Rt-8900 Serial Control

0.3.0

Generated by Doxygen 1.8.11

Contents

Chapter 1

Main Page

#FT8900 Controller Provides serial control for the YAESU FT-8900R Transceiver.

##Usage

```
Usage: rt8900c [OPTION...] <serial port path>
Provides serial control for the YAESU FT-8900R Transceiver.

  -d, --dtr-on               Use the DTR pin of the serial connection as a
                             power button for the rig. (REQUIRES compatible
                             hardware)
      --hard-emulation       Exactly emulates the radio head instead of being
                             lazy_sending (worse performance, no observed
                             benefit, only useful for debugging)
  -v, --verbose[=LEVEL]       Produce verbose output add a number to select
                              level (1 = ERROR, 2= WARNING, 3=INFO, 4=ERROR,
                              5=DEBUG) output default is 'warning'.
  -?, --help                  Give this help list
      --usage                 Give a short usage message
  -V, --version               Print program version

Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.

Report bugs to <cormac.brady@hotmail.co.uk>.
```

Build and Install

For ubuntu but can be adapted for other distribution's and OS

```
sudo apt install cmake git build-essential
git clone <this repo url> rt8900c
cd rt8900c
cmake .
make
```

##Run tests

```
cmake .
make test

#for verbose output
./test/test_librt8900/test_librt8900
```

Credits

- CmakeLists build files taken from `this example`

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 button_transmit_value Struct Reference

Data Fields

- signed char **row**
- signed char **column**

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

- librt8900/control_packet.h

4.2 cmd Struct Reference

Data Fields

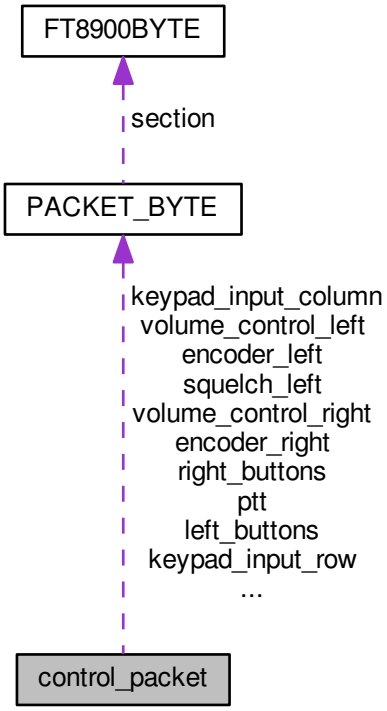
- int(∗ **cmd_pointer**)(char ∗∗args, SERIAL_CFG ∗config, struct control_packet ∗base_packet)
- char ∗ **keyword**
- char ∗ **discription**
- char ∗ **usage**
- int **num_args**

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

- main/main.c

4.3 control_packet Struct Reference

Collaboration diagram for control_packet:



Data Fields

- PACKET_BYTE **encoder_right**
- PACKET_BYTE **encoder_left**
- PACKET_BYTE **ptt**
- PACKET_BYTE **squelch_right**
- PACKET_BYTE **volume_control_right**
- PACKET_BYTE **keypad_input_row**
- PACKET_BYTE **volume_control_left**
- PACKET_BYTE **squelch_left**
- PACKET_BYTE **keypad_input_column**
- PACKET_BYTE **left_buttons**
- PACKET_BYTE **right_buttons**
- PACKET_BYTE **main_buttons**
- PACKET_BYTE **hyper_mem_buttons**

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

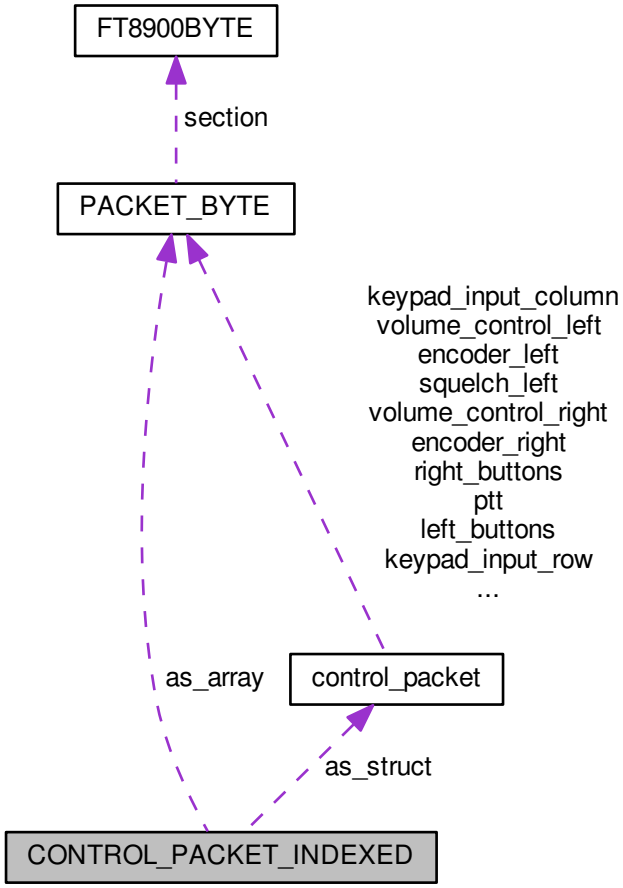
- librt8900/control_packet.h

4.4 CONTROL_PACKET_INDEXED Union Reference

used to get the packet as an array

`#include <control_packet.h>`

Collaboration diagram for CONTROL_PACKET_INDEXED:



Data Fields

- struct control_packet **as_struct**
- PACKET_BYTE **as_array** [13]

4.4.1 Detailed Description

used to get the packet as an array

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

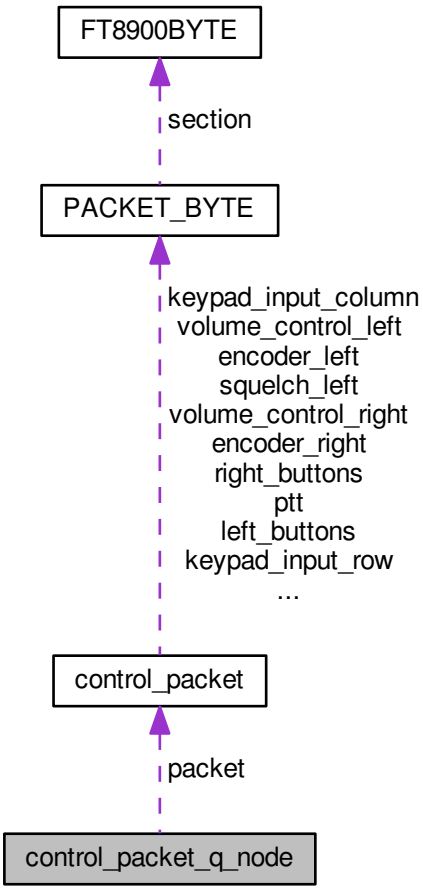
- librt8900/control_packet.h

4.5 control_packet_q_node Struct Reference

A internal, non intgrated queue struct for the packet queue.

`#include <control_packet.h>`

Collaboration diagram for control_packet_q_node:



Public Member Functions

- **TAILQ_ENTRY** (control_packet_q_node) nodes

Data Fields

- struct control_packet * **packet**
- enum pop_queue_behaviour **free_packet**

4.5.1 Detailed Description

A internal, non intgrated queue struct for the packet queue.

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

- librt8900/control_packet.h

4.6 control_packet_sender_config Struct Reference

Configuration for sending packets.

```
#include <librt8900.h>
```

Data Fields

- bool **lazy_sending**
- bool **dtr_pin_for_on**
- pthread_barrier_t * **initialised**
- struct CONTROL_PACKET_Q_HEAD * **queue**
- bool **keep_alive**

4.6.1 Detailed Description

Configuration for sending packets.

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

- librt8900/librt8900.h

4.7 display_packet_receiving_config Struct Reference

Configuration for receiving packets.

```
#include <librt8900.h>
```

Data Fields

- bool **keep_alive**
- bool **radio_seen**
- pthread_mutex_t **raw_packet_lock**
- unsigned char **latest_raw_packet** [DISPLAY_PACKET_SIZE]

4.7.1 Detailed Description

Configuration for receiving packets.

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

- librt8900/librt8900.h

4.8 FT8900BYTE Struct Reference

Data Fields

- unsigned int **data**: 7
- unsigned int **check_num**: 1

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

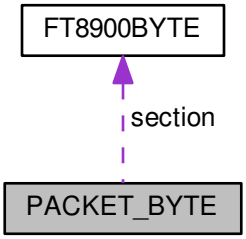
- librt8900/packet.h

4.9 PACKET_BYTE Union Reference

Used to store 1 byte of data from the packet.

`#include <packet.h>`

Collaboration diagram for PACKET_BYTE:



Data Fields

- FT8900BYTE **section**
- unsigned char **raw**

4.9.1 Detailed Description

Used to store 1 byte of data from the packet.

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

- librt8900/packet.h

4.10 radio_side Struct Reference

Data Fields

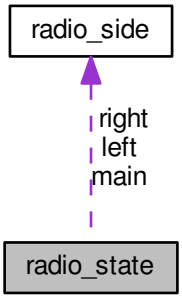
- int **busy**
- int **frequency**
- enum rt8900_power_level **power_level**

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

- librt8900/display_packet.h

4.11 radio_state Struct Reference

Collaboration diagram for radio_state:



Data Fields

- struct radio_side * **main**
- struct radio_side **left**
- struct radio_side **right**

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

- librt8900/display_packet.h

4.12 range_KHz Struct Reference

Represents one of the capable ranges of the radio.

`#include <librt8900.h>`

Data Fields

- const bool **tx_allowed**
- const char * **name**
- const int **low**
- const int **high**

4.12.1 Detailed Description

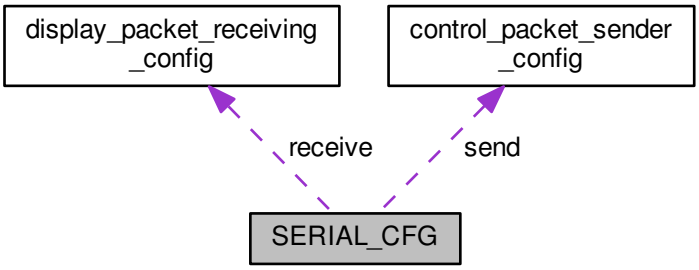
Represents one of the capable ranges of the radio.

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

- librt8900/librt8900.h

4.13 SERIAL_CFG Struct Reference

Collaboration diagram for SERIAL_CFG:



Data Fields

- char * **serial_path**
- int **serial_fd**
- bool **shutdown_on_timeout**
- struct control_packet_sender_config **send**
- struct display_packet_receiving_config **receive**

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

- librt8900/librt8900.h

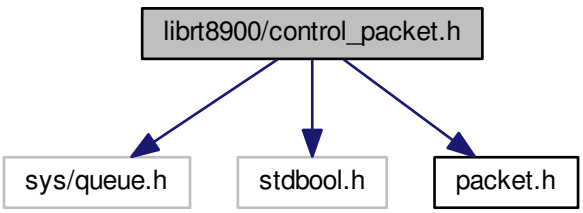
Chapter 5

File Documentation

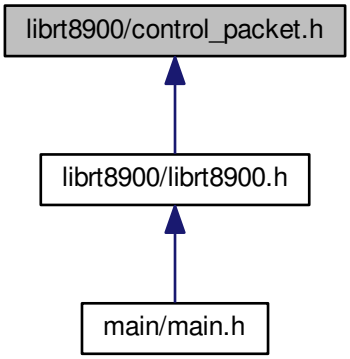
5.1 librt8900/control_packet.h File Reference

```
#include <sys/queue.h>
#include <stdbool.h>
#include "packet.h"
```

Include dependency graph for control_packet.h:



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



Data Structures

- struct control_packet
- struct button_transmit_value
- struct control_packet_q_node

  A internal, non intgrated queue struct for the packet queue.
- union CONTROL_PACKET_INDEXED

  used to get the packet as an array

Macros

- #define **MILLISECONDS_BETWEEN_PACKETS_STANDARD** 3
- #define **MILLISECONDS_DEBOUNCE_WAIT** 51
- #define **DEFAULT_VOLUME** 0x1f
- #define **maloc_control_packet**(pointer_name) struct control_packet ∗(pointer_name) = (struct control_packet∗) malloc(sizeof(∗(pointer_name)))
- #define **VD_INDEX_0** 0X00
- #define **VD_INDEX_1** 0X1A
- #define **VD_INDEX_2** 0X32
- #define **VD_INDEX_3** 0X4C
- #define **VD_INDEX_4** 0X64
- #define **VD_NONE** 0X7F
- #define **BUTTON_NONE_VALUE** {VD_NONE, VD_NONE}
- #define **BUTTON_1_VALUE** {VD_INDEX_0, VD_INDEX_1}
- #define **BUTTON_2_VALUE** {VD_INDEX_0, VD_INDEX_2}
- #define **BUTTON_3_VALUE** {VD_INDEX_0, VD_INDEX_3}
- #define **BUTTON_A_VALUE** {VD_INDEX_0, VD_INDEX_4}
- #define **BUTTON_UP_VALUE** {VD_INDEX_1, VD_INDEX_0}
- #define **BUTTON_4_VALUE** {VD_INDEX_1, VD_INDEX_1}
- #define **BUTTON_5_VALUE** {VD_INDEX_1, VD_INDEX_2}
- #define **BUTTON_6_VALUE** {VD_INDEX_1, VD_INDEX_3}
- #define **BUTTON_B_VALUE** {VD_INDEX_1, VD_INDEX_4}

- #define **BUTTON_DOWN_VALUE** {VD_INDEX_2, VD_INDEX_0}
- #define **BUTTON_7_VALUE** {VD_INDEX_2, VD_INDEX_1}
- #define **BUTTON_8_VALUE** {VD_INDEX_2, VD_INDEX_2}
- #define **BUTTON_9_VALUE** {VD_INDEX_2, VD_INDEX_3}
- #define **BUTTON_C_VALUE** {VD_INDEX_2, VD_INDEX_4}
- #define **BUTTON_X_VALUE** {VD_INDEX_3, VD_INDEX_1}
- #define **BUTTON_0_VALUE** {VD_INDEX_3, VD_INDEX_2}
- #define **BUTTON_HASH_VALUE** {VD_INDEX_3, VD_INDEX_3}
- #define **BUTTON_D_VALUE** {VD_INDEX_3, VD_INDEX_4}
- #define **BUTTON_P1_VALUE** {VD_INDEX_4, VD_INDEX_1}
- #define **BUTTON_P2_VALUE** {VD_INDEX_4, VD_INDEX_2}
- #define **BUTTON_P3_VALUE** {VD_INDEX_4, VD_INDEX_3}
- #define **BUTTON_P4_VALUE** {VD_INDEX_4, VD_INDEX_4}

Enumerations

- enum **main_menu_buttons** {
  NOT_PRESSED = 0, **R_ENCODER_BUTTON** = 1, **L_ENCODER_BUTTON** = (1 << 1), **SET_BUTTON** = (1 << 2),
  WIRES_BUTTON = (1 << 3) }
- enum **radios** { **RADIO_LEFT**, **RADIO_RIGHT** }
- enum **right_menu_buttons** {
  RIGHT_NONE = DATA_MAX_NUM, **RIGHT_LOW** = 0x00, **RIGHT_VM** = 0x20, **RIGHT_HM** = 0x40,
  RIGHT_SCN = 0x60 }
- enum **left_menu_buttons** {
  LEFT_NONE = DATA_MAX_NUM, **LEFT_LOW** = RIGHT_SCN, **LEFT_VM** = RIGHT_HM, **LEFT_HM** = RIGHT_VM,
  LEFT_SCN = RIGHT_LOW }
- enum pop_queue_behaviour { **PACKET_FREE_AFTER_SEND** = 0, **PACKET_ONLY_SEND** = 1 }

  Options on if the packet added to the queue will be freed once sent.

Functions

- typedef TAILQ_HEAD (CONTROL_PACKET_Q_HEAD, control_packet_q_node) CONTROL_PACKET_Q_HEAD

  Create our packet queue struct.
- void **set_keypad_button** (struct control_packet *packet, const struct button_transmit_value *button)
- void **set_main_button** (struct control_packet *packet, const enum main_menu_buttons button)
- void **set_left_button** (struct control_packet *packet, const enum left_menu_buttons button)
- void **set_right_button** (struct control_packet *packet, const enum right_menu_buttons button)
- const struct button_transmit_value * **button_from_int** (int i)
- signed char safe_int_char (int number)

  Takes and returns a int if it can fit into a PACKET_BYTE.
- int set_volume_left (struct control_packet *packet, int number)

  set the volume between 0-127. 0 is mute.
- int set_volume_right (struct control_packet *packet, int number)

  set the volume between 0-127. 0 is mute.
- int set_volume (struct control_packet *packet, int left, int right)

  Set the left and right volume. between 0-127. 0 is mute.
- int set_squelch_left (struct control_packet *packet, int number)
- int set_squelch_right (struct control_packet *packet, int number)
- int set_squelch (struct control_packet *packet, int left, int right)
- void ptt (struct control_packet *base_packet, int ptt)

  toggle transmission 2 to start 1 to stop
- void * send_control_packets (void *c)
- void **packet_debug** (const struct control_packet *packet, CONTROL_PACKET_INDEXED *input_packet_arr)

Variables

- const struct button_transmit_value **KEYPAD_BUTTON_NONE**
- const struct button_transmit_value **KEYPAD_NUMBER_BUTTONS** [10]
- const struct button_transmit_value **KEYPAD_BUTTON_A**
- const struct button_transmit_value **KEYPAD_BUTTON_B**
- const struct button_transmit_value **KEYPAD_BUTTON_C**
- const struct button_transmit_value **KEYPAD_BUTTON_D**
- const struct button_transmit_value **KEYPAD_BUTTON_HASH**
- const struct button_transmit_value **KEYPAD_BUTTON_X**
- const struct button_transmit_value **KEYPAD_BUTTON_P1**
- const struct button_transmit_value **KEYPAD_BUTTON_P2**
- const struct button_transmit_value **KEYPAD_BUTTON_P3**
- const struct button_transmit_value **KEYPAD_BUTTON_P4**
- const struct control_packet control_packet_defaults

  The recommended defaults for the control packet.

5.1.1 Detailed Description

Functions to manipulate control_packet structs

5.1.2 Function Documentation

5.1.2.1 signed char safe_int_char (int *number*)

Takes and returns a int if it can fit into a PACKET_BYTE.

Returns

NULL if the number will not fit into the packet (7 bits) else will return the provided int

5.1.2.2 void* send_control_packets (void * *c*)

Starts sending control packets as defined by SERIAL_CFG

This function is designed to be started as a thread

5.1.2.3 int set_squelch (struct **control_packet** * *packet,* int *left,* int *right*)

Set the left and right squelch. between 0-127. 127 filters no noise.

5.1.2.4 int set_squelch_left (struct **control_packet** * *packet,* int *number*)

Set the left squelch between 0-127. 127 filters no noise.e.

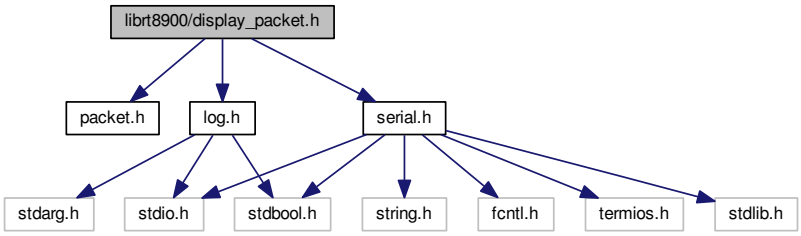
5.1.2.5 int set_squelch_right (struct control_packet * *packet*, int *number*)

Set the right squelch between 0-127. 127 filters no noise.

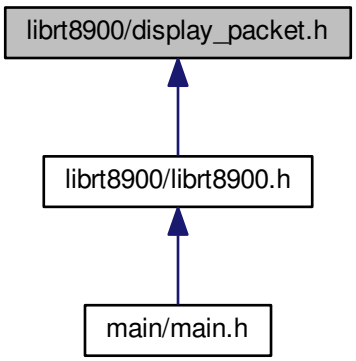
5.2 librt8900/display_packet.h File Reference

```
#include "packet.h"
#include "serial.h"
#include "log.h"
```

Include dependency graph for display_packet.h:



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



Data Structures

- struct radio_side
- struct radio_state

Macros

- #define **DISPLAY_PACKET_SIZE** 42
- #define **MS_PACKET_WAIT_TIME** 25
- #define **BIT_LOCATED_AT**(byte, bit) (((byte) ∗ 8) + (bit))

Typedefs

- typedef PACKET_BYTE **DISPLAY_PACKET**[DISPLAY_PACKET_SIZE]

Enumerations

- enum **display_packet_bitmasks** {
  AUTO_POWER_OFF = BIT_LOCATED_AT(0, 1), **KEYPAD_LOCK** = BIT_LOCATED_AT(30, 4), **SET** = B↩
  IT_LOCATED_AT(4, 0), **BD8600** = BIT_LOCATED_AT(4, 0),
  LEFT_BUSY = BIT_LOCATED_AT(12,2), **LEFT_MAIN** = BIT_LOCATED_AT(36, 2), **LEFT_TX** = BIT_LO↩
  CATED_AT(36, 3), **LEFT_MINUS** = BIT_LOCATED_AT(36, 5),
  LEFT_PLUS = BIT_LOCATED_AT(36, 4), **LEFT_DCS** = BIT_LOCATED_AT(9, 3), **LEFT_DEC** = BIT_LO↩
  CATED_AT(37, 0), **LEFT_ENC** = BIT_LOCATED_AT(36, 6),
  LEFT_SKIP = BIT_LOCATED_AT(37, 2), **LEFT_PMS** = BIT_LOCATED_AT(37, 1), **LEFT_POWER_LOW** =
  BIT_LOCATED_AT(1, 5), **LEFT_POWER_MEDIUM** = BIT_LOCATED_AT(0, 2),
  LEFT_FREQ_1_A = BIT_LOCATED_AT(13, 0), **LEFT_FREQ_1_B** = BIT_LOCATED_AT(13, 1), **LEFT_F**↩
  REQ_1_C = BIT_LOCATED_AT(13, 2), **LEFT_FREQ_1_D** = BIT_LOCATED_AT(13, 3),
  LEFT_FREQ_1_E = BIT_LOCATED_AT(13, 4), **LEFT_FREQ_1_F** = BIT_LOCATED_AT(13, 5), **LEFT_FR**↩
  EQ_1_G = BIT_LOCATED_AT(13, 6), **LEFT_FREQ_1_H** = BIT_LOCATED_AT(14, 1),
  LEFT_FREQ_1_I = BIT_LOCATED_AT(14, 2), **LEFT_FREQ_1_J** = BIT_LOCATED_AT(14, 3), **LEFT_FR**↩
  EQ_1_K = BIT_LOCATED_AT(14, 5), **LEFT_FREQ_1_L** = BIT_LOCATED_AT(14, 6),
  LEFT_FREQ_1_M = BIT_LOCATED_AT(15, 0), **LEFT_FREQ_2_A** = BIT_LOCATED_AT(10, 5), **LEFT_F**↩
  REQ_2_B = BIT_LOCATED_AT(10, 6), **LEFT_FREQ_2_C** = BIT_LOCATED_AT(11, 0),
  LEFT_FREQ_2_D = BIT_LOCATED_AT(11, 1), **LEFT_FREQ_2_E** = BIT_LOCATED_AT(11, 2), **LEFT_F**↩
  REQ_2_F = BIT_LOCATED_AT(11, 3), **LEFT_FREQ_2_G** = BIT_LOCATED_AT(11, 4),
  LEFT_FREQ_2_H = BIT_LOCATED_AT(11, 6), **LEFT_FREQ_2_I** = BIT_LOCATED_AT(12, 0), **LEFT_FR**↩
  EQ_2_J = BIT_LOCATED_AT(12, 1), **LEFT_FREQ_2_K** = BIT_LOCATED_AT(12, 3),
  LEFT_FREQ_2_L = BIT_LOCATED_AT(12, 4), **LEFT_FREQ_2_M** = BIT_LOCATED_AT(12, 5), **LEFT_F**↩
  REQ_3_A = BIT_LOCATED_AT(8, 3), **LEFT_FREQ_3_B** = BIT_LOCATED_AT(8, 4),
  LEFT_FREQ_3_C = BIT_LOCATED_AT(8, 5), **LEFT_FREQ_3_D** = BIT_LOCATED_AT(8, 6), **LEFT_FRE**↩
  Q_3_E = BIT_LOCATED_AT(9, 0), **LEFT_FREQ_3_F** = BIT_LOCATED_AT(9, 1),
  LEFT_FREQ_3_G = BIT_LOCATED_AT(9, 2), **LEFT_FREQ_3_H** = BIT_LOCATED_AT(9, 4), **LEFT_FRE**↩
  Q_3_I = BIT_LOCATED_AT(9, 5), **LEFT_FREQ_3_J** = BIT_LOCATED_AT(9, 6),
  LEFT_FREQ_3_K = BIT_LOCATED_AT(10, 1), **LEFT_FREQ_3_L** = BIT_LOCATED_AT(10, 2), **LEFT_F**↩
  REQ_3_M = BIT_LOCATED_AT(10, 3), **LEFT_FREQ_4_A** = BIT_LOCATED_AT(5, 2),
  LEFT_FREQ_4_B = BIT_LOCATED_AT(5, 3), **LEFT_FREQ_4_C** = BIT_LOCATED_AT(5, 4), **LEFT_FRE**↩
  Q_4_D = BIT_LOCATED_AT(5, 5), **LEFT_FREQ_4_E** = BIT_LOCATED_AT(5, 6),
  LEFT_FREQ_4_F = BIT_LOCATED_AT(6, 0), **LEFT_FREQ_4_G** = BIT_LOCATED_AT(6, 1), **LEFT_FRE**↩
  Q_4_H = BIT_LOCATED_AT(6, 3), **LEFT_FREQ_4_I** = BIT_LOCATED_AT(6, 4),
  LEFT_FREQ_4_J = BIT_LOCATED_AT(6, 5), **LEFT_FREQ_4_K** = BIT_LOCATED_AT(7, 0), **LEFT_FRE**↩
  Q_4_L = BIT_LOCATED_AT(7, 1), **LEFT_FREQ_4_M** = BIT_LOCATED_AT(7, 2),
  LEFT_FREQ_5_A = BIT_LOCATED_AT(3, 0), **LEFT_FREQ_5_B** = BIT_LOCATED_AT(3, 1), **LEFT_FRE**↩
  Q_5_C = BIT_LOCATED_AT(3, 2), **LEFT_FREQ_5_D** = BIT_LOCATED_AT(3, 3),
  LEFT_FREQ_5_E = BIT_LOCATED_AT(3, 4), **LEFT_FREQ_5_F** = BIT_LOCATED_AT(3, 5), **LEFT_FRE**↩
  Q_5_G = BIT_LOCATED_AT(3, 6), **LEFT_FREQ_5_H** = BIT_LOCATED_AT(4, 1),
  LEFT_FREQ_5_I = BIT_LOCATED_AT(4, 2), **LEFT_FREQ_5_J** = BIT_LOCATED_AT(4, 3), **LEFT_FREQ**↩
  _5_K = BIT_LOCATED_AT(4, 5), **LEFT_FREQ_5_L** = BIT_LOCATED_AT(4, 6),
  LEFT_FREQ_5_M = BIT_LOCATED_AT(5, 0), **LEFT_FREQ_6_A** = BIT_LOCATED_AT(0, 5), **LEFT_FRE**↩
  Q_6_B = BIT_LOCATED_AT(0, 6), **LEFT_FREQ_6_C** = BIT_LOCATED_AT(1, 0),
  LEFT_FREQ_6_D = BIT_LOCATED_AT(1, 1), **LEFT_FREQ_6_E** = BIT_LOCATED_AT(1, 2), **LEFT_FRE**↩

Q_6_F = BIT_LOCATED_AT(1, 3), **LEFT_FREQ_6_G** = BIT_LOCATED_AT(1, 4),
LEFT_FREQ_6_H = BIT_LOCATED_AT(1, 6), **LEFT_FREQ_6_I** = BIT_LOCATED_AT(2, 0), **LEFT_FREQ↩
_6_J** = BIT_LOCATED_AT(2, 1), **LEFT_FREQ_6_K** = BIT_LOCATED_AT(2, 3),
LEFT_FREQ_6_L = BIT_LOCATED_AT(2, 4), **LEFT_FREQ_6_M** = BIT_LOCATED_AT(2, 5), **LEFT_FRE↩
Q_7** = 0, **LEFT_FREQ_PERIOD** = BIT_LOCATED_AT(0, 3),
RIGHT_BUSY = BIT_LOCATED_AT(28,2), **RIGHT_MAIN** = BIT_LOCATED_AT(32, 0), **RIGHT_TX** = BIT_↩
LOCATED_AT(32, 1), **RIGHT_MINUS** = BIT_LOCATED_AT(32, 3),
RIGHT_PLUS = BIT_LOCATED_AT(32, 2), **RIGHT_DCS** = BIT_LOCATED_AT(22, 2), **RIGHT_DEC** = BIT↩
_LOCATED_AT(32, 5), **RIGHT_ENC** = BIT_LOCATED_AT(32, 4),
RIGHT_SKIP = BIT_LOCATED_AT(33, 0), **RIGHT_PMS** = BIT_LOCATED_AT(32, 6), **RIGHT_POWER_L↩
OW** = BIT_LOCATED_AT(17, 5), **RIGHT_POWER_MEDIUM** = BIT_LOCATED_AT(16, 2),
RIGHT_FREQ_1_A = BIT_LOCATED_AT(29, 0), **RIGHT_FREQ_1_B** = BIT_LOCATED_AT(29, 1), **RIGH↩
T_FREQ_1_C** = BIT_LOCATED_AT(29, 2), **RIGHT_FREQ_1_D** = BIT_LOCATED_AT(29, 3),
RIGHT_FREQ_1_E = BIT_LOCATED_AT(29, 4), **RIGHT_FREQ_1_F** = BIT_LOCATED_AT(29, 5), **RIGHT↩
_FREQ_1_G** = BIT_LOCATED_AT(29, 6), **RIGHT_FREQ_1_H** = BIT_LOCATED_AT(30, 1),
RIGHT_FREQ_1_I = BIT_LOCATED_AT(30, 1), **RIGHT_FREQ_1_J** = BIT_LOCATED_AT(30, 3), **RIGHT↩
_FREQ_1_K** = BIT_LOCATED_AT(30, 5), **RIGHT_FREQ_1_L** = BIT_LOCATED_AT(30, 6),
RIGHT_FREQ_1_M = BIT_LOCATED_AT(31, 0), **RIGHT_FREQ_2_A** = BIT_LOCATED_AT(26, 5), **RIGH↩
T_FREQ_2_B** = BIT_LOCATED_AT(26, 6), **RIGHT_FREQ_2_C** = BIT_LOCATED_AT(27, 0),
RIGHT_FREQ_2_D = BIT_LOCATED_AT(27, 1), **RIGHT_FREQ_2_E** = BIT_LOCATED_AT(27, 2), **RIGH↩
T_FREQ_2_F** = BIT_LOCATED_AT(27, 3), **RIGHT_FREQ_2_G** = BIT_LOCATED_AT(27, 4),
RIGHT_FREQ_2_H = BIT_LOCATED_AT(27, 6), **RIGHT_FREQ_2_I** = BIT_LOCATED_AT(28, 0), **RIGHT↩
_FREQ_2_J** = BIT_LOCATED_AT(28, 1), **RIGHT_FREQ_2_K** = BIT_LOCATED_AT(28, 3),
RIGHT_FREQ_2_L = BIT_LOCATED_AT(28, 4), **RIGHT_FREQ_2_M** = BIT_LOCATED_AT(28, 5), **RIGH↩
T_FREQ_3_A** = BIT_LOCATED_AT(24, 3), **RIGHT_FREQ_3_B** = BIT_LOCATED_AT(24, 4),
RIGHT_FREQ_3_C = BIT_LOCATED_AT(24, 5), **RIGHT_FREQ_3_D** = BIT_LOCATED_AT(24, 6), **RIGH↩
T_FREQ_3_E** = BIT_LOCATED_AT(25, 0), **RIGHT_FREQ_3_F** = BIT_LOCATED_AT(25, 1),
RIGHT_FREQ_3_G = BIT_LOCATED_AT(25, 2), **RIGHT_FREQ_3_H** = BIT_LOCATED_AT(25, 4), **RIGH↩
T_FREQ_3_I** = BIT_LOCATED_AT(25, 5), **RIGHT_FREQ_3_J** = BIT_LOCATED_AT(25, 6),
RIGHT_FREQ_3_K = BIT_LOCATED_AT(26, 1), **RIGHT_FREQ_3_L** = BIT_LOCATED_AT(26, 2), **RIGHT↩
_FREQ_3_M** = BIT_LOCATED_AT(26, 3), **RIGHT_FREQ_4_A** = BIT_LOCATED_AT(21, 2),
RIGHT_FREQ_4_B = BIT_LOCATED_AT(21, 3), **RIGHT_FREQ_4_C** = BIT_LOCATED_AT(21, 4), **RIGH↩
T_FREQ_4_D** = BIT_LOCATED_AT(21, 5), **RIGHT_FREQ_4_E** = BIT_LOCATED_AT(21, 6),
RIGHT_FREQ_4_F = BIT_LOCATED_AT(22, 0), **RIGHT_FREQ_4_G** = BIT_LOCATED_AT(22, 1), **RIGH↩
T_FREQ_4_H** = BIT_LOCATED_AT(22, 3), **RIGHT_FREQ_4_I** = BIT_LOCATED_AT(22, 4),
RIGHT_FREQ_4_J = BIT_LOCATED_AT(22, 5), **RIGHT_FREQ_4_K** = BIT_LOCATED_AT(23, 0), **RIGHT↩
_FREQ_4_L** = BIT_LOCATED_AT(23, 1), **RIGHT_FREQ_4_M** = BIT_LOCATED_AT(23, 2),
RIGHT_FREQ_5_A = BIT_LOCATED_AT(19, 0), **RIGHT_FREQ_5_B** = BIT_LOCATED_AT(19, 1), **RIGH↩
T_FREQ_5_C** = BIT_LOCATED_AT(19, 2), **RIGHT_FREQ_5_D** = BIT_LOCATED_AT(19, 3),
RIGHT_FREQ_5_E = BIT_LOCATED_AT(19, 4), **RIGHT_FREQ_5_F** = BIT_LOCATED_AT(19, 5), **RIGHT↩
_FREQ_5_G** = BIT_LOCATED_AT(19, 6), **RIGHT_FREQ_5_H** = BIT_LOCATED_AT(20, 1),
RIGHT_FREQ_5_I = BIT_LOCATED_AT(20, 2), **RIGHT_FREQ_5_J** = BIT_LOCATED_AT(20, 3), **RIGHT↩
_FREQ_5_K** = BIT_LOCATED_AT(20, 5), **RIGHT_FREQ_5_L** = BIT_LOCATED_AT(20, 6),
RIGHT_FREQ_5_M = BIT_LOCATED_AT(21, 0), **RIGHT_FREQ_6_A** = BIT_LOCATED_AT(16, 5), **RIGH↩
T_FREQ_6_B** = BIT_LOCATED_AT(16, 6), **RIGHT_FREQ_6_C** = BIT_LOCATED_AT(17, 0),
RIGHT_FREQ_6_D = BIT_LOCATED_AT(17, 1), **RIGHT_FREQ_6_E** = BIT_LOCATED_AT(17, 2), **RIGH↩
T_FREQ_6_F** = BIT_LOCATED_AT(17, 3), **RIGHT_FREQ_6_G** = BIT_LOCATED_AT(17, 4),
RIGHT_FREQ_6_H = BIT_LOCATED_AT(17, 6), **RIGHT_FREQ_6_I** = BIT_LOCATED_AT(18, 0), **RIGHT↩
_FREQ_6_J** = BIT_LOCATED_AT(18, 1), **RIGHT_FREQ_6_K** = BIT_LOCATED_AT(18, 3),
RIGHT_FREQ_6_L = BIT_LOCATED_AT(18, 4), **RIGHT_FREQ_6_M** = BIT_LOCATED_AT(18, 5), **RIGH↩
T_FREQ_7** = BIT_LOCATED_AT(16, 0), **RIGHT_FREQ_PERIOD** = BIT_LOCATED_AT(16, 3) }

- enum **rt8900_power_level** {
  POWER_UNKNOWEN = 0, **POWER_LOW**, **POWER_MEDIUM_FUZZY**, **POWER_MEDIUM_1**,
  POWER_MEDIUM_2, **POWER_HIGH** }

Functions

- int **display_packet_read** (DISPLAY_PACKET packet, const enum display_packet_bitmasks bit_number)

- void insert_shifted_packet (DISPLAY_PACKET packet, unsigned char buffer[], size_t buffer_length, int start_of_packet_index)
- int segment_to_int (int segment_bitmask)
- void read_busy (DISPLAY_PACKET packet, struct radio_state *state)

  Gets busy state from display_packet.
- void read_main (DISPLAY_PACKET packet, struct radio_state *state)
- void read_power_fuzzy (DISPLAY_PACKET packet, struct radio_state *state)

  Gets the power levels of the radios only using reads.
- int read_frequency (DISPLAY_PACKET packet, struct radio_state *state)

  Writes the frequency to the state packet.
- int is_main (struct radio_state *radio, struct radio_side *side)

  Check if the input radio is the main.
- void read_packet_state (DISPLAY_PACKET packet, struct radio_state *state)

  runs all packet read functions on the given packet

5.2.1 Detailed Description

Functions to get and sed data from the display_packet

5.2.2 Function Documentation

5.2.2.1 void insert_shifted_packet (DISPLAY_PACKET *packet*, unsigned char *buffer[]*, size_t *buffer_length*, int *start_of_packet_index*)

Write to the packet in the correct order.

Warning

This assumes buffer array length is DISPLAY_PACKET_SIZE (42)

5.2.2.2 int is_main (struct radio_state * *radio*, struct radio_side * *side*)

Check if the input radio is the main.

Returns

0 if radio is not currently main

5.2.2.3 int read_frequency (DISPLAY_PACKET *packet*, struct radio_state * *state*)

Writes the frequency to the state packet.

Returns

0 on success and 1 on error.

5.2.2.4 void read_main (DISPLAY_PACKET *packet*, struct radio_state * *state*)

Sets the main correct pointer to the correct radio,

Returns

NULL if nether selected

5.2.2.5 int segment_to_int (int *segment_bitfield*)

Takes a bitfield and matches to known numbers an char of bits (ordered as described above)

Returns

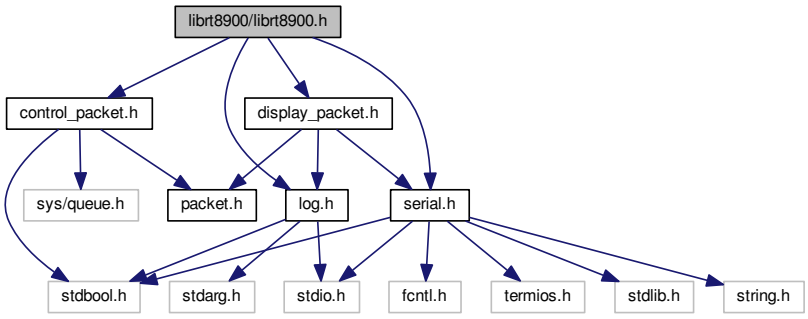
0-9 and -1 on error

5.3 librt8900/librt8900.h File Reference

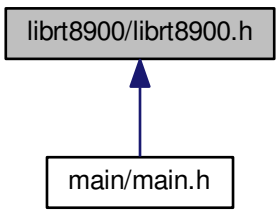
The main headder file for the librt8900 libary. Contains functions that use both the contol_packet and DISPLAY_↩ PACKET.

```
#include "log.h"
#include "serial.h"
#include "control_packet.h"
#include "display_packet.h"
```

Include dependency graph for librt8900.h:



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



Data Structures

- struct range_KHz

  Represents one of the capable ranges of the radio.
- struct control_packet_sender_config

  Configuration for sending packets.
- struct display_packet_receiving_config

  Configuration for receiving packets.
- struct SERIAL_CFG

Macros

- #define **VALID_POWER_LEVEL**(num) ((num) == POWER_LOW || (num) == POWER_MEDIUM_FUZZY || (num) == POWER_HIGH)

Enumerations

- enum frequency_permission { **INVALID_FREQUENCY** = 0, **VALID_FREQUENCY_RX_ONLY** = 1, **VALID_FREQUENCY** = 2 }

  Represents the abilities of the radio for a particular frequency.

Functions

- void * send_control_packets (void *c)
- void * receive_display_packets (void *c)
- const struct range_KHz * get_range (int frequency_khz)
- int in_freq_range (int frequency_khz)
- int get_frequency (struct radio_side *radio)

  Gets the current frequency of the radio.
- bool current_freq_valid (struct radio_side *radio)
- void send_new_packet (SERIAL_CFG *config, struct control_packet *new_packet, enum pop_queue_behaviour free_choice)
- bool check_radio_rx (SERIAL_CFG *config)

  Check that we are received from the radio at lest once.
- void wait_to_send (const SERIAL_CFG *cfg)

  Blocks until there are no new packets to send.
- void shutdown_threads (SERIAL_CFG *cfg)

  gracefully stops send_control_packets() and receive_display_packets()
- int set_frequency (SERIAL_CFG *cfg, struct control_packet *base_packet, int number)

  Adds the required packets to dial a number.
- int set_main_radio (SERIAL_CFG *cfg, struct control_packet *base_packet, enum radios side)

  Switches context to the desired radio.
- int set_left_power_level (SERIAL_CFG *cfg, struct control_packet *base_packet, enum rt8900_power_level power_level)

  sets left power level on radio
- int set_right_power_level (SERIAL_CFG *cfg, struct control_packet *base_packet, enum rt8900_power_level power_level)

  sets right power level on radio
- int set_power_button (SERIAL_CFG *cfg)

  Experimental! Sets the dtr pin low for one second to trigger radio on.
- int get_display_packet (SERIAL_CFG *config, DISPLAY_PACKET packet)

  writes the most recent packet to

5.3.1 Detailed Description

The main headder file for the librt8900 libary. Contains functions that use both the contol_packet and DISPLAY_↩ PACKET.

5.3.2 Function Documentation

5.3.2.1 bool check_radio_rx (SERIAL_CFG * *config*)

Check that we are received from the radio at lest once.

Warning

Will still return True if the radio is disconnected after some time.

Returns

true is the radio has been seen else false

5.3.2.2 bool current_freq_valid (struct radio_side * *radio*)

Returns

True if the radio is currently set to a frequency it can TX on

5.3.2.3 int get_display_packet (SERIAL_CFG * *config,* DISPLAY_PACKET *packet*)

writes the most recent packet to

Parameters

packet.	Gets the most recent packet. Will block if there is currently a half updated most recent packet

5.3.2.4 const struct range_KHz* get_range (int *frequency_khz*)

Returns (struct range_KHz) if input is within that range. Else returns NULL Looks for tx&rx ranges first

5.3.2.5 int in_freq_range (int *frequency_khz*)

returns 0 if invalid range, * or 1 if only rx allowed, * and 2 for all allowed

5.3.2.6 void* receive_display_packets (void * *c*)

Writes the latest packet to a segment of memory This function is designed to be started as a thread

5.3.2.7 void* send_control_packets (void * c)

Starts sending control packets as defined by SERIAL_CFG

This function is designed to be started as a thread

5.3.2.8 void send_new_packet (SERIAL_CFG * config, struct control_packet * new_packet, enum pop_queue_behaviour free_choice)

Adds a control_packet (pointer) to the send queue, should only be called once the queue has been initialized

5.3.2.9 int set_frequency (SERIAL_CFG * cfg, struct control_packet * base_packet, int number)

Adds the required packets to dial a number.

They are then be added to the queue

Returns

0 on success

5.3.2.10 int set_left_power_level (SERIAL_CFG * cfg, struct control_packet * base_packet, enum rt8900_power_level power_level)

sets left power level on radio

Presses the Low button on the radio until the selected power is set

5.3.2.11 int set_main_radio (SERIAL_CFG * cfg, struct control_packet * base_packet, enum radios side)

Switches context to the desired radio.

First check you are not already on this mode using is_main() else you will enter frequency edit mode!

5.3.2.12 int set_right_power_level (SERIAL_CFG * cfg, struct control_packet * base_packet, enum rt8900_power_level power_level)

sets right power level on radio

Presses the Low button on the radio until the selected power is set

Returns

0 on sucess, 1 on internal error and 2 on user error

5.3.2.13 void shutdown_threads (SERIAL_CFG * *cfg*)

gracefully stops send_control_packets() and receive_display_packets()

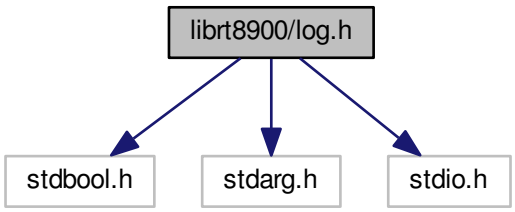
This can be used anytime to gracefully stop sending and receiving on serial Threads will be able to join after running ths function

5.4 librt8900/log.h File Reference

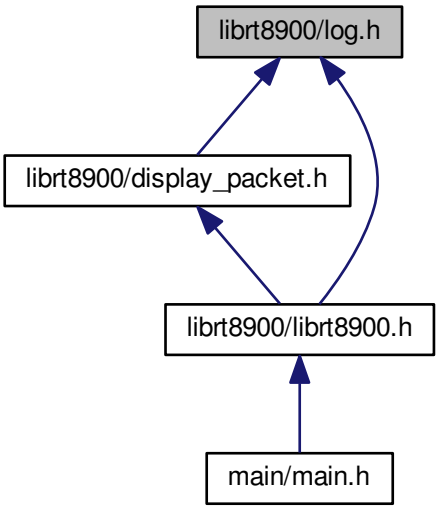
logging wrapper with levels for the librt8900 library.

```
#include <stdbool.h>
#include <stdarg.h>
#include <stdio.h>
```

Include dependency graph for log.h:



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



Enumerations

- enum **rt8900_logging_level** {
  RT8900_NOLOG = 0, **RT8900_FATAL**, **RT8900_ERROR**, **RT8900_WARNING**,
  RT8900_INFO, **RT8900_DEBUG**, **RT8900_TRACE** }

Functions

- void set_log_level (enum rt8900_logging_level i)

  sets the global logging level of lib rt8900
- void log_msg (enum rt8900_logging_level level, char const ∗fmt,...)

  Used in place of println for logging.

5.4.1 Detailed Description

logging wrapper with levels for the librt8900 library.

Contains functions that use both the control_packet and DISPLAY_PACKET.

5.4.2 Function Documentation

5.4.2.1 void log_msg (enum rt8900_logging_level *level,* char const ∗ *fmt,* ...)

Used in place of println for logging.

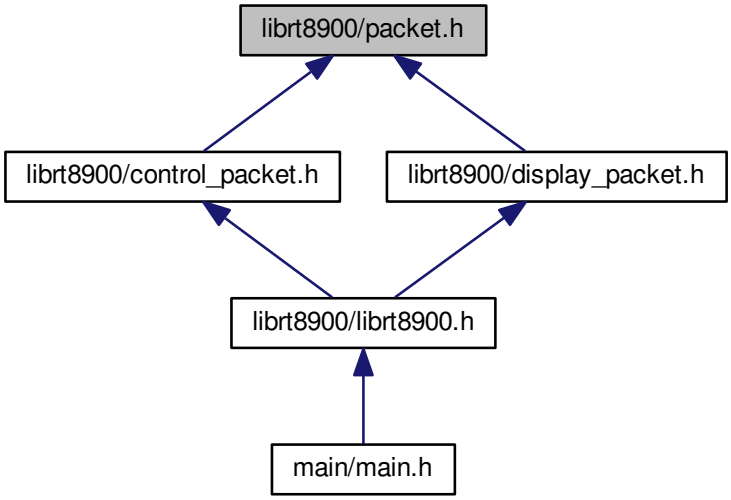
Parameters

level	is the log level this message should appear at

5.5 librt8900/packet.h File Reference

Data representation of a single byte in a packet.

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



Data Structures

- struct FT8900BYTE
- union PACKET_BYTE

  Used to store 1 byte of data from the packet.

Macros

- #define **NUM_DATA_BITS** 8
- #define **NUM_STOP_BITS** 1
- #define **NUM_PARITY_BITS** 0

Enumerations

- enum check_num_values { **SBZ** = 0X00, **SBO** = 0X01 }
- enum common_7bit_data_values { **DATA_MAX_NUM** = 0x7F, **DATA_MIN_NUM** = 0X00 }

  The highest and lowest possible values that can be held in the data section of the packet.

Functions

- void **print_char** (char byte)
- int find_packet_start (unsigned char buffer[ ], size_t length)

5.5.1 Detailed Description

Data representation of a single byte in a packet.

5.5.2 Enumeration Type Documentation

5.5.2.1 enum check_num_values

The last bit in a packet byte. 1 if it is the first in the packet.

5.5.3 Function Documentation

5.5.3.1 int find_packet_start (unsigned char *buffer[ ],* size_t *length*)

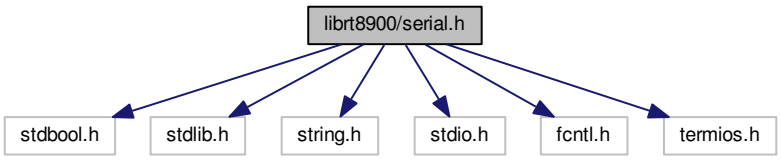
the start of the known packet could be anywhere in the buffer this function finds the starting index based of it's bit marker

5.6 librt8900/serial.h File Reference

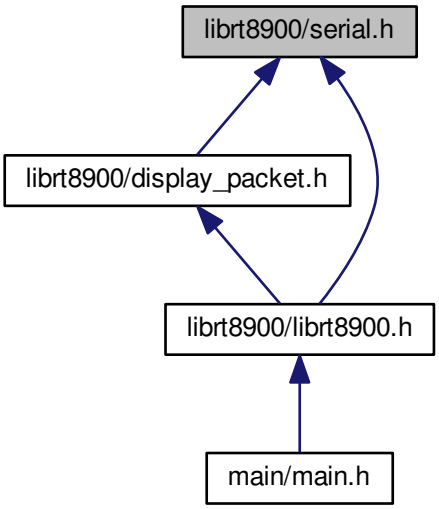
Serial handling.

```
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <termios.h>
```

Include dependency graph for serial.h:



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



Functions

- void open_serial (int *fd, char **serial_path, bool *dtr_pin_for_on)

  Open and configure a serial port for sending and receiving from radio.

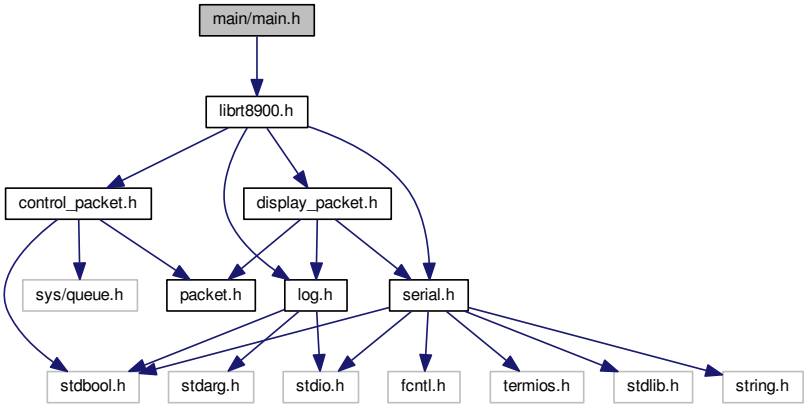
5.6.1 Detailed Description

Serial handling.

5.7 main/main.h File Reference

```
#include "librt8900.h"
```

Include dependency graph for main.h:



Macros

- #define **PROMPT_BUFFER_SIZE** 32
- #define **TURN_ON_RADIO_TRYS** 3
- #define **ANSI_COLOR_YELLOW** "\x1b[33m"
- #define **ANSI_COLOR_GREEN** "\x1b[32m"
- #define **ANSI_COLOR_RESET** "\x1b[0m"

Functions

- int **cmd_help** (char **args, SERIAL_CFG *config, struct control_packet *base_packet)
- int **cmd_exit** (char **args, SERIAL_CFG *config, struct control_packet *base_packet)
- int **cmd_get_frequency** (char **args, SERIAL_CFG *config, struct control_packet *base_packet)
- int **cmd_set_frequency** (char **args, SERIAL_CFG *config, struct control_packet *base_packet)
- int **cmd_get_busy** (char **args, SERIAL_CFG *config, struct control_packet *base_packet)
- int **cmd_get_main** (char **args, SERIAL_CFG *config, struct control_packet *base_packet)
- int **cmd_set_main** (char **args, SERIAL_CFG *config, struct control_packet *base_packet)
- int **cmd_get_power** (char **args, SERIAL_CFG *config, struct control_packet *base_packet)
- int **cmd_set_power** (char **args, SERIAL_CFG *config, struct control_packet *base_packet)
- int **cmd_get_ptt** (char **args, SERIAL_CFG *config, struct control_packet *base_packet)
- int **cmd_set_ptt** (char **args, SERIAL_CFG *config, struct control_packet *base_packet)
- int **cmd_set_volume** (char **args, SERIAL_CFG *config, struct control_packet *base_packet)
- int **cmd_set_squelch** (char **args, SERIAL_CFG *config, struct control_packet *base_packet)

5.7.1 Detailed Description

A command line shell for controlling the YAESU FT-8900R Transceiver.

Index