

SSRoboime

Generated by Doxygen 1.9.8

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 BasePlayer Class Reference	5
3.1.1 Detailed Description	7
3.1.2 Constructor & Destructor Documentation	7
3.1.2.1 BasePlayer()	7
3.1.3 Member Data Documentation	7
3.1.3.1 __scom	7
3.1.3.2 unum	7
3.2 ServerComm Class Reference	8
3.2.1 Detailed Description	9
3.2.2 Constructor & Destructor Documentation	9
3.2.2.1 ServerComm() [1/2]	9
3.2.2.2 ServerComm() [2/2]	9
3.2.2.3 ~ServerComm()	10
3.2.3 Member Function Documentation	10
3.2.3.1 __recv_all()	10
3.2.3.2 __set_blocking_mode()	10
3.2.3.3 clear_queue()	11
3.2.3.4 commit()	11
3.2.3.5 receive()	11
3.2.3.6 receive_async_sync()	11
3.2.3.7 send()	12
3.2.3.8 send_immediate()	12
3.2.4 Member Data Documentation	12
3.2.4.1 __HEADER_SIZE	12
3.2.4.2 __message_queue	12
3.2.4.3 __read_buffer	12
3.2.4.4 __sock_fd	13
4 File Documentation	15
4.1 src/Agent/BasePlayer.hpp File Reference	15
4.2 BasePlayer.hpp	16
4.3 src/Booting/booting_templates.hpp File Reference	16
4.3.1 Macro Definition Documentation	17
4.3.1.1 False	17
4.3.1.2 True	17
4.3.2 Variable Documentation	18

4.3.2.1 AGENT_HOST	18
4.3.2.2 AGENT_PORT	18
4.3.2.3 DEBUG_MODE	18
4.3.2.4 TEAM_NAME	18
4.4 booting_templates.hpp	18
4.5 src/Communication/ServerComm.hpp File Reference	19
4.6 ServerComm.hpp	20
4.7 src/run_full_team.cpp File Reference	23
4.8 run_full_team.cpp	23
4.9 src/run_player.cpp File Reference	23
4.9.1 Function Documentation	23
4.9.1.1 main()	23
4.10 run_player.cpp	24
Index	25

Chapter 1

Class Index

1.1 Class List

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

BasePlayer	5
ServerComm Responsável pela implementação da comunicação com servidor rcssserver3d	8

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

src/ run_full_team.cpp	23
src/ run_player.cpp	23
src/Agent/ BasePlayer.hpp	15
src/Booting/ booting_templates.hpp	16
src/Communication/ ServerComm.hpp	19

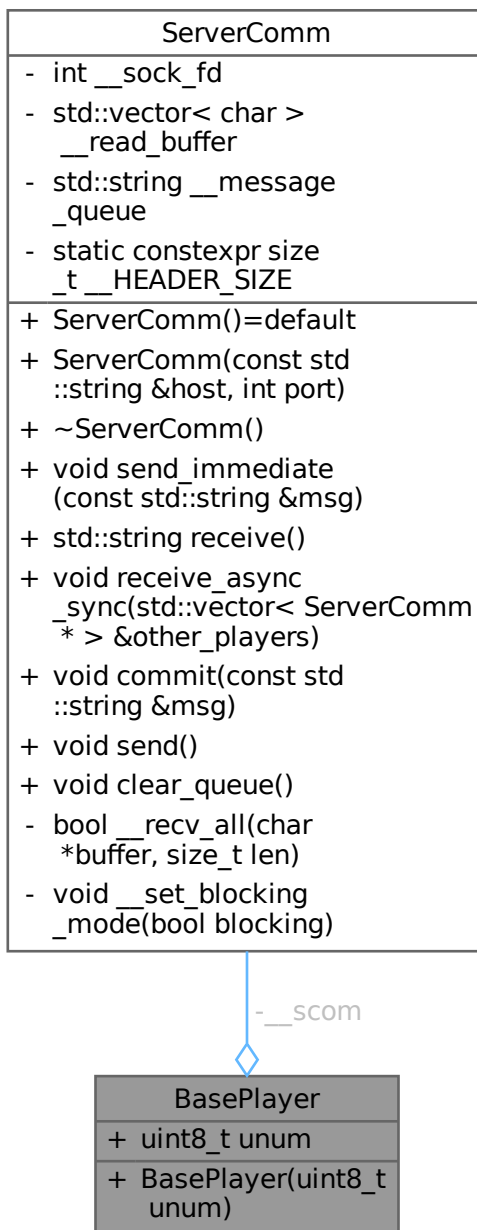
Chapter 3

Class Documentation

3.1 BasePlayer Class Reference

```
#include <BasePlayer.hpp>
```

Collaboration diagram for BasePlayer:



Public Member Functions

- [BasePlayer](#) (uint8_t unum)

Public Attributes

- uint8_t unum

Número de Uniforme, uint8_t para economizar memória, dado que varia de 1 à 11.

Private Attributes

- [ServerComm __scom](#)

Classe gerenciadora de comunicação com servidor rcssserver3d.

3.1.1 Detailed Description

Definition at line 6 of file [BasePlayer.hpp](#).

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BasePlayer()

```
BasePlayer::BasePlayer (
    uint8_t unum ) [inline]
```

< Realmente será útil em nosso código

Definition at line 14 of file [BasePlayer.hpp](#).

3.1.3 Member Data Documentation

3.1.3.1 __scom

```
ServerComm BasePlayer::__scom [private]
```

Classe gerenciadora de comunicação com servidor rcssserver3d.

Definition at line 8 of file [BasePlayer.hpp](#).

3.1.3.2 unum

```
uint8_t BasePlayer::unum
```

Número de Uniforme, `uint8_t` para economizar memória, dado que varia de 1 à 11.

Definition at line 11 of file [BasePlayer.hpp](#).

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

- [src/Agent/BasePlayer.hpp](#)

3.2 ServerComm Class Reference

Responsável pela implementação da comunicação com servidor rcssserver3d.

```
#include <ServerComm.hpp>
```

Collaboration diagram for ServerComm:

ServerComm
<ul style="list-style-type: none"> - int __sock_fd - std::vector< char > __read_buffer - std::string __message_queue - static constexpr size_t __HEADER_SIZE
<ul style="list-style-type: none"> + ServerComm()=default + ServerComm(const std::string &host, int port) + ~ServerComm() + void send_immediate(const std::string &msg) + std::string receive() + void receive_async_sync(std::vector< ServerComm * > &other_players) + void commit(const std::string &msg) + void send() + void clear_queue() - bool __recv_all(char *buffer, size_t len) - void __set_blocking_mode(bool blocking)

Public Member Functions

- [ServerComm](#) ()=default
- [ServerComm](#) (const std::string &host, int port)
Construtor que inicializa o socket e buffers.
- [~ServerComm](#) ()
Destrutor para limpeza de recursos.
- void [send_immediate](#) (const std::string &msg)
Envia uma mensagem instantânea usando Scatter/Gather I/O (writev).

- `std::string receive ()`
Recebe dados do socket, processa o cabeçalho e extrai o payload.
- `void receive_async_sync (std::vector< ServerComm * > &other_players)`
Lógica especial para sincronização inicial (handshake) com múltiplos agentes.
- `void commit (const std::string &msg)`
Adiciona uma mensagem à fila de envio (bufferização).
- `void send ()`
Envia todas as mensagens da fila de uma vez.
- `void clear_queue ()`
Limpa a fila de mensagens sem enviar.

Private Member Functions

- `bool __recv_all (char *buffer, size_t len)`
Garante o recebimento completo de N bytes (lida com fragmentação TCP).
- `void __set_blocking_mode (bool blocking)`
Configura o socket para modo bloqueante ou não-bloqueante.

Private Attributes

- `int __sock_fd`
File descriptor do socket.
- `std::vector< char > __read_buffer`
Buffer persistente para leitura de dados (evita realocações)
- `std::string __message_queue`
Fila de mensagens a serem enviadas (buffer de escrita)

Static Private Attributes

- `static constexpr size_t __HEADER_SIZE = 4`
Tamanho do cabeçalho padrão do protocolo rcssserver3d.

3.2.1 Detailed Description

Responsável pela implementação da comunicação com servidor rcssserver3d.

< — Bibliotecas da Standard Library (C++) — Container dinâmico usado para buffers de leitura Manipulação de strings para filas de mensagens Entrada e saída padrão (`std::cerr`, `std::cout`) < — Bibliotecas de Sistema (POSIX/↔ Linux) —

Definition at line 24 of file [ServerComm.hpp](#).

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ServerComm() [1/2]

```
ServerComm::ServerComm ( ) [default]
```

3.2.2.2 ServerComm() [2/2]

```
ServerComm::ServerComm (
    const std::string & host,
    int port ) [inline]
```

Construtor que inicializa o socket e buffers.

Parameters

<i>host</i>	Endereço IP do servidor.
<i>port</i>	Porta do servidor.

Definition at line 92 of file [ServerComm.hpp](#).

3.2.2.3 ~ServerComm()

```
ServerComm::~~ServerComm ( ) [inline]
```

Destrutor para limpeza de recursos.

Definition at line 151 of file [ServerComm.hpp](#).

3.2.3 Member Function Documentation**3.2.3.1 __recv_all()**

```
bool ServerComm::__recv_all (
    char * buffer,
    size_t len ) [inline], [private]
```

Garante o recebimento completo de N bytes (lida com fragmentação TCP).

Parameters

<i>buffer</i>	Ponteiro para onde os dados serão escritos.
<i>len</i>	Quantidade exata de bytes a ler.

Returns

True se leu tudo, False se a conexão caiu ou erro.

Definition at line 44 of file [ServerComm.hpp](#).

3.2.3.2 __set_blocking_mode()

```
void ServerComm::__set_blocking_mode (
    bool blocking ) [inline], [private]
```

Configura o socket para modo bloqueante ou não-bloqueante.

Parameters

<i>blocking</i>	True para bloqueante, False para não-bloqueante.
-----------------	--

Definition at line 68 of file [ServerComm.hpp](#).

3.2.3.3 clear_queue()

```
void ServerComm::clear_queue ( ) [inline]
```

Limpa a fila de mensagens sem enviar.

Definition at line 319 of file [ServerComm.hpp](#).

3.2.3.4 commit()

```
void ServerComm::commit (
    const std::string & msg ) [inline]
```

Adiciona uma mensagem à fila de envio (bufferização).

Parameters

<i>msg</i>	Mensagem em bytes/string.
------------	---------------------------

Definition at line 291 of file [ServerComm.hpp](#).

3.2.3.5 receive()

```
std::string ServerComm::receive ( ) [inline]
```

Recebe dados do socket, processa o cabeçalho e extrai o payload.

Returns

Uma string contendo o corpo da mensagem recebida (sem os 4 bytes de tamanho).

Definition at line 194 of file [ServerComm.hpp](#).

3.2.3.6 receive_async_sync()

```
void ServerComm::receive_async_sync (
    std::vector< ServerComm * > & other_players ) [inline]
```

Lógica especial para sincronização inicial (handshake) com múltiplos agentes.

Parameters

<i>other_players</i>	Ponteiro para vetor de outros agentes.
----------------------	--

Definition at line 247 of file [ServerComm.hpp](#).

3.2.3.7 send()

```
void ServerComm::send ( ) [inline]
```

Envia todas as mensagens da fila de uma vez.

Adiciona (syn) ao final automaticamente se o socket estiver livre para escrita.

Definition at line 299 of file [ServerComm.hpp](#).

3.2.3.8 send_immediate()

```
void ServerComm::send_immediate (
    const std::string & msg ) [inline]
```

Envia uma mensagem instantânea usando Scatter/Gather I/O (writev).

Parameters

<i>msg</i>	A string de dados a ser enviada.
------------	----------------------------------

Usa writev para enviar Header+Body em uma única syscall sem cópia de memória.

Definition at line 162 of file [ServerComm.hpp](#).

3.2.4 Member Data Documentation

3.2.4.1 __HEADER_SIZE

```
constexpr size_t ServerComm::__HEADER_SIZE = 4 [static], [constexpr], [private]
```

Tamanho do cabeçalho padrão do protocolo rcssserver3d.

Definition at line 36 of file [ServerComm.hpp](#).

3.2.4.2 __message_queue

```
std::string ServerComm::__message_queue [private]
```

Fila de mensagens a serem enviadas (buffer de escrita)

Definition at line 33 of file [ServerComm.hpp](#).

3.2.4.3 __read_buffer

```
std::vector<char> ServerComm::__read_buffer [private]
```

Buffer persistente para leitura de dados (evita realocações)

Definition at line 30 of file [ServerComm.hpp](#).

3.2.4.4 __sock_fd

```
int ServerComm::__sock_fd [private]
```

File descriptor do socket.

Definition at line 27 of file [ServerComm.hpp](#).

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

- [src/Communication/ServerComm.hpp](#)

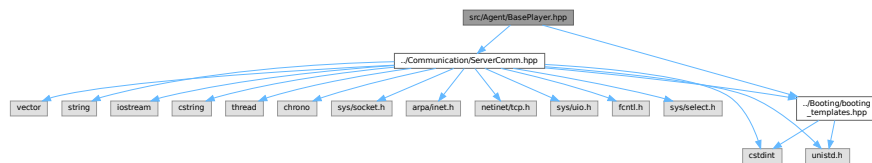
Chapter 4

File Documentation

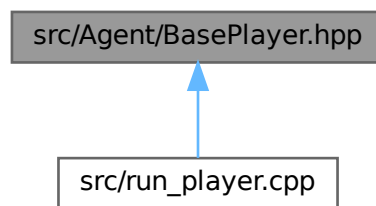
4.1 src/Agent/BasePlayer.hpp File Reference

```
#include "../Booting/booting_templates.hpp"  
#include "../Communication/ServerComm.hpp"
```

Include dependency graph for BasePlayer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [BasePlayer](#)

4.2 BasePlayer.hpp

[Go to the documentation of this file.](#)

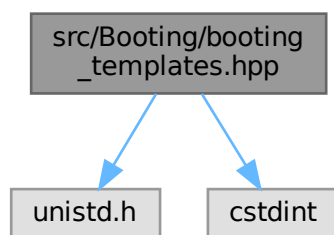
```
00001 #pragma once
00002
00003 #include "../Booting/booting_templates.hpp"
00004 #include "../Communication/ServerComm.hpp"
00005
00006 class BasePlayer {
00007 private:
00008     ServerComm __scom;
00009
00010 public:
00011     uint8_t unum;
00012
00013 public:
00014     BasePlayer(
00015         uint8_t unum
00016     ) {
00017
00019         this->unum = unum;
00020         this->__scom = ServerComm(AGENT_HOST, AGENT_PORT);
00021
00022
00023     }
00024
00025
00026
00027
00028
00029
00030
00031
00032 };
```

4.3 src/Booting/booting_templates.hpp File Reference

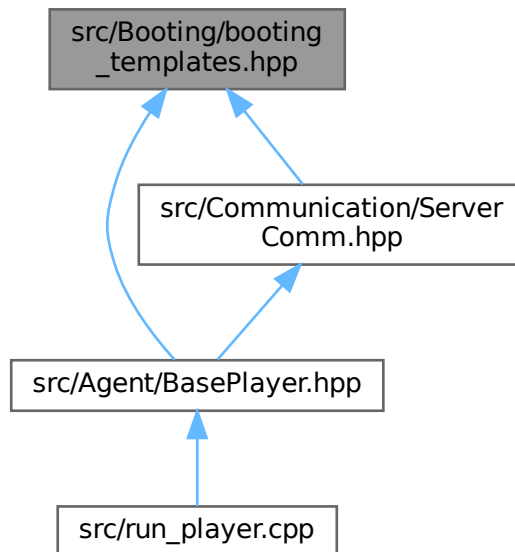
```
#include <unistd.h>
```

```
#include <cstdint>
```

Include dependency graph for booting_templates.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- `#define True true`
- `#define False false`

Variables

- `constexpr const char * AGENT_HOST = "localhost"`
- `constexpr int AGENT_PORT = 3100`
- `constexpr const char * TEAM_NAME = "RoboIME"`
- `constexpr bool DEBUG_MODE = False`

4.3.1 Macro Definition Documentation

4.3.1.1 False

```
#define False false
```

Definition at line 7 of file [booting_templates.hpp](#).

4.3.1.2 True

```
#define True true
```

Definition at line 6 of file [booting_templates.hpp](#).

4.3.2 Variable Documentation

4.3.2.1 AGENT_HOST

```
constexpr const char* AGENT_HOST = "localhost" [inline], [constexpr]
```

Definition at line 9 of file [booting_templates.hpp](#).

4.3.2.2 AGENT_PORT

```
constexpr int AGENT_PORT = 3100 [inline], [constexpr]
```

Definition at line 10 of file [booting_templates.hpp](#).

4.3.2.3 DEBUG_MODE

```
constexpr bool DEBUG_MODE = False [inline], [constexpr]
```

Definition at line 12 of file [booting_templates.hpp](#).

4.3.2.4 TEAM_NAME

```
constexpr const char* TEAM_NAME = "RoboIME" [inline], [constexpr]
```

Definition at line 11 of file [booting_templates.hpp](#).

4.4 booting_templates.hpp

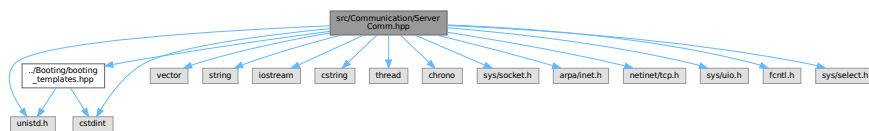
[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
00003 #include <unistd.h>
00004 #include <cstdint>
00005
00006 #define True true
00007 #define False false
00008
00009 inline constexpr const char* AGENT_HOST = "localhost";
00010 inline constexpr int AGENT_PORT = 3100;
00011 inline constexpr const char* TEAM_NAME = "RoboIME";
00012 inline constexpr bool DEBUG_MODE = False;
00013
```

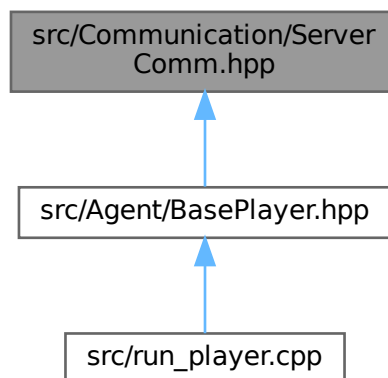
4.5 src/Communication/ServerComm.hpp File Reference

```
#include "../Booting/booting_templates.hpp"
#include <vector>
#include <string>
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <thread>
#include <chrono>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/tcp.h>
#include <unistd.h>
#include <sys/uio.h>
#include <fcntl.h>
#include <sys/select.h>
```

Include dependency graph for ServerComm.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ServerComm](#)

Responsável pela implementação da comunicação com servidor rcssserver3d.

4.6 ServerComm.hpp

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002
00003
00004 #include "../Bootimg/booting_templates.hpp"
00005
00006 #include <vector>
00007 #include <string>
00008 #include <iostream>
00009 #include <cstring>           // Manipulação de memória bruta (std::memcpy, std::memset)
00010 #include <stdint>           // Tipos de inteiros com tamanho fixo (uint32_t)
00011 #include <thread>           // (Opcional) Para sleep_for se desejar substituir usleep
00012 #include <chrono>           // (Opcional) Para unidades de tempo
00013
00014 #include <sys/socket.h> // API principal de Sockets (socket, connect, recv, send)
00015 #include <arpa/inet.h> // Conversão de endereços IP (sockaddr_in, inet_pton, htons)
00016 #include <netinet/tcp.h> // Definições específicas do protocolo TCP (TCP_NODELAY)
00017 #include <unistd.h> // Chamadas de sistema Unix padrão (close, writev, usleep)
00018 #include <sys/uio.h> // Estruturas para I/O vetorial (struct iovec para writev)
00019 #include <fcntl.h> // Controle de descritores de arquivo (bloqueante/não-bloqueante)
00020 #include <sys/select.h> // Multiplexação de I/O síncrono (select)
00021
00022 class ServerComm {
00023 private:
00024     int __sock_fd;
00025
00026     std::vector<char> __read_buffer;
00027
00028     std::string __message_queue;
00029
00030     static constexpr size_t __HEADER_SIZE = 4;
00031
00032     bool __recv_all(char* buffer, size_t len) {
00033         size_t total_read = 0;
00034
00035         while(total_read < len) {
00036             // Calcula onde escrever no buffer e quanto falta ler
00037             char* write_ptr = buffer + total_read;
00038             size_t bytes_needed = len - total_read;
00039
00040             ssize_t bytes = ::recv(this->__sock_fd, write_ptr, bytes_needed, 0);
00041
00042             if (bytes <= 0) {
00043                 return False;
00044             }
00045
00046             total_read += bytes;
00047         }
00048
00049         return True;
00050     }
00051
00052     void __set_blocking_mode(bool blocking) {
00053         int flags = fcntl(this->__sock_fd, F_GETFL, 0);
00054
00055         if (flags == -1) {
00056             return;
00057         }
00058
00059         if (blocking) {
00060             flags &= ~O_NONBLOCK;
00061         } else {
00062             flags |= O_NONBLOCK;
00063         }
00064
00065         fcntl(this->__sock_fd, F_SETFL, flags);
00066     }
00067
00068 public:
00069     ServerComm() = default;
00070
00071     ServerComm(const std::string& host, int port) {
00072         // Inicialização de variáveis membro
00073         this->__sock_fd = -1;
00074         this->__read_buffer.resize(4096); // Pré-aloca 4KB
00075         this->__message_queue.reserve(4096); // Reserva espaço
00076
00077         // 1. Criação do Socket
00078         this->__sock_fd = socket(AF_INET, SOCK_STREAM, 0);
00079
00080         if (this->__sock_fd < 0) {
00081             std::cerr << "Erro fatal: Falha ao criar socket." << std::endl;
00082             exit(1);
00083         }
00084     }
00085
00086     ~ServerComm() {
00087         if (__sock_fd != -1) {
00088             close(__sock_fd);
00089         }
00090     }
00091
00092     // Métodos públicos
00093     bool connect() {
00094         // ...
00095     }
00096
00097     bool send(const std::string& message) {
00098         // ...
00099     }
00100
00101     bool receive(std::string& message) {
00102         // ...
00103     }
00104
00105     // ...
00106
00107     // ...
00108
00109     // ...
00110
00111     // ...
00112
00113     // ...
00114
00115     // ...
00116
00117     // ...
00118
00119     // ...
00120
00121     // ...
00122
00123     // ...
00124
00125     // ...
00126
00127     // ...
00128
00129     // ...
00130
00131     // ...
00132
00133     // ...
00134
00135     // ...
00136
00137     // ...
00138
00139     // ...
00140
00141     // ...
00142
00143     // ...
00144
00145     // ...
00146
00147     // ...
00148
00149     // ...
00150
00151     // ...
00152
00153     // ...
00154
00155     // ...
00156
00157     // ...
00158
00159     // ...
00160
00161     // ...
00162
00163     // ...
00164
00165     // ...
00166
00167     // ...
00168
00169     // ...
00170
00171     // ...
00172
00173     // ...
00174
00175     // ...
00176
00177     // ...
00178
00179     // ...
00180
00181     // ...
00182
00183     // ...
00184
00185     // ...
00186
00187     // ...
00188
00189     // ...
00190
00191     // ...
00192
00193     // ...
00194
00195     // ...
00196
00197     // ...
00198
00199     // ...
00200
00201     // ...
00202
00203     // ...
00204
00205     // ...
00206
00207     // ...
00208
00209     // ...
00210
00211     // ...
00212
00213     // ...
00214
00215     // ...
00216
00217     // ...
00218
00219     // ...
00220
00221     // ...
00222
00223     // ...
00224
00225     // ...
00226
00227     // ...
00228
00229     // ...
00230
00231     // ...
00232
00233     // ...
00234
00235     // ...
00236
00237     // ...
00238
00239     // ...
00240
00241     // ...
00242
00243     // ...
00244
00245     // ...
00246
00247     // ...
00248
00249     // ...
00250
00251     // ...
00252
00253     // ...
00254
00255     // ...
00256
00257     // ...
00258
00259     // ...
00260
00261     // ...
00262
00263     // ...
00264
00265     // ...
00266
00267     // ...
00268
00269     // ...
00270
00271     // ...
00272
00273     // ...
00274
00275     // ...
00276
00277     // ...
00278
00279     // ...
00280
00281     // ...
00282
00283     // ...
00284
00285     // ...
00286
00287     // ...
00288
00289     // ...
00290
00291     // ...
00292
00293     // ...
00294
00295     // ...
00296
00297     // ...
00298
00299     // ...
00300
00301     // ...
00302
00303     // ...
00304
00305     // ...
00306
00307     // ...
00308
00309     // ...
00310
00311     // ...
00312
00313     // ...
00314
00315     // ...
00316
00317     // ...
00318
00319     // ...
00320
00321     // ...
00322
00323     // ...
00324
00325     // ...
00326
00327     // ...
00328
00329     // ...
00330
00331     // ...
00332
00333     // ...
00334
00335     // ...
00336
00337     // ...
00338
00339     // ...
00340
00341     // ...
00342
00343     // ...
00344
00345     // ...
00346
00347     // ...
00348
00349     // ...
00350
00351     // ...
00352
00353     // ...
00354
00355     // ...
00356
00357     // ...
00358
00359     // ...
00360
00361     // ...
00362
00363     // ...
00364
00365     // ...
00366
00367     // ...
00368
00369     // ...
00370
00371     // ...
00372
00373     // ...
00374
00375     // ...
00376
00377     // ...
00378
00379     // ...
00380
00381     // ...
00382
00383     // ...
00384
00385     // ...
00386
00387     // ...
00388
00389     // ...
00390
00391     // ...
00392
00393     // ...
00394
00395     // ...
00396
00397     // ...
00398
00399     // ...
00400
00401     // ...
00402
00403     // ...
00404
00405     // ...
00406
00407     // ...
00408
00409     // ...
00410
00411     // ...
00412
00413     // ...
00414
00415     // ...
00416
00417     // ...
00418
00419     // ...
00420
00421     // ...
00422
00423     // ...
00424
00425     // ...
00426
00427     // ...
00428
00429     // ...
00430
00431     // ...
00432
00433     // ...
00434
00435     // ...
00436
00437     // ...
00438
00439     // ...
00440
00441     // ...
00442
00443     // ...
00444
00445     // ...
00446
00447     // ...
00448
00449     // ...
00450
00451     // ...
00452
00453     // ...
00454
00455     // ...
00456
00457     // ...
00458
00459     // ...
00460
00461     // ...
00462
00463     // ...
00464
00465     // ...
00466
00467     // ...
00468
00469     // ...
00470
00471     // ...
00472
00473     // ...
00474
00475     // ...
00476
00477     // ...
00478
00479     // ...
00480
00481     // ...
00482
00483     // ...
00484
00485     // ...
00486
00487     // ...
00488
00489     // ...
00490
00491     // ...
00492
00493     // ...
00494
00495     // ...
00496
00497     // ...
00498
00499     // ...
00500
00501     // ...
00502
00503     // ...
00504
00505     // ...
00506
00507     // ...
00508
00509     // ...
00510
00511     // ...
00512
00513     // ...
00514
00515     // ...
00516
00517     // ...
00518
00519     // ...
00520
00521     // ...
00522
00523     // ...
00524
00525     // ...
00526
00527     // ...
00528
00529     // ...
00530
00531     // ...
00532
00533     // ...
00534
00535     // ...
00536
00537     // ...
00538
00539     // ...
00540
00541     // ...
00542
00543     // ...
00544
00545     // ...
00546
00547     // ...
00548
00549     // ...
00550
00551     // ...
00552
00553     // ...
00554
00555     // ...
00556
00557     // ...
00558
00559     // ...
00560
00561     // ...
00562
00563     // ...
00564
00565     // ...
00566
00567     // ...
00568
00569     // ...
00570
00571     // ...
00572
00573     // ...
00574
00575     // ...
00576
00577     // ...
00578
00579     // ...
00580
00581     // ...
00582
00583     // ...
00584
00585     // ...
00586
00587     // ...
00588
00589     // ...
00590
00591     // ...
00592
00593     // ...
00594
00595     // ...
00596
00597     // ...
00598
00599     // ...
00600
00601     // ...
00602
00603     // ...
00604
00605     // ...
00606
00607     // ...
00608
00609     // ...
00610
00611     // ...
00612
00613     // ...
00614
00615     // ...
00616
00617     // ...
00618
00619     // ...
00620
00621     // ...
00622
00623     // ...
00624
00625     // ...
00626
00627     // ...
00628
00629     // ...
00630
00631     // ...
00632
00633     // ...
00634
00635     // ...
00636
00637     // ...
00638
00639     // ...
00640
00641     // ...
00642
00643     // ...
00644
00645     // ...
00646
00647     // ...
00648
00649     // ...
00650
00651     // ...
00652
00653     // ...
00654
00655     // ...
00656
00657     // ...
00658
00659     // ...
00660
00661     // ...
00662
00663     // ...
00664
00665     // ...
00666
00667     // ...
00668
00669     // ...
00670
00671     // ...
00672
00673     // ...
00674
00675     // ...
00676
00677     // ...
00678
00679     // ...
00680
00681     // ...
00682
00683     // ...
00684
00685     // ...
00686
00687     // ...
00688
00689     // ...
00690
00691     // ...
00692
00693     // ...
00694
00695     // ...
00696
00697     // ...
00698
00699     // ...
00700
00701     // ...
00702
00703     // ...
00704
00705     // ...
00706
00707     // ...
00708
00709     // ...
00710
00711     // ...
00712
00713     // ...
00714
00715     // ...
00716
00717     // ...
00718
00719     // ...
00720
00721     // ...
00722
00723     // ...
00724
00725     // ...
00726
00727     // ...
00728
00729     // ...
00730
00731     // ...
00732
00733     // ...
00734
00735     // ...
00736
00737     // ...
00738
00739     // ...
00740
00741     // ...
00742
00743     // ...
00744
00745     // ...
00746
00747     // ...
00748
00749     // ...
00750
00751     // ...
00752
00753     // ...
00754
00755     // ...
00756
00757     // ...
00758
00759     // ...
00760
00761     // ...
00762
00763     // ...
00764
00765     // ...
00766
00767     // ...
00768
00769     // ...
00770
00771     // ...
00772
00773     // ...
00774
00775     // ...
00776
00777     // ...
00778
00779     // ...
00780
00781     // ...
00782
00783     // ...
00784
00785     // ...
00786
00787     // ...
00788
00789     // ...
00790
00791     // ...
00792
00793     // ...
00794
00795     // ...
00796
00797     // ...
00798
00799     // ...
00800
00801     // ...
00802
00803     // ...
00804
00805     // ...
00806
00807     // ...
00808
00809     // ...
00810
00811     // ...
00812
00813     // ...
00814
00815     // ...
00816
00817     // ...
00818
00819     // ...
00820
00821     // ...
00822
00823     // ...
00824
00825     // ...
00826
00827     // ...
00828
00829     // ...
00830
00831     // ...
00832
00833     // ...
00834
00835     // ...
00836
00837     // ...
00838
00839     // ...
00840
00841     // ...
00842
00843     // ...
00844
00845     // ...
00846
00847     // ...
00848
00849     // ...
00850
00851     // ...
00852
00853     // ...
00854
00855     // ...
00856
00857     // ...
00858
00859     // ...
00860
00861     // ...
00862
00863     // ...
00864
00865     // ...
00866
00867     // ...
00868
00869     // ...
00870
00871     // ...
00872
00873     // ...
00874
00875     // ...
00876
00877     // ...
00878
00879     // ...
00880
00881     // ...
00882
00883     // ...
00884
00885     // ...
00886
00887     // ...
00888
00889     // ...
00890
00891     // ...
00892
00893     // ...
00894
00895     // ...
00896
00897     // ...
00898
00899     // ...
00900
00901     // ...
00902
00903     // ...
00904
00905     // ...
00906
00907     // ...
00908
00909     // ...
00910
00911     // ...
00912
00913     // ...
00914
00915     // ...
00916
00917     // ...
00918
00919     // ...
00920
00921     // ...
00922
00923     // ...
00924
00925     // ...
00926
00927     // ...
00928
00929     // ...
00930
00931     // ...
00932
00933     // ...
00934
00935     // ...
00936
00937     // ...
00938
00939     // ...
00940
00941     // ...
00942
00943     // ...
00944
00945     // ...
00946
00947     // ...
00948
00949     // ...
00950
00951     // ...
00952
00953     // ...
00954
00955     // ...
00956
00957     // ...
00958
00959     // ...
00960
00961     // ...
00962
00963     // ...
00964
00965     // ...
00966
00967     // ...
00968
00969     // ...
00970
00971     // ...
00972
00973     // ...
00974
00975     // ...
00976
00977     // ...
00978
00979     // ...
00980
00981     // ...
00982
00983     // ...
00984
00985     // ...
00986
00987     // ...
00988
00989     // ...
00990
00991     // ...
00992
00993     // ...
00994
00995     // ...
00996
00997     // ...
00998
00999     // ...
10000     // ...

```



```

00105
00106 // 2. Otimização de Performance: TCP_NODELAY
00107 int flag = 1;
00108 int result_opt = setsockopt(
00109     this->__sock_fd,
00110     IPPROTO_TCP,
00111     TCP_NODELAY,
00112     (char*)&flag,
00113     sizeof(int)
00114 );
00115
00116 if (result_opt < 0) {
00117     std::cerr << "Aviso: Falha ao definir TCP_NODELAY." << std::endl;
00118 }
00119
00120 // 3. Configuração do Endereço
00121 struct sockaddr_in serv_addr;
00122 std::memset(&serv_addr, 0, sizeof(serv_addr));
00123
00124 serv_addr.sin_family = AF_INET;
00125 serv_addr.sin_port = htons(port);
00126 inet_pton(AF_INET, host.c_str(), &serv_addr.sin_addr);
00127
00128 // 4. Tentativa de Conexão (Verticalizado)
00129 while (true) {
00130     int connection_result = connect(
00131         this->__sock_fd,
00132         (struct sockaddr*)&serv_addr,
00133         sizeof(serv_addr)
00134     );
00135
00136     if (connection_result == 0) {
00137         // Conexão bem sucedida
00138         break;
00139     }
00140
00141     // Em produção, usar usleep para não travar CPU
00142     usleep(500000); // 0.5s
00143 }
00144
00145 printf("Consegui");
00146 }
00147
00151 ~ServerComm() {
00152     if (this->__sock_fd >= 0) {
00153         close(this->__sock_fd);
00154     }
00155 }
00156
00162 void send_immediate(const std::string& msg) {
00163     if (msg.empty()) {
00164         return;
00165     }
00166
00167     // Prepara o tamanho em Network Byte Order (Big Endian)
00168     uint32_t msg_len = static_cast<uint32_t>(msg.size());
00169     uint32_t net_len = htonl(msg_len);
00170
00171     // Estrutura para envio vetorial (Zero-Copy concatenation)
00172     struct iovec iov[2];
00173
00174     // Parte 1: Cabeçalho (4 bytes)
00175     iov[0].iov_base = &net_len;
00176     iov[0].iov_len = sizeof(net_len);
00177
00178     // Parte 2: Corpo da mensagem
00179     iov[1].iov_base = (void*)msg.data();
00180     iov[1].iov_len = msg_len;
00181
00182     // Envia ambos os buffers atomicamente
00183     ssize_t bytes_written = writev(this->__sock_fd, iov, 2);
00184
00185     if (bytes_written < 0) {
00186         std::cerr << "Erro ao enviar dados via writev." << std::endl;
00187     }
00188 }
00189
00194 std::string receive() {
00195     // Verifica se há dados disponíveis (non-blocking check via select)
00196     fd_set readfds;
00197     FD_ZERO(&readfds);
00198     FD_SET(this->__sock_fd, &readfds);
00199
00200     // Timeout zero = retorno imediato se não houver dados
00201     struct timeval tv = {0, 0};
00202
00203     int activity = select(

```

```

00204         this->__sock_fd + 1,
00205         &readfds,
00206         NULL,
00207         NULL,
00208         &tv
00209     );
00210
00211     if (activity <= 0) {
00212         return ""; // Sem dados
00213     }
00214
00215     // 1. Ler Cabeçalho (4 bytes)
00216     char header[this->__HEADER_SIZE];
00217     bool header_ok = this->__recv_all(header, this->__HEADER_SIZE);
00218
00219     if (!header_ok) {
00220         return ""; // Erro ou desconexão
00221     }
00222
00223     // 2. Converter tamanho
00224     uint32_t net_len;
00225     std::memcpy(&net_len, header, this->__HEADER_SIZE);
00226     uint32_t msg_len = ntohs(net_len);
00227
00228     // 3. Ajustar buffer se necessário (evita alocação se msg for pequena)
00229     if (this->__read_buffer.size() < msg_len) {
00230         this->__read_buffer.resize(msg_len);
00231     }
00232
00233     // 4. Ler Corpo
00234     bool body_ok = this->__recv_all(this->__read_buffer.data(), msg_len);
00235
00236     if (!body_ok) {
00237         return "";
00238     }
00239
00240     return std::string(this->__read_buffer.data(), msg_len);
00241 }
00242
00243 void receive_async_sync(std::vector<ServerComm*>& other_players) {
00244     if (other_players.empty()) {
00245         this->receive();
00246         return;
00247     }
00248
00249     this->__set_blocking_mode(false); // Ativa modo não-bloqueante
00250
00251     while (true) {
00252         // Tenta 'espiar' o socket para ver se tem dados
00253         char peek_buf;
00254         ssize_t ret = ::recv(this->__sock_fd, &peek_buf, 1, MSG_PEEK);
00255
00256         if (ret > 0) {
00257             // Dados encontrados! Volta para bloqueante e lê normal
00258             this->__set_blocking_mode(true);
00259             this->receive();
00260             break;
00261         }
00262         else if (ret == -1 && (errno == EAGAIN || errno == EWOULDBLOCK)) {
00263             // Socket vazio: manda (syn) para os outros agentes para mantê-los vivos
00264             for (auto* scom : other_players) {
00265                 scom->send_immediate("(syn)");
00266             }
00267
00268             for (auto* scom : other_players) {
00269                 scom->receive(); // Drena o buffer dos outros
00270             }
00271
00272             usleep(10000); // 10ms sleep
00273         }
00274         else {
00275             // Erro real
00276             break;
00277         }
00278     }
00279
00280     this->__set_blocking_mode(true); // Restaura
00281 }
00282
00283 void commit(const std::string& msg) {
00284     this->__message_queue += msg;
00285 }
00286
00287 void send() {
00288     if (this->__message_queue.empty()) {
00289         // Se vazio, apenas envia sync
00290         this->send_immediate("(syn)");
00291     }
00292 }

```

```

00303         return;
00304     }
00305
00306     // Adiciona syn
00307     this->__message_queue += "(syn) ";
00308
00309     // Envia tudo de uma vez
00310     this->send_immediate(this->__message_queue);
00311
00312     // Limpa buffer mantendo capacidade reservada (rápido)
00313     this->__message_queue.clear();
00314 }
00315
00319 void clear_queue() {
00320     this->__message_queue.clear();
00321 }
00322 };

```

4.7 src/run_full_team.cpp File Reference

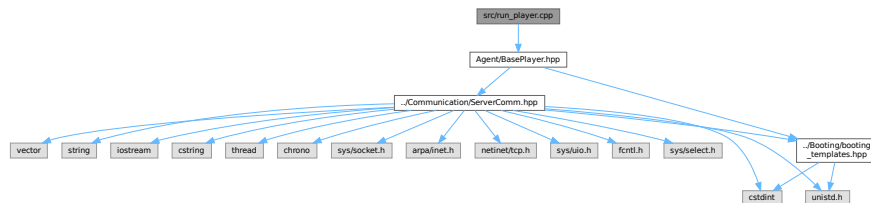
4.8 run_full_team.cpp

[Go to the documentation of this file.](#)

4.9 src/run_player.cpp File Reference

```
#include "Agent/BasePlayer.hpp"
```

Include dependency graph for run_player.cpp:



Functions

- int [main](#) ()

4.9.1 Function Documentation

4.9.1.1 main()

```
int main ( )
```

Definition at line 3 of file [run_player.cpp](#).

4.10 run_player.cpp

[Go to the documentation of this file.](#)

```
00001 #include "Agent/BasePlayer.hpp"
00002
00003 int main() {
00004
00005     BasePlayer p = BasePlayer(1);
00006
00007
00008
00009     return 0;
00010 }
```

Index

- __HEADER_SIZE
 - ServerComm, [12](#)
 - __message_queue
 - ServerComm, [12](#)
 - __read_buffer
 - ServerComm, [12](#)
 - __recv_all
 - ServerComm, [10](#)
 - __scom
 - BasePlayer, [7](#)
 - __set_blocking_mode
 - ServerComm, [10](#)
 - __sock_fd
 - ServerComm, [12](#)
 - ~ServerComm
 - ServerComm, [10](#)
- AGENT_HOST
 - booting_templates.hpp, [18](#)
- AGENT_PORT
 - booting_templates.hpp, [18](#)
- BasePlayer, [5](#)
 - __scom, [7](#)
 - BasePlayer, [7](#)
 - unum, [7](#)
- booting_templates.hpp
 - AGENT_HOST, [18](#)
 - AGENT_PORT, [18](#)
 - DEBUG_MODE, [18](#)
 - False, [17](#)
 - TEAM_NAME, [18](#)
 - True, [17](#)
- clear_queue
 - ServerComm, [11](#)
- commit
 - ServerComm, [11](#)
- DEBUG_MODE
 - booting_templates.hpp, [18](#)
- False
 - booting_templates.hpp, [17](#)
- main
 - run_player.cpp, [23](#)
- receive
 - ServerComm, [11](#)
- receive_async_sync
- ServerComm, [11](#)
- run_player.cpp
 - main, [23](#)
- send
 - ServerComm, [11](#)
- send_immediate
 - ServerComm, [12](#)
- ServerComm, [8](#)
 - __HEADER_SIZE, [12](#)
 - __message_queue, [12](#)
 - __read_buffer, [12](#)
 - __recv_all, [10](#)
 - __set_blocking_mode, [10](#)
 - __sock_fd, [12](#)
 - ~ServerComm, [10](#)
 - clear_queue, [11](#)
 - commit, [11](#)
 - receive, [11](#)
 - receive_async_sync, [11](#)
 - send, [11](#)
 - send_immediate, [12](#)
 - ServerComm, [9](#)
- src/Agent/BasePlayer.hpp, [15](#), [16](#)
- src/Booting/booting_templates.hpp, [16](#), [18](#)
- src/Communication/ServerComm.hpp, [19](#), [20](#)
- src/run_full_team.cpp, [23](#)
- src/run_player.cpp, [23](#), [24](#)
- TEAM_NAME
 - booting_templates.hpp, [18](#)
- True
 - booting_templates.hpp, [17](#)
- unum
 - BasePlayer, [7](#)