler

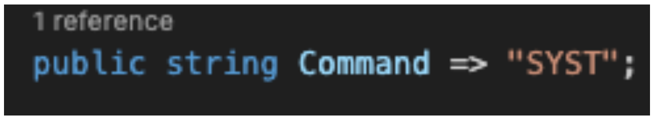


```
1 reference
public class SystCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

The SystCommandHandler class implements the FTP SYST command, which is used by clients to request the system type of the FTP server. The server responds with a standardized string that helps the client determine compatibility and behavior expectations.

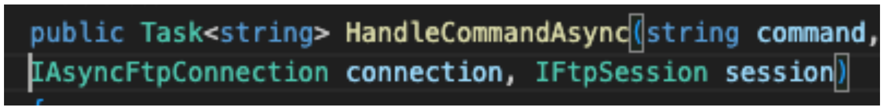
Properties



```
1 reference
public string Command => "SYST";
```

Specifying that this handler processes the SYST FTP command.

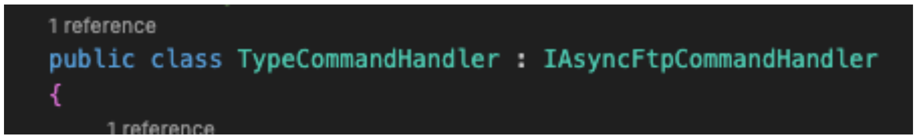
Methods



```
public Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
```

Processes the SYST command to return the server's system type.

TypeCommandHandler

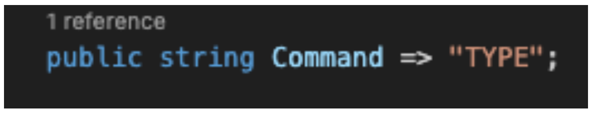


```
1 reference
public class TypeCommandHandler : IAsyncFtpCommandHandler
{
        1 reference
```

Summary:

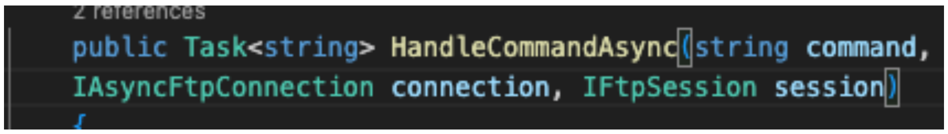
The TypeCommandHandler class handles the FTP TYPE command, which sets the file transfer mode for the current session. FTP supports at least two types: ASCII (A) and Binary (I).

Properties



```
1 reference
public string Command => "TYPE";
```

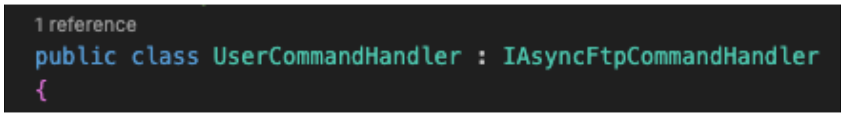
Methods



```
2 references
public Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
{
```

Processes the TYPE command to set the file transfer type.

UserCommandHandler

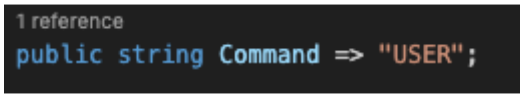


```
1 reference
public class UserCommandHandler : IAsyncFtpCommandHandler
{
```

Summary:

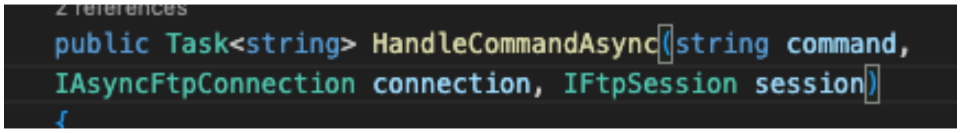
The UserCommandHandler class handles the FTP USER command, which sets the username for the current FTP session and signals that a password must be provided next. It handles input validation and updates session state but does not perform authentication, which that is completed later by the PASS command.

Properties



```
1 reference
public string Command => "USER";
```

Methods

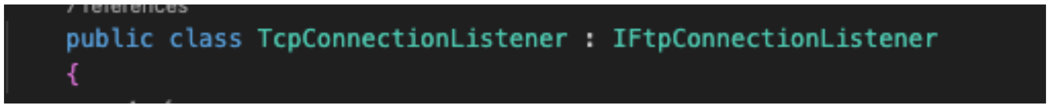


```
2 references
public Task<string> HandleCommandAsync(string command,
IAsyncFtpConnection connection, IFtpSession session)
{
```

Processes the USER command to set the username and prepare for password authentication.

Networking & Connections

TcpConnectionListener



```
7 references
public class TcpConnectionListener : IFtpConnectionListener
{
```

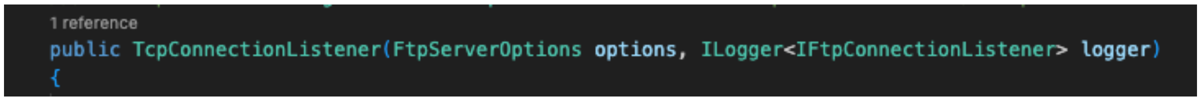
Summary:

The TcpConnectionListener class is responsible for listening to incoming FTP client connections over TCP. It uses the built-in TcpListener from .NET to and supports optional TLS encryption .

Its responsibilities include:

- Starting and stopping the underlying TCP listener
- Accepting incoming connections and validating TLS configuration
- Logging lifecycle events and socket errors
- Creating secure or plaintext FTP connections via TcpFtpConnection

Constructor:



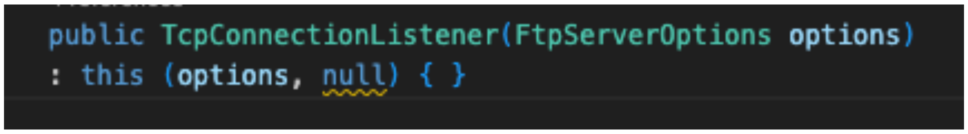
```
1 reference
public TcpConnectionListener(FtpServerOptions options, ILogger<IFtpConnectionListener> logger)
{
```

Initializes the TCP listener using the provided FtpServerOptions for network configuration and a logger instance for diagnostics.

Throws:

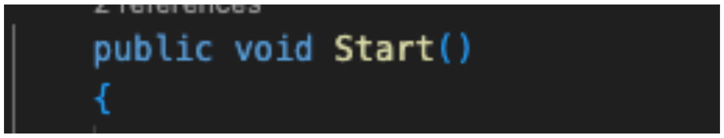
- ArgumentNullException if options is null.

Methods



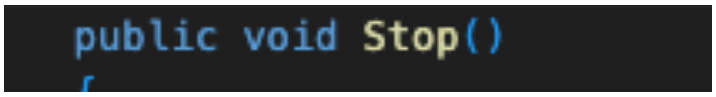
```
public TcpConnectionListener(FtpServerOptions options)
: this (options, null) { }
```

Provides a way to instantiate the TcpConnectionListener without the need to provde a logger. Not all users may want or need logging, making the class easier to use in simple scenarios.



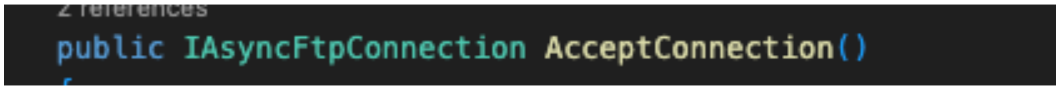
```
public void Start()
{
```

Starts the listener, listening for incoming FTP client connections. Logs the startup with the configured IP and port.



```
public void Stop()
```

Stops the TCP listener and prevents any further connections from being accepted. Also logs the shutdown of the listener.

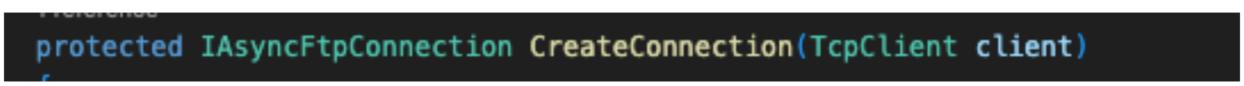


```
public IAsyncFtpConnection AcceptConnection()
```

Waits for a client to connect and returns a new IAsyncFtpConnection. Performs validation of TLS settings and throws an InvalidOperationException if:

- The listener is not running.
- TLS is enabled but no certificate is provided.

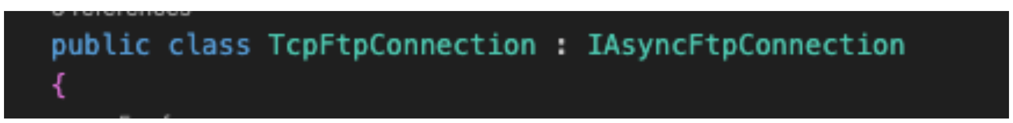
Also logs any errors related to sockets or unexpected errors during the accept process.



```
protected IAsyncFtpConnection CreateConnection(TcpClient client)
```

Creates a new TcpFtpConnection based on whether TLS is enabled in the configuration. This method is reused internally by AcceptConnecton().

TcpFtpConnection



```
public class TcpFtpConnection : IAsyncFtpConnection
{
```

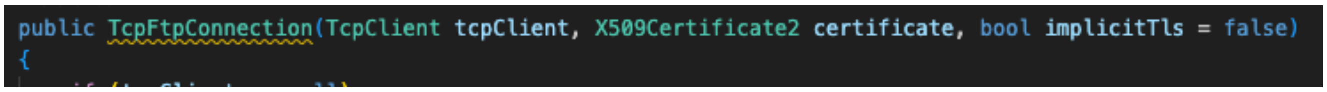
Summary:

The TcpFtpConnection class implements an FTP connection with optional support for TLS encryption. It acts as the communication layer between the FTP server and the client, providing asynchronous methods for reading commands and sending responses. It supports both implicit TLS (enabled immediately on connection) and explicit TLS (activated later via AUTH TLS command).

Its main responsibilities include:

- Reading FTP commands from the client.
- Sendiung responses back to the client.
- Upgrading to TLS if configured.
- Managing the connections lifetime and cleanup.

Constructors:



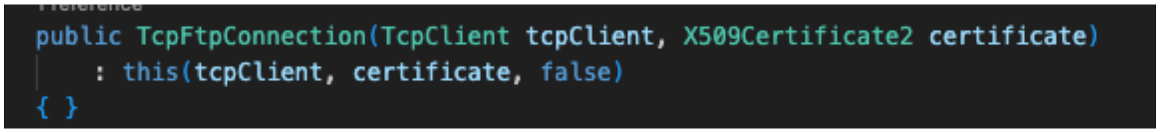
```
public TcpFtpConnection(TcpClient tcpClient, X509Certificate2 certificate, bool implicitTls = false)
{
```

Initializes a new FTP connection:

IF a TLS certificate is provided and implicitTls is true, the connection is secured immediately. If implicitTls is false, TLS can be enabled later via UpgradeToTlsAsync.

Throws:

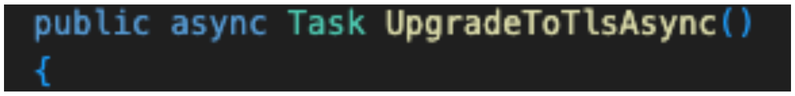
- ArgumentNullException if the tcpClient is null.



```
public TcpFtpConnection(TcpClient tcpClient, X509Certificate2 certificate)
    : this(tcpClient, certificate, false)
{ }
```

Initializes a nes instance of the TcpFtpConnection class with a TLS certificate.

Methods



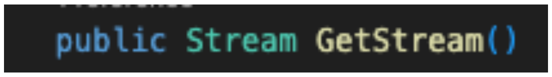
```
public async Task UpgradeToTlsAsync()
{
```

Upgrades the current connection to TLS encryption after the connection has started, usually triggered by an AUTH TLS command from the client.

Validates that a certificate is configured and TLS is not already active.

Throws:

- InvalidOperationException if TLS is not configured or already active.

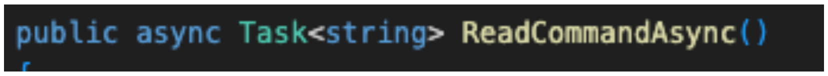


```
public Stream GetStream()
```

Returns the clients' stream for the data transfer.

Throws_:

- ObjectDisposedException if the connection has been disposed already.

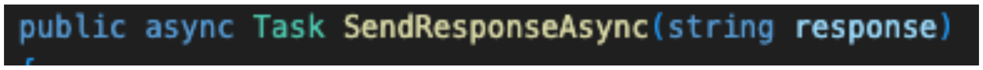


```
public async Task<string> ReadCommandAsync()
```

Asynchronously reads a line of text (an FTP command) from the client. Returns null if the connection is closed.

Throws:

- ObjectDisposedException if the connection is already disposed
- InvalidOperationException if there's a network error during read

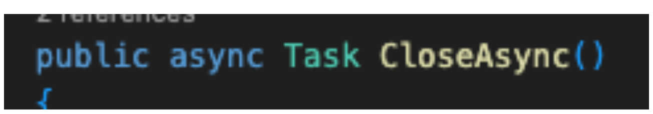


```
public async Task SendResponseAsync(string response)
{
```

Asynchronously sends a string response to the FTP client.

Throws:

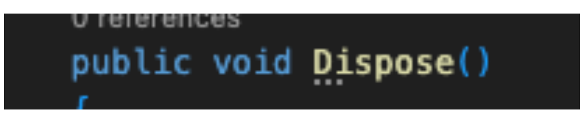
- ArgumentException if response is null or empty
- ObjectdisposedExpcetion if connection has been disposed.



```
2 references
public async Task CloseAsync()
{
```

Gracefully closes the connection:

- If using TLS, attempts to shut down the SslStream
- Closes the TcpClient
- Marks the connection as disposed

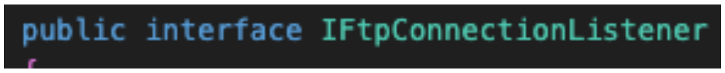


```
0 references
public void Dispose()
{
```

Releases all resources:

- Disposes the writer, reader, stream, and TCP client
- Marks the connection as disposed
- Can be called multiple times safely

IFtpConnectionListener (Interface)

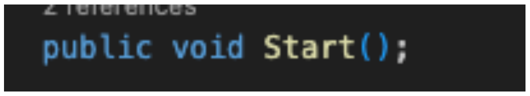


```
public interface IFtpConnectionListener
```

Summary:

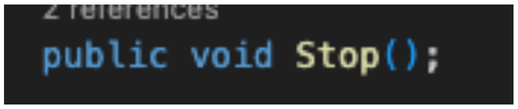
The IFtpConnectionListener interface defines the contract for accepting an incoming FTP client connection over a network.

Methods



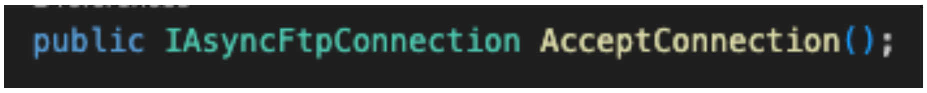
```
2 references
public void Start();
```

Begins listening for incoming FTP client connections.



```
2 references
public void Stop();
```

Stops listening for incoming connections.



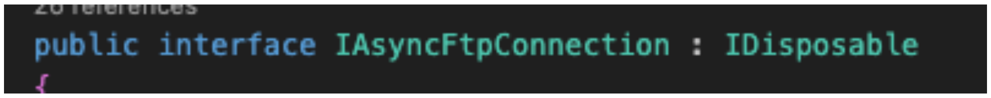
```
public IAsyncFtpConnection AcceptConnection();
```

Accepts a client connection and wraps it in a IAsyncFtpConnection for further use by the server.

Throws:

- InvalidOperationException of the listener is not currently running.

IAsyncFtpConnection



```
public interface IAsyncFtpConnection : IDisposable
{
```

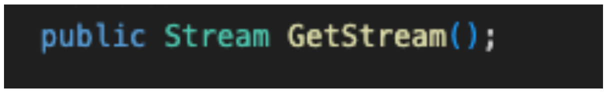
Summary:

The IAsyncFtpConnection interface defines the contract for an asynchronous FTP connection to a client. It abstracts how commands and responses are exchanged over the network, whether through a plain TCP stream or an encrypted TLS stream.

It supports:

- Reading FTP commands asynchronously.
- Sending responses.
- Handling the raw data stream.
- Managing graceful connection shutdown.

Methods

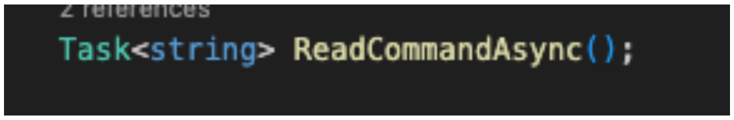


```
public Stream GetStream();
```

Returns the underlying stream used for reading or writing data to the connected client.This stream may be a plain network stream or an encrypted SslStream.

Throws:

- ObjectDisposedException if the connection has been disposed



```
Task<string> ReadCommandAsync();
```

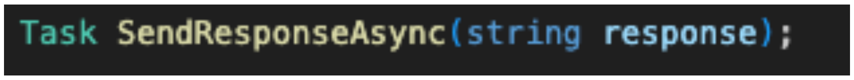
Reads a command sent by the client, asynchronously.

Returns a Task<string> resolving to the received FTP command.

Returns null if the connection is closed

Throws:

- ObjectDisposedException if the connection has been disposed.
- InvalidOperationException if a network error occurs during read.

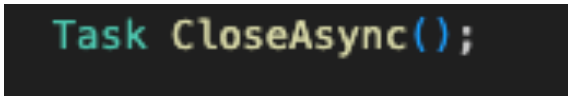


```
Task SendResponseAsync(string response);
```

Parameters: response: The text to send as an FTP respinse.

Throws:

- ArgumentException if the response is null or empty.
- ObjectDisposedException if the connection has been disposed.
- InvalidOperationException if a network error occurs while sending.

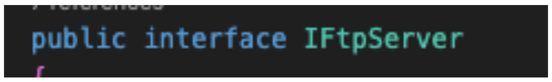


```
Task CloseAsync();
```

Closing the connection asynchronously and releases all related resources.

Sessions & Server Control

IFtpServer (Interface)

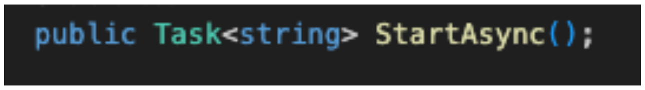


```
public interface IFtpServer
```

Summary:

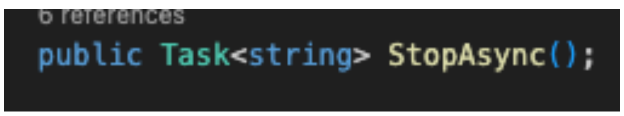
The IFtpServer interface defines the contract for an FTP server instance that can be started and stopped asynchronously. The interface is implemented by classes like FtpServer, which coordinate the entire lifecycle of the server process.

Methods



```
public Task<string> StartAsync();
```

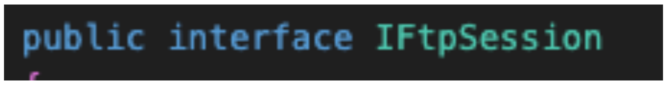
Starts the FTP server asynchronously, preparing it to accept incoming connections and handle client sessions.



```
6 references
public Task<string> StopAsync();
```

Stops the FTP server asynchronously, disconnecting clients and releasing resources.

IFtpSession (Interface)

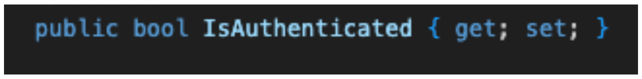


```
public interface IFtpSession
```

Summary:

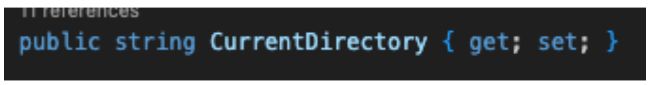
The IFtpSession interface defines the structure and state of an active FTP session between a client and the server. It provides access to session-specific data such as authentication status, current working directory, transfer settings, and associated backend storage.

Properties



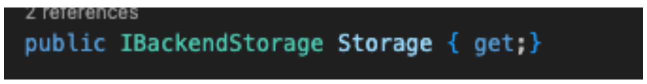
```
public bool IsAuthenticated { get; set; }
```

Indicates whether the client has successfully completed authentication.



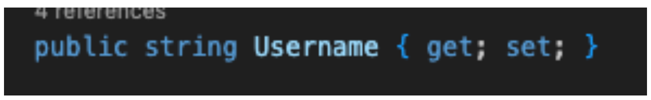
```
11 references
public string CurrentDirectory { get; set; }
```

Tracks the current working directory of the session.



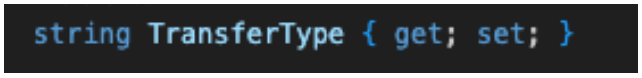
```
2 references
public IBackendStorage Storage { get;}
```

Provides access to the storage backend responsible for handling file operations during the session.



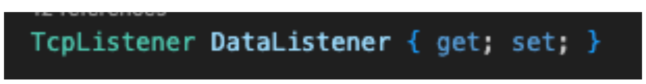
```
4 references
public string Username { get; set; }
```

Stores the username provided by the client during login.



```
string TransferType { get; set; }
```

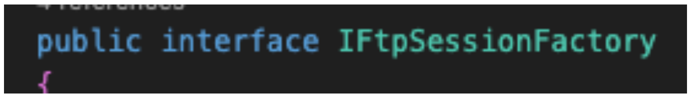
Represents the current transfer type mode for the session. ASCII or binary.



```
TcpListener DataListener { get; set; }
```

Holds the TCP listener used for passive (PASV) mode data transfers.

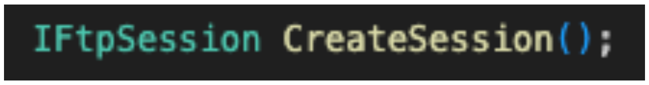
IFtpSessionFactory (Interface)


```
public interface IFtpSessionFactory
{
```

Summary:

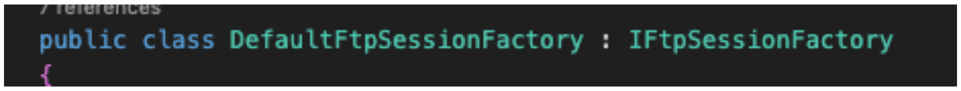
The IFtpSessionFactory interface defines a factory contract for creating new FTP session instances. It is designed to allow the FTP server to generate an isolated session object (IFtpSession) for each incoming client connection.

Methods


```
IFtpSession CreateSession();
```

Creates and returns a new FTP session instance that will be used to track the state of a connected client.

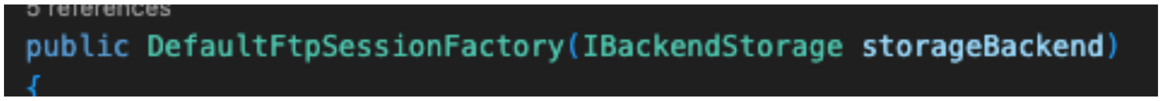
DefaultFTPSessionFactory


```
public class DefaultFtpSessionFactory : IFtpSessionFactory
{
```

Summary:

The DefaultFtpSessionFactory class provides a standard implementation of the IFtpSessionFactory interface. It creates new FTP sessions (FtpSession) using a predefined or custom backend storage system. This class enables the server to generate isolated session instances for each client. If no backend is specified, the factory defaults to using LocalFileStorage.

Constructors

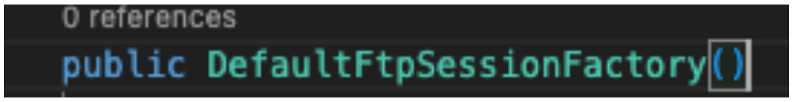


```
5 references
public DefaultFtpSessionFactory(IBackendStorage storageBackend)
{
```

Initializes the factory with a custom IBackendStorage implementation.

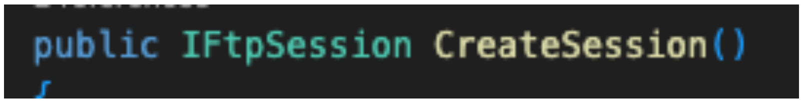
Throws:

- ArgumentNullException if storageBackend is null



```
0 references
public DefaultFtpSessionFactory()
```

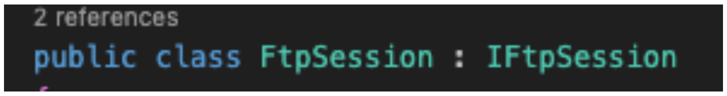
Creates a new session factory using LocalFileStorage as the default backend. Useful for local FTP servers without any extra setup.



```
public IFtpSession CreateSession()
{
```

Creates and returns a new FtpSession instance using the factory's configured backend storage.

FtpSession

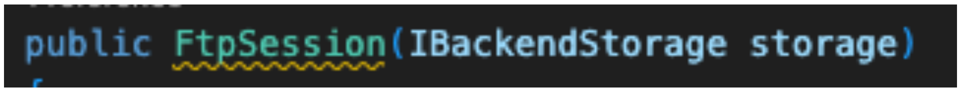


```
2 references
public class FtpSession : IFtpSession
```

Summary:

The FtpSession class implements the IFtpSession interface and represents an active session for a connectedd FTP client. It tracks session-specific information such as the clients authentication state, current directory, transfer mode, and data connection listener. This class also provides access to the configured storage backend, enabling file operations throughout the session.

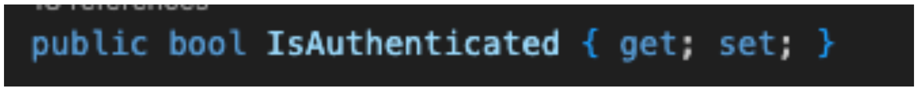
Constructor



```
public FtpSession(IBackendStorage storage)
```

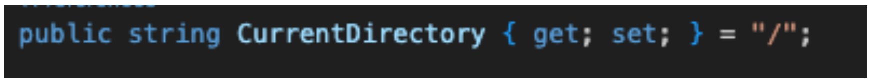
Initializes a new FTP session using the specified backend storage.

Properties



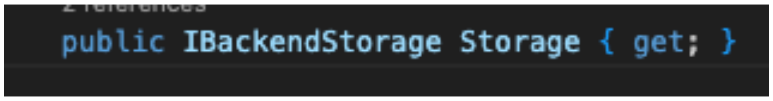
```
public bool IsAuthenticated { get; set; }
```

Indicates whether the user is sucessfylly authenticated.



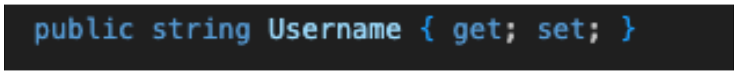
```
public string CurrentDirectory { get; set; } = "/";
```

Represents the current working directory of the session. Defaults to the root ("/") upon session creation.



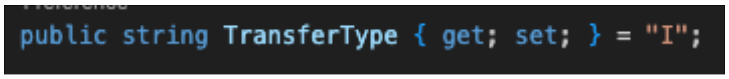
```
public IBackendStorage Storage { get; }
```

Provides access to the sessions backend storage system for file operations like LIST, RETR, and STOR.



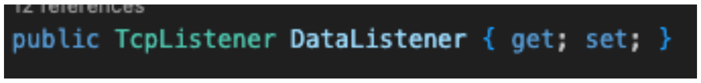
```
public string Username { get; set; }
```

Stores the authenticated username for the current session.



```
public string TransferType { get; set; } = "I";
```

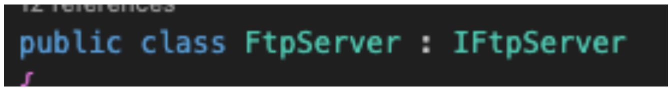
Represents the file transfer mode, I for binary which is set to default.



```
public TcpListener DataListener { get; set; }
```

Represents the data listener for passive mode (PASV).

FtpServer



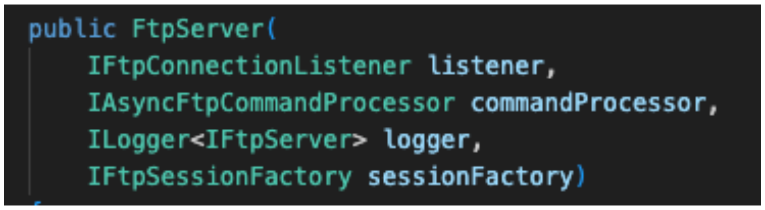
```
public class FtpServer : IFtpServer
```

The FtpServer class represents the core FTP server engine responsible for managing client connections, handling session creation, and processing incoming FTP commands. It serves as the high-level coordinator of all server components, including the connection listener, command processor, session factory, and logging.

This class supports multiple constructors for flexibility:

- Full DI-based (Dependency injectuion) initialization with all dependencies provided
- Simplified configuration with default components and options

Constructors

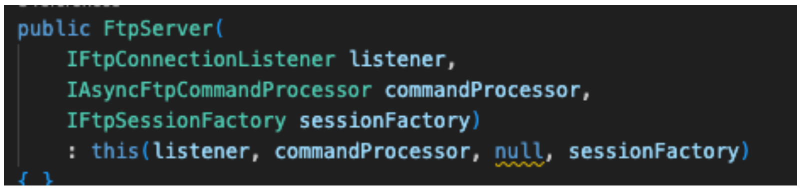


```
public FtpServer(
    IFtpConnectionListener listener,
    IAsyncFtpCommandProcessor commandProcessor,
    ILogger<IFtpServer> logger,
    IFtpSessionFactory sessionFactory)
```

Initializes a new instance of the FtpServer class with the specified components and an optional logger.

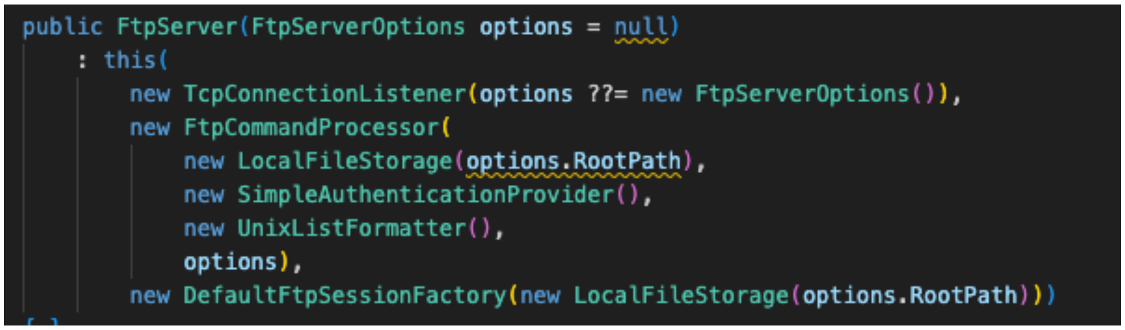
Throws:

- ArgumentNullException if any required dependency is null.



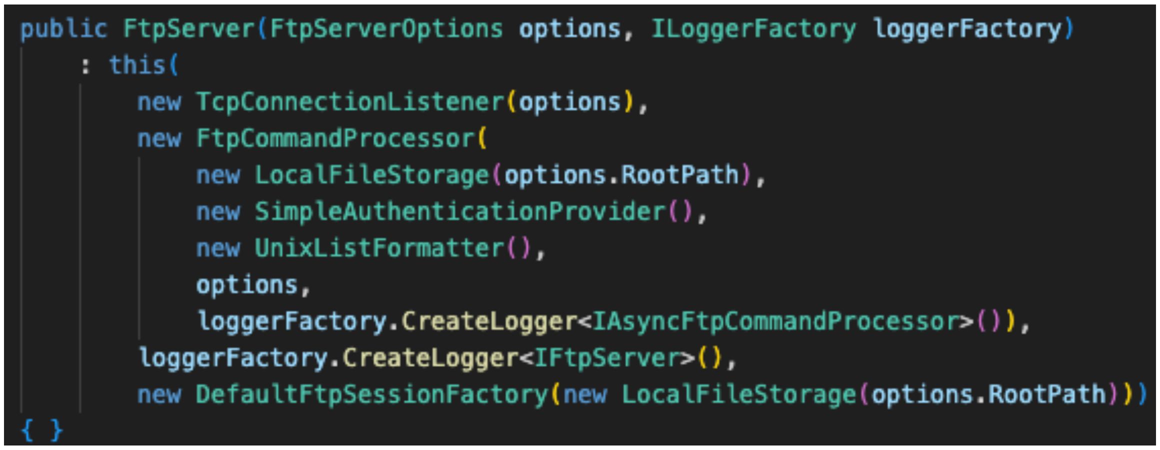
```
public FtpServer(
    IFtpConnectionListener listener,
    IAsyncFtpCommandProcessor commandProcessor,
    IFtpSessionFactory sessionFactory)
    : this(listener, commandProcessor, null, sessionFactory)
{ }
```

Initializes a new instance of the FtpServer class with the specified components, excluding logging support.



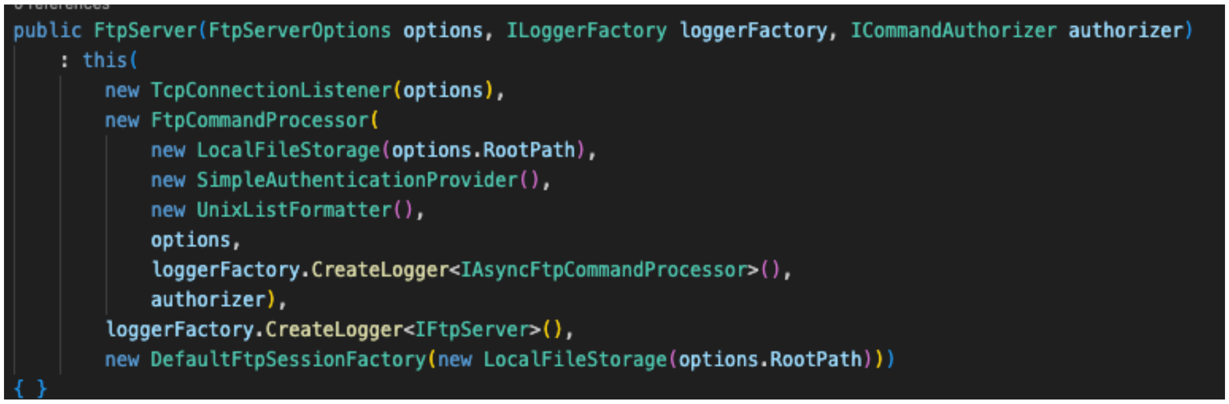
```
public FtpServer(FtpServerOptions options = null)
    : this(
        new TcpConnectionListener(options ??= new FtpServerOptions()),
        new FtpCommandProcessor(
            new LocalFileStorage(options.RootPath),
            new SimpleAuthenticationProvider(),
            new UnixListFormatter(),
            options),
        new DefaultFtpSessionFactory(new LocalFileStorage(options.RootPath)))
{ }
```

Initializes a new instance of the FtpServer class with default component implementations.



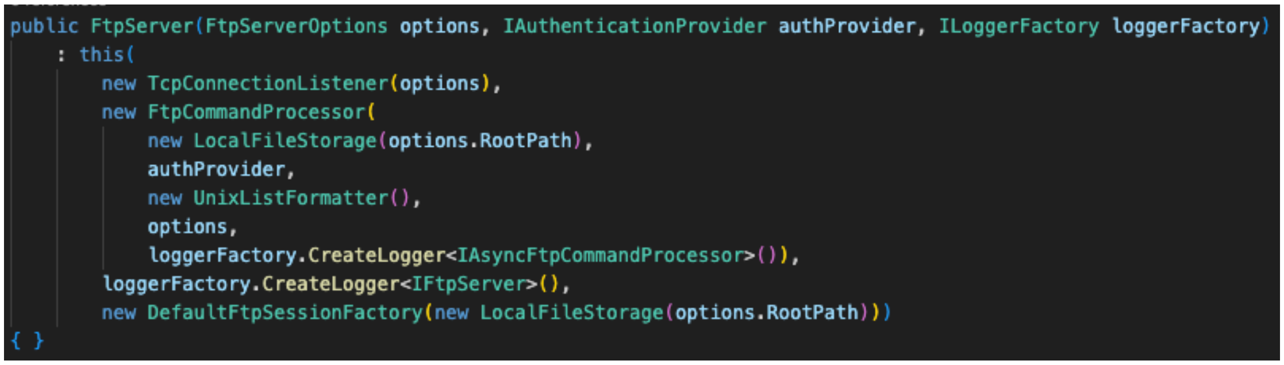
```
public FtpServer(FtpServerOptions options, ILoggerFactory loggerFactory)
    : this(
        new TcpConnectionListener(options),
        new FtpCommandProcessor(
            new LocalFileStorage(options.RootPath),
            new SimpleAuthenticationProvider(),
            new UnixListFormatter(),
            options,
            loggerFactory.CreateLogger<IAsyncFtpCommandProcessor>()),
        loggerFactory.CreateLogger<IFtpServer>(),
        new DefaultFtpSessionFactory(new LocalFileStorage(options.RootPath)))
{ }
```

Initializes a new instance of FTPServer using ILoggerFactory to create loggers.



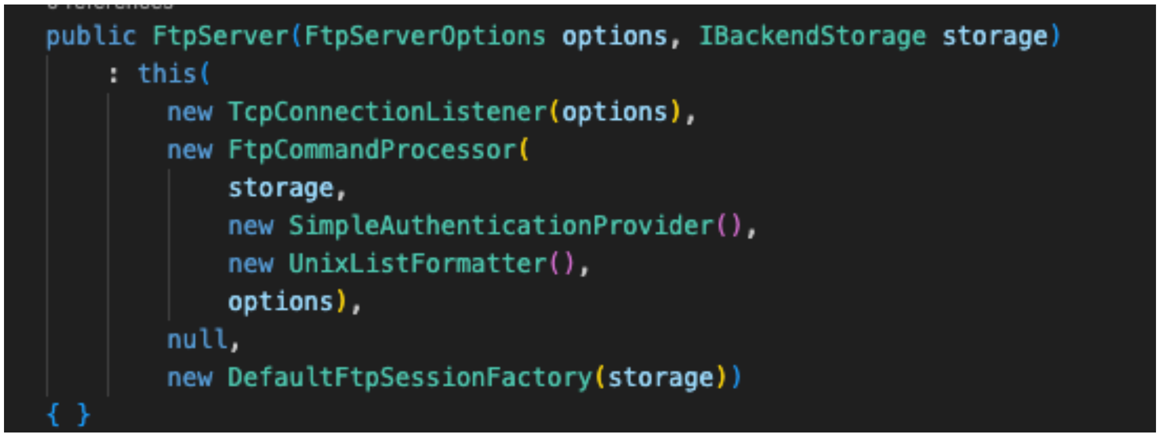
```
public FtpServer(FtpServerOptions options, ILoggerFactory loggerFactory, ICommandAuthorizer authorizer)
    : this(
        new TcpConnectionListener(options),
        new FtpCommandProcessor(
            new LocalFileStorage(options.RootPath),
            new SimpleAuthenticationProvider(),
            new UnixListFormatter(),
            options,
            loggerFactory.CreateLogger<IAsyncFtpCommandProcessor>(),
            authorizer),
        loggerFactory.CreateLogger<IFtpServer>(),
        new DefaultFtpSessionFactory(new LocalFileStorage(options.RootPath)))
{ }
```

Initializes a new instance of the FTPServer with logger factory and a command authorizer.



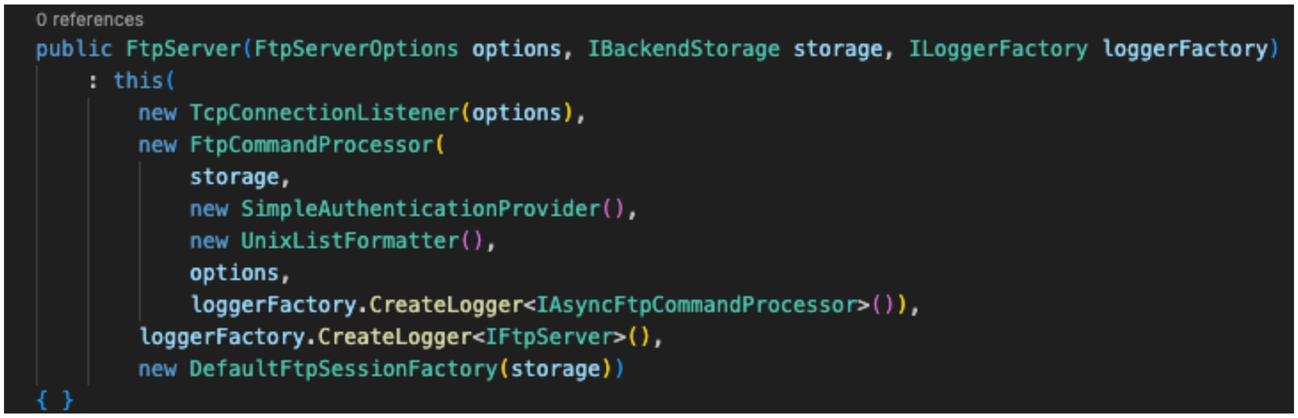
```
public FtpServer(FtpServerOptions options, IAuthenticationProvider authProvider, ILoggerFactory loggerFactory)
    : this(
        new TcpConnectionListener(options),
        new FtpCommandProcessor(
            new LocalFileStorage(options.RootPath),
            authProvider,
            new UnixListFormatter(),
            options,
            loggerFactory.CreateLogger<IAsyncFtpCommandProcessor>()),
        loggerFactory.CreateLogger<IFtpServer>(),
        new DefaultFtpSessionFactory(new LocalFileStorage(options.RootPath)))
{ }
```

Initializes a new instance of FTPServer with custom logger factory and authentication provider.



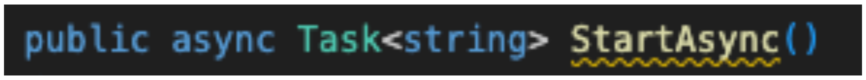
```
public FtpServer(FtpServerOptions options, IBackendStorage storage)
    : this(
        new TcpConnectionListener(options),
        new FtpCommandProcessor(
            storage,
            new SimpleAuthenticationProvider(),
            new UnixListFormatter(),
            options),
        null,
        new DefaultFtpSessionFactory(storage))
{ }
```

Initializes the FTPServer class with a custom backend storage.

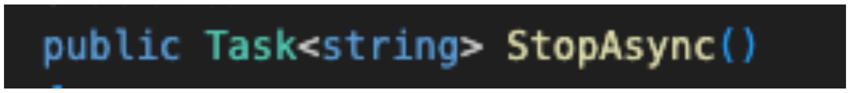

```
0 references
public FtpServer(FtpServerOptions options, IBackendStorage storage, ILoggerFactory loggerFactory)
    : this(
        new TcpConnectionListener(options),
        new FtpCommandProcessor(
            storage,
            new SimpleAuthenticationProvider(),
            new UnixListFormatter(),
            options,
            loggerFactory.CreateLogger<IAsyncFtpCommandProcessor>()),
        loggerFactory.CreateLogger<IFtpServer>(),
        new DefaultFtpSessionFactory(storage))
{ }
```

Initializes a new instance of the FTPServer class with a custom backend storage and logger factory.

Methods

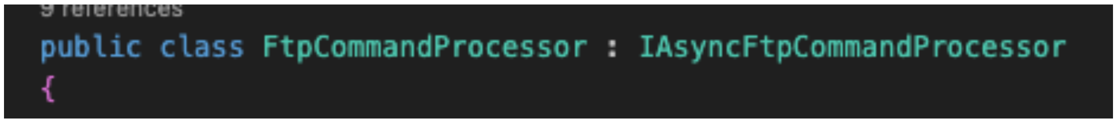

```
public async Task<string> StartAsync()
```

Starts the FTP server asynchronously and begins accepting client connections.


```
public Task<string> StopAsync()
```

Stops the FTP server asynchronously, ceasing to accept new connections.

FtpCommandProcessor


```
9 references
public class FtpCommandProcessor : IAsyncFtpCommandProcessor
{
```

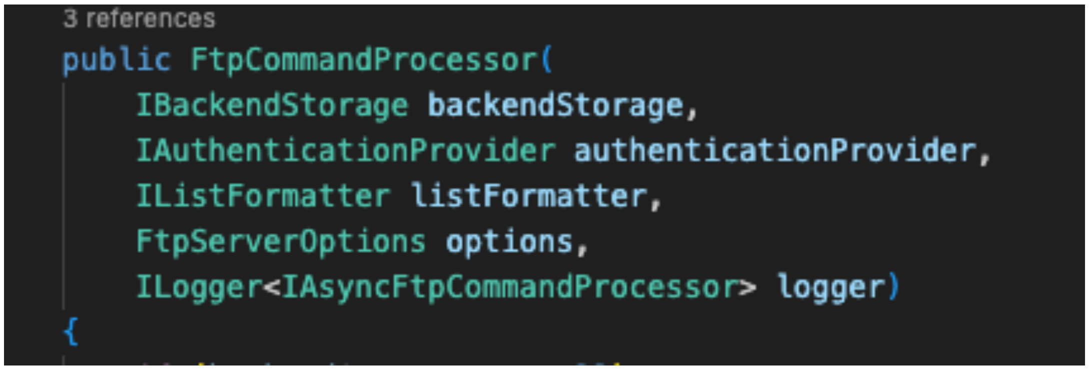
Summary:

The FtpCommandProcessor class is responsible for handling and executing FTP commands received from the client. It parses incoming command strings, uses a command handler factory to resolve the appropriate handler, and delegates the processing logic to that handler. It is a core part of the FTP session loop and implements the IAsyncFtpCommandProcessor interface.

This class supports full dependency injection of:

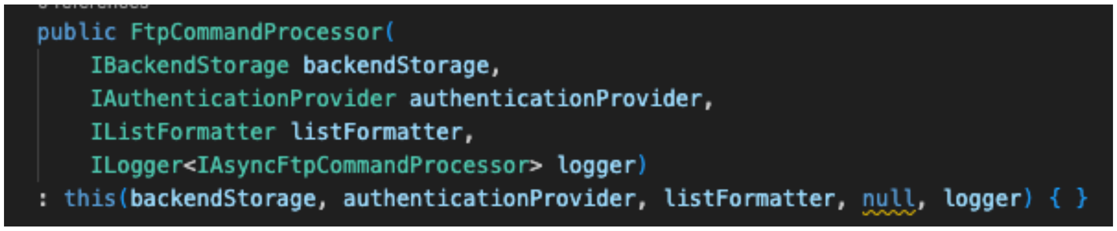
- Storage backend (IBackendStorage)
- Authentication logic (IAuthenticationProvider)
- Directory listing formatter (IListFormatter)
- Server configuration (FtpServerOptions)
- Optional logging via ILogger

Constructors



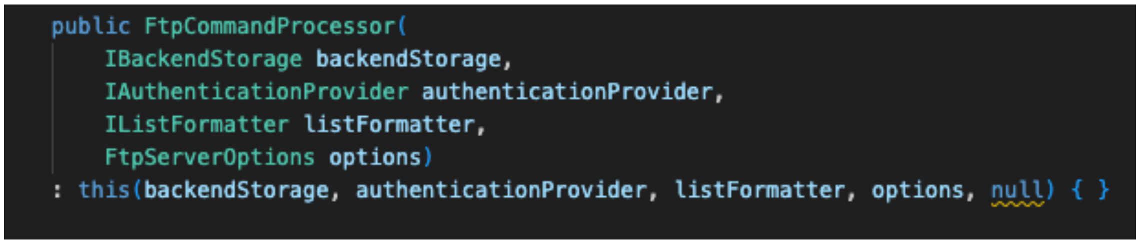
```
3 references
public FtpCommandProcessor(
    IBackendStorage backendStorage,
    IAuthenticationProvider authenticationProvider,
    IListFormatter listFormatter,
    FtpServerOptions options,
    ILogger<IAsyncFtpCommandProcessor> logger)
{
```

Initializes the processor with all major dependencies, including server configuration and logging support.



```
public FtpCommandProcessor(
    IBackendStorage backendStorage,
    IAuthenticationProvider authenticationProvider,
    IListFormatter listFormatter,
    ILogger<IAsyncFtpCommandProcessor> logger)
: this(backendStorage, authenticationProvider, listFormatter, null, logger) { }
```

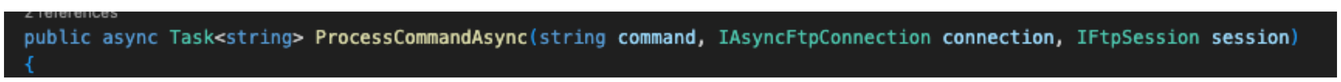
Initializes a new instance of theFtpCommandProcessor class without server options



```
public FtpCommandProcessor(
    IBackendStorage backendStorage,
    IAuthenticationProvider authenticationProvider,
    IListFormatter listFormatter,
    FtpServerOptions options)
: this(backendStorage, authenticationProvider, listFormatter, options, null) { }
```

Initializes the processor without logging support.

Methods



```
public async Task<string> ProcessCommandAsync(string command, IAsyncFtpConnection connection, IFtpSession session)
{
```

Processes a full command string received from the client. Logs the received command (if a logger is present) Uses FtpCommandHandlerFactory to resolve the appropriate command handler based on the command keyword

Delegates command handling to the resolved handlers HandleCommandAsync method

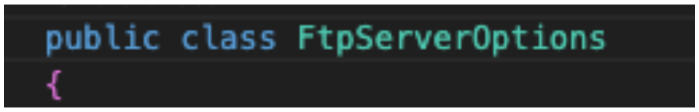
Throws:

- ArgumentException if the command is null or empty
- NotSupportedException if no handler exists for the given command

---

Configuration

FtpServerOptions

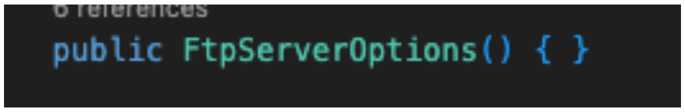


```
public class FtpServerOptions
{
```

Summary:

The FtpServerOptions class represents configuration settings for the FTP server. It defines properties related to networking, encryption, file storage, and passive mode behavior.

Constructor

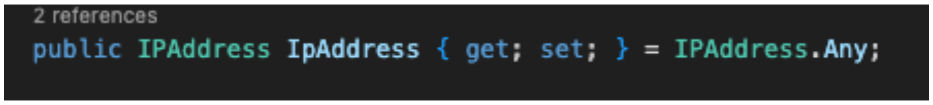


```
6 references
public FtpServerOptions() { }
```

Default constructor. Initializes a new instance of FtpServerOptions with default values:

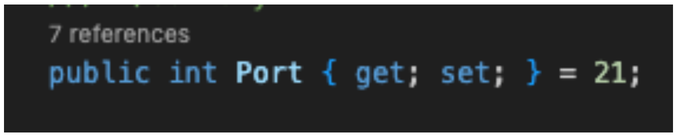
- Listens on all IPs (IPAddress.Any)
- Port 21
- TLS disabled
- No local root path specified

Properties



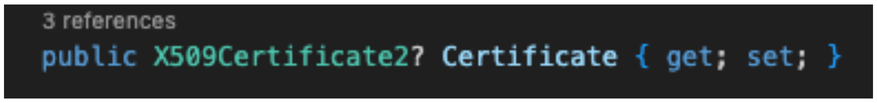
```
2 references
public IPAddress IpAddress { get; set; } = IPAddress.Any;
```

Specifies the IP address the server listens on. Defaults to IPAddress.Any



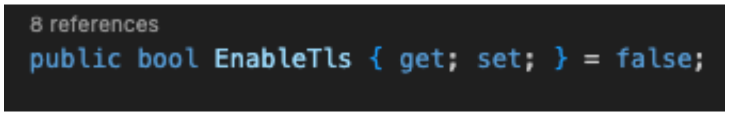
```
7 references
public int Port { get; set; } = 21;
```

Specifies the port number the server listens on. Defaults to the standard FTP port 21.



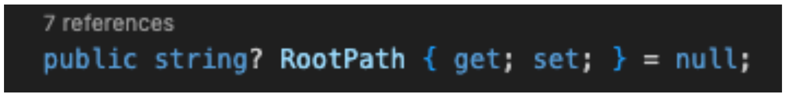
```
3 references
public X509Certificate2? Certificate { get; set; }
```

Holds the TLS certificate to be used for encrypted connections. If null, TLS is disabled unless configured later.



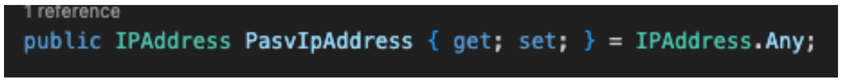
```
8 references
public bool EnableTls { get; set; } = false;
```

Indicates whether TLS encryption is enabled for client connections. When true, the server expects a valid certificate.



```
7 references
public string? RootPath { get; set; } = null;
```

Defines the root directory for file storage on the server. If null, local file system storage is considered disabled or defaults to the current directory.

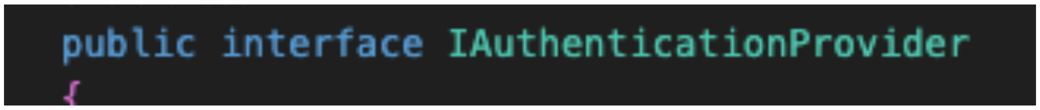


```
1 reference
public IPAddress PasvIpAddress { get; set; } = IPAddress.Any;
```

Specifies the IP address to advertise in FTP passive mode (PASV) responses. Defaults to the same IP the server listens on.

Authentication

IAuthenticationProvider

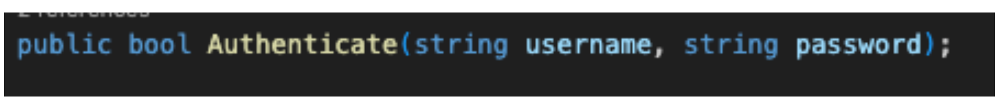


```
public interface IAuthenticationProvider
{
```

Summary:

The IAuthenticationProvider interface defines the contract for implementing user authentication in the FTP server. It provides a single method, Authenticate, which is responsible for validating client credentials (username and password).

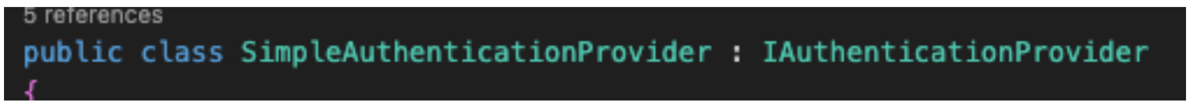
Methods



```
public bool Authenticate(string username, string password);
```

Authenticates a user with the given username and password.

SimpleAuthenticationProvider



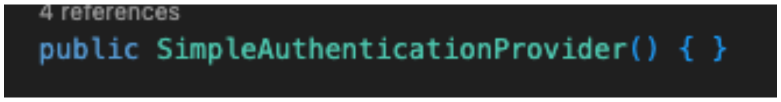
```
5 references
public class SimpleAuthenticationProvider : IAuthenticationProvider
{
```

Summary:

The SimpleAuthenticationProvider class provides a basic implementation of the IAuthenticationProvider interface using hardcoded credentials. It is currently designed for demonstration, testing, or local development purposes and is not suitable for production use.

The current implementation authenticates users only if the username is "test" and the password is "1234".

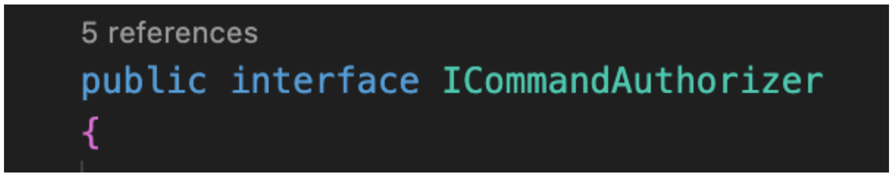
Constructors



```
4 references
public SimpleAuthenticationProvider() { }
```

Default constructor.

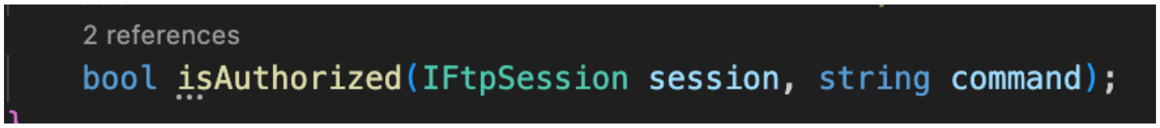
ICommandAuthorizer



```
5 references
public interface ICommandAuthorizer
{
```

Summary: Defines an interface/contract for authorized FTP-commands based on user roles assigned.

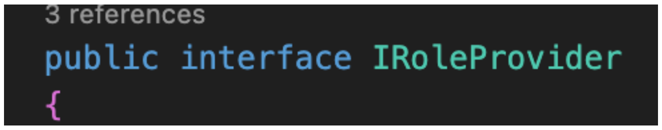
Methods



```
2 references
bool isAuthorized(IFtpSession session, string command);
```

Checks whether a specified command is authorized for the given session.

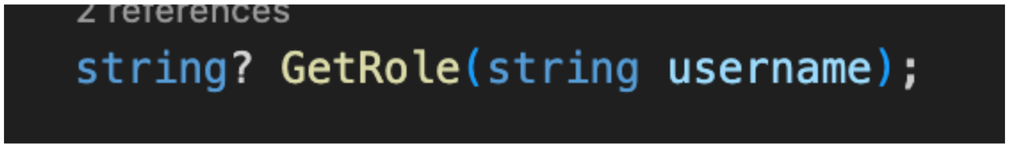
IRoleProvider



```
3 references
public interface IRoleProvider
{
```

Summary: Defines an interface for providing user roles.

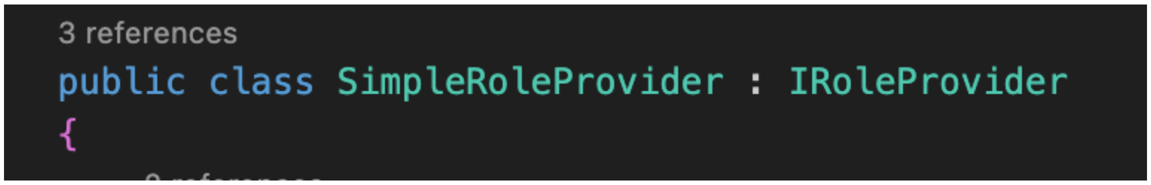
Methods



```
  2 references
  string? GetRole(string username);
```

Gets the role assigned for a specific username.

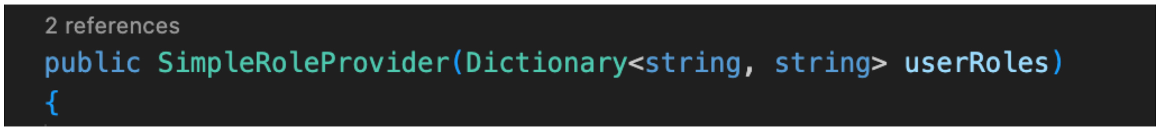
SimpleRoleProvider



```
3 references
public class SimpleRoleProvider : IRoleProvider
{
```

Summary: A simple in memory role provider for testing purposes.

Constructor



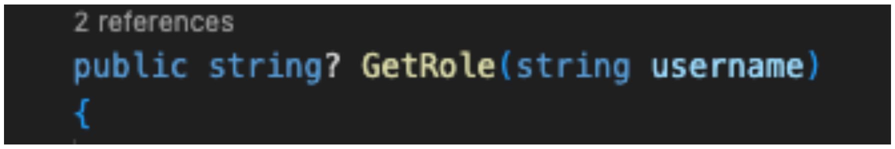
```
2 references
public SimpleRoleProvider(Dictionary<string, string> userRoles)
{
```

Initialises a nes instance of the SimpleRoleProvider class. Uses a dictionary to map each username to a respective role.

Throws:

- ArgumentNullException if the provided dictionary is null.

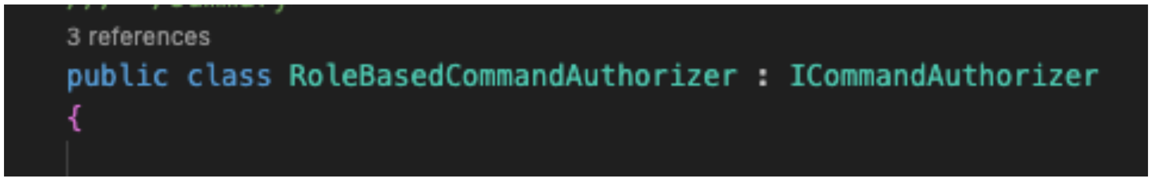
Methods



```
2 references
public string? GetRole(string username)
{
```

Gets the role for the specified username. Returns user role or null if user does not exist.

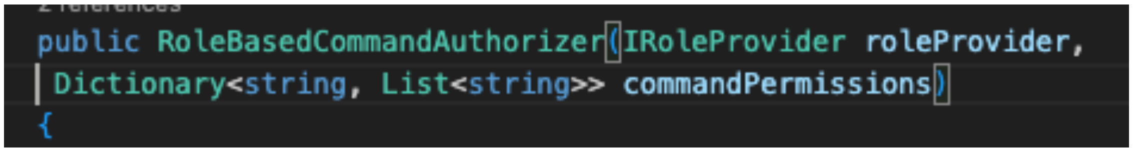
RoleBasedCommandAuthorizer



```
3 references
public class RoleBasedCommandAuthorizer : ICommandAuthorizer
{
```

Summary: This class is the implementation of the ICommandAuthorizer contract. The class enforces role-based, per command access control once a user is successfully authenticated. Delegates the user role lookup to IRoleProvider.

Constructors



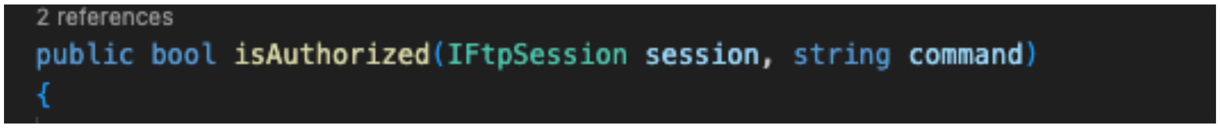
```
public RoleBasedCommandAuthorizer(IRoleProvider roleProvider,
    Dictionary<string, List<string>> commandPermissions)
{
```

Initializes a new instance of the RoleBasedCommandAuthorizer class.

Throws:

- ArgumentNullException if either parameter is null.

Methods

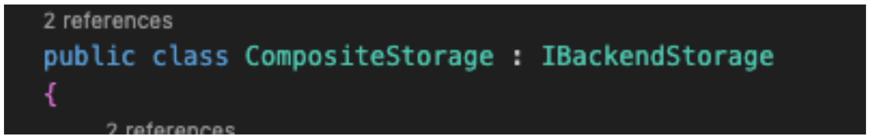


```
2 references
public bool isAuthorized(IFtpSession session, string command)
{
```

Determines whether the specified command is authorized for a given session. Always permits login lifecycle commands: USER, PASS and QUIT to make sure that users can sign in and out regardless of their respective role.

Storage & Backend

CompositeStorage

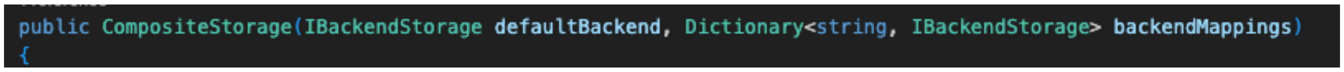


```
2 references
public class CompositeStorage : IBackendStorage
{
```

Summary:

The CompositeStorage class is an advanced storage backend that implements the IBackendStorage interface. It routes file operations to different storage backends depending on the requested file or directory path. This is useful when you want to support multiple storage systems (e.g. one for local, another for cloud) within the same FTP server.

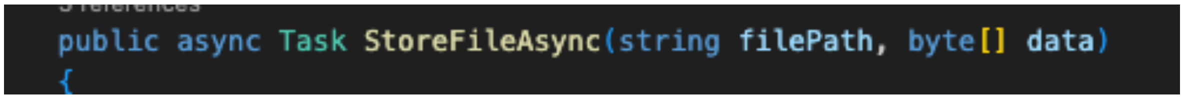
Constructors



```
public CompositeStorage(IBackendStorage defaultBackend, Dictionary<string, IBackendStorage> backendMappings)
{
```

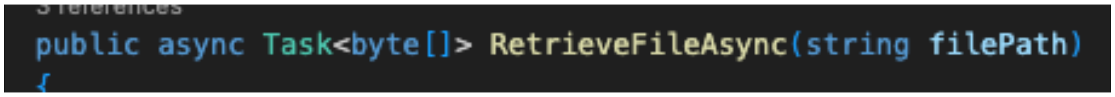
Initializes a new instance of the CompositeStorage class with a default backend when no path macth is found.

Methods



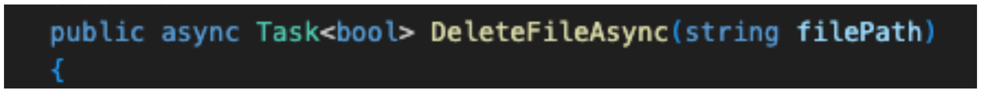
```
public async Task StoreFileAsync(string filePath, byte[] data)
{
```

Stores a file at the specified path using the appropriate backend determined by the path prefix.



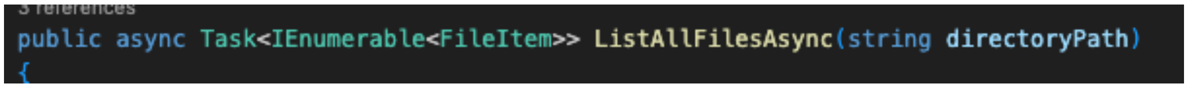
```
public async Task<byte[]> RetrieveFileAsync(string filePath)
{
```

Retrieves a file from the specified path using the matched backend.



```
public async Task<bool> DeleteFileAsync(string filePath)
{
```

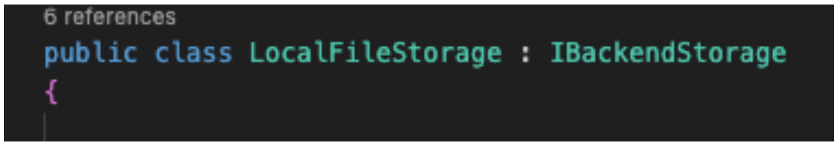
Deletes a file from the storage backend responsible for the given path.



```
3 references
public async Task<IEnumerable<FileItem>> ListAllFilesAsync(string directoryPath)
{
```

Lists files from the backend assigned to the given directory prefix.

LocalFileStorage

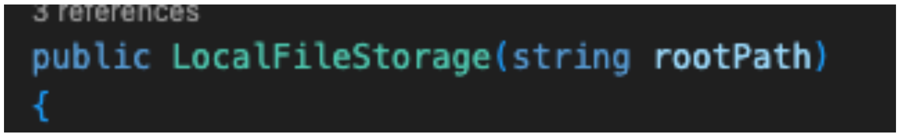


```
6 references
public class LocalFileStorage : IBackendStorage
{
```

Summary:

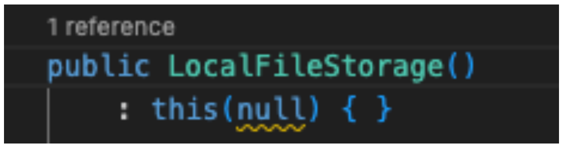
The LocalFileStorage class provides a concrete implementation of the IBackendStorage interface using the local file system. It allows storing, retrieving, deleting, and listing files and directories. This class is the default backend used by the FTP server when no custom storage provider is configured.

Constructors



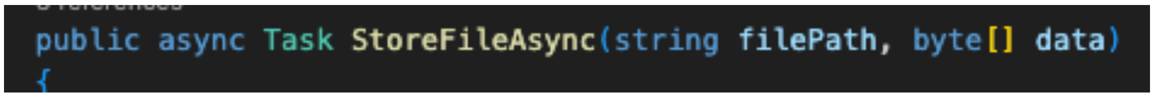
```
3 references
public LocalFileStorage(string rootPath)
{
```

Initializes a new instance of the LocalFileStorage class using the specified directory as the root for storage.

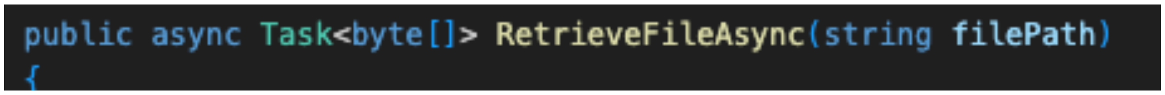

```
1 reference
public LocalFileStorage()
    : this(null) { }
```

Initializes a new instance of the LocalFileStorage class using the current directory as the root.

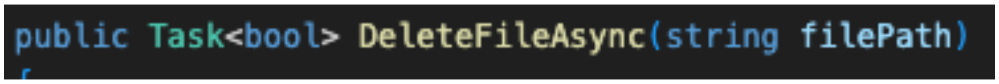
Methods


```
public async Task StoreFileAsync(string filePath, byte[] data)
{
```

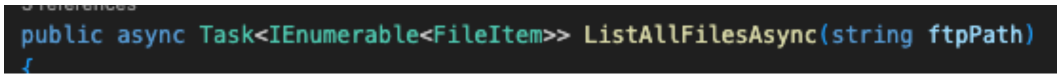
Stores the provided file content at the given FTP path.


```
public async Task<byte[]> RetrieveFileAsync(string filePath)
{
```

Retrieves the file at the specified path and returns its contents as a byte array.

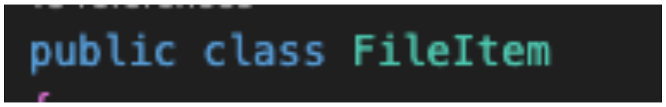

```
public Task<bool> DeleteFileAsync(string filePath)
```

Deletes the file located at the specified path.


```
public async Task<IEnumerable<FileItem>> ListAllFilesAsync(string ftpPath)
{
```

Lists all files in a specific directory.

FileItem

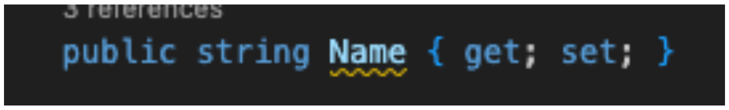


```
public class FileItem
```

Summary:

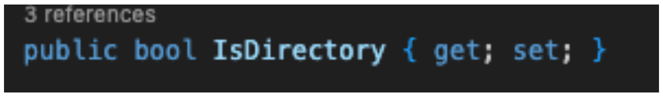
The FileItem class represents a single file or directory on the FTP server. It holds metadata such as the item's name, type (file or directory), size in bytes, and last modified timestamp.

Properties



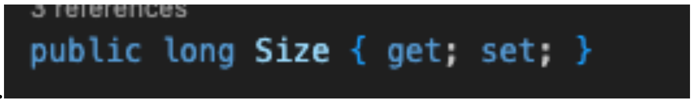
```
3 references
public string Name { get; set; }
```

Specifies the name of the file or directory.



```
3 references
public bool IsDirectory { get; set; }
```

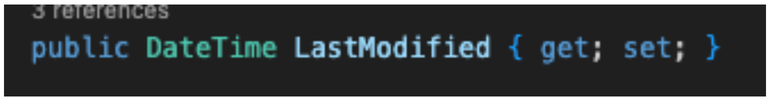
Indicates whether the item represents a directory (true) or a file (false)



```
3 references
public long Size { get; set; }
```

.

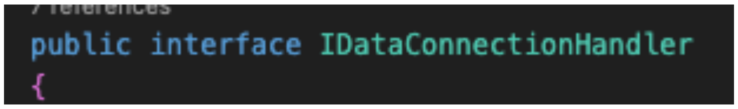
Defines the size of the file in bytes.



```
3 references
public DateTime LastModified { get; set; }
```

The timestamp of the last time the file or directory was modified.

IDataConnectionHandler

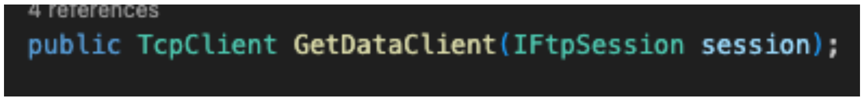


```
7 references
public interface IDataConnectionHandler
{
```

Summary:

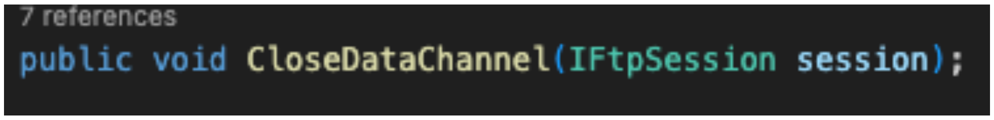
The IDataConnectionHandler interface defines a contract for managing the data connection used during FTP file transfers. It provides methods for retrieving the underlying TCP client used for active or passive mode data transfer and for safely closing the data channel after use.

Methods



```
4 references
public TcpClient GetDataClient(IFtpSession session);
```

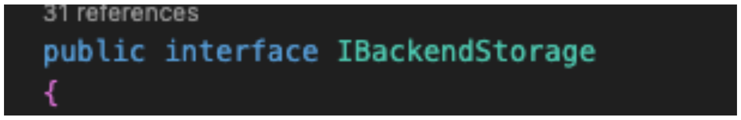
Retrieves the active TCP client used for sending or receiving file data with the FTP client.



```
7 references
public void CloseDataChannel(IFtpSession session);
```

Closes the data channel if open.

IBackendStorage

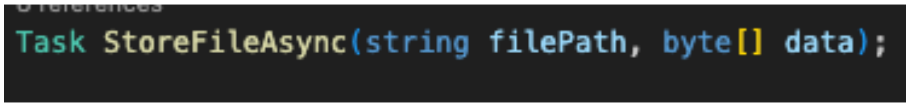

```
31 references
public interface IBackendStorage
{
```

Summart:

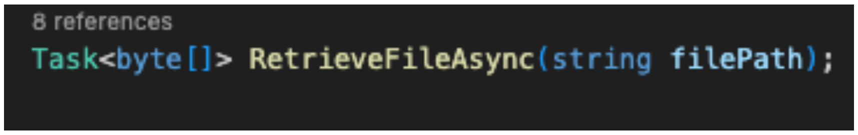
The IBackendStorage interface defines the contract for backend file storage operations used by the FTP server. It abstracts away the file system logicc, allowing the server to interact with different storage implementations (e.g., local disk, cloud) in a consistent and testable way.

This interface includes methods for storing, retrieving, deleting, and listing files in a directory.

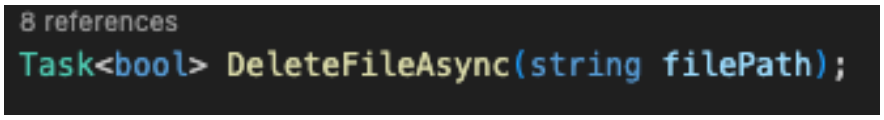
Methods


```
Task StoreFileAsync(string filePath, byte[] data);
```

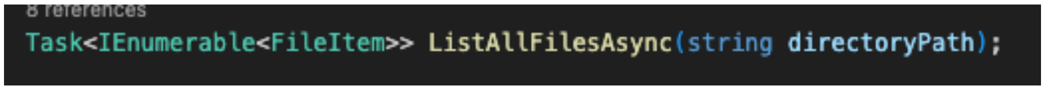
Stores a file at the specified path with the given binary content.


```
8 references
Task<byte[]> RetrieveFileAsync(string filePath);
```

Retrieves the binary content of a file located at the specified path.


```
8 references
Task<bool> DeleteFileAsync(string filePath);
```

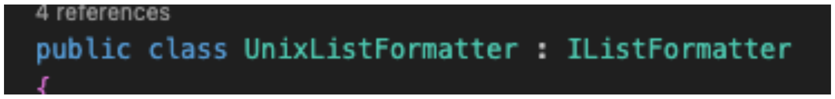
Deletes the file at the specified path


```
8 references
Task<IEnumerable<FileItem>> ListAllFilesAsync(string directoryPath);
```

Lists all file items in a given directory path.

Directory Listings

UnixListFormatter

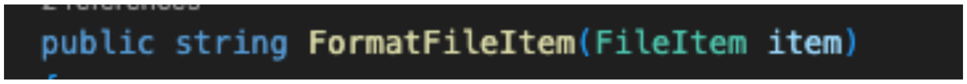


```
4 references
public class UnixListFormatter : IListFormatter
{
```

Summary:

The UnixListFormatter class provides a concrete implementation of the IListFormatter interface, formatting file items into Unix-style directory listing strings.

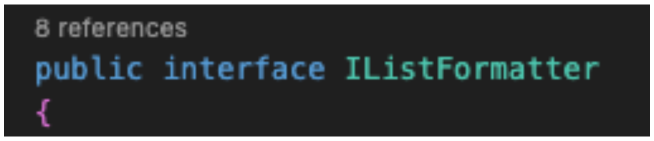
Methods



```
public string FormatFileItem(FileItem item)
```

Formats a file item into a UNIX-style string.

IListFormatter

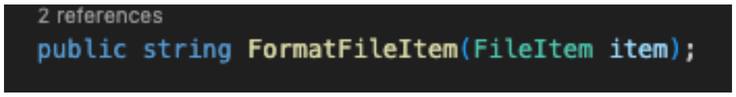


```
8 references
public interface IListFormatter
{
```

Summay:

The IListFormatter interface defines a contract for converting file system items into formatted text suitable for FTP directory listings (e.g., when handling the LIST command)

Methods



```
2 references
public string FormatFileItem(FileItem item);
```

Formats a FileItem object into a string representation appropriate for FTP directory listing responses.