

**Description of an IPv6 Linux-Based UTRAN Testbed Reference Manual**  
8.1(final)

Generated by Doxygen 1.4.6

Wed Nov 28 13:59:02 2007



# Contents

<b>1</b>	<b>Description of an IPv6 Linux-Based UTRAN Testbed Directory Hierarchy</b>	<b>1</b>
1.1	Description of an IPv6 Linux-Based UTRAN Testbed Directories . . . . .	1
<b>2</b>	<b>Description of an IPv6 Linux-Based UTRAN Testbed File Index</b>	<b>3</b>
2.1	Description of an IPv6 Linux-Based UTRAN Testbed File List . . . . .	3
<b>3</b>	<b>Description of an IPv6 Linux-Based UTRAN Testbed Directory Documentation</b>	<b>5</b>
3.1	NODEB/ Directory Reference . . . . .	5
3.2	RNC/ Directory Reference . . . . .	6
3.3	UE/ Directory Reference . . . . .	7
<b>4</b>	<b>Description of an IPv6 Linux-Based UTRAN Testbed File Documentation</b>	<b>9</b>
4.1	NODEB/cell_throughput.c File Reference . . . . .	9
4.2	NODEB/qos.c File Reference . . . . .	11
4.3	RNC/background.c File Reference . . . . .	16
4.4	UE/background.c File Reference . . . . .	24
4.5	RNC/cac.c File Reference . . . . .	25
4.6	RNC/channelshift.c File Reference . . . . .	32
4.7	NODEB/channelshift.c File Reference . . . . .	34
4.8	RNC/clock.c File Reference . . . . .	39
4.9	RNC/conversational.c File Reference . . . . .	42
4.10	UE/conversational.c File Reference . . . . .	49
4.11	RNC/finalisation.c File Reference . . . . .	50
4.12	RNC/handover.c File Reference . . . . .	51
4.13	RNC/initialisation.c File Reference . . . . .	58
4.14	RNC/interactive.c File Reference . . . . .	61
4.15	RNC/interfaces.c File Reference . . . . .	67
4.16	UE/interfaces.c File Reference . . . . .	69
4.17	RNC/main.c File Reference . . . . .	70

---

4.18 UE/main.c File Reference . . . . .	72
4.19 NODEB/main.c File Reference . . . . .	75
4.20 RNC/mobility.c File Reference . . . . .	78
4.21 RNC/network.c File Reference . . . . .	85
4.22 UE/network.c File Reference . . . . .	89
4.23 NODEB/network.c File Reference . . . . .	92
4.24 RNC/stat.c File Reference . . . . .	94
4.25 UE/stat.c File Reference . . . . .	98
4.26 RNC/streaming.c File Reference . . . . .	102
4.27 RNC/utills.c File Reference . . . . .	107
4.28 UE/utills.c File Reference . . . . .	113
4.29 RNC/wireless.c File Reference . . . . .	116

# Chapter 1

## Description of an IPv6 Linux-Based UTRAN Testbed Directory Hierarchy

### 1.1 Description of an IPv6 Linux-Based UTRAN Testbed Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

NODEB . . . . .	5
RNC . . . . .	6
UE . . . . .	7



## Chapter 2

# Description of an IPv6 Linux-Based UTRAN Testbed File Index

### 2.1 Description of an IPv6 Linux-Based UTRAN Testbed File List

Here is a list of all files with brief descriptions:

NODEB/cell_throughput.c	9
NODEB/channelshift.c	34
NODEB/main.c	75
NODEB/network.c	92
NODEB/qos.c	11
RNC/background.c	16
RNC/cac.c	25
RNC/channelshift.c	32
RNC/clock.c	39
RNC/conversational.c	42
RNC/finalisation.c	50
RNC/handover.c	51
RNC/initialisation.c	58
RNC/interactive.c	61
RNC/interfaces.c	67
RNC/main.c	70
RNC/mobility.c	78
RNC/network.c	85
RNC/stat.c	94
RNC/streaming.c	102
RNC/utls.c	107
RNC/wireless.c	116
UE/background.c	24
UE/conversational.c	49
UE/interfaces.c	69
UE/main.c	72
UE/network.c	89
UE/stat.c	98
UE/utls.c	113



## Chapter 3

# Description of an IPv6 Linux-Based UTRAN Testbed Directory Documentation

### 3.1 NODEB/ Directory Reference

#### Files

- file [cell\\_throughput.c](#)
- file [channelshift.c](#)
- file [main.c](#)
- file [network.c](#)
- file [qos.c](#)

## 3.2 RNC/ Directory Reference

### Files

- file [background.c](#)
- file [cac.c](#)
- file [channelshift.c](#)
- file [clock.c](#)
- file [conversational.c](#)
- file [finalisation.c](#)
- file [handover.c](#)
- file [initialisation.c](#)
- file [interactive.c](#)
- file [interfaces.c](#)
- file [main.c](#)
- file [mobility.c](#)
- file [network.c](#)
- file [stat.c](#)
- file [streaming.c](#)
- file [utils.c](#)
- file [wireless.c](#)

## 3.3 UE/ Directory Reference

### Files

- file [background.c](#)
- file [conversational.c](#)
- file [interfaces.c](#)
- file [main.c](#)
- file [network.c](#)
- file [stat.c](#)
- file [utils.c](#)

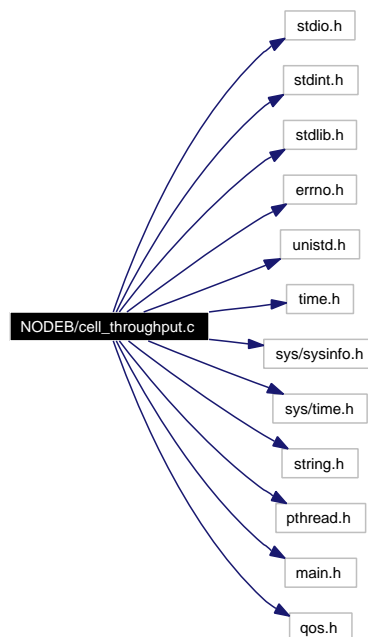


## Chapter 4

# Description of an IPv6 Linux-Based UTRAN Testbed File Documentation

### 4.1 NODEB/cell\_throughput.c File Reference

Include dependency graph for cell\_throughput.c:



### Functions

- void \* [cell\\_throughput\\_management](#) ()

## 4.1.1 Function Documentation

### 4.1.1.1 void\* cell\_throughput\_management ()

This thread computes each second the cell throughput reached during this second.

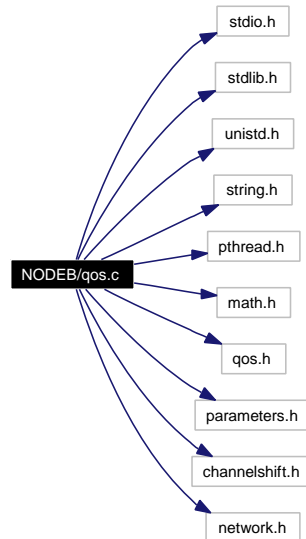
Definition at line 34 of file cell\_throughput.c.

References dch\_bandwidth, dsch\_bandwidth, and simulation\_end.

Referenced by main().

## 4.2 NODEB/qos.c File Reference

Include dependency graph for qos.c:



### Functions

- void [qos\\_init](#) ()
- void [qos\\_dsch\\_bandwidth\\_update](#) (int sf, char se, int begin)
- void [qos\\_begin](#) (char \*ip, int up, int cu, int dt, int tc, int sf, char se, int bler)
- void [qos\\_update](#) (char \*ip, int cu, int tc, char se, int bler)
- void [qos\\_stop](#) (char \*ip, int cu, int tc, char se, int ha)
- void [qos\\_pre\\_emptive\\_shifting](#) (char \*ip, int tc, char se, int pre)
- void [qos\\_fin](#) ()

### Variables

- [htb\\_t fach](#) [3][400]
- [pthread\\_mutex\\_t fach\\_access](#)
- [htb\\_t dch](#) [3][400]
- [pthread\\_mutex\\_t dch\\_access](#)
- [htb\\_t dsch](#) [3][400]
- [pthread\\_mutex\\_t dsch\\_access](#)
- int [dsch\\_bandwidth](#) [3]
- int [dch\\_bandwidth](#) [3]

### 4.2.1 Function Documentation

#### 4.2.1.1 void qos\_init ()

This procedure initialises the QoS management on the NodeBs. It creates the tc root qdisc and prepares the DCH and FACH arrays.

Definition at line 68 of file qos.c.

References dch, dch\_bandwidth, dsch, dsch\_bandwidth, and fach.

Referenced by main().

#### 4.2.1.2 void qos\_dsch\_bandwidth\_update (int sf, char se, int begin)

This procedure adapts the bandwidth available in a DSCH based on the newly allocated/removed SF.

##### Parameters:

*sf* represents the spreading factor allocated/removed to/from a DCH,

*se* represents the sector where the DCH is located, and

*begin* if begin equal to 1, the SF is allocated, and if not, the SF is removed.

Definition at line 136 of file qos.c.

References dch\_bandwidth, and dsch\_bandwidth.

Referenced by qos\_begin(), and qos\_stop().

#### 4.2.1.3 void qos\_begin (char \* ip, int up, int cu, int dt, int tc, int sf, char se, int bler)

This procedure creates a tc "class/qdisc/filter" triple following the parameters given.

##### Parameters:

*ip* represents the source IPv6 address of the traffic to filter,

*up* represents the user profile of the UE (1: Platinum, 2: Gold, 3: Silver and 4: Bronze),

*cu* represents the channel which will be used to transport the traffic (1: DCH, 2: FACH and 3: DSCH),

*dt* represents the value to which the DCH holding inactivity timer as to be initialised,

*tc* represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),

*sf* represents the spreading factor allocated to the UE,

*se* represents the sector where the UE is located, and

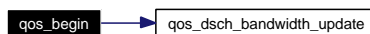
*bler* represents the BLER the transport channel undergoes.

Definition at line 169 of file qos.c.

References dch, dch\_access, dsch, dsch\_access, dsch\_bandwidth, fach, fach\_access, and qos\_dsch\_bandwidth\_update().

Referenced by action\_analysis().

Here is the call graph for this function:



**4.2.1.4 void qos\_update (char \* ip, int cu, int tc, char se, int bler)**

This procedure adapts the BLER of the FACH and the DSCH.

**Parameters:**

- ip* represents the source IPv6 address of the traffic to filter,
- cu* represents the channel which will be used to transport the traffic (1: DCH, 2: FACH and 3: DSCH),
- tc* represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),
- se* represents the sector where the UE is located, and
- bler* represents the BLER the transport channel undergoes.

Definition at line 280 of file qos.c.

References dsch, dsch\_bandwidth, and fach.

Referenced by action\_analysis().

**4.2.1.5 void qos\_stop (char \* ip, int cu, int tc, char se, int ha)**

This procedure deletes a tc "class-qdisc-filter" triple following the parameters given.

**Parameters:**

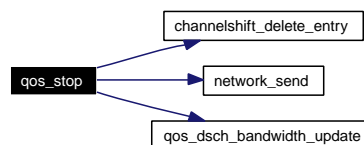
- ip* represents the source IPv6 address of the traffic to filter,
- cu* represents the channel which will be used to transport the traffic (1: DCH, 2: FACH and 3: DSCH),
- tc* represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),
- se* represents the sector where the UE is located, and
- ha* represents the fact that the destruction of the tc class and filter is due to an handover (1) or not (0).

Definition at line 324 of file qos.c.

References channelshift\_delete\_entry(), dch, dch\_access, dsch, dsch\_access, fach, fach\_access, network\_send(), and qos\_dsch\_bandwidth\_update().

Referenced by action\_analysis().

Here is the call graph for this function:

**4.2.1.6 void qos\_pre\_emptive\_shifting (char \* ip, int tc, char se, int pre)**

This procedure moves a UE from non pre-emptive to pre-emptive mode (or the opposite) still staying on a DCH.

**Parameters:**

- ip* represents the source IPv6 address of the traffic to filter,
- tc* represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),
- se* represents the sector where the UE is located, and
- pre* 1 : non pre-emptive -> pre-emptive, 0: pre-emptive -> non pre-emptive.

Definition at line 424 of file qos.c.

References dch, and dch\_access.

Referenced by action\_analysis().

**4.2.1.7 void qos\_fin ()**

This procedure finalises the Qos management on the NodeBs by deleting the tc root qdisc.

Definition at line 452 of file qos.c.

References chchange\_array\_access, dch\_access, dsch\_access, and fach\_access.

Referenced by main().

**4.2.2 Variable Documentation****4.2.2.1 htb\_t fach[3][400]**

This matrix represents, for each sector of the NodeB, the occupancy of the FACH channel.

Definition at line 33 of file qos.c.

Referenced by channelchift\_fach\_data\_update(), qos\_begin(), qos\_init(), qos\_stop(), and qos\_update().

**4.2.2.2 pthread\_mutex\_t fach\_access**

This mutex avoids concurrent accesses on the FACH array.

Definition at line 37 of file qos.c.

Referenced by channelchift\_fach\_data\_update(), qos\_begin(), qos\_fin(), and qos\_stop().

**4.2.2.3 htb\_t dch[3][400]**

This matrix represents, for each sector of the NodeB, the occupancy of the DCH channels.

Definition at line 41 of file qos.c.

Referenced by channelchift\_dch\_data\_update(), qos\_begin(), qos\_init(), qos\_pre\_emptive\_shifting(), and qos\_stop().

**4.2.2.4 pthread\_mutex\_t dch\_access**

This mutex avoids concurrent accesses on the DCH array.

Definition at line 45 of file qos.c.

Referenced by `channelchift_dch_data_update()`, `qos_begin()`, `qos_fin()`, `qos_pre_emptive_shifting()`, and `qos_stop()`.

#### 4.2.2.5 `htb_t dsch[3][400]`

This matrix represents, for each sector of the NodeB, the occupancy of the DSCH channel.

Definition at line 49 of file `qos.c`.

Referenced by `channelchift_dsch_data_update()`, `qos_begin()`, `qos_init()`, `qos_stop()`, and `qos_update()`.

#### 4.2.2.6 `pthread_mutex_t dsch_access`

This mutex avoids concurrent accesses on the DSCH array.

Definition at line 53 of file `qos.c`.

Referenced by `channelchift_dsch_data_update()`, `qos_begin()`, `qos_fin()`, and `qos_stop()`.

#### 4.2.2.7 `int dsch_bandwidth[3]`

This array represents the bandwidth allocated at any time to the 3 DSCHs of the NodeB.

Definition at line 57 of file `qos.c`.

Referenced by `cell_throughput_management()`, `qos_begin()`, `qos_dsch_bandwidth_update()`, `qos_init()`, and `qos_update()`.

#### 4.2.2.8 `int dch_bandwidth[3]`

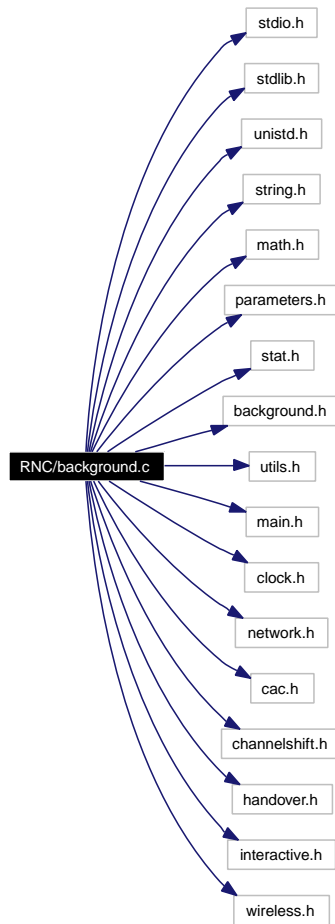
This array represents the bandwidth allocated at any time to the DCHs of the NodeB.

Definition at line 61 of file `qos.c`.

Referenced by `cell_throughput_management()`, `qos_dsch_bandwidth_update()`, and `qos_init()`.

### 4.3 RNC/background.c File Reference

Include dependency graph for background.c:



#### Functions

- void [background\\_maj\\_input\\_file](#) (int ue\_id)
- int [background\\_rnc\\_to\\_ue](#) (network\_info\_t network\_info, int ue\_id)
- void [background\\_ue\\_to\\_rnc](#) (network\_info\_t network\_info, int ue\_id)
- void [background\\_ue\\_to\\_rnc\\_launch](#) (network\_info\_t network\_info, int ue\_id)
- void [background\\_nodeb\\_info](#) (int ue\_id, int start)
- void [background\\_activity\\_start](#) (int ue\_id)
- int [background\\_cac\\_select](#) (int ue\_id)
- void [background\\_Cell\\_FACH\\_Cell\\_DCH](#) (int ue\_id)
- void [background\\_Cell\\_DCH\\_Cell\\_FACH](#) (int ue\_id)
- void [background\\_downswitch\\_user](#) (int ue\_id, int forced)
- void [background\\_channelshift](#) (int ue\_id)
- void [background\\_management](#) (int ue\_id)

### 4.3.1 Function Documentation

#### 4.3.1.1 void background\_maj\_input\_file (int ue\_id)

This procedure updates the TG input file due to a channel switching (DCH <-> FACH).

**Parameters:**

*ue\_id* represents the identification number of the UE.

Definition at line 42 of file background.c.

References actual\_time, ue\_array, and utils\_get\_ue\_port().

Referenced by background\_channelshift(), background\_downswitch\_user(), conversational\_Cell\_-DCH\_Cell\_FACH(), conversational\_Cell\_FACH\_Cell\_DCH(), interactive\_Cell\_DCH\_Cell\_FACH(), interactive\_Cell\_FACH\_Cell\_DCH(), streaming\_Cell\_DCH\_Cell\_FACH(), and streaming\_Cell\_FACH\_-Cell\_DCH().

Here is the call graph for this function:



#### 4.3.1.2 int background\_rnc\_to\_ue (network\_info\_t network\_info, int ue\_id)

This procedure takes care of the background downlink part of the emulation, computing session and inter-session length, connection behaviours, etc. based on statistical distributions.

**Parameters:**

*network\_info* represents a structure containing all the network information the procedure needs (source/destination IP, source/destination port, etc.) to compute well,

*ue\_id* represents the identification number of the UE.

**Returns:**

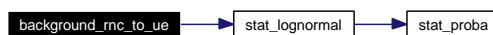
the number of connection the background session will have.

Definition at line 125 of file background.c.

References stat\_lognormal(), and ue\_array.

Referenced by background\_activity\_start().

Here is the call graph for this function:



#### 4.3.1.3 void background\_ue\_to\_rnc (network\_info\_t network\_info, int ue\_id)

This procedure sends queries to the UEs to create the tg input file of the uplink part of a background flow.

**Parameters:**

*network\_info* represents a structure containing all the network information the procedure needs (source/destination IP, source/destination port, session length, etc.) to compute well,

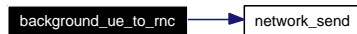
*ue\_id* represents the identification number of the UE.

Definition at line 194 of file background.c.

References network\_send(), and ue\_array.

Referenced by action\_analysis(), and background\_activity\_start().

Here is the call graph for this function:

**4.3.1.4 void background\_ue\_to\_rnc\_launch (network\_info\_t network\_info, int ue\_id)**

This procedure sends queries to the UEs to start the uplink part of a background flow.

**Parameters:**

*network\_info* represents a structure containing all the network information the procedure needs (source/destination IP, source/destination port, session length, etc.) to compute well,

*ue\_id* represents the identification number of the UE.

Definition at line 211 of file background.c.

References network\_send(), and ue\_array.

Referenced by background\_activity\_start().

Here is the call graph for this function:

**4.3.1.5 void background\_nodeb\_info (int ue\_id, int start)**

This procedure alerts the NodeBs about the flow creation/destruction. The messages may have various formats:

- "b ip up cu dt tc sf se bler": message corresponding to the creation of a new tc class and filter:
  1. ip represents the source IPv6 address of the traffic to filter,
  2. up represents the user profile of the source UE of the traffic (1: Platinum, 2: Gold, 3: Silver and 4: Bronze),
  3. cu represents the channel used to transport the traffic (1: DCH, 2: DSCH and 3: FACH),
  4. dt represents the value to which the DCH holding inactivity timer as to be initialised,
  5. tc represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),
  6. sf represents the spreading factor allocated to the source UE of the traffic,

7. *se* represents the sector of the NodeB where the UE is located, and
  8. *bler* represents the BLER the transport channel undergoes.
- "s ip cu tc se ha": message corresponding to the destruction of a tc class and filter:
    1. *ip* represents the source IPv6 address of the traffic to filter,
    2. *cu* represents the channel used to transport the traffic (1: DCH, 2: DSCH and 3: FACH),
    3. *tc* represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),
    4. *se* represents the sector of the NodeB where the UE is located, and
    5. *ha* represents the fact that the destruction of the tc class and filter is due to an handover (1) or not (0).

**Parameters:**

*ue\_id* represents the identification number of the UE,

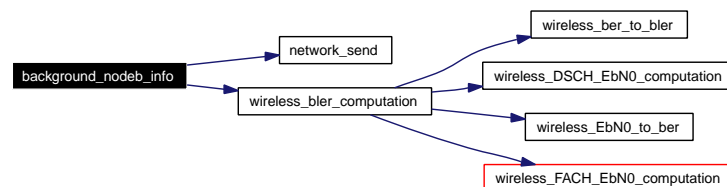
*start* indicates that we are starting (1) or stopping (0) an activity.

Definition at line 244 of file background.c.

References `network_send()`, `ue_array`, and `wireless_bler_computation()`.

Referenced by `background_activity_start()`, `background_channelshift()`, `background_downswitch_user()`, `background_management()`, `conversational_Cell_DCH_Cell_FACH()`, `conversational_Cell_FACH_Cell_DCH()`, `interactive_Cell_DCH_Cell_FACH()`, `interactive_Cell_FACH_Cell_DCH()`, `streaming_Cell_DCH_Cell_FACH()`, and `streaming_Cell_FACH_Cell_DCH()`.

Here is the call graph for this function:

**4.3.1.6 void background\_activity\_start (int ue\_id)**

This procedure starts the background stream to both uplink and downlink directions.

**Parameters:**

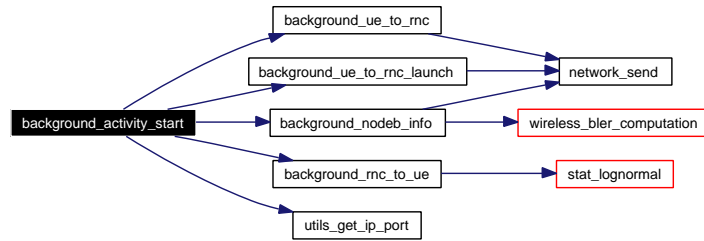
*ue\_id* represents the identification number of the UE.

Definition at line 265 of file background.c.

References `actual_time`, `background_nodeb_info()`, `background_rnc_to_ue()`, `background_ue_to_rnc()`, `background_ue_to_rnc_launch()`, `end_time`, `init_desynchro`, `start_time`, `ue_array`, and `utils_get_ip_port()`.

Referenced by `background_management()`.

Here is the call graph for this function:



#### 4.3.1.7 int background\_cac\_select (int ue\_id)

This procedure tries to allocate an acceptable spreading factor (based to the user profile of the UE) to a background session.

**Parameters:**

*ue\_id* represents the identification number of the UE.

**Returns:**

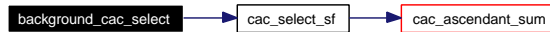
1 if the UE gets an acceptable spreading factor and 0 otherwise.

Definition at line 307 of file background.c.

References cac\_select\_sf(), and ue\_array.

Referenced by background\_channelshift(), background\_management(), and handover\_cac().

Here is the call graph for this function:



#### 4.3.1.8 void background\_Cell\_FACH\_Cell\_DCH (int ue\_id)

This procedure upswitches only the interactive sessions.

**Parameters:**

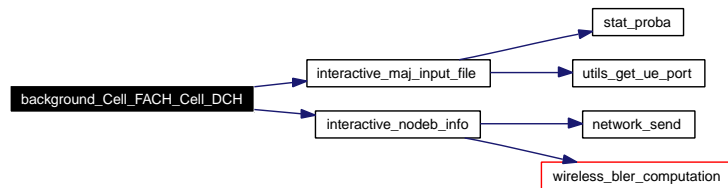
*ue\_id* represents the identification number of the UE.

Definition at line 403 of file background.c.

References actual\_time, handover\_log\_file, init\_desynchro, interactive\_maj\_input\_file(), interactive\_nodeb\_info(), start\_time, and ue\_array.

Referenced by background\_channelshift(), and background\_management().

Here is the call graph for this function:



#### 4.3.1.9 void background\_Cell\_DCH\_Cell\_FACH (int ue\_id)

This procedure downswitches only the interactive sessions.

##### Parameters:

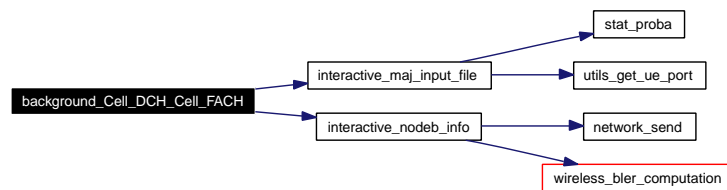
*ue\_id* represents the identification number of the UE.

Definition at line 439 of file background.c.

References `actual_time`, `handover_log_file`, `init_desynchro`, `interactive_maj_input_file()`, `interactive_nodeb_info()`, `start_time`, and `ue_array`.

Referenced by `background_downswitch_user()`, and `background_management()`.

Here is the call graph for this function:



#### 4.3.1.10 void background\_downswitch\_user (int ue\_id, int forced)

This procedure downswitches a UE from Cell\_DCHp to Cell\_FACH.

##### Parameters:

*ue\_id* represents the identification number of the UE, and

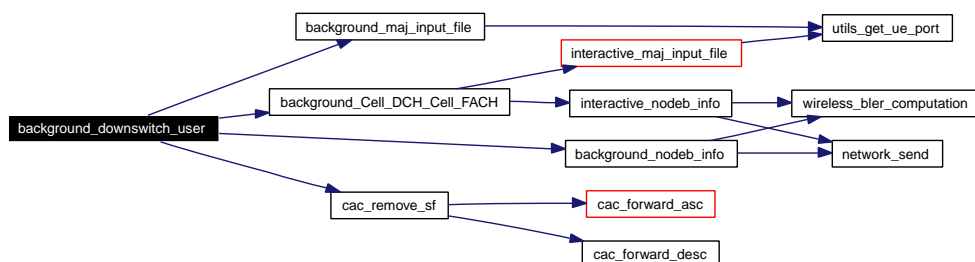
*forced* 1 if the downswitched is forced (the Ue was in pre-emptive mode and has to downswitch before the end of the pre-emptive timer) and 2 if dropped (not enough transmission power).

Definition at line 476 of file background.c.

References `actual_time`, `background_Cell_DCH_Cell_FACH()`, `background_maj_input_file()`, `background_nodeb_info()`, `cac_remove_sf()`, `handover_log_file`, `init_desynchro`, `start_time`, and `ue_array`.

Referenced by `background_channelshift()`, and `cac_downswitch_pre_emptive_users()`.

Here is the call graph for this function:



#### 4.3.1.11 void background\_channelshift (int ue\_id)

When a NodeB warns that Ue has to up/downswitch (DCH <-> FACH), this procedure is called to do so.

**Parameters:**

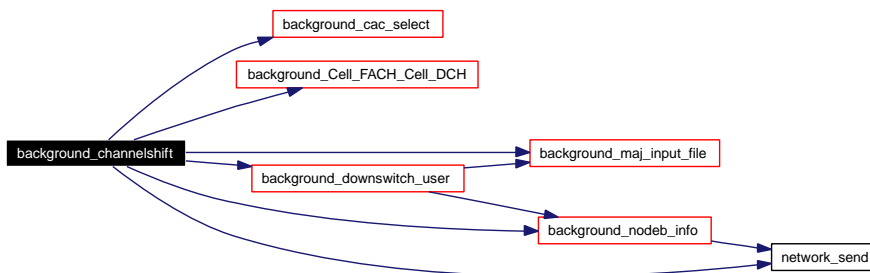
*ue\_id* represents the identification number of the UE.

Definition at line 541 of file background.c.

References actual\_time, background\_cac\_select(), background\_Cell\_FACH\_Cell\_DCH(), background\_downswitch\_user(), background\_maj\_input\_file(), background\_nodeb\_info(), chchange\_array, handover\_log\_file, init\_desynchro, network\_send(), ovsf\_log\_file, start\_time, and ue\_array.

Referenced by background\_management().

Here is the call graph for this function:



#### 4.3.1.12 void background\_management (int ue\_id)

This procedure checks if the UE has to start a new background session or not, based on the actual\_time variable.

**Parameters:**

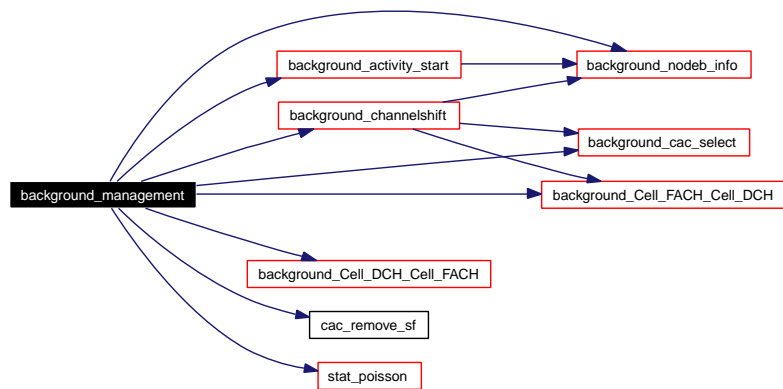
*ue\_id* represents the identification number of the UE.

Definition at line 627 of file background.c.

References actual\_time, background\_activity\_start(), background\_cac\_select(), background\_Cell\_DCH\_Cell\_FACH(), background\_Cell\_FACH\_Cell\_DCH(), background\_channelshift(), background\_nodeb\_info(), cac\_remove\_sf(), init\_desynchro, ovsf\_log\_file, start\_time, stat\_poisson(), and ue\_array.

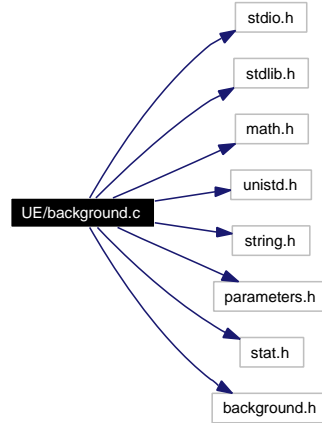
Referenced by main().

Here is the call graph for this function:



## 4.4 UE/background.c File Reference

Include dependency graph for background.c:



### Functions

- void `background_ue_to_rnc` (int session\_length\_max, char \*ip, int port, int launch)

#### 4.4.1 Function Documentation

##### 4.4.1.1 void background\_ue\_to\_rnc (int session\_length\_max, char \* ip, int port, int launch)

This procedure takes care of the background uplink part of the emulation, computing session and inter-session length, connection behaviours, etc. based on statistical distributions.

##### Parameters:

*session\_length\_max* represents the maximum session length (compared to the emulation total length).

*ip* represents the source IP address (UE).

*port* represents the destination port to reach (RNC).

*launch* is true (1) if the TG input file is already created and the traffic generation process has to start immediately, and false (0) otherwise.

Definition at line 36 of file background.c.

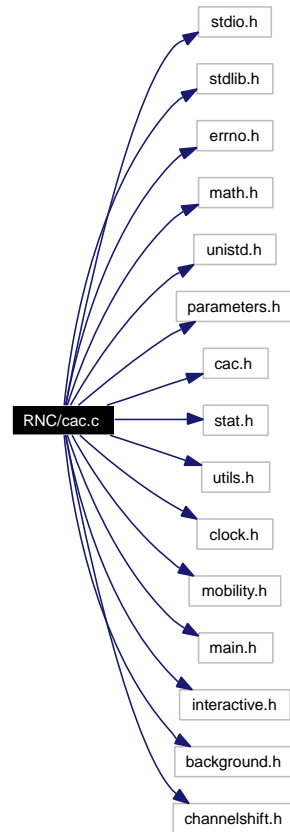
References `stat_lognormal()`, and `stat_pareto()`.

Here is the call graph for this function:



## 4.5 RNC/cac.c File Reference

Include dependency graph for cac.c:



### Functions

- void [cac\\_init](#) ()
- void [cac\\_print\\_ovsf\\_rec](#) (FILE \*f, int sf\_array, int level, int father\_x, int father\_y, int id\_a, int id\_b)
- void [cac\\_print\\_ovsf](#) (int NodeB, char Sector)
- void [cac\\_forward\\_desc](#) (int sf\_array, int root, int remove)
- void [cac\\_forward\\_asc](#) (int sf\_array, int leaf, int remove)
- int [cac\\_ascendant\\_sum](#) (int sf\_array, int leaf)
- void [cac\\_remove\\_sf](#) (int indice, int ue\_id)
- int [cac\\_ovsf\\_place\\_available](#) (int sf\_array, int sf, int ue\_id)
- int [cac\\_reorganise\\_ovsf\\_tree\\_rec](#) (int sf\_array, int indice, int level)
- int [cac\\_reorganise\\_ovsf\\_tree](#) (int sf\_array)
- downswitch\_t [cac\\_get\\_user\\_to\\_downswitch](#) (int ue\_id)
- void [cac\\_downswitch\\_pre\\_emptive\\_users](#) (int sf\_array, int sf, int ue\_id)
- int [cac\\_select\\_sf](#) (int sf, int ue\_id, int tc)
- int [cac\\_get\\_DSCH\\_sf](#) (int ue\_id)

### Variables

- ovsf\_t [ovsf](#) [12][OVSF\_MAX\_ELEMENTS]

## 4.5.1 Function Documentation

### 4.5.1.1 void cac\_init ()

This procedure initialises the CAC management by given an user profile to each of the active UEs in the simulation (1: Platinum, 2: Gold, 3: Silver and 4: Bronze) and by forming the ovsf array.

Definition at line 87 of file cac.c.

References stat\_proba(), and ue\_array.

Here is the call graph for this function:



### 4.5.1.2 void cac\_print\_ovsf\_rec (FILE \*f, int sf\_array, int level, int father\_x, int father\_y, int id\_a, int id\_b)

This is the recursive part of the cac\_print\_ovsf procedure.

#### Parameters:

- f* is the output file where the ovsf tree will be printed out,
- sf\_array* is the indice identifying the ovsf tree into the ovsf array,
- level* represents the depth level ([3;10]) into the ovsf tree,
- father\_x* represents the x coordinate of the father into the ovsf tree,
- father\_y* represents the y coordinate of the father into the ovsf tree,
- id\_a* represents the id of the first child of the father,
- id\_b* represents the id of the second child of the father.

Definition at line 157 of file cac.c.

References ovsf.

Referenced by cac\_print\_ovsf().

### 4.5.1.3 void cac\_print\_ovsf (int NodeB, char Sector)

This procedure recursively prints out an ovsf tree.

#### Parameters:

- NodeB* represents the identification number of the NodeB from which the OVSF tree has to be extracted,
- Sector* represents the identification the Sector's NodeB from which the OVSF tree has to be extracted.

Definition at line 183 of file cac.c.

References actual\_time, cac\_print\_ovsf\_rec(), init\_desynchro, ovsf, and start\_time.

Here is the call graph for this function:



#### 4.5.1.4 void cac\_forward\_desc (int sf\_array, int root, int remove)

This procedure recursively updates an the upper levels of an ovsf tree.

**Parameters:**

*sf\_array* is the indice identifying the ovsf tree into the ovsf array,

*root* represents the origin id of the recursive process,

*remove* is 1 if we are removing a spreading factor and is 0 if not.

Definition at line 235 of file cac.c.

References ovsf.

Referenced by cac\_remove\_sf(), and cac\_reorganise\_ovsf\_tree\_rec().

#### 4.5.1.5 void cac\_forward\_asc (int sf\_array, int leaf, int remove)

This procedure recursively updates an the lower levels of an ovsf tree.

**Parameters:**

*sf\_array* is the indice identifying the ovsf tree into the ovsf array,

*leaf* represents the id of the recursive process,

*remove* is 1 if we are removing a spreading factor and is 0 if not.

Definition at line 262 of file cac.c.

References ovsf, and utils\_even().

Referenced by cac\_remove\_sf(), and cac\_reorganise\_ovsf\_tree\_rec().

Here is the call graph for this function:



#### 4.5.1.6 int cac\_ancestor\_sum (int sf\_array, int leaf)

This procedure recursively sums the total allocated spreading factors in the leaf's ovsf tree branch.

**Parameters:**

*sf\_array* is the indice identifying the ovsf tree into the ovsf array,

*leaf* represents the origin id of the recursive process.

**Returns:**

the sum of the total allocated spreading factors in the leaf's ovsf tree branch.

Definition at line 287 of file cac.c.

References ovsf, and utils\_even().

Referenced by cac\_select\_sf().

Here is the call graph for this function:



#### 4.5.1.7 void cac\_remove\_sf (int indice, int ue\_id)

This procedure removes a spreading factor from an ovsf tree.

##### Parameters:

*indice* represents the indice of the spreading factor into the ovsf tree,

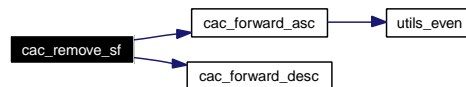
*ue\_id* represents the identification number of the UE.

Definition at line 306 of file cac.c.

References cac\_forward\_asc(), cac\_forward\_desc(), ovsf, and ue\_array.

Referenced by background\_downswitch\_user(), background\_management(), conversational\_drop\_user(), conversational\_management(), handover\_cac(), interactive\_downswitch\_user(), interactive\_management(), streaming\_drop\_user(), and streaming\_management().

Here is the call graph for this function:



#### 4.5.1.8 int cac\_ovsf\_place\_available (int sf\_array, int sf, int ue\_id)

This procedure searches into a ovsf tree to see if there is place enough to add a new spreading factor.

##### Parameters:

*sf\_array* is the indice identifying the ovsf tree into the ovsf array,

*sf* represents the length of the spreading factor asked,

*ue\_id* represents the identification number of the UE.

##### Returns:

1 if there is enough place without any user downswitch, 2 if there is enough place with user downswitching and 0 if there isn't enough place to accept the new call.

Definition at line 334 of file cac.c.

References ovsf, and ue\_array.

Referenced by cac\_downswitch\_pre\_emptive\_users().

#### 4.5.1.9 int cac\_reorganise\_ovsf\_tree\_rec (int sf\_array, int indice, int level)

This is the recursive part of the cac\_reorganise\_ovsf\_tree procedure.

**Parameters:**

*sf\_array* is the indice identifying the ovsf tree iinto the ovsf array,  
*indice* represents the id of the recursive process,  
*level* represents the depth level ([3;10]) into the ovsf tree.

**Returns:**

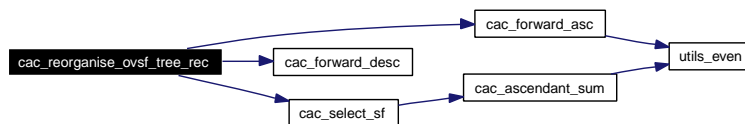
1 if reorganisation has been made and 0 otherwise.

Definition at line 377 of file cac.c.

References cac\_forward\_asc(), cac\_forward\_desc(), cac\_select\_sf(), ovsf, and ue\_array.

Referenced by cac\_reorganise\_ovsf\_tree().

Here is the call graph for this function:

**4.5.1.10 int cac\_reorganise\_ovsf\_tree (int sf\_array)**

If there is enough place to accept a new spreading factor, but it is not possible to place it, the solution is to reorganise the tree to create place before placing the new spreading factor.

**Parameters:**

*sf\_array* is the indice identifying the ovsf tree iinto the ovsf array.

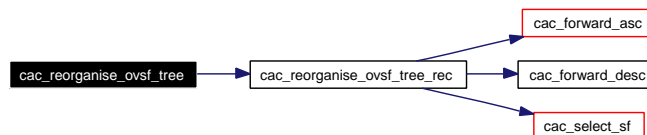
**Returns:**

1 if reorganisation has been made and 0 otherwise.

Definition at line 474 of file cac.c.

References cac\_reorganise\_ovsf\_tree\_rec().

Here is the call graph for this function:

**4.5.1.11 downswitch\_t cac\_get\_user\_to\_downswitch (int ue\_id)**

This procedure identifies which UE can be forced to be downswitched (in pre-emptive mode).

**Parameters:**

*ue\_id* represents the identification number of the UE.

**Returns:**

a structure containing the indice and the traffic class of the UE to downswitch.

Definition at line 491 of file cac.c.

References ue\_array.

Referenced by cac\_downswitch\_pre\_emptive\_users().

**4.5.1.12 void cac\_downswitch\_pre\_emptive\_users (int sf\_array, int sf, int ue\_id)**

This procedure downswitches UEs in pre-emptive mode as long as there is enough place to insert the new sf query.

**Parameters:**

*sf\_array* is the indice identifying the ovsf tree into the ovsf array,

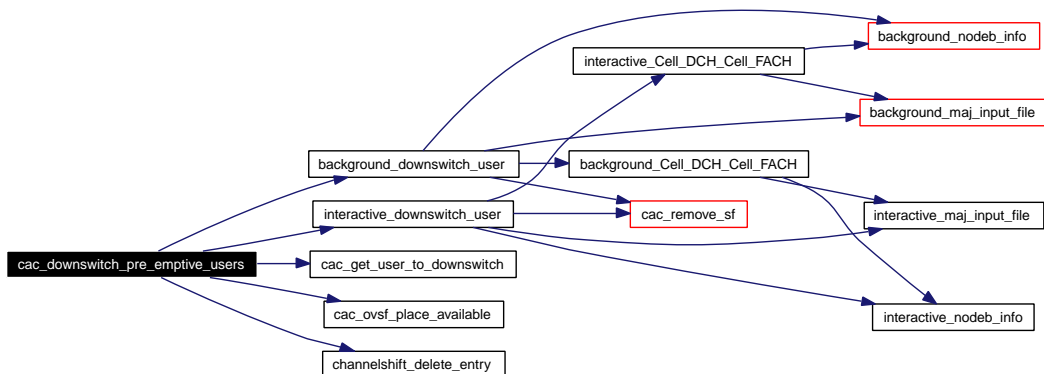
*sf* represents the length of the spreading factor asked,

*ue\_id* represents the identification number of the UE.

Definition at line 536 of file cac.c.

References background\_downswitch\_user(), cac\_get\_user\_to\_downswitch(), cac\_ovsf\_place\_available(), channelshift\_delete\_entry(), interactive\_downswitch\_user(), and ue\_array.

Here is the call graph for this function:

**4.5.1.13 int cac\_select\_sf (int sf, int ue\_id, int tc)**

This procedure adds a new spreading factor into an ovsf tree.

**Parameters:**

*sf* represents the length of the spreading factor asked,

*ue\_id* represents the identification number of the UE,

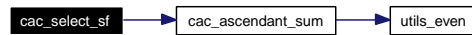
*tc* represents the traffic class linked to the spreading factor request.

Definition at line 560 of file cac.c.

References cac\_ascendant\_sum(), ovsf, and ue\_array.

Referenced by `background_cac_select()`, `cac_reorganise_ovsf_tree_rec()`, `conversational_cac_select()`, `interactive_cac_select()`, and `streaming_cac_select()`.

Here is the call graph for this function:



#### 4.5.1.14 `int cac_get_DSCH_sf(int ue_id)`

Returns an \*approximation\* of the SF allocated to the DSCH of the sector the Ue is in.

##### Parameters:

`ue_id` represents the identification number of the UE.

Definition at line 65 of file `cac.c`.

References `ovsf`, `ue_array`, and `utils_max()`.

Here is the call graph for this function:



## 4.5.2 Variable Documentation

### 4.5.2.1 `ovsf_t ovsf[12][OVSF_MAX_ELEMENTS]`

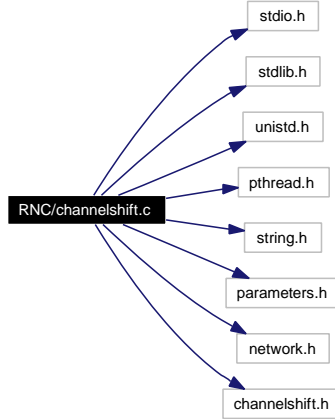
Represents a matrix of the 12 different ovsf tree (4 NodeB \* 3 Sectors).

Definition at line 38 of file `cac.c`.

Referenced by `cac_ascendant_sum()`, `cac_forward_asc()`, `cac_forward_desc()`, `cac_get_DSCH_sf()`, `cac_ovsf_place_available()`, `cac_print_ovsf()`, `cac_print_ovsf_rec()`, `cac_remove_sf()`, `cac_reorganise_ovsf_tree_rec()`, and `cac_select_sf()`.

## 4.6 RNC/channelshift.c File Reference

Include dependency graph for channelshift.c:



### Functions

- void [channelshift\\_delete\\_entry](#) (char \*ip, int tc)
- void [channelshift\\_reset\\_chchange\\_array](#) ()
- void [channelshift\\_management](#) ()

### Variables

- change\_t [chchange\\_array](#) [200]

### 4.6.1 Function Documentation

#### 4.6.1.1 void [channelshift\\_delete\\_entry](#) (char \* *ip*, int *tc*)

This procedure deletes a given "IP address-traffic class" couple from the channel change array.

##### Parameters:

*ip* represents the IP address of the couple,

*tc* represents the traffic class of the couple (1: Conversational, 2: Interactive, 3: Streaming and 4: Background).

Definition at line 41 of file channelshift.c.

References [chchange\\_array](#).

Referenced by [cac\\_downswitch\\_pre\\_emptive\\_users\(\)](#), and [qos\\_stop\(\)](#).

#### 4.6.1.2 void [channelshift\\_reset\\_chchange\\_array](#) ()

This procedure resets (put to 0) all the fields of the channel change array.

Definition at line 72 of file channelshift.c.

References `chchange_array`.

Referenced by `channelshift_management()`.

#### 4.6.1.3 void channelshift\_management ()

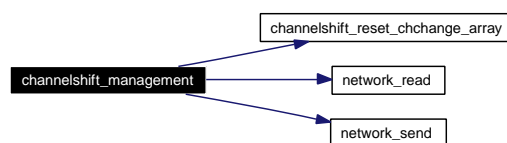
This procedure asks to the NodeBs if UEs has to up/downswitch. In case of, it fills the channel shift array.

Definition at line 89 of file `channelshift.c`.

References `channelshift_reset_chchange_array()`, `chchange_array`, `network_read()`, and `network_send()`.

Referenced by `main()`.

Here is the call graph for this function:



## 4.6.2 Variable Documentation

### 4.6.2.1 change\_t chchange\_array[200]

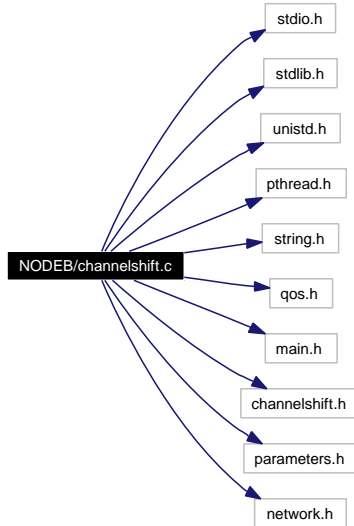
This channel change array indicates which UE has to up/downswitch (DCH <-> FACH).

Definition at line 31 of file `channelshift.c`.

Referenced by `background_channelshift()`, `channelchift_dch_data_update()`, `channelchift_dsch_data_update()`, `channelchift_fach_data_update()`, `channelshift_change_present()`, `channelshift_delete_entry()`, `channelshift_management()`, `channelshift_query()`, `channelshift_reset_chchange_array()`, and `interactive_channelshift()`.

## 4.7 NODEB/channelshift.c File Reference

Include dependency graph for channelshift.c:



### Functions

- void [channelshift\\_delete\\_entry](#) (char \*ip, int tc)
- void [channelshift\\_query](#) ()
- int [channelshift\\_change\\_present](#) (char \*ip, int tc)
- void [channelchift\\_fach\\_data\\_update](#) (int sector\_id, int id\_queue, int sent, int backlog\_b, int backlog\_p)
- void [channelchift\\_dsch\\_data\\_update](#) (int sector\_id, int id\_queue, int sent, int backlog\_b, int backlog\_p)
- void [channelchift\\_dch\\_data\\_update](#) (int sector\_id, int id\_queue, int sent, int backlog\_b, int backlog\_p)
- void \* [channelshift\\_management](#) ()

### Variables

- change\_t [chchange\\_array](#) [200]
- int [chchange\\_array\\_indice](#)
- pthread\_mutex\_t [chchange\\_array\\_access](#)

#### 4.7.1 Function Documentation

##### 4.7.1.1 void channelshift\_delete\_entry (char \* ip, int tc)

This procedure deletes a given "IP address-traffic class" couple from the channel change array.

##### Parameters:

*ip* represents the IP address of the couple,

*tc* represents the traffic class of the couple (1: Conversational, 2: Interactive, 3: Streaming and 4: Background).

Definition at line 49 of file channelshift.c.

References `chchange_array`, `chchange_array_access`, and `chchange_array_indice`.

#### 4.7.1.2 void channelshift\_query ()

This procedure sends to the RNC the changes to operate in channel allocation (up/downswitch).

Definition at line 79 of file channelshift.c.

References `chchange_array`, `chchange_array_access`, `chchange_array_indice`, and `network_send()`.

Referenced by `action_analysis()`.

Here is the call graph for this function:



#### 4.7.1.3 int channelshift\_change\_present (char \* ip, int tc)

This procedure looks for a given "IP address-traffic class" couple into the channel change array.

##### Parameters:

*ip* represents the IP address of the couple,

*tc* represents the traffic class of the couple (1: Conversational, 2: Interactive, 3: Streaming and 4: Background).

##### Returns:

the array indice where the couple is located in the array or 0 if the couple isn't present.

Definition at line 109 of file channelshift.c.

References `chchange_array`, and `chchange_array_indice`.

Referenced by `channelchift_dch_data_update()`, `channelchift_dsch_data_update()`, and `channelchift_fach_data_update()`.

#### 4.7.1.4 void channelchift\_fach\_data\_update (int sector\_id, int id\_queue, int sent, int backlog\_b, int backlog\_p)

This procedure updates the FACH array and, in case of upswitch, it specifies it into the channel change array.

##### Parameters:

*sector\_id* identifies the sector where the FACH is located ([0;2]),

*id\_queue* represents the queue identification number of the FACH ([1000;1999] | [4000;4999] | [7000;7999]),

*sent* represents the number of bytes sent using this FACH since its opening,

*backlog\_b* represents the number of bytes currently located in the FACH queue waiting to be sent,  
*backlog\_p* represents the number of packets currently located in the FACH queue waiting to be sent.

Definition at line 129 of file channelshift.c.

References channelshift\_change\_present(), chchange\_array, chchange\_array\_access, chchange\_array\_-,  
 indice, fach, and fach\_access.

Referenced by channelshift\_management().

Here is the call graph for this function:



#### 4.7.1.5 void channelshift\_dsch\_data\_update (int sector\_id, int id\_queue, int sent, int backlog\_b, int backlog\_p)

This procedure updates the DCH array and, in case of downswitch, it specifies it into the channel change array.

##### Parameters:

*sector\_id* identifies the sector where the DCH is located ([0;3]),  
*id\_queue* represents the queue identification number of the DCH ([3000;3999] | [6000;6999] | [9000;9999]),  
*sent* represents the number of bytes sent using this DCH since its opening,  
*backlog\_b* represents the number of bytes currently located in the DCH queue waiting to be sent,  
*backlog\_p* represents the number of packets currently located in the DCH queue waiting to be sent.

Definition at line 193 of file channelshift.c.

References channelshift\_change\_present(), chchange\_array, chchange\_array\_access, chchange\_array\_-,  
 indice, dsch, and dsch\_access.

Referenced by channelshift\_management().

Here is the call graph for this function:



#### 4.7.1.6 void channelshift\_dch\_data\_update (int sector\_id, int id\_queue, int sent, int backlog\_b, int backlog\_p)

This procedure updates the DCH array and, in case of downswitch, it specifies it into the channel change array.

##### Parameters:

*sector\_id* identifies the sector where the DCH is located ([0;3]),  
*id\_queue* represents the queue identification number of the DCH ([3000;3999] | [6000;6999] | [9000;9999]),

*sent* represents the number of bytes sent using this DCH since its opening,

*backlog\_b* represents the number of bytes currently located in the DCH queue waiting to be sent,

*backlog\_p* represents the number of packets currently located in the DCH queue waiting to be sent.

Definition at line 257 of file channelshift.c.

References channelshift\_change\_present(), chchange\_array, chchange\_array\_access, chchange\_array\_indice, dch, and dch\_access.

Referenced by channelshift\_management().

Here is the call graph for this function:



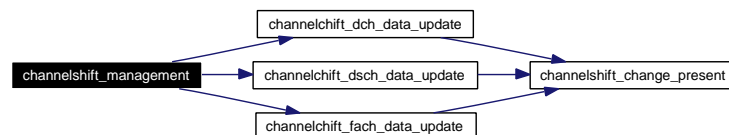
#### 4.7.1.7 void\* channelshift\_management ()

This thread controls the DCH and FACH (in)activity 200 times a second (5 ms TTI).

Definition at line 336 of file channelshift.c.

References channelchift\_dch\_data\_update(), channelchift\_dsch\_data\_update(), channelchift\_fach\_data\_update(), and simulation\_end.

Here is the call graph for this function:



## 4.7.2 Variable Documentation

### 4.7.2.1 change\_t chchange\_array[200]

This array contains all the channel changes the RNC has to operate.

Definition at line 31 of file channelshift.c.

### 4.7.2.2 int chchange\_array\_indice

This variable always equal to the indice of the highest used cell of the channel change array.

Definition at line 35 of file channelshift.c.

Referenced by channelchift\_dch\_data\_update(), channelchift\_dsch\_data\_update(), channelchift\_fach\_data\_update(), channelshift\_change\_present(), channelshift\_delete\_entry(), and channelshift\_query().

### 4.7.2.3 pthread\_mutex\_t chchange\_array\_access

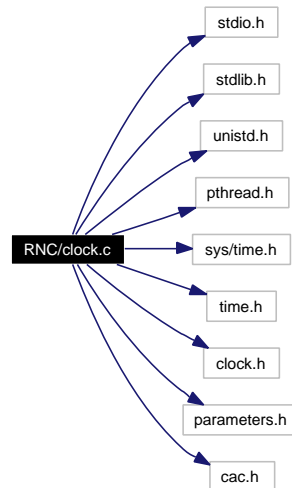
This mutex avoids concurrent accesses on the channel change array.

Definition at line 39 of file channelshift.c.

Referenced by channelshift\_dch\_data\_update(), channelshift\_dsch\_data\_update(), channelshift\_fach\_data\_update(), channelshift\_delete\_entry(), channelshift\_query(), and qos\_fin().

## 4.8 RNC/clock.c File Reference

Include dependency graph for clock.c:



### Functions

- void \* [clock\\_management](#) ()

### Variables

- int [actual\\_time](#)
- int [end\\_time](#)
- int [start\\_time](#)
- int [init\\_desynchro](#)

### 4.8.1 Function Documentation

#### 4.8.1.1 void\* [clock\\_management](#) ()

This thread manages the time during the whole emulation. It initialise the variables 'start\_time' and 'end\_time', and updates each seconds the 'actual\_time' variable.

Definition at line 53 of file clock.c.

References [actual\\_time](#), [end\\_time](#), and [start\\_time](#).

Referenced by [initialisation\\_starting\\_clock\(\)](#).

### 4.8.2 Variable Documentation

#### 4.8.2.1 int [actual\\_time](#)

This global variable represents the number of seconds passed since Epoch (01/01/1970).

Definition at line 34 of file clock.c.

Referenced by background\_activity\_start(), background\_Cell\_DCH\_Cell\_FACH(), background\_Cell\_FACH\_Cell\_DCH(), background\_channelshift(), background\_downswitch\_user(), background\_maj\_input\_file(), background\_management(), cac\_print\_ovsf(), clock\_management(), conversational\_activity\_start(), conversational\_Cell\_DCH\_Cell\_FACH(), conversational\_Cell\_FACH\_Cell\_DCH(), conversational\_drop\_user(), conversational\_management(), handover\_activity\_shifting(), handover\_cac(), handover\_maj\_back\_input\_file(), handover\_maj\_conv\_input\_file(), handover\_maj\_inter\_input\_file(), handover\_maj\_stream\_input\_file(), handover\_management(), initialisation\_ue\_desynchro(), interactive\_activity\_start(), interactive\_Cell\_DCH\_Cell\_FACH(), interactive\_Cell\_FACH\_Cell\_DCH(), interactive\_channelshift(), interactive\_downswitch\_user(), interactive\_maj\_input\_file(), interactive\_management(), interfaces\_creation(), main(), mobility\_management(), streaming\_activity\_start(), streaming\_Cell\_DCH\_Cell\_FACH(), streaming\_Cell\_FACH\_Cell\_DCH(), streaming\_drop\_user(), and streaming\_management().

#### 4.8.2.2 int end\_time

This global variable represents the time when the emulation will end (expressed in seconds since Epoch).

Definition at line 38 of file clock.c.

Referenced by background\_activity\_start(), clock\_management(), conversational\_activity\_start(), interactive\_activity\_start(), main(), and streaming\_activity\_start().

#### 4.8.2.3 int start\_time

This global variable represents the time when the emulation has started (expressed in seconds since Epoch).

Definition at line 42 of file clock.c.

Referenced by background\_activity\_start(), background\_Cell\_DCH\_Cell\_FACH(), background\_Cell\_FACH\_Cell\_DCH(), background\_channelshift(), background\_downswitch\_user(), background\_management(), cac\_print\_ovsf(), clock\_management(), conversational\_activity\_start(), conversational\_Cell\_DCH\_Cell\_FACH(), conversational\_Cell\_FACH\_Cell\_DCH(), conversational\_drop\_user(), conversational\_management(), handover\_activity\_shifting(), handover\_cac(), handover\_management(), interactive\_Cell\_DCH\_Cell\_FACH(), interactive\_Cell\_FACH\_Cell\_DCH(), interactive\_channelshift(), interactive\_downswitch\_user(), interactive\_management(), mobility\_management(), streaming\_activity\_start(), streaming\_Cell\_DCH\_Cell\_FACH(), streaming\_Cell\_FACH\_Cell\_DCH(), streaming\_drop\_user(), and streaming\_management().

#### 4.8.2.4 int init\_desynchro

Represents the time the different parts of the emulation (TG servers, TC qdisc creation, etc.) will take to start.

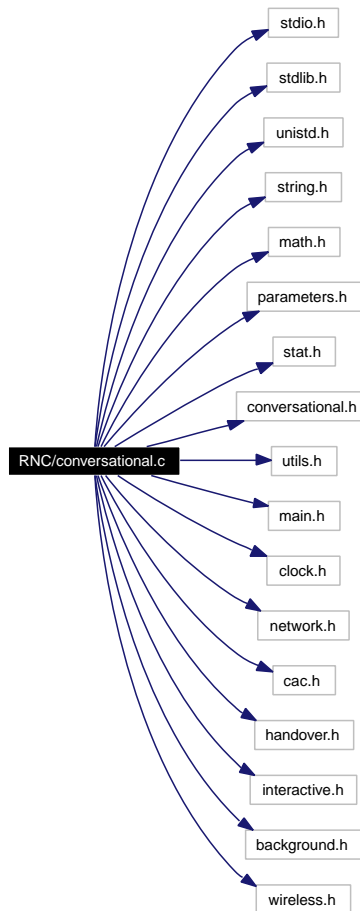
Definition at line 46 of file clock.c.

Referenced by background\_activity\_start(), background\_Cell\_DCH\_Cell\_FACH(), background\_Cell\_FACH\_Cell\_DCH(), background\_channelshift(), background\_downswitch\_user(), background\_management(), cac\_print\_ovsf(), conversational\_activity\_start(), conversational\_Cell\_DCH\_Cell\_FACH(), conversational\_Cell\_FACH\_Cell\_DCH(), conversational\_drop\_user(), conversational\_management(), handover\_activity\_shifting(), handover\_cac(), handover\_management(), interactive\_Cell\_DCH\_Cell\_FACH(), interactive\_Cell\_FACH\_Cell\_DCH(), interactive\_channelshift(), interactive\_downswitch\_user(), interactive\_management(), mobility\_management(), streaming\_activity\_start(),

streaming\_Cell\_DCH\_Cell\_FACH(), streaming\_Cell\_FACH\_Cell\_DCH(), streaming\_drop\_user(), and streaming\_management().

## 4.9 RNC/conversational.c File Reference

Include dependency graph for conversational.c:



### Functions

- void [conversational\\_rnc\\_to\\_ue](#) (network\_info\_t network\_info, int ue\_id)
- void [conversational\\_ue\\_to\\_rnc](#) (network\_info\_t network\_info, int ue\_id)
- void [conversational\\_ue\\_to\\_rnc\\_launch](#) (network\_info\_t network\_info, int ue\_id)
- void [conversational\\_nodeb\\_info](#) (int ue\_id, int start)
- void [conversational\\_activity\\_start](#) (int ue\_id)
- int [conversational\\_cac\\_select](#) (int ue\_id)
- void [conversational\\_Cell\\_FACH\\_Cell\\_DCH](#) (int ue\_id)
- void [conversational\\_Cell\\_DCH\\_Cell\\_FACH](#) (int ue\_id)
- void [conversational\\_drop\\_user](#) (int ue\_id)
- void [conversational\\_management](#) (int ue\_id)

## 4.9.1 Function Documentation

### 4.9.1.1 void conversational\_rnc\_to\_ue (network\_info\_t network\_info, int ue\_id)

This procedure takes care of the conversational downlink part of the emulation, computing session and inter-session length, connection behaviours, etc. based on statistical distributions.

**Parameters:**

*network\_info* represents a structure containing all the network information the procedure needs (source/destination IP, source/destination port, session length, etc.) to compute well,

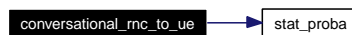
*ue\_id* represents the identification number of the UE.

Definition at line 43 of file conversational.c.

References stat\_proba(), and ue\_array.

Referenced by conversational\_activity\_start().

Here is the call graph for this function:



### 4.9.1.2 void conversational\_ue\_to\_rnc (network\_info\_t network\_info, int ue\_id)

This procedure sends queries to the UEs to create the tg input file of the uplink part of a conversational flow.

**Parameters:**

*network\_info* represents a structure containing all the network information the procedure needs (source/destination IP, source/destination port, session length, etc.) to compute well,

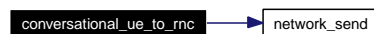
*ue\_id* represents the identification number of the UE.

Definition at line 112 of file conversational.c.

References network\_send(), and ue\_array.

Referenced by action\_analysis(), and conversational\_activity\_start().

Here is the call graph for this function:



### 4.9.1.3 void conversational\_ue\_to\_rnc\_launch (network\_info\_t network\_info, int ue\_id)

This procedure sends queries to the UEs to start the uplink part of a conversational flow.

**Parameters:**

*network\_info* represents a structure containing all the network information the procedure needs (source/destination IP, source/destination port, session length, etc.) to compute well.,

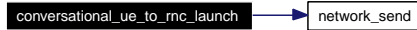
*ue\_id* represents the identification number of the UE.

Definition at line 129 of file conversational.c.

References network\_send(), and ue\_array.

Referenced by conversational\_activity\_start().

Here is the call graph for this function:



#### 4.9.1.4 void conversational\_nodeb\_info (int ue\_id, int start)

This procedure alerts the NodeBs about the flow creation/destruction. The messages may have various formats:

- "b ip up cu dt tc sf se bler": message corresponding to the creation of a new tc class and filter:
  1. ip represents the source IPv6 address of the traffic to filter,
  2. up represents the user profile of the source UE of the traffic (1: Platinum, 2: Gold, 3: Silver and 4: Bronze),
  3. cu represents the channel used to transport the traffic (1: DCH, 2: DSCH and 3: FACH),
  4. dt represents the value to which the DCH holding inactivity timer as to be initialised,
  5. tc represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),
  6. sf represents the spreading factor allocated to the source UE of the traffic,
  7. se represents the sector of the NodeB where the UE is located, and
  8. bler represents the BLER the transport channel undergoes.
- "s ip cu tc se ha": message corresponding to the destruction of a tc class and filter:
  1. ip represents the source IPv6 address of the traffic to filter,
  2. cu represents the channel used to transport the traffic (1: DCH, 2: DSCH and 3: FACH),
  3. tc represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),
  4. se represents the sector of the NodeB where the UE is located, and
  5. ha represents the fact that the destruction of the tc class and filter is due to an handover (1) or not (0).

#### Parameters:

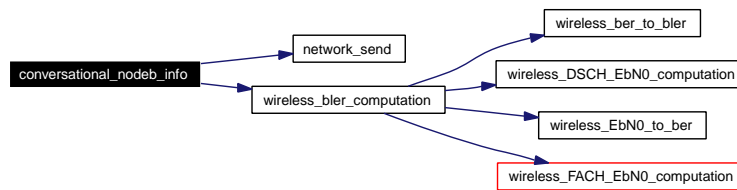
- ue\_id* represents the identification number of the UE,
- start* indicates if we are starting or stopping an activity.

Definition at line 162 of file conversational.c.

References network\_send(), ue\_array, and wireless\_bler\_computation().

Referenced by conversational\_activity\_start(), conversational\_drop\_user(), and conversational\_management().

Here is the call graph for this function:



#### 4.9.1.5 void conversational\_activity\_start (int ue\_id)

This procedure starts the conversational flow to both uplink and downlink directions.

##### Parameters:

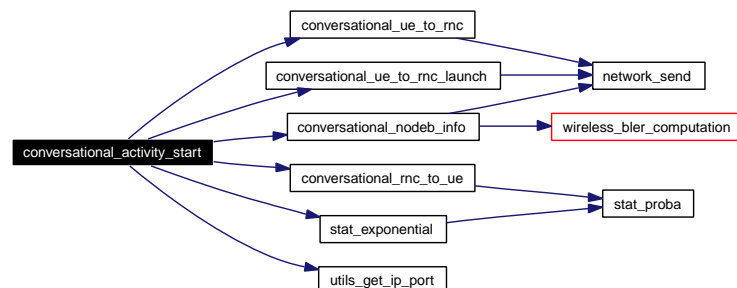
*ue\_id* represents the identification number of the UE.

Definition at line 183 of file conversational.c.

References actual\_time, conversational\_nodeb\_info(), conversational\_rnc\_to\_ue(), conversational\_ue\_to\_rnc(), conversational\_ue\_to\_rnc\_launch(), end\_time, init\_desynchro, start\_time, stat\_exponential(), ue\_array, and utils\_get\_ip\_port().

Referenced by conversational\_management().

Here is the call graph for this function:



#### 4.9.1.6 int conversational\_cac\_select (int ue\_id)

This procedure tries to allocate an acceptable spreading factor (based to the user profile of the UE) to a conversational session.

##### Parameters:

*ue\_id* represents the identification number of the UE.

##### Returns:

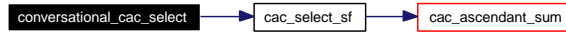
1 if the UE gets an acceptable spreading factor and 0 otherwise.

Definition at line 226 of file conversational.c.

References cac\_select\_sf(), and ue\_array.

Referenced by conversational\_management(), and handover\_cac().

Here is the call graph for this function:



#### 4.9.1.7 void conversational\_Cell\_FACH\_Cell\_DCH (int ue\_id)

This procedure upswitches only the interactive and background sessions.

##### Parameters:

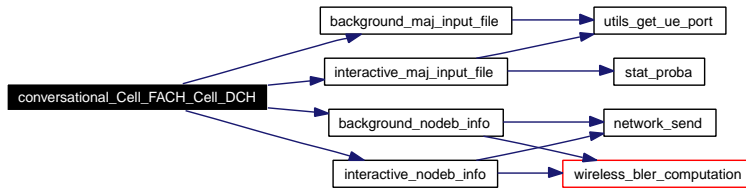
*ue\_id* represents the identification number of the UE.

Definition at line 322 of file conversational.c.

References actual\_time, background\_maj\_input\_file(), background\_nodeb\_info(), handover\_log\_file, init\_desynchro, interactive\_maj\_input\_file(), interactive\_nodeb\_info(), start\_time, and ue\_array.

Referenced by conversational\_management().

Here is the call graph for this function:



#### 4.9.1.8 void conversational\_Cell\_DCH\_Cell\_FACH (int ue\_id)

This procedure downswitches only the interactive and background sessions.

##### Parameters:

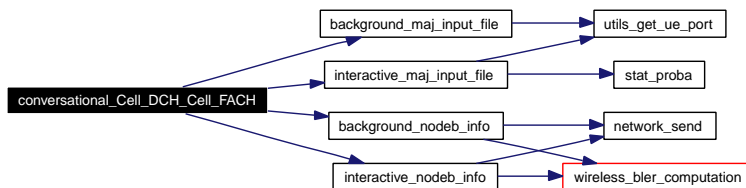
*ue\_id* represents the identification number of the UE.

Definition at line 382 of file conversational.c.

References actual\_time, background\_maj\_input\_file(), background\_nodeb\_info(), handover\_log\_file, init\_desynchro, interactive\_maj\_input\_file(), interactive\_nodeb\_info(), start\_time, and ue\_array.

Referenced by conversational\_drop\_user(), and conversational\_management().

Here is the call graph for this function:



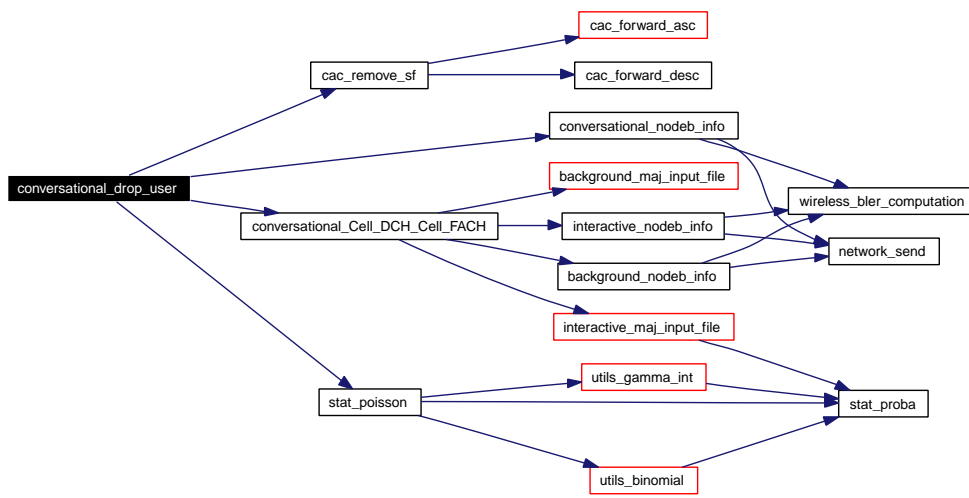
#### 4.9.1.9 void conversational\_drop\_user (int ue\_id)

This procedure drops a conversational session of a given user.

Definition at line 440 of file conversational.c.

References actual\_time, cac\_remove\_sf(), conversational\_Cell\_DCH\_Cell\_FACH(), conversational\_nodeb\_info(), init\_desynchro, ovsf\_log\_file, start\_time, stat\_poisson(), and ue\_array.

Here is the call graph for this function:



#### 4.9.1.10 void conversational\_management (int ue\_id)

This procedure checks if the UE has to start a new conversational session or not, based on the actual\_time variable.

##### Parameters:

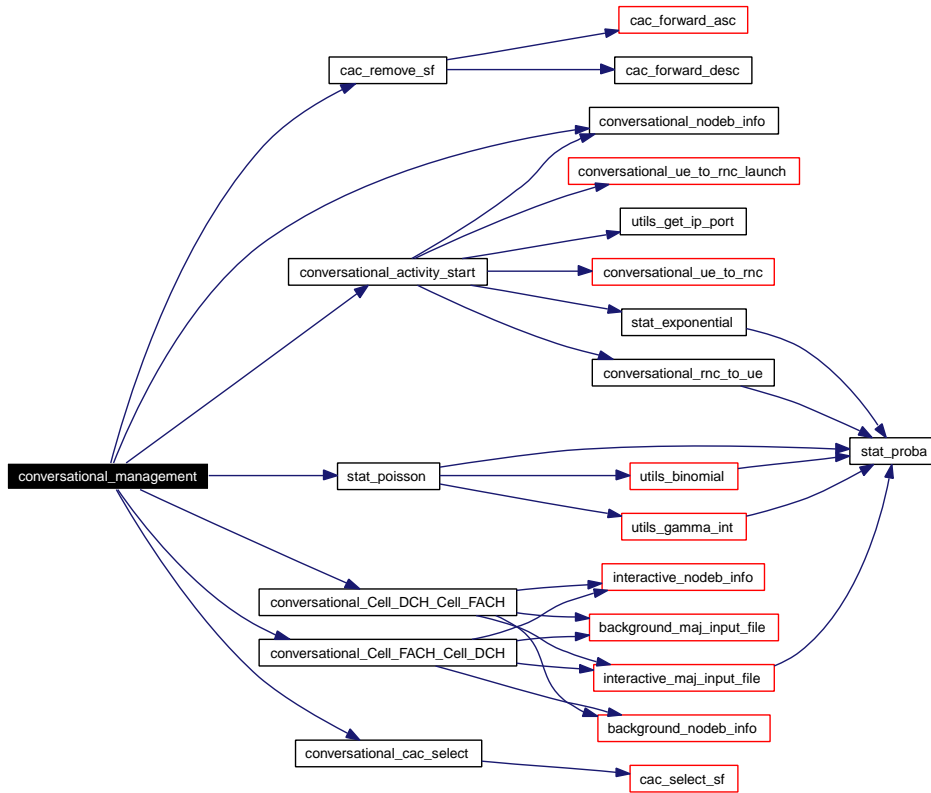
*ue\_id* represents the identification number of the UE.

Definition at line 473 of file conversational.c.

References actual\_time, cac\_remove\_sf(), conversational\_activity\_start(), conversational\_cac\_select(), conversational\_Cell\_DCH\_Cell\_FACH(), conversational\_Cell\_FACH\_Cell\_DCH(), conversational\_nodeb\_info(), init\_desynchro, ovsf\_log\_file, start\_time, stat\_poisson(), and ue\_array.

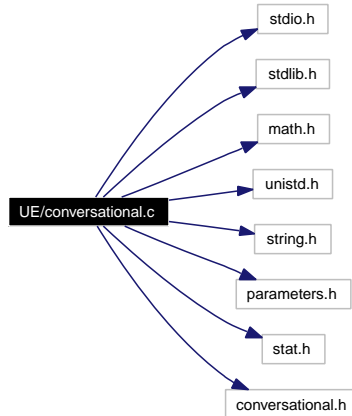
Referenced by main().

Here is the call graph for this function:



## 4.10 UE/conversational.c File Reference

Include dependency graph for conversational.c:



### Functions

- void [conversational\\_ue\\_to\\_rnc](#) (int session\_length, char \*ip, int port, int launch)

#### 4.10.1 Function Documentation

##### 4.10.1.1 void [conversational\\_ue\\_to\\_rnc](#) (int *session\_length*, char \* *ip*, int *port*, int *launch*)

This procedure takes care of the conversational uplink part of the emulation, computing session and inter-session length, connection behaviours, etc. based on statistical distributions.

#### Parameters:

*session\_length* represents the conversational session length,

*ip* represents the source IP address (UE),

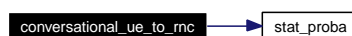
*port* represents the destination port to reach (RNC),

*launch* is true (1) if the TG input file is already created and the traffic generation process has to start immediately, and false (0) otherwise.

Definition at line 36 of file conversational.c.

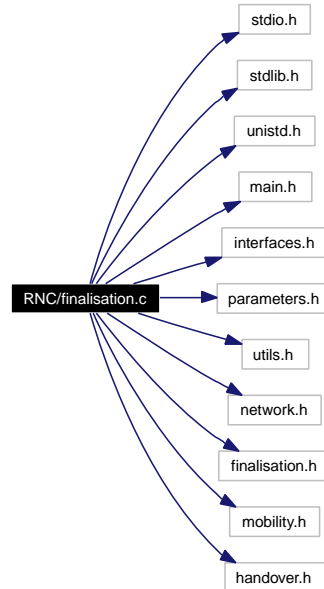
References [stat\\_proba\(\)](#).

Here is the call graph for this function:



## 4.11 RNC/finalisation.c File Reference

Include dependency graph for finalisation.c:



### Functions

- void [finalisation\\_management](#) ()

#### 4.11.1 Function Documentation

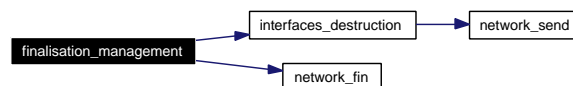
##### 4.11.1.1 void finalisation\_management ()

This procedure is responsible of the emulation finalisation (clock, network, mobility, etc.).

Definition at line 34 of file finalisation.c.

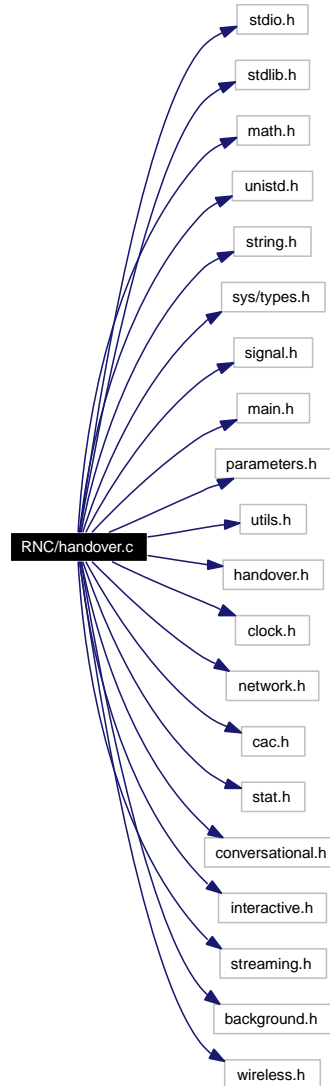
References `handover_log_file`, `interfaces_destruction()`, `mob_file_array`, `network_fin()`, and `ovsf_log_file`.

Here is the call graph for this function:



## 4.12 RNC/handover.c File Reference

Include dependency graph for handover.c:



### Functions

- void [handover\\_maj\\_conv\\_input\\_file](#) (char \*ip, char \*new\_ip, int ue\_id, int downlink)
- void [handover\\_maj\\_inter\\_input\\_file](#) (char \*ip, char \*new\_ip, int ue\_id)
- void [handover\\_maj\\_stream\\_input\\_file](#) (char \*ip, char \*new\_ip, int ue\_id)
- void [handover\\_maj\\_back\\_input\\_file](#) (char \*ip, char \*new\_ip, int ue\_id, int downlink)
- void [handover\\_maj\\_up\\_input\\_file](#) (char \*ip, char \*new\_ip, int ue\_id)
- void [handover\\_maj\\_tg\\_input\\_files](#) (char \*ip, char \*new\_ip, int ue\_id)
- void [handover\\_activity\\_shifting](#) (char \*ip, char \*new\_ip, int ue\_id, int NodeB\_dst, char Sector\_dst)
- void [handover\\_cac](#) (int ue\_id, int NodeB\_dst, char Sector\_dst)
- void [handover\\_management](#) (int ue\_id, int NodeB\_dst, char Sector\_dst)

## Variables

- FILE \* [handover\\_log\\_file](#)
- FILE \* [ovsf\\_log\\_file](#)

## 4.12.1 Function Documentation

### 4.12.1.1 void `handover_maj_conv_input_file` (`char * ip`, `char * new_ip`, `int ue_id`, `int downlink`)

When an handover occurs and the UE is performing a conversational session, this procedure updates the TG input file of the conversational stream.

#### Parameters:

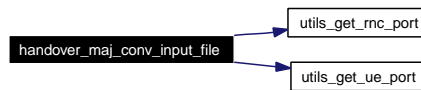
- ip* represents the stream former destination IP address,
- new\_ip* represents the stream new destination IP address,
- ue\_id* represents the identification number of the UE,
- downlink* is true (1) if the TG input file to update is in the downlink direction, and false (0) if it is in the uplink direction.

Definition at line 59 of file `handover.c`.

References `actual_time`, `ue_array`, `utils_get_rnc_port()`, and `utils_get_ue_port()`.

Referenced by `handover_maj_tg_input_files()`, and `handover_maj_up_input_file()`.

Here is the call graph for this function:



### 4.12.1.2 void `handover_maj_inter_input_file` (`char * ip`, `char * new_ip`, `int ue_id`)

When an handover occurs and the UE is performing an interactive session, this procedure updates the TG input file of the interactive stream.

#### Parameters:

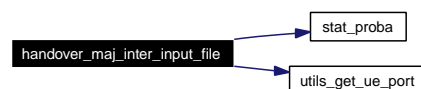
- ip* represents the stream former destination IP address,
- new\_ip* represents the stream new destination IP address,
- ue\_id* represents the identification number of the UE.

Definition at line 161 of file `handover.c`.

References `actual_time`, `stat_proba()`, `ue_array`, and `utils_get_ue_port()`.

Referenced by `handover_maj_tg_input_files()`.

Here is the call graph for this function:



#### 4.12.1.3 void handover\_maj\_stream\_input\_file (char \* ip, char \* new\_ip, int ue\_id)

When an handover occurs and the UE is performing a streaming session, this procedure updates the TG input file of the video streaming stream.

**Parameters:**

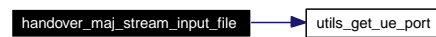
- ip* represents the stream former destination IP address,
- new\_ip* represents the stream new destination IP address,
- ue\_id* represents the identification number of the UE.

Definition at line 264 of file handover.c.

References actual\_time, ue\_array, and utils\_get\_ue\_port().

Referenced by handover\_maj\_tg\_input\_files().

Here is the call graph for this function:



#### 4.12.1.4 void handover\_maj\_back\_input\_file (char \* ip, char \* new\_ip, int ue\_id, int downlink)

When an handover occurs and a UE is performing a background session, this procedure updates the TG input file of the background stream.

**Parameters:**

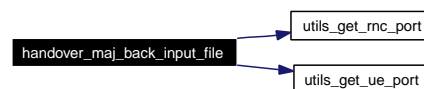
- ip* represents the stream former destination IP address,
- new\_ip* represents the stream new destination IP address,
- ue\_id* represents the identification number of the UE,
- downlink* is true (1) if the TG input file to update is in the downlink direction, and false (0) if it is in the uplink direction.

Definition at line 331 of file handover.c.

References actual\_time, ue\_array, utils\_get\_rnc\_port(), and utils\_get\_ue\_port().

Referenced by handover\_maj\_tg\_input\_files(), and handover\_maj\_up\_input\_file().

Here is the call graph for this function:



#### 4.12.1.5 void handover\_maj\_up\_input\_file (char \* ip, char \* new\_ip, int ue\_id)

This procedure decides which uplink TG input file is to be updated.

**Parameters:**

- ip* represents the stream former destination IP address,

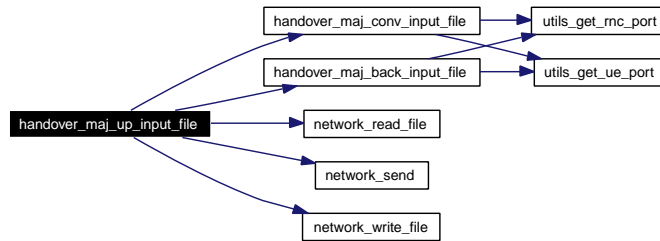
*new\_ip* represents the stream new destination IP address,  
*ue\_id* represents the identification number of the UE.

Definition at line 438 of file handover.c.

References `handover_maj_back_input_file()`, `handover_maj_conv_input_file()`, `network_read_file()`, `network_send()`, `network_write_file()`, and `ue_array`.

Referenced by `handover_maj_tg_input_files()`.

Here is the call graph for this function:



#### 4.12.1.6 void handover\_maj\_tg\_input\_files (char \* ip, char \* new\_ip, int ue\_id)

This procedure decides which TG input file is to be updated.

##### Parameters:

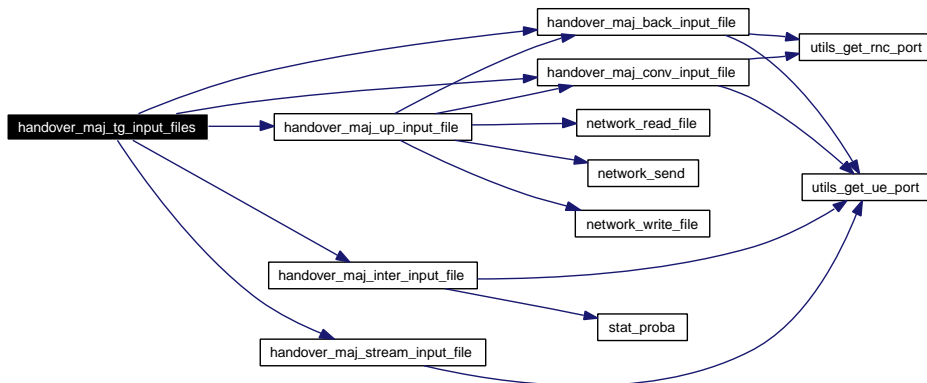
*ip* represents the stream former destination IP address,  
*new\_ip* represents the stream new destination IP address,  
*ue\_id* represents the identification number of the UE

Definition at line 488 of file handover.c.

References `handover_maj_back_input_file()`, `handover_maj_conv_input_file()`, `handover_maj_inter_input_file()`, `handover_maj_stream_input_file()`, `handover_maj_up_input_file()`, and `ue_array`.

Referenced by `handover_management()`.

Here is the call graph for this function:



#### 4.12.1.7 void handover\_activity\_shifting (char \* ip, char \* new\_ip, int ue\_id, int NodeB\_dst, char Sector\_dst)

This procedure decides the activities to be shifted.

##### Parameters:

*ip* represents the stream former destination IP address,

*new\_ip* represents the stream new destination IP address,

*ue\_id* represents the identification number of the UE,

*NodeB\_dst* represents the identification number of the destination NodeB of the handover ([0;3]),

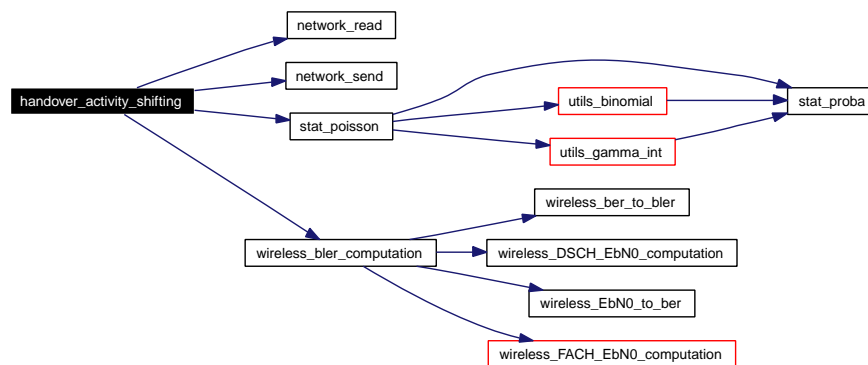
*Sector\_dst* represents the identification character of the destination Sector of the handover ('a', 'b' or 'c').

Definition at line 508 of file handover.c.

References actual\_time, init\_desynchro, network\_read(), network\_send(), start\_time, stat\_poisson(), ue\_array, and wireless\_bler\_computation().

Referenced by handover\_management().

Here is the call graph for this function:



#### 4.12.1.8 void handover\_cac (int ue\_id, int NodeB\_dst, char Sector\_dst)

This procedure tries to allocate an acceptable spreading factor to a UE based to its user profile and the traffic class.

##### Parameters:

*ue\_id* represents the identification number of the UE,

*NodeB\_dst* represents the identification number of the destination NodeB of the handover ([0;3]),

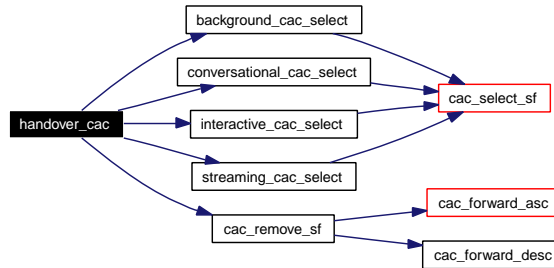
*Sector\_dst* represents the identification character of the destination Sector of the handover ('a', 'b' or 'c').

Definition at line 662 of file handover.c.

References actual\_time, background\_cac\_select(), cac\_remove\_sf(), conversational\_cac\_select(), init\_desynchro, interactive\_cac\_select(), ovsf\_log\_file, start\_time, streaming\_cac\_select(), and ue\_array.

Referenced by handover\_management().

Here is the call graph for this function:



#### 4.12.1.9 void handover\_management (int ue\_id, int NodeB\_dst, char Sector\_dst)

This procedure directs the entire handover process, deciding which process to stop or kill, updating TG input files, etc.

##### Parameters:

*ue\_id* represents the identification number of the UE,

*NodeB\_dst* represents the identification number of the destination NodeB of the handover ([0;3]),

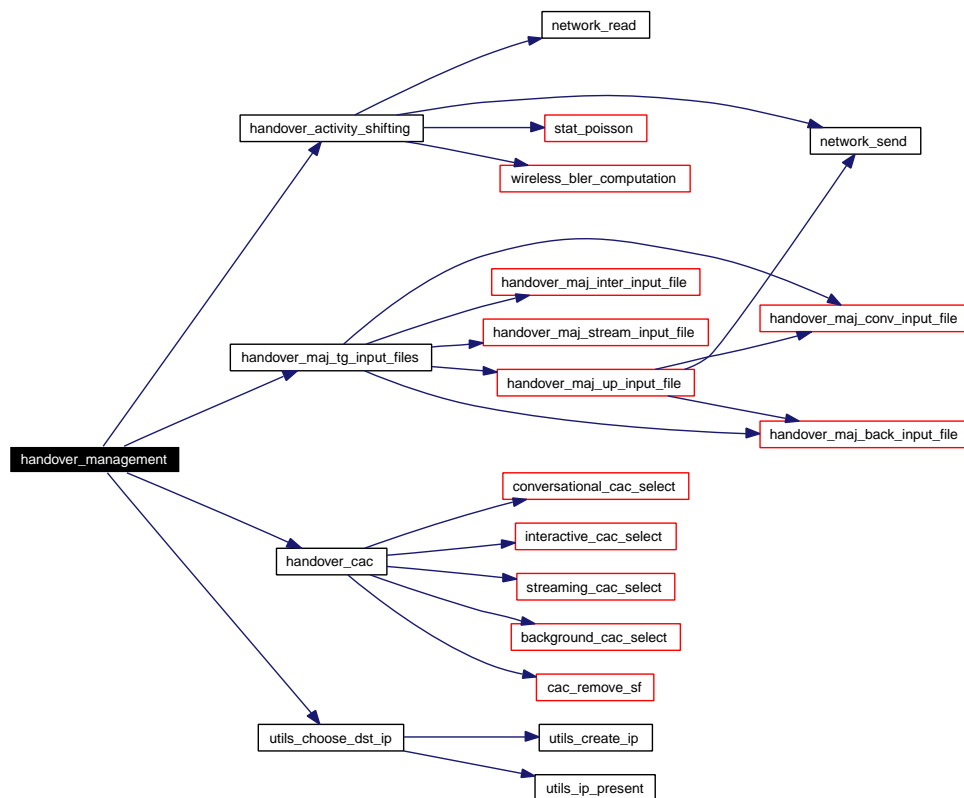
*Sector\_dst* represents the identification character of the destination Sector of the handover ('a', 'b' or 'c').

Definition at line 788 of file handover.c.

References `actual_time`, `handover_activity_shifting()`, `handover_cac()`, `handover_log_file`, `handover_maj_tg_input_files()`, `init_desynchro`, `start_time`, `ue_array`, and `utils_choose_dst_ip()`.

Referenced by `mobility_management()`.

Here is the call graph for this function:



## 4.12.2 Variable Documentation

### 4.12.2.1 FILE\* [handover\\_log\\_file](#)

Global variable representing the file where the handover logs are written.

Definition at line 43 of file handover.c.

Referenced by `background_Cell_DCH_Cell_FACH()`, `background_Cell_FACH_Cell_DCH()`, `background_channelshift()`, `background_downswitch_user()`, `conversational_Cell_DCH_Cell_FACH()`, `conversational_Cell_FACH_Cell_DCH()`, `finalisation_management()`, `handover_management()`, `interactive_Cell_DCH_Cell_FACH()`, `interactive_Cell_FACH_Cell_DCH()`, `interactive_channelshift()`, `interactive_downswitch_user()`, `streaming_Cell_DCH_Cell_FACH()`, and `streaming_Cell_FACH_Cell_DCH()`.

### 4.12.2.2 FILE\* [ovsf\\_log\\_file](#)

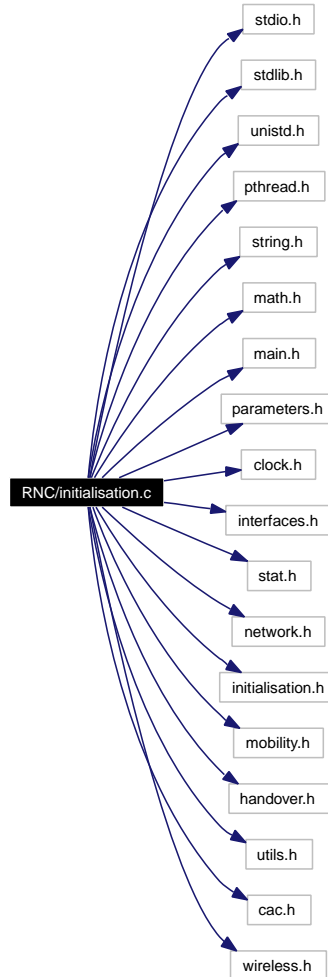
Global variable representing the file where the ovsf logs are written.

Definition at line 47 of file handover.c.

Referenced by `background_channelshift()`, `background_management()`, `conversational_drop_user()`, `conversational_management()`, `finalisation_management()`, `handover_cac()`, `interactive_channelshift()`, `interactive_management()`, `streaming_drop_user()`, and `streaming_management()`.

## 4.13 RNC/initialisation.c File Reference

Include dependency graph for initialisation.c:



### Functions

- void [initialisation\\_starting\\_clock \(\)](#)
- void [initialisation\\_ip\\_init \(\)](#)
- void [initialisation\\_ue\\_desynchro \(\)](#)
- void [initialisation\\_management \(\)](#)

#### 4.13.1 Function Documentation

##### 4.13.1.1 void [initialisation\\_starting\\_clock \(\)](#)

This procedure starts a new thread taking care of the clock management throughout the emulation.

Definition at line 41 of file `initialisation.c`.

References `clock_management()`.

Referenced by `initialisation_management()`.

Here is the call graph for this function:



#### 4.13.1.2 void initialisation\_ip\_init ()

This procedure allocates an IPv6 address to each UE based on the serving NodeB and Sector.

Definition at line 65 of file `initialisation.c`.

References `ue_array`.

Referenced by `initialisation_management()`.

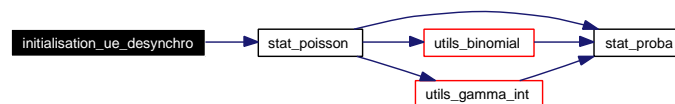
#### 4.13.1.3 void initialisation\_ue\_desynchro ()

This procedure enables a desynchronisation of all UE at the beginning of the emulation.

Definition at line 245 of file `initialisation.c`.

References `actual_time`, `stat_poisson()`, and `ue_array`.

Here is the call graph for this function:



#### 4.13.1.4 void initialisation\_management ()

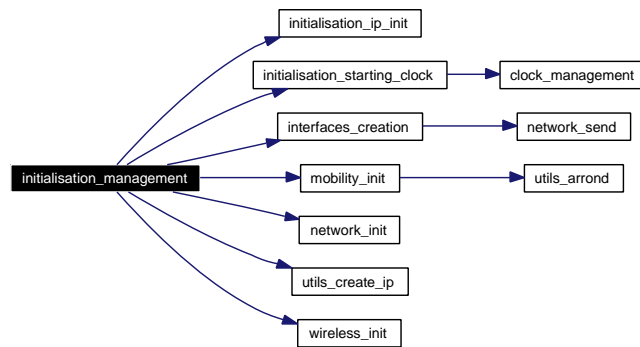
This procedure is responsible of the emulation initialisation (clock, network, mobility, etc.).

Definition at line 287 of file `initialisation.c`.

References `initialisation_ip_init()`, `initialisation_starting_clock()`, `interfaces_creation()`, `mobility_init()`, `network_init()`, `utils_create_ip()`, and `wireless_init()`.

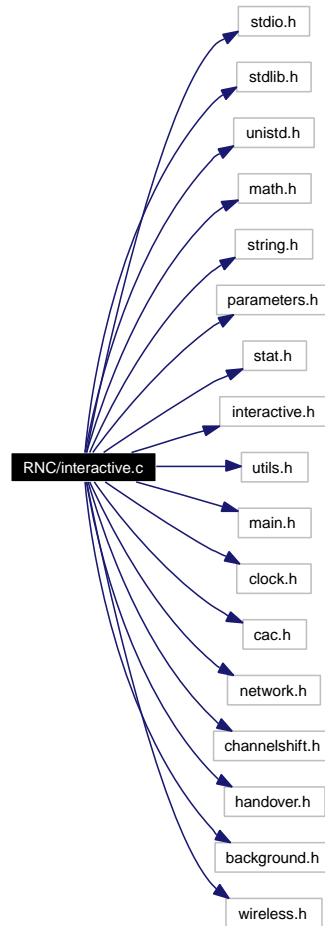
Referenced by `main()`.

Here is the call graph for this function:



## 4.14 RNC/interactive.c File Reference

Include dependency graph for interactive.c:



### Functions

- void [interactive\\_maj\\_input\\_file](#) (int ue\_id)
- void [interactive\\_nodeb\\_info](#) (int ue\_id, int start)
- void [interactive\\_activity\\_start](#) (int ue\_id)
- int [interactive\\_cac\\_select](#) (int ue\_id)
- void [interactive\\_Cell\\_FACH\\_Cell\\_DCH](#) (int ue\_id)
- void [interactive\\_Cell\\_DCH\\_Cell\\_FACH](#) (int ue\_id)
- void [interactive\\_downswitch\\_user](#) (int ue\_id, int forced)
- void [interactive\\_channelshift](#) (int ue\_id)
- void [interactive\\_management](#) (int ue\_id)

### 4.14.1 Function Documentation

#### 4.14.1.1 void [interactive\\_maj\\_input\\_file](#) (int ue\_id)

This procedure updates the TG input file following a channel switching (DCH <-> FACH).

**Parameters:**

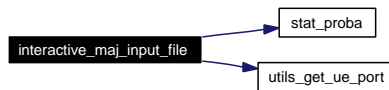
*ue\_id* represents the identification number of the UE.

Definition at line 42 of file interactive.c.

References actual\_time, stat\_proba(), ue\_array, and utils\_get\_ue\_port().

Referenced by background\_Cell\_DCH\_Cell\_FACH(), background\_Cell\_FACH\_Cell\_DCH(), conversational\_Cell\_DCH\_Cell\_FACH(), conversational\_Cell\_FACH\_Cell\_DCH(), interactive\_channelshift(), interactive\_downswitch\_user(), streaming\_Cell\_DCH\_Cell\_FACH(), and streaming\_Cell\_FACH\_Cell\_DCH().

Here is the call graph for this function:

**4.14.1.2 void interactive\_nodeb\_info (int ue\_id, int start)**

This procedure alerts the NodeBs about the flow creation/destruction. The messages may have various formats:

- "b ip up cu dt tc sf se bler": message corresponding to the creation of a new tc class and filter:
  1. ip represents the source IPv6 address of the traffic to filter,
  2. up represents the user profile of the source UE of the traffic (1: Platinum, 2: Gold, 3: Silver and 4: Bronze),
  3. cu represents the channel used to transport the traffic (1: DCH, 2: DSCH and 3: FACH),
  4. dt represents the value to which the DCH holding inactivity timer as to be initialised,
  5. tc represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),
  6. sf represents the spreading factor allocated to the source UE of the traffic,
  7. se represents the sector of the NodeB where the UE is located, and
  8. bler represents the BLER the transport channel undergoes.
- "s ip cu tc se ha": message corresponding to the destruction of a tc class and filter:
  1. ip represents the source IPv6 address of the traffic to filter,
  2. cu represents the channel used to transport the traffic (1: DCH, 2: DSCH and 3: FACH),
  3. tc represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),
  4. se represents the sector of the NodeB where the UE is located, and
  5. ha represents the fact that the destruction of the tc class and filter is due to an handover (1) or not (0).

**Parameters:**

*ue\_id* represents the identification number of the UE,

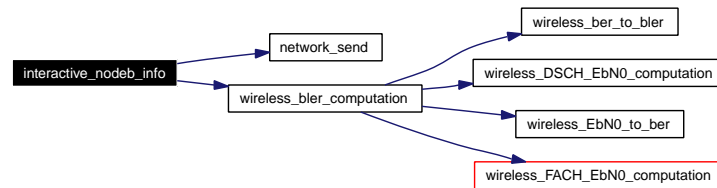
*start* indicates if we are starting or stopping an activity.

Definition at line 159 of file interactive.c.

References network\_send(), ue\_array, and wireless\_bler\_computation().

Referenced by background\_Cell\_DCH\_Cell\_FACH(), background\_Cell\_FACH\_Cell\_DCH(), conversational\_Cell\_DCH\_Cell\_FACH(), conversational\_Cell\_FACH\_Cell\_DCH(), interactive\_activity\_start(), interactive\_channelshift(), interactive\_downswitch\_user(), interactive\_management(), streaming\_Cell\_DCH\_Cell\_FACH(), and streaming\_Cell\_FACH\_Cell\_DCH().

Here is the call graph for this function:



#### 4.14.1.3 void interactive\_activity\_start (int ue\_id)

This procedure takes care of the interactive part of the emulation, computing session and inter-session length, connection behaviours, etc. based on statistical distributions.

##### Parameters:

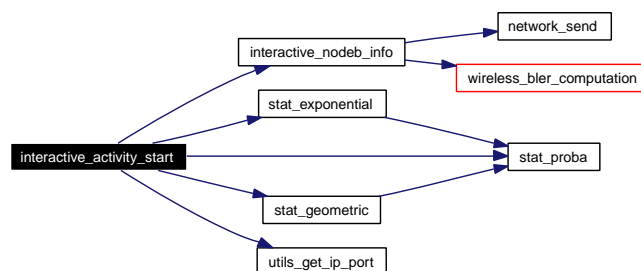
*ue\_id* represents the identification number of the UE.

Definition at line 180 of file interactive.c.

References actual\_time, end\_time, interactive\_nodeb\_info(), stat\_exponential(), stat\_geometric(), stat\_proba(), ue\_array, and utils\_get\_ip\_port().

Referenced by interactive\_management().

Here is the call graph for this function:



#### 4.14.1.4 int interactive\_cac\_select (int ue\_id)

This procedure tries to allocate an acceptable spreading factor (based to the user profile of the UE) to an interactive session.

##### Parameters:

*ue\_id* represents the identification number of the UE.

**Returns:**

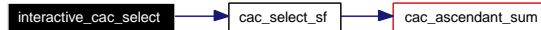
1 if the UE gets an acceptable spreading factor and 0 otherwise.

Definition at line 253 of file interactive.c.

References cac\_select\_sf(), and ue\_array.

Referenced by handover\_cac(), interactive\_channelshift(), and interactive\_management().

Here is the call graph for this function:

**4.14.1.5 void interactive\_Cell\_FACH\_Cell\_DCH (int ue\_id)**

This procedure upswitches only the background sessions.

**Parameters:**

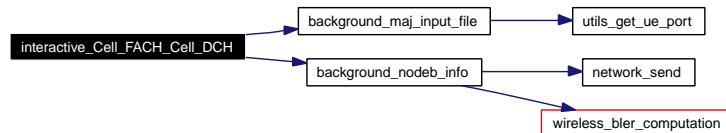
*ue\_id* represents the identification number of the UE.

Definition at line 350 of file interactive.c.

References actual\_time, background\_maj\_input\_file(), background\_nodeb\_info(), handover\_log\_file, init\_desynchro, start\_time, and ue\_array.

Referenced by interactive\_channelshift(), and interactive\_management().

Here is the call graph for this function:

**4.14.1.6 void interactive\_Cell\_DCH\_Cell\_FACH (int ue\_id)**

This procedure downswitches only the background sessions.

**Parameters:**

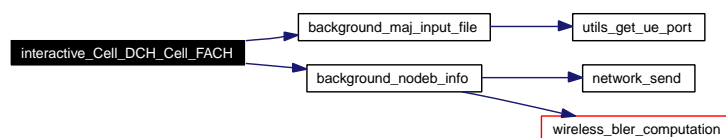
*ue\_id* represents the identification number of the UE.

Definition at line 386 of file interactive.c.

References actual\_time, background\_maj\_input\_file(), background\_nodeb\_info(), handover\_log\_file, init\_desynchro, start\_time, and ue\_array.

Referenced by interactive\_downswitch\_user(), and interactive\_management().

Here is the call graph for this function:



#### 4.14.1.7 void interactive\_downswitch\_user (int ue\_id, int forced)

This procedure downswitches a UE from Cell\_DCHp to Cell\_FACH.

##### Parameters:

*ue\_id* represents the identification number of the UE, and

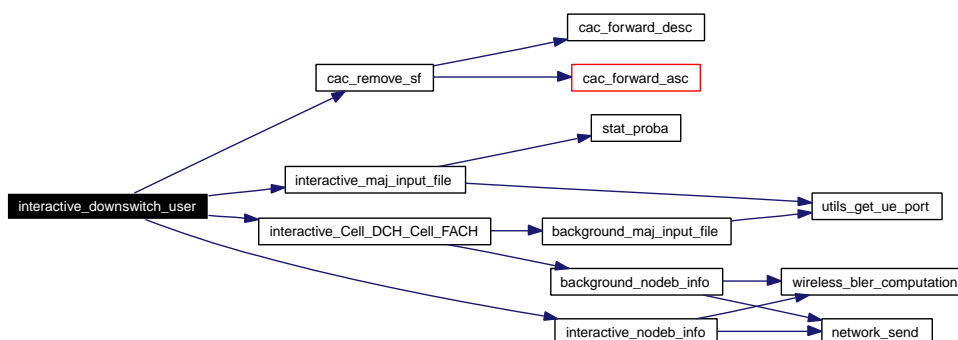
*forced* 1 if the downswitched is forced (the UE was in pre-emptive mode and has to downswitch before the end of the pre-emptive timer) and 2 if dropped (not enough transmission power).

Definition at line 423 of file interactive.c.

References actual\_time, cac\_remove\_sf(), handover\_log\_file, init\_desynchro, interactive\_Cell\_DCH\_Cell\_FACH(), interactive\_maj\_input\_file(), interactive\_nodeb\_info(), start\_time, and ue\_array.

Referenced by cac\_downswitch\_pre\_emptive\_users(), and interactive\_channelshift().

Here is the call graph for this function:



#### 4.14.1.8 void interactive\_channelshift (int ue\_id)

When a NodeB warns that UE has to up/downswitch (DCH <-> FACH), this procedure is called to do so.

##### Parameters:

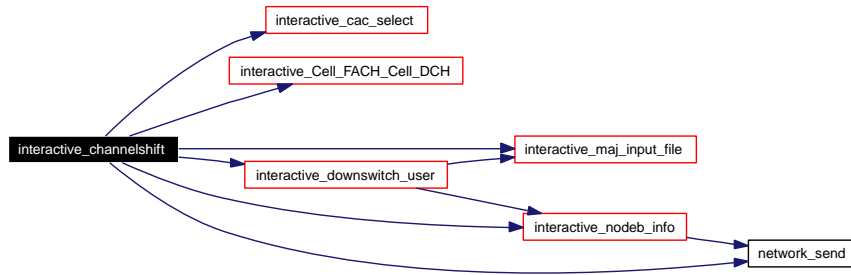
*ue\_id* represents the identification number of the UE.

Definition at line 488 of file interactive.c.

References actual\_time, chchange\_array, handover\_log\_file, init\_desynchro, interactive\_cac\_select(), interactive\_Cell\_FACH\_Cell\_DCH(), interactive\_downswitch\_user(), interactive\_maj\_input\_file(), interactive\_nodeb\_info(), network\_send(), ovsf\_log\_file, start\_time, and ue\_array.

Referenced by interactive\_management().

Here is the call graph for this function:



#### 4.14.1.9 void interactive\_management (int ue\_id)

This procedure checks if the UE has to start a new interactive session or not, based on the `actual_time` variable.

##### Parameters:

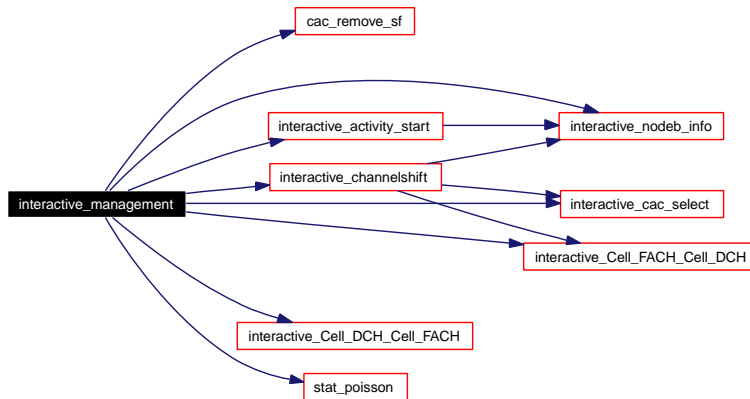
*ue\_id* represents the identification number of the UE.

Definition at line 574 of file `interactive.c`.

References `actual_time`, `cac_remove_sf()`, `init_desynchro`, `interactive_activity_start()`, `interactive_cac_select()`, `interactive_Cell_DCH_Cell_FACH()`, `interactive_Cell_FACH_Cell_DCH()`, `interactive_channelshift()`, `interactive_nodeb_info()`, `ovsf_log_file`, `start_time`, `stat_poisson()`, and `ue_array`.

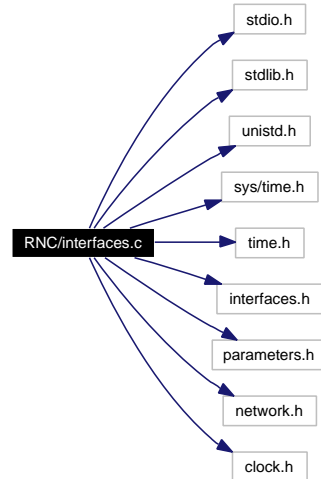
Referenced by `main()`.

Here is the call graph for this function:



## 4.15 RNC/interfaces.c File Reference

Include dependency graph for interfaces.c:



### Functions

- void [interfaces\\_creation](#) ()
- void [interfaces\\_destruction](#) ()

#### 4.15.1 Function Documentation

##### 4.15.1.1 void interfaces\_creation ()

This procedure creates the virtual interfaces and the TG servers on the RNC and the UEs.

Definition at line 32 of file interfaces.c.

References `actual_time`, and `network_send`().

Referenced by `action_analysis`(), and `initialisation_management`().

Here is the call graph for this function:



##### 4.15.1.2 void interfaces\_destruction ()

This procedure disables the virtual interfaces and the TG servers on the RNC and the UEs.

Definition at line 212 of file interfaces.c.

References `network_send`().

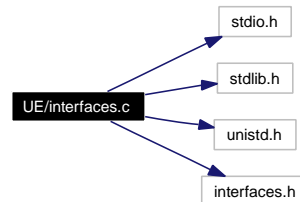
Referenced by `action_analysis`(), and `finalisation_management`().

Here is the call graph for this function:



## 4.16 UE/interfaces.c File Reference

Include dependency graph for interfaces.c:



### Functions

- void [interfaces\\_creation](#) (int number\_interfaces, char \*ip\_address)
- void [interfaces\\_destruction](#) (int number\_interfaces, char \*ip\_address)

#### 4.16.1 Function Documentation

##### 4.16.1.1 void `interfaces_creation` (int *number\_interfaces*, char \* *ip\_address*)

This procedure creates the virtual interfaces and the TG servers on the UEs.

**Parameters:**

*number\_interfaces* represents the number of virtual interfaces to create,

*ip\_address* represents the physical IP address of the computer emulating the UEs.

Definition at line 30 of file interfaces.c.

##### 4.16.1.2 void `interfaces_destruction` (int *number\_interfaces*, char \* *ip\_address*)

This procedure disables the virtual interfaces and the TG servers on the UEs.

**Parameters:**

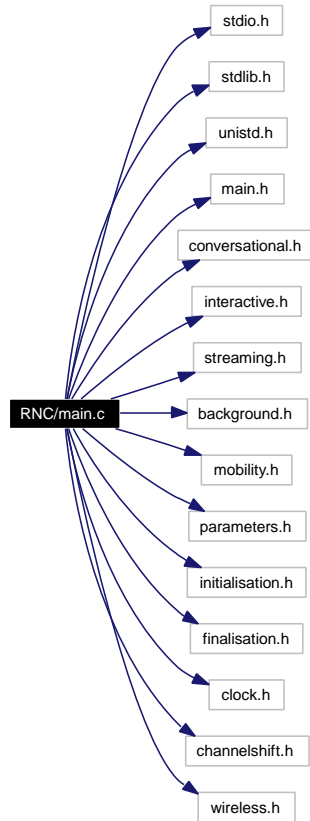
*number\_interfaces* represents the number of virtual interfaces to disable,

*ip\_address* represents the physical IP address of the computer emulating the UEs.

Definition at line 121 of file interfaces.c.

## 4.17 RNC/main.c File Reference

Include dependency graph for main.c:



### Functions

- int [main](#) ()

### Variables

- ue\_t [ue\\_array](#) [400]

#### 4.17.1 Function Documentation

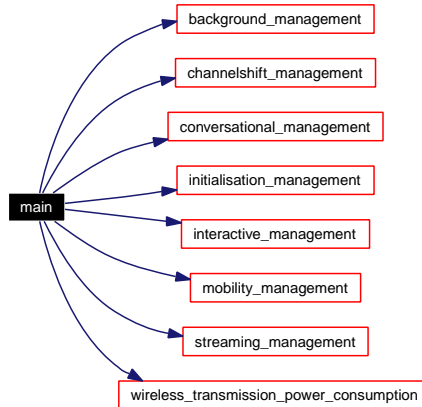
##### 4.17.1.1 int main ()

The main procedure launches a series of procedures corresponding to the traffic generated by a UE or its mobility, etc.

Definition at line 46 of file main.c.

References [actual\\_time](#), [background\\_management\(\)](#), [channelshift\\_management\(\)](#), [conversational\\_management\(\)](#), [end\\_time](#), [initialisation\\_management\(\)](#), [interactive\\_management\(\)](#), [mobility\\_management\(\)](#), [streaming\\_management\(\)](#), and [wireless\\_transmission\\_power\\_consumption\(\)](#).

Here is the call graph for this function:



## 4.17.2 Variable Documentation

### 4.17.2.1 ue\_t ue\_array[400]

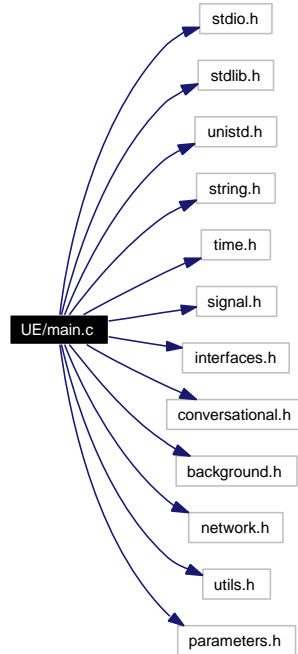
Global variable containing all the information about the UEs (ip, activity, user profile, etc.).

Definition at line 39 of file main.c.

Referenced by background\_activity\_start(), background\_cac\_select(), background\_Cell\_DCH\_Cell\_FACH(), background\_Cell\_FACH\_Cell\_DCH(), background\_channelshift(), background\_downswitch\_user(), background\_maj\_input\_file(), background\_management(), background\_nodeb\_info(), background\_rnc\_to\_ue(), background\_ue\_to\_rnc(), background\_ue\_to\_rnc\_launch(), cac\_downswitch\_pre\_emptive\_users(), cac\_get\_DSCH\_sf(), cac\_get\_user\_to\_downswitch(), cac\_init(), cac\_ovsf\_place\_available(), cac\_remove\_sf(), cac\_reorganise\_ovsf\_tree\_rec(), cac\_select\_sf(), conversational\_activity\_start(), conversational\_cac\_select(), conversational\_Cell\_DCH\_Cell\_FACH(), conversational\_Cell\_FACH\_Cell\_DCH(), conversational\_drop\_user(), conversational\_management(), conversational\_nodeb\_info(), conversational\_rnc\_to\_ue(), conversational\_ue\_to\_rnc(), conversational\_ue\_to\_rnc\_launch(), handover\_activity\_shifting(), handover\_cac(), handover\_maj\_back\_input\_file(), handover\_maj\_conv\_input\_file(), handover\_maj\_inter\_input\_file(), handover\_maj\_stream\_input\_file(), handover\_maj\_tg\_input\_files(), handover\_maj\_up\_input\_file(), handover\_management(), initialisation\_ip\_init(), initialisation\_ue\_desynchro(), interactive\_activity\_start(), interactive\_cac\_select(), interactive\_Cell\_DCH\_Cell\_FACH(), interactive\_Cell\_FACH\_Cell\_DCH(), interactive\_channelshift(), interactive\_downswitch\_user(), interactive\_maj\_input\_file(), interactive\_management(), interactive\_nodeb\_info(), mobility\_bler\_evolution(), mobility\_init(), mobility\_management(), streaming\_activity\_start(), streaming\_cac\_select(), streaming\_Cell\_DCH\_Cell\_FACH(), streaming\_Cell\_FACH\_Cell\_DCH(), streaming\_drop\_user(), streaming\_management(), streaming\_nodeb\_info(), utils\_get\_ip\_port(), utils\_ip\_present(), wireless\_bler\_computation(), wireless\_decrease\_transmission\_power(), wireless\_DSCH\_EbN0\_computation(), wireless\_FACH\_EbN0\_computation(), wireless\_transmission\_power\_consumption\_Sector(), and wireless\_transmission\_power\_consumption\_UE().

## 4.18 UE/main.c File Reference

Include dependency graph for main.c:



### Functions

- void [action\\_analysis](#) (char \*buffer)
- int [main](#) (int argc, char \*argv[ ])

### 4.18.1 Function Documentation

#### 4.18.1.1 void [action\\_analysis](#) (char \* *buffer*)

This procedure analyses the messages received by the UEs. The messages may have various formats:

- "ic num ip": message corresponding to the virtual Interfaces Creation:
  1. num is the number of virtual interfaces to create, and
  2. ip is the physical IP address of the computer emulating the UEs.
- "id num ip": message corresponding to the virtual Interfaces Destruction:
  1. num is the number of virtual interfaces to disable, and
  2. ip is the physical IP address of the computer emulating the UEs.
- "cc num port ip": message corresponding to the Conversational TG input file Creation:
  1. num is the conversational session length,
  2. port is the destination port to reach (RNC), and

3. ip is the source IP address (UE).
- "cl ip": message corresponding to the Conversational traffic Launch:
    1. ip is the source IP address (UE).
  - "bc num port ip": message corresponding to the Background TG input file Creation:
    1. num is the maximal background session length,
    2. port is the destination port to reach (RNC), and
    3. ip' is the source IP address (UE).
  - "bl ip": message corresponding to the Background traffic Launch:
    1. ip is the source IP address (UE).
  - "cs ip": message corresponding to the Conversational traffic Stop:
    1. ip is the source IP address (UE).
  - "bs ip": message corresponding to the Background traffic Stop:
    1. ip is the source IP address (UE).
  - "rf num ip": message corresponding to a querie from the RNC asking the UE to send it a file over the network.
    1. tc is the traffic class of the corresponding file, and
    2. ip is the IP address of the corresponding UE.
  - "sf num ip": message corresponding to a querie from the RNC asking the UE to read a file over the network.
    1. tc is the traffic class of the corresponding file, and
    2. ip is the IP address of the corresponding UE.

**Parameters:**

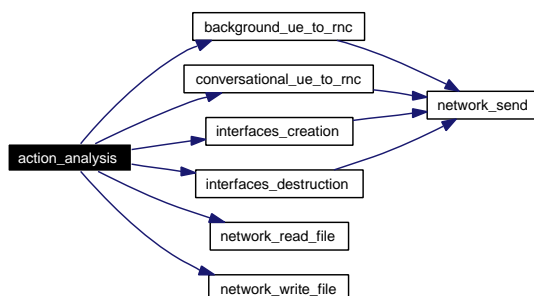
*buffer* represents the arriving message.

Definition at line 65 of file main.c.

References `background_ue_to_rnc()`, `conversational_ue_to_rnc()`, `interfaces_creation()`, `interfaces_destruction()`, `network_read_file()`, and `network_write_file()`.

Referenced by `main()`.

Here is the call graph for this function:



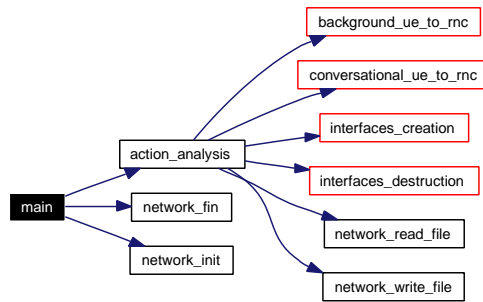
#### 4.18.1.2 `int main (int argc, char * argv[])`

The main procedure initialises the UEs and always listen to the RNC exchange socket.

Definition at line 139 of file main.c.

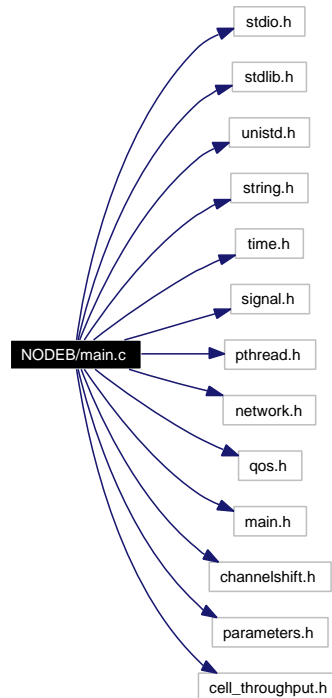
References `action_analysis()`, `network_fin()`, `network_init()`, and `sock_from_rnc`.

Here is the call graph for this function:



## 4.19 NODEB/main.c File Reference

Include dependency graph for main.c:



### Functions

- void [action\\_analysis](#) (char \*buffer)
- int [main](#) (int argc, char \*argv[ ])

### Variables

- int [simulation\\_end](#)

#### 4.19.1 Function Documentation

##### 4.19.1.1 void [action\\_analysis](#) (char \* *buffer*)

This procedure analyses the messages received by the NodeBs. The messages may have various formats:

- "b ip up cu dt tc sf se bler": message corresponding to the creation of a new tc class and filter:
  1. ip represents the source IPv6 address of the traffic to filter,
  2. up represents the user profile of the source UE of the traffic (1: Platinum, 2: Gold, 3: Silver and 4: Bronze),
  3. cu represents the channel used to transport the traffic (1: DCH, 2: DSCH and 3: FACH),
  4. dt represents the value to which the DCH holding inactivity timer as to be initialised,

5. tc represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),
  6. sf represents the spreading factor allocated to the source UE of the traffic,
  7. se represents the sector of the NodeB where the UE is located, and
  8. bler represents the BLER the transport channel undergoes.
- "s ip cu tc se ha": message corresponding to the destruction of a tc class and filter:
    1. ip represents the source IPv6 address of the traffic to filter,
    2. cu represents the channel used to transport the traffic (1: DCH, 2: DSCH and 3: FACH),
    3. tc represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),
    4. se represents the sector of the NodeB where the UE is located, and
    5. ha represents the fact that the destruction of the tc class and filter is due to an handover (1) or not (0).
  - "p ip tc se pre": message corresponding to a query from the RNC asking to move a UE form non pre-emptive to pre-emptive mode (or the opposite) still staying on a DCH.
    1. ip represents the source IPv6 address of the traffic to filter,
    2. tc represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),
    3. se represents the sector where the UE is located, and
    4. pre 1 : non pre-emptive -> pre-emptive, 0: pre-emptive -> non pre-emptive.
  - "u ip cu tc se bler": message corresponding to the update of the transport channel BLER.
    1. ip represents the source IPv6 address of the traffic to filter,
    2. cu represents the channel used to transport the traffic (1: DCH, 2: DSCH and 3: FACH),
    3. tc represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),
    4. se represents the sector of the NodeB where the UE is located, and
    5. bler represents the BLER the transport channel undergoes.
  - "a": message corresponding to a query from the RNC asking the changes to operate in channel allocation (up/downswitch).

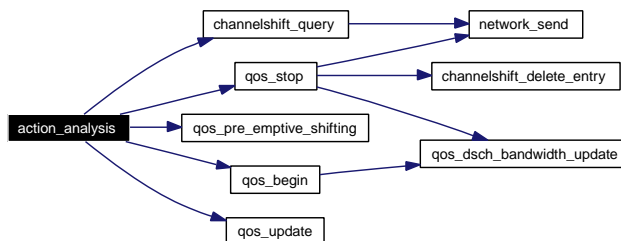
#### Parameters:

*buffer* represents the arriving message.

Definition at line 76 of file main.c.

References channelshift\_query(), qos\_begin(), qos\_pre\_emptive\_shifting(), qos\_stop(), and qos\_update().

Here is the call graph for this function:



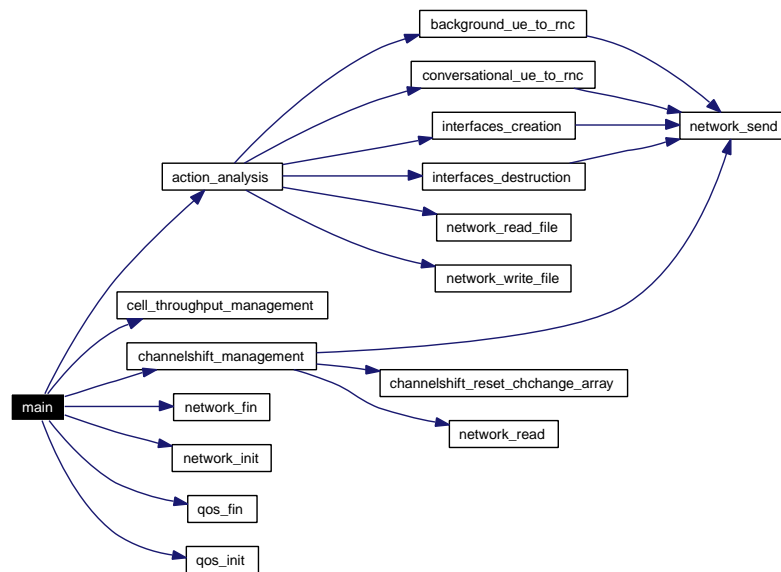
### 4.19.1.2 int main (int argc, char \* argv[])

The main procedure initialises the NodeBs and always listen to the RNC exchange socket.

Definition at line 117 of file main.c.

References `action_analysis()`, `cell_throughput_management()`, `channelshift_management()`, `network_fin()`, `network_init()`, `qos_fin()`, `qos_init()`, `simulation_end`, and `sock_from_rnc`.

Here is the call graph for this function:



## 4.19.2 Variable Documentation

### 4.19.2.1 int simulation\_end

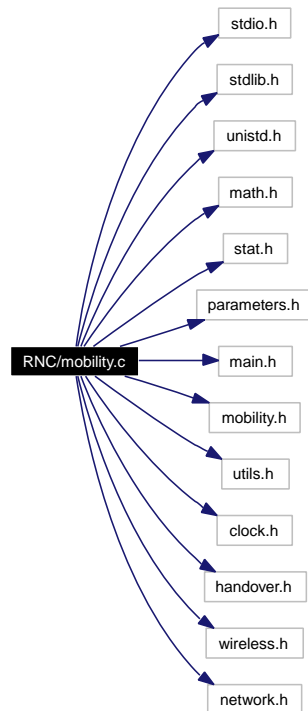
This variable is initialised to 0 at the beginning of the simulation and keep this value during it. At the end of the simulation, the variable equals 1.

Definition at line 36 of file main.c.

Referenced by `cell_throughput_management()`, `channelshift_management()`, and `main()`.

## 4.20 RNC/mobility.c File Reference

Include dependency graph for mobility.c:



### Functions

- `coo_t mobility_diag_A` (double x, double y, `coo_t NodeB_1`, `coo_t NodeB_2`)
- `coo_t mobility_diag_B` (double x, double y, `coo_t NodeB_1`, `coo_t NodeB_2`)
- `coo_t mobility_territory_identification` (double x, double y)
- void `mobility_init` ()
- double `mobility_gain_angle_attenuation` (double angle)
- double `mobility_angle_NodeB` (double x, double y, double distance, int NodeB\_id)
- double `mobility_distance_NodeB` (double x, double y, int NodeB\_id)
- double `mobility_reception` (double x, double y, int NodeB, char Sector)
- double `mobility_interferences` (double x, double y, int target\_NodeB, char target\_Sector)
- double `mobility_noise_computation` ()
- `Ec_I0_info_t mobility_Ec_I0_computation` (double x, double y, int NodeB\_id, char Sector\_id)
- void `mobility_bler_evolution` (int ue\_id)
- void `mobility_management` (int ue\_id)

### Variables

- FILE \* `mob_file_array` [400]

## 4.20.1 Function Documentation

### 4.20.1.1 `coo_t mobility_diag_A (double x, double y, coo_t NodeB_1, coo_t NodeB_2)`

Contains a subgroup of the territory identification computation (/).

Definition at line 43 of file mobility.c.

Referenced by `mobility_territory_identification()`.

### 4.20.1.2 `coo_t mobility_diag_B (double x, double y, coo_t NodeB_1, coo_t NodeB_2)`

Contains a subgroup of the territory identification computation (\).

Definition at line 54 of file mobility.c.

Referenced by `mobility_territory_identification()`.

### 4.20.1.3 `coo_t mobility_territory_identification (double x, double y)`

This procedure determines on which NodeB territory a UE is placed.

#### Parameters:

*x* represents the X coordinate of the UE,

*y* represents the Y coordinate of the UE.

#### Returns:

a structure containing the NodeB and Sector identification the UE is on.

Definition at line 71 of file mobility.c.

References `mobility_diag_A()`, and `mobility_diag_B()`.

Here is the call graph for this function:



### 4.20.1.4 `void mobility_init ()`

This procedure initialises the mobility management of the UE (coordinates, angle, speed, etc.).

Definition at line 414 of file mobility.c.

References `ue_array`, and `utils_arrond()`.

Referenced by `initialisation_management()`.

Here is the call graph for this function:



#### 4.20.1.5 double mobility\_gain\_angle\_attenuation (double *angle*)

This procedure computes the attenuation a signal emitted by a NodeB undergoes based on the UE position.

**Parameters:**

*angle* represents the angle between the NodeB and the UE.

**Returns:**

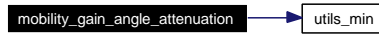
the angle attenuation of the signal.

Definition at line 547 of file mobility.c.

References `utils_min()`.

Referenced by `mobility_reception()`.

Here is the call graph for this function:



#### 4.20.1.6 double mobility\_angle\_NodeB (double *x*, double *y*, double *distance*, int *NodeB\_id*)

This procedure computes the angle existing between an UE position and a NodeB ([0:360], East=0).

**Parameters:**

*x* represents the X coordinate of the UE,

*y* represents the Y coordinate of the UE,

*distance* represents the distance separating the UE from the NodeB,

*NodeB\_id* represents the identification number of the NodeB ([0:3]).

**Returns:**

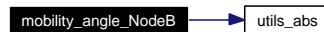
the angle between the NodeB and the UE.

Definition at line 564 of file mobility.c.

References `utils_abs()`.

Referenced by `mobility_reception()`, and `wireless_path_loss_and_attenuation()`.

Here is the call graph for this function:



#### 4.20.1.7 double mobility\_distance\_NodeB (double *x*, double *y*, int *NodeB\_id*)

This procedure computes the distance between an UE and its serving NodeB.

**Parameters:**

*x* represents the X coordinate of the UE,

*y* represents the Y coordinate of the UE,

*NodeB\_id* represents the identification number of the NodeB ([0;3]),

**Returns:**

the distance expressed in meters separating the UE and the NodeB.

Definition at line 660 of file mobility.c.

Referenced by mobility\_reception(), and wireless\_path\_loss\_and\_attenuation().

#### 4.20.1.8 double mobility\_reception (double x, double y, int NodeB, char Sector)

This procedure linearly computes the power of the signal received by a UE.

**Parameters:**

*x* represents the X coordinate of the UE,

*y* represents the Y coordinate of the UE,

*NodeB* represents the identification number of the NodeB ([0;3]),

*Sector* represents the identification character of the sector ('a', 'b' or 'c').

**Returns:**

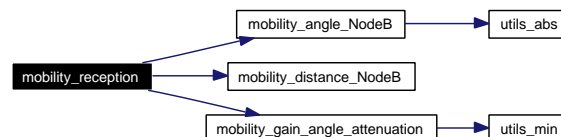
the power of the signal received by a UE.

Definition at line 748 of file mobility.c.

References mobility\_angle\_NodeB(), mobility\_distance\_NodeB(), and mobility\_gain\_angle\_attenuation().

Referenced by mobility\_Ec\_I0\_computation(), and mobility\_interferences().

Here is the call graph for this function:



#### 4.20.1.9 double mobility\_interferences (double x, double y, int target\_NodeB, char target\_Sector)

This procedure linearly computes the total interferences received by a UE.

**Parameters:**

*x* represents the X coordinate of the UE,

*y* represents the Y coordinate of the UE,

*target\_NodeB* represents the identification number of the NodeB ([0;3]),

*target\_Sector* represents the identification character of the sector ('a', 'b' or 'c').

**Returns:**

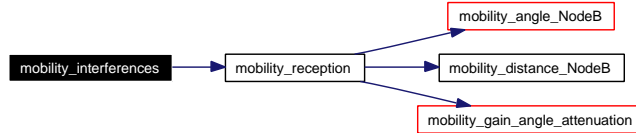
the total interferences received by a UE.

Definition at line 813 of file mobility.c.

References mobility\_reception().

Referenced by mobility\_Ec\_I0\_computation().

Here is the call graph for this function:



#### 4.20.1.10 double mobility\_noise\_computation ()

This procedure returns the noise linear computation of the emulation.

Definition at line 844 of file mobility.c.

Referenced by mobility\_Ec\_I0\_computation().

#### 4.20.1.11 Ec\_I0\_info\_t mobility\_Ec\_I0\_computation (double x, double y, int NodeB\_id, char Sector\_id)

This procedure determines the highest Ec/I0 (signal/interferences) a UE is capable to received based on its position.

##### Parameters:

*x* represents the X coordinate of the UE,

*y* represents the Y coordinate of the UE,

*NodeB\_id* represents the identification number of the NodeB ([0;3]),

*Sector\_id* represents the identification character of the sector ('a', 'b' or 'c').

##### Returns:

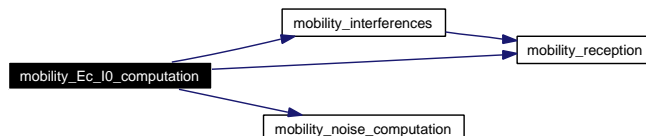
a data structure containing the highest Ec/I0 receivable and the serving NodeB and Sector.

Definition at line 863 of file mobility.c.

References mobility\_interferences(), mobility\_noise\_computation(), and mobility\_reception().

Referenced by mobility\_management().

Here is the call graph for this function:



#### 4.20.1.12 void mobility\_bler\_evolution (int ue\_id)

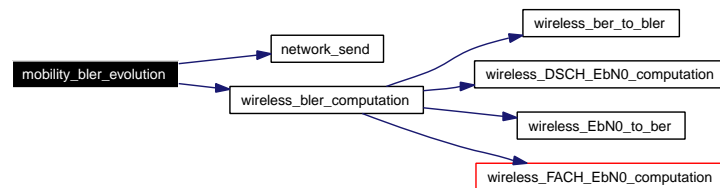
This procedure computes the BLER evolution of a communication and alerts the corresponding NodeB.

Definition at line 921 of file mobility.c.

References network\_send(), ue\_array, and wireless\_bler\_computation().

Referenced by mobility\_management().

Here is the call graph for this function:



#### 4.20.1.13 void mobility\_management (int ue\_id)

This procedure generates the UE mobility graphs following a random walk algorithm. It decides also when an handover has to occur.

##### Parameters:

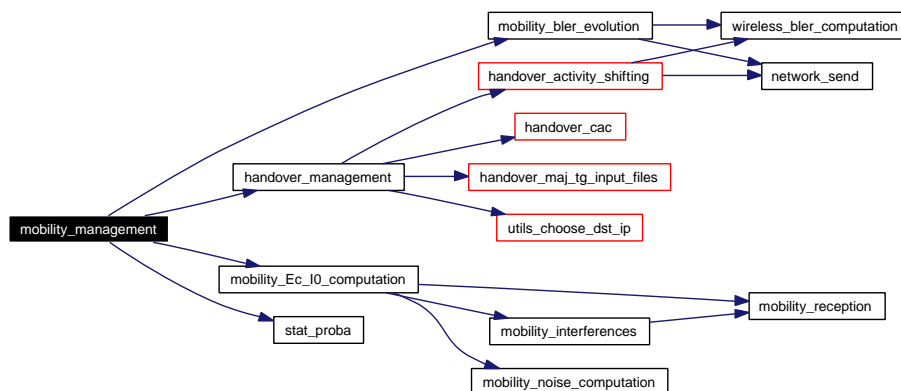
*ue\_id* represents the identification number of the UE.

Definition at line 960 of file mobility.c.

References actual\_time, handover\_management(), init\_desynchro, mob\_file\_array, mobility\_bler\_evolution(), mobility\_Ec\_I0\_computation(), start\_time, stat\_proba(), and ue\_array.

Referenced by main().

Here is the call graph for this function:



## 4.20.2 Variable Documentation

### 4.20.2.1 FILE\* `mob_file_array`[400]

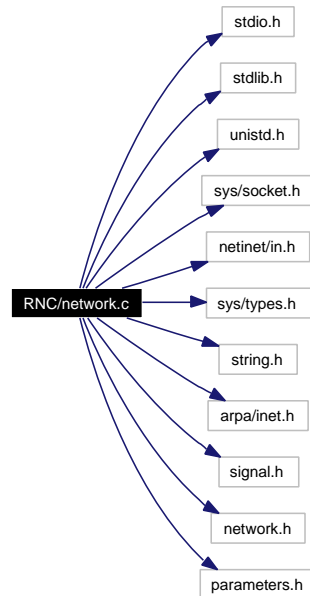
Global variable representing the files where the mobility logs are written.

Definition at line 36 of file `mobility.c`.

Referenced by `finalisation_management()`, and `mobility_management()`.

## 4.21 RNC/network.c File Reference

Include dependency graph for network.c:



### Functions

- void [network\\_init](#) ()
- void [network\\_send](#) (int dst, char \*message)
- char \* [network\\_read](#) (int dst)
- void [network\\_read\\_file](#) (int src, char \*file\_name)
- void [network\\_write\\_file](#) (int dst, char \*file\_name)
- void [network\\_fin](#) ()

### Variables

- int [local\\_sock](#)
- int [sock\\_to\\_1](#)
- int [sock\\_to\\_2](#)
- int [sock\\_to\\_3](#)
- int [sock\\_to\\_4](#)
- int [sock\\_to\\_nb1](#)
- int [sock\\_to\\_nb2](#)
- int [sock\\_to\\_nb3](#)
- int [sock\\_to\\_nb4](#)

### 4.21.1 Function Documentation

#### 4.21.1.1 void [network\\_init](#) ()

This procedure initialises the exchange sockets between the RNC, the UEs and the NodeBs.

Definition at line 73 of file network.c.

References sock\_to\_1, sock\_to\_2, sock\_to\_3, sock\_to\_4, sock\_to\_nb1, sock\_to\_nb2, sock\_to\_nb3, and sock\_to\_nb4.

Referenced by initialisation\_management(), and main().

#### 4.21.1.2 void network\_send (int dst, char \* message)

This procedure sends messages to the UEs or the NodeBs using the network.

##### Parameters:

*dst* represents the UE group destination ([1;8]),

*message* represents to message to be sent.

Definition at line 188 of file network.c.

References sock\_to\_1, sock\_to\_2, sock\_to\_3, sock\_to\_4, sock\_to\_nb1, sock\_to\_nb2, sock\_to\_nb3, and sock\_to\_nb4.

Referenced by background\_channelshift(), background\_nodeb\_info(), background\_ue\_to\_rnc(), background\_ue\_to\_rnc\_launch(), channelshift\_management(), channelshift\_query(), conversational\_nodeb\_info(), conversational\_ue\_to\_rnc(), conversational\_ue\_to\_rnc\_launch(), handover\_activity\_shifting(), handover\_maj\_up\_input\_file(), interactive\_channelshift(), interactive\_nodeb\_info(), interfaces\_creation(), interfaces\_destruction(), mobility\_bler\_evolution(), qos\_stop(), and streaming\_nodeb\_info().

#### 4.21.1.3 char\* network\_read (int dst)

This procedure analyses the messages received by the RNC. The message has only one format.

##### Parameters:

*dst* represents the UE group source ([1;8]).

##### Returns:

the message received.

Definition at line 281 of file network.c.

References sock\_to\_1, sock\_to\_2, sock\_to\_3, sock\_to\_4, sock\_to\_nb1, sock\_to\_nb2, sock\_to\_nb3, and sock\_to\_nb4.

Referenced by channelshift\_management(), and handover\_activity\_shifting().

#### 4.21.1.4 void network\_read\_file (int src, char \* file\_name)

This procedure reads a complete text file from the ue using the network.

##### Parameters:

*file\_name* represents the file name.

*src* represents the UE source of the file.

Definition at line 375 of file network.c.

References sock\_to\_1, sock\_to\_2, sock\_to\_3, and sock\_to\_4.

Referenced by action\_analysis(), and handover\_maj\_up\_input\_file().

#### 4.21.1.5 void network\_write\_file (int dst, char \*file\_name)

This procedure sends a complete text file to the UE using the network.

**Parameters:**

*file\_name* represents the file name.

Definition at line 428 of file network.c.

References sock\_to\_1, sock\_to\_2, sock\_to\_3, and sock\_to\_4.

Referenced by action\_analysis(), and handover\_maj\_up\_input\_file().

#### 4.21.1.6 void network\_fin ()

This procedure closes the exchange socket between the RNC, the UEs and the NodeBs.

Definition at line 527 of file network.c.

References local\_sock, sock\_to\_1, sock\_to\_2, sock\_to\_3, sock\_to\_4, sock\_to\_nb1, sock\_to\_nb2, sock\_to\_nb3, and sock\_to\_nb4.

Referenced by finalisation\_management(), and main().

### 4.21.2 Variable Documentation

#### 4.21.2.1 int local\_sock

Represents the local part of the exchange socket between the RNC and the UEs.

Definition at line 34 of file network.c.

Referenced by network\_fin(), and network\_init().

#### 4.21.2.2 int sock\_to\_1

Represents the distant part of the exchange socket between the RNC and the UEs emulated on the computer having the physical IP address 10.0.1.3.

Definition at line 38 of file network.c.

Referenced by network\_fin(), network\_init(), network\_read(), network\_read\_file(), network\_send(), and network\_write\_file().

#### 4.21.2.3 int sock\_to\_2

Represents the distant part of the exchange socket between the RNC and the UEs emulated on the computer having the physical IP address 10.0.2.2.

Definition at line 42 of file network.c.

Referenced by network\_fin(), network\_init(), network\_read(), network\_read\_file(), network\_send(), and network\_write\_file().

**4.21.2.4 int sock\_to\_3**

Represents the distant part of the exchange socket between the RNC and the UEs emulated on the computer having the physical IP address 10.0.3.3.

Definition at line 46 of file network.c.

Referenced by network\_fin(), network\_init(), network\_read(), network\_read\_file(), network\_send(), and network\_write\_file().

**4.21.2.5 int sock\_to\_4**

Represents the distant part of the exchange socket between the RNC and the UEs emulated on the computer having the physical IP address 10.0.4.2.

Definition at line 50 of file network.c.

Referenced by network\_fin(), network\_init(), network\_read(), network\_read\_file(), network\_send(), and network\_write\_file().

**4.21.2.6 int sock\_to\_nb1**

Represents the distant part of the exchange socket between the RNC and the NodeB emulated on the computer having the physical IP address 10.0.0.1.

Definition at line 54 of file network.c.

Referenced by network\_fin(), network\_init(), network\_read(), and network\_send().

**4.21.2.7 int sock\_to\_nb2**

Represents the distant part of the exchange socket between the RNC and the NodeB emulated on the computer having the physical IP address 10.0.0.2.

Definition at line 58 of file network.c.

Referenced by network\_fin(), network\_init(), network\_read(), and network\_send().

**4.21.2.8 int sock\_to\_nb3**

Represents the distant part of the exchange socket between the RNC and the NodeB emulated on the computer having the physical IP address 10.0.0.3.

Definition at line 62 of file network.c.

Referenced by network\_fin(), network\_init(), network\_read(), and network\_send().

**4.21.2.9 int sock\_to\_nb4**

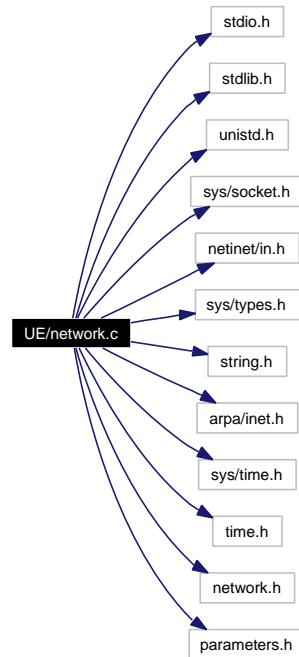
Represents the distant part of the exchange socket between the RNC and the NodeB emulated on the computer having the physical IP address 10.0.0.4.

Definition at line 66 of file network.c.

Referenced by network\_fin(), network\_init(), network\_read(), and network\_send().

## 4.22 UE/network.c File Reference

Include dependency graph for network.c:



### Functions

- void [network\\_init](#) (char \*ip)
- void [network\\_send](#) (char \*message)
- void [network\\_read\\_file](#) (char \*file\_name)
- void [network\\_write\\_file](#) (char \*file\_name)
- void [network\\_fin](#) ()

### Variables

- int [local\\_sock](#)
- int [sock\\_from\\_rnc](#)

#### 4.22.1 Function Documentation

##### 4.22.1.1 void [network\\_init](#) (char \* *ip*)

This procedure initialises the exchange socket between the RNC and the UEs.

#### Parameters:

*ip* represents the physical IP address of the computer emulating the UEs.

Definition at line 48 of file network.c.

References [local\\_sock](#), and [sock\\_from\\_rnc](#).

#### 4.22.1.2 void network\_send (char \* *message*)

This procedure sends messages to the RNC using the network.

**Parameters:**

*message* represents to message to be sent.

Definition at line 88 of file network.c.

References sock\_from\_rnc.

#### 4.22.1.3 void network\_read\_file (char \* *file\_name*)

This procedure reads a complete text file from the RNC using the network.

**Parameters:**

*file\_name* represents the file name.

Definition at line 110 of file network.c.

References sock\_from\_rnc.

#### 4.22.1.4 void network\_write\_file (char \* *file\_name*)

This procedure sens a complete text file to the RNC using the network.

**Parameters:**

*file\_name* represents the file name.

Definition at line 133 of file network.c.

References sock\_from\_rnc.

#### 4.22.1.5 void network\_fin ()

This procedure closes the exchange socket between the RNC and the UEs.

Definition at line 166 of file network.c.

References local\_sock, and sock\_from\_rnc.

### 4.22.2 Variable Documentation

#### 4.22.2.1 int local\_sock

Represents the local part of the exchange socket between the RNC and the UEs.

Definition at line 35 of file network.c.

#### 4.22.2.2 int [sock\\_from\\_rnc](#)

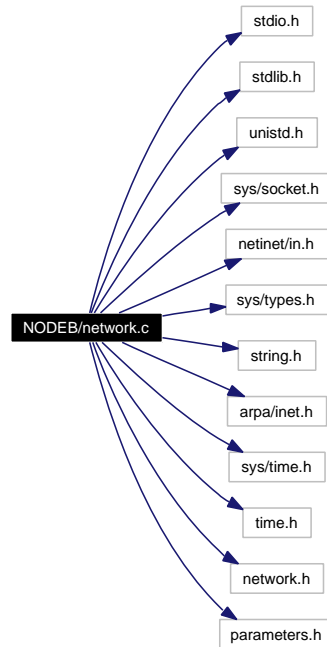
Represents the distant part of the exchange socket between the RNC and the UEs.

Definition at line 39 of file network.c.

Referenced by [main\(\)](#), [network\\_fin\(\)](#), [network\\_init\(\)](#), [network\\_read\\_file\(\)](#), [network\\_send\(\)](#), and [network\\_write\\_file\(\)](#).

## 4.23 NODEB/network.c File Reference

Include dependency graph for network.c:



### Functions

- void [network\\_init](#) (char \*ip)
- void [network\\_send](#) (char \*message)
- void [network\\_fin](#) ()

### Variables

- int [local\\_sock](#)
- int [sock\\_from\\_rnc](#)

### 4.23.1 Function Documentation

#### 4.23.1.1 void [network\\_init](#) (char \* ip)

This procedure initialises the exchange socket between the RNC and the NodeBs.

#### Parameters:

*ip* represents the physical IP address of the computer emulating the NodeBs.

Definition at line 48 of file network.c.

References [local\\_sock](#), and [sock\\_from\\_rnc](#).

#### 4.23.1.2 void network\_send (char \* *message*)

This procedure sends messages to the RNC using the network.

**Parameters:**

*message* represents to message to be sent.

Definition at line 88 of file network.c.

References sock\_from\_rnc.

#### 4.23.1.3 void network\_fin ()

This procedure closes the exchange socket between the RNC and the NodeBs.

Definition at line 108 of file network.c.

References local\_sock, and sock\_from\_rnc.

### 4.23.2 Variable Documentation

#### 4.23.2.1 int [local\\_sock](#)

Represents the local part of the exchange socket between the RNC and the NodeBs.

Definition at line 35 of file network.c.

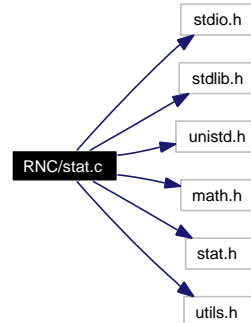
#### 4.23.2.2 int [sock\\_from\\_rnc](#)

Represents the distant part of the exchange socket between the RNC and the NodeBs.

Definition at line 39 of file network.c.

## 4.24 RNC/stat.c File Reference

Include dependency graph for stat.c:



### Functions

- double [stat\\_proba](#) ()
- double [stat\\_gamma](#) (double a, double b)
- int [stat\\_poisson](#) (double mu)
- double [stat\\_gaussian](#) (double sigma, double mu)
- double [stat\\_exponential](#) (double mu, double min, double max)
- double [stat\\_pareto](#) (double a, double b, double m)
- int [stat\\_geometric](#) (double p)
- double [stat\\_lognormal](#) (double zeta, double sigma)

### 4.24.1 Function Documentation

#### 4.24.1.1 double stat\_proba ()

Returns a randomly generated double ]0;1[.

Definition at line 29 of file stat.c.

Referenced by [cac\\_init\(\)](#), [conversational\\_rnc\\_to\\_ue\(\)](#), [conversational\\_ue\\_to\\_rnc\(\)](#), [handover\\_maj\\_inter\\_input\\_file\(\)](#), [interactive\\_activity\\_start\(\)](#), [interactive\\_maj\\_input\\_file\(\)](#), [mobility\\_management\(\)](#), [stat\\_exponential\(\)](#), [stat\\_gaussian\(\)](#), [stat\\_geometric\(\)](#), [stat\\_lognormal\(\)](#), [stat\\_pareto\(\)](#), [stat\\_poisson\(\)](#), [utils\\_binomial\(\)](#), [utils\\_gamma\\_frac\(\)](#), [utils\\_gamma\\_int\(\)](#), and [utils\\_gamma\\_large\(\)](#).

#### 4.24.1.2 double stat\_gamma (double a, double b)

Returns a double randomly generated by the Gamma distribution with parameters a and b.

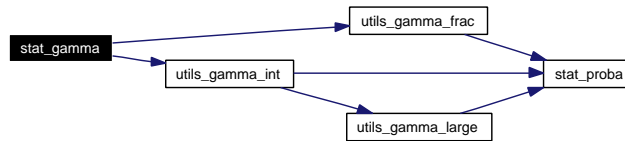
$$cdf : f(x) = \frac{1}{\Gamma(a)b^a} x^{a-1} e^{-\frac{x}{b}}, x > 0$$

Definition at line 43 of file stat.c.

References [utils\\_gamma\\_frac\(\)](#), and [utils\\_gamma\\_int\(\)](#).

Referenced by [streaming\\_activity\\_start\(\)](#), and [utils\\_beta\(\)](#).

Here is the call graph for this function:



#### 4.24.1.3 int stat\_poisson (double mu)

Returns a randomly generated integer by the Poisson distribution with parameter mu.

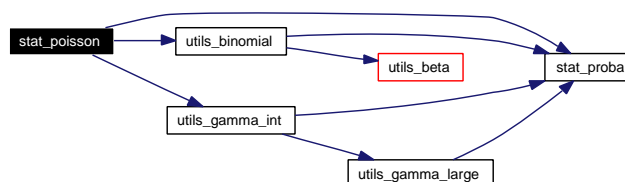
$$cdf : f(x) = \frac{\mu^x}{x!} e^{-\mu}, x \geq 0$$

Definition at line 59 of file stat.c.

References stat\_proba(), utils\_binomial(), and utils\_gamma\_int().

Referenced by background\_management(), conversational\_drop\_user(), conversational\_management(), handover\_activity\_shifting(), initialisation\_ue\_desynchro(), interactive\_management(), streaming\_drop\_user(), and streaming\_management().

Here is the call graph for this function:



#### 4.24.1.4 double stat\_gaussian (double sigma, double mu)

Returns a randomly generated double by the Gaussian distribution with parameters sigma and mu.

$$cdf : f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

Definition at line 97 of file stat.c.

References stat\_proba().

Here is the call graph for this function:



**4.24.1.5 double stat\_exponential (double mu, double min, double max)**

Returns a randomly generated double by the Exponential distribution with parameters mu, min and max.

$$cdf : f(x) = \frac{1}{\mu} e^{-\frac{x}{\mu}}, x \geq 0$$

Definition at line 119 of file stat.c.

References stat\_proba().

Referenced by conversational\_activity\_start(), interactive\_activity\_start(), and streaming\_activity\_start().

Here is the call graph for this function:

**4.24.1.6 double stat\_pareto (double a, double b, double m)**

Returns a randomly generated double by the Pareto distribution with parameters a, b and m.

$$cdf : f(x) = \frac{a}{b} \left(\frac{x}{b}\right)^{-a-1}, x \geq b$$

Definition at line 138 of file stat.c.

References stat\_proba().

Referenced by background\_ue\_to\_rmc().

Here is the call graph for this function:

**4.24.1.7 int stat\_geometric (double p)**

Returns a randomly generated integer by the Geometric distribution with parameter p.

$$cdf : f(x) = p(1 - p)^{x-1}, x \geq 1$$

Definition at line 159 of file stat.c.

References stat\_proba().

Referenced by interactive\_activity\_start().

Here is the call graph for this function:



**4.24.1.8 double stat\_lognormal (double zeta, double sigma)**

Returns a randomly generated double by the Lognormal distribution with parameters zeta and sigma.

$$cdf : f(x) = \frac{1}{x\sqrt{2\pi\sigma^2}} e^{-\frac{(\ln(x)-\zeta)^2}{2\sigma^2}}, x > 0$$

Definition at line 175 of file stat.c.

References stat\_proba().

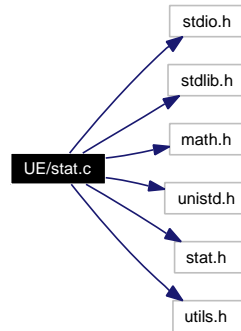
Referenced by background\_rnc\_to\_ue(), and background\_ue\_to\_rnc().

Here is the call graph for this function:



## 4.25 UE/stat.c File Reference

Include dependency graph for stat.c:



### Functions

- double [stat\\_proba](#) ()
- double [stat\\_gamma](#) (double a, double b)
- int [stat\\_poisson](#) (double mu)
- double [stat\\_gaussian](#) (double sigma, double mu)
- double [stat\\_exponential](#) (double mu, double min, double max)
- double [stat\\_pareto](#) (double a, double b, double m)
- int [stat\\_geometric](#) (double p)
- double [stat\\_lognormal](#) (double zeta, double sigma)

### 4.25.1 Function Documentation

#### 4.25.1.1 double [stat\\_proba](#) ()

Returns a randomly generated double ]0;1[.

Definition at line 29 of file stat.c.

#### 4.25.1.2 double [stat\\_gamma](#) (double *a*, double *b*)

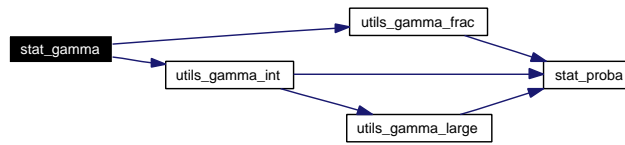
Returns a double randomly generated by the Gamma distribution with parameters *a* and *b*.

$$cdf : f(x) = \frac{1}{\Gamma(a)b^a} x^{a-1} e^{-\frac{x}{b}}, x > 0$$

Definition at line 43 of file stat.c.

References [utils\\_gamma\\_frac](#)(), and [utils\\_gamma\\_int](#)().

Here is the call graph for this function:



#### 4.25.1.3 int stat\_poisson (double mu)

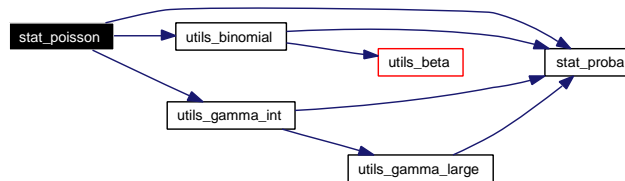
Returns a randomly generated integer by the Poisson distribution with parameter mu.

$$cdf : f(x) = \frac{\mu^x}{x!} e^{-\mu}, x \geq 0$$

Definition at line 59 of file stat.c.

References stat\_proba(), utils\_binomial(), and utils\_gamma\_int().

Here is the call graph for this function:



#### 4.25.1.4 double stat\_gaussian (double sigma, double mu)

Returns a randomly generated double by the Gaussian distribution with parameters sigma and mu.

$$cdf : f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

Definition at line 97 of file stat.c.

References stat\_proba().

Here is the call graph for this function:



#### 4.25.1.5 double stat\_exponential (double mu, double min, double max)

Returns a randomly generated double by the Exponential distribution with parameters mu, min and max.

$$cdf : f(x) = \frac{1}{\mu} e^{-\frac{x}{\mu}}, x \geq 0$$

Definition at line 119 of file stat.c.

References stat\_proba().

Here is the call graph for this function:



#### 4.25.1.6 double stat\_pareto (double a, double b, double m)

Returns a randomly generated double by the Pareto distribution with parameters a, b and m.

$$cdf : f(x) = \frac{\frac{a}{b}}{\left(\frac{x}{b}\right)^{a+1}}, x \geq b$$

Definition at line 138 of file stat.c.

References stat\_proba().

Here is the call graph for this function:



#### 4.25.1.7 int stat\_geometric (double p)

Returns a randomly generated integer by the Geometric distribution with parameter p.

$$cdf : f(x) = p(1 - p)^{x-1}, x \geq 1$$

Definition at line 159 of file stat.c.

References stat\_proba().

Here is the call graph for this function:



#### 4.25.1.8 double stat\_lognormal (double zeta, double sigma)

Returns a randomly generated double by the Lognormal distribution with parameters zeta and sigma.

$$cdf : f(x) = \frac{1}{x\sqrt{2\pi\sigma^2}} e^{-\frac{(\ln(x)-\zeta)^2}{2\sigma^2}}, x > 0$$

Definition at line 175 of file stat.c.

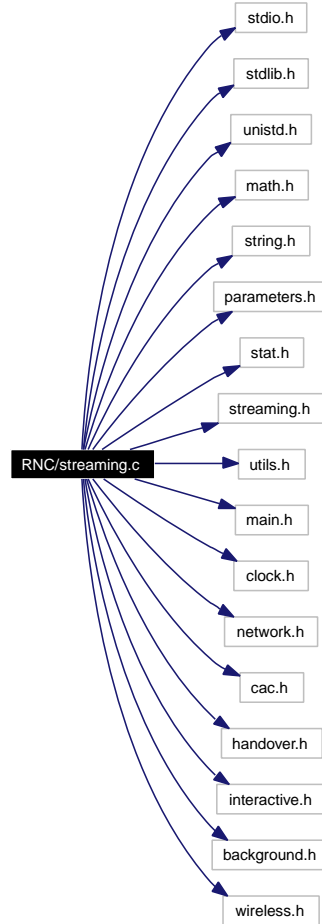
References stat\_proba().

Here is the call graph for this function:



## 4.26 RNC/streaming.c File Reference

Include dependency graph for streaming.c:



### Functions

- void [streaming\\_nodeb\\_info](#) (int ue\_id, int start)
- void [streaming\\_activity\\_start](#) (int ue\_id)
- int [streaming\\_cac\\_select](#) (int ue\_id)
- void [streaming\\_Cell\\_FACH\\_Cell\\_DCH](#) (int ue\_id)
- void [streaming\\_Cell\\_DCH\\_Cell\\_FACH](#) (int ue\_id)
- void [streaming\\_drop\\_user](#) (int ue\_id)
- void [streaming\\_management](#) (int ue\_id)

### 4.26.1 Function Documentation

#### 4.26.1.1 void [streaming\\_nodeb\\_info](#) (int ue\_id, int start)

This procedure alerts the NodeBs about the flow creation/destruction. The messages may have various formats:

- "b ip up cu dt tc sf se bler": message corresponding to the creation of a new tc class and filter:
  1. ip represents the source IPv6 address of the traffic to filter,
  2. up represents the user profile of the source UE of the traffic (1: Platinum, 2: Gold, 3: Silver and 4: Bronze),
  3. cu represents the channel used to transport the traffic (1: DCH, 2: DSCH and 3: FACH),
  4. dt represents the value to which the DCH holding inactivity timer as to be initialised,
  5. tc represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),
  6. sf represents the spreading factor allocated to the source UE of the traffic,
  7. se represents the sector of the NodeB where the UE is located, and
  8. bler represents the BLER the transport channel undergoes.
  
- "s ip cu tc se ha": message corresponding to the destruction of a tc class and filter:
  1. ip represents the source IPv6 address of the traffic to filter,
  2. cu represents the channel used to transport the traffic (1: DCH, 2: DSCH and 3: FACH),
  3. tc represents the UMTS traffic class of the traffic (1: Conversational, 2: Interactive, 3: Streaming and 4: Background),
  4. se represents the sector of the NodeB where the UE is located, and
  5. ha represents the fact that the destruction of the tc class and filter is due to an handover (1) or not (0).

**Parameters:**

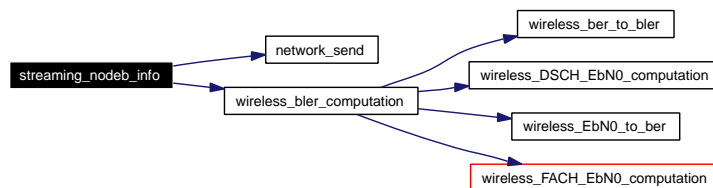
- ue\_id* represents the identification number of the UE,
- start* indicates if we are starting (1) or stopping (0) an activity.

Definition at line 59 of file streaming.c.

References network\_send(), ue\_array, and wireless\_bler\_computation().

Referenced by streaming\_activity\_start(), streaming\_drop\_user(), and streaming\_management().

Here is the call graph for this function:

**4.26.1.2 void streaming\_activity\_start (int ue\_id)**

This procedure takes care of the streaming part of the emulation, computing session and inter-session length, connection behaviours, etc. based on statistical distributions.

**Parameters:**

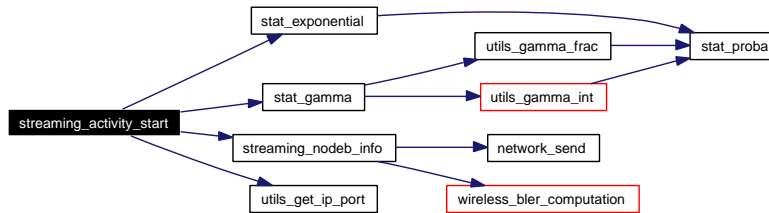
- ue\_id* represents the identification number of the UE.

Definition at line 80 of file streaming.c.

References `actual_time`, `end_time`, `init_desynchro`, `start_time`, `stat_exponential()`, `stat_gamma()`, `streaming_nodeb_info()`, `ue_array`, and `utils_get_ip_port()`.

Referenced by `streaming_management()`.

Here is the call graph for this function:



#### 4.26.1.3 int streaming\_cac\_select (int ue\_id)

This procedure tries to allocate an acceptable spreading factor (based to the user profile of the UE) to a streaming session.

##### Parameters:

*ue\_id* represents the identification number of the UE.

##### Returns:

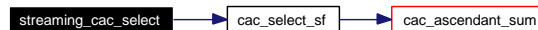
1 if the UE gets an acceptable spreading factor and 0 otherwise.

Definition at line 180 of file streaming.c.

References `cac_select_sf()`, and `ue_array`.

Referenced by `handover_cac()`, and `streaming_management()`.

Here is the call graph for this function:



#### 4.26.1.4 void streaming\_Cell\_FACH\_Cell\_DCH (int ue\_id)

This procedure upswitches only the interactive and background sessions.

##### Parameters:

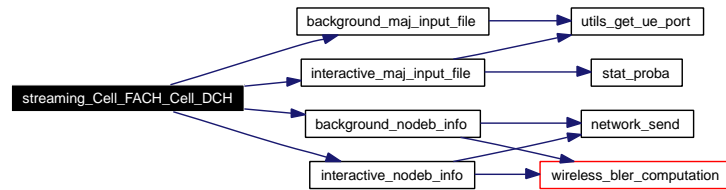
*ue\_id* represents the identification number of the UE.

Definition at line 277 of file streaming.c.

References `actual_time`, `background_maj_input_file()`, `background_nodeb_info()`, `handover_log_file`, `init_desynchro`, `interactive_maj_input_file()`, `interactive_nodeb_info()`, `start_time`, and `ue_array`.

Referenced by `streaming_management()`.

Here is the call graph for this function:



#### 4.26.1.5 void streaming\_Cell\_DCH\_Cell\_FACH (int ue\_id)

This procedure downswitches only the interactive and background sessions.

##### Parameters:

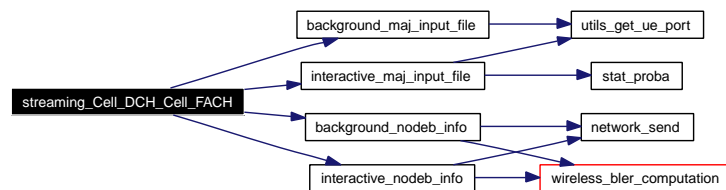
`ue_id` represents the identification number of the UE.

Definition at line 337 of file `streaming.c`.

References `actual_time`, `background_maj_input_file()`, `background_nodeb_info()`, `handover_log_file`, `init_desynchro`, `interactive_maj_input_file()`, `interactive_nodeb_info()`, `start_time`, and `ue_array`.

Referenced by `streaming_drop_user()`, and `streaming_management()`.

Here is the call graph for this function:



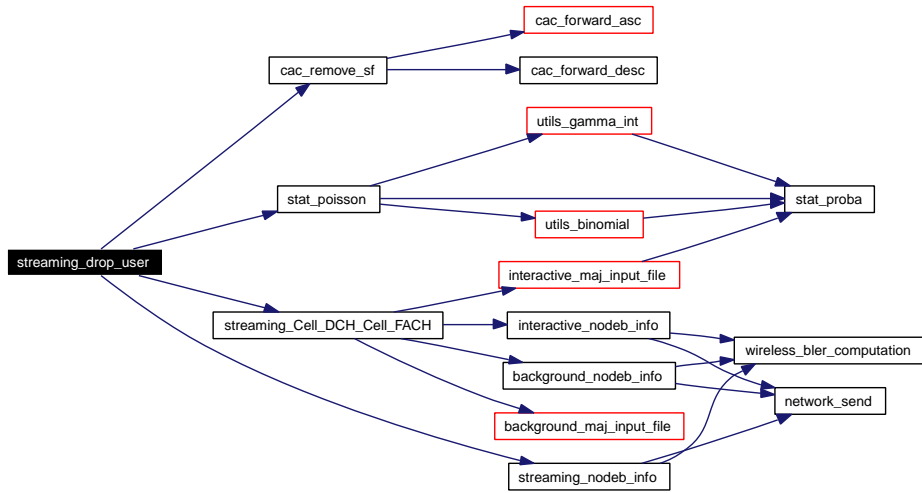
#### 4.26.1.6 void streaming\_drop\_user (int ue\_id)

This procedure drops a streaming session of a given user.

Definition at line 395 of file `streaming.c`.

References `actual_time`, `cac_remove_sf()`, `init_desynchro`, `ovsf_log_file`, `start_time`, `stat_poisson()`, `streaming_Cell_DCH_Cell_FACH()`, `streaming_nodeb_info()`, and `ue_array`.

Here is the call graph for this function:



**4.26.1.7 void streaming\_management (int ue\_id)**

This procedure checks if a UE has to start a new streaming session or not, based on the `actual_time` variable.

**Parameters:**

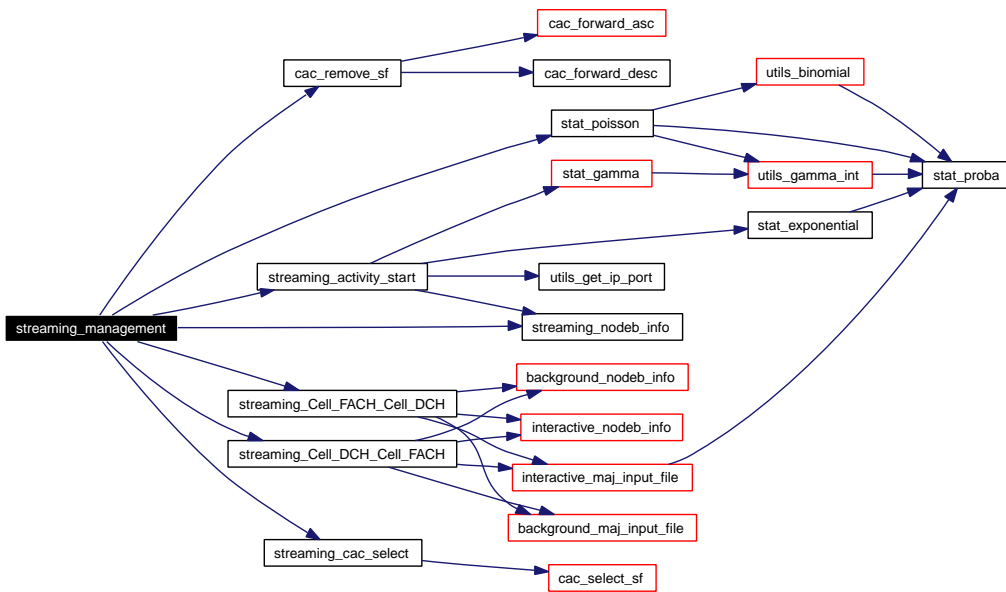
`ue_id` represents the identification number of the UE.

Definition at line 429 of file `streaming.c`.

References `actual_time`, `cac_remove_sf()`, `init_desynchro`, `ovsf_log_file`, `start_time`, `stat_poisson()`, `streaming_activity_start()`, `streaming_cac_select()`, `streaming_Cell_DCH_Cell_FACH()`, `streaming_Cell_FACH_Cell_DCH()`, `streaming_nodeb_info()`, and `ue_array`.

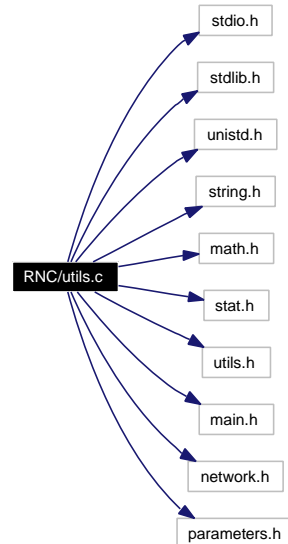
Referenced by `main()`.

Here is the call graph for this function:



## 4.27 RNC/utills.c File Reference

Include dependency graph for utills.c:



### Functions

- int [utils\\_even](#) (int a)
- double [utils\\_min](#) (double a, double b)
- int [utils\\_max](#) (int a, int b)
- int [utils\\_arrond](#) (double a)
- double [utils\\_abs](#) (double a)
- char \* [utils\\_choose\\_dst\\_ip](#) (int NodeB\_dst, char Sector\_dst)
- char \* [utils\\_create\\_ip](#) (int NodeB, int ue)
- int [utils\\_ip\\_present](#) (char \*ip)
- network\_info\_t [utils\\_get\\_ip\\_port](#) (int id)
- int [utils\\_get\\_ue\\_port](#) (char \*ip, int traffic\_class)
- int [utils\\_get\\_rnc\\_port](#) (char \*ip, int traffic\_class)
- double [utils\\_gamma\\_int](#) (int a)
- double [utils\\_gamma\\_large](#) (double a)
- double [utils\\_gamma\\_frac](#) (double a)
- double [utils\\_beta](#) (double a, double b)
- int [utils\\_binomial](#) (double p, int n)

### 4.27.1 Function Documentation

#### 4.27.1.1 int [utils\\_even](#) (int a)

Returns 1 if the argument is even and 0 if it is odd.

Definition at line 33 of file utills.c.

Referenced by [cac\\_ascendant\\_sum\(\)](#), and [cac\\_forward\\_asc\(\)](#).

**4.27.1.2 double utils\_min (double a, double b)**

Returns the minimum of the two double passed in parameters.

Definition at line 43 of file utils.c.

Referenced by mobility\_gain\_angle\_attenuation(), and wireless\_path\_loss\_and\_attenuation().

**4.27.1.3 int utils\_max (int a, int b)**

Returns the minimum of the two int passed in parameters.

Definition at line 54 of file utils.c.

Referenced by cac\_get\_DSCH\_sf().

**4.27.1.4 int utils\_arrond (double a)**

Returns the interger part of the double passed in parameter.

Definition at line 65 of file utils.c.

Referenced by mobility\_init().

**4.27.1.5 double utils\_abs (double a)**

Returns the absolute value of the double passed in parameter.

Definition at line 77 of file utils.c.

Referenced by mobility\_angle\_NodeB().

**4.27.1.6 char\* utils\_choose\_dst\_ip (int NodeB\_dst, char Sector\_dst)**

This procedure chooses a new IP address to allocate to an UE facing an handover.

**Parameters:**

*NodeB\_dst* represents the destination NodeB of the handover([1;4]),

*Sector\_dst* represents the destination Sector of the handover ([0;2]).

**Returns:**

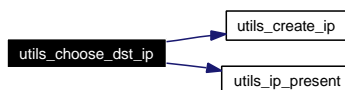
an IP address connected to the destination NodeB/Sector of the handover.

Definition at line 93 of file utils.c.

References utils\_create\_ip(), and utils\_ip\_present().

Referenced by handover\_management().

Here is the call graph for this function:



**4.27.1.7 char\* utils\_create\_ip (int NodeB, int ue)**

This procedure creates an IP address based on an UE and a NodeB number.

**Parameters:**

*ue* represents the identification of the UE ([0;62]),

*NodeB* represents the identification of the NodeB ([1;4]).

**Returns:**

the correct IP address.

Definition at line 125 of file utls.c.

Referenced by initialisation\_management(), and utils\_choose\_dst\_ip().

**4.27.1.8 int utils\_ip\_present (char \* ip)**

This procedure checks if the IP address is already used (present in the "dst\_ip\_array") or not.

**Parameters:**

*ip* represents the IP address to check.

**Returns:**

1 if the IP address is already used and 0 otherwise.

Definition at line 204 of file utls.c.

References ue\_array.

Referenced by utils\_choose\_dst\_ip().

**4.27.1.9 network\_info\_t utils\_get\_ip\_port (int id)**

This procedure fills a data structure with all the network information that might be deduced from a UE identification number.

**Parameters:**

*id* represents the identification of the UE ([0;62]).

**Returns:**

a data structure filled with the network information.

Definition at line 230 of file utls.c.

References ue\_array.

Referenced by background\_activity\_start(), conversational\_activity\_start(), interactive\_activity\_start(), and streaming\_activity\_start().

**4.27.1.10 int utils\_get\_ue\_port (char \* ip, int traffic\_class)**

This procedure returns the port number of one UE allocated to a traffic class.

**Parameters:**

*ip* represents the IP address of the UE ([0;62]),

*traffic\_class* represents the identification number of the traffic class (1:Conversational, 2:Interactive, 3:Streaming, 4:Background).

**Returns:**

the port number.

Definition at line 307 of file utils.c.

Referenced by `background_maj_input_file()`, `handover_maj_back_input_file()`, `handover_maj_conv_input_file()`, `handover_maj_inter_input_file()`, `handover_maj_stream_input_file()`, and `interactive_maj_input_file()`.

**4.27.1.11 int utils\_get\_rnc\_port (char \* ip, int traffic\_class)**

This procedure returns the port number of the RNC allocated to an UE and a traffic class.

**Parameters:**

*ip* represents the IP address of the UE ([0;62]),

*traffic\_class* represents the identification number of the traffic class (1:Conversational, 4:Background).

**Returns:**

the port number.

Definition at line 342 of file utils.c.

Referenced by `handover_maj_back_input_file()`, and `handover_maj_conv_input_file()`.

**4.27.1.12 double utils\_gamma\_int (int a)**

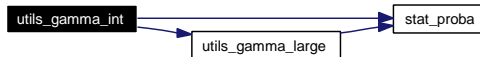
Contains a subgroup of a Gamma distribution computing.

Definition at line 396 of file utils.c.

References `stat_proba()`, and `utils_gamma_large()`.

Referenced by `stat_gamma()`, and `stat_poisson()`.

Here is the call graph for this function:

**4.27.1.13 double utils\_gamma\_large (double a)**

Contains a subgroup of a Gamma distribution computing.

Definition at line 417 of file utils.c.

References `stat_proba()`.

Referenced by `utils_gamma_int()`.

Here is the call graph for this function:



#### 4.27.1.14 double `utils_gamma_frac` (double *a*)

Contains a subgroup of a Gamma distribution computing.

Definition at line 439 of file `utils.c`.

References `stat_proba()`.

Referenced by `stat_gamma()`.

Here is the call graph for this function:



#### 4.27.1.15 double `utils_beta` (double *a*, double *b*)

Returns a randomly generated double by the Beta distribution with parameters *a* and *b*.

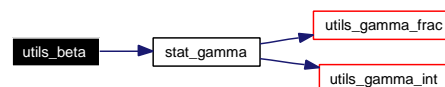
$$cdf : f(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1} (1-x)^{b-1}, 0 \leq x \leq 1$$

Definition at line 468 of file `utils.c`.

References `stat_gamma()`.

Referenced by `utils_binomial()`.

Here is the call graph for this function:



#### 4.27.1.16 int `utils_binomial` (double *p*, int *n*)

Returns a randomly generated integer by the Binomial distribution with parameters *p* and *n*.

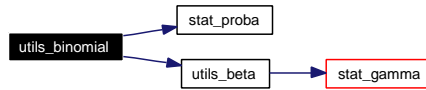
$$cdf : f(x) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x}, 0 \leq x \leq n$$

Definition at line 483 of file `utils.c`.

References `stat_proba()`, and `utils_beta()`.

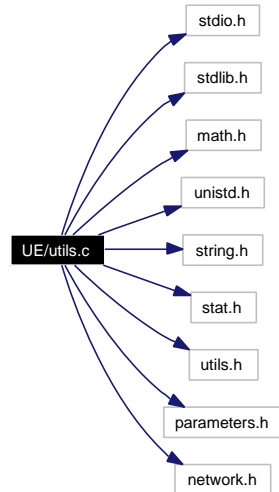
Referenced by `stat_poisson()`.

Here is the call graph for this function:



## 4.28 UE/utls.c File Reference

Include dependency graph for utls.c:



### Functions

- double [utls\\_gamma\\_int](#) (int *a*)
- double [utls\\_gamma\\_large](#) (double *a*)
- double [utls\\_gamma\\_frac](#) (double *a*)
- double [utls\\_beta](#) (double *a*, double *b*)
- int [utls\\_binomial](#) (double *p*, int *n*)

### 4.28.1 Function Documentation

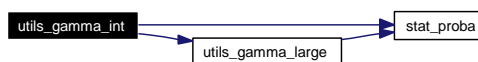
#### 4.28.1.1 double utls\_gamma\_int (int *a*)

Contains a subgroup of a Gamma distribution computing.

Definition at line 32 of file utls.c.

References [stat\\_proba\(\)](#), and [utls\\_gamma\\_large\(\)](#).

Here is the call graph for this function:



#### 4.28.1.2 double utls\_gamma\_large (double *a*)

Contains a subgroup of a Gamma distribution computing.

Definition at line 53 of file utls.c.

References stat\_proba().

Here is the call graph for this function:



#### 4.28.1.3 double utils\_gamma\_frac (double a)

Contains a subgroup of a Gamma distribution computing.

Definition at line 75 of file utils.c.

References stat\_proba().

Here is the call graph for this function:



#### 4.28.1.4 double utils\_beta (double a, double b)

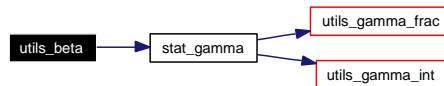
Returns a randomly generated double by the Beta distribution with parameters a and b.

$$cdf : f(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1} (1-x)^{b-1}, 0 \leq x \leq 1$$

Definition at line 104 of file utils.c.

References stat\_gamma().

Here is the call graph for this function:



#### 4.28.1.5 int utils\_binomial (double p, int n)

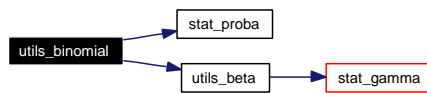
Returns a randomly generated integer by the Binomial distribution with parameters p and n.

$$cdf : f(x) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x}, 0 \leq x \leq n$$

Definition at line 119 of file utils.c.

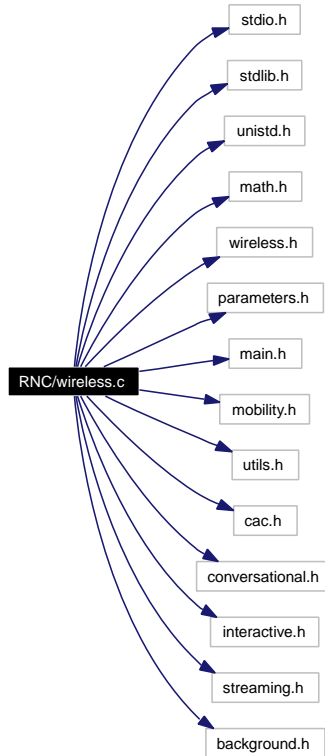
References stat\_proba(), and utils\_beta().

Here is the call graph for this function:



## 4.29 RNC/wireless.c File Reference

Include dependency graph for wireless.c:



### Functions

- void [wireless\\_init](#) ()
- long double [wireless\\_EbN0\\_to\\_ber](#) (double EbNO\_dB)
- double [wireless\\_path\\_loss\\_and\\_attenuation](#) (double x, double y, int NodeB, char Sector)
- double [wireless\\_DSCH\\_EbN0\\_computation](#) (int ue\_id, int tc)
- double [wireless\\_FACH\\_EbN0\\_computation](#) (int ue\_id)
- int [wireless\\_ber\\_to\\_bler](#) (long double ber)
- void [wireless\\_bler\\_computation](#) (int ue\_id, int tc)
- double [wireless\\_ber\\_to\\_EbNO](#) (long double ber)
- long double [wireless\\_bler\\_to\\_ber](#) (int bler)
- double [wireless\\_transmission\\_power\\_consumption\\_UE](#) (int ue\_id, int tc)
- double [wireless\\_transmission\\_power\\_consumption\\_Sector](#) (int NodeB, char Sector)
- void [wireless\\_decrease\\_transmission\\_power](#) (int NodeB, char Sector)
- void [wireless\\_transmission\\_power\\_consumption](#) ()

### Variables

- float [Shadow\\_Fading\\_Map](#) [560][485]

## 4.29.1 Function Documentation

### 4.29.1.1 void wireless\_init ()

This procedur initialise the shadow fading matrix.

Definition at line 43 of file wireless.c.

References Shadow\_Fading\_Map.

Referenced by initialisation\_management().

### 4.29.1.2 long double wireless\_EbN0\_to\_ber (double EbN0\_dB)

Based on a normalized energy per information bit a UE receives, this procedures returns the BER the communication undergo.

Definition at line 72 of file wireless.c.

Referenced by wireless\_bler\_computation().

### 4.29.1.3 double wireless\_path\_loss\_and\_attenuation (double x, double y, int NodeB, char Sector)

This procedure computes the path loss a UE have to mmanage from a certain NodeB and a certain Sector.

#### Parameters:

*x* represents the coordinate of a UE,

*y* represents the coordinate of a UE,

*NodeB* represents the NodeB from which we have to calculate the path loss, and

*Sector* represents the Sector of the NodeB from which we have to calculate the path loss.

#### Returns:

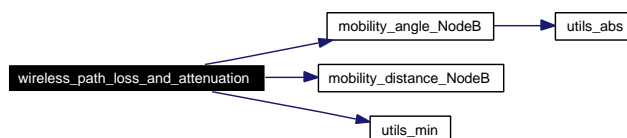
the path loss (linearly).

Definition at line 146 of file wireless.c.

References mobility\_angle\_NodeB(), mobility\_distance\_NodeB(), and utils\_min().

Referenced by wireless\_FACH\_EbN0\_computation(), and wireless\_transmission\_power\_consumption\_UE().

Here is the call graph for this function:



### 4.29.1.4 double wireless\_DSCH\_EbN0\_computation (int ue\_id, int tc)

This procedure computes the normalized energy per information bit a Ue gets in a DSCH.

Definition at line 201 of file wireless.c.

References `ue_array`.

Referenced by `wireless_bler_computation()`.

#### 4.29.1.5 `double wireless_FACH_EbN0_computation (int ue_id)`

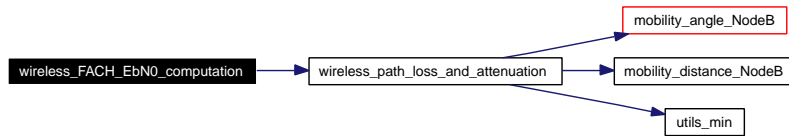
Based on the Ue position, this procedure computes the normalized energy per information bit this UE gets.

Definition at line 239 of file `wireless.c`.

References `Shadow_Fading_Map`, `ue_array`, and `wireless_path_loss_and_attenuation()`.

Referenced by `wireless_bler_computation()`.

Here is the call graph for this function:



#### 4.29.1.6 `int wireless_ber_to_bler (long double ber)`

Based on a BER, this procedure computes a BLER.

Definition at line 310 of file `wireless.c`.

Referenced by `wireless_bler_computation()`.

#### 4.29.1.7 `void wireless_bler_computation (int ue_id, int tc)`

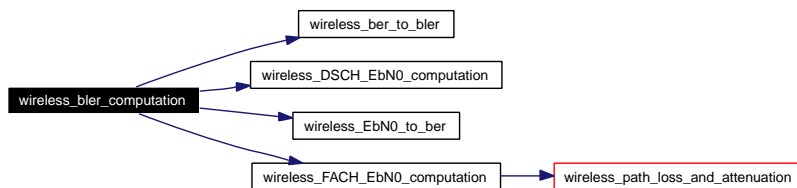
This procedure computes the BLER a communication undergoes based on a transport channel and the UE position.

Definition at line 324 of file `wireless.c`.

References `ue_array`, `wireless_ber_to_bler()`, `wireless_DSCH_EbN0_computation()`, `wireless_EbN0_to_ber()`, and `wireless_FACH_EbN0_computation()`.

Referenced by `background_nodeb_info()`, `conversational_nodeb_info()`, `handover_activity_shifting()`, `interactive_nodeb_info()`, `mobility_bler_evolution()`, and `streaming_nodeb_info()`.

Here is the call graph for this function:



#### 4.29.1.8 double wireless\_ber\_to\_EbNO (long double *ber*)

Based on a certain BER, this procedure computes the normalized energy per information bit a UE gets.

Definition at line 400 of file wireless.c.

Referenced by wireless\_transmission\_power\_consumption\_UE().

#### 4.29.1.9 long double wireless\_bler\_to\_ber (int *bler*)

Based on a certain BLER, this procedure computes the associated BER.

Definition at line 466 of file wireless.c.

Referenced by wireless\_transmission\_power\_consumption\_UE().

#### 4.29.1.10 double wireless\_transmission\_power\_consumption\_UE (int *ue\_id*, int *tc*)

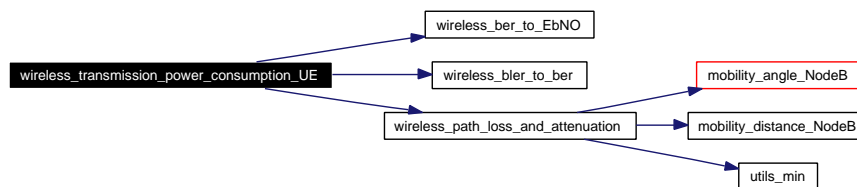
Linearly computes the DCH transmission power consumption of one UE.

Definition at line 480 of file wireless.c.

References Shadow\_Fading\_Map, ue\_array, wireless\_ber\_to\_EbNO(), wireless\_bler\_to\_ber(), and wireless\_path\_loss\_and\_attenuation().

Referenced by wireless\_decrease\_transmission\_power(), and wireless\_transmission\_power\_consumption\_Sector().

Here is the call graph for this function:



#### 4.29.1.11 double wireless\_transmission\_power\_consumption\_Sector (int *NodeB*, char *Sector*)

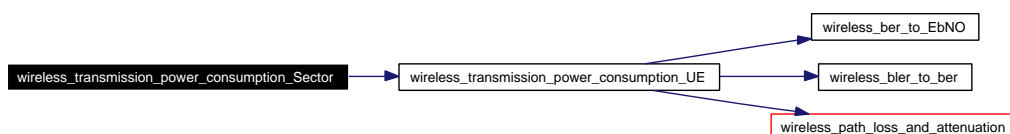
Linearly computes the transmission power a sector consumes.

Definition at line 727 of file wireless.c.

References ue\_array, and wireless\_transmission\_power\_consumption\_UE().

Referenced by wireless\_transmission\_power\_consumption().

Here is the call graph for this function:



#### 4.29.1.12 void wireless\_decrease\_transmission\_power (int NodeB, char Sector)

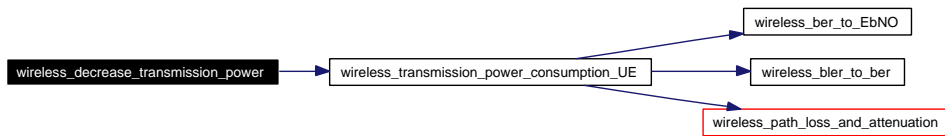
This procedure downswitch interactive and background sessions (from DCH to a common channel) of Bronze users.

Definition at line 760 of file wireless.c.

References ue\_array, and wireless\_transmission\_power\_consumption\_UE().

Referenced by wireless\_transmission\_power\_consumption().

Here is the call graph for this function:



#### 4.29.1.13 void wireless\_transmission\_power\_consumption ()

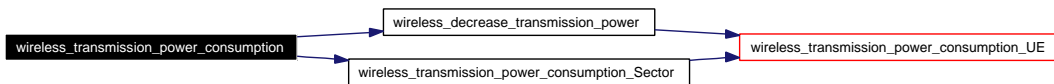
This procedure computes the transmission power a sector is using and decreases its transmission power if it is above 43dB.

Definition at line 834 of file wireless.c.

References wireless\_decrease\_transmission\_power(), and wireless\_transmission\_power\_consumption\_Sector().

Referenced by main().

Here is the call graph for this function:



## 4.29.2 Variable Documentation

### 4.29.2.1 float Shadow\_Fading\_Map[560][485]

This matrix represents the level of shadow fading (expressed in dB) in every 10 meters square of the emulated world.

Definition at line 36 of file wireless.c.

Referenced by wireless\_FACH\_EbN0\_computation(), wireless\_init(), and wireless\_transmission\_power\_consumption\_UE().