

# TFTP Iteration Four

## Setup Instructions

Set your eclipse workspace to the top level, which is the project root directory. Eclipse should automatically recognize and import the project.

If that did not work, follow these steps: Ensure the Java Perspective is open in Eclipse.

Create the Core project:

- In the package explorer in eclipse, right click and select `New` -> `Java Project`
- In the Project name text box, type in `core`
- Click Finish This project will now be recognized and imported. Repeat this process with the project names: `client`, `server`, and `errorSimulator`.

The client, server, and errorSimulator projects will now show errors. The `core` project is required to be on the buildpath of the other three projects. To add the `core` project to the buildpath of `client`:

- Right click on the `client` project
- Click `properties`
- Click `java build path`
- Click the `projects` tab
- Click the `add` button
- Select the checkbox next to the `core` project
- Click OK Repeat this process for the `server` project and the `errorSimulator` project.

Eclipse should now be configured properly.

## Running Instructions

**Note** All transfers take place from the present working directory of the respective project. For example a write operation from the client will write from `project-root/client/`

There is a change directory command `cd` which allows you to move the present working directory to a different folder. For example `cd ../` will change the directory to the directory above the current project. All directory changes are relative to the original project directory. To return to the project directory, simply type `cd .`

## Server

- To run the server, run the server project from eclipse.

- To shutdown the server, type `shutdown` in the servers command line interface.
- The working directory of the server can be changed with `cd` . All read and writes will be relative to the new folder

## Error Simulator

- To run the Error Simulator, run the error simulator from eclipse.
- To shutdown the Error Simulator, type `shutdown` in the error simulators command line interface.

## Simulation Commands

TFTP Error Simulator  
 <type> must be either 'ack', 'data', or 'req'  
 <packetNum> starts counting at 1 (The first data packet is data 1).  
 Each simulation runs for a single transfer. The mode reset to norm after each transfer  
 The server and client timeout is 2400ms

### Commands:

help		Prints this message
shutdown		Exits the simulator
norm		Forward packets through without alteration
rend		Removes the end byte of the next request packet. ie Removes the 0 Byte after Mode
rrs		Removes the Request Seperator of the next request packet. ie Removes 0 Byte after Filename
mode	<m>	Changes the mode of the next request packet
csa	<t> <n>	Changes the sender TID of a specified packet
op	<t> <n> <o>	Changes the opcode of a specified packet
cl	<t> <n> <o>	Changes the length of a specified packet
delay	<t> <n> <o>	Delays the specified packet by a number of timeouts
duplicate	<t> <n>	Sends a duplicate of the specified packet
drop	<t> <n>	Drops the specified packet

## Notes

- If the type parameter is equal to `req` then the packet number must be `1` for desired functionality

## Client

By default, client will connect directly to the server when run through eclipse. To have the client run through the error simulator, run the client with the `-t` command line argument.

- To see the usage information, type `help` in the cli.
- To perform a read operation from the server to the client, type `read` followed by a space and the filename.
- To perform a write operation from the client to the server, type `write` followed by a space and the filename.
- The working directory of the client can be changed with `cd`. All read and writes will be relative to the new folder

## Command Line Arguments

To enter a command line argument in eclipse:

- Right click on the client project and select `Run as -> Run Configurations`
- Ensure that the client project is selected in the tree view in the left hand side of the popup window
- Select the `Arguments` tab on the right hand side
- In the `Program Arguments` text box, add `-t`

## Error Codes

### #1 File Not Found

#### Read Request

During a read request, the client will send a request to read a file. If the file does not exist then the server will spawn a special `Error Responder` thread which will respond with an error code 1. This indicates to the client that the file does not exist to on the server.

#### Write Request

Before sending a request to write a file to the server, the client performs a check to find the desired file. If the file cannot be found then the client immediately notifies the user and never sends a request.

### #2 Access Violation

#### Read Request

When a file request is made to the server for a file that cannot be read then an `Error`

**Responder** will immediately respond with an error code 2. Alternatively, if the client does not have permission to write to its destination folder then the read request will never be sent.

### **Write Request**

If a client does not have permission to access a local file then a request will never be sent and the user will be notified immediately. If the client sends a write request and the server cannot write to its destination folder then it will respond with an error code 2.

## **#3 Disk Full or Allocation Exceeded**

### **Read Request**

If a client runs out of disk space during a transfer then it will terminate the transfer and send an error code 3 to the server.

### **Write Request**

If a server runs out of disk space during a transfer then it will terminate the transfer and send an error code 3 to its clients

## **#4 Illegal TFTP Operation**

If an endpoint receives a TFTP message that contains an invalid opcode then it will respond with an error code 4 packet.

## **#5 Unknown Transfer ID**

If an endpoint receives a packet from a sender that differs from the one that is currently performing transfer then it will respond with an error code 5 to the unknown sender and continue the transfer.

## **#6 File Already Exists**

### **Read Request**

If a request is made to read a file from a server that already exists on a client then the client will immediately notify the user and terminate without sending a request.

### **Write Request**

If a server receives a request to write a file that already exists, it will respond with an error code 6 and never start the transfer.

## **Project Structure**

The file TeamResponsibilities.txt describes the responsibilities of each team member for this iteration. The document is split into a different section for each member, and lists their contributions.

The file TestPlan.pdf describes the test procedure followed to ensure correct functionality of the program. It also describes the command line arguments and cli commands available for the client, server, and error simulator. The javadoc for each project is located in that projects doc folder.

There is a top level folder called ucms where the ucm diagrams are located.

The source code for this deliverable is split up into four main projects: core, client, errorSimulator, and server.

## **Core**

The core project contains all of the common core functionality shared between projects. The core project contains all code relating to the TFTP standard. This project is a dependency for all other projects.

## **Client**

The client project contains the code specific to the client application.

## **errorSimulator**

The errorSimulator project contains the code specific to the Error Simulator application.

## **Server**

The server project contains the code specific to the server application.

# **UCM Diagrams**

## **Request**

This diagram demonstrates the flow of a request from the client to the server and its response. This is generic, as the same logic is used for a read and a write request.

## **Read Transfer**

This shows the steady state transfer when reading from a file on the server to the client.

## **Write Transfer**

This shows the steady state transfer when writing to a file on the server from the client.

# Source Code Structure

## Client

The client project contains only the client class. This class is responsible for parsing command line arguments, starting the clients command line interface, and starting the client. This contains the main method for the client application.

## Error Simulator

The error simulator project is responsible for relaying datagram packets between the client and the server. The main method for the error simulator application is contained in the `ErrorSimulator.java` class.

## sim

This package holds the error simulation controller and command line interface for the error simulator. It handles all sending and receiving of request packets

## stream

This package has all of the simulation streams that decorate a basic stream to modify a transfer.

## threads

This package has simulation threads which run an individual simulation for a request

## Server

The server project contains only the server class. This class is responsible for starting the servers command line interface, and starting the server. This class contains the main method for the server application

## Core

The core project contains functionality needed by the client, server, and errorSimulator projects. It has several packages: `core.cli`, `core.ctrl`, `core.log`, `core.net`, `core.req`, and `core.util`

## core.cli

This package contains the generic implementation of the command line interface.

## core.ctrl

This package contains the parent classes of the client, server, and errorSimulator. Its purpose is to abstract client and server logic to allow it to be reused by the error simulator.

## **core.log**

This package is responsible for setting up a logger that allows a global logging severity level to be defined.

## **core.net**

This package contains the core file transfer logic and network operations. This includes actions such as writing to sockets, reading from sockets, etc.

## **core.req**

This package contains the logic for encoding and decoding TFTP protocol messages.

## **core.util**

This package contains two utility classes: ByteUtils.java and Worker.java

### **ByteUtils.java**

This class contains static methods to find the index of a value in a byte array, and to convert a byte array to a String.

### **Worker.java**

This is an abstract base class for long running asynchronous jobs.

For further details on any specific class from a core package, refer to the provided javadoc.

## **Known Issues**

Due to the error simulator only running one thread per request, a duplicate request is not throwing an error 5. This is due to the fact that the error simulator and client communicate on a single port and therefore it never has a mismatched transfer id. Instead the second response is just ignored by the client.

**Note** this was an issue discovered in the last iteration but due to the lack of time caused by the late feedback it was not fixed for this iteration.