Code Documentation

Table of Contents

[Resource Files 2](#_Toc63974112)

[compile 3](#_Toc63974113)

[Purpose 3](#_Toc63974114)

[Variables 3](#_Toc63974115)

[Connections / Uses 3](#_Toc63974116)

[DRTBL 4](#_Toc63974117)

[Purpose 4](#_Toc63974118)

[Variables 4](#_Toc63974119)

[Connections / Uses 4](#_Toc63974120)

[EXEC 5](#_Toc63974121)

[Purpose 5](#_Toc63974122)

[Variables 5](#_Toc63974123)

[Connections / Uses 5](#_Toc63974124)

[GUIMain 6](#_Toc63974125)

[Purpose 6](#_Toc63974126)

[Variables 6](#_Toc63974127)

[Connections / Uses 6](#_Toc63974128)

[Makefile 7](#_Toc63974129)

[Purpose 7](#_Toc63974130)

[Variables 7](#_Toc63974131)

[Connections / Uses 7](#_Toc63974132)

[Header / Cpp Files 8](#_Toc63974133)

[Antenna 9](#_Toc63974134)

[Purpose 9](#_Toc63974135)

[Variables 9](#_Toc63974136)

[Connections / Uses 9](#_Toc63974137)

[BaseStation 10](#_Toc63974138)

[Purpose 10](#_Toc63974139)

[Variables 10](#_Toc63974140)

[Connections / Uses 10](#_Toc63974141)

[BSFailureParams 11](#_Toc63974142)

[Purpose 11](#_Toc63974143)

[Variables 11](#_Toc63974144)

[Connections / Uses 11](#_Toc63974145)

[DataRateTable 12](#_Toc63974146)

[Purpose 12](#_Toc63974147)

[Variables 12](#_Toc63974148)

[Connections / Uses 12](#_Toc63974149)

[DebugMain 13](#_Toc63974150)

[Purpose 13](#_Toc63974151)

[Variables 13](#_Toc63974152)

[Connections / Uses 13](#_Toc63974153)

[EnvironmentInitialization 14](#_Toc63974154)

[Purpose 14](#_Toc63974155)

[Variables 14](#_Toc63974156)

[Connections / Uses 14](#_Toc63974157)

[ErrorTracer 15](#_Toc63974158)

[Purpose 15](#_Toc63974159)

[Variables 15](#_Toc63974160)

[Connections / Uses 15](#_Toc63974161)

[FileIO 16](#_Toc63974162)

[Purpose 16](#_Toc63974163)

[Variables 16](#_Toc63974164)

[Connections / Uses 16](#_Toc63974165)

[GUIDataContainer 17](#_Toc63974166)

[Purpose 17](#_Toc63974167)

[Variables 17](#_Toc63974168)

[Connections / Uses 17](#_Toc63974169)

[IRPManager 18](#_Toc63974170)

[Purpose 18](#_Toc63974171)

[Variables 18](#_Toc63974172)

[Connections / Uses 18](#_Toc63974173)

[NetworkLogBuffer 19](#_Toc63974174)

[Purpose 19](#_Toc63974175)

[Variables 19](#_Toc63974176)

[Connections / Uses 19](#_Toc63974177)

[Setup 20](#_Toc63974178)

[Purpose 20](#_Toc63974179)

[Variables 20](#_Toc63974180)

[Connections / Uses 20](#_Toc63974181)

[Simulator 21](#_Toc63974182)

[Purpose 21](#_Toc63974183)

[Variables 21](#_Toc63974184)

[Connections / Uses 21](#_Toc63974185)

[Transceivers 22](#_Toc63974186)

[Purpose 22](#_Toc63974187)

[Variables 22](#_Toc63974188)

[Connections / Uses 22](#_Toc63974189)

[Transmission 23](#_Toc63974190)

[Purpose 23](#_Toc63974191)

[Variables 23](#_Toc63974192)

[Connections / Uses 23](#_Toc63974193)

[UEDataBase 24](#_Toc63974194)

[Purpose 24](#_Toc63974195)

[Variables 24](#_Toc63974196)

[Connections / Uses 24](#_Toc63974197)

[UELogData 25](#_Toc63974198)

[Purpose 25](#_Toc63974199)

[Variables 25](#_Toc63974200)

[Connections / Uses 25](#_Toc63974201)

[UERecord 26](#_Toc63974202)

[Purpose 26](#_Toc63974203)

[Variables 26](#_Toc63974204)

[Connections / Uses 26](#_Toc63974205)

[UserEquipment 27](#_Toc63974206)

[Purpose 27](#_Toc63974207)

[Variables 27](#_Toc63974208)

[Connections / Uses 27](#_Toc63974209)

# Resource Files

Used to generate the Graphical User Interface (GUI)

Make output, make clean, etc. is here too

## compile.sh (not implemented yet)

Purpose

Automatically compile code by running:

make clean

make output

./output

Variables

N/A

Connections / Uses

N/A

## DRTBL

Purpose

The data rate table class is used to load the data rate table that provides the SNR values into memory while also providing an interface to use it with. The data rate table values are used to give a user connected to the eNodeB a random SNR value between -12 db and 12 db in order to provide the network with a realistic model for random SNR value generation. The process the script takes to generate these SNR values is covered in more details in the section “Data Rate and Power Calculations”.

The values for spectral efficiency (SE), found in columns labeled Value1-Value4, are generated using QAM 16 precoding.

Variables

**SNR, SNRNext:** Used to index through values 1-4 depending on the SNR (range of -12dB to 12dB)

**Values 1-4:** Spectral Efficiency

Connections / Uses

TBD

## EXEC (not implemented yet)

Purpose

To execute some code with Linux. Refer to complie.sh for more information.

Variables

N/A

Connections / Uses

compile.sh

## GUIMain.h

Purpose

Header file used to define structures, variables, and functions used in GUIMain.cpp.

Variables

Uses GTK Function / Variable definitions.

Connections / Uses

GUIMain.cpp

## GUIMain.cpp

Purpose

Variables

Connections / Uses

## Makefile

Purpose

To generate .o (aka .obj [object]) files for GCC’s make library.

make output, make clean

refer to documentation via their official website: <https://gcc.gnu.org/>

Variables

N/A

Connections / Uses

Compile.sh

# Header / Cpp Files

Used in the main code to do stuff ok thanks

## Antenna

Purpose

The Antenna class functions as a container for the transceivers that connect to the user equipment. The antennae are generated only in the construction of the BS. This should be renamed to sector someday.

Variables

**antID:** This represents the place in the BaseStation's BSAntennae vector that the antenna is in

**loc:** The exact location of the antenna in the environment

**antAngle:** This is the angle of the antenna relative to the BaseStation. This is always some multiple of 2PI/#Antenna per BS

**connectionInfo:** This is the container for the transceivers, which can be viewed as the connections to the UEs. To see more go to TransceiverList.h

Connections / Uses

## BaseStation

Purpose

The BaseStation class function models a "cell tower". It contains a database with information about its connected users. It also has a user-specified number of Antennae, which in turn contains Transceivers that connect to the users. It also has a buffer that stores the data that is to be sent out to the UEs. This buffer allows up to twice the maximum amount of data as the BS can send out in a tick. This means that if the total demand on the BS is more than twice the maximum BS data rate, data will be dropped. Every tick the BS is guaranteed to send out at most its maximum data rate unless its "failed" parameter is set to true.

Variables

**bsID:** This represents the place in the Simulator's BaseStationList vector that the BS is in.

**loc:** The exact location of the BS in the environment

**dataRate:** This is the total data that is sent out by the BS in the last tick that occurred. It is reset to 0 before sending data out in a new tick.

**failed:** This is by default 'false', it is only flipped to 'true' by the Environment Controller if the BS is designated to be in failure.

**BSAntennae:** This is the container that holds the Antennae possessed by the BS, for more look at Antenna.h

**userRecords:** This database contains valuable info about the connected UEs including their SNR, antenna, transceiver, last known datarate, location, etc. Check out UEDataBase.h or UERecord.h for more

**outgoingTransmissions:** Buffer that contains all the outgoing data to be sent out to the users. Limited to 2x the max BS Datarate arbitrarily. Each Transmission contains info regarding the recipient UE, as well as the antenna and transceiver that is "used" to send the data out, and more. See Transmission.h for more info.

Connections / Uses

**EnvironmentInitialization.cpp:** Used to add base stations

**FileIO.cpp:** Reads base station ID, coordinates to export them to CSV later.

**Simulator.cpp:** Lots of getters to be able to access the private variables inside of each base station.

## BSFailureParams

### Purpose

The Transmission struct functions as a simple information container that has everything the EnvironmentController needs to maintain a BS. Each BS has a corresponding BSFailureParams. The BSFailureParams are stored in the Environment Controller's BSRegionControlInfo in the same order as the BSs are within the Simulators BaseStationList.

### Variables

**bsID:** the ID of the BS the BSFailureParams pertains to

**currentState:** (sum of all UE demands in the UEsInRegion for the previous tick) / (Maximum BS datarate)

**endState:** the target goal, this is the number that the currentState should approach once failure is activated. If the endState is unacheiveable, perhaps due to being unable to add enough users to a BS, the endState will adjust itself to the currentState that was acheived.

**currentStatus:** this is the current status, this is set to the endstatus at the end of its risetime.

**endStatus:** this is the target status.

**startTime:** this is the time that the BS should begin its transition to the endState

**riseTime:** this is the time it will take for the BS to reach its endState after starting its rise. Must be >= 1 or div/0 will occur.

**UEsInregion:** This contains all the userIDs that are present within the BSs range. For illustration purposes: suppose the IRPManager does not ever transfer any UEs between BSs, then the UEsInRegion will perfectly match the UEs linked to the BS by the EnvironmentController. If the IRPManager were to transfer a UE from its initial BS to another BS, the UE would still be present in the initial BS's UEsInRegion. Reason why I did this: Pretend that the EnvironmentController is a force of nature that ensures that a certain BS is Congested. To acheive this the EnvironmentController magically creates UEs to load down the BS to raise the demand placed upon it. Now consider the IRPManager. Its job is basically shuffle users around to allieviate load, and does so by directly looking at the UEs linked to the BS. Now, from the POV of the EnvironmentController, the "load" it's trying to place on the BS is measured by looking at the UEsInRegion and summing all their demands and comparing that to the max BS Datarate. If the EnvironmentController were measuring the demands of the users based off those that are DEFINITELY linked to the BS, then it would try and add users or adjust their demands at the same time the IRPManager is trying to transfer users based off the same measurement. The result would be the continual addition of users while the IRPManager does the exact opposite. Clearly this is undesirable. Therefore, the currentState that is measured is only the theoretical load the BS would be under if nothing were done by the IRPManager. Thus, ideally, with a functioning IRPManager, the BS could be in a healthy or less congested state while the EnvironmentController thinks that the BS is congested.

### Connections / Uses

## DataRateTable

### Purpose

### Variables

### Connections / Uses

## DebugMain

### Purpose

### Variables

### Connections / Uses

## EnvironmentInitialization

### Purpose

### Variables

### Connections / Uses

## ErrorTracer

### Purpose

### Variables

### Connections / Uses

## FileIO

### Purpose

### Variables

### Connections / Uses

## GUIDataContainer

### Purpose

### Variables

### Connections / Uses

## IRPManager

### Purpose

### Variables

### Connections / Uses

## NetworkLogBuffer

### Purpose

### Variables

### Connections / Uses

## Setup

### Purpose

### Variables

### Connections / Uses

## Simulator

### Purpose

### Variables

### Connections / Uses

## Transceivers

### Purpose

### Variables

### Connections / Uses

## Transmission

### Purpose

The Transmission struct functions as a simple "packet" container.

### Variables

**sender:** the bsID of the sending BS.

**destination:** the userID of the receiving UE.

**ant:** the antID of the sending Antenna

**tr:** the transceiver in the Antenna that is linked to the UE.

**data:** the size of the packet, unit does not matter.

### Connections / Uses

## UEDataBase

### Purpose

### Variables

### Connections / Uses

## UELogData

### Purpose

Log Data of User Equipment that is written to by the simulator and outputted onto the .csv by classes like FileIO.cpp.

### Variables

**TIME:** Simulation time

**BS\_ID:** BaseStation ID of the User Equipment

**BS\_LOC\_X:** X coordinate location of the User Equipment

**BS\_LOC\_Y:** Y coordinate location of the User Equipment

**ANT\_ID:** The ID of the antenna on the BaseStation that the User Equipment is connected to

**ANT\_SEC:** The angle of the antenna on the BaseStation that the User Equipment is connected to

**TRX\_ID:** The ID of the transceiver on the BaseStation that the User Equipment is connected to

**TRX\_X:** The X coordinate of that transceiver

**TRX\_Y:** The X coordinate of that transceiver

**TRX\_ANG:** The angle of that transceiver

**UE\_ID:** The ID of the user equipment

**UE\_MID:** The mobility ID of the user equipment that determines how the user will move based on the moveUE functions. (0 = Stationary, 1 = Walking, 2 = Driving)

**UE\_LOC\_X:** The X coordinate of that transceiver

**UE\_LOC\_Y:** The Y coordinate of that transceiver

**MAX\_DR:** The maximum possible data rate that the user equipment can request from a BaseStation

**DEMAND\_DR:** The data rate demanded by the user equipment from their BaseStation

**REAL\_DR:** The actual data rate that the user equipment gets from the BaseStation

**TRANS\_PWR:** The power transmitted by the BaseStation to the User Equipment

**REC\_PWR:** The power received by the User Equipment from the BaseStation

**RSRP:** Reference Signal Received Power (RSRP) of the User Equipment (power received)

**RSSI:** Reference Signal Strength Index (RSSI) of the User Equipment (how strong the signal is to the BaseStation)

**RSRQ:** Reference Signal Received Quality (RSRQ) of the User Equipment (the quality of the signal to the BaseStation)

**DDR:** Data Drop Rate of the User Equipment (packet loss in percentage)

### Connections / Uses

**void IRPManager::IRPDataCollection()** – Reads UELogData from the BaseStation and creates IRPManager::TickData

## UERecord

### Purpose

Recorded information about a specific User Equipment. This information is stored on a DataBase that lives on the BaseStation (UEDB). This information is updated by the BaseStation when users are added and removed.

### Variables

**userID:** The ID of the user equipment

**mobilityID:** The mobility ID of the user equipment that determines how the user will move based on the moveUE functions. (0 = Stationary, 1 = Walking, 2 = Driving)

**loc:** The X and Y coordinates of the User Equipment

**antenna:** The ID of the antenna on the BaseStation that the User Equipment is connected to

**currentTransceiver:** The ID of the transceiver on the antenna that the User Equipment is connected to

**currentSNR:** The current Signal to Noise Ratio of the User Equipment

**demand:** The current demand (in bits) of the User Equipment

**bitsSent:** How many bits are sent to the User Equipment

**powerSent:** How much power (watts) is being sent to the User Equipment

**rsrp:** Reference Signal Received Power (RSRP) of the User Equipment (power received)

**rssi:** Reference Signal Strength Index (RSSI) of the User Equipment (how strong the signal is to the BaseStation)

**rsrq:** Reference Signal Received Quality (RSRQ) of the User Equipment (the quality of the signal to the BaseStation)

**ddr**: Data Drop Rate of the User Equipment (packet loss in percentage)

### Connections / Uses

**BaseStation.cpp** – User Records are added to BaseStations

**void EnvironmentController::addUsers(BSFailureParams& bsfp, const uint32\_t& numUsers, float& diff)** – User Equipment Records used by EnvironmentController to create user congestion / demand congestion are created here.

**bool EnvironmentInitialization::setDefaultUsers()** – Initial User Equipment is created here, then assigned to a BaseStation along with their User Record.

**bool FileIO::readSaveFileIntoSim()** – User data is read and stored into a file so that the simulator can restart with the same initial conditions the next time around.

**UEDataBase.cpp** – This is a list of User Equipment Records that are managed by other classes like BaseStation.cpp.

## UserEquipment

### Purpose

Store information about a single user that lives within the simulator. They are connected to a single transceiver which helps them communicate and send data to a BaseStation via a transmission.

### Variables

**userID:** The ID of the user equipment

**mobilityID:** The mobility ID of the user equipment that determines how the user will move based on the moveUE functions. (0 = Stationary, 1 = Walking, 2 = Driving)

**loc:** The X and Y coordinates of the User Equipment

**dataDemanded:** How much data (in bits) that the user equipment is demanding from the BaseStation

**dataReceived:** How much data (in bits) that the user equipment is receiving from the BaseStation

**powerReceived:** How much power (watts?) that the user equipment is receiving from the BaseStation

**possibleMaxDrs:** A value that is read from the DataRateTable. It is related to the possible max data rates for a channel depending on the UE’s Signal to Noise Ratio (SNR)

### Connections / Uses

**Simulator.cpp** – User Equipments live in a list called UserEquipmentList inside of the Simulator.cpp environment. They can be modified with the get\_m() method and read with read-only properties by using the get() method.

**bool FileIO::readSaveFileIntoSim()** – User data is read and stored into a file so that the simulator can restart with the same initial conditions the next time around.

**bool EnvironmentInitialization::setDefaultUsers()** – Initial User Equipment information like ID, location, SNR, bitrate, etc. are created in this method.

**void EnvironmentController::addUsers(BSFailureParams& bsfp, const uint32\_t& numUsers, float& diff)** – User Equipments used by EnvironmentController to create user congestion / demand congestion are created here.