HTCPCP implementation

Pranav Gade	LCI2020010
Nehal Sharma	LCS2020001
Priyanshu Upadhyay	LIT2020011
Manasvi Agrawal	LIT2020029

Table of contents

1 Introduction

We have implemented a client and server conforming to `RFC2324 HTCPCP/1.0 (Hyper Text Coffee Pot Control Protocol)`.

The RFC was originally intended to be an April Fools' joke, but Error 418 (I'm a teapot) has gained popularity as a classic easter egg in developer circles, so we decided to implement a server and a client conforming to this protocol to communicate. The server will have several classes(pot, coffee pot, tea pot) modeling real-world entities. And the client is used to send commands (start/stop for brewing the coffee) to the server using HTCPCP/1.0. Depending on the server state(coffee pot/tea pot), the server will respond accordingly to the client (success/error, etc).

1.1 Documentation

You can find detailed doxygen generated documentation here: `htcpcp-implementation-docs.netlify.app`

1.2 Building and Usage

1. Use cmake to build.
2. `docker build --tag htcpcp .` to build the docker image
3. `kubectl apply -f k8s.yml` to deploy to your kubernetes cluster
4. Alternatively, run the built binary `htcpcp` locally with: `./htcpcp 8080 coffeepot`
5. To start brewing coffee:`curl -i -X POST -H "Accept-Additions: milk-type=Cream; syrup-type=Almond; alcohol-type=Whisky; milk-type=Skim;" --data $'start\r\n' localhost:8080`
6. To stop brewing and get your coffee:`curl -i -X POST -H "Accept-Additions: milk-type=Cream; syrup-type=Almond; alcohol-type=Whisky; milk-type=Skim;" --data $'stop\r\n' localhost:8080`

1.3 Links to Wiki Report:

1. `Introduction`
2. `Goal`
3. `Tech-Stack/Tooling`
4. Project Structure
    - `Pot`

2 Goal

Our goals for this project are as follows

- Implementing RFC 2324
- Creating an HTTP1.1 (RFC 2616) compatible web server.
- Creating various classes modeling multiple types of pots.

3 Tech Stack/Tooling

All the major technologies used are listed below, and detailed documents elaborating on our use cases can be found on further pages in the documentation.

- C++/C - for writing the application
- CMake - as a build system
- CTest - as the testing framework
- Docker - to containerize our application for kubernetes
- Kubernetes - for container orchestration and load balancing
- MermaidJS - to generate class and stare diagram
- Doxygen - for generating documentation
- git - for version control

4 Pot

The most important entity in this system is the pot, which receives commands and accordingly brews coffee as per the user's orders

---

It accepts requests to `start` brewing coffee, take up a cup of plain coffee and add to it milk, sweetener, syrup, spice and/or alcohol if possible (allowed in coffee pot but not in tea pot) and as requested by the client, else signal inability of the pot to brew coffee, and keep brewing the cup of coffee until receipt of command to `stop` brewing, and then remove the ready cup of coffee from itself.
A state variable `brewing` keeps track of if the Pot server is currently occupied in brewing a cup of coffee, and accordingly responds to `start` / `stop` requests. If the request method header is anything other than `POST` or `BREW` it responds with the appropriate error. The Pot class defines a `brew` method that takes in a `request` object and returns the appropriate `response` object.
If not yet `brewing`, on receiving the `start` command, the pot instantiates a `Cup` object, switches to `brewing` state, and adds all the requested additions to it one by one. If all the additions are successful, it responds appropriately saying that the coffee is being brewed. If `brewing`, on receiving the `stop` command, switches of the brewing state, detaches the cup, gets a suitable description of this brewed cup of coffee, and responds accordingly with the description as the response body.

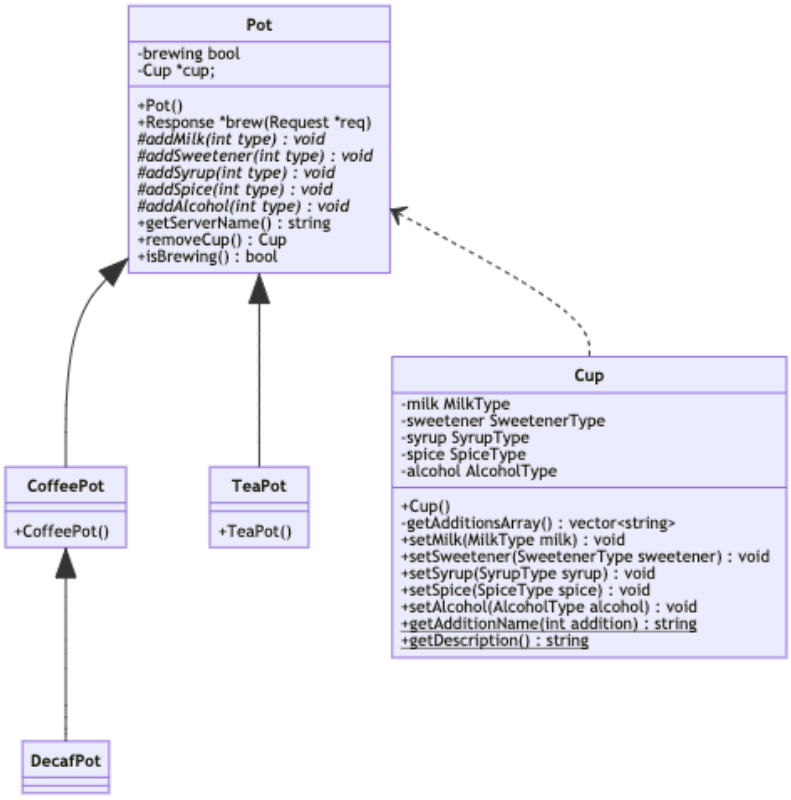


Figure 1

5 Derived Pots

The 2 most important types of pots to be implemented are a coffee pot and a tea pot

5.1 Coffee Pot:

It should accept requests to start brewing coffee, take up a cup of plain coffee and add to it milk, sweetener, syrup, spice, alcohol as requested by the client and keep brewing the cup of coffee until receipt of a command to stop brewing, and then remove the ready cup of coffee from itself.

The `CoffeePot` class extends the `Pot` class by inheriting it, hence provides the same functionality and more. It sets the server name to "Coffee Pot", according to which the brew method sets appropriate response headers (`Server: Coffee Pot`). The `DecafPot` derives from this class

5.2 Tea Pot:

It should respond to every POST or BREW method request with Error 418 I'm a Teapot (to signal that coffee can not be brewed by a tea pot)
The `TeaPot` class extends and overrides parts of the `Pot` class by inheriting it, hence provides the same functions (names) but different functionality. It sets the server name to "Tea Pot", according to which the brew method sets appropriate response headers (`Server: Tea Pot`). It overrides all the `add[Addition]()` methods to disallow addition and instead throws an error with integer code 418. This error is caught in the brew method and accordingly, a response object is returned

6 Cup Additions

The Coffee brewed by the Pot is contained in a Cup where all the additions of special ingredients to it are made

---

It should allow addition of a variety of milk, sweetener, syrup, spice and / or alcohol to the coffee inside it, and it should be possible to describe the cup of coffee by it's additions
The `Cup` class has private fields for milk, sweetener, syrup, spice, and alcohol, and public setters for each of these additions. A public method `getDescription()` provides a string with a well formatted list of all the additions in this cup of coffee. Internally it uses a private method `getAdditionsArray()` which returns an array of the string equivalents of each addition, using the utility static function `getAdditionName()` which returns the string version of each addition enum (explained below)

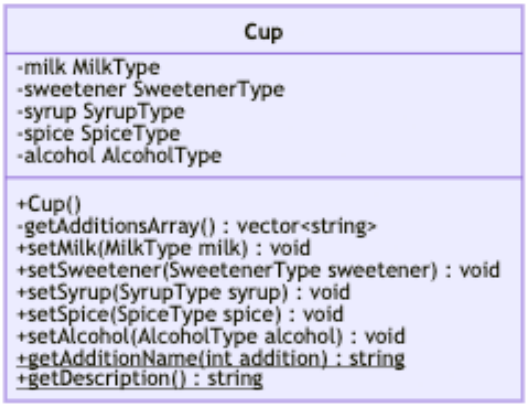


Figure 2

Additions (MilkType, SweetenerType, SyrupType, SpiceType, AlcoholType) Each addition type should allow only a certain set of values
Each addition is an enum class defined with a set of values according to the specification. Each addition type and it's permitted values are as follows:

- Milk: Cream, Half and Half, Whole Milk, Part Skim, Skim, Non Dairy
- Sweetener: Sugar, Stevia, Honey, Maple Syrup, Agave
- Syrup: Vanilla, Almond, Raspberry, Chocolate
- Spice: Cinnamon, Nutmeg, Clove, Cardamom
- Alcohol: Whisky, Rum, Kahlua, Aquavit

7 Request/Response

1. **Request**

We start with listening for incoming requests and then pass them onto the request constructor. The request constructor will parse input from the socket and parse the text as per the HTTP Request `RFC 2616`/HTCPCP `RFC 2324`. This gives us the headers in a `header map`(Host, User, Accept, Accept-Language, Content-Type, and Accept-Additions) and additions in `additions map`(milk-type,syrup-type,alcohol-type,sweetener-type, and spice-type) and the body. All of these are stored as a state variable in the request object.

1. **Response**

It should accept a response_code integer and get a response string accordingly. Additionally, it will set headers and send them to the client using sockets.
On creating a new Response object, either of the two constructors is called depending on the parameters. Inside the constructor, `getResponseString()` is called to set the response string. The `getResponseString()` function works in a manner such that it returns a Response string from the response code (For instance, "I'm a teapot" from error code 418). Finally, we send the Response using our `sendresponse()` function, which is then used in the main file.

8 Networking

The `rfc2324` protocol describes a web server, so we developed a simple interface to underlying UNIX sockets to make reading and writing easier. Inspired by Java networking, I have wrapped the bind, listen, and accept calls with a ServerSocket class. This class allows you to listen for incoming connections, and returns a Socket object when a connection is received. This Socket object extends istream and ostream, and is a wrapper around the underlying read and write calls. It is especially important for buffering reading and writing to the socket.

9 Docker/K8s

After the webserver is written, we need to host it in some way. We decided to use kubernetes for two reasons:

1. Its declarative interface makes scaling, adding, and removing instances very easy. This especially makes sense in this case, as we might want to add, remove, or replace pots depending on dynamic requirements.
2. NGINX ingress controller manages routing and security very well. So, we do not have to worry about malformed requests, overloading, and path verification.

While satisfying this requirement, we also created a docker container. To keep it's size down, we used the `scratch` base image. Since this image contains practically nothing, not even standard libraries like libc, we are creating a static linked binary.

10 Processing

The flow of the program

- We start by receiving a socket connection
- A request object is created from the socket
- The request object parses all the required data into different strings and arrays
- The request object is then passed onto the brew method of the pot
- Then the brew method changes the internal state of the pot to start brewing
- It stops brewing when given the stop command
- A `Cup` object is returned
- We then get the cup's description, and send it back as a HTTP/1.1 response

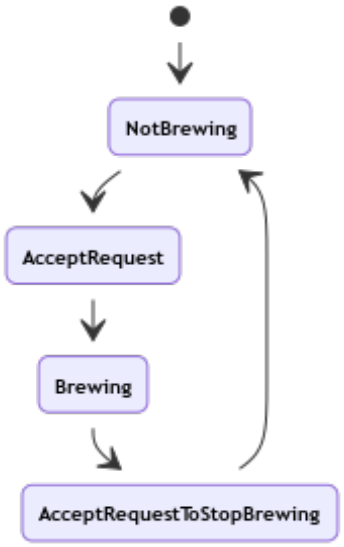


Figure 3 State diagram of the system

Figure 4

11 Results

```
p@crimson htcpcp]$ curl -i -X POST -H "Accept-Additions: milk-type=Cream; syrup-type=Almond; sweetener-type=Sugar; milk-type=Skim;" --da
ta $'start\r\n' localhost/coffeepot
HTTP/1.1 200 OK
Date: Sat, 10 Jul 2021 12:50:36 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive

Started brewing your coffee...

p@crimson htcpcp]$ curl -i -X POST -H "Accept-Additions: milk-type=Cream; syrup-type=Almond; sweetener-type=Sugar; milk-type=Skim;" --da
ta $'stop\r\n' localhost/coffeepot
HTTP/1.1 200 OK
Date: Sat, 10 Jul 2021 12:50:47 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive

Your Coffee with Skim Milk, Sugar, and Almond Syrup is ready!

p@crimson htcpcp]$
```

Figure 5 Sending requests to coffee pot

```
p@crimson htcpcp]$ kubectl apply -f k8s.yml
pod/htcpcp-coffeepot created
service/coffeepot-service created
pod/htcpcp-teapot created
service/teapot-service created
ingress.networking.k8s.io/htcpcp-ingress created
p@crimson htcpcp]$ kubectl logs htcpcp-coffeepot
listening as a coffeepot on port 80
Milk set to Skim Milk
Sweetener set to Sugar
Syrup set to Almond Syrup
Spice set to None
Alcohol set to None
Response sent.
Response sent.
```

Figure 6 Coffee pot server-side logs

```
p@crimson htcpcp]$ curl -i -X POST -H "Accept-Additions: milk-type=Cream; syrup-type=Almond; sweetener-type=Sugar; milk-type=Skim;" --da
ta $'start\r\n' localhost/teapot
HTTP/1.1 418 I'm a teapot
Date: Sat, 10 Jul 2021 12:53:57 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive

I'm a teapot

p@crimson htcpcp]$
```

Figure 7 Sending requests to tea pot

```
p@crimson htcpcp]$ kubectl logs htcpcp-teapot
listening as a teapot on port 80
Response sent.
p@crimson htcpcp]$
```

Figure 8 Tea pot server-side logs

12 Future Scope of the project

As for the scope of future developments in this project, there can be several additions such as:

- Extending support towards more IoT devices
- Adding more tests to cover more cases
- Increasing the number of available additions
- The development of a GUI client for ease of use
- Adding more pot classes representing combined pot, instant pot, etc
- Further improving documentation.

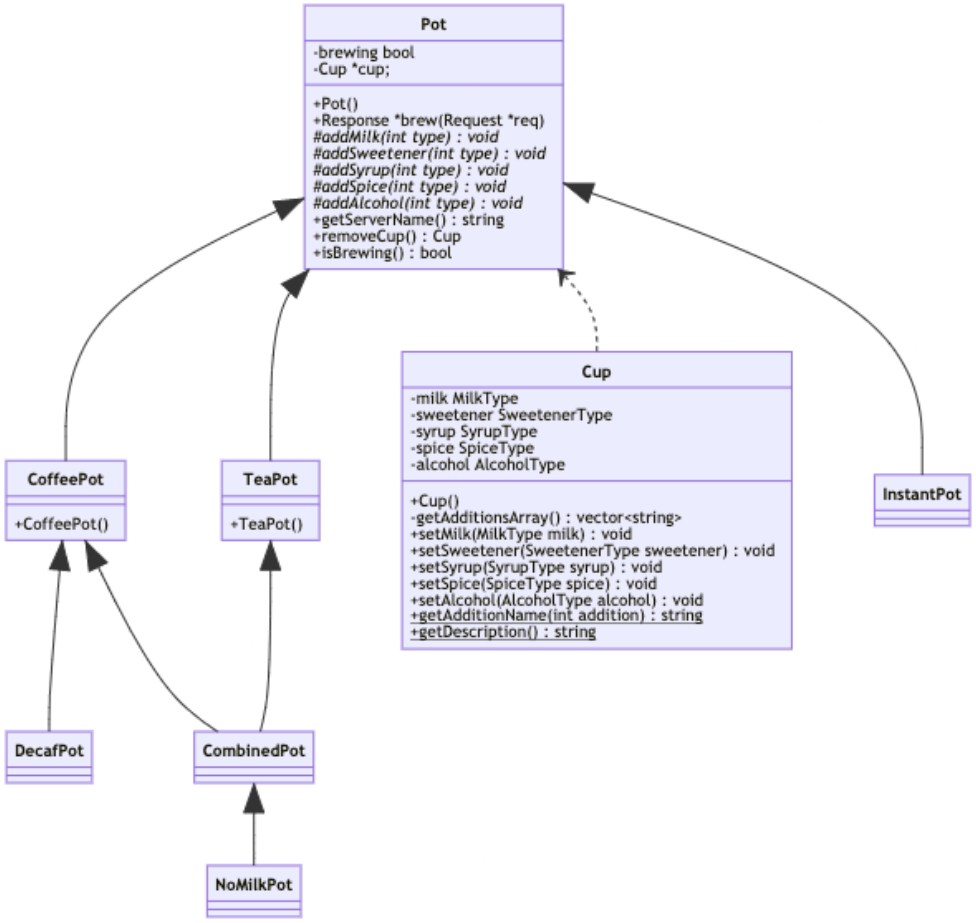


Figure 9

13 Hierarchical Index

13.1 Class Hierarchy

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

14 Class Index

14.1 Class List

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

15 File Index

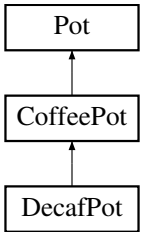
15.1 File List

Here is a list of all files with brief descriptions:

16 Class Documentation

16.1 CoffeePot Class Reference

`#include <CoffeePot.h>`
Inheritance diagram for CoffeePot:



Public Member Functions

- CoffeePot ()
- virtual std::string getServerName ()

Additional Inherited Members

16.1.1 Constructor & Destructor Documentation

16.1.1.1 CoffeePot() `CoffeePot::CoffeePot ( ) [inline]`
the default constructor

16.1.2 Member Function Documentation

16.1.2.1 getServerName() `std::string CoffeePot::getServerName ( ) [virtual]`
get the name of this pot

Returns

this pot's name

Reimplemented from Pot.
Reimplemented in DecafPot.
The documentation for this class was generated from the following files:

- pots/CoffeePot.h
- pots/CoffeePot.cpp

16.2 Cup Class Reference

`#include <Cup.h>`

Public Member Functions

- Cup ()
- void setMilk (MilkType milk)
- void setSweetener (SweetenerType sweetener)
- void setSyrup (SyrupType syrup)
- void setSpice (SpiceType spice)
- void setAlcohol (AlcoholType alcohol)
- std::string getDescription ()

Static Public Member Functions

- static std::string getAdditionName (int addition)

Private Member Functions

- std::vector< std::string > getAdditionsArray ()

Private Attributes

- MilkType milk
- SweetenerType sweetener
- SyrupType syrup
- SpiceType spice
- AlcoholType alcohol

16.2.1 Constructor & Destructor Documentation

16.2.1.1 Cup() `Cup::Cup ( )`

basic constructor

16.2.2 Member Function Documentation

16.2.2.1 getAdditionName() `std::string Cup::getAdditionName ( int addition ) [static]`

get addition name

Parameters

addition	code of addition

Returns

string of addition

16.2.2.2 getAdditionsArray() `std::vector< std::string > Cup::getAdditionsArray ( ) [private]`

Returns

array of strings of additions in this cup

16.2.2.3 getDescription() `std::string Cup::getDescription ( )`

get description of cup

16.2.2.4 setAlcohol() `void Cup::setAlcohol ( AlcoholType alcohol )`

setter for alcohol

16.2.2.5 setMilk() `void Cup::setMilk ( MilkType milk )`

setter for milk

16.2.2.6 setSpice() `void Cup::setSpice (`

`SpiceType spice )`

setter for spice

16.2.2.7 setSweetener() `void Cup::setSweetener (`

`SweetenerType sweetener )`

setter for sweetener

16.2.2.8 setSyrup() `void Cup::setSyrup (`

`SyrupType syrup )`

setter for syrup

16.2.3 Member Data Documentation

16.2.3.1 alcohol `AlcoholType Cup::alcohol [private]`

16.2.3.2 milk `MilkType Cup::milk [private]`

16.2.3.3 spice `SpiceType Cup::spice [private]`

16.2.3.4 sweetener `SweetenerType Cup::sweetener [private]`

16.2.3.5 syrup `SyrupType Cup::syrup [private]`

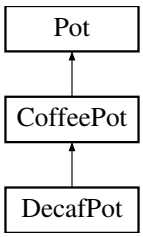
The documentation for this class was generated from the following files:

- pots/Cup.h
- pots/Cup.cpp

16.3 DecafPot Class Reference

`#include <DecafPot.h>`

Inheritance diagram for DecafPot:



Public Member Functions

- DecafPot ()
- virtual std::string getServerName () override

Additional Inherited Members

16.3.1 Constructor & Destructor Documentation

16.3.1.1 DecafPot() `DecafPot::DecafPot ( ) [inline]`

the default constructor

16.3.2 Member Function Documentation

16.3.2.1 getServerName() `std::string DecafPot::getServerName ( ) [override], [virtual]`

get the name of this pot

Returns

this pot's name

Reimplemented from CoffeePot.

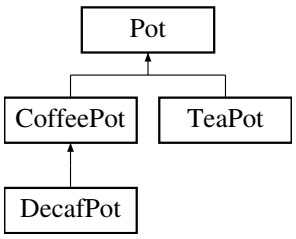
The documentation for this class was generated from the following files:

- pots/DecafPot.h
- pots/DecafPot.cpp

16.4 Pot Class Reference

`#include <Pot.h>`

Inheritance diagram for Pot:



Public Member Functions

- Pot ()
- Response * brew (Request *req)
- virtual std::string getServerName ()
- Cup * removeCup ()
- bool isBrewing ()

Protected Member Functions

- virtual void addMilk (int type)
- virtual void addSweetener (int type)
- virtual void addSyrup (int type)
- virtual void addSpice (int type)
- virtual void addAlcohol (int type)

Private Attributes

- bool brewing
- Cup * cup

16.4.1 Constructor & Destructor Documentation

16.4.1.1 Pot() `Pot::Pot ( ) [inline]`

a constructor

16.4.2 Member Function Documentation

16.4.2.1 addAlcohol() `void Pot::addAlcohol ( int type ) [protected], [virtual]`

adds the requested alcohol to the cup while brewing
Reimplemented in TeaPot.

16.4.2.2 addMilk() `void Pot::addMilk ( int type ) [protected], [virtual]`

adds the requested milk to the cup while brewing
Reimplemented in TeaPot.

16.4.2.3 addSpice() `void Pot::addSpice ( int type ) [protected], [virtual]`

adds the requested spice to the cup while brewing
Reimplemented in TeaPot.

16.4.2.4 addSweetener() `void Pot::addSweetener ( int type ) [protected], [virtual]`

adds the requested sweetener to the cup while brewing
Reimplemented in TeaPot.

16.4.2.5 addSyrup() `void Pot::addSyrup ( int type ) [protected], [virtual]`

adds the requested syrup to the cup while brewing
Reimplemented in TeaPot.

16.4.2.6 brew() `Response * Pot::brew ( Request * req )`

responds to BREW and POST requests, else sends a response with appropriate error

16.4.2.7 getServerName() `std::string Pot::getServerName ( ) [virtual]`

getter for current server name
Reimplemented in DecafPot, TeaPot, and CoffeePot.

16.4.2.8 isBrewing() `bool Pot::isBrewing ( )`

getter for brewing

16.4.2.9 removeCup() `Cup * Pot::removeCup ( )`

removes and returns the cup pointed to by this pot
get the cup from this pot

Returns

pointer to cup if not removed already, nullptr otherwise

16.4.3 Member Data Documentation

16.4.3.1 brewing `bool Pot::brewing [private]`

a state variable

16.4.3.2 cup `Cup* Pot::cup [private]`

pointer to a cup in brewing state

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

- pots/Pot.h
- pots/Pot.cpp

16.5 Request Class Reference

`#include <Request.h>`

Public Member Functions

- Request (Socket *socket)
- std::string getMethod ()
- std::map< std::string, int > getAdditions ()
- std::string getBody ()

Private Attributes

- std::map< std::string, std::string > headers
- std::map< std::string, int > addition_map
- std::vector< std::string > body_headers
- std::string method
- std::string path
- std::string protocol
- std::string body

16.5.1 Constructor & Destructor Documentation

16.5.1.1 Request() `Request::Request ( Socket * socket )`

Construct a request object from incoming connection.

Parameters

socket	the socket representing incoming connection

16.5.2 Member Function Documentation

16.5.2.1 getAdditions() `std::map< std::string, int > Request::getAdditions ( )`

get the additions requested in this request

Returns

the additions requested

16.5.2.2 getBody() `std::string Request::getBody ( )`

get the body of this request

Returns

the request body

16.5.2.3 getMethod() `std::string Request::getMethod ( )`

get the method of the request(for example, BREW or POST)

Returns

the request method

16.5.3 Member Data Documentation

16.5.3.1 addition_map `std::map<std::string, int> Request::addition_map [private]`

Map for storing addition type and value pairs

16.5.3.2 body `std::string Request::body [private]`

The request body

16.5.3.3 body_headers `std::vector<std::string> Request::body_headers [private]`

A list of unparsed headers in the body of the request

16.5.3.4 headers `std::map<std::string, std::string> Request::headers [private]`

Map for storing header names and value pairs

16.5.3.5 method `std::string Request::method [private]`

The method of the request(for example, BREW, or POST)

16.5.3.6 path `std::string Request::path [private]`

The request path

16.5.3.7 protocol `std::string Request::protocol [private]`

The request protocol

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

- networking/Request.h
- networking/Request.cpp

16.6 Response Class Reference

`#include <Response.h>`

Public Member Functions

- Response (int response_code)
- Response (int response_code, std::string body)
- void addHeader (std::string key, std::string value)
- void sendResponse (Socket *socket)

Static Public Member Functions

- static std::string getResponseString (int code)

Private Attributes

- int response_code
- std::string response_code_string
- std::map< std::string, std::string > headers
- std::string body

16.6.1 Constructor & Destructor Documentation

16.6.1.1 Response() [1/2] Response::Response (int *response_code*)

A constructor

16.6.1.2 Response() [2/2] Response::Response (int *response_code*, std::string *body*)

A constructor

16.6.2 Member Function Documentation

16.6.2.1 addHeader() void Response::addHeader (std::string *key*, std::string *value*)

16.6.2.2 getResponseString() std::string Response::getResponseString (int *code*) [static]

Returns

a Response string

16.6.2.3 sendResponse() void Response::sendResponse (Socket * *socket*)

sends the Response with appropriate Response strings and header values

16.6.3 Member Data Documentation

16.6.3.1 body std::string Response::body [private]

16.6.3.2 headers std::map<std::string, std::string> Response::headers [private]

16.6.3.3 response_code int Response::response_code [private]

16.6.3.4 response_code_string std::string Response::response_code_string [private]

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

- networking/Response.h
- networking/Response.cpp

16.7 ServerSocket Class Reference

`#include <ServerSocket.h>`

Public Member Functions

- ServerSocket (int port)
- Socket * accept ()

Private Attributes

- struct sockaddr_in address
- int server_fd

16.7.1 Constructor & Destructor Documentation

16.7.1.1 ServerSocket() `ServerSocket::ServerSocket ( int port )`

creates a ServerSocket listening on the specified port

Parameters

port	the port to listen on

16.7.2 Member Function Documentation

16.7.2.1 accept() `Socket * ServerSocket::accept ( )`

wait for an incoming connection, and return a Socket object representing the incoming connection

Returns

a Socket object representing the incoming connection

16.7.3 Member Data Documentation

16.7.3.1 address `struct sockaddr_in ServerSocket::address [private]`

port and address we will listen on

16.7.3.2 server_fd `int ServerSocket::server_fd [private]`

file descriptor of incoming connections

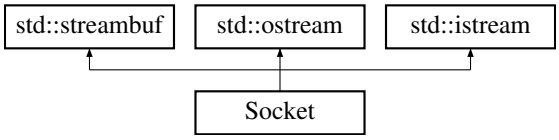
The documentation for this class was generated from the following files:

- networking/ServerSocket.h
- networking/ServerSocket.cpp

16.8 Socket Class Reference

`#include <Socket.h>`

Inheritance diagram for Socket:



Public Member Functions

- Socket (int fd)
- ~Socket ()
- void close ()

Protected Member Functions

- int overflow (int c)
- int underflow ()
- int sync ()

Private Attributes

- char outBuf_ [bufSize]
- char inBuf_ [bufSize+16 - sizeof(int)]
- int fd_

Static Private Attributes

- static const int bufSize = 1024

16.8.1 Constructor & Destructor Documentation

16.8.1.1 Socket() `Socket::Socket ( int fd )`

constructs a Socket object that uses fd for read/write calls

Parameters

fd	the file descriptor this socket is wrapped around

16.8.1.2 ~Socket() `Socket::~Socket ( )`

destructor to flush buffers and close file descriptor

16.8.2 Member Function Documentation

16.8.2.1 close() `void Socket::close ( )`

closes the connection

16.8.2.2 overflow() `int Socket::overflow ( int c ) [protected]`

writes a byte to the socket. we are required to override this as we are inheriting from ostream

Parameters

c	the byte to write.

Returns

EOF if writing is not possible

16.8.2.3 sync() `int Socket::sync ( ) [protected]`

flush the output buffer

Returns

-1 if we can't write to the socket, 0 otherwise

16.8.2.4 underflow() `int Socket::underflow ( ) [protected]`

read a byte from the socket. we are required to override this as we are inheriting from istream

Returns

one byte read from the socket, EOF otherwise

16.8.3 Member Data Documentation

16.8.3.1 bufSize `const int Socket::bufSize = 1024 [static], [private]`

The size of our buffers

16.8.3.2 fd_ `int Socket::fd_ [private]`

file descriptor pointing to our socket

16.8.3.3 inBuf_ `char Socket::inBuf_[bufSize+16 - sizeof(int)] [private]`

Array to buffer input

16.8.3.4 outBuf_ `char Socket::outBuf_[bufSize] [private]`

Array to buffer output

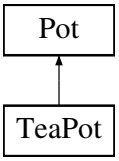
The documentation for this class was generated from the following files:

- networking/Socket.h
- networking/Socket.cpp

16.9 TeaPot Class Reference

`#include <TeaPot.h>`

Inheritance diagram for TeaPot:



Public Member Functions

- TeaPot ()
- virtual std::string getServerName ()

Protected Member Functions

- void addMilk (int type) override
- void addSweetener (int type) override
- void addSyrup (int type) override
- void addSpice (int type) override
- void addAlcohol (int type) override

16.9.1 Constructor & Destructor Documentation

16.9.1.1 TeaPot() `TeaPot::TeaPot ( ) [inline]`

16.9.2 Member Function Documentation

16.9.2.1 addAlcohol() `void TeaPot::addAlcohol ( int type ) [override], [protected], [virtual]`

throws error 418
Reimplemented from Pot.

16.9.2.2 addMilk() `void TeaPot::addMilk ( int type ) [override], [protected], [virtual]`

throws error 418
Reimplemented from Pot.

16.9.2.3 addSpice() `void TeaPot::addSpice ( int type ) [override], [protected], [virtual]`

throws error 418
Reimplemented from Pot.

16.9.2.4 addSweetener() `void TeaPot::addSweetener ( int type ) [override], [protected], [virtual]`

throws error 418
Reimplemented from Pot.

16.9.2.5 addSyrup() `void TeaPot::addSyrup ( int type ) [override], [protected], [virtual]`

throws error 418
Reimplemented from Pot.

16.9.2.6 getServerName() `std::string TeaPot::getServerName ( ) [virtual]`

get the name of this pot

Returns

this pot's name

Reimplemented from Pot.

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

- pots/TeaPot.h
- pots/TeaPot.cpp

17 File Documentation

17.1 htcpcp-impl.wiki/1.-Introduction.md File Reference

17.2 htcpcp-impl.wiki/2.-Goal.md File Reference

17.3 htcpcp-impl.wiki/3.-Tech-Stack-\-Tooling.md File Reference

17.4 htcpcp-impl.wiki/4.1-Pot.md File Reference

17.5 htcpcp-impl.wiki/4.2-Derived-Pots.md File Reference

17.6 htcpcp-impl.wiki/4.3-Cup-\--Additions.md File Reference

17.7 htcpcp-impl.wiki/4.4-Request\Response.md File Reference

17.8 htcpcp-impl.wiki/4.5-Networking.md File Reference

17.9 htcpcp-impl.wiki/4.6-Docker-K8s.md File Reference

17.10 htcpcp-impl.wiki/5.-Processing.md File Reference

17.11 htcpcp-impl.wiki/6.-Results.md File Reference

17.12 htcpcp-impl.wiki/7.-Future-Scope-of-the-project.md File Reference

17.13 htcpcp-impl.wiki/_Footer.md File Reference

17.14 main.cpp File Reference

```
#include <iostream>
#include <string>
#include <cstring>
#include "networking/ServerSocket.h"
#include "networking/Socket.h"
#include "networking/Request.h"
#include "networking/Response.h"
#include "pots/Pot.h"
#include "pots/CoffeePot.h"
#include "pots/DecafPot.h"
#include "pots/TeaPot.h"
```

Functions

- int main (int argc, char **argv)

17.14.1 Function Documentation

17.14.1.1 main() `int main (`

```
int argc,
char ** argv )
```

The entrypoint to our program. It takes in the port and pot type, ans starts a ServerSocket to listen at the specified port. When a request is received, it uses the brew method of the appropriate pot to brew your coffee and return the correct response

Parameters

argc	
argv	

Returns

nothing, it listens forever

17.15 networking/CMakeFiles/networking.dir/Request.cpp.o.d File Reference

17.16 networking/CMakeFiles/networking.dir/Response.cpp.o.d File Reference

17.17 networking/CMakeFiles/networking.dir/ServerSocket.cpp.o.d File Reference

17.18 networking/CMakeFiles/networking.dir/Socket.cpp.o.d File Reference

17.19 networking/Request.cpp File Reference

```
#include "Request.h"
#include "additions/AlcoholType.h"
#include "additions/MilkType.h"
#include "additions/SpiceType.h"
#include "additions/SweetenerType.h"
#include "additions/SyrupType.h"
#include <iostream>
#include <sstream>
```

Functions

- int getAddition (std::string type, std::string content)
- std::string readuntil (Socket *in, std::string delimiter)

17.19.1 Function Documentation

17.19.1.1 getAddition() `int getAddition (`

```
std::string type,
std::string content )
```

17.19.1.2 readuntil() `std::string readuntil (`

```
Socket * in,
std::string delimiter )
```

17.20 networking/Request.h File Reference

```
#include "Socket.h"
#include <string>
#include <map>
```

```
#include <vector>
```

Classes

- class Request

17.21 networking/Response.cpp File Reference

```
#include "Response.h"
#include <iostream>
```

17.22 networking/Response.h File Reference

```
#include "Request.h"
#include <string>
#include <map>
```

Classes

- class Response

17.23 networking/ServerSocket.cpp File Reference

```
#include "ServerSocket.h"
```

17.24 networking/ServerSocket.h File Reference

```
#include <sys/socket.h>
#include <netinet/in.h>
#include "Socket.h"
```

Classes

- class ServerSocket

17.25 networking/Socket.cpp File Reference

```
#include "Socket.h"
```

17.26 networking/Socket.h File Reference

```
#include <streambuf>
#include <ostream>
#include <istream>
#include <sys/socket.h>
#include <unistd.h>
```

Classes

- class Socket

17.27 pots/additions/AlcoholType.h File Reference

Enumerations

- enum class AlcoholType {
  NONE = 0 , WHISKY , RUM , KAHLUA ,
  AQUAVIT }

17.27.1 Enumeration Type Documentation

17.27.1.1 AlcoholType `enum AlcoholType [strong]`

This enum represents all the available alcohol type additions

Enumerator

NONE	
WHISKY	
RUM	
KAHLUA	
AQUAVIT	

17.28 pots/additions/MilkType.h File Reference

Enumerations

- enum class MilkType {
  NONE = 1 << 29 , CREAM , HALF_AND_HALF , WHOLE_MILK ,
  PART_SKIM , SKIM , NON_DAIRY }

17.28.1 Enumeration Type Documentation

17.28.1.1 MilkType `enum MilkType [strong]`

This enum represents all the available milk type additions

Enumerator

NONE	
CREAM	
HALF_AND_HALF	
WHOLE_MILK	
PART_SKIM	
SKIM	
NON_DAIRY	

17.29 pots/additions/SpiceType.h File Reference

Enumerations

- enum class SpiceType {
  NONE = 2 << 29 , CINNAMON , NUTMEG , CARDAMOM ,
  CLOVE }

17.29.1 Enumeration Type Documentation

17.29.1.1 SpiceType `enum SpiceType [strong]`

This enum represents all the available spice type additions

Enumerator

NONE	
CINNAMON	
NUTMEG	
CARDAMOM	
CLOVE	

17.30 pots/additions/SweetenerType.h File Reference

Enumerations

- enum class SweetenerType {
  NONE = 3 << 29 , SUGAR , STEVIA , HONEY ,
  MAPLE_SYRUP , AGAVE }

17.30.1 Enumeration Type Documentation

17.30.1.1 SweetenerType `enum SweetenerType [strong]`

This enum represents all the available sweetener type additions

Enumerator

NONE	
SUGAR	
STEVIA	
HONEY	
MAPLE_SYRUP	
AGAVE	

17.31 pots/additions/SyrupType.h File Reference

Enumerations

- enum class SyrupType {
  NONE = 4 << 29 , VANILLA , ALMOND , RASPBERRY ,
  CHOCOLATE }

17.31.1 Enumeration Type Documentation

17.31.1.1 SyrupType `enum SyrupType [strong]`

This enum represents all the available syrup type additions

Enumerator

NONE	
VANILLA	
ALMOND	
RASPBERRY	
CHOCOLATE	

17.32 pots/CMakeFiles/pots.dir/CoffeePot.cpp.o.d File Reference

17.33 pots/CMakeFiles/pots.dir/Cup.cpp.o.d File Reference

17.34 pots/CMakeFiles/pots.dir/DecafPot.cpp.o.d File Reference

17.35 pots/CMakeFiles/pots.dir/Pot.cpp.o.d File Reference

17.36 pots/CMakeFiles/pots.dir/TeaPot.cpp.o.d File Reference

17.37 pots/CoffeePot.cpp File Reference

```
#include "CoffeePot.h"
```

17.38 pots/CoffeePot.h File Reference

```
#include "Pot.h"
```

Classes

- class CoffeePot

17.39 pots/Cup.cpp File Reference

```
#include <iostream>
#include <string>
#include "Cup.h"
#include "additions/AlcoholType.h"
#include "additions/MilkType.h"
#include "additions/SpiceType.h"
#include "additions/SweetenerType.h"
#include "additions/SyrupType.h"
```

17.40 pots/Cup.h File Reference

```
#include <string>
#include <vector>
#include "additions/AlcoholType.h"
#include "additions/MilkType.h"
#include "additions/SpiceType.h"
#include "additions/SweetenerType.h"
#include "additions/SyrupType.h"
```

Classes

- class Cup

17.41 pots/DecafPot.cpp File Reference

```
#include "DecafPot.h"
```

17.42 pots/DecafPot.h File Reference

```
#include "CoffeePot.h"
```

Classes

- class DecafPot

17.43 pots/Pot.cpp File Reference

```
#include "Pot.h"
#include "Cup.h"
#include "additions/AlcoholType.h"
#include "additions/MilkType.h"
#include "additions/SpiceType.h"
#include "additions/SweetenerType.h"
#include "additions/SyrupType.h"
#include <iostream>
```

17.44 pots/Pot.h File Reference

```
#include <string>
#include <map>
#include "Cup.h"
#include <Socket.h>
#include <Request.h>
#include <Response.h>
#include "additions/AlcoholType.h"
#include "additions/MilkType.h"
#include "additions/SpiceType.h"
#include "additions/SweetenerType.h"
#include "additions/SyrupType.h"
```

Classes

- class Pot

17.45 pots/TeaPot.cpp File Reference

```
#include "TeaPot.h"
```

17.46 pots/TeaPot.h File Reference

```
#include "Pot.h"
```

Classes

- class TeaPot

17.47 README.md File Reference

17.48 tests/pots/CupDescriptionTest.cpp File Reference

```
#include <iostream>
#include "Cup.h"
```

Functions

- int main ()

17.48.1 Function Documentation

17.48.1.1 main() `int main ( )`

17.49 tests/pots/PotBrewTest.cpp File Reference

```
#include <iostream>
#include <Pot.h>
```

Functions

- int main ()

17.49.1 Function Documentation

17.49.1.1 main() `int main ( )`

17.50 tests/pots/PotCupTest.cpp File Reference

```
#include <iostream>
#include <Pot.h>
```

Functions

- int main ()

17.50.1 Function Documentation

17.50.1.1 main() `int main ( )`

Index