# Documentación Api Gateway

Generated by Doxygen 1.9.8

# Chapter 1

# Class Index

## 1.1 Class List

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

# Chapter 2

# File Index

## 2.1 File List

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

# Chapter 3

# Class Documentation

## 3.1 api_request_tracker_t Struct Reference

Tracks information from an original client (elevator) request OR a gateway-originated request.

`#include <api_handlers.h>`

### Public Attributes

- coap_session_t * original_elevator_session
- coap_binary_t original_token
- coap_mid_t original_mid
- char * log_tag
- gw_request_type_t request_type
- int origin_floor
- int target_floor_for_task
- char requesting_elevator_id_cabin [ID_STRING_MAX_LEN]
- movement_direction_enum_t requested_direction_floor

### 3.1.1 Detailed Description

Tracks information from an original client (elevator) request OR a gateway-originated request.

This structure is used to maintain state across the two-stage request process:

1. Event (simulated client or internal) -> Gateway
2. Gateway -> Central Server It allows the Gateway to correctly respond to the original client (if any) or update its internal state after receiving a response from the Central Server.

### 3.1.2 Member Data Documentation

#### 3.1.2.1 log_tag

`char* api_request_tracker_t::log_tag`

Tag for logging (e.g., "FloorCall", duplicated).

#### 3.1.2.2 origin_floor

```
int api_request_tracker_t::origin_floor
```

Piso origen (para GW_REQUEST_TYPE_FLOOR_CALL).

#### 3.1.2.3 original_elevator_session

```
coap_session_t* api_request_tracker_t::original_elevator_session
```

The CoAP session with the original elevator client (NULL if gateway-originated internal event).

#### 3.1.2.4 original_mid

```
coap_mid_t api_request_tracker_t::original_mid
```

The CoAP Message ID from the original client's request.

#### 3.1.2.5 original_token

```
coap_binary_t api_request_tracker_t::original_token
```

The CoAP token from the original client's request (duplicated).

#### 3.1.2.6 request_type

```
gw_request_type_t api_request_tracker_t::request_type
```

Tipo de solicitud gestionada por el gateway.

#### 3.1.2.7 requested_direction_floor

```
movement_direction_enum_t api_request_tracker_t::requested_direction_floor
```

Dirección solicitada (para GW_REQUEST_TYPE_FLOOR_CALL).

#### 3.1.2.8 requesting_elevator_id_cabin

```
char api_request_tracker_t::requesting_elevator_id_cabin[ID_STRING_MAX_LEN]
```

ID del ascensor (para GW_REQUEST_TYPE_CABIN_REQUEST).

#### 3.1.2.9 target_floor_for_task

```
int api_request_tracker_t::target_floor_for_task
```

Piso destino para la tarea a asignar (puede ser piso_origen para floor_call o destino_cabina para cabin_request).

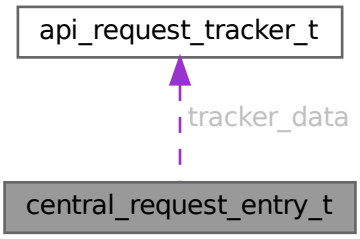
The documentation for this struct was generated from the following file:

- src/api_handlers.h

## 3.2 central_request_entry_t Struct Reference

Entrada de tracker para solicitudes al servidor central.

Collaboration diagram for central_request_entry_t:




### Public Attributes

- coap_bin_const_t **token**

  *Token de la solicitud enviada AL SERVIDOR CENTRAL.*
- api_request_tracker_t * **tracker_data**

  *Datos del tracker original (sesión del ascensor, token original, etc.)*

### 3.2.1 Detailed Description

Entrada de tracker para solicitudes al servidor central.

Estructura que asocia un token de solicitud enviada al servidor central con los datos del tracker original que contiene información de la solicitud inicial del ascensor.

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

- src/api_handlers.c

## 3.3 psk_keys_t Struct Reference

Estructura para almacenar las claves PSK.

### Public Attributes

- char ** **keys**

  *Array dinámico de strings con las claves PSK.*
- int **count**

  *Número actual de claves cargadas.*
- int **capacity**

  *Capacidad máxima del array (para futuras expansiones)*

### 3.3.1 Detailed Description

Estructura para almacenar las claves PSK.

Esta estructura gestiona dinámicamente un array de claves PSK cargadas desde un archivo de configuración.

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

- src/psk_manager.c

# Chapter 4

# File Documentation

## 4.1 src/api_handlers.c File Reference

Implementación de manejadores CoAP para el API Gateway.

```
#include <coap3/coap.h>
#include <coap3/coap_pdu.h>
#include <coap3/coap_option.h>
#include <coap3/coap_uri.h>
#include <coap3/coap_session.h>
#include <coap3/coap_net.h>
#include "api_gateway/api_handlers.h"
#include "api_gateway/elevator_state_manager.h"
#include "api_gateway/can_bridge.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <ctype.h>
#include "api_gateway/execution_logger.h"
```

Include dependency graph for api_handlers.c:

### Classes

- struct central_request_entry_t

  *Entrada de tracker para solicitudes al servidor central.*

### Macros

- #define MAX_PENDING_REQUESTS_TO_CENTRAL 32

  *Número máximo de solicitudes pendientes al servidor central.*

### Functions

- void handle_sigint_gw (int signum)

  *Manejador de señal para SIGINT (Ctrl+C)*
- coap_response_t hnd_central_server_response_gw (coap_session_t ∗session_from_server, const coap_↩ pdu_t ∗sent_to_central, const coap_pdu_t ∗received_from_central, const coap_mid_t mid_from_server)

  *Manejador de respuestas del servidor central.*
- void hnd_elevator_api_request_gw (coap_resource_t ∗resource, coap_session_t ∗elevator_session, const coap_pdu_t ∗elevator_request_pdu, const coap_string_t ∗query, coap_pdu_t ∗response_placeholder)

  *Handler para solicitudes legacy de ascensor.*
- void hnd_cabin_request_from_elevator_gw (coap_resource_t ∗resource, coap_session_t ∗elevator_session, const coap_pdu_t ∗elevator_request_pdu, const coap_string_t ∗query, coap_pdu_t ∗response_placeholder)

  *Handler para solicitudes de cabina desde ascensor.*
- void hnd_floor_call_from_elevator_gw (coap_resource_t ∗resource, coap_session_t ∗elevator_session, const coap_pdu_t ∗elevator_request_pdu, const coap_string_t ∗query, coap_pdu_t ∗response_placeholder)

  *Handler para llamadas de piso desde ascensor.*

### Variables

- volatile sig_atomic_t quit_main_loop

  *Bandera para indicar si el bucle principal debe terminar.*
- elevator_group_state_t managed_elevator_group

  *Estado del grupo de ascensores gestionado por este gateway.*
- coap_session_t ∗ g_dtls_session_to_central_server

  *Sesión DTLS global con el servidor central.*

## 4.1.1 Detailed Description

Implementación de manejadores CoAP para el API Gateway.

**Author**

Sistema de Control de Ascensores

**Date**

2025

**Version**

2.0

Este archivo contiene la lógica para manejar:

- **Solicitudes entrantes**: Requests de clientes (ascensores) vía CoAP
- **Respuestas del servidor central**: Procesamiento de respuestas de asignación
- **Gestión de trackers**: Seguimiento de solicitudes pendientes
- **Manejo de señales**: Terminación elegante con SIGINT

- **Puente CAN-CoAP**: Integración con el sistema de comunicación CAN

Funcionalidades principales:

- Recepción de solicitudes de llamada de piso y cabina
- Reenvío de solicitudes al servidor central con estado de ascensores
- Gestión de sesiones DTLS para comunicación segura
- Tracking de solicitudes pendientes para correlación de respuestas
- Integración con el gestor de estado de ascensores

**See also**

api_handlers.h
elevator_state_manager.h
can_bridge.h
coap_config.h

### 4.1.2 Macro Definition Documentation

#### 4.1.2.1 MAX_PENDING_REQUESTS_TO_CENTRAL

```
#define MAX_PENDING_REQUESTS_TO_CENTRAL 32
```

Número máximo de solicitudes pendientes al servidor central.

Define el límite de solicitudes que pueden estar pendientes de respuesta del servidor central simultáneamente. Ajustar según la capacidad del sistema.

### 4.1.3 Function Documentation

#### 4.1.3.1 handle_sigint_gw()

```
void handle_sigint_gw (
            int signum )
```

Manejador de señal para SIGINT (Ctrl+C)

Signal handler for SIGINT (Ctrl+C) to allow graceful shutdown.

**Parameters**

| | |
|---|---|
| *signum* | Número de señal recibida (se espera SIGINT). No utilizado. |

Establece la bandera quit_main_loop a 1 para señalar al bucle principal de eventos que debe terminar, permitiendo una terminación elegante del API Gateway.

Esta función es registrada como manejador de SIGINT en main.c y proporciona una forma limpia de cerrar la aplicación.

#### 4.1.3.2 hnd_cabin_request_from_elevator_gw()

```
void hnd_cabin_request_from_elevator_gw (
            coap_resource_t * resource,
            coap_session_t * elevator_session,
            const coap_pdu_t * elevator_request_pdu,
            const coap_string_t * query,
            coap_pdu_t * response_placeholder )
```

Handler para solicitudes de cabina desde ascensor.

**Parameters**

| | |
|---|---|
| *resource* | Recurso CoAP asociado |
| *elevator_session* | Sesión CoAP del ascensor solicitante |
| *elevator_request_pdu* | PDU de la solicitud CoAP recibida |
| *query* | Cadena de consulta de la solicitud (si existe) |
| *response_placeholder* | PDU de respuesta (reservado para futuro uso) |

Este handler procesa solicitudes de cabina originadas desde el interior de los ascensores. Las solicitudes incluyen el piso destino solicitado por usuarios dentro del ascensor.

**Note**

Función stub - la funcionalidad principal se maneja vía puente CAN

**See also**

can_bridge.h

api_handlers.h

#### 4.1.3.3 hnd_central_server_response_gw()

```
coap_response_t hnd_central_server_response_gw (
            coap_session_t * session_from_server,
            const coap_pdu_t * sent_to_central,
            const coap_pdu_t * received_from_central,
            const coap_mid_t mid_from_server )
```

Manejador de respuestas del servidor central.

Handles responses received from the Central Server.

**Parameters**

| | |
|---|---|
| *session_from_server* | Sesión CoAP desde la cual se recibió la respuesta |
| *sent_to_central* | PDU que fue enviado al servidor central |
| *received_from_central* | PDU recibido del servidor central |
| *mid_from_server* | Message ID de la respuesta del servidor |

**Returns**

COAP_RESPONSE_OK si la respuesta se procesó correctamente

Esta función procesa las respuestas recibidas del servidor central para solicitudes de asignación de ascensores. Realiza las siguientes operaciones:

1. **Validación**: Verifica que se recibió un PDU válido
2. **Parsing JSON**: Extrae y parsea el payload JSON de la respuesta
3. **Correlación**: Encuentra el tracker original usando el token
4. **Procesamiento**: Extrae información de asignación (ascensor_asignado_id, tarea_id)
5. **Notificación CAN**: Envía respuesta al controlador CAN correspondiente
6. **Limpieza**: Libera recursos del tracker y memoria asociada

Maneja diferentes códigos de respuesta:

- 2.01 Created: Asignación exitosa
- 4.xx Client Error: Errores de solicitud
- 5.xx Server Error: Errores del servidor

**See also**

add_central_request_tracker()

find_and_remove_central_request_tracker()

can_bridge.h

#### 4.1.3.4 hnd_elevator_api_request_gw()

```
void hnd_elevator_api_request_gw (
            coap_resource_t * resource,
            coap_session_t * elevator_session,
            const coap_pdu_t * elevator_request_pdu,
            const coap_string_t * query,
            coap_pdu_t * response_placeholder )
```

Handler para solicitudes legacy de ascensor.

**Parameters**

| | |
|---|---|
| *resource* | Recurso CoAP asociado |
| *elevator_session* | Sesión CoAP del ascensor solicitante |
| *elevator_request_pdu* | PDU de la solicitud CoAP recibida |
| *query* | Cadena de consulta de la solicitud (si existe) |
| *response_placeholder* | PDU de respuesta (reservado para futuro uso) |

Este handler procesa solicitudes legacy de ascensores que utilizan el protocolo CoAP anterior. Actualmente implementado como stub para compatibilidad con versiones anteriores del sistema.

**Note**

Función stub - no implementa funcionalidad específica

**See also**

api_handlers.h

#### 4.1.3.5 hnd_floor_call_from_elevator_gw()

```
void hnd_floor_call_from_elevator_gw (
            coap_resource_t * resource,
            coap_session_t * elevator_session,
            const coap_pdu_t * elevator_request_pdu,
            const coap_string_t * query,
            coap_pdu_t * response_placeholder )
```

Handler para llamadas de piso desde ascensor.

**Parameters**

| | |
|---|---|
| *resource* | Recurso CoAP asociado |
| *elevator_session* | Sesión CoAP del ascensor solicitante |
| *elevator_request_pdu* | PDU de la solicitud CoAP recibida |
| *query* | Cadena de consulta de la solicitud (si existe) |
| *response_placeholder* | PDU de respuesta (reservado para futuro uso) |

Este handler procesa llamadas de piso que se originan desde botones externos en los pisos del edificio. Las solicitudes incluyen el piso origen y la dirección deseada (subir/bajar).

**Note**

Función stub - la funcionalidad principal se maneja vía puente CAN

**See also**

can_bridge.h

api_handlers.h

### 4.1.4 Variable Documentation

#### 4.1.4.1 g_dtls_session_to_central_server

```
coap_session_t* g_dtls_session_to_central_server  [extern]
```

Sesión DTLS global con el servidor central.

Declaración externa de la sesión DTLS definida en main.c. Se utiliza para enviar solicitudes al servidor central de manera eficiente reutilizando la misma conexión segura.

**See also**

main.c

get_or_create_central_server_dtls_session()

Mantiene la conexión segura DTLS con el servidor central para el envío de solicitudes de asignación de ascensores. Se reutiliza para múltiples solicitudes para eficiencia.

#### 4.1.4.2 managed_elevator_group

```
elevator_group_state_t managed_elevator_group  [extern]
```

Estado del grupo de ascensores gestionado por este gateway.

Declaración externa para acceder al estado del grupo de ascensores gestionado en main.c. Contiene información completa de todos los ascensores del edificio.

**See also**

elevator_group_state_t

main.c

Estado del grupo de ascensores gestionado por este gateway.

Contiene el estado completo de todos los ascensores del edificio gestionado por este API Gateway, incluyendo posiciones actuales, tareas asignadas, estados de puertas y disponibilidad.

**See also**

elevator_group_state_t

elevator_state_manager.h

#### 4.1.4.3 quit_main_loop

```
volatile sig_atomic_t quit_main_loop  [extern]
```

Bandera para indicar si el bucle principal debe terminar.

Esta variable se declara como extern en api_handlers.h y se define en main.c. Es establecida por el manejador de señal (handle_sigint_gw) para detener el bucle principal de eventos.

**See also**

handle_sigint_gw()

Bandera para indicar si el bucle principal debe terminar.

Esta bandera se establece a 1 por el manejador de señal SIGINT (handle_sigint_gw) para indicar que la aplicación debe terminar de manera elegante.

**See also**

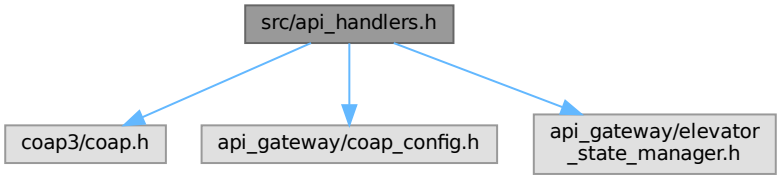
handle_sigint_gw()

## 4.2 src/api_handlers.h File Reference

Declares CoAP handler functions and related data structures for the API Gateway.

```
#include <coap3/coap.h>
#include "api_gateway/coap_config.h"
#include "api_gateway/elevator_state_manager.h"
```

Include dependency graph for api_handlers.h:




### Classes

- struct api_request_tracker_t

  *Tracks information from an original client (elevator) request OR a gateway-originated request.*

### Enumerations

- enum **gw_request_type_t** { **GW_REQUEST_TYPE_UNKNOWN** , **GW_REQUEST_TYPE_FLOOR_CALL** , **GW_REQUEST_TYPE_CABIN_REQUEST** }

### Functions

- void handle_sigint_gw (int signum)

  *Signal handler for SIGINT (Ctrl+C) to allow graceful shutdown.*
- coap_response_t hnd_central_server_response_gw (coap_session_t ∗session_to_central, const coap_↩ pdu_t ∗sent_to_central, const coap_pdu_t ∗received_from_central, const coap_mid_t mid_to_central)

  *Handles responses received from the Central Server.*

### 4.2.1 Detailed Description

Declares CoAP handler functions and related data structures for the API Gateway.

This file provides the interface for the CoAP NACK/response handler for client requests made by the gateway.

### 4.2.2 Function Documentation

#### 4.2.2.1 handle_sigint_gw()

```
void handle_sigint_gw (
            int signum )
```

Signal handler for SIGINT (Ctrl+C) to allow graceful shutdown.

**Parameters**

| | |
|---|---|
| *signum* | The signal number (unused). |

Signal handler for SIGINT (Ctrl+C) to allow graceful shutdown.

**Parameters**

| | |
|---|---|
| *signum* | Número de señal recibida (se espera SIGINT). No utilizado. |

Establece la bandera quit_main_loop a 1 para señalar al bucle principal de eventos que debe terminar, permitiendo una terminación elegante del API Gateway.

Esta función es registrada como manejador de SIGINT en main.c y proporciona una forma limpia de cerrar la aplicación.

#### 4.2.2.2 hnd_central_server_response_gw()

```
coap_response_t hnd_central_server_response_gw (
            coap_session_t * session_from_server,
            const coap_pdu_t * sent_to_central,
            const coap_pdu_t * received_from_central,
            const coap_mid_t mid_from_server )
```

Handles responses received from the Central Server.

This function is registered as a CoAP NACK/response handler for requests made by the API Gateway (acting as a client) to the Central Server. It processes the Central Server's response and forwards it to the original elevator client (if applicable) or updates internal state (e.g. for CAN originated).

**Parameters**

| | |
|---|---|
| *session_to_central* | The CoAP session between the API Gateway and the Central Server. |
| *sent_to_central* | The PDU that the API Gateway sent to the Central Server (can be NULL). |
| *received_from_central* | The PDU received from the Central Server. |
| *mid_to_central* | The CoAP Message ID of the exchange with the Central Server. |

**Returns**

COAP_RESPONSE_OK to indicate the response was handled.

Handles responses received from the Central Server.

**Parameters**

| | |
|---|---|
| *session_from_server* | Sesión CoAP desde la cual se recibió la respuesta |
| *sent_to_central* | PDU que fue enviado al servidor central |
| *received_from_central* | PDU recibido del servidor central |
| *mid_from_server* | Message ID de la respuesta del servidor |

**Returns**

COAP_RESPONSE_OK si la respuesta se procesó correctamente

Esta función procesa las respuestas recibidas del servidor central para solicitudes de asignación de ascensores. Realiza las siguientes operaciones:

1. **Validación**: Verifica que se recibió un PDU válido
2. **Parsing JSON**: Extrae y parsea el payload JSON de la respuesta
3. **Correlación**: Encuentra el tracker original usando el token
4. **Procesamiento**: Extrae información de asignación (ascensor_asignado_id, tarea_id)
5. **Notificación CAN**: Envía respuesta al controlador CAN correspondiente
6. **Limpieza**: Libera recursos del tracker y memoria asociada

Maneja diferentes códigos de respuesta:

- 2.01 Created: Asignación exitosa
- 4.xx Client Error: Errores de solicitud
- 5.xx Server Error: Errores del servidor

**See also**

add_central_request_tracker()

find_and_remove_central_request_tracker()

can_bridge.h

## 4.3 api_handlers.h

Go to the documentation of this file.

```
00001
00007 #ifndef API_GATEWAY_API_HANDLERS_H
00008 #define API_GATEWAY_API_HANDLERS_H
00009
00010 #include <coap3/coap.h>
00011 #include "api_gateway/coap_config.h" // Asegurar que coap_config.h esté incluido
00012 #include "api_gateway/elevator_state_manager.h" // Para enums y ID_STRING_MAX_LEN
00013
00014 // Resource paths the API Gateway would listen on - REMOVED FOR SIMPLIFICATION
00015 // #define GW_ELEVATOR_LEGACY_PATH "peticion_ascensor"
00016 // #define GW_CABIN_REQUEST_PATH "cabin_request_gw"
00017 // #define GW_FLOOR_CALL_PATH "floor_call_gw"
00018 // #define GW_ARRIVAL_NOTIFICATION_PATH "arrival_notification_gw" // Not used
00019
00020 // Tipo de solicitud originada/procesada por el Gateway
00021 typedef enum {
00022     GW_REQUEST_TYPE_UNKNOWN,
00023     GW_REQUEST_TYPE_FLOOR_CALL,
00024     GW_REQUEST_TYPE_CABIN_REQUEST
00025 } gw_request_type_t;
00026
00036 typedef struct {
00037     coap_session_t *original_elevator_session;
00038     coap_binary_t original_token;
00039     coap_mid_t original_mid;
00040     char *log_tag;
00042     // Nuevos campos para la lógica de gestión de estado del Gateway
00043     gw_request_type_t request_type;
00044     int origin_floor;
```

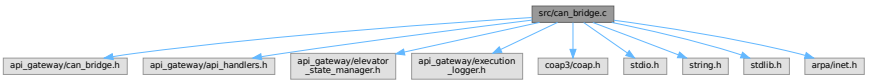
```
00045     int target_floor_for_task;
00046     char requesting_elevator_id_cabin[ID_STRING_MAX_LEN];
00047     movement_direction_enum_t requested_direction_floor;
00049 } api_request_tracker_t;
00050
00056 void handle_sigint_gw(int signum);
00057
00071 coap_response_t
00072 hnd_central_server_response_gw(coap_session_t *session_to_central,
00073                                const coap_pdu_t *sent_to_central,
00074                                const coap_pdu_t *received_from_central,
00075                                const coap_mid_t mid_to_central);
00076
00077 // HANDLERS FOR GATEWAY'S OWN RESOURCES - REMOVED FOR SIMPLIFICATION
00078 /*
00079 void
00080 hnd_elevator_api_request_gw(coap_resource_t *resource,
00081                             coap_session_t *elevator_session,
00082                             const coap_pdu_t *elevator_request_pdu,
00083                             const coap_string_t *query,
00084                             coap_pdu_t *response_placeholder);
00085
00086 void
00087 hnd_cabin_request_from_elevator_gw(coap_resource_t *resource,
00088                                    coap_session_t *elevator_session,
00089                                    const coap_pdu_t *elevator_request_pdu,
00090                                    const coap_string_t *query,
00091                                    coap_pdu_t *response_placeholder);
00092
00093 void
00094 hnd_floor_call_from_elevator_gw(coap_resource_t *resource,
00095                                 coap_session_t *elevator_session,
00096                                 const coap_pdu_t *elevator_request_pdu,
00097                                 const coap_string_t *query,
00098                                 coap_pdu_t *response_placeholder);
00099
00100 void
00101 hnd_arrival_notification_from_elevator_gw(coap_resource_t *resource,
00102                                           coap_session_t *elevator_session,
00103                                           const coap_pdu_t *elevator_request_pdu,
00104                                           const coap_string_t *query,
00105                                           coap_pdu_t *response_placeholder);
00106 */
00107
00108 #endif // API_GATEWAY_API_HANDLERS_H
```

## 4.4 src/can_bridge.c File Reference

Implementación del Puente CAN-CoAP para el API Gateway.

```
#include "api_gateway/can_bridge.h"
#include "api_gateway/api_handlers.h"
#include "api_gateway/elevator_state_manager.h"
#include "api_gateway/execution_logger.h"
#include <coap3/coap.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <arpa/inet.h>
```

Include dependency graph for can_bridge.c:




### Macros

- #define CAN_MAX_DATA_LEN 8

*Longitud máxima de datos en un frame CAN estándar.*

- #define MAX_CAN_ORIGIN_TRACKERS 10

  *Número máximo de trackers CAN simultáneos.*

### Functions

- can_origin_tracker_t * find_can_tracker (coap_bin_const_t token)

  *Busca un tracker CAN por token CoAP.*
- void ag_can_bridge_init (void)

  *Inicializa el puente CAN-CoAP.*
- void ag_can_bridge_register_send_callback (can_send_callback_t callback)

  *Registra el callback para envío de frames CAN al simulador.*
- void ag_can_bridge_process_incoming_frame (simulated_can_frame_t *frame, coap_context_t *coap_ctx)

  *Procesa un frame CAN entrante y lo convierte a solicitud CoAP.*
- void **ag_can_bridge_send_response_frame** (uint32_t original_can_id, coap_pdu_code_t response_code, cJSON *server_response_json)

  *Envía una respuesta (traducida de CoAP) como un frame CAN simulado a la simulación.*

### Variables

- elevator_group_state_t managed_elevator_group

  *Estado del grupo de ascensores gestionado.*

## 4.4.1 Detailed Description

Implementación del Puente CAN-CoAP para el API Gateway.

**Author**

Sistema de Control de Ascensores

**Date**

2025

**Version**

2.0

Este archivo implementa el puente de comunicación entre el protocolo CAN (Controller Area Network) y CoAP (Constrained Application Protocol) para el sistema de control de ascensores. Sus funciones principales:

- **Procesamiento de frames CAN**: Interpretación de mensajes CAN entrantes
- **Conversión CAN-CoAP**: Transformación de solicitudes CAN a CoAP
- **Gestión de trackers**: Seguimiento de solicitudes originadas por CAN
- **Respuestas CAN**: Envío de respuestas del servidor central vía CAN
- **Simulación**: Interfaz con simulador de red CAN para testing

- **Logging**: Registro detallado de operaciones para debugging

Tipos de mensajes CAN soportados:

- **0x100**: Llamadas de piso (floor calls) con dirección
- **0x200**: Solicitudes de cabina (cabin requests) con destino
- **0x300**: Notificaciones de llegada de ascensores

El puente mantiene un buffer circular de trackers para correlacionar respuestas del servidor central con las solicitudes CAN originales.

**See also**

can_bridge.h

api_handlers.h

elevator_state_manager.h

coap_config.h

## 4.4.2 Macro Definition Documentation

### 4.4.2.1 CAN_MAX_DATA_LEN

```
#define CAN_MAX_DATA_LEN 8
```

Longitud máxima de datos en un frame CAN estándar.

Define el número máximo de bytes de datos que puede contener un frame CAN estándar según la especificación CAN 2.0.

### 4.4.2.2 MAX_CAN_ORIGIN_TRACKERS

```
#define MAX_CAN_ORIGIN_TRACKERS 10
```

Número máximo de trackers CAN simultáneos.

Define el tamaño del buffer circular utilizado para almacenar información de solicitudes originadas por CAN que están pendientes de respuesta del servidor central.

### 4.4.3 Function Documentation

#### 4.4.3.1 ag_can_bridge_init()

```
void ag_can_bridge_init (
            void  )
```

Inicializa el puente CAN-CoAP.

Esta función inicializa el sistema de puente CAN-CoAP, preparando todas las estructuras de datos necesarias para el funcionamiento.

Operaciones realizadas:

- Limpia el callback de envío CAN
- Libera memoria de tokens almacenados en trackers
- Inicializa a cero el buffer de trackers CAN
- Resetea el índice del buffer circular

Debe llamarse una vez al inicio del programa antes de procesar cualquier frame CAN o registrar callbacks.

**See also**

ag_can_bridge_register_send_callback()

ag_can_bridge_process_incoming_frame()

#### 4.4.3.2 ag_can_bridge_process_incoming_frame()

```
void ag_can_bridge_process_incoming_frame (
            simulated_can_frame_t * frame,
            coap_context_t * coap_ctx )
```

Procesa un frame CAN entrante y lo convierte a solicitud CoAP.

**Parameters**

| | |
|---|---|
| *frame* | Puntero al frame CAN simulado a procesar |
| *coap_ctx* | Contexto CoAP para enviar solicitudes al servidor central |

Esta función es el punto de entrada principal para el procesamiento de frames CAN entrantes. Interpreta el contenido del frame según el esquema de mensajes CAN definido y genera las solicitudes CoAP correspondientes al servidor central.

**Tipos de frames CAN soportados:**

- **0x100 - Llamada de piso**:
  - data[0]: Piso origen (0-255)

  - data[1]: Dirección (0=UP, 1=DOWN)

- **0x200 - Solicitud de cabina**:
  - data[0]: Índice del ascensor (0-based)
  - data[1]: Piso destino (0-255)

- **0x300 - Notificación de llegada**:
  - data[0]: Índice del ascensor (0-based)
  - data[1]: Piso actual (0-255)

Para cada frame válido, la función:

1. Valida el formato y longitud de datos
2. Extrae los parámetros específicos del tipo de mensaje
3. Genera una solicitud CoAP al servidor central
4. Almacena un tracker para correlacionar la respuesta

**Note**

Los IDs CAN y formato de datos deben adaptarse según el esquema específico del sistema de ascensores

**See also**

forward_can_originated_request_to_central_server()

store_can_tracker()

simulated_can_frame_t

#### 4.4.3.3 ag_can_bridge_register_send_callback()

```
void ag_can_bridge_register_send_callback (
            can_send_callback_t callback )
```

Registra el callback para envío de frames CAN al simulador.

**Parameters**

| | |
|---|---|
| *callback* | Función callback que será llamada para enviar frames CAN |

Esta función registra una función callback que será utilizada por el puente CAN-CoAP para enviar frames CAN de respuesta al simulador.

El callback debe implementar la interfaz can_send_callback_t y será invocado cuando el puente necesite enviar respuestas del servidor central de vuelta al sistema CAN simulado.

**Note**

Solo se puede registrar un callback a la vez. Llamadas posteriores sobrescriben el callback anterior.

**See also**

can_send_callback_t

ag_can_bridge_send_response_frame()

#### 4.4.3.4 find_can_tracker()

```
can_origin_tracker_t * find_can_tracker (
            coap_bin_const_t token )
```

Busca un tracker CAN por token CoAP.

**Parameters**

| | |
|---|---|
| *token* | Token CoAP a buscar en los trackers almacenados |

**Returns**

Puntero al tracker encontrado, o NULL si no se encuentra

Esta función busca en el buffer circular de trackers CAN un tracker que corresponda al token CoAP especificado. Se utiliza para correlacionar respuestas del servidor central con solicitudes CAN originales.

La búsqueda se realiza comparando tanto la longitud como el contenido binario del token. No remueve el tracker de la lista (a diferencia de find_and_remove_central_request_tracker).

**See also**

store_can_tracker()

can_origin_tracker_t

### 4.4.4 Variable Documentation

#### 4.4.4.1 managed_elevator_group

```
elevator_group_state_t managed_elevator_group  [extern]
```

Estado del grupo de ascensores gestionado.

Referencia externa al estado global del grupo de ascensores definido en main.c. Se utiliza para acceder a información del edificio y ascensores al procesar mensajes CAN.

**See also**

main.c

elevator_group_state_t

Estado del grupo de ascensores gestionado.

Estado del grupo de ascensores gestionado por este gateway.

Contiene el estado completo de todos los ascensores del edificio gestionado por este API Gateway, incluyendo posiciones actuales, tareas asignadas, estados de puertas y disponibilidad.

**See also**

elevator_group_state_t

elevator_state_manager.h

Estado del grupo de ascensores gestionado.

Declaración externa para acceder al estado del grupo de ascensores gestionado en main.c. Contiene información completa de todos los ascensores del edificio.

**See also**

elevator_group_state_t

main.c

Estado del grupo de ascensores gestionado por este gateway.

Contiene el estado completo de todos los ascensores del edificio gestionado por este API Gateway, incluyendo posiciones actuales, tareas asignadas, estados de puertas y disponibilidad.

**See also**

elevator_group_state_t

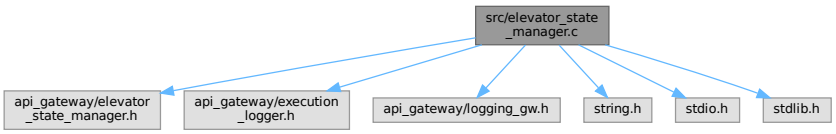
elevator_state_manager.h

## 4.5 src/elevator_state_manager.c File Reference

Implementación del Gestor de Estado de Ascensores del API Gateway.

```
#include "api_gateway/elevator_state_manager.h"
#include "api_gateway/execution_logger.h"
#include "api_gateway/logging_gw.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for elevator_state_manager.c:



### Functions

- const char * door_state_to_string (door_state_enum_t state)

  *Convierte un estado de puerta a su representación en string.*
- const char * movement_direction_to_string (movement_direction_enum_t direction)

  *Convierte una dirección de movimiento a su representación en string.*
- void init_elevator_group (elevator_group_state_t *group, const char *edificio_id_str, int num_elevadores, int num_pisos)

  *Inicializa un grupo de ascensores con configuración específica.*
- cJSON * elevator_group_to_json_for_server (const elevator_group_state_t *group, gw_request_type_t request_type, const api_request_details_for_json_t *details)

  *Serializa el estado del grupo de ascensores a JSON para el servidor central.*
- void assign_task_to_elevator (elevator_group_state_t *group, const char *elevator_id_to_update, const char *task_id, int target_floor, int current_request_floor)

  *Actualiza el estado de un ascensor tras recibir asignación de tarea.*

## 4.5.1 Detailed Description

Implementación del Gestor de Estado de Ascensores del API Gateway.

**Author**

Sistema de Control de Ascensores

**Date**

2025

**Version**

2.0

Este archivo implementa la gestión completa del estado de ascensores en el API Gateway, proporcionando funcionalidades para:

- **Inicialización de grupos**: Configuración inicial de grupos de ascensores
- **Gestión de estado**: Mantenimiento del estado actual de cada ascensor
- **Asignación de tareas**: Asignación de tareas a ascensores específicos
- **Notificaciones de llegada**: Procesamiento de llegadas a destinos
- **Serialización JSON**: Conversión del estado a formato JSON para servidor central
- **Utilidades de conversión**: Funciones helper para strings de estado

El gestor mantiene información completa de cada ascensor incluyendo:

- Posición actual (piso)
- Estado de puertas (abierta/cerrada/abriendo/cerrando)
- Dirección de movimiento (subiendo/bajando/parado)
- Tarea actual asignada
- Destino actual
- Estado de ocupación

**See also**

elevator_state_manager.h

api_common_defs.h

api_handlers.h

## 4.5.2 Function Documentation

### 4.5.2.1 assign_task_to_elevator()

```
void assign_task_to_elevator (
            elevator_group_state_t * group,
            const char * elevator_id_to_update,
            const char * task_id,
            int target_floor,
            int current_request_floor )
```

Actualiza el estado de un ascensor tras recibir asignación de tarea.

**Parameters**

| | |
|---|---|
| *group* | Puntero al grupo de ascensores |
| *elevator_id_to_update* | ID del ascensor a actualizar |
| *task_id* | El nuevo ID de tarea asignado |
| *target_floor* | El piso destino para esta tarea |
| *current_request_floor* | El piso donde se originó la solicitud |

Esta función actualiza el estado de un ascensor específico después de recibir una asignación de tarea del servidor central. Realiza:

**Operaciones realizadas:**

- Busca el ascensor por ID en el grupo
- Asigna la nueva tarea y piso destino
- Marca el ascensor como ocupado
- Determina y actualiza la dirección de movimiento
- Registra la asignación en el sistema de logging

La dirección de movimiento se calcula comparando el piso actual del ascensor con el piso destino de la tarea.

**See also**

elevator_group_state_t

elevator_status_t

exec_logger_log_task_assigned()

### 4.5.2.2 door_state_to_string()

```
const char * door_state_to_string (
            door_state_enum_t state )
```

Convierte un estado de puerta a su representación en string.

**Parameters**

| | |
|---|---|
| *state* | Estado de puerta a convertir |

**Returns**

String representando el estado de la puerta

Esta función proporciona una representación legible del estado de las puertas del ascensor para logging y serialización JSON.

Estados soportados:

- DOOR_CLOSED: "CERRADA"
- DOOR_OPEN: "ABIERTA"
- DOOR_OPENING: "ABRIENDO"
- DOOR_CLOSING: "CERRANDO"
- Otros: "DESCONOCIDO"

#### 4.5.2.3 elevator_group_to_json_for_server()

```
cJSON * elevator_group_to_json_for_server (
            const elevator_group_state_t * group,
            gw_request_type_t request_type,
            const api_request_details_for_json_t * details )
```

Serializa el estado del grupo de ascensores a JSON para el servidor central.

**Parameters**

| | |
|---|---|
| *group* | Puntero al estado del grupo de ascensores |
| *request_type* | El tipo de la solicitud original que motiva este payload |
| *details* | Puntero a estructura con detalles específicos de la solicitud (puede ser NULL) |

**Returns**

Puntero a objeto cJSON que representa el payload, o NULL en caso de error

Esta función convierte el estado completo del grupo de ascensores y los detalles de la solicitud específica a un objeto JSON formateado como espera el servidor central para procesamiento de asignaciones.

**JSON generado incluye:**

- ID del edificio
- Detalles específicos de la solicitud (piso origen, destino, etc.)

- Array con estado de todos los ascensores del grupo
- Para cada ascensor: ID, piso actual, estado puertas, disponibilidad

El llamador es responsable de liberar el objeto cJSON con cJSON_Delete().

**See also**

gw_request_type_t

api_request_details_for_json_t

elevator_group_state_t

#### 4.5.2.4 init_elevator_group()

```
void init_elevator_group (
            elevator_group_state_t * group,
            const char * edificio_id_str,
            int num_elevadores,
            int num_pisos )
```

Inicializa un grupo de ascensores con configuración específica.

**Parameters**

| | |
|---|---|
| *group* | Puntero al grupo de ascensores a inicializar |
| *edificio_id_str* | ID del edificio al que pertenece el grupo |
| *num_elevadores* | Número de ascensores en el grupo |
| *num_pisos* | Número de pisos del edificio |

Esta función inicializa completamente un grupo de ascensores con la configuración especificada. Realiza las siguientes operaciones:

1. **Validación**: Verifica parámetros de entrada válidos
2. **Limpieza**: Inicializa la estructura a cero
3. **Configuración del grupo**: Establece ID del edificio y número de ascensores
4. **Inicialización individual**: Configura cada ascensor con:
    - ID único (formato: {edificio_id}A{numero})
    - Piso inicial: 0 (planta baja)
    - Puertas cerradas
    - Sin tarea asignada
    - Estado parado y disponible

**Note**

El número de ascensores debe estar entre 1 y MAX_ELEVATORS_PER_GATEWAY

**See also**

elevator_group_state_t

elevator_status_t

#### 4.5.2.5 movement_direction_to_string()

```
const char * movement_direction_to_string (
            movement_direction_enum_t direction )
```

Convierte una dirección de movimiento a su representación en string.

**Parameters**

| | |
|---|---|
| *direction* | Dirección de movimiento a convertir |

**Returns**

String representando la dirección de movimiento

Esta función proporciona una representación legible de la dirección de movimiento del ascensor para logging y serialización JSON.

Direcciones soportadas:

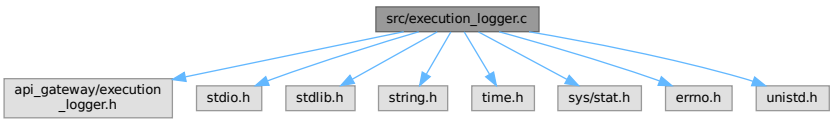
- MOVING_UP: "SUBIENDO"
- MOVING_DOWN: "BAJANDO"
- STOPPED: "PARADO"
- Otros: "DESCONOCIDO"

## 4.6 src/execution_logger.c File Reference

Implementación del Sistema de Logging de Ejecuciones.

```
#include "api_gateway/execution_logger.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/stat.h>
#include <errno.h>
#include <unistd.h>
```

Include dependency graph for execution_logger.c:




**Macros**

- #define **_POSIX_C_SOURCE** 199309L

### Functions

- bool exec_logger_init (void)

  *Inicializa el sistema de logging de ejecuciones.*
- void exec_logger_finish (void)

  *Finaliza el sistema de logging de ejecuciones.*
- void exec_logger_log_event (log_event_type_t type, const char *description, const char *details)

  *Registra un evento general en el sistema de logging.*
- void exec_logger_log_simulation_start (const char *building_id, int num_requests)

  *Registra el inicio de una simulación.*
- void exec_logger_log_simulation_end (int successful_requests, int total_requests)

  *Registra el fin de una simulación.*
- void exec_logger_log_can_sent (unsigned int can_id, int dlc, const unsigned char *data, const char *description)

  *Registra el envío de un frame CAN.*
- void exec_logger_log_can_received (unsigned int can_id, int dlc, const unsigned char *data, const char *description)

  *Registra la recepción de un frame CAN.*
- void exec_logger_log_coap_sent (const char *method, const char *uri, const char *payload)

  *Registra el envío de un mensaje CoAP.*
- void exec_logger_log_coap_received (const char *code, const char *payload)

  *Registra la recepción de un mensaje CoAP.*
- void exec_logger_log_task_assigned (const char *task_id, const char *elevator_id, int target_floor)

  *Registra la asignación de una tarea a un ascensor.*
- void exec_logger_log_elevator_moved (const char *elevator_id, int from_floor, int to_floor, const char *direction)

  *Registra el movimiento de un ascensor.*
- void exec_logger_log_task_completed (const char *task_id, const char *elevator_id, int final_floor)

  *Registra la finalización de una tarea de ascensor.*
- void exec_logger_log_error (const char *error_code, const char *error_message)

  *Registra un error del sistema.*
- const execution_stats_t * exec_logger_get_stats (void)

  *Obtiene las estadísticas actuales de ejecución.*
- bool exec_logger_is_active (void)

  *Verifica si el sistema de logging está activo.*

## 4.6.1 Detailed Description

Implementación del Sistema de Logging de Ejecuciones.

**Author**

Sistema de Control de Ascensores

**Date**

2025

**Version**

1.0

### 4.6.2 Function Documentation

#### 4.6.2.1 exec_logger_finish()

```
void exec_logger_finish (
            void  )
```

Finaliza el sistema de logging de ejecuciones.

Esta función cierra el sistema de logging escribiendo las estadísticas finales y cerrando el archivo de log. También intenta generar un PDF del reporte usando pandoc si está disponible.

**Operaciones realizadas:**

- Calcula la duración total de ejecución
- Escribe el footer Markdown con estadísticas finales
- Cierra el archivo de log
- Intenta generar PDF del reporte
- Marca el logger como inactivo

**See also**

exec_logger_init()
exec_logger_is_active()

#### 4.6.2.2 exec_logger_get_stats()

```
const execution_stats_t * exec_logger_get_stats (
            void  )
```

Obtiene las estadísticas actuales de ejecución.

**Returns**

Puntero a las estadísticas actuales, o NULL si el logger no está activo

Esta función proporciona acceso de solo lectura a las estadísticas actuales del sistema de logging para monitoreo en tiempo real.

**See also**

execution_stats_t
exec_logger_is_active()

#### 4.6.2.3 exec_logger_init()

```
bool exec_logger_init (
            void  )
```

Inicializa el sistema de logging de ejecuciones.

**Returns**

true si se inicializó correctamente, false en caso de error

Esta función inicializa el sistema de logging creando el directorio de logs necesario y abriendo el archivo de registro correspondiente.

**Operaciones realizadas:**

- Crea el directorio de logs con timestamp
- Abre el archivo de log con timestamp único
- Inicializa las estadísticas de ejecución
- Escribe el header Markdown del reporte
- Marca el logger como activo

**See also**

exec_logger_finish()
exec_logger_is_active()

#### 4.6.2.4 exec_logger_is_active()

```
bool exec_logger_is_active (
            void  )
```

Verifica si el sistema de logging está activo.

**Returns**

true si el logger está activo, false en caso contrario

Esta función permite verificar el estado del sistema de logging antes de intentar registrar eventos.

**See also**

exec_logger_init()
exec_logger_finish()

#### 4.6.2.5 exec_logger_log_can_received()

```
void exec_logger_log_can_received (
            unsigned int can_id,
            int dlc,
            const unsigned char * data,
            const char * description )
```

Registra la recepción de un frame CAN.

**Parameters**

| | |
|---|---|
| *can_id* | ID del frame CAN recibido |
| *dlc* | Data Length Code (número de bytes de datos) |
| *data* | Datos del frame CAN |
| *description* | Descripción del frame recibido |

Esta función registra la recepción de frames CAN con formato hexadecimal de los datos para análisis posterior.

**See also**

exec_logger_log_can_sent()

#### 4.6.2.6 exec_logger_log_can_sent()

```
void exec_logger_log_can_sent (
            unsigned int can_id,
            int dlc,
            const unsigned char * data,
            const char * description )
```

Registra el envío de un frame CAN.

**Parameters**

| | |
|---|---|
| *can_id* | ID del frame CAN enviado |
| *dlc* | Data Length Code (número de bytes de datos) |
| *data* | Datos del frame CAN |
| *description* | Descripción del frame enviado |

Esta función registra el envío de frames CAN con formato hexadecimal de los datos para análisis posterior.

**See also**

exec_logger_log_can_received()

#### 4.6.2.7 exec_logger_log_coap_received()

```
void exec_logger_log_coap_received (
            const char * code,
            const char * payload )
```

Registra la recepción de un mensaje CoAP.

**Parameters**

| | |
|---|---|
| *code* | Código de respuesta CoAP (2.01, 4.04, etc.) |
| *payload* | Payload del mensaje CoAP (puede ser NULL) |

Esta función registra la recepción de mensajes CoAP con información del código de respuesta y payload para análisis de tráfico.

**See also**

exec_logger_log_coap_sent()

#### 4.6.2.8 exec_logger_log_coap_sent()

```
void exec_logger_log_coap_sent (
            const char * method,
            const char * uri,
            const char * payload )
```

Registra el envío de un mensaje CoAP.

**Parameters**

| | |
|---|---|
| *method* | Método CoAP (GET, POST, PUT, DELETE) |
| *uri* | URI del recurso CoAP |
| *payload* | Payload del mensaje CoAP (puede ser NULL) |

Esta función registra el envío de mensajes CoAP con información completa del método, URI y payload para análisis de tráfico.

**See also**

exec_logger_log_coap_received()

#### 4.6.2.9 exec_logger_log_elevator_moved()

```
void exec_logger_log_elevator_moved (
            const char * elevator_id,
            int from_floor,
            int to_floor,
            const char * direction )
```

Registra el movimiento de un ascensor.

**Parameters**

| | |
|---|---|
| *elevator_id* | ID del ascensor que se movió |
| *from_floor* | Piso de origen del movimiento |
| *to_floor* | Piso de destino del movimiento |
| *direction* | Dirección del movimiento (UP/DOWN) |

Esta función registra los movimientos de ascensores entre pisos para análisis de tráfico y eficiencia del sistema.

**See also**

exec_logger_log_task_assigned()

#### 4.6.2.10 exec_logger_log_error()

```
void exec_logger_log_error (
            const char * error_code,
            const char * error_message )
```

Registra un error del sistema.

**Parameters**

| | |
|---|---|
| *error_code* | Código de error (ej: "COAP_001", "CAN_002") |
| *error_message* | Mensaje descriptivo del error |

Esta función registra errores del sistema para análisis posterior y incrementa el contador de errores en las estadísticas.

**See also**

exec_logger_log_event()

#### 4.6.2.11 exec_logger_log_event()

```
void exec_logger_log_event (
            log_event_type_t type,
            const char * description,
            const char * details )
```

Registra un evento general en el sistema de logging.

**Parameters**

| | |
|---|---|
| *type* | Tipo de evento a registrar |
| *description* | Descripción del evento |
| *details* | Detalles adicionales del evento (puede ser NULL) |

Esta función registra eventos generales del sistema con timestamp de alta precisión y formateo consistente.

**See also**

log_event_type_t

exec_logger_is_active()

#### 4.6.2.12 exec_logger_log_simulation_end()

```
void exec_logger_log_simulation_end (
            int successful_requests,
            int total_requests )
```

Registra el fin de una simulación.

**Parameters**

| | |
|---|---|
| *successful_requests* | Número de peticiones procesadas exitosamente |
| *total_requests* | Número total de peticiones procesadas |

Esta función registra el final de una simulación de ascensores con estadísticas de éxito y métricas de rendimiento.

**See also**

exec_logger_log_simulation_start()

#### 4.6.2.13 exec_logger_log_simulation_start()

```
void exec_logger_log_simulation_start (
            const char * building_id,
            int num_requests )
```

Registra el inicio de una simulación.

**Parameters**

| | |
|---|---|
| *building_id* | ID del edificio simulado |
| *num_requests* | Número de peticiones programadas para la simulación |

Esta función registra el inicio de una simulación de ascensores, almacenando información clave para las estadísticas finales.

**See also**

exec_logger_log_simulation_end()

#### 4.6.2.14 exec_logger_log_task_assigned()

```
void exec_logger_log_task_assigned (
            const char * task_id,
            const char * elevator_id,
            int target_floor )
```

Registra la asignación de una tarea a un ascensor.

**Parameters**

| | |
|---|---|
| *task_id* | ID de la tarea asignada |
| *elevator_id* | ID del ascensor asignado |
| *target_floor* | Piso destino de la tarea |

Esta función registra cuando el servidor central asigna una tarea específica a un ascensor del grupo gestionado por el gateway.

**See also**

exec_logger_log_task_completed()

#### 4.6.2.15 exec_logger_log_task_completed()

```
void exec_logger_log_task_completed (
            const char * task_id,
            const char * elevator_id,
            int final_floor )
```

Registra la finalización de una tarea de ascensor.

**Parameters**

| | |
|---|---|
| *task_id* | ID de la tarea completada |
| *elevator_id* | ID del ascensor que completó la tarea |
| *final_floor* | Piso final donde terminó la tarea |

Esta función registra cuando un ascensor completa exitosamente una tarea asignada, llegando al piso destino.

**See also**

exec_logger_log_task_assigned()

## 4.7 src/main.c File Reference

Punto de entrada principal del API Gateway del Sistema de Control de Ascensores.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include <stdbool.h>
```

```
#include <coap3/coap.h>
#include <coap3/coap_address.h>
#include <coap3/coap_event.h>
#include "api_gateway/api_handlers.h"
#include "api_gateway/elevator_state_manager.h"
#include "api_gateway/can_bridge.h"
#include <cJSON.h>
#include "api_gateway/logging_gw.h"
#include "dotenv.h"
#include <unistd.h>
#include "psk_manager.h"
#include "api_gateway/execution_logger.h"
```

Include dependency graph for main.c:

### Functions

- void inicializar_mi_simulacion_ascensor (void)

  *Inicializa el simulador de ascensores.*
- void simular_eventos_ascensor (void)

  *Ejecuta una secuencia de eventos simulados de ascensor desde JSON de forma no-bloqueante.*
- bool procesar_siguiente_peticion_simulacion (void)

  *Procesa la siguiente petición de la simulación no-bloqueante.*
- coap_session_t * get_or_create_central_server_dtls_session (coap_context_t *ctx)

  *Obtiene o crea una sesión DTLS con el servidor central.*
- int main (int argc, char *argv[ ])

  *Main function for the API Gateway.*

### Variables

- volatile sig_atomic_t quit_main_loop = 0

  *Bandera global para controlar el bucle principal de la aplicación.*
- elevator_group_state_t managed_elevator_group

  *Estado global del grupo de ascensores gestionado por este gateway.*
- coap_context_t * g_coap_context = NULL

  *Contexto CoAP global para la simulación y gestión de sesiones.*
- coap_session_t * g_dtls_session_to_central_server = NULL

  *Sesión DTLS global con el servidor central.*

## 4.7.1 Detailed Description

Punto de entrada principal del API Gateway del Sistema de Control de Ascensores.

**Author**

Sistema de Control de Ascensores

**Date**

2025

**Version**

2.0

Este archivo implementa el API Gateway que actúa como intermediario entre los controladores CAN de ascensores y el servidor central. Sus funciones principales:

- **Inicialización CoAP**: Configuración del contexto CoAP con soporte DTLS
- **Gestión de Recursos**: Registro de endpoints CoAP para recibir solicitudes
- **Puente CAN-CoAP**: Transformación de mensajes CAN a solicitudes CoAP
- **Gestión de Estado**: Mantenimiento del estado local de ascensores
- **Comunicación Segura**: Establecimiento de sesiones DTLS con servidor central
- **Simulación**: Simulación de movimiento de ascensores para testing
- **Manejo de Señales**: Terminación elegante con SIGINT

El gateway procesa dos tipos principales de solicitudes:

- Llamadas de piso (floor calls) desde botones externos
- Solicitudes de cabina (cabin requests) desde interior del ascensor

**See also**

api_handlers.h
elevator_state_manager.h
can_bridge.h
coap_config.h

## 4.7.2 Function Documentation

### 4.7.2.1 get_or_create_central_server_dtls_session()

```
coap_session_t * get_or_create_central_server_dtls_session (
            coap_context_t * ctx )
```

Obtiene o crea una sesión DTLS con el servidor central.

**Parameters**

| | |
|---|---|
| *ctx* | Contexto CoAP a utilizar para la sesión |

**Returns**

Puntero a la sesión DTLS establecida, o NULL en caso de error

Esta función implementa un patrón singleton para la gestión de sesiones DTLS con el servidor central. Reutiliza sesiones existentes cuando están activas y crea nuevas sesiones cuando es necesario.

Comportamiento:

1. Si existe una sesión activa y establecida, la reutiliza
2. Si la sesión existe pero no está establecida, la libera y crea una nueva
3. Si no existe sesión, crea una nueva con configuración DTLS-PSK
4. Registra el manejador de eventos para gestión automática de la sesión

La función utiliza las siguientes configuraciones:

- IP del servidor: CENTRAL_SERVER_IP
- Puerto del servidor: CENTRAL_SERVER_PORT
- Identidad PSK: IDENTITY_TO_PRESENT_TO_SERVER
- Clave PSK: KEY_FOR_SERVER

**See also**

event_handler_gw()
coap_config.h

#### 4.7.2.2 inicializar_mi_simulacion_ascensor()

```
void inicializar_mi_simulacion_ascensor (
            void  )
```

Inicializa el simulador de ascensores.

Esta función configura el simulador registrando el callback de respuesta CAN en el puente CAN-CoAP del API Gateway y cargando los datos de simulación desde el archivo JSON.

**Operaciones realizadas:**

- Registro del callback de respuesta CAN
- Carga de datos de simulación desde JSON
- Inicialización de estructuras internas
- Configuración de logging del simulador

**See also**

ag_can_bridge_register_send_callback()
cargar_datos_simulacion()
mi_simulador_recibe_can_gw()

#### 4.7.2.3 main()

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

Main function for the API Gateway.

Initializes libcoap, sets up the CoAP server endpoint for listening to elevator clients, registers CoAP resource handlers, and enters the main I/O processing loop. The gateway listens for client requests, forwards them to a central server, and relays responses back to the clients.

**Parameters**

| | |
|---|---|
| *argc* | Número de argumentos de línea de comandos |
| *argv* | Array de argumentos de línea de comandos argv[1] (opcional): Puerto de escucha (por defecto usa GW_LISTEN_PORT) |

**Returns**

EXIT_SUCCESS on successful execution and shutdown, EXIT_FAILURE on error.

#### 4.7.2.4 procesar_siguiente_peticion_simulacion()

```
bool procesar_siguiente_peticion_simulacion (
            void  )
```

Procesa la siguiente petición de la simulación no-bloqueante.

Esta función debe llamarse desde el main loop para ejecutar las peticiones de forma incremental, permitiendo que el simulador de movimiento funcione en paralelo.

**Returns**

true si la simulación continúa, false si ha terminado

#### 4.7.2.5 simular_eventos_ascensor()

```
void simular_eventos_ascensor (
            void  )
```

Ejecuta una secuencia de eventos simulados de ascensor desde JSON de forma no-bloqueante.

Esta función reemplaza la simulación bloqueante anterior. Ahora la simulación se ejecuta de forma incremental, permitiendo que el main loop procese I/O y ejecute la simulación de movimiento entre peticiones.

**Cambios principales:**

- Simulación no-bloqueante (una petición por llamada)
- Control regresa al main loop para permitir movimiento de ascensores

- Intervalo configurable entre peticiones
- Estado global para rastrear progreso

**Proceso de simulación:**

1. Inicialización: selecciona edificio y configura estado
2. Ejecución incremental: una petición por llamada
3. El main loop ejecuta tanto I/O como simulación de movimiento
4. Finalización automática al completar todas las peticiones

**Note**

Esta función debe llamarse desde el main loop para ser no-bloqueante

**See also**

simulate_elevator_group_step() - se ejecuta en paralelo

### 4.7.3 Variable Documentation

#### 4.7.3.1 g_coap_context

```
coap_context_t* g_coap_context = NULL
```

Contexto CoAP global para la simulación y gestión de sesiones.

Contexto CoAP global del API Gateway.

Contexto principal utilizado para todas las operaciones CoAP, incluyendo la simulación de ascensores y la gestión de sesiones DTLS.

#### 4.7.3.2 g_dtls_session_to_central_server

```
coap_session_t* g_dtls_session_to_central_server = NULL
```

Sesión DTLS global con el servidor central.

Mantiene la conexión segura DTLS con el servidor central para el envío de solicitudes de asignación de ascensores. Se reutiliza para múltiples solicitudes para eficiencia.

#### 4.7.3.3 managed_elevator_group

```
elevator_group_state_t managed_elevator_group
```

Estado global del grupo de ascensores gestionado por este gateway.

Grupo de ascensores gestionado.

Estado del grupo de ascensores gestionado.

Estado del grupo de ascensores gestionado por este gateway.

Contiene el estado completo de todos los ascensores del edificio gestionado por este API Gateway, incluyendo posiciones actuales, tareas asignadas, estados de puertas y disponibilidad.

**See also**

elevator_group_state_t

elevator_state_manager.h

#### 4.7.3.4 quit_main_loop

```
volatile sig_atomic_t quit_main_loop = 0
```

Bandera global para controlar el bucle principal de la aplicación.

Bandera para indicar si el bucle principal debe terminar.

Esta bandera se establece a 1 por el manejador de señal SIGINT (handle_sigint_gw) para indicar que la aplicación debe terminar de manera elegante.

**See also**

handle_sigint_gw()

## 4.8 src/main_dynamic_port.c File Reference

Versión del API Gateway que acepta puerto como parámetro.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include <coap3/coap.h>
#include <coap3/coap_address.h>
#include <coap3/coap_event.h>
#include "api_gateway/api_handlers.h"
#include "api_gateway/elevator_state_manager.h"
```

```
#include "api_gateway/can_bridge.h"
#include <cJSON.h>
#include "api_gateway/logging_gw.h"
#include "api_gateway/execution_logger.h"
#include "dotenv.h"
#include <unistd.h>
```

Include dependency graph for main_dynamic_port.c:

### Functions

- void inicializar_mi_simulacion_ascensor (void)

  *Inicializa el simulador de ascensores.*
- void simular_eventos_ascensor (void)

  *Ejecuta una secuencia de eventos simulados de ascensor desde JSON de forma no-bloqueante.*
- coap_session_t * get_or_create_central_server_dtls_session (coap_context_t *ctx)

  *Obtiene o crea una sesión DTLS con el servidor central.*
- int main (int argc, char *argv[ ])

  *Función principal del API Gateway con puerto dinámico.*

### Variables

- volatile sig_atomic_t quit_main_loop = 0

  *Bandera para indicar si el bucle principal debe terminar.*
- elevator_group_state_t managed_elevator_group

  *Estado del grupo de ascensores gestionado por este gateway.*
- coap_context_t * g_coap_context = NULL

  *Contexto CoAP global del API Gateway.*
- coap_session_t * g_dtls_session_to_central_server = NULL

  *Sesión DTLS global con el servidor central.*

## 4.8.1 Detailed Description

Versión del API Gateway que acepta puerto como parámetro.

Esta versión permite ejecutar múltiples instancias del API Gateway cada una escuchando en un puerto diferente para pruebas de carga.

Uso: ./api_gateway_dynamic [puerto_escucha] Si no se especifica puerto, usa 5683 por defecto.

## 4.8.2 Function Documentation

### 4.8.2.1 get_or_create_central_server_dtls_session()

```
coap_session_t * get_or_create_central_server_dtls_session (
            coap_context_t * ctx )
```

Obtiene o crea una sesión DTLS con el servidor central.

**Parameters**

| | |
|---|---|
| *ctx* | Contexto CoAP a utilizar para la sesión |

**Returns**

Puntero a la sesión DTLS establecida, o NULL en caso de error

Versión simplificada de la función para main_dynamic_port.c. Implementa un patrón singleton para la gestión de sesiones DTLS con el servidor central, reutilizando sesiones existentes cuando están activas y creando nuevas cuando es necesario.

**Comportamiento:**

- Verifica si existe una sesión activa y la reutiliza
- Si la sesión no está establecida, la libera y crea una nueva
- Utiliza identidad y clave PSK únicas para cada instancia
- Configura dirección del servidor desde variables de entorno

**See also**

coap_config.h
generate_unique_identity()
generate_unique_psk_key()

#### 4.8.2.2 inicializar_mi_simulacion_ascensor()

```
void inicializar_mi_simulacion_ascensor (
            void  )
```

Inicializa el simulador de ascensores.

Esta función configura el simulador registrando el callback de respuesta CAN en el puente CAN-CoAP del API Gateway y cargando los datos de simulación desde el archivo JSON.

**Operaciones realizadas:**

- Registro del callback de respuesta CAN
- Carga de datos de simulación desde JSON
- Inicialización de estructuras internas
- Configuración de logging del simulador

**See also**

ag_can_bridge_register_send_callback()
cargar_datos_simulacion()
mi_simulador_recibe_can_gw()

#### 4.8.2.3 main()

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

Función principal del API Gateway con puerto dinámico.

**Parameters**

| | |
|---|---|
| *argc* | Número de argumentos de línea de comandos |
| *argv* | Array de argumentos de línea de comandos argv[1] (opcional): Puerto de escucha personalizado |

**Returns**

EXIT_SUCCESS si la ejecución fue exitosa, EXIT_FAILURE en caso de error

Versión del API Gateway que permite especificar el puerto de escucha como parámetro de línea de comandos. Útil para ejecutar múltiples instancias del gateway en diferentes puertos.

**Funcionalidades:**

- Carga de configuración desde gateway.env
- Configuración de puerto dinámico via argumentos
- Inicialización del contexto CoAP y puente CAN
- Gestión de estado de ascensores
- Bucle principal de procesamiento de eventos
- Terminación elegante con SIGINT

**Uso:**

- `./main_dynamic_port` - Usa puerto por defecto (5683)
- `./main_dynamic_port 8080` - Usa puerto personalizado (8080)

**See also**

main.c

get_or_create_central_server_dtls_session()

#### 4.8.2.4 simular_eventos_ascensor()

```
void simular_eventos_ascensor (
            void  )
```

Ejecuta una secuencia de eventos simulados de ascensor desde JSON de forma no-bloqueante.

Esta función reemplaza la simulación bloqueante anterior. Ahora la simulación se ejecuta de forma incremental, permitiendo que el main loop procese I/O y ejecute la simulación de movimiento entre peticiones.

**Cambios principales:**

- Simulación no-bloqueante (una petición por llamada)
- Control regresa al main loop para permitir movimiento de ascensores

- Intervalo configurable entre peticiones
- Estado global para rastrear progreso

**Proceso de simulación:**

1. Inicialización: selecciona edificio y configura estado
2. Ejecución incremental: una petición por llamada
3. El main loop ejecuta tanto I/O como simulación de movimiento
4. Finalización automática al completar todas las peticiones

**Note**

Esta función debe llamarse desde el main loop para ser no-bloqueante

**See also**

simulate_elevator_group_step() - se ejecuta en paralelo

### 4.8.3 Variable Documentation

#### 4.8.3.1 g_coap_context

```
coap_context_t* g_coap_context = NULL
```

Contexto CoAP global del API Gateway.

Referencia externa al contexto CoAP principal definido en main.c del API Gateway. Utilizado para procesar eventos CoAP y enviar solicitudes al servidor central.

**See also**

api_gateway/main.c

Contexto CoAP global del API Gateway.

Contexto principal utilizado para todas las operaciones CoAP, incluyendo la simulación de ascensores y la gestión de sesiones DTLS.

#### 4.8.3.2 g_dtls_session_to_central_server

```
coap_session_t* g_dtls_session_to_central_server = NULL
```

Sesión DTLS global con el servidor central.

Declaración externa de la sesión DTLS definida en main.c. Se utiliza para enviar solicitudes al servidor central de manera eficiente reutilizando la misma conexión segura.

**See also**

main.c

get_or_create_central_server_dtls_session()

Mantiene la conexión segura DTLS con el servidor central para el envío de solicitudes de asignación de ascensores. Se reutiliza para múltiples solicitudes para eficiencia.

#### 4.8.3.3 managed_elevator_group

```
elevator_group_state_t managed_elevator_group
```

Estado del grupo de ascensores gestionado por este gateway.

Estado global del grupo de ascensores gestionado por este gateway.

Grupo de ascensores gestionado.

Estado del grupo de ascensores gestionado.

Declaración externa para acceder al estado del grupo de ascensores gestionado en main.c. Contiene información completa de todos los ascensores del edificio.

**See also**

elevator_group_state_t

main.c

Estado del grupo de ascensores gestionado por este gateway.

Contiene el estado completo de todos los ascensores del edificio gestionado por este API Gateway, incluyendo posiciones actuales, tareas asignadas, estados de puertas y disponibilidad.

**See also**

elevator_group_state_t

elevator_state_manager.h

#### 4.8.3.4 quit_main_loop

```
volatile sig_atomic_t quit_main_loop = 0
```

Bandera para indicar si el bucle principal debe terminar.

Esta variable se declara como extern en api_handlers.h y se define en main.c. Es establecida por el manejador de señal (handle_sigint_gw) para detener el bucle principal de eventos.

**See also**

handle_sigint_gw()

Bandera para indicar si el bucle principal debe terminar.

Esta bandera se establece a 1 por el manejador de señal SIGINT (handle_sigint_gw) para indicar que la aplicación debe terminar de manera elegante.

**See also**

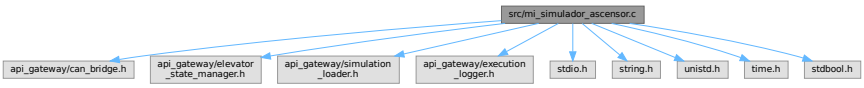
handle_sigint_gw()

## 4.9 src/mi_simulador_ascensor.c File Reference

Simulador de Ascensores para Testing del API Gateway.

```
#include "api_gateway/can_bridge.h"
#include "api_gateway/elevator_state_manager.h"
#include "api_gateway/simulation_loader.h"
#include "api_gateway/execution_logger.h"
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <stdbool.h>
```

Include dependency graph for mi_simulador_ascensor.c:




### Functions

- void mi_simulador_recibe_can_gw (simulated_can_frame_t *frame)

  *Callback para recibir frames CAN de respuesta del gateway.*
- void inicializar_mi_simulacion_ascensor (void)

  *Inicializa el simulador de ascensores.*
- void simular_llamada_de_piso_via_can (int piso_origen, movement_direction_enum_t direccion)

  *Simula una llamada de piso vía CAN.*
- void simular_solicitud_cabina_via_can (int indice_ascensor, int piso_destino)

  *Simula una solicitud de cabina vía CAN.*
- void simular_eventos_ascensor (void)

  *Ejecuta una secuencia de eventos simulados de ascensor desde JSON de forma no-bloqueante.*
- bool procesar_siguiente_peticion_simulacion (void)

  *Procesa la siguiente petición de la simulación no-bloqueante.*

### Variables

- coap_context_t * g_coap_context

  *Contexto CoAP global del API Gateway.*
- elevator_group_state_t managed_elevator_group

  *Grupo de ascensores gestionado.*

### 4.9.1 Detailed Description

Simulador de Ascensores para Testing del API Gateway.

**Author**

Sistema de Control de Ascensores

**Date**

2025

**Version**

2.0

Este archivo implementa un simulador de ascensores que genera eventos CAN simulados para probar la funcionalidad del API Gateway. Sus funciones principales:

- **Simulación de eventos CAN**: Generación de frames CAN simulados
- **Callback de respuesta**: Procesamiento de respuestas del gateway
- **Integración con puente CAN**: Interfaz con el sistema CAN-CoAP
- **Testing automatizado**: Secuencias de prueba predefinidas
- **Carga desde JSON**: Sistema de simulación basado en archivos de datos

**Tipos de eventos simulados:**

- Llamadas de piso (floor calls) desde botones externos
- Solicitudes de cabina (cabin requests) desde interior del ascensor
- Notificaciones de llegada a destinos

El simulador utiliza el contexto CoAP global del API Gateway para procesar eventos y recibir respuestas del servidor central.

**Sistema de simulación JSON:** Cada ejecución del API Gateway carga un archivo JSON con 100 edificios, selecciona uno aleatoriamente y ejecuta sus 10 peticiones secuencialmente.

**See also**

can_bridge.h

elevator_state_manager.h

simulation_loader.h

api_common_defs.h

### 4.9.2 Function Documentation

#### 4.9.2.1 inicializar_mi_simulacion_ascensor()

```
void inicializar_mi_simulacion_ascensor (
            void  )
```

Inicializa el simulador de ascensores.

Esta función configura el simulador registrando el callback de respuesta CAN en el puente CAN-CoAP del API Gateway y cargando los datos de simulación desde el archivo JSON.

**Operaciones realizadas:**

- Registro del callback de respuesta CAN
- Carga de datos de simulación desde JSON
- Inicialización de estructuras internas
- Configuración de logging del simulador

**See also**

ag_can_bridge_register_send_callback()
cargar_datos_simulacion()
mi_simulador_recibe_can_gw()

#### 4.9.2.2 mi_simulador_recibe_can_gw()

```
void mi_simulador_recibe_can_gw (
            simulated_can_frame_t * frame )
```

Callback para recibir frames CAN de respuesta del gateway.

**Parameters**

| | |
|---|---|
| *frame* | Puntero al frame CAN simulado recibido del gateway |

Esta función actúa como callback registrado en el puente CAN para recibir y procesar las respuestas del API Gateway. Interpreta diferentes tipos de frames CAN de respuesta:

**Tipos de frames procesados:**

- 0x101: Respuesta a llamada de piso (0x100)
- 0x201: Respuesta a solicitud de cabina (0x200)
- 0xFE: Error genérico del gateway

**Información extraída:**

- Índice del ascensor asignado
- ID de tarea (parcial, limitado por tamaño CAN)
- Códigos de error y diagnóstico

La función proporciona logging detallado para debugging y verificación del comportamiento del sistema.

**See also**

ag_can_bridge_register_send_callback()

simulated_can_frame_t

#### 4.9.2.3 procesar_siguiente_peticion_simulacion()

```
bool procesar_siguiente_peticion_simulacion (
            void  )
```

Procesa la siguiente petición de la simulación no-bloqueante.

Esta función debe llamarse desde el main loop para ejecutar las peticiones de forma incremental, permitiendo que el simulador de movimiento funcione en paralelo.

**Returns**

true si la simulación continúa, false si ha terminado

#### 4.9.2.4 simular_eventos_ascensor()

```
void simular_eventos_ascensor (
            void  )
```

Ejecuta una secuencia de eventos simulados de ascensor desde JSON de forma no-bloqueante.

Esta función reemplaza la simulación bloqueante anterior. Ahora la simulación se ejecuta de forma incremental, permitiendo que el main loop procese I/O y ejecute la simulación de movimiento entre peticiones.

**Cambios principales:**

- Simulación no-bloqueante (una petición por llamada)
- Control regresa al main loop para permitir movimiento de ascensores
- Intervalo configurable entre peticiones
- Estado global para rastrear progreso

**Proceso de simulación:**

1. Inicialización: selecciona edificio y configura estado
2. Ejecución incremental: una petición por llamada
3. El main loop ejecuta tanto I/O como simulación de movimiento
4. Finalización automática al completar todas las peticiones

**Note**

Esta función debe llamarse desde el main loop para ser no-bloqueante

**See also**

simulate_elevator_group_step() - se ejecuta en paralelo

#### 4.9.2.5 simular_llamada_de_piso_via_can()

```
void simular_llamada_de_piso_via_can (
            int piso_origen,
            movement_direction_enum_t direccion )
```

Simula una llamada de piso vía CAN.

**Parameters**

| | |
|---|---|
| *piso_origen* | Piso desde el cual se realiza la llamada |
| *direccion* | Dirección solicitada (MOVING_UP o MOVING_DOWN) |

Esta función genera un frame CAN simulado que representa una llamada de piso desde un botón externo. El frame se envía al puente CAN-CoAP del API Gateway para su procesamiento.

**Formato del frame CAN:**

- ID: 0x100 (identificador para llamadas de piso)
- data[0]: Piso origen (0-255)
- data[1]: Dirección (0=UP, 1=DOWN)
- DLC: 2 bytes

**Validaciones:**

- Verifica disponibilidad del contexto CoAP
- Valida parámetros de entrada

**See also**

ag_can_bridge_process_incoming_frame()

movement_direction_enum_t

#### 4.9.2.6 simular_solicitud_cabina_via_can()

```
void simular_solicitud_cabina_via_can (
            int indice_ascensor,
            int piso_destino )
```

Simula una solicitud de cabina vía CAN.

**Parameters**

| | |
|---|---|
| *indice_ascensor* | Índice del ascensor que realiza la solicitud (0-based) |
| *piso_destino* | Piso destino solicitado |

Esta función genera un frame CAN simulado que representa una solicitud de cabina desde el interior de un ascensor. El frame se envía al puente CAN-CoAP del API Gateway para su procesamiento.

**Formato del frame CAN:**

- ID: 0x200 (identificador para solicitudes de cabina)
- data[0]: Índice del ascensor (0-based, ej: 0 para E1A1)
- data[1]: Piso destino (0-255)
- DLC: 2 bytes

**Validaciones:**

- Verifica disponibilidad del contexto CoAP
- Valida parámetros de entrada

**Note**

El índice del ascensor se mapea a IDs como E1A1, E1A2, etc.

**See also**

ag_can_bridge_process_incoming_frame()
simulated_can_frame_t

### 4.9.3 Variable Documentation

#### 4.9.3.1 g_coap_context

```
coap_context_t* g_coap_context  [extern]
```

Contexto CoAP global del API Gateway.

Referencia externa al contexto CoAP principal definido en main.c del API Gateway. Utilizado para procesar eventos CoAP y enviar solicitudes al servidor central.

**See also**

api_gateway/main.c

Contexto CoAP global del API Gateway.

Contexto principal utilizado para todas las operaciones CoAP, incluyendo la simulación de ascensores y la gestión de sesiones DTLS.

#### 4.9.3.2 managed_elevator_group

```
elevator_group_state_t managed_elevator_group  [extern]
```

Grupo de ascensores gestionado.

Referencia externa al grupo de ascensores principal definido en main.c. Se utiliza para configurar el ID del edificio durante la simulación.

**See also**

elevator_group_state_t

Grupo de ascensores gestionado.

Estado del grupo de ascensores gestionado.

Estado del grupo de ascensores gestionado por este gateway.

Contiene el estado completo de todos los ascensores del edificio gestionado por este API Gateway, incluyendo posiciones actuales, tareas asignadas, estados de puertas y disponibilidad.

**See also**

elevator_group_state_t

elevator_state_manager.h

Grupo de ascensores gestionado.

Estado del grupo de ascensores gestionado.

Declaración externa para acceder al estado del grupo de ascensores gestionado en main.c. Contiene información completa de todos los ascensores del edificio.

**See also**

elevator_group_state_t

main.c

Estado del grupo de ascensores gestionado por este gateway.

Contiene el estado completo de todos los ascensores del edificio gestionado por este API Gateway, incluyendo posiciones actuales, tareas asignadas, estados de puertas y disponibilidad.

**See also**

elevator_group_state_t

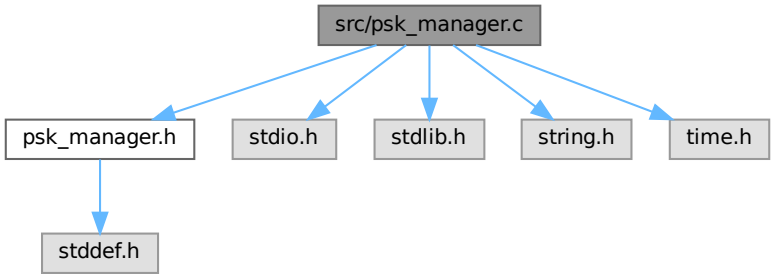
elevator_state_manager.h

## 4.10 src/psk_manager.c File Reference

Implementación del gestor de claves PSK.

```
#include "psk_manager.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
```

Include dependency graph for psk_manager.c:




### Classes

- struct psk_keys_t

  *Estructura para almacenar las claves PSK.*

### Functions

- int psk_manager_init (const char ∗keys_file_path)

  *Inicializa el gestor de claves PSK.*
- int psk_manager_get_random_key (char ∗key_buffer, size_t buffer_size)

  *Obtiene una clave PSK aleatoria del archivo.*
- int psk_manager_get_first_key (char ∗key_buffer, size_t buffer_size)

  *Obtiene la primera clave PSK del archivo como fallback.*
- int psk_manager_get_deterministic_key (const char ∗identity, char ∗key_buffer, size_t buffer_size)

  *Obtiene una clave PSK determinística basada en la identidad.*
- void **psk_manager_cleanup** (void)

  *Libera los recursos del gestor de claves PSK.*

### 4.10.1 Detailed Description

Implementación del gestor de claves PSK.

**Author**

Sistema de Control de Ascensores

**Date**

2025

**Version**

1.0

### 4.10.2 Function Documentation

#### 4.10.2.1 psk_manager_get_deterministic_key()

```
int psk_manager_get_deterministic_key (
            const char * identity,
            char * key_buffer,
            size_t buffer_size )
```

Obtiene una clave PSK determinística basada en la identidad.

**Parameters**

| | |
|---|---|
| *identity* | Identidad del cliente |
| *key_buffer* | Buffer donde se almacenará la clave |
| *buffer_size* | Tamaño del buffer |

**Returns**

0 si se obtuvo correctamente, -1 en caso de error

#### 4.10.2.2 psk_manager_get_first_key()

```
int psk_manager_get_first_key (
            char * key_buffer,
            size_t buffer_size )
```

Obtiene la primera clave PSK del archivo como fallback.

**Parameters**

| | |
|---|---|
| *key_buffer* | Buffer donde se almacenará la clave |
| *buffer_size* | Tamaño del buffer |

**Returns**

0 si se obtuvo correctamente, -1 en caso de error

#### 4.10.2.3 psk_manager_get_random_key()

```
int psk_manager_get_random_key (
```

```
            char * key_buffer,
            size_t buffer_size )
```

Obtiene una clave PSK aleatoria del archivo.

**Parameters**

| | |
|---|---|
| *key_buffer* | Buffer donde se almacenará la clave |
| *buffer_size* | Tamaño del buffer |

**Returns**

0 si se obtuvo correctamente, -1 en caso de error

#### 4.10.2.4 psk_manager_init()

```
int psk_manager_init (
            const char * keys_file_path )
```

Inicializa el gestor de claves PSK.

**Parameters**

| | |
|---|---|
| *keys_file_path* | Ruta al archivo de claves PSK |

**Returns**

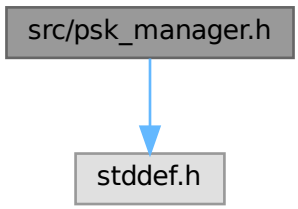
0 si se inicializó correctamente, -1 en caso de error

## 4.11 src/psk_manager.h File Reference

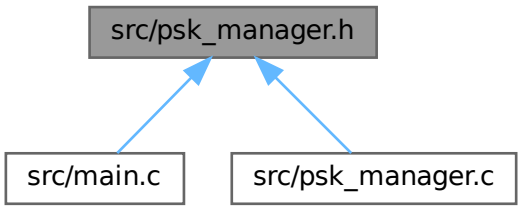
Gestor de claves PSK para API Gateway.

```
#include <stddef.h>
```

Include dependency graph for psk_manager.h:



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




**Functions**

- int psk_manager_init (const char ∗keys_file_path)
  *Inicializa el gestor de claves PSK.*
- int psk_manager_get_random_key (char ∗key_buffer, size_t buffer_size)
  *Obtiene una clave PSK aleatoria del archivo.*
- int psk_manager_get_first_key (char ∗key_buffer, size_t buffer_size)
  *Obtiene la primera clave PSK del archivo como fallback.*
- int psk_manager_get_deterministic_key (const char ∗identity, char ∗key_buffer, size_t buffer_size)
  *Obtiene una clave PSK determinística basada en la identidad.*
- void **psk_manager_cleanup** (void)
  *Libera los recursos del gestor de claves PSK.*

### 4.11.1 Detailed Description

Gestor de claves PSK para API Gateway.

**Author**

Sistema de Control de Ascensores

**Date**

2025

**Version**

1.0

Este archivo define las funciones para gestionar claves PSK desde un archivo de claves predefinidas.

### 4.11.2 Function Documentation

#### 4.11.2.1 psk_manager_get_deterministic_key()

```
int psk_manager_get_deterministic_key (
            const char * identity,
            char * key_buffer,
            size_t buffer_size )
```

Obtiene una clave PSK determinística basada en la identidad.

**Parameters**

| | |
|---|---|
| *identity* | Identidad del cliente |
| *key_buffer* | Buffer donde se almacenará la clave |
| *buffer_size* | Tamaño del buffer |

**Returns**

0 si se obtuvo correctamente, -1 en caso de error

#### 4.11.2.2 psk_manager_get_first_key()

```
int psk_manager_get_first_key (
            char * key_buffer,
            size_t buffer_size )
```

Obtiene la primera clave PSK del archivo como fallback.

**Parameters**

| | |
|---|---|
| *key_buffer* | Buffer donde se almacenará la clave |
| *buffer_size* | Tamaño del buffer |

**Returns**

0 si se obtuvo correctamente, -1 en caso de error

#### 4.11.2.3 psk_manager_get_random_key()

```
int psk_manager_get_random_key (
            char * key_buffer,
            size_t buffer_size )
```

Obtiene una clave PSK aleatoria del archivo.

**Parameters**

| | |
|---|---|
| *key_buffer* | Buffer donde se almacenará la clave |
| *buffer_size* | Tamaño del buffer |

**Returns**

0 si se obtuvo correctamente, -1 en caso de error

#### 4.11.2.4 psk_manager_init()

```
int psk_manager_init (
            const char * keys_file_path )
```

Inicializa el gestor de claves PSK.

**Parameters**

| | |
|---|---|
| *keys_file_path* | Ruta al archivo de claves PSK |

**Returns**

0 si se inicializó correctamente, -1 en caso de error

## 4.12 psk_manager.h

Go to the documentation of this file.

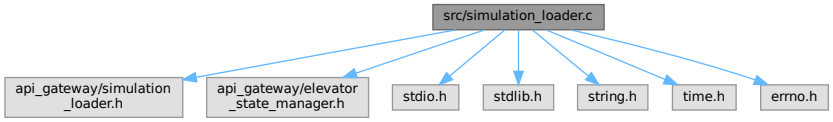
```
00001
00012 #ifndef PSK_MANAGER_H
00013 #define PSK_MANAGER_H
00014
00015 #include <stddef.h>
00016
00022 int psk_manager_init(const char* keys_file_path);
00023
00030 int psk_manager_get_random_key(char* key_buffer, size_t buffer_size);
00031
00038 int psk_manager_get_first_key(char* key_buffer, size_t buffer_size);
00039
00047 int psk_manager_get_deterministic_key(const char* identity, char* key_buffer, size_t buffer_size);
00048
00052 void psk_manager_cleanup(void);
00053
00054 #endif // PSK_MANAGER_H
```

## 4.13 src/simulation_loader.c File Reference

Implementación del Sistema de Carga y Ejecución de Simulaciones.

```
#include "api_gateway/simulation_loader.h"
#include "api_gateway/elevator_state_manager.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <errno.h>
```

Include dependency graph for simulation_loader.c:



### Functions

- bool cargar_datos_simulacion (const char *archivo_json, datos_simulacion_t *datos)
  *Carga los datos de simulación desde un archivo JSON.*
- void liberar_datos_simulacion (datos_simulacion_t *datos)
  *Libera la memoria asignada para los datos de simulación.*
- edificio_simulacion_t * seleccionar_edificio_aleatorio (datos_simulacion_t *datos)
  *Selecciona un edificio aleatorio de los datos de simulación.*
- int convertir_direccion_string (const char *direccion_str)
  *Convierte una cadena de dirección a valor numérico.*
- int ejecutar_peticiones_edificio (edificio_simulacion_t *edificio, coap_context_t *ctx)
  *Ejecuta las peticiones de un edificio específico.*

### Variables

- elevator_group_state_t managed_elevator_group
  *Estado global del grupo de ascensores gestionado por este gateway.*

## 4.13.1 Detailed Description

Implementación del Sistema de Carga y Ejecución de Simulaciones.

**Author**

Sistema de Control de Ascensores

**Date**

2025

**Version**

1.0

## 4.13.2 Function Documentation

### 4.13.2.1 cargar_datos_simulacion()

```
bool cargar_datos_simulacion (
            const char * archivo_json,
            datos_simulacion_t * datos )
```

Carga los datos de simulación desde un archivo JSON.

**Parameters**

| | |
|---|---|
| *archivo_json* | Ruta al archivo JSON con la configuración de simulación |
| *datos* | Puntero a la estructura donde se almacenarán los datos cargados |

**Returns**

true si se cargaron correctamente, false en caso de error

Esta función lee y parsea un archivo JSON que contiene la configuración de simulación de ascensores, incluyendo edificios y peticiones.

**Estructura JSON esperada:**

```
{
  "edificios": [
    {
      "id_edificio": "E1",
      "peticiones": [
        {
          "tipo": "llamada_piso",
          "piso_origen": 3,
          "direccion": "up"
        },
        {
          "tipo": "solicitud_cabina",
          "indice_ascensor": 0,
          "piso_destino": 5
        }
      ]
    }
  ]
}
```

**Operaciones realizadas:**

- Apertura y lectura del archivo JSON
- Parsing y validación de la estructura JSON
- Asignación de memoria para edificios y peticiones
- Validación de tipos de peticiones y parámetros
- Inicialización de estructuras de datos

En caso de error, libera automáticamente toda la memoria asignada.

**See also**

liberar_datos_simulacion()

datos_simulacion_t

edificio_simulacion_t

#### 4.13.2.2 convertir_direccion_string()

```
int convertir_direccion_string (
            const char * direccion_str )
```

Convierte una cadena de dirección a valor numérico.

**Parameters**

| | |
|---|---|
| *direccion_str* | Cadena que representa la dirección ("up", "down", etc.) |

**Returns**

Valor numérico correspondiente a la dirección

Esta función convierte cadenas de texto que representan direcciones de movimiento a valores numéricos utilizados en frames CAN.

**Conversiones soportadas:**

- "up" → 0 (MOVING_UP)
- "down" → 1 (MOVING_DOWN)
- Otros valores → 0 (por defecto)

La función es insensible a mayúsculas y minúsculas.

**See also**

movement_direction_enum_t

simulated_can_frame_t

#### 4.13.2.3 ejecutar_peticiones_edificio()

```
int ejecutar_peticiones_edificio (
            edificio_simulacion_t * edificio,
            coap_context_t * ctx )
```

Ejecuta las peticiones de un edificio específico.

**Parameters**

| | |
|---|---|
| *edificio* | Puntero al edificio cuyas peticiones se van a ejecutar |
| *ctx* | Contexto CoAP para el procesamiento de frames CAN |

**Returns**

Número de peticiones procesadas exitosamente

Esta función ejecuta secuencialmente todas las peticiones de un edificio, convirtiéndolas a frames CAN simulados y procesándolas a través del puente CAN-CoAP.

**Tipos de peticiones soportadas:**

- **Llamada de piso**: Genera frame CAN 0x100 con piso origen y dirección
- **Solicitud de cabina**: Genera frame CAN 0x200 con ascensor y destino

**Operaciones realizadas:**

- Actualiza el grupo de ascensores con el ID del edificio

- Procesa cada petición según su tipo
- Crea frames CAN simulados apropiados
- Envía frames al puente CAN-CoAP para procesamiento
- Registra estadísticas de peticiones procesadas

**See also**

peticion_simulacion_t

simulated_can_frame_t

ag_can_bridge_process_incoming_frame()

#### 4.13.2.4 liberar_datos_simulacion()

```
void liberar_datos_simulacion (
            datos_simulacion_t * datos )
```

Libera la memoria asignada para los datos de simulación.

**Parameters**

| | |
|---|---|
| *datos* | Puntero a la estructura de datos de simulación a liberar |

Esta función libera toda la memoria asignada dinámicamente para los datos de simulación, incluyendo arrays de edificios y peticiones.

**Operaciones realizadas:**

- Libera memoria de peticiones para cada edificio
- Libera memoria del array de edificios
- Resetea los contadores y punteros
- Inicializa la estructura a cero

La función es segura para llamar múltiples veces o con punteros NULL.

**See also**

cargar_datos_simulacion()

datos_simulacion_t

#### 4.13.2.5 seleccionar_edificio_aleatorio()

```
edificio_simulacion_t * seleccionar_edificio_aleatorio (
            datos_simulacion_t * datos )
```

Selecciona un edificio aleatorio de los datos de simulación.

**Parameters**

| | |
|---|---|
| *datos* | Puntero a los datos de simulación cargados |

**Returns**

Puntero al edificio seleccionado, o NULL si no hay edificios

Esta función selecciona aleatoriamente un edificio de los disponibles en los datos de simulación cargados. Utiliza la función rand() para la selección aleatoria.

**Comportamiento:**

- Verifica que existan edificios disponibles
- Genera un índice aleatorio válido
- Retorna el puntero al edificio seleccionado
- Registra la selección en el sistema de logging

**Note**

Se debe llamar a srand() antes de usar esta función para garantizar aleatoriedad real.

**See also**

datos_simulacion_t

edificio_simulacion_t

### 4.13.3 Variable Documentation

#### 4.13.3.1 managed_elevator_group

```
elevator_group_state_t managed_elevator_group  [extern]
```

Estado global del grupo de ascensores gestionado por este gateway.

Grupo de ascensores gestionado.

Estado del grupo de ascensores gestionado.

Estado del grupo de ascensores gestionado por este gateway.

Contiene el estado completo de todos los ascensores del edificio gestionado por este API Gateway, incluyendo posiciones actuales, tareas asignadas, estados de puertas y disponibilidad.

**See also**

elevator_group_state_t

elevator_state_manager.h

Estado global del grupo de ascensores gestionado por este gateway.

Grupo de ascensores gestionado.

Estado del grupo de ascensores gestionado.

Declaración externa para acceder al estado del grupo de ascensores gestionado en main.c. Contiene información completa de todos los ascensores del edificio.

**See also**

elevator_group_state_t

main.c

Estado del grupo de ascensores gestionado por este gateway.

Contiene el estado completo de todos los ascensores del edificio gestionado por este API Gateway, incluyendo posiciones actuales, tareas asignadas, estados de puertas y disponibilidad.

**See also**

elevator_group_state_t

elevator_state_manager.h

# Index