FTP File Transfer Service

A report presented to
The Department of Electrical & Computer Engineering
Concordia University 2022
PhD. Chadi Assi

In partial fulfullement of the requirements of
COEN 366

By

Kevin de Oliveira - 40054907

1 Software Design Document 1

1.0.1 Scope . . . 1
1.0.2 Design Overview . . . 2
1.0.3 Technology Used . . . 2
1.0.4 System Architecture . . . 2

2 README 3

2.0.1 Network Application Development for File Transfer Service . . . 3
2.0.1.1 Overview . . . 3
2.0.1.2 Requirements . . . 3
2.0.1.3 Running . . . 3

3 Namespace Index 5

3.1 Packages . . . 5

4 Hierarchical Index 7

4.1 Class Hierarchy . . . 7

5 Class Index 9

5.1 Class List . . . 9

6 File Index 11

6.1 File List . . . 11

7 Namespace Documentation 13

7.1 ftp Namespace Reference . . . 13
7.2 ftp.cmd Namespace Reference . . . 13
7.2.1 Detailed Description . . . 13
7.3 ftp.cmd.arguments Namespace Reference . . . 13
7.4 ftp.parser Namespace Reference . . . 13
7.4.1 Detailed Description . . . 14
7.5 ftp.parser.message Namespace Reference . . . 14
7.6 ftp.parser.message_type Namespace Reference . . . 14
7.7 ftp.parser.packet Namespace Reference . . . 14
7.8 ftp.tcp Namespace Reference . . . 14
7.8.1 Detailed Description . . . 14
7.9 ftp.tcp.client Namespace Reference . . . 15
7.10 ftp.tcp.server Namespace Reference . . . 15
7.11 tcp_client Namespace Reference . . . 15
7.11.1 Function Documentation . . . 15
7.11.1.1 on_response_get() . . . 16
7.11.1.2 on_response_help() . . . 16
7.11.1.3 on_response_no_change() . . . 16
7.11.1.4 on_response_not_found() . . . 16

7.11.1.5 on_response_put_change() . . . 16
7.11.1.6 on_response_unknown() . . . 17
7.11.1.7 on_send_change() . . . 17
7.11.1.8 on_send_get() . . . 17
7.11.1.9 on_send_help() . . . 17
7.11.1.10 on_send_put() . . . 17
7.11.2 Variable Documentation . . . 17
7.11.2.1 arg_helper . . . 18
7.11.2.2 BASE_DIR . . . 18
7.11.2.3 client . . . 18
7.11.2.4 client_dir . . . 18
7.11.2.5 cmd . . . 18
7.11.2.6 params . . . 19
7.12 tcp_server Namespace Reference . . . 19
7.12.1 Function Documentation . . . 19
7.12.1.1 on_receive_change() . . . 19
7.12.1.2 on_receive_get() . . . 20
7.12.1.3 on_receive_help() . . . 20
7.12.1.4 on_receive_put() . . . 20
7.12.1.5 response_error_no_change() . . . 20
7.12.1.6 response_error_not_found() . . . 20
7.12.1.7 response_get() . . . 21
7.12.1.8 response_ok() . . . 21
7.12.1.9 response_ok_help() . . . 21
7.12.2 Variable Documentation . . . 21
7.12.2.1 arg_helper . . . 21
7.12.2.2 BASE_DIR . . . 21
7.12.2.3 cmd . . . 22
7.12.2.4 params . . . 22
7.12.2.5 server_dir . . . 22
7.12.2.6 tcp_server . . . 22

8 Class Documentation **23**

8.1 ftp.parser.message.Message Class Reference . . . 23
8.1.1 Detailed Description . . . 23
8.1.2 Constructor & Destructor Documentation . . . 23
8.1.2.1 __init__() . . . 23
8.1.3 Member Function Documentation . . . 24
8.1.3.1 __repr__() . . . 24
8.1.3.2 __str__() . . . 24
8.1.3.3 add_payload() . . . 24
8.1.3.4 has_payload() . . . 25

8.1.3.5 parse() . . . 25
8.1.4 Member Data Documentation . . . 25
8.1.4.1 header . . . 25
8.1.4.2 payload . . . 26
8.1.4.3 size . . . 26
8.2 ftp.parser.message_type.MessageType Class Reference . . . 26
8.2.1 Detailed Description . . . 27
8.2.2 Member Data Documentation . . . 27
8.2.2.1 REQUEST . . . 27
8.2.2.2 RESPONSE . . . 27
8.3 ftp.parser.message_type.MethodType Class Reference . . . 28
8.3.1 Detailed Description . . . 28
8.3.2 Member Function Documentation . . . 29
8.3.2.1 get_format() . . . 29
8.4 ftp.parser.packet.packet Class Reference . . . 29
8.4.1 Detailed Description . . . 29
8.4.2 Constructor & Destructor Documentation . . . 29
8.4.2.1 __init__() . . . 29
8.4.3 Member Function Documentation . . . 30
8.4.3.1 __call__() . . . 30
8.4.3.2 __repr__() . . . 30
8.4.3.3 __str__() . . . 31
8.4.3.4 set_size() . . . 31
8.4.3.5 to_bytes() . . . 31
8.4.3.6 value() . . . 31
8.4.4 Member Data Documentation . . . 32
8.4.4.1 body . . . 32
8.4.4.2 size . . . 32
8.5 ftp.cmd.arguments.ParserArgs Class Reference . . . 32
8.5.1 Detailed Description . . . 32
8.5.2 Constructor & Destructor Documentation . . . 33
8.5.2.1 __init__() . . . 33
8.5.3 Member Function Documentation . . . 33
8.5.3.1 get_args() . . . 33
8.5.3.2 parameters() . . . 34
8.5.4 Member Data Documentation . . . 34
8.5.4.1 argn . . . 34
8.5.4.2 argv . . . 34
8.5.4.3 default_helper . . . 34
8.5.4.4 default_version . . . 35
8.6 ftp.parser.message_type.RequestType Class Reference . . . 35
8.6.1 Detailed Description . . . 36

8.6.2 Member Function Documentation . . . 36
8.6.2.1 get_format() . . . 37
8.6.3 Member Data Documentation . . . 37
8.6.3.1 CHANGE . . . 37
8.6.3.2 GET . . . 37
8.6.3.3 HELP . . . 37
8.6.3.4 PUT . . . 37
8.7 ftp.parser.message_type.ResponseType Class Reference . . . 38
8.7.1 Detailed Description . . . 39
8.7.2 Member Function Documentation . . . 39
8.7.2.1 get_format() . . . 39
8.7.3 Member Data Documentation . . . 39
8.7.3.1 ERROR_NO_CHANGE . . . 39
8.7.3.2 ERROR_NOT_FOUND . . . 40
8.7.3.3 ERROR_UNKNOWN . . . 40
8.7.3.4 HELP . . . 40
8.7.3.5 OK_GET . . . 40
8.7.3.6 OK_PUT_CHANGE . . . 40
8.8 ftp.tcp.client.TcpClient Class Reference . . . 41
8.8.1 Detailed Description . . . 41
8.8.2 Constructor & Destructor Documentation . . . 41
8.8.2.1 __init__() . . . 41
8.8.3 Member Function Documentation . . . 42
8.8.3.1 check_response() . . . 42
8.8.3.2 cin() . . . 42
8.8.3.3 connect() . . . 43
8.8.3.4 handle_connection() . . . 43
8.8.3.5 handler() . . . 43
8.8.3.6 on_response() . . . 43
8.8.3.7 on_send() . . . 44
8.8.4 Member Data Documentation . . . 44
8.8.4.1 ip_address . . . 44
8.8.4.2 socket . . . 44
8.8.4.3 thread . . . 45
8.9 ftp.tcp.server.TcpServer Class Reference . . . 45
8.9.1 Detailed Description . . . 45
8.9.2 Constructor & Destructor Documentation . . . 45
8.9.2.1 __init__() . . . 45
8.9.3 Member Function Documentation . . . 46
8.9.3.1 handle_listen() . . . 46
8.9.3.2 listen() . . . 46
8.9.3.3 on_receive() . . . 46

8.9.3.4 parse_packet() . . . 47
8.9.4 Member Data Documentation . . . 47
8.9.4.1 ip_address . . . 47
8.9.4.2 is_connected . . . 47
8.9.4.3 socket . . . 48
8.9.4.4 thread . . . 48
8.10 ftp.parser.message.Util Class Reference . . . 48
8.10.1 Detailed Description . . . 48
8.10.2 Member Function Documentation . . . 48
8.10.2.1 bit2byte() . . . 48
8.10.2.2 deserialize() . . . 49
8.10.2.3 serialize() . . . 49
8.10.2.4 str2bit() . . . 50

9 File Documentation **51**

9.1 ftp/cmd/arguments.py File Reference . . . 51
9.1.1 Detailed Description . . . 51
9.1.2 Author(s) . . . 51
9.2 arguments.py . . . 52
9.3 ftp/__init__.py File Reference . . . 53
9.4 __init__.py . . . 53
9.5 ftp/cmd/__init__.py File Reference . . . 53
9.6 __init__.py . . . 53
9.7 ftp/parser/__init__.py File Reference . . . 53
9.8 __init__.py . . . 53
9.9 ftp/tcp/__init__.py File Reference . . . 54
9.10 __init__.py . . . 54
9.11 ftp/parser/message.py File Reference . . . 54
9.11.1 Detailed Description . . . 54
9.11.2 Author(s) . . . 54
9.12 message.py . . . 55
9.13 ftp/parser/message_type.py File Reference . . . 58
9.13.1 Detailed Description . . . 58
9.13.2 Author(s) . . . 58
9.14 message_type.py . . . 59
9.15 ftp/parser/packet.py File Reference . . . 60
9.16 packet.py . . . 60
9.17 ftp/tcp/client.py File Reference . . . 61
9.17.1 Detailed Description . . . 61
9.17.2 Author(s) . . . 61
9.18 client.py . . . 62
9.19 ftp/tcp/server.py File Reference . . . 64

9.19.1 Detailed Description 64
9.19.2 Author(s) 65
9.20 server.py 65
9.21 tcp_client.py File Reference 67
9.21.1 Detailed Description 67
9.21.2 Author(s) 67
9.22 tcp_client.py 68
9.23 tcp_server.py File Reference 70
9.23.1 Detailed Description 70
9.23.2 Author(s) 70
9.24 tcp_server.py 71

Index **73**

Chapter 1

Software Design Document

The purpose of this document is provide a detailed description of the implementation of the File Transfer Service as per required by the project description. This application partially emulates the service provided for file transfer protocols and follows its defined standards for encoding and transporting data between the client adn server.

1.0.1 Scope

This project consists of two applications, tcp_client and tcp_server. Since both programs share common functionalities, the implementation has been divided into different packages under the `ftp` namespace which they both make use of. The list of implemented modules are provided below:

- cmd
  - arguments
- parser
  - message_type
  - message
  - packet
- tcp
  - client
  - server

Furthermore, some of the above modules depends on external packages which should be properly installed with the help of a package manager.

- autopep8==1.6.0
- bitarray-ph4==1.9.1
- importlib-metadata==4.11.3
- Mako==1.2.0

A complete list of required packages is provided in the `requirements.txt` text file.

1.0.2 Design Overview

The file transfer protocol provides a set of standard communication protocol used for sending and receiving files over the network. The connection should be established between a client and server, and the communication should be done over a secure channel where data is transfered. The data sent is usually encapsulated by a packet which contains the information that the device wish to transmit. Such data is expected to follow a sequence of request and response message flow, so both devices can properly communicate with each other. The following list defines the set of request-response message expected by both server and client.

Request	Header
put name	000
get name	001
change old new	010
help	011

Response	Header
ok put or change	000
ok get	001
error not found	010
error unknown	011
help	110

1.0.3 Technology Used

The implementation of both application has been done using the Python3's socket API which provides several functions and methods to perform IPC communications over the network.

1.0.4 System Architecture

The file transfer servioce is constructed over a set of defined objects, each defining a specific type or set of behavior. Both clients and servers use a well-defined packet type to communicate with each other. Therefore, in order to properly evaluate this communication, each packet sent or received are properly parsed and converted to a known bit representation, which is then verified against the set of expected Response or Request message.

- Message Type: Provides the defintion of each Method used in this system
- Message: Object that represents the data used during FTP communication. Also provides an utility class that provides helper functions.
- Packet: Object that is represented as a string of bit values
- Client: TCP Client algorithm and logic
- Server: TCP Server algorithm and logic

Chapter 2

README

2.0.1 Network Application Development for File Transfer Service

Develop a pair of client-server programs that communicate via Python stream sockets and simulate partially the file transfer protocol (FTP). The main purpose of these client/server programs is to give the client the ability to download files from the server directory to the client directory and upload files from the client directory to the server directory. We should be able to transfer any file type such as txt, doc, jpg.

2.0.1.1 Overview

- The application consist of two parts, an FTP Client and FTP Server
- The main purpose of these client/server programs is to give the client the ability to download files from the server directory to the client directory and upload files from the client directory to the server directory
- All required packages for this project are provided in the requirement text file

2.0.1.2 Requirements

- Python 3
- Required packages installed (requirements.txt)

2.0.1.3 Running

Before execution, make sure the main directory for both server and client are created

To run the server: `tcp_server.py`
To run the client: `tcp_client.py`

The list of available commands can be found by passing any of those help arguments `-h --help -?`. Below you can find the set of arguments available for both applications.

- -d: Activate Debug mode
- -a `addr`: Address which socket should connect to. Default value is localhost (127.0.0.1)
- -p `arg`: Port number which socket should attach its connection. Default value is 1025
- -F `arg`: Relative path used by the application
- -f `arg`: Absolute path used by the application

Chapter 3

Namespace Index

3.1 Packages

Here are the packages with brief descriptions (if available):

ftp . . . 13
ftp.cmd . . . 13
ftp.cmd.arguments . . . 13
ftp.parser . . . 13
ftp.parser.message . . . 14
ftp.parser.message_type . . . 14
ftp.parser.packet . . . 14
ftp.tcp . . . 14
ftp.tcp.client . . . 15
ftp.tcp.server . . . 15
tcp_client . . . 15
tcp_server . . . 19

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

enum.Enum
- ftp.parser.message_type.MessageType . . . 26
- ftp.parser.message_type.MethodType . . . 28
  - ftp.parser.message_type.RequestType . . . 35
  - ftp.parser.message_type.ResponseType . . . 38

ftp.parser.message.Message . . . 23
ftp.parser.packet.packet . . . 29
ftp.cmd.arguments.ParserArgs . . . 32
ftp.tcp.client.TcpClient . . . 41
ftp.tcp.server.TcpServer . . . 45
ftp.parser.message.Util . . . 48

Chapter 5

Class Index

5.1 Class List

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

ftp.parser.message.Message 23
ftp.parser.message_type.MessageType
    Enum class for type of message sent 26
ftp.parser.message_type.MethodType 28
ftp.parser.packet.packet 29
ftp.cmd.arguments.ParserArgs 32
ftp.parser.message_type.RequestType
    Enum class for type of Request sent 35
ftp.parser.message_type.ResponseType
    Enum class for type of Response sent 38
ftp.tcp.client.TcpClient 41
ftp.tcp.server.TcpServer 45
ftp.parser.message.Util 48

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

tcp_client.py
: Main TCP client application which contains the implementation of the client-end of this FTP service . . . 67

tcp_server.py
: Main TCP server application which contains the implementation of the server-end of this FTP service . . . 70

ftp/__init__.py . . . 53

ftp/cmd/__init__.py . . . 53

ftp/cmd/arguments.py
: Parses arguemnts provided in command-line . . . 51

ftp/parser/__init__.py . . . 53

ftp/parser/message.py
: Segment containing the packets for Request and Response messages . . . 54

ftp/parser/message_type.py
: Enumaration classes for each method type used during FTP communication . . . 58

ftp/parser/packet.py . . . 60

ftp/tcp/__init__.py . . . 54

ftp/tcp/client.py
: TCP Client logic and algorithm . . . 61

ftp/tcp/server.py
: TCP Server logic and algorithm . . . 64

Chapter 7

Namespace Documentation

7.1 ftp Namespace Reference

Namespaces

- namespace cmd
- namespace parser
- namespace tcp

7.2 ftp.cmd Namespace Reference

Namespaces

- namespace arguments

7.2.1 Detailed Description

```
Module responsible for parsing command-line arguments
```

7.3 ftp.cmd.arguments Namespace Reference

Classes

- class ParserArgs

7.4 ftp.parser Namespace Reference

Namespaces

- namespace message
- namespace message_type
- namespace packet

7.4.1 Detailed Description

```
Module defining ADT objects used for TCP service
```

7.5 ftp.parser.message Namespace Reference

Classes

- class Message
- class Util

7.6 ftp.parser.message_type Namespace Reference

Classes

- class MessageType

  Enum class for type of message sent.
- class MethodType
- class RequestType

  Enum class for type of Request sent.
- class ResponseType

  Enum class for type of Response sent.

7.7 ftp.parser.packet Namespace Reference

Classes

- class packet

7.8 ftp.tcp Namespace Reference

Namespaces

- namespace client
- namespace server

7.8.1 Detailed Description

```
Module responsible managing and craeting TCP services
```

7.9 ftp.tcp.client Namespace Reference

Classes

- class TcpClient

7.10 ftp.tcp.server Namespace Reference

Classes

- class TcpServer

7.11 tcp_client Namespace Reference

Functions

- bytes on_send_put (List[str] inp, MethodType type)
- bytes on_send_get (List[str] inp, MethodType type)
- bytes on_send_change (List[str] inp, MethodType type)
- bytes on_send_help (List[str] inp, MethodType type)
- def on_response_put_change (Message message)
- def on_response_get (Message message)

  Checks if file already exists.
- def on_response_help (Message message)
- def on_response_unknown (Message message)
- def on_response_not_found (Message message)
- def on_response_no_change (Message mesage)

Variables

- string client_dir = "client"
- string BASE_DIR = os.getcwd() + os.sep + "dir" + os.sep + client_dir + os.sep
- string arg_helper
- cmd = arguments.ParserArgs(helper = arg_helper, version="tcp_server version 1.0")
- params = cmd.parameters(["-a", "-p", "-f", "-F", "-d"])
- client = TcpClient("127.0.0.1", ∗∗params)

7.11.1 Function Documentation

7.11.1.1 on_response_get()

```
def tcp_client.on_response_get (
            Message message )
```

Checks if file already exists.

Definition at line 95 of file tcp_client.py.

7.11.1.2 on_response_help()

```
def tcp_client.on_response_help (
            Message message )
```

Definition at line 121 of file tcp_client.py.

7.11.1.3 on_response_no_change()

```
def tcp_client.on_response_no_change (
            Message mesage )
```

Definition at line 131 of file tcp_client.py.

7.11.1.4 on_response_not_found()

```
def tcp_client.on_response_not_found (
            Message message )
```

Definition at line 128 of file tcp_client.py.

7.11.1.5 on_response_put_change()

```
def tcp_client.on_response_put_change (
            Message message )
```

Definition at line 92 of file tcp_client.py.

7.11.1.6 on_response_unknown()

```
def tcp_client.on_response_unknown (
            Message message )
```

Definition at line 125 of file tcp_client.py.

7.11.1.7 on_send_change()

```
bytes tcp_client.on_send_change (
            List[str] inp,
            MethodType type )
```

Definition at line 73 of file tcp_client.py.

7.11.1.8 on_send_get()

```
bytes tcp_client.on_send_get (
            List[str] inp,
            MethodType type )
```

Definition at line 62 of file tcp_client.py.

7.11.1.9 on_send_help()

```
bytes tcp_client.on_send_help (
            List[str] inp,
            MethodType type )
```

Definition at line 83 of file tcp_client.py.

7.11.1.10 on_send_put()

```
bytes tcp_client.on_send_put (
            List[str] inp,
            MethodType type )
```

Definition at line 35 of file tcp_client.py.

7.11.2 Variable Documentation

7.11.2.1 arg_helper

```
string tcp_client.arg_helper
```

Initial value:

```
00001 =  """usage: tcp_client [-a address] [-p port] [-f base_folder] [-F absolute_folder] [-v | --version]
       [-h | --help | -?]
00002
00003 This are the commands used:
00004 \t-d\t\t Activate debug mode
00005 \t-a address\t\t Set address of this client (default: 127.0.0.1)
00006 \t-p port\t\t\t Set port number of this client (default: 1025)
00007 \t-f base_folder\t\t Set relative base path of the FTP client (default: /dir/client)
00008 \t-F absolute_folder\t Set absolute base path of the FTP client (default: $pwd)"""
```

Definition at line 136 of file tcp_client.py.

7.11.2.2 BASE_DIR

```
tcp_client.BASE_DIR = os.getcwd() + os.sep + "dir" + os.sep + client_dir + os.sep
```

Definition at line 30 of file tcp_client.py.

7.11.2.3 client

```
tcp_client.client = TcpClient("127.0.0.1", **params)
```

Definition at line 161 of file tcp_client.py.

7.11.2.4 client_dir

```
tcp_client.client_dir = "client"
```

Definition at line 28 of file tcp_client.py.

7.11.2.5 cmd

```
tcp_client.cmd = arguments.ParserArgs(helper = arg_helper, version="tcp_server version 1.0")
```

Definition at line 148 of file tcp_client.py.

7.11.2.6 params

```
tcp_client.params = cmd.parameters(["-a", "-p", "-f", "-F", "-d"])
```

Definition at line 152 of file tcp_client.py.

7.12 tcp_server Namespace Reference

Functions

- bytes response_ok ()
- bytes response_get (str file_name, int size, str payload)
- def response_error_not_found ()
- def response_error_no_change ()
- def response_ok_help ()
- bytes on_receive_put (addr, Message data)
- bytes on_receive_get (addr, Message data)
- bytes on_receive_change (addr, Message data)
- bytes on_receive_help (addr, Message data)

Variables

- string server_dir = "server"
- string BASE_DIR = os.getcwd() + os.sep + "dir" + os.sep + server_dir + os.sep
- string arg_helper
- cmd = arguments.ParserArgs(helper = arg_helper, version="tcp_server version 1.0")
- params = cmd.parameters(["-a", "-p", "-f", "-F", "-d"])
- tcp_server = TcpServer("127.0.0.1", ∗∗params)

7.12.1 Function Documentation

7.12.1.1 on_receive_change()

```
bytes tcp_server.on_receive_change (
              addr,
            Message data )
```

Definition at line 108 of file tcp_server.py.

7.12.1.2 on_receive_get()

```
bytes tcp_server.on_receive_get (
              addr,
             Message data )
```

Definition at line 85 of file tcp_server.py.

7.12.1.3 on_receive_help()

```
bytes tcp_server.on_receive_help (
              addr,
             Message data )
```

Definition at line 124 of file tcp_server.py.

7.12.1.4 on_receive_put()

```
bytes tcp_server.on_receive_put (
              addr,
             Message data )
```

Definition at line 67 of file tcp_server.py.

7.12.1.5 response_error_no_change()

```
def tcp_server.response_error_no_change ( )
```

Definition at line 54 of file tcp_server.py.

7.12.1.6 response_error_not_found()

```
def tcp_server.response_error_not_found ( )
```

Definition at line 49 of file tcp_server.py.

7.12.1.7 response_get()

```
bytes tcp_server.response_get (
            str file_name,
            int size,
            str payload )
```

Definition at line 38 of file tcp_server.py.

7.12.1.8 response_ok()

```
bytes tcp_server.response_ok ( )
```

Definition at line 33 of file tcp_server.py.

7.12.1.9 response_ok_help()

```
def tcp_server.response_ok_help ( )
```

Definition at line 59 of file tcp_server.py.

7.12.2 Variable Documentation

7.12.2.1 arg_helper

```
string tcp_server.arg_helper
```

Initial value:

```
00001 =  """usage: tcp_server [-a address] [-p port] [-f base_folder] [-F absolute_folder] [-v | --version]
       [-h | --help | -?]
00002
00003 This are the commands used:
00004 \t-d\t\t Activate debug mode
00005 \t-a address\t\t Set address of this server (default: 127.0.0.1)
00006 \t-p port\t\t\t Set port number of this server (default: 1025)
00007 \t-f base_folder\t\t Set relative base path of the FTP server (default: /dir/server)
00008 \t-F absolute_folder\t Set absolute base path of the FTP server (default: $pwd)"""
```

Definition at line 130 of file tcp_server.py.

7.12.2.2 BASE_DIR

```
tcp_server.BASE_DIR = os.getcwd() + os.sep + "dir" + os.sep + server_dir + os.sep
```

Definition at line 30 of file tcp_server.py.

7.12.2.3 cmd

```
tcp_server.cmd = arguments.ParserArgs(helper = arg_helper, version="tcp_server version 1.0")
```

Definition at line 141 of file tcp_server.py.

7.12.2.4 params

```
tcp_server.params = cmd.parameters(["-a", "-p", "-f", "-F", "-d"])
```

Definition at line 145 of file tcp_server.py.

7.12.2.5 server_dir

```
tcp_server.server_dir = "server"
```

Definition at line 27 of file tcp_server.py.

7.12.2.6 tcp_server

```
tcp_server.tcp_server = TcpServer("127.0.0.1", **params)
```

Definition at line 152 of file tcp_server.py.

Chapter 8

Class Documentation

8.1 ftp.parser.message.Message Class Reference

Public Member Functions

- None __init__ (self, int header_size, MethodType type)

  Encapsulates a packet object that is sent to the socket.
- Tuple[packet, List[packet]] or None parse (self, str value)

  Parses input values into its respective data field.
- None add_payload (self, str value)

  Attach payload to this message.
- bool has_payload (self)

  Verifies if this message has a payload attached.
- str __repr__ (self)
- str __str__ (self)

Public Attributes

- header
- size
- payload

8.1.1 Detailed Description

Definition at line 22 of file message.py.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 __init__()

```
None ftp.parser.message.Message.__init__ (
              self,
             int header_size,
             MethodType type )
```

Encapsulates a packet object that is sent to the socket.

message is represented as: [header | data_1 | data_2 | ... | payload?]

Parameters

header_size	int Size of the header packet
type	MethodType Type of message that will be sent

Returns

Returns None

Definition at line 23 of file message.py.

8.1.3 Member Function Documentation

8.1.3.1 __repr__()

```
str ftp.parser.message.Message.__repr__ (
              self )
```

Definition at line 116 of file message.py.

8.1.3.2 __str__()

```
str ftp.parser.message.Message.__str__ (
              self )
```

Definition at line 122 of file message.py.

8.1.3.3 add_payload()

```
None ftp.parser.message.Message.add_payload (
              self,
             str value )
```

Attach payload to this message.

Parameters

value	str Value to be parsed into packet. Note that value must be represented in binary format

Returns

None

Definition at line 93 of file message.py.

8.1.3.4 has_payload()

```
bool ftp.parser.message.Message.has_payload (
              self )
```

Verifies if this message has a payload attached.

Returns

bool: Returns true if contains a paylaod; otherwise false

Definition at line 106 of file message.py.

8.1.3.5 parse()

```
Tuple[packet, List[packet]] or None ftp.parser.message.Message.parse (
              self,
             str value )
```

Parses input values into its respective data field.

Parameters

value	str Inserts the values its respective data. Note that value must be represented in binary format

Returns

Tuple or None: An optional tuple containing the header packet and its data packets

Definition at line 70 of file message.py.

8.1.4 Member Data Documentation

8.1.4.1 header

```
ftp.parser.message.Message.header
```

Definition at line 34 of file message.py.

8.1.4.2 payload

`ftp.parser.message.Message.payload`

Definition at line 103 of file message.py.

8.1.4.3 size

`ftp.parser.message.Message.size`

Definition at line 41 of file message.py.

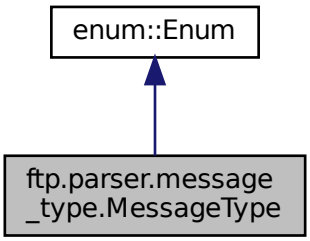
The documentation for this class was generated from the following file:

- ftp/parser/message.py

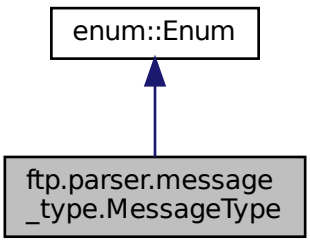
8.2 ftp.parser.message_type.MessageType Class Reference

Enum class for type of message sent.

Inheritance diagram for ftp.parser.message_type.MessageType:



Collaboration diagram for ftp.parser.message_type.MessageType:



Static Public Attributes

- int REQUEST = 0
- int RESPONSE = 1

8.2.1 Detailed Description

Enum class for type of message sent.

Definition at line 19 of file message_type.py.

8.2.2 Member Data Documentation

8.2.2.1 REQUEST

```
int ftp.parser.message_type.MessageType.REQUEST = 0  [static]
```

Definition at line 23 of file message_type.py.

8.2.2.2 RESPONSE

```
int ftp.parser.message_type.MessageType.RESPONSE = 1  [static]
```

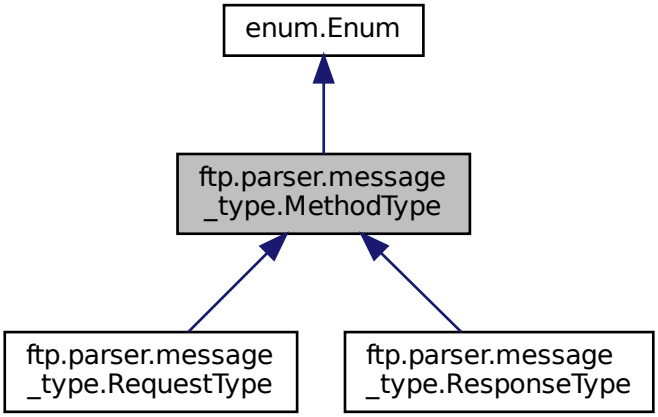
Definition at line 24 of file message_type.py.

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

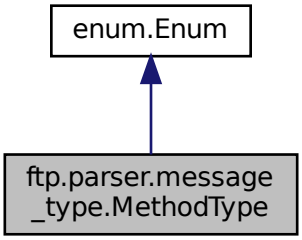
- ftp/parser/message_type.py

8.3 ftp.parser.message_type.MethodType Class Reference

Inheritance diagram for ftp.parser.message_type.MethodType:



Collaboration diagram for ftp.parser.message_type.MethodType:



Public Member Functions

- List[int] get_format (self)

8.3.1 Detailed Description

Definition at line 26 of file message_type.py.

8.3.2 Member Function Documentation

8.3.2.1 get_format()

```
List[int] ftp.parser.message_type.MethodType.get_format (
              self )
```

Reimplemented in ftp.parser.message_type.RequestType, and ftp.parser.message_type.ResponseType.

Definition at line 28 of file message_type.py.

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

- ftp/parser/message_type.py

8.4 ftp.parser.packet.packet Class Reference

Public Member Functions

- None __init__ (self, int max_bits, bool fill=False, *args)
  Object representing a packet as binary.
- 'packet' __call__ (self, str val)
  Append value to this object.
- bitarray.bitarray value (self)
  Getter of the body attribute.
- bytes to_bytes (self)
  Returns the byte value of the body attribute.
- None set_size (self, int val)
  Redefine the packet object size.
- str __str__ (self)
- str __repr__ (self)

Public Attributes

- body
- size

8.4.1 Detailed Description

Definition at line 16 of file packet.py.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 __init__()

```
None ftp.parser.packet.packet.__init__ (
              self,
             int max_bits,
             bool  fill = False,
             * args )
```

Object representing a packet as binary.

Parameters

max_bits	int Maximum size of this packet object
fill	bool Align the length of this packet to a multiple of 8 (1 byte)

Returns

packet

Definition at line 19 of file packet.py.

8.4.3 Member Function Documentation

8.4.3.1 __call__()

```
'packet' ftp.parser.packet.packet.__call__ (
              self,
             str val )
```

Append value to this object.

Parameters

val	str Value to be appended

Returns

packet

Definition at line 36 of file packet.py.

8.4.3.2 __repr__()

```
str ftp.parser.packet.packet.__repr__ (
              self )
```

Definition at line 84 of file packet.py.

8.4.3.3 __str__()

```
str ftp.parser.packet.packet.__str__ (
              self )
```

Definition at line 80 of file packet.py.

8.4.3.4 set_size()

```
None ftp.parser.packet.packet.set_size (
              self,
             int val )
```

Redefine the packet object size.

Parameters

val	int Size of this packet object

Returns

None

Definition at line 67 of file packet.py.

8.4.3.5 to_bytes()

```
bytes ftp.parser.packet.packet.to_bytes (
              self )
```

Returns the byte value of the body attribute.

Returns

Byte object

Definition at line 58 of file packet.py.

8.4.3.6 value()

```
bitarray.bitarray ftp.parser.packet.packet.value (
              self )
```

Getter of the body attribute.

Returns

Bitarray object

Definition at line 50 of file packet.py.

8.4.4 Member Data Documentation

8.4.4.1 body

`ftp.parser.packet.packet.body`

Definition at line 28 of file packet.py.

8.4.4.2 size

`ftp.parser.packet.packet.size`

Definition at line 29 of file packet.py.

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

- ftp/parser/packet.py

8.5 ftp.cmd.arguments.ParserArgs Class Reference

Public Member Functions

- None __init__ (self, ∗∗kwargs)

  Creates a ParserArgs object.
- Tuple[list, int] get_args (self, List[str] or None args=None)

  Get either arguments passed to parameter args or directly from sys.argv.
- dict parameters (self, List[str] or None format=None)

  List of accepted parameters on command-line.

Public Attributes

- argv
- argn

Static Public Attributes

- list default_helper = ["-h", "--help", "-?"]
- list default_version = ["-v", "--version"]

8.5.1 Detailed Description

Definition at line 20 of file arguments.py.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 __init__()

```
None ftp.cmd.arguments.ParserArgs.__init__ (
              self,
             ** kwargs )
```

Creates a ParserArgs object.

Parameters

helper	str Helper message to be displayed
version	str Version message to be displayed

Returns

ParserArgs

Definition at line 26 of file arguments.py.

8.5.3 Member Function Documentation

8.5.3.1 get_args()

```
Tuple[list, int] ftp.cmd.arguments.ParserArgs.get_args (
              self,
             List[str] or None  args = None )
```

Get either arguments passed to parameter args or directly from sys.argv.

Parameters

args	List[str] List of arguments to be parsed

Returns

Tuple[list, int]: Tuple containg arguments and its length

Definition at line 48 of file arguments.py.

8.5.3.2 parameters()

```
dict ftp.cmd.arguments.ParserArgs.parameters (
              self,
             List[str] or None  format = None )
```

List of accepted parameters on command-line.

Parameters

format	List[str] List of accepted parameters

Returns

dict: Dictionary containing paramters and respective values

Definition at line 71 of file arguments.py.

8.5.4 Member Data Documentation

8.5.4.1 argn

```
ftp.cmd.arguments.ParserArgs.argn
```

Definition at line 59 of file arguments.py.

8.5.4.2 argv

```
ftp.cmd.arguments.ParserArgs.argv
```

Definition at line 58 of file arguments.py.

8.5.4.3 default_helper

```
list ftp.cmd.arguments.ParserArgs.default_helper = ["-h", "--help", "-?"]  [static]
```

Definition at line 22 of file arguments.py.

8.5.4.4 default_version

```
list ftp.cmd.arguments.ParserArgs.default_version = ["-v", "--version"]  [static]
```

Definition at line 23 of file arguments.py.

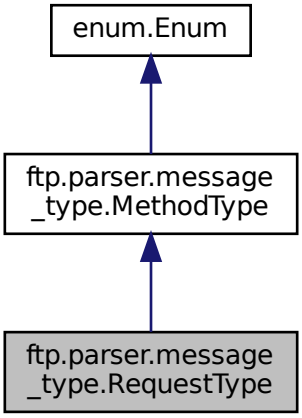
The documentation for this class was generated from the following file:

- ftp/cmd/arguments.py

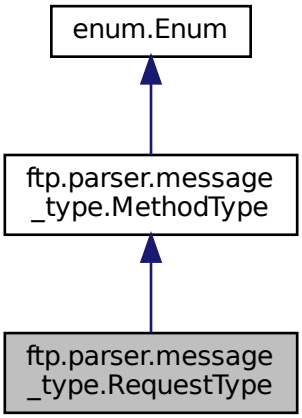
8.6 ftp.parser.message_type.RequestType Class Reference

Enum class for type of Request sent.

Inheritance diagram for ftp.parser.message_type.RequestType:



Collaboration diagram for ftp.parser.message_type.RequestType:



Public Member Functions

- def get_format (self)

  Format as the number of bits it is expected of the given method type.

Static Public Attributes

- string PUT = "000"
- string GET = "001"
- string CHANGE = "010"
- string HELP = "011"

8.6.1 Detailed Description

Enum class for type of Request sent.

Definition at line 32 of file message_type.py.

8.6.2 Member Function Documentation

8.6.2.1 get_format()

```
def ftp.parser.message_type.RequestType.get_format (
              self )
```

Format as the number of bits it is expected of the given method type.

Returns

List[int]: List containing number of bits per packet

Reimplemented from ftp.parser.message_type.MethodType.

Definition at line 41 of file message_type.py.

8.6.3 Member Data Documentation

8.6.3.1 CHANGE

```
string ftp.parser.message_type.RequestType.CHANGE = "010"  [static]
```

Definition at line 38 of file message_type.py.

8.6.3.2 GET

```
string ftp.parser.message_type.RequestType.GET = "001"  [static]
```

Definition at line 37 of file message_type.py.

8.6.3.3 HELP

```
string ftp.parser.message_type.RequestType.HELP = "011"  [static]
```

Definition at line 39 of file message_type.py.

8.6.3.4 PUT

```
string ftp.parser.message_type.RequestType.PUT = "000"  [static]
```

Definition at line 36 of file message_type.py.

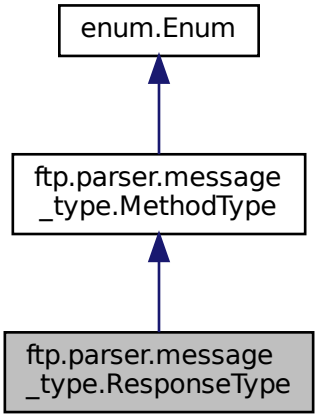
The documentation for this class was generated from the following file:

- ftp/parser/message_type.py

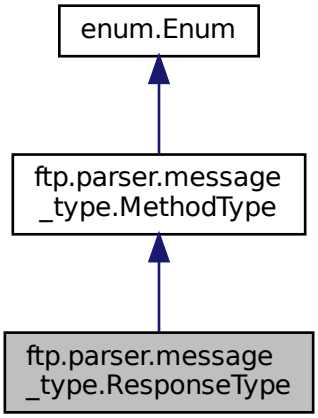
8.7 ftp.parser.message_type.ResponseType Class Reference

Enum class for type of Response sent.

Inheritance diagram for ftp.parser.message_type.ResponseType:



Collaboration diagram for ftp.parser.message_type.ResponseType:



Public Member Functions

- def get_format (self)

  Format as the number of bits it is expected of the given method type.

Static Public Attributes

- string OK_PUT_CHANGE = "000"
- string OK_GET = "001"
- string ERROR_NOT_FOUND = "010"
- string ERROR_UNKNOWN = "011"
- string ERROR_NO_CHANGE = "101"
- string HELP = "110"

8.7.1 Detailed Description

Enum class for type of Response sent.

Definition at line 58 of file message_type.py.

8.7.2 Member Function Documentation

8.7.2.1 get_format()

```
def ftp.parser.message_type.ResponseType.get_format (
              self )
```

Format as the number of bits it is expected of the given method type.

Returns

List[int]: List containing number of bits per packet

Reimplemented from ftp.parser.message_type.MethodType.

Definition at line 69 of file message_type.py.

8.7.3 Member Data Documentation

8.7.3.1 ERROR_NO_CHANGE

```
string ftp.parser.message_type.ResponseType.ERROR_NO_CHANGE = "101"  [static]
```

Definition at line 66 of file message_type.py.

8.7.3.2 ERROR_NOT_FOUND

```
string ftp.parser.message_type.ResponseType.ERROR_NOT_FOUND = "010"  [static]
```

Definition at line 64 of file message_type.py.

8.7.3.3 ERROR_UNKNOWN

```
string ftp.parser.message_type.ResponseType.ERROR_UNKNOWN = "011"  [static]
```

Definition at line 65 of file message_type.py.

8.7.3.4 HELP

```
string ftp.parser.message_type.ResponseType.HELP = "110"  [static]
```

Definition at line 67 of file message_type.py.

8.7.3.5 OK_GET

```
string ftp.parser.message_type.ResponseType.OK_GET = "001"  [static]
```

Definition at line 63 of file message_type.py.

8.7.3.6 OK_PUT_CHANGE

```
string ftp.parser.message_type.ResponseType.OK_PUT_CHANGE = "000"  [static]
```

Definition at line 62 of file message_type.py.

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

- ftp/parser/message_type.py

8.8 ftp.tcp.client.TcpClient Class Reference

Public Member Functions

- None __init__ (self, ip_addr, ∗∗kwargs)

  TCP Client interface that creates a new socket given the ip address provided for FTP communication.
- def connect (self)

  Creates a TCP socket bounded to the given IP address and port provided.
- def handle_connection (self)

  Internal function which handles all single client-server communication Method is responsible for parsing any incoming and outgoing message sent through the TCP socket.
- Message check_response (self, bytes data)

  Deserialize the incoming message.
- def on_send (self, ∗Tuple[MethodType, Callable[[List[str], MethodType], bytes]] args)

  Attach a callback that is called when a message is sent.
- def on_response (self, ∗Tuple[MethodType, Callable[[Message], None]] args)

  Attach a callback that is called when a message is received.
- List[str] cin (self)

  Reads stdin from command-line.
- def handler (self, signum, frame)

  Internal funtion used to define a signal handler that is called when a SIGINT signal is raised by this process.

Public Attributes

- thread
- socket
- ip_address

8.8.1 Detailed Description

Definition at line 27 of file client.py.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 __init__()

```
None ftp.tcp.client.TcpClient.__init__ (
              self,
              ip_addr,
             ** kwargs )
```

TCP Client interface that creates a new socket given the ip address provided for FTP communication.

Parameters

ip_addr	str IP address which the TCP service would be listening to
-p	int Port value which socket will bind its connection
-a	str IP address that TCP service will be using

Returns

TcpClient

Definition at line 31 of file client.py.

8.8.3 Member Function Documentation

8.8.3.1 check_response()

```
Message ftp.tcp.client.TcpClient.check_response (
              self,
             bytes data )
```

Deserialize the incoming message.

Parameters

data	bytes Byte object received by the socket

Returns

Deserialized message object

Definition at line 154 of file client.py.

8.8.3.2 cin()

```
List[str] ftp.tcp.client.TcpClient.cin (
              self )
```

Reads stdin from command-line.

By default prints 'ftp>' before reading input. Note that default input function is a blocking stdin command.

Returns

List[str] List of message inputs

Definition at line 188 of file client.py.

8.8.3.3 connect()

```
def ftp.tcp.client.TcpClient.connect (
              self )
```

Creates a TCP socket bounded to the given IP address and port provided.

Object creates a new thread where each new connection will respond to.

Returns

None

Definition at line 60 of file client.py.

8.8.3.4 handle_connection()

```
def ftp.tcp.client.TcpClient.handle_connection (
              self )
```

Internal function which handles all single client-server communication Method is responsible for parsing any incoming and outgoing message sent through the TCP socket.

Returns

None

Definition at line 86 of file client.py.

8.8.3.5 handler()

```
def ftp.tcp.client.TcpClient.handler (
              self,
              signum,
              frame )
```

Internal funtion used to define a signal handler that is called when a SIGINT signal is raised by this process.

Definition at line 203 of file client.py.

8.8.3.6 on_response()

```
def ftp.tcp.client.TcpClient.on_response (
              self,
             *Tuple[MethodType, Callable[[Message], None ]]  args )
```

Attach a callback that is called when a message is received.

Parameters

∗args	List[Tuple[MethodType, Callable[[Message], None]]] List of callable objects containing its Method type and respective callback function

Returns

None

Definition at line 176 of file client.py.

8.8.3.7 on_send()

```
def ftp.tcp.client.TcpClient.on_send (
              self,
             *Tuple[ MethodType ,Callable[[List[str], MethodType], bytes]] args )
```

Attach a callback that is called when a message is sent.

Parameters

∗args	List[Tuple[MethodType ,Callable[[List[str], MethodType], bytes]]] List of callable objects containing its Method type and respective callback function

Returns

None

Definition at line 165 of file client.py.

8.8.4 Member Data Documentation

8.8.4.1 ip_address

```
ftp.tcp.client.TcpClient.ip_address
```

Definition at line 52 of file client.py.

8.8.4.2 socket

```
ftp.tcp.client.TcpClient.socket
```

Definition at line 43 of file client.py.

8.8.4.3 thread

`ftp.tcp.client.TcpClient.thread`

Definition at line 42 of file client.py.

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

- ftp/tcp/client.py

8.9 ftp.tcp.server.TcpServer Class Reference

Public Member Functions

- None __init__ (self, ip_addr, ∗∗kwargs)

  TCP Server interface that creates a new socket given the ip address provided for FTP communication purposes.
- def listen (self)

  Starts a new threded TCP service by connecting to the respective server.
- def handle_listen (self, sok.socket conn, sok.AddressInfo addr)

  Internal function that is responsible for parsing any incoming and outgoing message sent to the TCP socket.
- def on_receive (self, ∗Callable[[sok.AddressInfo, Message], bytes] args)

  Attach a callback that is called when a message is received.

Static Public Member Functions

- Message parse_packet (bytes data)

  Deserializes incoming byte received by the TCP socket.

Public Attributes

- thread
- socket
- ip_address
- is_connected

8.9.1 Detailed Description

Definition at line 26 of file server.py.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 __init__()

```
None ftp.tcp.server.TcpServer.__init__ (
              self,
              ip_addr,
             ** kwargs )
```

TCP Server interface that creates a new socket given the ip address provided for FTP communication purposes.

Parameters

ip_addr	str IP address which the TCP service would be listening to
-p	int Port value which socket will bind its connection
-a	str IP address that TCP service will be using

Returns

TcpServer

Definition at line 30 of file server.py.

8.9.3 Member Function Documentation

8.9.3.1 handle_listen()

```
def ftp.tcp.server.TcpServer.handle_listen (
             self,
            sok.socket conn,
            sok.AddressInfo addr )
```

Internal function that is responsible for parsing any incoming and outgoing message sent to the TCP socket.

Returns

None

Definition at line 96 of file server.py.

8.9.3.2 listen()

```
def ftp.tcp.server.TcpServer.listen (
             self )
```

Starts a new threded TCP service by connecting to the respective server.

Returns

None

Definition at line 65 of file server.py.

8.9.3.3 on_receive()

```
def ftp.tcp.server.TcpServer.on_receive (
             self,
            *Callable[[sok.AddressInfo, Message], bytes] args )
```

Attach a callback that is called when a message is received.

Parameters

∗args	List[Callable[[sok.AddressInfo, Message], bytes]] List of callable objects containing its Method type and respective callback function

Definition at line 135 of file server.py.

8.9.3.4 parse_packet()

```
Message ftp.tcp.server.TcpServer.parse_packet (
            bytes data )  [static]
```

Deserializes incoming byte received by the TCP socket.

Parameters

data	bytes Byte object received by socket

Returns

Message: object

Definition at line 145 of file server.py.

8.9.4 Member Data Documentation

8.9.4.1 ip_address

```
ftp.tcp.server.TcpServer.ip_address
```

Definition at line 52 of file server.py.

8.9.4.2 is_connected

```
ftp.tcp.server.TcpServer.is_connected
```

Definition at line 55 of file server.py.

8.9.4.3 socket

`ftp.tcp.server.TcpServer.socket`

Definition at line 42 of file server.py.

8.9.4.4 thread

`ftp.tcp.server.TcpServer.thread`

Definition at line 41 of file server.py.

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

- ftp/tcp/server.py

8.10 ftp.parser.message.Util Class Reference

Static Public Member Functions

- bytes serialize (Message msg)

  Serializes a Message object into bytes following its binary representation.
- Message deserialize (bytes binary, MessageType type)

  Deserialize the byte value into a Message object.
- str str2bit (str val, int size, bool with_count=True, int or None size_count=None)

  Converts a string value its equivalent binary representation in utf-8 encoding.
- List[bytes] bit2byte (Message msg)

  Converts a Message object to a list of byte equivalent values.

8.10.1 Detailed Description

Definition at line 129 of file message.py.

8.10.2 Member Function Documentation

8.10.2.1 bit2byte()

```
List[bytes] ftp.parser.message.Util.bit2byte (
            Message msg )  [static]
```

Converts a Message object to a list of byte equivalent values.

Parameters

msg	Message Message object to be converted

Returns

List[bytes]: List of bytes containing all packets encapsulated by the Message object

Definition at line 257 of file message.py.

8.10.2.2 deserialize()

```
Message ftp.parser.message.Util.deserialize (
            bytes binary,
            MessageType type )  [static]
```

Deserialize the byte value into a Message object.

Parameters

binary	bytes Input byte object to be deserialized
type	MessageType Type of message that is to be deserialized

Returns

Message: Returns a message object

Definition at line 162 of file message.py.

8.10.2.3 serialize()

```
bytes ftp.parser.message.Util.serialize (
            Message msg )  [static]
```

Serializes a Message object into bytes following its binary representation.

Parameters

msg	Message Message object

Returns

bytes: Returns the extended byte representation of that message object

Definition at line 132 of file message.py.

8.10.2.4 str2bit()

```
str ftp.parser.message.Util.str2bit (
            str val,
            int size,
            bool  with_count = True,
            int or None  size_count = None )  [static]
```

Converts a string value its equivalent binary representation in utf-8 encoding.

Note that python does not use standard representation of variable size. Hence, manual conversion should be done.

Parameters

val	str String to be converted
size	int Size of the expected binary value
with_count	bool Attach the binary represented size of paramters val
size_count	int Size of the binary represented value for the portion representing the size of val

Returns

str: Retuns a binary representation of the value passed

Definition at line 215 of file message.py.

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

- ftp/parser/message.py

Chapter 9

File Documentation

9.1 ftp/cmd/arguments.py File Reference

Parses arguemnts provided in command-line.

Classes

- class ftp.cmd.arguments.ParserArgs

Namespaces

- namespace ftp
- namespace ftp.cmd
- namespace ftp.cmd.arguments

9.1.1 Detailed Description

Parses arguemnts provided in command-line.

9.1.2 Author(s)

- Created by Kevin de Oliveira on 04/01/2022.
- Student ID: 40054907

Definition in file arguments.py.

9.2 arguments.py

Go to the documentation of this file.

```
00001
00012
00013 import sys
00014 from types import FunctionType
00015 from typing import List, Tuple
00016
00017
00018
00019
00020 class ParserArgs:
00021
00022     default_helper = ["-h", "--help", "-?"]
00023     default_version = ["-v", "--version"]
00024
00025
00026     def __init__(self, **kwargs) -> None:
00027         """!
00028         Creates a ParserArgs object
00029
00030
00031         @param helper: str      Helper message to be displayed
00032         @param version: str     Version message to be displayed
00033
00034         @return ParserArgs
00035         """
00036         self.argv : list = []
00037         self.argn : int = 0
00038         self._helper : str = ""
00039
00040
00041         if kwargs.__len__ == 0 :
00042             return
00043
00044         self._helper = kwargs.get("helper")
00045         self._version = kwargs.get("version")
00046
00047
00048     def get_args(self, args : List[str] or None = None) -> Tuple[list, int]:
00049         """!
00050         Get either arguments passed to parameter args or directly from sys.argv
00051
00052         @param args: List[str]      List of arguments to be parsed
00053
00054         @return Tuple[list, int]:   Tuple containg arguments and its length
00055         """
00056
00057         if args is None or args.__len__ <= 1 :
00058             self.argv = sys.argv[1:]
00059             self.argn = len(sys.argv)
00060             return (self.argv, self.argn)
00061
00062         self.argv = args[1:]
00063         self.argn = len(args)
00064
00065         return (self.argv, self.argn)
00066
00067
00068
00069
00070
00071     def parameters(self, format : List[str] or None = None) -> dict:
00072         """!
00073         List of accepted parameters on command-line
00074
00075         @param format: List[str]    List of accepted parameters
00076
00077         @return dict: Dictionary containing paramters and respective values
00078         """
00079         temp = {}
00080
00081         for i,x in enumerate(self.argv):
00082
00083             if x in self.default_helper:
00084                 sys.exit(self._helper) if self._helper else 0
00085             elif x in self.default_version:
00086                 sys.exit(self._version) if self._version else 0
00087
00088             if format and x[0] == "-" and x not in format:
00089                 sys.exit("unknown option: "+x+"\n"+self._helper)
00090
00091             elif format and x in format and i < self.argn - 1:
00092                 try:
```

```
00093                     temp.update({
00094                         x: self.argv[i + 1]
00095                     })
00096                 except IndexError:
00097                     temp.update({
00098                         x: True
00099                     })
00100             else:
00101                 sys.exit("unknown option: "+x+"\n"+self._helper)
00102 
00103 
00104         return temp
```

9.3 ftp/__init__.py File Reference

9.4 __init__.py

Go to the documentation of this file.

9.5 ftp/cmd/__init__.py File Reference

Namespaces

- namespace ftp
- namespace ftp.cmd

9.6 __init__.py

Go to the documentation of this file.

```
00001 """
00002 Module responsible for parsing command-line arguments
00003 """
```

9.7 ftp/parser/__init__.py File Reference

Namespaces

- namespace ftp
- namespace ftp.parser

9.8 __init__.py

Go to the documentation of this file.

```
00001 """
00002 Module defining ADT objects used for TCP service
00003 """
```

9.9 ftp/tcp/__init__.py File Reference

Namespaces

- namespace ftp
- namespace ftp.tcp

9.10 __init__.py

Go to the documentation of this file.

```
00001 """
00002 Module responsible managing and craeting TCP services
00003 """
```

9.11 ftp/parser/message.py File Reference

Segment containing the packets for Request and Response messages.

Classes

- class ftp.parser.message.Message
- class ftp.parser.message.Util

Namespaces

- namespace ftp
- namespace ftp.parser
- namespace ftp.parser.message

9.11.1 Detailed Description

Segment containing the packets for Request and Response messages.

9.11.2 Author(s)

- Created by Kevin de Oliveira on 04/01/2022.
- Student ID: 40054907

Definition in file message.py.

9.12 message.py

Go to the documentation of this file.

```
00001
00012
00013 import math
00014 from typing import List, Tuple
00015
00016 import bitarray
00017 from bitarray import util
00018 from ftp.parser.message_type import MessageType, MethodType, RequestType, ResponseType
00019
00020 from ftp.parser.packet import packet
00021
00022 class Message:
00023     def __init__(self, header_size : int, type: MethodType) -> None:
00024         """!
00025         Encapsulates a packet object that is sent to the socket
00026
00027         message is represented as: [header | data_1 | data_2 | ... | payload?]
00028
00029         @param header_size: int     Size of the header packet
00030         @param type: MethodType     Type of message that will be sent
00031
00032         @return Returns None
00033         """
00034         self.header = packet(header_size)
00035
00036         self.data : list[packet] = []
00037         self.type : MethodType = type
00038
00039         self.header(self.type.value)
00040
00041         self.size = header_size
00042
00043         self.payload : packet or None = None
00044
00045
00046
00047         self._token(
00048             self.type.get_format()
00049         )
00050
00051
00052     def _token(self, format : List[int]):
00053         """!
00054         Internal function to include data into the message
00055
00056         @param format: List[int]       List containing format of each packet encapsulated by this
      message
00057         """
00058         if len(format) == 0:
00059             return None
00060
00061
00062         for x in format:
00063             self.data.append(
00064                 packet(x)
00065             )
00066
00067         self.size += sum(format)
00068
00069
00070     def parse(self, value : str) -> Tuple[packet, List[packet]] or None:
00071         """!
00072         Parses input values into its respective data field
00073
00074         @param value: str           Inserts the values its respective data. Note that value must be
      represented in binary format
00075
00076         @return Tuple or None: An optional tuple containing the header packet and its data packets
00077         """
00078         if(value.__len__ == 0 or len(value) > self.size):
00079             return None
00080
00081         if not len(self.data):
00082             return None
00083
00084         ind = 0
00085
00086         for i, x in enumerate(self.data):
00087             x(value[ind :  ind + x.size ])
00088             ind += x.size
00089
00090
```

```
00091         return (self.header, self.data)
00092
00093     def add_payload(self, value : str) -> None:
00094         """!
00095         Attach payload to this message
00096
00097         @param value: str       Value to be parsed into packet. Note that value must be represented in
      binary format
00098
00099         @return None
00100         """
00101         temp = bitarray.util.hex2ba( value.encode("utf-8").hex() ).to01()
00102
00103         self.payload = packet(len(temp))
00104         self.payload(temp)
00105
00106     def has_payload(self) -> bool:
00107         """!
00108         Verifies if this message has a payload attached
00109
00110         @return bool: Returns true if contains a paylaod; otherwise false
00111         """
00112         return self.payload is not None
00113
00114
00115
00116     def __repr__(self) -> str:
00117         temp = "["+ self.header.__str__() +"]"
00118         for x in self.data:
00119             temp += x.__str__()
00120         return  temp
00121
00122     def __str__(self) -> str:
00123         temp = "["+ self.header.__str__() +"]"
00124         for x in self.data:
00125             temp += x.__str__()
00126         return  temp
00127
00128
00129 class Util:
00130
00131     @staticmethod
00132     def serialize(msg : Message) -> bytes:
00133         """!
00134         Serializes a Message object into bytes following its binary representation
00135
00136         @param msg: Message     Message object
00137
00138         @return bytes:          Returns the extended byte representation of that message object
00139         """
00140         binary : bitarray.bitarray = bitarray.bitarray(endian="big")
00141
00142         binary.extend(
00143             msg.header.body
00144         )
00145
00146
00147         for x in msg.data:
00148             binary.extend(
00149                 x.body
00150             )
00151
00152         if msg.has_payload():
00153             binary.extend(
00154                 msg.payload.body
00155             )
00156
00157
00158         return util.serialize(binary)
00159
00160
00161     @staticmethod
00162     def deserialize(binary : bytes, type : MessageType) -> Message:
00163         """!
00164         Deserialize the byte value into a Message object
00165
00166         @param  binary: bytes       Input byte object to be deserialized
00167         @param  type: MessageType   Type of message that is to be deserialized
00168
00169         @return Message:            Returns a message object
00170         """
00171
00172         temp : bitarray.bitarray = util.deserialize(binary)
00173
00174         if type == MessageType.REQUEST:
00175             msg = Message(3, RequestType(temp.to01()[:3]))
00176
```

```
00177             if msg.type == RequestType.PUT:
00178                 msg.parse(
00179                     temp.to01()[3:msg.size]
00180                 )
00181                 pk = packet(len(temp.to01()[msg.size:]))
00182                 pk(temp.to01()[msg.size:])
00183                 msg.payload = pk
00184             else:
00185                 msg.parse(
00186                     temp.to01()[3:]
00187                 )
00188 
00189             return msg
00190 
00191         elif type == MessageType.RESPONSE:
00192             msg = Message(3, ResponseType(temp.to01()[:3]))
00193             if msg.type == ResponseType.OK_GET:
00194                 msg.parse(
00195                     temp.to01()[3:msg.size]
00196                 )
00197                 pk = packet(len(temp.to01()[msg.size:]))
00198                 pk(temp.to01()[msg.size:])
00199                 msg.payload = pk
00200 
00201             else:
00202                 msg.parse(
00203                     temp.to01()[3:]
00204                 )
00205 
00206             return msg
00207 
00208         else:
00209             raise ValueError("Incorrect Message type")
00210 
00211 
00212 
00213 
00214     @staticmethod
00215     def str2bit(val : str, size : int, with_count : bool = True, size_count : int or None = None ) ->
    str:
00216         """!
00217         Converts a string value its equivalent binary representation in utf-8 encoding.
00218 
00219         Note that python does not use standard representation of variable size.
00220         Hence, manual conversion should be done.
00221 
00222         @param val: str         String to be converted
00223         @param size: int        Size of the expected binary value
00224         @param with_count: bool Attach the binary represented size of paramters val
00225         @param size_count: int  Size of the binary represented value for the portion representing the
    size of val
00226 
00227         @return str:            Retuns a binary representation of the value passed
00228         """
00229         val = val.strip()
00230 
00231         data : str = util.hex2ba( val.encode("utf-8").hex() ).to01()
00232 
00233         if(len(data) > size): raise ValueError("Value passed is too big")
00234 
00235 
00236         if(len(data) < size):
00237             for _ in range(0, size - len(data)):
00238                 data = "0" + data
00239 
00240 
00241         if with_count:
00242             count : int = util.int2ba(len(val)).to01()
00243 
00244             if size_count:
00245                 count_size = size_count
00246             else:
00247                 count_size = int( math.log2( int(size / 8) ) )
00248 
00249             if(len(count) < count_size):
00250                 for _ in range(0, count_size - len(count)):
00251                     count = "0" + count
00252 
00253             return count + data
00254         else:
00255             return data
00256     @staticmethod
00257     def bit2byte(msg : Message) -> List[bytes]:
00258         """!
00259         Converts a Message object to a list of byte equivalent values
00260 
00261         @param msg: Message     Message object to be converted
```

```
00262
00263         @return List[bytes]:    List of bytes containing all packets encapsulated by the Message
      object
00264         """
00265         temp : list[bytes] = []
00266         for x in msg.data:
00267             temp.append(
00268                 x.body.tobytes()
00269             )
00270         if msg.has_payload():
00271             temp.append(
00272                 msg.payload.body.tobytes()
00273             )
00274
00275         return temp
00276
00277
00278
```

9.13 ftp/parser/message_type.py File Reference

Enumaration classes for each method type used during FTP communication.

Classes

- class ftp.parser.message_type.MessageType

  Enum class for type of message sent.
- class ftp.parser.message_type.MethodType
- class ftp.parser.message_type.RequestType

  Enum class for type of Request sent.
- class ftp.parser.message_type.ResponseType

  Enum class for type of Response sent.

Namespaces

- namespace ftp
- namespace ftp.parser
- namespace ftp.parser.message_type

9.13.1 Detailed Description

Enumaration classes for each method type used during FTP communication.

9.13.2 Author(s)

- Created by Kevin de Oliveira on 04/01/2022.
- Student ID: 40054907

Definition in file message_type.py.

9.14 message_type.py

Go to the documentation of this file.

```
00001
00012
00013 from abc import abstractmethod
00014 import enum
00015 from typing import List
00016
00017
00018
00019 class MessageType(enum.Enum):
00020     """!
00021     Enum class for type of message sent
00022     """
00023     REQUEST = 0
00024     RESPONSE = 1
00025
00026 class MethodType(enum.Enum):
00027     @abstractmethod
00028     def get_format(self) -> List[int]:
00029         pass
00030
00031
00032 class RequestType(MethodType):
00033     """!
00034     Enum class for type of Request sent
00035     """
00036     PUT = "000"
00037     GET = "001"
00038     CHANGE = "010"
00039     HELP = "011"
00040
00041     def get_format(self):
00042         """!
00043         Format as the number of bits it is expected of the given method type
00044
00045         @return List[int]: List containing number of bits per packet
00046         """
00047         if self == RequestType.PUT:
00048             return [5, 32 * 8, 4 * 8]
00049         elif self == RequestType.GET:
00050             return [5, 32*8]
00051         elif self == RequestType.CHANGE:
00052             return [5, 32 * 8, 8, 32 * 8]
00053         elif self == RequestType.HELP :
00054             return [5]
00055         else:
00056             return []
00057
00058 class ResponseType(MethodType):
00059     """!
00060     Enum class for type of Response sent
00061     """
00062     OK_PUT_CHANGE = "000"
00063     OK_GET = "001"
00064     ERROR_NOT_FOUND = "010"
00065     ERROR_UNKNOWN = "011"
00066     ERROR_NO_CHANGE = "101"
00067     HELP = "110"
00068
00069     def get_format(self):
00070         """!
00071         Format as the number of bits it is expected of the given method type
00072
00073         @return List[int]: List containing number of bits per packet
00074         """
00075         if self == ResponseType.OK_PUT_CHANGE:
00076             return [5]
00077         elif self == ResponseType.OK_GET:
00078             return [5, 32*8, 4*8]
00079         elif self == ResponseType.ERROR_NOT_FOUND:
00080             return [5]
00081         elif self == ResponseType.ERROR_UNKNOWN:
00082             return [5]
00083         elif self == ResponseType.ERROR_NO_CHANGE:
00084             return [5]
00085         elif self == ResponseType.HELP :
00086             return [5, 32 * 8]
00087         else:
00088             return []
```

9.15 ftp/parser/packet.py File Reference

Classes

- class ftp.parser.packet.packet

Namespaces

- namespace ftp
- namespace ftp.parser
- namespace ftp.parser.packet

9.16 packet.py

Go to the documentation of this file.

```
00001
00012
00013 import bitarray
00014
00015
00016 class packet:
00017
00018
00019     def __init__(self, max_bits : int, fill : bool = False, *args) -> None:
00020         """!
00021         Object representing a packet as binary
00022
00023         @param max_bits: int    Maximum size of this packet object
00024         @param fill: bool       Align the length of this packet to a multiple of 8 (1 byte)
00025
00026         @return packet
00027         """
00028         self.body = bitarray.bitarray(max_bits)
00029         self.size = max_bits
00030         self.body.setall(0)
00031         if fill:
00032             self.body.fill()
00033             self.size = len(self.body)
00034
00035
00036     def __call__(self, val : str) -> 'packet':
00037         """!
00038         Append value to this object
00039
00040         @param val: str Value to be appended
00041
00042         @return packet
00043         """
00044         if len(val) != self.size:
00045             raise ValueError("Incorrect size provided")
00046
00047         self.body = bitarray.bitarray(val)
00048         return self
00049
00050     def value(self) -> bitarray.bitarray:
00051         """!
00052         Getter of the body attribute
00053
00054         @return Bitarray object
00055         """
00056         return self.body
00057
00058     def to_bytes(self) -> bytes:
00059         """!
00060         Returns the byte value of the body attribute
00061
00062         @return Byte object
00063         """
00064         return self.body.tobytes()
00065
00066
00067     def set_size(self, val: int) -> None:
```

```
00068         """!
00069         Redefine the packet object size
00070
00071         @param val: int Size of this packet object
00072
00073         @return None
00074         """
00075         self.body = bitarray.bitarray(val)
00076         self.body.clear()
00077         self.size = val
00078
00079
00080     def __str__(self) -> str:
00081         return self.body.to01()
00082
00083
00084     def __repr__(self) -> str:
00085         return self.body.to01()
00086
00087
```

9.17 ftp/tcp/client.py File Reference

TCP Client logic and algorithm.

Classes

- class ftp.tcp.client.TcpClient

Namespaces

- namespace ftp
- namespace ftp.tcp
- namespace ftp.tcp.client

9.17.1 Detailed Description

TCP Client logic and algorithm.

9.17.2 Author(s)

- Created by Kevin de Oliveira on 04/01/2022.
- Student ID: 40054907

Definition in file client.py.

9.18 client.py

Go to the documentation of this file.

```
00001
00012
00013 import socket as sok
00014 import sys
00015 from threading import Thread
00016 from types import FunctionType
00017 from typing import Callable, List, Optional, Tuple
00018 from ftp.parser.message import Message, Util
00019
00020 from ftp.parser.message_type import MessageType, MethodType, RequestType, ResponseType
00021
00022 import signal
00023
00024
00025
00026
00027 class TcpClient():
00028     _DEFAULT_PORT = 1025
00029     _MAX_BUFFER = 4096
00030
00031     def __init__(self, ip_addr, **kwargs) -> None:
00032         """!
00033         TCP Client interface that creates a new socket given the ip address provided for FTP
      communication
00034
00035         @param ip_addr: str     IP address which the TCP service would be listening to
00036         @param -p: int          Port value which socket will bind its connection
00037         @param -a: str          IP address that TCP service will be using
00038
00039         @return TcpClient
00040         """
00041         self._debug = False
00042         self.thread = None
00043         self.socket = sok.socket(sok.AF_INET, sok.SOCK_STREAM)
00044
00045         for k,v in kwargs.items():
00046             if "-p" in k:
00047                 self._DEFAULT_PORT_DEFAULT_PORT = int(v)
00048             elif "-a" in k:
00049                 ip_addr = v
00050             elif "-d" in k:
00051                 self._debug = True
00052         self.ip_address = ip_addr
00053         self._is_connected = False
00054         self.create_message_functions : List[Tuple[MethodType, FunctionType] ] = []
00055         self.on_response_functions: List[Tuple[MethodType, FunctionType]] = []
00056
00057         signal.signal(signal.SIGINT, self.handler)
00058
00059
00060     def connect(self):
00061         """!
00062         Creates a TCP socket bounded to the given IP address and port provided.
00063         Object creates a new thread where each new connection will respond to.
00064
00065         @return None
00066         """
00067         try:
00068
00069             self.socket.connect( (self.ip_address, self._DEFAULT_PORT_DEFAULT_PORT) )
00070             self._is_connected = True
00071             self.thread = Thread(target = self.handle_connection, args=())
00072             self.thread.start()
00073             if self.thread:
00074                 self.thread.join()
00075
00076         except ConnectionRefusedError:
00077             # @brief Implement timeout or repetition
00078             print("Unable to connect to server, try again")
00079         except KeyboardInterrupt:
00080             self.socket.close()
00081             self._is_connected = False
00082         except Exception:
00083             return
00084
00085
00086     def handle_connection(self):
00087         """!
00088         Internal function which handles all single client-server communication
00089         Method is responsible for parsing any incoming and outgoing message sent through the TCP
      socket
00090
```

```
00091         @return None
00092         """
00093         lcls = locals()
00094         cmd_format : RequestType or None = None
00095         if self._debug:
00096             print("Debug mode ON")
00097         while self._is_connected:
00098             try:
00099                 msg = self.cin()
00100                 if len(msg):
00101                     try:
00102                         # @brief Dynamically converts the input string to local MethodType variable
00103                         exec("cmd_format_lcls = RequestType.%s" % msg[0].upper(), globals(), lcls)
00104                         cmd_format = lcls["cmd_format_lcls"]
00105                         for x in self.create_message_functions:
00106                             if(x[0] == cmd_format):
00107                                 _data_send = x[1](msg, cmd_format)
00108                                 if self._debug:
00109                                     print("[DEBUG]", _data_send)
00110 
00111                                 self.socket.send( _data_send )
00112                     except (AttributeError ,SyntaxError, IndexError) as e:
00113                         # @brief self.socket.send not necessary as the client application should be
    aware of the supported commands (eg. offline). However, due to requirements empty message is sent.
00114                         if msg[0] == "bye":
00115                             raise KeyboardInterrupt()
00116 
00117                         self.socket.send(
00118                             " ".join(msg).encode("utf-8")
00119                         )
00120                     except OSError as e:
00121                         print("OS Error occured: ",e)
00122                         return
00123                     except ValueError as e:
00124                         print(e)
00125                         continue
00126 
00127                     # @brief Socket is only able to receive MAX_BUFFER;
00128                     # In order to guarantee efficiency of sockets, longer buffers will be stripped to
    fit the maximum sendable buffer size
00129                     # Therefore, ensure that maximum packet sent is no longer then _MAX_BUFFER or set
    socket to nonblocking -> socket.setblocking(False)
00130                     # Otherwise, wait for any possible subsequent packet receival (parallel or
    different loop)
00131 
00132                     recv = self.socket.recv(self._MAX_BUFFER)
00133 
00134                     if self._debug:
00135                         print("[DEBUG]", recv)
00136                     if recv:
00137 
00138                         res = self.check_response(recv)
00139                         if type(res.type) is ResponseType:
00140                             for x in self.on_response_functions:
00141                                 if res.type == x[0]:
00142                                     x[1](res)
00143                         else:
00144                             self._is_connected = False
00145                     else:
00146                         self._is_connected = False
00147 
00148             except (KeyboardInterrupt, OSError):
00149                 return
00150             except ValueError as e:
00151                 #@brief In case of large incming packet, self.socke.recv will raise an Exception for
    invalid header byte
00152                 print("invalid response", e)
00153 
00154     def check_response(self, data : bytes) -> Message:
00155         """!
00156         Deserialize the incoming message
00157 
00158         @param data: bytes  Byte object received by the socket
00159 
00160         @return Deserialized message object
00161         """
00162         return Util.deserialize(data, MessageType.RESPONSE)
00163 
00164 
00165     def on_send(self, *args: Tuple[ MethodType ,Callable[[List[str], MethodType], bytes]]):
00166         """!
00167         Attach a callback that is called when a message is sent
00168 
00169         @param *args: List[Tuple[ MethodType ,Callable[[List[str], MethodType], bytes]]]    List of
    callable objects containing its Method type and respective callback function
00170 
00171         @return None
```

```
00172         """
00173         for x in args:
00174             self.create_message_functions.append(x)
00175 
00176     def on_response(self, *args: Tuple[MethodType,  Callable[[Message], None ]] ):
00177         """!
00178         Attach a callback that is called when a message is received
00179 
00180         @param *args: List[Tuple[MethodType,  Callable[[Message], None ]]]  List of callable objects
      containing its Method type and respective callback function
00181 
00182         @return None
00183         """
00184         for x in args:
00185             self.on_response_functions.append(x)
00186 
00187 
00188     def cin(self) -> List[str]:
00189         """!
00190         Reads stdin from command-line. By default prints 'ftp>' before reading input.
00191         Note that default input function is a blocking stdin command.
00192 
00193         @return List[str] List of message inputs
00194         """
00195         try:
00196             msg = input("ftp> ")
00197             return msg.strip().split()
00198         except EOFError:
00199             raise KeyboardInterrupt
00200 
00201 
00202 
00203     def handler(self, signum , frame):
00204         """!
00205         Internal funtion used to define a signal handler that is called when a SIGINT signal is raised
      by this process
00206         """
00207 
00208         #stdin is locking. Hence after SIGINT the application will still hang
00209         #probable solution would be implementing the thread as daemon
00210 
00211         print("\nClosing FTP Connection. Press [ENTER] to finish")
00212         self._is_connected = False
00213         self.socket.close()
00214 
00215         # sys.stdin.close()
```

9.19 ftp/tcp/server.py File Reference

TCP Server logic and algorithm.

Classes

- class ftp.tcp.server.TcpServer

Namespaces

- namespace ftp
- namespace ftp.tcp
- namespace ftp.tcp.server

9.19.1 Detailed Description

TCP Server logic and algorithm.

9.19.2 Author(s)

- Created by Kevin de Oliveira on 04/01/2022.
- Student ID: 40054907

Definition in file server.py.

9.20 server.py

Go to the documentation of this file.

```
00001
00012
00013 from ast import arg
00014 import pickle
00015 import socket as sok
00016 import sys
00017 from threading import Thread
00018 from types import FunctionType
00019 from typing import Any, Callable, List, Tuple
00020 from bitarray import util
00021
00022 from ftp.parser.message import Message, Util
00023 from ftp.parser.message_type import MessageType, RequestType, ResponseType
00024
00025
00026 class TcpServer():
00027     _DEFAULT_PORT = 1025
00028     _MAX_BUFFER = 4096
00029
00030     def __init__(self, ip_addr, **kwargs) -> None:
00031         """!
00032         TCP Server interface that creates a new socket given the ip address provided for FTP
       communication purposes
00033
00034         @param ip_addr: str     IP address which the TCP service would be listening to
00035         @param -p: int          Port value which socket will bind its connection
00036         @param -a: str          IP address that TCP service will be using
00037
00038         @return TcpServer
00039         """
00040         self._debug = False
00041         self.thread = None
00042         self.socket = sok.socket(sok.AF_INET, sok.SOCK_STREAM)
00043         self.socket.setsockopt(sok.SOL_SOCKET, sok.SO_REUSEADDR, 1)
00044         for k,v in kwargs.items():
00045             if "-p" in k:
00046                 self._DEFAULT_PORT_DEFAULT_PORT = int(v)
00047             elif "-a" in k:
00048                 ip_addr = v
00049             elif "-d" in k:
00050                 self._debug = True
00051         self.socket.bind( (ip_addr, self._DEFAULT_PORT_DEFAULT_PORT) )
00052         self.ip_address = ip_addr
00053         self.recv_functions : List[Tuple[RequestType, FunctionType]] = []
00054
00055         self.is_connected = False
00056
00057     def _init_app(self) -> str:
00058         """!
00059         Initial message printed into command-line when TCP service is initiated
00060         """
00061         return """-- FTP Server initializing on {ip}:{port}
00062 -- Version 1.0.0 by Kevin de Oliveira
00063 -- README contains a list of available commands and some concepts guiding""".format(ip =
       self.ip_address, port = self._DEFAULT_PORT_DEFAULT_PORT)
00064
00065     def listen(self):
00066         """!
00067         Starts a new threded TCP service by connecting to the respective server.
00068
00069         @return None
00070         """
00071         self.socket.listen()
```

```
00072         print(self._init_app())
00073         if self._debug:
00074             print("Debug mode ON")
00075         while True:
00076             try:
00077                 conn, addr = self.socket.accept()
00078                 self.thread = Thread(target=self.handle_listen, args=(conn, addr))
00079                 self.is_connected = True
00080                 self.thread.start()
00081 
00082                 # For non concurrent connection, join current thread so loop awaits for any active
      connection to be terminated before connecting any other socket
00083                 # if self.thread:
00084                 #     self.thread.join()
00085             except KeyboardInterrupt:
00086                     print("Closing server")
00087                     self.is_connected = False
00088                     self.socket.close()
00089                     return
00090             except BlockingIOError:
00091                 print("EAGAIN error")
00092                 continue
00093 
00094 
00095 
00096     def handle_listen(self, conn : sok.socket, addr : sok.AddressInfo):
00097         """!
00098         Internal function that is responsible for parsing any incoming and outgoing message sent to
      the TCP socket
00099 
00100         @return None
00101         """
00102         print("""> New connection {addr}:{port}""".format(addr = addr[0], port=addr[1]))
00103 
00104         while self.is_connected:
00105             try:
00106                 # Socket is only able to receive MAX_BUFFER;
00107                 # Therefore, ensure that maximum packet sent is no longer then _MAX_BUFFER or set
      socket to nonblocking -> socket.setblocking(False)
00108                 # Otherwise, wait for any possible subsequent packet receival (parallel or different
      loop)
00109                 data = conn.recv(self._MAX_BUFFER)
00110                 if not data:
00111                     print("Closing connection with:", addr)
00112                     return None
00113                 if self._debug:
00114                     print("[DEBUG]", data)
00115                 out = self.parse_packet(data)
00116 
00117                 for x in self.recv_functions:
00118                     if(x[0] == out.type):
00119                         _data_send = x[1](addr, out)
00120                         if self._debug:
00121                             print("[DEBUG]", _data_send)
00122                         conn.sendto(_data_send, addr)
00123 
00124 
00125             except (BrokenPipeError, ConnectionResetError) as e:
00126                 print(e, addr)
00127             except ValueError:
00128                 message = Message(3, ResponseType.ERROR_UNKNOWN)
00129                 message.parse("00000")
00130                 conn.sendto( Util.serialize(message) , addr )
00131             except KeyboardInterrupt:
00132                 return
00133 
00134 
00135     def on_receive(self, *args : Callable[[sok.AddressInfo, Message], bytes]):
00136         """!
00137         Attach a callback that is called when a message is received
00138 
00139         @param *args: List[Callable[[sok.AddressInfo, Message], bytes]] List of callable objects
      containing its Method type and respective callback function
00140         """
00141         for x in args:
00142             self.recv_functions.append(x)
00143 
00144     @staticmethod
00145     def parse_packet(data : bytes) -> Message:
00146         """!
00147         Deserializes incoming byte received by the TCP socket
00148 
00149         @param data: bytes  Byte object received by socket
00150 
00151         @return Message: object
00152         """
00153         return Util.deserialize(data, MessageType.REQUEST)
```

```
00154
00155
00156
00157
```

9.21 tcp_client.py File Reference

Main TCP client application which contains the implementation of the client-end of this FTP service.

Namespaces

- namespace tcp_client

Functions

- bytes tcp_client.on_send_put (List[str] inp, MethodType type)
- bytes tcp_client.on_send_get (List[str] inp, MethodType type)
- bytes tcp_client.on_send_change (List[str] inp, MethodType type)
- bytes tcp_client.on_send_help (List[str] inp, MethodType type)
- def tcp_client.on_response_put_change (Message message)
- def tcp_client.on_response_get (Message message)

  Checks if file already exists.
- def tcp_client.on_response_help (Message message)
- def tcp_client.on_response_unknown (Message message)
- def tcp_client.on_response_not_found (Message message)
- def tcp_client.on_response_no_change (Message mesage)

Variables

- string tcp_client.client_dir = "client"
- string tcp_client.BASE_DIR = os.getcwd() + os.sep + "dir" + os.sep + client_dir + os.sep
- string tcp_client.arg_helper
- tcp_client.cmd = arguments.ParserArgs(helper = arg_helper, version="tcp_server version 1.0")
- tcp_client.params = cmd.parameters(["-a", "-p", "-f", "-F", "-d"])
- tcp_client.client = TcpClient("127.0.0.1", ∗∗params)

9.21.1 Detailed Description

Main TCP client application which contains the implementation of the client-end of this FTP service.

All protocols and file handling is defined in this file.

9.21.2 Author(s)

- Created by Kevin de Oliveira on 04/01/2022.
- Student ID: 40054907

Definition in file tcp_client.py.

9.22 tcp_client.py

Go to the documentation of this file.

```
00001 #!/usr/bin/python3.8
00002
00003
00014
00015 import os
00016 from ftp.cmd import arguments
00017
00018 from typing import List
00019 import bitarray
00020 from bitarray import util
00021 from ftp.parser.message import Message, Util
00022
00023 from ftp.parser.packet import packet
00024
00025 from ftp.parser.message_type import MessageType, MethodType, RequestType, ResponseType
00026 from ftp.tcp.client import TcpClient
00027
00028 client_dir = "client"
00029
00030 BASE_DIR = os.getcwd() + os.sep + "dir" + os.sep + client_dir + os.sep
00031
00032
00033
00034
00035 def on_send_put(inp : List[str], type: MethodType) -> bytes:
00036     message = Message(3, type)
00037
00038     try:
00039         with open(BASE_DIR + inp[1], "r") as f:
00040             payload = f.read()
00041             size = os.path.getsize(BASE_DIR + inp[1])
00042     except IndexError:
00043         # @brief if exception occurs during parsing of input, return as if the command syntax as
      invalid
00044         raise ValueError("invalid command")
00045     except Exception:
00046         raise ValueError("cannot send file")
00047
00048
00049     file_data = Util.str2bit(inp[1], message.data[1].size, with_count=True)
00050     file_size = bitarray.util.int2ba(size, length=32, endian='big').to01()
00051
00052     message.parse(
00053     file_data + file_size
00054     )
00055
00056     message.add_payload(payload)
00057
00058
00059     return Util.serialize(message)
00060
00061
00062 def on_send_get(inp : List[str], type: MethodType) -> bytes:
00063     try:
00064         message = Message(3, type)
00065         file_data = Util.str2bit(inp[1], message.data[1].size, with_count=True)
00066         message.parse(file_data)
00067         return Util.serialize(message)
00068     except IndexError:
00069         # @brief if exception occurs during parsing of input, return as if the command syntax as
      invalid
00070         raise ValueError("invalid command")
00071
00072
00073 def on_send_change(inp : List[str], type: MethodType) -> bytes:
00074     message = Message(3, type)
00075     file_data_old = Util.str2bit(inp[1], message.data[1].size, with_count=True)
00076     file_data_new = Util.str2bit(inp[2], message.data[3].size, with_count=True, size_count=8)
00077     message.parse(
00078     file_data_old + file_data_new
00079     )
00080
00081     return Util.serialize(message)
00082
00083 def on_send_help(inp : List[str], type: MethodType) -> bytes:
00084     message = Message(3, type)
00085
00086     message.parse("00000")
00087
00088     return Util.serialize(message)
00089
00090
```

```
00091
00092 def on_response_put_change(message : Message) :
00093     return
00094
00095 def on_response_get(message: Message) :
00096     val = Util.bit2byte(message)
00097
00098     file_name = val[1].decode("utf-8").replace(chr(0), "")
00099
00100     with os.scandir(BASE_DIR) as dir:
00101         flag : bool = False
00102         for x in dir:
00103             if x.name == file_name:
00104                 flag = True
00105
00106
00109         if flag:
00110             ch : str = ”
00111             while ch not in ["y", "n"]:
00112                 ch = input(file_name + " already exists. Do you want to overwrite? (y/n) ")
00113     try:
00114         if ch == "y":
00115             with open(BASE_DIR + file_name, "w") as f:
00116                 f.write(val[-1].decode("utf-8").replace(chr(0), ""))
00117
00118     except Exception as e:
00119         print(e)
00120
00121 def on_response_help(message: Message):
00122     val = Util.bit2byte(message)
00123     print(val[1].decode("utf-8").replace(chr(0), ""))
00124
00125 def on_response_unknown(message: Message):
00126     print("unknown command")
00127
00128 def on_response_not_found(message: Message):
00129     print("file not found")
00130
00131 def on_response_no_change(mesage : Message):
00132     print("operation failed")
00133
00134
00135
00136 arg_helper = """usage: tcp_client [-a address] [-p port] [-f base_folder] [-F absolute_folder] [-v |
       --version] [-h | --help | -?]
00137
00138 This are the commands used:
00139 \t-d\t\t Activate debug mode
00140 \t-a address\t\t Set address of this client (default: 127.0.0.1)
00141 \t-p port\t\t\t Set port number of this client (default: 1025)
00142 \t-f base_folder\t\t Set relative base path of the FTP client (default: /dir/client)
00143 \t-F absolute_folder\t Set absolute base path of the FTP client (default: $pwd)"""
00144
00145
00146
00147 if __name__ == "__main__":
00148     cmd = arguments.ParserArgs(helper = arg_helper, version="tcp_server version 1.0")
00149
00150     cmd.get_args()
00151
00152     params = cmd.parameters(["-a", "-p", "-f", "-F", "-d"])
00153
00154     if "-f" in params:
00155         client_dir = params["-f"]
00156     elif "-F" in params:
00157         BASE_DIR = params["-F"]
00158
00159
00160
00161     client = TcpClient("127.0.0.1", **params)
00162
00163     client.on_send(
00164     (RequestType.PUT, on_send_put),
00165     (RequestType.GET, on_send_get),
00166     (RequestType.CHANGE, on_send_change),
00167     (RequestType.HELP, on_send_help),
00168     )
00169
00170     client.on_response(
00171     (ResponseType.OK_PUT_CHANGE, on_response_put_change),
00172     (ResponseType.OK_GET, on_response_get),
00173     (ResponseType.HELP, on_response_help),
00174     (ResponseType.ERROR_UNKNOWN, on_response_unknown),
00175     (ResponseType.ERROR_NOT_FOUND, on_response_not_found),
00176     (ResponseType.ERROR_NO_CHANGE, on_response_no_change),
00177     )
00178     client.connect()
```

9.23 tcp_server.py File Reference

Main TCP server application which contains the implementation of the server-end of this FTP service.

Namespaces

- namespace tcp_server

Functions

- bytes tcp_server.response_ok ()
- bytes tcp_server.response_get (str file_name, int size, str payload)
- def tcp_server.response_error_not_found ()
- def tcp_server.response_error_no_change ()
- def tcp_server.response_ok_help ()
- bytes tcp_server.on_receive_put (addr, Message data)
- bytes tcp_server.on_receive_get (addr, Message data)
- bytes tcp_server.on_receive_change (addr, Message data)
- bytes tcp_server.on_receive_help (addr, Message data)

Variables

- string tcp_server.server_dir = "server"
- string tcp_server.BASE_DIR = os.getcwd() + os.sep + "dir" + os.sep + server_dir + os.sep
- string tcp_server.arg_helper
- tcp_server.cmd = arguments.ParserArgs(helper = arg_helper, version="tcp_server version 1.0")
- tcp_server.params = cmd.parameters(["-a", "-p", "-f", "-F", "-d"])
- tcp_server.tcp_server = TcpServer("127.0.0.1", ∗∗params)

9.23.1 Detailed Description

Main TCP server application which contains the implementation of the server-end of this FTP service.

All protocols and file handling is defined in this file.

9.23.2 Author(s)

- Created by Kevin de Oliveira on 04/01/2022.
- Student ID: 40054907

Definition in file tcp_server.py.

9.24 tcp_server.py

Go to the documentation of this file.

```
00001 #!/usr/bin/python3.8
00002
00003
00014
00015
00016 import os
00017 from re import S
00018 from socket import socket
00019
00020 import bitarray
00021 from ftp.cmd import arguments
00022 from ftp.parser.message_type import MessageType, RequestType, ResponseType
00023 from ftp.parser.message import Message, Util
00024 from ftp.tcp.server import TcpServer
00025
00026
00027 server_dir = "server"
00028
00029
00030 BASE_DIR = os.getcwd() + os.sep + "dir" + os.sep + server_dir + os.sep
00031
00032
00033 def response_ok() -> bytes:
00034     message = Message(3, ResponseType.OK_PUT_CHANGE)
00035     message.parse("00000")
00036     return Util.serialize(message)
00037
00038 def response_get(file_name : str, size : int, payload: str) -> bytes:
00039     message = Message(3, ResponseType.OK_GET)
00040     file_data = Util.str2bit(file_name, ResponseType.OK_GET.get_format()[1], with_count=True)
00041     file_size = bitarray.util.int2ba(size, length=32, endian="big").to01()
00042     message.parse(
00043     file_data + file_size
00044     )
00045
00046     message.add_payload(payload)
00047     return Util.serialize(message)
00048
00049 def response_error_not_found():
00050     message = Message(3, ResponseType.ERROR_NOT_FOUND)
00051     message.parse("00000")
00052     return Util.serialize(message)
00053
00054 def response_error_no_change():
00055     message = Message(3, ResponseType.ERROR_NO_CHANGE)
00056     message.parse("00000")
00057     return Util.serialize(message)
00058
00059 def response_ok_help():
00060     message = Message(3, ResponseType.HELP)
00061     val = Util.str2bit("get put change bye",ResponseType.HELP.get_format()[1])
00062     message.parse(val)
00063     return Util.serialize(message)
00064
00065
00066
00067 def on_receive_put(addr, data : Message) -> bytes:
00068
00069     result = Util.bit2byte(data)
00070
00071     file_name = result[1].decode("utf-8").replace(chr(0), "")
00072
00073     try:
00074         with open(BASE_DIR + file_name, "w") as f:
00075             f.write(result[-1].decode("utf-8").replace(chr(0), ""))
00076
00077     except Exception:
00078         # @brief Although no response type is defined for PUT errors in the project description,
      excpetions may still occur if server has not enough privilege for accessing file or f.write fails.
00079         # Sending an ERROR_NO_CHANGE response in case the above issue happends
00080         return response_error_no_change()
00081
00082     return response_ok()
00083
00084
00085 def on_receive_get(addr, data : Message) -> bytes:
00086
00087     result = Util.bit2byte(data)
00088
00089     file_name = BASE_DIR + result[1].decode("utf-8")
00090
00091     # @brief Remove embedded null pointer coming from raw data
```

```
00092     file_name = file_name.replace(chr(0), "")
00093
00094
00095
00096     try:
00097         with open(file_name, "r") as open_file:
00098             f = open_file.read()
00099             size = os.path.getsize(file_name)
00100
00101         return response_get(result[1].decode("utf-8").replace(chr(0), ""), size, f)
00102
00103     except Exception as e:
00104         return response_error_not_found()
00105
00106
00107
00108 def on_receive_change(addr, data : Message) -> bytes:
00109     result = Util.bit2byte(data)
00110
00111     file_name_old = BASE_DIR + result[1].decode("utf-8").replace(chr(0), '')
00112     file_name_new = BASE_DIR + result[3].decode("utf-8").replace(chr(0), '')
00113
00114
00115
00116     try:
00117         os.rename(file_name_old, file_name_new)
00118     except Exception as e:
00119         return response_error_no_change()
00120
00121
00122     return response_ok()
00123
00124 def on_receive_help(addr, data : Message) -> bytes:
00125     return response_ok_help()
00126
00127
00128
00129
00130 arg_helper = """usage: tcp_server [-a address] [-p port] [-f base_folder] [-F absolute_folder] [-v |
       --version] [-h | --help | -?]
00131
00132 This are the commands used:
00133 \t-d\t\t Activate debug mode
00134 \t-a address\t\t Set address of this server (default: 127.0.0.1)
00135 \t-p port\t\t\t Set port number of this server (default: 1025)
00136 \t-f base_folder\t\t Set relative base path of the FTP server (default: /dir/server)
00137 \t-F absolute_folder\t Set absolute base path of the FTP server (default: $pwd)"""
00138
00139 if __name__ == "__main__":
00140
00141     cmd = arguments.ParserArgs(helper = arg_helper, version="tcp_server version 1.0")
00142
00143     cmd.get_args()
00144
00145     params = cmd.parameters(["-a", "-p", "-f", "-F", "-d"])
00146
00147     if "-f" in params:
00148         server_dir = params["-f"]
00149     elif "-F" in params:
00150         BASE_DIR = params["-F"]
00151
00152     tcp_server = TcpServer("127.0.0.1", **params)
00153
00154
00155
00156     tcp_server.on_receive(
00157     (RequestType.PUT , on_receive_put),
00158     (RequestType.GET, on_receive_get),
00159     (RequestType.CHANGE, on_receive_change),
00160     (RequestType.HELP, on_receive_help),
00161     )
00162
00163     tcp_server.listen()
```

Index

__call__
    ftp.parser.packet.packet, 30
__init__
    ftp.cmd.arguments.ParserArgs, 33
    ftp.parser.message.Message, 23
    ftp.parser.packet.packet, 29
    ftp.tcp.client.TcpClient, 41
    ftp.tcp.server.TcpServer, 45
__repr__
    ftp.parser.message.Message, 24
    ftp.parser.packet.packet, 30
__str__
    ftp.parser.message.Message, 24
    ftp.parser.packet.packet, 30

add_payload
    ftp.parser.message.Message, 24
arg_helper
    tcp_client, 17
    tcp_server, 21
argn
    ftp.cmd.arguments.ParserArgs, 34
argv
    ftp.cmd.arguments.ParserArgs, 34

BASE_DIR
    tcp_client, 18
    tcp_server, 21
bit2byte
    ftp.parser.message.Util, 48
body
    ftp.parser.packet.packet, 32

CHANGE
    ftp.parser.message_type.RequestType, 37
check_response
    ftp.tcp.client.TcpClient, 42
cin
    ftp.tcp.client.TcpClient, 42
client
    tcp_client, 18
client_dir
    tcp_client, 18
cmd
    tcp_client, 18
    tcp_server, 21
connect
    ftp.tcp.client.TcpClient, 42

default_helper
    ftp.cmd.arguments.ParserArgs, 34
default_version
    ftp.cmd.arguments.ParserArgs, 34
deserialize
    ftp.parser.message.Util, 49

ERROR_NO_CHANGE
    ftp.parser.message_type.ResponseType, 39
ERROR_NOT_FOUND
    ftp.parser.message_type.ResponseType, 39
ERROR_UNKNOWN
    ftp.parser.message_type.ResponseType, 40

ftp, 13
ftp.cmd, 13
ftp.cmd.arguments, 13
ftp.cmd.arguments.ParserArgs, 32
    __init__, 33
    argn, 34
    argv, 34
    default_helper, 34
    default_version, 34
    get_args, 33
    parameters, 33
ftp.parser, 13
ftp.parser.message, 14
ftp.parser.message.Message, 23
    __init__, 23
    __repr__, 24
    __str__, 24
    add_payload, 24
    has_payload, 25
    header, 25
    parse, 25
    payload, 25
    size, 26
ftp.parser.message.Util, 48
    bit2byte, 48
    deserialize, 49
    serialize, 49
    str2bit, 50
ftp.parser.message_type, 14
ftp.parser.message_type.MessageType, 26
    REQUEST, 27
    RESPONSE, 27
ftp.parser.message_type.MethodType, 28
    get_format, 29
ftp.parser.message_type.RequestType, 35
    CHANGE, 37
    GET, 37

get_format, 36
HELP, 37
PUT, 37
ftp.parser.message_type.ResponseType, 38
ERROR_NO_CHANGE, 39
ERROR_NOT_FOUND, 39
ERROR_UNKNOWN, 40
get_format, 39
HELP, 40
OK_GET, 40
OK_PUT_CHANGE, 40
ftp.parser.packet, 14
ftp.parser.packet.packet, 29
__call__, 30
__init__, 29
__repr__, 30
__str__, 30
body, 32
set_size, 31
size, 32
to_bytes, 31
value, 31
ftp.tcp, 14
ftp.tcp.client, 15
ftp.tcp.client.TcpClient, 41
__init__, 41
check_response, 42
cin, 42
connect, 42
handle_connection, 43
handler, 43
ip_address, 44
on_response, 43
on_send, 44
socket, 44
thread, 44
ftp.tcp.server, 15
ftp.tcp.server.TcpServer, 45
__init__, 45
handle_listen, 46
ip_address, 47
is_connected, 47
listen, 46
on_receive, 46
parse_packet, 47
socket, 47
thread, 48
ftp/__init__.py, 53
ftp/cmd/__init__.py, 53
ftp/cmd/arguments.py, 51, 52
ftp/parser/__init__.py, 53
ftp/parser/message.py, 54, 55
ftp/parser/message_type.py, 58, 59
ftp/parser/packet.py, 60
ftp/tcp/__init__.py, 54
ftp/tcp/client.py, 61, 62
ftp/tcp/server.py, 64, 65

GET
ftp.parser.message_type.RequestType, 37
get_args
ftp.cmd.arguments.ParserArgs, 33
get_format
ftp.parser.message_type.MethodType, 29
ftp.parser.message_type.RequestType, 36
ftp.parser.message_type.ResponseType, 39

handle_connection
ftp.tcp.client.TcpClient, 43
handle_listen
ftp.tcp.server.TcpServer, 46
handler
ftp.tcp.client.TcpClient, 43
has_payload
ftp.parser.message.Message, 25
header
ftp.parser.message.Message, 25
HELP
ftp.parser.message_type.RequestType, 37
ftp.parser.message_type.ResponseType, 40

ip_address
ftp.tcp.client.TcpClient, 44
ftp.tcp.server.TcpServer, 47
is_connected
ftp.tcp.server.TcpServer, 47

listen
ftp.tcp.server.TcpServer, 46

OK_GET
ftp.parser.message_type.ResponseType, 40
OK_PUT_CHANGE
ftp.parser.message_type.ResponseType, 40
on_receive
ftp.tcp.server.TcpServer, 46
on_receive_change
tcp_server, 19
on_receive_get
tcp_server, 19
on_receive_help
tcp_server, 20
on_receive_put
tcp_server, 20
on_response
ftp.tcp.client.TcpClient, 43
on_response_get
tcp_client, 15
on_response_help
tcp_client, 16
on_response_no_change
tcp_client, 16
on_response_not_found
tcp_client, 16
on_response_put_change
tcp_client, 16
on_response_unknown
tcp_client, 16

on_send
ftp.tcp.client.TcpClient, 44
on_send_change
tcp_client, 17
on_send_get
tcp_client, 17
on_send_help
tcp_client, 17
on_send_put
tcp_client, 17

parameters
ftp.cmd.arguments.ParserArgs, 33
params
tcp_client, 18
tcp_server, 22
parse
ftp.parser.message.Message, 25
parse_packet
ftp.tcp.server.TcpServer, 47
payload
ftp.parser.message.Message, 25
PUT
ftp.parser.message_type.RequestType, 37

REQUEST
ftp.parser.message_type.MessageType, 27
RESPONSE
ftp.parser.message_type.MessageType, 27
response_error_no_change
tcp_server, 20
response_error_not_found
tcp_server, 20
response_get
tcp_server, 20
response_ok
tcp_server, 21
response_ok_help
tcp_server, 21

serialize
ftp.parser.message.Util, 49
server_dir
tcp_server, 22
set_size
ftp.parser.packet.packet, 31
size
ftp.parser.message.Message, 26
ftp.parser.packet.packet, 32
socket
ftp.tcp.client.TcpClient, 44
ftp.tcp.server.TcpServer, 47
str2bit
ftp.parser.message.Util, 50

tcp_client, 15
arg_helper, 17
BASE_DIR, 18
client, 18
client_dir, 18
cmd, 18
on_response_get, 15
on_response_help, 16
on_response_no_change, 16
on_response_not_found, 16
on_response_put_change, 16
on_response_unknown, 16
on_send_change, 17
on_send_get, 17
on_send_help, 17
on_send_put, 17
params, 18
tcp_client.py, 67
tcp_server, 19
arg_helper, 21
BASE_DIR, 21
cmd, 21
on_receive_change, 19
on_receive_get, 19
on_receive_help, 20
on_receive_put, 20
params, 22
response_error_no_change, 20
response_error_not_found, 20
response_get, 20
response_ok, 21
response_ok_help, 21
server_dir, 22
tcp_server, 22
tcp_server.py, 70
thread
ftp.tcp.client.TcpClient, 44
ftp.tcp.server.TcpServer, 48
to_bytes
ftp.parser.packet.packet, 31

value
ftp.parser.packet.packet, 31