API Gateway - Sistema de Control de Ascensores

2.0

Generated by Doxygen 1.9.8

Chapter 1

Topic Documentation

1.1 Configuración de Escucha del Gateway

Configuraciones para el servidor CoAP del API Gateway.

Configuraciones para el servidor CoAP del API Gateway.

1.2 Configuración del Servidor Central

Configuraciones para conectar con el servidor central.

Configuraciones para conectar con el servidor central.

1.3 Recursos CoAP

Definiciones de rutas de recursos CoAP.

Definiciones de rutas de recursos CoAP.

1.4 Configuraciones de Timeout

Configuraciones de tiempo para solicitudes CoAP.

Configuraciones de tiempo para solicitudes CoAP.

1.5 Códigos de Color ANSI para Gateway

Definiciones de códigos de escape ANSI para colorear la salida del gateway.

Definiciones de códigos de escape ANSI para colorear la salida del gateway.

1.6 Macros de Logging del Gateway

Macros para generar mensajes de log con formato específico del gateway.

Macros para generar mensajes de log con formato específico del gateway.

1.7 Estructuras de Datos de Simulación

Estructuras para almacenar datos de simulación.

Data Structures

- struct **peticion_simulacion_t**
  Estructura para una petición individual.
- struct **edificio_simulacion_t**
  Estructura para un edificio de simulación.
- struct **datos_simulacion_t**
  Estructura principal de datos de simulación.

Enumerations

- enum **tipo_peticion_t** { **PETICION_LLAMADA_PISO** , **PETICION_SOLICITUD_CABINA** }
  Tipos de peticiones de simulación.

1.7.1 Detailed Description

Estructuras para almacenar datos de simulación.

1.7.2 Enumeration Type Documentation

1.7.2.1 tipo_peticion_t

`enum` **`tipo_peticion_t`**

Tipos de peticiones de simulación.

Enumeración que define los tipos de peticiones que pueden ser ejecutadas durante la simulación de ascensores.

Enumerator

PETICION_LLAMADA_PISO	Llamada de piso desde botón externo
PETICION_SOLICITUD_CABINA	Solicitud desde interior de cabina

1.8 Funciones de Simulación

Funciones para cargar y ejecutar simulaciones.

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 utilizada por los datos de simulación.
- **edificio_simulacion_t** ∗ **seleccionar_edificio_aleatorio** (**datos_simulacion_t** ∗datos)
  Selecciona un edificio aleatorio de los datos cargados.
- int **ejecutar_peticiones_edificio** (**edificio_simulacion_t** ∗edificio, coap_context_t ∗ctx)
  Ejecuta todas las peticiones de un edificio.
- int **convertir_direccion_string** (const char ∗direccion_str)
  Convierte una cadena de dirección a enum.

1.8.1 Detailed Description

Funciones para cargar y ejecutar simulaciones.

1.8.2 Function Documentation

1.8.2.1 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 los datos de simulación
datos	Puntero a la estructura donde almacenar los datos cargados

Returns

true si la carga fue exitosa, false en caso de error

Esta función lee un archivo JSON que contiene edificios y sus peticiones asociadas, parseando la información y almacenándola en estructuras C para su posterior ejecución.

Formato JSON esperado:

```
{
  "edificios": [
    {
      "id_edificio": "E001",
      "peticiones": [
```

```
        {"tipo": "llamada_piso", "piso_origen": 0, "direccion": "up"},
        {"tipo": "solicitud_cabina", "indice_ascensor": 0, "piso_destino": 5}
      ]
    }
  ]
}
```

See also

liberar_datos_simulacion() (p. **??**)

cJSON_Parse()

1.8.2.2 convertir_direccion_string()

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

Convierte una cadena de dirección a enum.

Parameters

direccion_str	Cadena con la dirección ("up" o "down")

Returns

Valor del enum correspondiente

Función auxiliar para convertir las direcciones en formato texto del JSON a los valores enum utilizados por el simulador.

See also

movement_direction_enum_t (p. **??**)

1.8.2.3 ejecutar_peticiones_edificio()

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

Ejecuta todas las peticiones de un edificio.

Parameters

edificio	Puntero al edificio cuyas peticiones ejecutar
ctx	Contexto CoAP para el procesamiento de peticiones

Returns

Número de peticiones ejecutadas exitosamente

Esta función ejecuta secuencialmente todas las peticiones de un edificio específico, con pausas entre peticiones para permitir el procesamiento de respuestas CoAP.

Proceso de ejecución:

1. Configura el ID del edificio en el sistema
2. Ejecuta cada petición según su tipo
3. Procesa I/O CoAP entre peticiones
4. Proporciona logging detallado del progreso

See also

simular_llamada_de_piso_via_can() (p. **??**)

simular_solicitud_cabina_via_can() (p. **??**)

coap_io_process()

1.8.2.4 liberar_datos_simulacion()

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

Libera la memoria utilizada por los datos de simulación.

Parameters

datos	Puntero a la estructura de datos a liberar

Esta función libera toda la memoria dinámica asignada durante la carga de datos de simulación, incluyendo arrays de edificios y peticiones.

See also

cargar_datos_simulacion() (p. **??**)

1.8.2.5 seleccionar_edificio_aleatorio()

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

Selecciona un edificio aleatorio de los datos cargados.

Parameters

datos	Puntero a los datos de simulación cargados

Returns

Puntero al edificio seleccionado, NULL si no hay datos

Esta función utiliza un generador de números aleatorios para seleccionar un edificio de la lista cargada. La selección es uniforme entre todos los edificios disponibles.

Note

Inicializa el generador aleatorio con el tiempo actual

See also

srand()

rand()

Chapter 2

Data Structure Documentation

2.1 api_request_details_for_json_t Struct Reference

Detalles específicos de solicitud para serialización JSON.

`#include <elevator_state_manager.h>`

Data Fields

- int **origin_floor_fc**
  Piso origen de la llamada.
- **movement_direction_enum_t direction_fc**
  Dirección solicitada.
- char **requesting_elevator_id_cr** [ID_STRING_MAX_LEN]
  ID del ascensor solicitante.
- int **target_floor_cr**
  Piso destino solicitado.

2.1.1 Detailed Description

Detalles específicos de solicitud para serialización JSON.

Estructura que contiene información específica del tipo de solicitud para incluir en el payload JSON enviado al servidor central.

2.1.2 Field Documentation

2.1.2.1 direction_fc

movement_direction_enum_t api_request_details_for_json_t::direction_fc

Dirección solicitada.

2.1.2.2 origin_floor_fc

```
int api_request_details_for_json_t::origin_floor_fc
```

Piso origen de la llamada.

2.1.2.3 requesting_elevator_id_cr

```
char api_request_details_for_json_t::requesting_elevator_id_cr[ID_STRING_MAX_LEN]
```

ID del ascensor solicitante.

2.1.2.4 target_floor_cr

```
int api_request_details_for_json_t::target_floor_cr
```

Piso destino solicitado.

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

- include/api_gateway/ **elevator_state_manager.h**

2.2 api_request_tracker_t Struct Reference

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

```
#include <api_handlers.h>
```

Data Fields

- 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**

2.2.1 Detailed Description

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

Structure to track client requests forwarded to the central server.

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.

This structure holds information about an original client request that the API Gateway has forwarded to the central server. It is used to correlate the server's response back to the original client session and to manage state or context related to the request.

2.2.2 Field Documentation

2.2.2.1 log_tag

```
char * api_request_tracker_t::log_tag
```

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

Tag for logging purposes (dynamically allocated).

2.2.2.2 origin_floor

```
int api_request_tracker_t::origin_floor
```

Piso origen (para GW_REQUEST_TYPE_FLOOR_CALL).

For floor calls: the floor where the call originated.

2.2.2.3 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).

Session of the original elevator client.

2.2.2.4 original_mid

```
coap_mid_t api_request_tracker_t::original_mid
```

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

CoAP Message ID of the original request.

2.2.2.5 original_token

```
coap_binary_t api_request_tracker_t::original_token
```

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

CoAP token of the original request (dynamically allocated).

2.2.2.6 request_type

```
gw_request_type_t api_request_tracker_t::request_type
```

Tipo de solicitud gestionada por el gateway.

Type of the original request.

2.2.2.7 requested_direction_floor

```
movement_direction_enum_t api_request_tracker_t::requested_direction_floor
```

Dirección solicitada (para GW_REQUEST_TYPE_FLOOR_CALL).

For floor calls: UP/DOWN.

2.2.2.8 requesting_elevator_id_cabin

```
char api_request_tracker_t::requesting_elevator_id_cabin
```

ID del ascensor (para GW_REQUEST_TYPE_CABIN_REQUEST).

For cabin requests: ID of the elevator.

2.2.2.9 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 floor the assigned elevator should go to.

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

- src/ **api_handlers.h**
- include/api_gateway/ **api_handlers.h**

2.3 can_origin_tracker_t Struct Reference

Tracker para correlacionar solicitudes CAN con respuestas CoAP.

```
#include <can_bridge.h>
```

Data Fields

- coap_binary_t **coap_token**

  Token de la solicitud CoAP enviada al servidor central.
- uint32_t **original_can_id**

  ID del frame CAN original que originó la solicitud.
- **gw_request_type_t request_type**

  Tipo de solicitud original (floor call, cabin request)
- int **target_floor_for_task**

  Piso destino de la tarea asignada.
- int **call_reference_floor**

  Piso origen de la llamada (para floor calls)
- char **requesting_elevator_id_if_cabin** [ID_STRING_MAX_LEN]

  ID del ascensor si fue cabin request.

2.3.1 Detailed Description

Tracker para correlacionar solicitudes CAN con respuestas CoAP.

Estructura utilizada para asociar un token CoAP de una solicitud enviada al servidor central con la información original del frame CAN que la originó. Permite correlacionar respuestas para enviar confirmaciones apropiadas vía CAN.

2.3.2 Field Documentation

2.3.2.1 call_reference_floor

```
int can_origin_tracker_t::call_reference_floor
```

Piso origen de la llamada (para floor calls)

2.3.2.2 coap_token

```
coap_binary_t can_origin_tracker_t::coap_token
```

Token de la solicitud CoAP enviada al servidor central.

2.3.2.3 original_can_id

```
uint32_t can_origin_tracker_t::original_can_id
```

ID del frame CAN original que originó la solicitud.

2.3.2.4 request_type

```
gw_request_type_t can_origin_tracker_t::request_type
```

Tipo de solicitud original (floor call, cabin request)

2.3.2.5 requesting_elevator_id_if_cabin

```
char can_origin_tracker_t::requesting_elevator_id_if_cabin[ID_STRING_MAX_LEN]
```

ID del ascensor si fue cabin request.

2.3.2.6 target_floor_for_task

```
int can_origin_tracker_t::target_floor_for_task
```

Piso destino de la tarea asignada.

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

- include/api_gateway/ **can_bridge.h**

2.4 central_request_entry_t Struct Reference

Entrada de tracker para solicitudes al servidor central.

Data Fields

- 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.)

2.4.1 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.

2.4.2 Field Documentation

2.4.2.1 token

```
coap_bin_const_t central_request_entry_t::token
```

Token de la solicitud enviada AL SERVIDOR CENTRAL.

2.4.2.2 tracker_data

```
api_request_tracker_t* central_request_entry_t::tracker_data
```

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

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

- src/ **api_handlers.c**

2.5 datos_simulacion_t Struct Reference

Estructura principal de datos de simulación.

```
#include <simulation_loader.h>
```

Data Fields

- **edificio_simulacion_t** * **edificios**
- int **num_edificios**
- bool **datos_cargados**

2.5.1 Detailed Description

Estructura principal de datos de simulación.

Contiene todos los edificios cargados desde el archivo JSON y proporciona acceso a los datos de simulación.

2.5.2 Field Documentation

2.5.2.1 datos_cargados

```
bool datos_simulacion_t::datos_cargados
```

Indica si los datos fueron cargados correctamente

2.5.2.2 edificios

```
edificio_simulacion_t* datos_simulacion_t::edificios
```

Array de edificios

2.5.2.3 num_edificios

```
int datos_simulacion_t::num_edificios
```

Número total de edificios

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

- include/api_gateway/ **simulation_loader.h**

2.6 edificio_simulacion_t Struct Reference

Estructura para un edificio de simulación.

```
#include <simulation_loader.h>
```

Data Fields

- char **id_edificio** [16]
- **peticion_simulacion_t** * **peticiones**
- int **num_peticiones**

2.6.1 Detailed Description

Estructura para un edificio de simulación.

Representa un edificio completo con su ID único y conjunto de peticiones asociadas.

2.6.2 Field Documentation

2.6.2.1 id_edificio

```
char edificio_simulacion_t::id_edificio[16]
```

ID único del edificio (ej: "E001")

2.6.2.2 num_peticiones

```
int edificio_simulacion_t::num_peticiones
```

Número de peticiones en el array

2.6.2.3 peticiones

```
peticion_simulacion_t* edificio_simulacion_t::peticiones
```

Array de peticiones

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

- include/api_gateway/ **simulation_loader.h**

2.7 elevator_group_state_t Struct Reference

Estado del grupo de ascensores gestionado por el gateway.

```
#include <elevator_state_manager.h>
```

Data Fields

- **elevator_status_t ascensores** [MAX_ELEVATORS_PER_GATEWAY]
  Array de ascensores del grupo.
- int **num_elevadores_en_grupo**
  Número de ascensores en el grupo.
- char **edificio_id_str_grupo** [ID_STRING_MAX_LEN]
  ID del edificio gestionado.

2.7.1 Detailed Description

Estado del grupo de ascensores gestionado por el gateway.

Estructura que contiene el estado completo de todos los ascensores de un edificio gestionados por un API Gateway específico.

2.7.2 Field Documentation

2.7.2.1 ascensores

```
elevator_status_t elevator_group_state_t::ascensores[MAX_ELEVATORS_PER_GATEWAY]
```

Array de ascensores del grupo.

2.7.2.2 edificio_id_str_grupo

```
char elevator_group_state_t::edificio_id_str_grupo[ID_STRING_MAX_LEN]
```

ID del edificio gestionado.

2.7.2.3 num_elevadores_en_grupo

```
int elevator_group_state_t::num_elevadores_en_grupo
```

Número de ascensores en el grupo.

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

- include/api_gateway/ **elevator_state_manager.h**

2.8 elevator_status_t Struct Reference

Estado de un ascensor individual.

```
#include <elevator_state_manager.h>
```

Data Fields

- char **ascensor_id** [ID_STRING_MAX_LEN]
  ID único del ascensor (ej: "E1A1")
- char **id_edificio_str** [ID_STRING_MAX_LEN]
  ID del edificio (ej: "E1")
- int **piso_actual**
  Piso actual del ascensor.
- **door_state_enum_t estado_puerta_enum**
  Estado actual de las puertas.
- char **tarea_actual_id** [TASK_ID_MAX_LEN]
  ID de la tarea asignada por el servidor central.
- int **destino_actual**
  Piso objetivo de la tarea actual (-1 si no hay)
- **movement_direction_enum_t direccion_movimiento_enum**
  Dirección de movimiento actual.
- bool **ocupado**
  True si el ascensor está asignado a una tarea.

2.8.1 Detailed Description

Estado de un ascensor individual.

Estructura que contiene toda la información de estado de un ascensor específico gestionado por el gateway.

2.8.2 Field Documentation

2.8.2.1 ascensor_id

```
char elevator_status_t::ascensor_id[ID_STRING_MAX_LEN]
```

ID único del ascensor (ej: "E1A1")

2.8.2.2 destino_actual

```
int elevator_status_t::destino_actual
```

Piso objetivo de la tarea actual (-1 si no hay)

2.8.2.3 direccion_movimiento_enum

```
movement_direction_enum_t elevator_status_t::direccion_movimiento_enum
```

Dirección de movimiento actual.

2.8.2.4 estado_puerta_enum

```
door_state_enum_t elevator_status_t::estado_puerta_enum
```

Estado actual de las puertas.

2.8.2.5 id_edificio_str

```
char elevator_status_t::id_edificio_str[ID_STRING_MAX_LEN]
```

ID del edificio (ej: "E1")

2.8.2.6 ocupado

```
bool elevator_status_t::ocupado
```

True si el ascensor está asignado a una tarea.

2.8.2.7 piso_actual

```
int elevator_status_t::piso_actual
```

Piso actual del ascensor.

2.8.2.8 tarea_actual_id

```
char elevator_status_t::tarea_actual_id[TASK_ID_MAX_LEN]
```

ID de la tarea asignada por el servidor central.

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

- include/api_gateway/ **elevator_state_manager.h**

2.9 execution_stats_t Struct Reference

Estadísticas de ejecución.

`#include <execution_logger.h>`

Data Fields

- int **total_can_frames_sent**
  Total de frames CAN enviados.
- int **total_can_frames_received**
  Total de frames CAN recibidos.
- int **total_coap_requests**
  Total de peticiones CoAP.
- int **total_coap_responses**
  Total de respuestas CoAP.
- int **total_tasks_assigned**
  Total de tareas asignadas.
- int **total_tasks_completed**
  Total de tareas completadas.
- int **total_elevator_movements**
  Total de movimientos de ascensores.
- int **total_errors**
  Total de errores.
- double **execution_duration_sec**
  Duración total en segundos.
- char **building_id** [32]
  ID del edificio simulado.
- int **building_requests**
  Número de peticiones del edificio.

2.9.1 Detailed Description

Estadísticas de ejecución.

2.9.2 Field Documentation

2.9.2.1 building_id

`char execution_stats_t::building_id[32]`

ID del edificio simulado.

2.9.2.2 building_requests

`int execution_stats_t::building_requests`

Número de peticiones del edificio.

2.9.2.3 execution_duration_sec

```
double execution_stats_t::execution_duration_sec
```

Duración total en segundos.

2.9.2.4 total_can_frames_received

```
int execution_stats_t::total_can_frames_received
```

Total de frames CAN recibidos.

2.9.2.5 total_can_frames_sent

```
int execution_stats_t::total_can_frames_sent
```

Total de frames CAN enviados.

2.9.2.6 total_coap_requests

```
int execution_stats_t::total_coap_requests
```

Total de peticiones CoAP.

2.9.2.7 total_coap_responses

```
int execution_stats_t::total_coap_responses
```

Total de respuestas CoAP.

2.9.2.8 total_elevator_movements

```
int execution_stats_t::total_elevator_movements
```

Total de movimientos de ascensores.

2.9.2.9 total_errors

```
int execution_stats_t::total_errors
```

Total de errores.

2.9.2.10 total_tasks_assigned

```
int execution_stats_t::total_tasks_assigned
```

Total de tareas asignadas.

2.9.2.11 total_tasks_completed

```
int execution_stats_t::total_tasks_completed
```

Total de tareas completadas.

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

- include/api_gateway/ **execution_logger.h**

2.10 log_event_t Struct Reference

Estructura de un evento de log.

```
#include <execution_logger.h>
```

Data Fields

- **log_event_type_t type**
  Tipo de evento.
- time_t **timestamp**
  Timestamp del evento.
- char **description** [MAX_EVENT_DESCRIPTION]
  Descripción del evento.
- char **details** [MAX_LOG_MESSAGE]
  Detalles adicionales.

2.10.1 Detailed Description

Estructura de un evento de log.

2.10.2 Field Documentation

2.10.2.1 description

```
char log_event_t::description[MAX_EVENT_DESCRIPTION]
```

Descripción del evento.

2.10.2.2 details

```
char log_event_t::details[MAX_LOG_MESSAGE]
```

Detalles adicionales.

2.10.2.3 timestamp

```
time_t log_event_t::timestamp
```

Timestamp del evento.

2.10.2.4 type

```
log_event_type_t log_event_t::type
```

Tipo de evento.

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

- include/api_gateway/ **execution_logger.h**

2.11 peticion_simulacion_t Struct Reference

Estructura para una petición individual.

```
#include <simulation_loader.h>
```

Data Fields

- **tipo_peticion_t tipo**
- int **piso_origen**
- char **direccion** [8]
- int **indice_ascensor**
- int **piso_destino**

2.11.1 Detailed Description

Estructura para una petición individual.

Contiene toda la información necesaria para ejecutar una petición específica durante la simulación.

2.11.2 Field Documentation

2.11.2.1 direccion

```
char peticion_simulacion_t::direccion[8]
```

Dirección: "up" o "down"

2.11.2.2 indice_ascensor

```
int peticion_simulacion_t::indice_ascensor
```

Índice del ascensor (0-based)

2.11.2.3 piso_destino

```
int peticion_simulacion_t::piso_destino
```

Piso destino solicitado

2.11.2.4 piso_origen

```
int peticion_simulacion_t::piso_origen
```

Piso desde el cual se llama

2.11.2.5 tipo

```
tipo_peticion_t peticion_simulacion_t::tipo
```

Tipo de petición

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

- include/api_gateway/ **simulation_loader.h**

2.12 psk_keys_t Struct Reference

Data Fields

- char ** **keys**
- int **count**
- int **capacity**

2.12.1 Field Documentation

2.12.1.1 capacity

```
int psk_keys_t::capacity
```

2.12.1.2 count

```
int psk_keys_t::count
```

2.12.1.3 keys

```
char** psk_keys_t::keys
```

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

- src/ **psk_manager.c**

2.13 simulated_can_frame_t Struct Reference

Estructura para un frame CAN simulado.

```
#include <can_bridge.h>
```

Data Fields

- uint32_t **id**
  CAN ID (identificador del mensaje)
- uint8_t **data** [8]
  Datos CAN (hasta 8 bytes según estándar CAN)
- uint8_t **dlc**
  Data Length Code (0-8, número de bytes válidos)

2.13.1 Detailed Description

Estructura para un frame CAN simulado.

Representa un frame CAN estándar con ID, datos y longitud. Utilizada para la comunicación entre el simulador de ascensores y el API Gateway.

2.13.2 Field Documentation

2.13.2.1 data

```
uint8_t simulated_can_frame_t::data[8]
```

Datos CAN (hasta 8 bytes según estándar CAN)

2.13.2.2 dlc

```
uint8_t simulated_can_frame_t::dlc
```

Data Length Code (0-8, número de bytes válidos)

2.13.2.3 id

```
uint32_t simulated_can_frame_t::id
```

CAN ID (identificador del mensaje)

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

- include/api_gateway/ **can_bridge.h**

Chapter 3

File Documentation

3.1 include/api_gateway/api_common_defs.h File Reference

3.2 api_common_defs.h

Go to the documentation of this file.

```
00001 #ifndef API_COMMON_DEFS_H
00002 #define API_COMMON_DEFS_H
00003
00004 #include "api_gateway/logging_gw.h"
00005
00006 // Common ID and String Lengths
00007 #define ID_STRING_MAX_LEN 25
00008 #define TASK_ID_MAX_LEN 32
00009 #define STATUS_STRING_MAX_LEN 16
00010
00011 // ANSI Color Codes y Logging Macros ahora están en logging_gw.h
00012
00013 #endif // API_COMMON_DEFS_H
```

3.3 include/api_gateway/can_bridge.h File Reference

Puente CAN-CoAP para el API Gateway.

Data Structures

- struct **simulated_can_frame_t**

  Estructura para un frame CAN simulado.
- struct **can_origin_tracker_t**

  Tracker para correlacionar solicitudes CAN con respuestas CoAP.

Typedefs

- typedef void(∗ **can_send_callback_t**) (**simulated_can_frame_t** ∗frame)

  Callback para envío de frames CAN al simulador.

Functions

- void **ag_can_bridge_init** (void)

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

  Registra la función de callback que la API Gateway usará para enviar frames CAN.
- void **ag_can_bridge_process_incoming_frame** (**simulated_can_frame_t** ∗frame, struct coap_context_t ∗coap_context)

  Procesa un frame CAN simulado entrante desde la simulación de ascensores.
- **can_origin_tracker_t** ∗ **find_can_tracker** (coap_bin_const_t token)

  Busca un tracker de origen CAN basado en un token 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.

3.3.1 Detailed Description

Puente CAN-CoAP para el API Gateway.

Author

Sistema de Control de Ascensores

Date

2025

Version

2.0

Este archivo define la interfaz del puente de comunicación entre el protocolo CAN (Controller Area Network) y CoAP (Constrained Application Protocol) para el sistema de control de ascensores.

Funcionalidades principales:

- Procesamiento de frames CAN entrantes desde simuladores
- Conversión de mensajes CAN a solicitudes CoAP
- Gestión de trackers para correlación de respuestas
- Envío de respuestas del servidor central vía CAN
- Interfaz con simuladores de ascensores

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.c (p. **??**)

elevator_state_manager.h (p. **??**)

api_handlers.h

3.3.2 Typedef Documentation

3.3.2.1 can_send_callback_t

```
typedef void(* can_send_callback_t) ( simulated_can_frame_t *frame)
```

Callback para envío de frames CAN al simulador.

Parameters

frame	El frame CAN simulado a enviar al simulador del ascensor

Tipo de función callback que debe implementar el simulador de ascensores para recibir frames CAN de respuesta del API Gateway. El simulador registra una función de este tipo usando **ag_can_bridge_register_send_callback()** (p. **??**).

Responsabilidades del callback:

- Procesar el frame CAN recibido
- Interpretar códigos de respuesta y datos
- Actualizar estado del simulador según corresponda
- Generar logging apropiado para debugging

See also

ag_can_bridge_register_send_callback() (p. **??**)
ag_can_bridge_send_response_frame() (p. **??**)

3.3.3 Function Documentation

3.3.3.1 ag_can_bridge_init()

```
void ag_can_bridge_init (
            void  )
```

Inicializa el puente CAN simulado.

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() (p. **??**)

ag_can_bridge_process_incoming_frame() (p. **??**)

Inicializa el puente CAN simulado.

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() (p. **??**)

ag_can_bridge_process_incoming_frame() (p. **??**)

3.3.3.2 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 simulado entrante desde la simulación de ascensores.

Parameters

frame	Puntero al frame CAN simulado recibido
coap_context	El contexto CoAP de la API Gateway, necesario para enviar solicitudes

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)

Procesamiento realizado:

1. Validación del formato y longitud de datos
2. Extracción de parámetros específicos del tipo de mensaje
3. Generación de solicitud CoAP al servidor central
4. Almacenamiento de tracker para correlación de 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() (p. **??**)
store_can_tracker() (p. **??**)
simulated_can_frame_t (p. **??**)

Procesa un frame CAN simulado entrante desde la simulación de ascensores.

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() (p. **??**)
store_can_tracker() (p. **??**)
simulated_can_frame_t (p. **??**)

3.3.3.3 ag_can_bridge_register_send_callback()

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

Registra la función de callback que la API Gateway usará para enviar frames CAN.

Parameters

callback	Puntero a la función de callback implementada por la simulación

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.

Características:

- Solo se puede registrar un callback a la vez
- Llamadas posteriores sobrescriben el callback anterior
- El callback debe ser thread-safe si se usa en entorno multihilo

Note

El callback debe permanecer válido durante toda la vida del programa

See also

can_send_callback_t (p. **??**)

ag_can_bridge_send_response_frame() (p. **??**)

Registra la función de callback que la API Gateway usará para enviar frames CAN.

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 (p. **??**)

ag_can_bridge_send_response_frame() (p. **??**)

3.3.3.4 ag_can_bridge_send_response_frame()

```
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.

Parameters

original_can_id	El ID del frame CAN original que originó esta respuesta
response_code	El código de respuesta CoAP recibido del servidor central
server_response_json	El objeto cJSON que contiene la respuesta del servidor central (puede ser NULL)

Esta función convierte una respuesta CoAP del servidor central en un frame CAN simulado que se envía de vuelta al simulador de ascensores. Maneja tanto respuestas exitosas como códigos de error.

Procesamiento de respuestas exitosas:

- Extrae ascensor_asignado_id y tarea_id del JSON
- Convierte ID de ascensor a índice numérico
- Construye frame CAN con información de asignación

- ID de respuesta: original_can_id + 1

Procesamiento de errores:

- Detecta códigos de error CoAP o campo "error" en JSON
- Genera frame CAN de error con ID 0xFE
- Incluye código de error específico en data[1]

Códigos de error definidos:

- 0x01: JSON faltante o nulo del servidor
- 0x02: Servidor reportó error o código CoAP de error
- 0x03: Fallo al parsear JSON de éxito

La función utiliza el callback registrado para enviar el frame al simulador de ascensores.

See also

ag_can_bridge_register_send_callback() (p. **??**)
can_send_callback_t (p. **??**)
simulated_can_frame_t (p. **??**)

3.3.3.5 find_can_tracker()

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

Busca un tracker de origen CAN basado en un token CoAP.

Parameters

token	El token CoAP de la respuesta recibida del servidor central

Returns

Puntero al **can_origin_tracker_t** (p. **??**) si se encuentra, NULL en caso contrario

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.

Características de la búsqueda:

- Comparación binaria exacta del token (longitud + contenido)
- Búsqueda lineal en buffer circular
- No remueve el tracker de la lista (a diferencia de find_and_remove_central_request_tracker)

- Thread-safe para lecturas concurrentes

Uso típico:

1. Se recibe respuesta del servidor central con token
2. Se busca el tracker correspondiente
3. Se extrae información del frame CAN original
4. Se genera respuesta CAN apropiada

See also

store_can_tracker() (p. **??**)
can_origin_tracker_t (p. **??**)
ag_can_bridge_send_response_frame() (p. **??**)

Busca un tracker de origen CAN basado en un 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() (p. **??**)
can_origin_tracker_t (p. **??**)

3.4 can_bridge.h

Go to the documentation of this file.

```
00001
00031 #ifndef CAN_BRIDGE_H
00032 #define CAN_BRIDGE_H
00033
00034 #include <stdint.h>
00035 #include <stdbool.h>
00036 #include <coap3/coap.h> // Para coap_bin_const_t y otros tipos CoAP
00037 #include <cjson/cJSON.h>  // Para cJSON tipo
00038 #include "api_gateway/elevator_state_manager.h" // Para gw_request_type_t
00039 #include "api_gateway/api_common_defs.h"    // Para ID_STRING_MAX_LEN
00040
00048 typedef struct {
00049     uint32_t id;
```

```
00050     uint8_t data[8];
00051     uint8_t dlc;
00052 } simulated_can_frame_t;
00053
00062 typedef struct {
00063     coap_binary_t coap_token;
00064     uint32_t original_can_id;
00065     gw_request_type_t request_type;
00066     int target_floor_for_task;
00067     int call_reference_floor;
00068     char requesting_elevator_id_if_cabin[ID_STRING_MAX_LEN];
00069 } can_origin_tracker_t;
00070
00088 typedef void (*can_send_callback_t)(simulated_can_frame_t* frame);
00089
00108 void ag_can_bridge_init(void);
00109
00131 void ag_can_bridge_register_send_callback(can_send_callback_t callback);
00132
00167 void ag_can_bridge_process_incoming_frame(simulated_can_frame_t* frame, struct coap_context_t
      *coap_context);
00168
00194 can_origin_tracker_t* find_can_tracker(coap_bin_const_t token);
00195
00229 void ag_can_bridge_send_response_frame(uint32_t original_can_id, coap_pdu_code_t response_code, cJSON*
      server_response_json);
00230
00231 #endif // CAN_BRIDGE_H
```

3.5 include/api_gateway/coap_config.h File Reference

Configuraciones CoAP para el API Gateway.

3.5.1 Detailed Description

Configuraciones CoAP para el API Gateway.

Author

Sistema de Control de Ascensores

Date

2025

Version

2.0

Este archivo define las configuraciones del protocolo CoAP (Constrained Application Protocol) utilizadas por el API Gateway para comunicarse con el servidor central.

Configuraciones incluidas:

- Dirección IP y puerto del servidor central
- Dirección IP y puerto de escucha del gateway
- Rutas de recursos CoAP para diferentes tipos de solicitudes
- Configuraciones de timeout y reintentos

- Parámetros de conexión DTLS-PSK

Recursos CoAP definidos:

- `/peticion_piso`: Para solicitudes de llamada de piso
- `/peticion_cabina`: Para solicitudes desde cabina de ascensor

Las configuraciones aquí definidas deben coincidir con las rutas y configuraciones del servidor central para garantizar la comunicación.

See also

servidor_central/main.c

3.6 coap_config.h

Go to the documentation of this file.

```
00001
00027 #ifndef COAP_CONFIG_H
00028 #define COAP_CONFIG_H
00029
00044 #define GW_LISTEN_IP "0.0.0.0"
00045
00052 #define GW_LISTEN_PORT "5683"
00053
// end of gateway_listen group  00055
00071 #define CENTRAL_SERVER_IP "192.168.49.2"
00072
00081 #define CENTRAL_SERVER_PORT "5684"
00082
// end of central_server group  00084
00106 #define FLOOR_CALL_RESOURCE "peticion_piso"
00107
00123 #define CABIN_REQUEST_RESOURCE "peticion_cabina"
00124
00133 #define CENTRAL_SERVER_FLOOR_CALL_PATH "/" FLOOR_CALL_RESOURCE
00134
00143 #define CENTRAL_SERVER_CABIN_REQUEST_PATH "/" CABIN_REQUEST_RESOURCE
00144
// end of coap_resources group  00146
00161 #define COAP_REQUEST_TIMEOUT_MS 5000
00162
00171 #define COAP_MAX_RETRIES 3
00172
// end of coap_timeouts group  00174
00175 #endif // COAP_CONFIG_H
```

3.7 include/api_gateway/elevator_state_manager.h File Reference

Gestor de Estado de Ascensores para el API Gateway.

Data Structures

- struct **elevator_status_t**

  Estado de un ascensor individual.
- struct **elevator_group_state_t**

  Estado del grupo de ascensores gestionado por el gateway.
- struct **api_request_details_for_json_t**

  Detalles específicos de solicitud para serialización JSON.

Enumerations

- enum **door_state_enum_t** {
  DOOR_CLOSED , **DOOR_OPEN** , **DOOR_OPENING** , **DOOR_CLOSING** ,
  DOOR_UNKNOWN }

  Enumeración de estados de puertas de ascensor.
- enum **movement_direction_enum_t** { **MOVING_UP** , **MOVING_DOWN** , **STOPPED** , **DIRECTION_↩ UNKNOWN** }

  Enumeración de direcciones de movimiento de ascensor.
- enum **gw_request_type_t** { **GW_REQUEST_TYPE_UNKNOWN** = 0 , **GW_REQUEST_TYPE_FLOOR↩ _CALL** , **GW_REQUEST_TYPE_CABIN_REQUEST** }

  Enumeración de tipos de solicitud del gateway.

Functions

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

  Inicializa el estado de un grupo de ascensores.
- 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.
- 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.

3.7.1 Detailed Description

Gestor de Estado de Ascensores para el API Gateway.

Author

Sistema de Control de Ascensores

Date

2025

Version

2.0

Este header define las estructuras de datos y funciones para la gestión del estado de ascensores en el API Gateway. Proporciona:

- **Estructuras de estado**: Definiciones para ascensores individuales y grupos
- **Enumeraciones**: Estados de puertas, direcciones de movimiento, tipos de solicitud

- **Funciones de gestión**: Inicialización, asignación de tareas, notificaciones
- **Serialización JSON**: Conversión de estado a formato para servidor central
- **Utilidades**: Funciones helper para conversión de enums a strings

El gestor mantiene información completa de cada ascensor incluyendo:

- Identificación única y edificio
- Posición actual y destino
- Estado de puertas y dirección de movimiento
- Tarea asignada y estado de ocupación

See also

elevator_state_manager.c (p. **??**)
api_handlers.h
api_common_defs.h (p. **??**)

3.7.2 Enumeration Type Documentation

3.7.2.1 door_state_enum_t

`enum` **`door_state_enum_t`**

Enumeración de estados de puertas de ascensor.

Define los posibles estados de las puertas de un ascensor para control y monitoreo del sistema.

Enumerator

DOOR_CLOSED	Puertas completamente cerradas.
DOOR_OPEN	Puertas completamente abiertas.
DOOR_OPENING	Puertas en proceso de apertura.
DOOR_CLOSING	Puertas en proceso de cierre.
DOOR_UNKNOWN	Estado de puertas desconocido.

3.7.2.2 gw_request_type_t

`enum` **`gw_request_type_t`**

Enumeración de tipos de solicitud del gateway.

Define los tipos de solicitudes que el gateway puede procesar y reenviar al servidor central. Utilizado para serialización JSON y gestión de trackers.

Enumerator

GW_REQUEST_TYPE_UNKNOWN	Tipo de solicitud desconocido.
GW_REQUEST_TYPE_FLOOR_CALL	Llamada de piso (botón externo)
GW_REQUEST_TYPE_CABIN_REQUEST	Solicitud de cabina (botón interno)

3.7.2.3 movement_direction_enum_t

```
enum movement_direction_enum_t
```

Enumeración de direcciones de movimiento de ascensor.

Define las posibles direcciones de movimiento de un ascensor para control de tráfico y optimización de rutas.

Enumerator

MOVING_UP	Ascensor moviéndose hacia arriba.
MOVING_DOWN	Ascensor moviéndose hacia abajo.
STOPPED	Ascensor detenido.
DIRECTION_UNKNOWN	Dirección de movimiento desconocida.

3.7.3 Function Documentation

3.7.3.1 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. Establece la tarea, destino, dirección de movimiento y marca el ascensor como ocupado.

See also

elevator_status_t (p. **??**)

3.7.3.2 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

Función helper para convertir enumeraciones de estado de puerta a strings legibles para JSON y logging.

See also

door_state_enum_t (p. **??**)

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"

3.7.3.3 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.

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

See also

gw_request_type_t (p. **??**)

api_request_details_for_json_t (p. **??**)

3.7.3.4 init_elevator_group()

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

Inicializa el estado de un grupo de ascensores.

Parameters

group	Puntero al grupo de ascensores a inicializar
edificio_id_str	El ID del edificio (ej: "E1")
num_elevadores	El número de ascensores en este grupo
num_pisos	El número total de pisos en el edificio

Esta función inicializa completamente un grupo de ascensores con la configuración especificada, estableciendo IDs únicos, posiciones iniciales y estados por defecto para todos los ascensores.

See also

elevator_group_state_t (p. **??**)

elevator_status_t (p. **??**)

Inicializa el estado de un grupo de ascensores.

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 (p. **??**)
elevator_status_t (p. **??**)

3.7.3.5 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

Función helper para convertir enumeraciones de dirección de movimiento a strings legibles para JSON y logging.

See also

movement_direction_enum_t (p. **??**)

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"

3.8 elevator_state_manager.h

Go to the documentation of this file.

```
00001
00028 #ifndef ELEVATOR_STATE_MANAGER_H
00029 #define ELEVATOR_STATE_MANAGER_H
00030
00031 #include <cJSON.h>
00032 #include <stdbool.h> // Para bool
00033
00040 #define MAX_ELEVATORS_PER_GATEWAY 6
00041
00048 #define ID_STRING_MAX_LEN 25
00049
00056 #define TASK_ID_MAX_LEN 32
00057
00064 #define STATUS_STRING_MAX_LEN 16
00065
00072 typedef enum {
00073     DOOR_CLOSED,
00074     DOOR_OPEN,
00075     DOOR_OPENING,
00076     DOOR_CLOSING,
00077     DOOR_UNKNOWN
00078 } door_state_enum_t;
00079
00086 typedef enum {
00087     MOVING_UP,
00088     MOVING_DOWN,
00089     STOPPED,
00090     DIRECTION_UNKNOWN
00091 } movement_direction_enum_t;
00092
00100 typedef enum {
00101     GW_REQUEST_TYPE_UNKNOWN = 0,
00102     GW_REQUEST_TYPE_FLOOR_CALL,
00103     GW_REQUEST_TYPE_CABIN_REQUEST
00104 } gw_request_type_t;
00105
00112 typedef struct {
00113     char ascensor_id[ID_STRING_MAX_LEN];
00114     char id_edificio_str[ID_STRING_MAX_LEN];
00115     int piso_actual;
```

```
00116     door_state_enum_t estado_puerta_enum;
00117     char tarea_actual_id[TASK_ID_MAX_LEN];
00118     int destino_actual;
00119     movement_direction_enum_t direccion_movimiento_enum;
00120     bool ocupado;
00121 } elevator_status_t;
00122
00129 typedef struct {
00130     elevator_status_t ascensores[MAX_ELEVATORS_PER_GATEWAY];
00131     int num_elevadores_en_grupo;
00132     char edificio_id_str_grupo[ID_STRING_MAX_LEN];
00133 } elevator_group_state_t;
00134
00141 typedef struct {
00142     // Para Floor Call (GW_REQUEST_TYPE_FLOOR_CALL)
00143     int origin_floor_fc;
00144     movement_direction_enum_t direction_fc;
00145
00146     // Para Cabin Request (GW_REQUEST_TYPE_CABIN_REQUEST)
00147     char requesting_elevator_id_cr[ID_STRING_MAX_LEN];
00148     int target_floor_cr;
00149
00150     // Otros tipos de solicitud pueden añadir sus campos aquí si es necesario.
00151 } api_request_details_for_json_t;
00152
00153 // --- Prototipos de Funciones ---
00154
00169 void init_elevator_group(elevator_group_state_t *group, const char* edificio_id_str, int
      num_elevadores, int num_pisos);
00170
00187 cJSON* elevator_group_to_json_for_server(const elevator_group_state_t *group,
00188                                         gw_request_type_t request_type,
00189                                         const api_request_details_for_json_t* details);
00190
00205 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);
00206
00217 const char* door_state_to_string(door_state_enum_t state);
00218
00229 const char* movement_direction_to_string(movement_direction_enum_t direction);
00230
00231 #endif // ELEVATOR_STATE_MANAGER_H
```

3.9 include/api_gateway/execution_logger.h File Reference

Sistema de Logging de Ejecuciones para Resultados de TFG.

Data Structures

- struct **log_event_t**

  Estructura de un evento de log.
- struct **execution_stats_t**

  Estadísticas de ejecución.

Enumerations

- enum **log_event_type_t** {
  LOG_EVENT_SYSTEM_START , **LOG_EVENT_SYSTEM_END** , **LOG_EVENT_SIMULATION_START** ,
  LOG_EVENT_SIMULATION_END ,
  LOG_EVENT_BUILDING_SELECTED , **LOG_EVENT_CAN_SENT** , **LOG_EVENT_CAN_RECEIVED** ,
  LOG_EVENT_COAP_SENT ,
  LOG_EVENT_COAP_RECEIVED , **LOG_EVENT_TASK_ASSIGNED** , **LOG_EVENT_ELEVATOR_↩**
  MOVED , **LOG_EVENT_TASK_COMPLETED** ,
  LOG_EVENT_ERROR }

  Tipos de eventos que se pueden registrar.

Functions

- bool **exec_logger_init** (void)

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

  Finaliza el sistema de logging y cierra el archivo.
- void **exec_logger_log_event** (**log_event_type_t** type, const char ∗description, const char ∗details)

  Registra un evento en el log.
- void **exec_logger_log_simulation_start** (const char ∗building_id, int num_requests)

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

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

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

  Registra un frame CAN recibido.
- void **exec_logger_log_coap_sent** (const char ∗method, const char ∗uri, const char ∗payload)

  Registra una petición CoAP enviada.
- void **exec_logger_log_coap_received** (const char ∗code, const char ∗payload)

  Registra una respuesta CoAP recibida.
- void **exec_logger_log_task_assigned** (const char ∗task_id, const char ∗elevator_id, int target_floor)

  Registra la asignación de una tarea.
- 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 completación de una tarea.
- 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 logger está activo.

3.9.1 Detailed Description

Sistema de Logging de Ejecuciones para Resultados de TFG.

Author

Sistema de Control de Ascensores

Date

2025

Version

1.0

Este módulo implementa un sistema de logging avanzado que genera archivos Markdown con el flujo completo de comunicación del API Gateway.

Características:

- Archivos organizados por fecha (carpetas por día)
- Formato Markdown para visualización bonita
- Captura de eventos CoAP, CAN y simulación
- Estadísticas automáticas de rendimiento
- Timestamps precisos para cada evento

Estructura de archivos:
`api_gateway/logs/YYYY-MM-DD/ejecucion_HH-MM-SS.md`

3.9.2 Enumeration Type Documentation

3.9.2.1 log_event_type_t

`enum log_event_type_t`

Tipos de eventos que se pueden registrar.

Enumerator

LOG_EVENT_SYSTEM_START	Inicio del sistema.
LOG_EVENT_SYSTEM_END	Fin del sistema.
LOG_EVENT_SIMULATION_START	Inicio de simulación.
LOG_EVENT_SIMULATION_END	Fin de simulación.
LOG_EVENT_BUILDING_SELECTED	Edificio seleccionado para simulación.
LOG_EVENT_CAN_SENT	Frame CAN enviado.
LOG_EVENT_CAN_RECEIVED	Frame CAN recibido.
LOG_EVENT_COAP_SENT	Mensaje CoAP enviado.
LOG_EVENT_COAP_RECEIVED	Mensaje CoAP recibido.
LOG_EVENT_TASK_ASSIGNED	Tarea asignada a ascensor.
LOG_EVENT_ELEVATOR_MOVED	Ascensor se movió
LOG_EVENT_TASK_COMPLETED	Tarea completada.
LOG_EVENT_ERROR	Error del sistema.

3.9.3 Function Documentation

3.9.3.1 exec_logger_finish()

```
void exec_logger_finish (
            void  )
```

Finaliza el sistema de logging y cierra el archivo.

Esta función:

- Calcula estadísticas finales
- Escribe el resumen de ejecución
- Cierra el archivo de log
- Limpia recursos

3.9.3.2 exec_logger_get_stats()

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

Obtiene las estadísticas actuales de ejecución.

Returns

Puntero a las estadísticas (solo lectura)

3.9.3.3 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:

- Crea las carpetas necesarias por fecha
- Genera el nombre del archivo de log
- Inicializa las estructuras de estadísticas
- Escribe el header del archivo Markdown

3.9.3.4 exec_logger_is_active()

```
bool exec_logger_is_active (
            void  )
```

Verifica si el logger está activo.

Returns

true si está activo, false si no

3.9.3.5 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 un frame CAN recibido.

Parameters

can_id	ID del frame CAN
dlc	Longitud de datos
data	Datos del frame
description	Descripción del evento

3.9.3.6 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 un frame CAN enviado.

Parameters

can_id	ID del frame CAN
dlc	Longitud de datos
data	Datos del frame
description	Descripción del evento

3.9.3.7 exec_logger_log_coap_received()

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

Registra una respuesta CoAP recibida.

Parameters

code	Código de respuesta CoAP
payload	Payload de la respuesta (puede ser NULL)

3.9.3.8 exec_logger_log_coap_sent()

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

Registra una petición CoAP enviada.

Parameters

method	Método CoAP (GET, POST, etc.)
uri	URI del recurso
payload	Payload de la petición (puede ser NULL)

3.9.3.9 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
from_floor	Piso origen
to_floor	Piso destino
direction	Dirección del movimiento

3.9.3.10 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
error_message	Mensaje de error

3.9.3.11 exec_logger_log_event()

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

Registra un evento en el log.

Parameters

type	Tipo de evento
description	Descripción breve del evento
details	Detalles adicionales (puede ser NULL)

3.9.3.12 exec_logger_log_simulation_end()

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

Registra el fin de la simulación.

Parameters

successful_requests	Número de peticiones exitosas
total_requests	Número total de peticiones

3.9.3.13 exec_logger_log_simulation_start()

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

Registra el inicio de la simulación.

Parameters

building_id	ID del edificio seleccionado
num_requests	Número de peticiones a ejecutar

3.9.3.14 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.

Parameters

task_id	ID de la tarea
elevator_id	ID del ascensor asignado
target_floor	Piso destino

3.9.3.15 exec_logger_log_task_completed()

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

Registra la completación de una tarea.

Parameters

task_id	ID de la tarea completada
elevator_id	ID del ascensor que completó la tarea
final_floor	Piso final donde se completó

3.10 execution_logger.h

Go to the documentation of this file.

```
00001
00024 #ifndef EXECUTION_LOGGER_H
00025 #define EXECUTION_LOGGER_H
00026
00027 #include <stdio.h>
00028 #include <stdbool.h>
00029 #include <time.h>
00030
00031 #ifdef __cplusplus
00032 extern "C" {
00033 #endif
00034
00035 // ============================================================================
00036 // CONSTANTES Y CONFIGURACIÓN
00037 // ============================================================================
00038
00039 #define MAX_LOG_PATH 512
00040 #define MAX_LOG_MESSAGE 1024
00041 #define MAX_EVENT_DESCRIPTION 256
00042
00043 // ============================================================================
00044 // TIPOS DE EVENTOS
00045 // ============================================================================
00046
00050 typedef enum {
00051     LOG_EVENT_SYSTEM_START,
00052     LOG_EVENT_SYSTEM_END,
00053     LOG_EVENT_SIMULATION_START,
00054     LOG_EVENT_SIMULATION_END,
00055     LOG_EVENT_BUILDING_SELECTED,
00056     LOG_EVENT_CAN_SENT,
00057     LOG_EVENT_CAN_RECEIVED,
00058     LOG_EVENT_COAP_SENT,
00059     LOG_EVENT_COAP_RECEIVED,
00060     LOG_EVENT_TASK_ASSIGNED,
00061     LOG_EVENT_ELEVATOR_MOVED,
00062     LOG_EVENT_TASK_COMPLETED,
00063     LOG_EVENT_ERROR
00064 } log_event_type_t;
00065
00069 typedef struct {
00070     log_event_type_t type;
00071     time_t timestamp;
00072     char description[MAX_EVENT_DESCRIPTION];
00073     char details[MAX_LOG_MESSAGE];
00074 } log_event_t;
00075
00079 typedef struct {
00080     int total_can_frames_sent;
00081     int total_can_frames_received;
00082     int total_coap_requests;
```

```
00083     int total_coap_responses;
00084     int total_tasks_assigned;
00085     int total_tasks_completed;
00086     int total_elevator_movements;
00087     int total_errors;
00088     double execution_duration_sec;
00089     char building_id[32];
00090     int building_requests;
00091 } execution_stats_t;
00092
00093 // ============================================================================
00094 // FUNCIONES PÚBLICAS
00095 // ============================================================================
00096
00107 bool exec_logger_init(void);
00108
00118 void exec_logger_finish(void);
00119
00126 void exec_logger_log_event(log_event_type_t type, const char* description, const char* details);
00127
00133 void exec_logger_log_simulation_start(const char* building_id, int num_requests);
00134
00140 void exec_logger_log_simulation_end(int successful_requests, int total_requests);
00141
00149 void exec_logger_log_can_sent(unsigned int can_id, int dlc, const unsigned char* data, const char*
      description);
00150
00158 void exec_logger_log_can_received(unsigned int can_id, int dlc, const unsigned char* data, const char*
      description);
00159
00166 void exec_logger_log_coap_sent(const char* method, const char* uri, const char* payload);
00167
00173 void exec_logger_log_coap_received(const char* code, const char* payload);
00174
00181 void exec_logger_log_task_assigned(const char* task_id, const char* elevator_id, int target_floor);
00182
00190 void exec_logger_log_elevator_moved(const char* elevator_id, int from_floor, int to_floor, const char*
      direction);
00191
00198 void exec_logger_log_task_completed(const char* task_id, const char* elevator_id, int final_floor);
00199
00205 void exec_logger_log_error(const char* error_code, const char* error_message);
00206
00211 const execution_stats_t* exec_logger_get_stats(void);
00212
00217 bool exec_logger_is_active(void);
00218
00219 #ifdef __cplusplus
00220 }
00221 #endif
00222
00223 #endif // EXECUTION_LOGGER_H
```

3.11 include/api_gateway/logging_gw.h File Reference

Sistema de Logging para el API Gateway.

3.11.1 Detailed Description

Sistema de Logging para el API Gateway.

Author

Sistema de Control de Ascensores

Date

2025

Version

2.0

Este archivo define las macros de logging utilizadas por el API Gateway para generar mensajes de depuración, información, advertencias y errores con formato consistente y colores ANSI.

Características:

- Colores ANSI para diferenciación visual de niveles
- Macros variádicas compatibles con printf
- Prefijo específico del gateway [∗-GW] para identificación
- Flush automático para salida inmediata
- Separación entre stdout (info/debug) y stderr (warn/error)

Niveles de logging:

- INFO: Información general del gateway
- DEBUG: Información detallada para depuración
- WARN: Advertencias que no afectan funcionalidad crítica
- ERROR: Errores que requieren atención
- CRIT: Errores críticos que pueden preceder a una salida

See also

servidor_central/logging.h

api_common_defs.h (p. **??**)

3.12 logging_gw.h

Go to the documentation of this file.

```
00001
00029 #ifndef LOGGING_GW_H
00030 #define LOGGING_GW_H
00031
00032 #include <stdio.h> // Para printf
00033
00041 #define ANSI_COLOR_RED     "\x1b[31m"
00042
00044 #define ANSI_COLOR_GREEN   "\x1b[32m"
00045
00047 #define ANSI_COLOR_YELLOW  "\x1b[33m"
00048
00050 #define ANSI_COLOR_BLUE    "\x1b[34m"
00051
00053 #define ANSI_COLOR_MAGENTA "\x1b[35m"
00054
00056 #define ANSI_COLOR_CYAN    "\x1b[36m"
00057
00059 #define ANSI_COLOR_RESET   "\x1b[0m"
00060
// end of ansi_colors_gw group  00062
00078 #define LOG_INFO_GW(format, ...) \
00079     do { \
00080         printf(ANSI_COLOR_GREEN "[INFO-GW] " ANSI_COLOR_RESET format "\n", ##__VA_ARGS__); \
00081         fflush(stdout); \
00082     } while (0)
```

```
00083
00095 #define LOG_DEBUG_GW(format, ...) \
00096     do { \
00097         printf(ANSI_COLOR_BLUE "[DEBUG-GW] " ANSI_COLOR_RESET format "\n", ##__VA_ARGS__); \
00098         fflush(stdout); \
00099     } while (0)
00100
00110 #define LOG_WARN_GW(format, ...) \
00111     do { \
00112         fprintf(stderr, ANSI_COLOR_YELLOW "[WARN-GW] " ANSI_COLOR_RESET format "\n", ##__VA_ARGS__); \
00113         fflush(stderr); \
00114     } while (0)
00115
00125 #define LOG_ERROR_GW(format, ...) \
00126     do { \
00127         fprintf(stderr, ANSI_COLOR_RED "[ERROR-GW] " ANSI_COLOR_RESET format "\n", ##__VA_ARGS__); \
00128         fflush(stderr); \
00129     } while (0)
00130
00142 #define LOG_CRIT_GW(format, ...) \
00143     do { \
00144         fprintf(stderr, ANSI_COLOR_MAGENTA "[CRIT-GW] " ANSI_COLOR_RESET format "\n", ##__VA_ARGS__);
    \
00145         fflush(stderr); \
00146     } while (0)
00147
// end of logging_macros_gw group  00149
00150 #endif // LOGGING_GW_H
```

3.13 include/api_gateway/simulation_loader.h File Reference

Sistema de Carga y Ejecución de Simulaciones de Ascensores.

Data Structures

- struct **peticion_simulacion_t**
  Estructura para una petición individual.
- struct **edificio_simulacion_t**
  Estructura para un edificio de simulación.
- struct **datos_simulacion_t**
  Estructura principal de datos de simulación.

Enumerations

- enum **tipo_peticion_t** { **PETICION_LLAMADA_PISO** , **PETICION_SOLICITUD_CABINA** }
  Tipos de peticiones de simulación.

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 utilizada por los datos de simulación.
- **edificio_simulacion_t** * **seleccionar_edificio_aleatorio** (**datos_simulacion_t** *datos)
  Selecciona un edificio aleatorio de los datos cargados.
- int **ejecutar_peticiones_edificio** (**edificio_simulacion_t** *edificio, coap_context_t *ctx)
  Ejecuta todas las peticiones de un edificio.
- int **convertir_direccion_string** (const char *direccion_str)
  Convierte una cadena de dirección a enum.

3.13.1 Detailed Description

Sistema de Carga y Ejecución de Simulaciones de Ascensores.

Author

Sistema de Control de Ascensores

Date

2025

Version

1.0

Este archivo define las funciones para cargar y ejecutar simulaciones de ascensores desde archivos JSON. Permite probar el sistema con múltiples escenarios de edificios y peticiones variadas.

Funcionalidades principales:

- Carga de datos de simulación desde JSON
- Selección aleatoria de edificios
- Ejecución secuencial de peticiones
- Gestión de memoria para datos de simulación

Formato de datos esperado: El archivo JSON debe contener un array de edificios, cada uno con un ID único y un conjunto de peticiones específicas.

See also

mi_simulador_ascensor.h

cJSON

3.14 simulation_loader.h

Go to the documentation of this file.

```
00001
00025 #ifndef SIMULATION_LOADER_H
00026 #define SIMULATION_LOADER_H
00027
00028 #include <coap3/coap.h>
00029 #include <cjson/cJSON.h>
00030 #include <stdbool.h>
00031
00044 typedef enum {
00045     PETICION_LLAMADA_PISO,
00046     PETICION_SOLICITUD_CABINA
00047 } tipo_peticion_t;
00048
00055 typedef struct {
00056     tipo_peticion_t tipo;
00058     // Para llamadas de piso
00059     int piso_origen;
00060     char direccion[8];
00062     // Para solicitudes de cabina
```

```
00063     int indice_ascensor;
00064     int piso_destino;
00065 } peticion_simulacion_t;
00066
00073 typedef struct {
00074     char id_edificio[16];
00075     peticion_simulacion_t *peticiones;
00076     int num_peticiones;
00077 } edificio_simulacion_t;
00078
00085 typedef struct {
00086     edificio_simulacion_t *edificios;
00087     int num_edificios;
00088     bool datos_cargados;
00089 } datos_simulacion_t;
00090
// end of simulation_data group  00092
00127 bool cargar_datos_simulacion(const char *archivo_json, datos_simulacion_t *datos);
00128
00139 void liberar_datos_simulacion(datos_simulacion_t *datos);
00140
00154 edificio_simulacion_t* seleccionar_edificio_aleatorio(datos_simulacion_t *datos);
00155
00176 int ejecutar_peticiones_edificio(edificio_simulacion_t *edificio, coap_context_t *ctx);
00177
00188 int convertir_direccion_string(const char *direccion_str);
00189
// end of simulation_functions group  00191
00192 #endif // SIMULATION_LOADER_H
```

3.15 src/api_handlers.c File Reference

Implementación de manejadores CoAP para el API Gateway.

Data Structures

- struct **central_request_entry_t**

  Entrada de tracker para solicitudes al servidor central.

Functions

- void **handle_sigint_gw** (int signum)

  Manejador de señal para SIGINT (Ctrl+C)
- static void **add_central_request_tracker** (coap_bin_const_t token_to_central, **api_request_tracker_t** ∗tracker)

  Añade un tracker a la lista de solicitudes pendientes.
- static **api_request_tracker_t** ∗ **find_and_remove_central_request_tracker** (coap_bin_const_t received_token)

  Encuentra y remueve un tracker de la lista de solicitudes pendientes.
- static void **forward_request_to_central_server** (coap_session_t ∗original_client_session, coap_mid_t original_mid, coap_bin_const_t original_token_const, const char ∗central_server_path, const char ∗log_tag_param, **gw_request_type_t** request_type_param, int origin_floor_param, int target_floor_for_task_param, const char ∗requesting_elevator_id_cabin_param, **movement_direction_enum_t** requested_direction_floor_param)

  Constructs and sends a CoAP request to the central server with the gateway's elevator group state.
- 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 for legacy elevator API requests (e.g., /peticion_ascensor). Currently logs and discards such requests.

- 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 for cabin requests from elevator clients. Path: GW_CABIN_REQUEST_PATH (e.g., /solicitud_cabina_gw) Parses query params: elevator_id, target_floor.
- 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 for floor calls from elevator clients/external. Path: GW_FLOOR_CALL_PATH (e.g., /llamada_piso_gw) Parses query params: floor, dir (UP/DOWN)

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.
- static **central_request_entry_t pending_central_requests** [MAX_PENDING_REQUESTS_TO_↩ CENTRAL]

  Array de solicitudes pendientes al servidor central.
- static int **num_pending_central_requests** = 0

  Número actual de solicitudes pendientes al servidor central.

3.15.1 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-PSK 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 (p. **??**)
can_bridge.h (p. **??**)
coap_config.h (p. **??**)

3.15.2 Function Documentation

3.15.2.1 add_central_request_tracker()

```
static void add_central_request_tracker (
            coap_bin_const_t token_to_central,
             api_request_tracker_t * tracker )  [static]
```

Añade un tracker a la lista de solicitudes pendientes.

Parameters

token_to_central	Token de la solicitud enviada al servidor central
tracker	Datos del tracker original a asociar

Esta función almacena la asociación entre un token de solicitud al servidor central y el tracker original que contiene información de la solicitud del ascensor. Crea una copia del token para evitar problemas de memoria.

Si no hay espacio disponible o hay errores de memoria, libera automáticamente el tracker para evitar fugas de memoria.

See also

find_and_remove_central_request_tracker() (p. **??**)

3.15.2.2 find_and_remove_central_request_tracker()

```
static  api_request_tracker_t * find_and_remove_central_request_tracker (
            coap_bin_const_t received_token )  [static]
```

Encuentra y remueve un tracker de la lista de solicitudes pendientes.

Parameters

received_token	Token recibido en la respuesta del servidor central

Returns

Puntero al tracker encontrado, o NULL si no se encuentra

Esta función busca en la lista de solicitudes pendientes un tracker que corresponda al token recibido en una respuesta del servidor central. Si lo encuentra, lo remueve de la lista y libera la memoria del token copiado.

La función compacta el array después de remover una entrada para mantener la integridad de la estructura de datos. Es responsabilidad del llamador liberar la memoria del tracker retornado.

Note

En un entorno multihilo, esta función requeriría sincronización

See also

add_central_request_tracker() (p. **??**)

3.15.2.3 forward_request_to_central_server()

```
static void forward_request_to_central_server (
            coap_session_t * original_client_session,
            coap_mid_t original_mid,
            coap_bin_const_t original_token_const,
            const char * central_server_path,
            const char * log_tag_param,
             gw_request_type_t request_type_param,
            int origin_floor_param,
            int target_floor_for_task_param,
            const char * requesting_elevator_id_cabin_param,
             movement_direction_enum_t requested_direction_floor_param )  [static]
```

Constructs and sends a CoAP request to the central server with the gateway's elevator group state.

This function populates an **api_request_tracker_t** (p. **??**) with details of the intended operation (floor call, cabin request), serializes the current state of the managed elevator group to JSON, and sends this as a CoAP request to the specified path on the central server.

Parameters

original_client_session	The CoAP session from which the original request came (can be NULL if internally generated).
original_mid	The CoAP MID from the original request (if applicable).
original_token_const	The CoAP token from the original request (if applicable, will be duplicated).
central_server_path	The target CoAP resource path on the central server.
log_tag_param	A string tag for logging (e.g., "FloorCallGW", will be duplicated).
request_type_param	The type of request being made (e.g., GW_REQUEST_TYPE_FLOOR_CALL).
origin_floor_param	Floor of origin for a floor call.
target_floor_for_task_param	The target floor for the task to be assigned by the server.
requesting_elevator_id_cabin_param	ID of the elevator making a cabin request (can be NULL).

3.15.2.4 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** (p. **??**) y proporciona una forma limpia de cerrar la aplicación.

3.15.2.5 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 for cabin requests from elevator clients. Path: GW_CABIN_REQUEST_PATH (e.g., /solicitud_cabina_gw) Parses query params: elevator_id, target_floor.

3.15.2.6 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() (p. **??**)
find_and_remove_central_request_tracker() (p. **??**)
can_bridge.h (p. **??**)

3.15.2.7 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 for legacy elevator API requests (e.g., /peticion_ascensor). Currently logs and discards such requests.

3.15.2.8 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 for floor calls from elevator clients/external. Path: GW_FLOOR_CALL_PATH (e.g., /llamada_piso_gw) Parses query params: floor, dir (UP/DOWN)

3.15.3 Variable Documentation

3.15.3.1 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** (p. **??**). Se utiliza para enviar solicitudes al servidor central de manera eficiente reutilizando la misma conexión segura.

See also

main.c (p. **??**)

get_or_create_central_server_dtls_session() (p. **??**)

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

3.15.3.2 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** (p. **??**). Contiene información completa de todos los ascensores del edificio.

See also

elevator_group_state_t (p. **??**)

main.c (p. **??**)

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 (p. **??**)

elevator_state_manager.h (p. **??**)

3.15.3.3 num_pending_central_requests

```
int num_pending_central_requests = 0  [static]
```

Número actual de solicitudes pendientes al servidor central.

Contador que mantiene el número de entradas activas en el array pending_central_requests. Para un entorno multihilo, se necesitaría un mutex.

3.15.3.4 pending_central_requests

```
central_request_entry_t pending_central_requests[MAX_PENDING_REQUESTS_TO_CENTRAL]  [static]
```

Array de solicitudes pendientes al servidor central.

Almacena todas las solicitudes que han sido enviadas al servidor central y están esperando respuesta. Se utiliza para correlacionar respuestas con las solicitudes originales de los ascensores.

3.15.3.5 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** (p. **??**). Es establecida por el manejador de señal (handle_sigint_gw) para detener el bucle principal de eventos.

See also

handle_sigint_gw() (p. **??**)

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() (p. **??**)

3.16 include/api_gateway/api_handlers.h File Reference

Data Structures

- struct **api_request_tracker_t**

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

Typedefs

- typedef struct **api_request_tracker_t** **api_request_tracker_t**

  Structure to track client requests forwarded to the central server.

Functions

- coap_session_t * **get_or_create_central_server_dtls_session** (struct coap_context_t *ctx)

  Obtiene o crea una sesión DTLS con el servidor central.
- 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_to_central, const coap_pdu_t *sent_to_central, const coap_pdu_t *received_from_central, const coap_mid_t mid_to_central)

  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 for legacy elevator API requests (e.g., /peticion_ascensor). Currently logs and discards such requests.
- 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 for cabin requests from elevator clients. Path: GW_CABIN_REQUEST_PATH (e.g., /solicitud_cabina_gw) Parses query params: elevator_id, target_floor.
- 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 for floor calls from elevator clients/external. Path: GW_FLOOR_CALL_PATH (e.g., /llamada_piso_gw) Parses query params: floor, dir (UP/DOWN)

Variables

- volatile sig_atomic_t **quit_main_loop**

  Bandera global para controlar el bucle principal de la aplicación.

3.16.1 Typedef Documentation

3.16.1.1 api_request_tracker_t

```
typedef struct  api_request_tracker_t  api_request_tracker_t
```

Structure to track client requests forwarded to the central server.

This structure holds information about an original client request that the API Gateway has forwarded to the central server. It is used to correlate the server's response back to the original client session and to manage state or context related to the request.

3.16.2 Function Documentation

3.16.2.1 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() (p. **??**)
coap_config.h (p. **??**)

3.16.2.2 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** (p. **??**) y proporciona una forma limpia de cerrar la aplicación.

3.16.2.3 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 for cabin requests from elevator clients. Path: GW_CABIN_REQUEST_PATH (e.g., /solicitud_cabina_gw) Parses query params: elevator_id, target_floor.

3.16.2.4 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() (p. **??**)
find_and_remove_central_request_tracker() (p. **??**)
can_bridge.h (p. **??**)

3.16.2.5 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 for legacy elevator API requests (e.g., /peticion_ascensor). Currently logs and discards such requests.

3.16.2.6 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 for floor calls from elevator clients/external. Path: GW_FLOOR_CALL_PATH (e.g., /llamada_piso_gw) Parses query params: floor, dir (UP/DOWN)

3.16.3 Variable Documentation

3.16.3.1 quit_main_loop

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

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() (p. **??**)

Bandera global para controlar el bucle principal de la aplicación.

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

See also

handle_sigint_gw() (p. **??**)

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() (p. **??**)

3.17 api_handlers.h

Go to the documentation of this file.

```
00001 #ifndef API_HANDLERS_H
00002 #define API_HANDLERS_H
00003
00004 #include <coap3/coap.h>                // For coap_session_t, coap_pdu_t, etc.
00005 #include <signal.h>                    // For sig_atomic_t
00006 #include "api_common_defs.h" // Ensuring this is just the filename
00007 #include "elevator_state_manager.h" // For movement_direction_enum_t, and gw_request_type_t now moved
      here
00008
00009 // typedef enum gw_request_type_t is MOVED to elevator_state_manager.h
00010
00018 typedef struct api_request_tracker_t {
00019     coap_session_t *original_elevator_session;
00020     coap_mid_t original_mid;
00021     coap_binary_t original_token;
00022     char *log_tag;
00024     // New fields for detailed request tracking
00025     gw_request_type_t request_type;
00026     int origin_floor;
00027     int target_floor_for_task;
00028     char requesting_elevator_id_cabin[ID_STRING_MAX_LEN];
00029     movement_direction_enum_t requested_direction_floor;
00031     // Potentially other fields like timestamp, retry count, etc.
00032 } api_request_tracker_t;
00033
00034 // Forward declaration de coap_session_t y coap_context_t para la función helper
00035 // Esto evita tener que incluir coap_session.h y coap_net.h aquí si no son necesarios para otras
      declaraciones.
00036 struct coap_session_t;
00037 struct coap_context_t;
00038
00039 // Declaración de la función helper para la gestión de sesiones DTLS (definida en main.c)
00040 coap_session_t* get_or_create_central_server_dtls_session(struct coap_context_t *ctx);
00041
00042 // --- Signal Handler ---
00043 void handle_sigint_gw(int signum);
00044
00045 // --- Response Handler (for responses FROM Central Server TO Gateway) ---
00046 coap_response_t hnd_central_server_response_gw(coap_session_t *session_to_central,
00047                                                const coap_pdu_t *sent_to_central,
00048                                                const coap_pdu_t *received_from_central,
00049                                                const coap_mid_t mid_to_central);
00050
00051 // --- Resource Handlers (for requests FROM Clients TO Gateway) ---
00052
```

```
00057 void hnd_elevator_api_request_gw(coap_resource_t *resource,
00058                                 coap_session_t *elevator_session,
00059                                 const coap_pdu_t *elevator_request_pdu,
00060                                 const coap_string_t *query,
00061                                 coap_pdu_t *response_placeholder);
00062
00068 void hnd_cabin_request_from_elevator_gw(coap_resource_t *resource,
00069                                         coap_session_t *elevator_session,
00070                                         const coap_pdu_t *elevator_request_pdu,
00071                                         const coap_string_t *query,
00072                                         coap_pdu_t *response_placeholder);
00073
00079 void hnd_floor_call_from_elevator_gw(coap_resource_t *resource,
00080                                     coap_session_t *elevator_session,
00081                                     const coap_pdu_t *elevator_request_pdu,
00082                                     const coap_string_t *query,
00083                                     coap_pdu_t *response_placeholder);
00084
00085 // Note: hnd_arrival_notification_from_elevator_gw has been removed as the gateway
00086 // now simulates arrivals and sends its own notifications to the central server.
00087
00088 extern volatile sig_atomic_t quit_main_loop; // Defined in main.c
00089
00090 #endif // API_HANDLERS_H
```

3.18 src/api_handlers.h File Reference

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

Data Structures

- 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.

3.18.1 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.

3.18.2 Enumeration Type Documentation

3.18.2.1 gw_request_type_t

```
enum gw_request_type_t
```

Enumerator

GW_REQUEST_TYPE_UNKNOWN	
GW_REQUEST_TYPE_FLOOR_CALL	
GW_REQUEST_TYPE_CABIN_REQUEST	

3.18.3 Function Documentation

3.18.3.1 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** (p. **??**) y proporciona una forma limpia de cerrar la aplicación.

3.18.3.2 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() (p. **??**)
find_and_remove_central_request_tracker() (p. **??**)
can_bridge.h (p. **??**)

3.19 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
```

3.20 src/can_bridge.c File Reference

Implementación del Puente CAN-CoAP para el API Gateway.

Functions

- static void **log_coap_token** (const char ∗prefix, coap_bin_const_t token)

  Función helper para logging de tokens CoAP.
- static void **store_can_tracker** (coap_bin_const_t token, uint32_t can_id, **gw_request_type_t** req_type, int target_floor, int ref_floor, const char ∗elevator_id_if_cabin)

  Almacena un tracker CAN para correlación de respuestas.
- **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.
- static void **forward_can_originated_request_to_central_server** (coap_context_t ∗ctx, uint32_t original_↩ can_id, const char ∗central_server_path, const char ∗log_tag_param, **gw_request_type_t** request_type_↩ param, int origin_floor_param, int target_floor_for_task_param, const char ∗requesting_elevator_id_cabin↩ _param, **movement_direction_enum_t** requested_direction_floor_param)
- 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

- static **can_send_callback_t** **send_to_simulation_callback** = NULL

  Callback registrado para enviar frames CAN a la simulación.
- static **can_origin_tracker_t** **can_trackers** [MAX_CAN_ORIGIN_TRACKERS]

  Buffer circular de trackers para solicitudes CAN.
- static int **next_can_tracker_idx** = 0

  Índice del siguiente tracker CAN disponible.
- **elevator_group_state_t** **managed_elevator_group**

  Estado del grupo de ascensores gestionado.

3.20.1 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 (p. **??**)
api_handlers.h
elevator_state_manager.h (p. **??**)
coap_config.h (p. **??**)

3.20.2 Function Documentation

3.20.2.1 ag_can_bridge_init()

```
void ag_can_bridge_init (
            void  )
```

Inicializa el puente CAN-CoAP.

Inicializa el puente CAN simulado.

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() (p. **??**)
ag_can_bridge_process_incoming_frame() (p. **??**)

3.20.2.2 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.

Procesa un frame CAN simulado entrante desde la simulación de ascensores.

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() (p. **??**)
store_can_tracker() (p. **??**)
simulated_can_frame_t (p. **??**)

3.20.2.3 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.

Registra la función de callback que la API Gateway usará para enviar frames CAN.

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 (p. **??**)

ag_can_bridge_send_response_frame() (p. **??**)

3.20.2.4 ag_can_bridge_send_response_frame()

```
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.

Parameters

original_can_id	El ID del frame CAN original que originó esta respuesta
response_code	El código de respuesta CoAP recibido del servidor central
server_response_json	El objeto cJSON que contiene la respuesta del servidor central (puede ser NULL)

Esta función convierte una respuesta CoAP del servidor central en un frame CAN simulado que se envía de vuelta al simulador de ascensores. Maneja tanto respuestas exitosas como códigos de error.

Procesamiento de respuestas exitosas:

- Extrae ascensor_asignado_id y tarea_id del JSON

- Convierte ID de ascensor a índice numérico
- Construye frame CAN con información de asignación
- ID de respuesta: original_can_id + 1

Procesamiento de errores:

- Detecta códigos de error CoAP o campo "error" en JSON
- Genera frame CAN de error con ID 0xFE
- Incluye código de error específico en data[1]

Códigos de error definidos:

- 0x01: JSON faltante o nulo del servidor
- 0x02: Servidor reportó error o código CoAP de error
- 0x03: Fallo al parsear JSON de éxito

La función utiliza el callback registrado para enviar el frame al simulador de ascensores.

See also

ag_can_bridge_register_send_callback() (p. **??**)
can_send_callback_t (p. **??**)
simulated_can_frame_t (p. **??**)

3.20.2.5 find_can_tracker()

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

Busca un tracker CAN por token CoAP.

Busca un tracker de origen CAN basado en un 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() (p. **??**)

can_origin_tracker_t (p. **??**)

3.20.2.6 forward_can_originated_request_to_central_server()

```
static void forward_can_originated_request_to_central_server (
            coap_context_t * ctx,
            uint32_t original_can_id,
            const char * central_server_path,
            const char * log_tag_param,
             gw_request_type_t request_type_param,
            int origin_floor_param,
            int target_floor_for_task_param,
            const char * requesting_elevator_id_cabin_param,
             movement_direction_enum_t requested_direction_floor_param )  [static]
```

3.20.2.7 log_coap_token()

```
static void log_coap_token (
            const char * prefix,
            coap_bin_const_t token )  [static]
```

Función helper para logging de tokens CoAP.

Parameters

prefix	Prefijo para el mensaje de log
token	Token CoAP a imprimir en formato hexadecimal

Esta función convierte un token CoAP binario a su representación hexadecimal para facilitar el debugging y seguimiento de solicitudes. Limita la impresión a 8 bytes máximo para evitar logs excesivamente largos.

3.20.2.8 store_can_tracker()

```
static void store_can_tracker (
            coap_bin_const_t token,
            uint32_t can_id,
             gw_request_type_t req_type,
            int target_floor,
            int ref_floor,
            const char * elevator_id_if_cabin )  [static]
```

Almacena un tracker CAN para correlación de respuestas.

Parameters

token	Token CoAP de la solicitud enviada al servidor central
can_id	ID del frame CAN original

Parameters

req_type	Tipo de solicitud (floor call, cabin request, etc.)
target_floor	Piso destino de la tarea
ref_floor	Piso de referencia (origen para floor calls)
elevator_id_if_cabin	ID del ascensor (solo para cabin requests)

Esta función almacena información de correlación en el buffer circular de trackers CAN. Crea una copia del token CoAP para evitar problemas de memoria y libera automáticamente tokens anteriores si es necesario.

See also

find_can_tracker() (p. **??**)

can_origin_tracker_t (p. **??**)

3.20.3 Variable Documentation

3.20.3.1 can_trackers

```
can_origin_tracker_t can_trackers[MAX_CAN_ORIGIN_TRACKERS]  [static]
```

Buffer circular de trackers para solicitudes CAN.

Array que almacena información de correlación entre solicitudes CAN enviadas al servidor central y sus respuestas esperadas. Se gestiona como un buffer circular para reutilizar entradas.

See also

can_origin_tracker_t (p. **??**)

3.20.3.2 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** (p. **??**). Se utiliza para acceder a información del edificio y ascensores al procesar mensajes CAN.

See also

main.c (p. **??**)

elevator_group_state_t (p. **??**)

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 (p. **??**)

elevator_state_manager.h (p. **??**)

Estado del grupo de ascensores gestionado.

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

See also

elevator_group_state_t (p. **??**)

main.c (p. **??**)

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 (p. **??**)

elevator_state_manager.h (p. **??**)

3.20.3.3 next_can_tracker_idx

```
int next_can_tracker_idx = 0  [static]
```

Índice del siguiente tracker CAN disponible.

Índice que apunta a la siguiente posición disponible en el buffer circular can_trackers. Se incrementa módulo MAX_CAN_ORIGIN_TRACKERS.

3.20.3.4 send_to_simulation_callback

```
can_send_callback_t send_to_simulation_callback = NULL  [static]
```

Callback registrado para enviar frames CAN a la simulación.

Puntero a función que se utiliza para enviar frames CAN de respuesta al simulador. Se registra mediante **ag_↩ can_bridge_register_send_callback()** (p. **??**).

See also

ag_can_bridge_register_send_callback() (p. **??**)

can_send_callback_t (p. **??**)

3.21 src/elevator_state_manager.c File Reference

Implementación del Gestor de Estado de Ascensores del API Gateway.

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.

3.21.1 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 (p. **??**)

api_common_defs.h (p. **??**)

api_handlers.h

3.21.2 Function Documentation

3.21.2.1 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. Establece la tarea, destino, dirección de movimiento y marca el ascensor como ocupado.

See also

elevator_status_t (p. **??**)

3.21.2.2 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"

3.21.2.3 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.

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

See also

gw_request_type_t (p. **??**)
api_request_details_for_json_t (p. **??**)

3.21.2.4 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.

Inicializa el estado de un grupo de ascensores.

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 (p. **??**)
elevator_status_t (p. **??**)

3.21.2.5 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"

3.22 src/execution_logger.c File Reference

Implementación del Sistema de Logging de Ejecuciones.

Functions

- static bool **create_directory** (const char *path)
  Crea un directorio recursivamente.
- static void **get_formatted_timestamp** (char *buffer, size_t buffer_size, const char *format)
  Obtiene el timestamp actual formateado.
- static void **write_markdown_header** (void)
  Escribe el header del archivo Markdown.
- static void **write_markdown_footer** (void)
  Escribe el footer del archivo Markdown con estadísticas finales.
- static const char * **get_event_label** (**log_event_type_t** type)
  Convierte el tipo de evento a etiqueta de texto profesional.
- bool **exec_logger_init** (void)
  Inicializa el sistema de logging de ejecuciones.
- void **exec_logger_finish** (void)
  Finaliza el sistema de logging y cierra el archivo.
- void **exec_logger_log_event** (**log_event_type_t** type, const char *description, const char *details)
  Registra un evento en el log.
- void **exec_logger_log_simulation_start** (const char *building_id, int num_requests)
  Registra el inicio de la simulación.
- void **exec_logger_log_simulation_end** (int successful_requests, int total_requests)
  Registra el fin de la simulación.
- void **exec_logger_log_can_sent** (unsigned int can_id, int dlc, const unsigned char *data, const char *description)
  Registra un frame CAN enviado.
- void **exec_logger_log_can_received** (unsigned int can_id, int dlc, const unsigned char *data, const char *description)
  Registra un frame CAN recibido.
- void **exec_logger_log_coap_sent** (const char *method, const char *uri, const char *payload)
  Registra una petición CoAP enviada.
- void **exec_logger_log_coap_received** (const char *code, const char *payload)
  Registra una respuesta CoAP recibida.
- void **exec_logger_log_task_assigned** (const char *task_id, const char *elevator_id, int target_floor)
  Registra la asignación de una tarea.
- 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 completación de una tarea.
- 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 logger está activo.

Variables

- static FILE * **log_file** = NULL

  Archivo de log actual.
- static **execution_stats_t** **stats**

  Estadísticas de ejecución.
- static time_t **start_time**

  Tiempo de inicio de ejecución.
- static bool **logger_active** = false

  Estado del logger.
- static char **current_log_path** [MAX_LOG_PATH]

  Ruta del archivo actual.

3.22.1 Detailed Description

Implementación del Sistema de Logging de Ejecuciones.

Author

Sistema de Control de Ascensores

Date

2025

Version

1.0

3.22.2 Function Documentation

3.22.2.1 create_directory()

```
static bool create_directory (
            const char * path )  [static]
```

Crea un directorio recursivamente.

Parameters

path	Ruta del directorio a crear

Returns

true si se creó exitosamente, false en caso de error

3.22.2.2 exec_logger_finish()

```
void exec_logger_finish (
            void  )
```

Finaliza el sistema de logging y cierra el archivo.

Esta función:

- Calcula estadísticas finales
- Escribe el resumen de ejecución
- Cierra el archivo de log
- Limpia recursos

3.22.2.3 exec_logger_get_stats()

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

Obtiene las estadísticas actuales de ejecución.

Returns

Puntero a las estadísticas (solo lectura)

3.22.2.4 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:

- Crea las carpetas necesarias por fecha
- Genera el nombre del archivo de log
- Inicializa las estructuras de estadísticas
- Escribe el header del archivo Markdown

3.22.2.5 exec_logger_is_active()

```
bool exec_logger_is_active (
            void  )
```

Verifica si el logger está activo.

Returns

true si está activo, false si no

3.22.2.6 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 un frame CAN recibido.

Parameters

can_id	ID del frame CAN
dlc	Longitud de datos
data	Datos del frame
description	Descripción del evento

3.22.2.7 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 un frame CAN enviado.

Parameters

can_id	ID del frame CAN
dlc	Longitud de datos
data	Datos del frame
description	Descripción del evento

3.22.2.8 exec_logger_log_coap_received()

```
void exec_logger_log_coap_received (
```

```
            const char * code,
            const char * payload )
```

Registra una respuesta CoAP recibida.

Parameters

code	Código de respuesta CoAP
payload	Payload de la respuesta (puede ser NULL)

3.22.2.9 exec_logger_log_coap_sent()

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

Registra una petición CoAP enviada.

Parameters

method	Método CoAP (GET, POST, etc.)
uri	URI del recurso
payload	Payload de la petición (puede ser NULL)

3.22.2.10 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
from_floor	Piso origen
to_floor	Piso destino
direction	Dirección del movimiento

3.22.2.11 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
error_message	Mensaje de error

3.22.2.12 exec_logger_log_event()

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

Registra un evento en el log.

Parameters

type	Tipo de evento
description	Descripción breve del evento
details	Detalles adicionales (puede ser NULL)

3.22.2.13 exec_logger_log_simulation_end()

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

Registra el fin de la simulación.

Parameters

successful_requests	Número de peticiones exitosas
total_requests	Número total de peticiones

3.22.2.14 exec_logger_log_simulation_start()

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

Registra el inicio de la simulación.

Parameters

building_id	ID del edificio seleccionado
num_requests	Número de peticiones a ejecutar

3.22.2.15 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.

Parameters

task_id	ID de la tarea
elevator_id	ID del ascensor asignado
target_floor	Piso destino

3.22.2.16 exec_logger_log_task_completed()

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

Registra la completación de una tarea.

Parameters

task_id	ID de la tarea completada
elevator_id	ID del ascensor que completó la tarea
final_floor	Piso final donde se completó

3.22.2.17 get_event_label()

```
static const char * get_event_label (
             log_event_type_t type )  [static]
```

Convierte el tipo de evento a etiqueta de texto profesional.

3.22.2.18 get_formatted_timestamp()

```
static void get_formatted_timestamp (
            char * buffer,
            size_t buffer_size,
            const char * format )  [static]
```

Obtiene el timestamp actual formateado.

Parameters

buffer	Buffer donde escribir el timestamp
buffer_size	Tamaño del buffer
format	Formato del timestamp

3.22.2.19 write_markdown_footer()

```
static void write_markdown_footer (
            void  )  [static]
```

Escribe el footer del archivo Markdown con estadísticas finales.

3.22.2.20 write_markdown_header()

```
static void write_markdown_header (
            void  )  [static]
```

Escribe el header del archivo Markdown.

3.22.3 Variable Documentation

3.22.3.1 current_log_path

```
char current_log_path[MAX_LOG_PATH]  [static]
```

Ruta del archivo actual.

3.22.3.2 log_file

```
FILE* log_file = NULL  [static]
```

Archivo de log actual.

3.22.3.3 logger_active

```
bool logger_active = false  [static]
```

Estado del logger.

3.22.3.4 start_time

```
time_t start_time  [static]
```

Tiempo de inicio de ejecución.

3.22.3.5 stats

```
execution_stats_t stats  [static]
```

Estadísticas de ejecución.

3.23 src/main.c File Reference

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

Functions

- static char * **generate_unique_identity** ()
- static char * **generate_unique_psk_key** ()
- static void **simulate_elevator_group_step** (coap_context_t *ctx, **elevator_group_state_t** *group)

  Simula un paso de movimiento para todos los ascensores del grupo.
- 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.
- static int **event_handler_gw** (coap_session_t *session, coap_event_t event)

  Manejador de eventos CoAP para sesiones DTLS.
- 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.
- static coap_context_t * **g_coap_context_for_session_mgnt** = NULL

  Contexto CoAP para la gestión de sesiones DTLS.

3.23.1 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-PSK
- **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 (p. **??**)
can_bridge.h (p. **??**)
coap_config.h (p. **??**)

3.23.2 Function Documentation

3.23.2.1 event_handler_gw()

```
static int event_handler_gw (
            coap_session_t * session,
            coap_event_t event )  [static]
```

Manejador de eventos CoAP para sesiones DTLS.

Parameters

session	Sesión CoAP que generó el evento
event	Tipo de evento ocurrido

Returns

0 si el evento se procesó correctamente

Esta función maneja eventos relacionados con sesiones DTLS, especialmente cierres de conexión y errores. Limpia automáticamente la sesión global cuando se detecta que se ha cerrado o ha fallado.

Eventos manejados:

- COAP_EVENT_DTLS_CLOSED: Sesión DTLS cerrada
- COAP_EVENT_DTLS_ERROR: Error en sesión DTLS
- COAP_EVENT_SESSION_CLOSED: Sesión cerrada
- COAP_EVENT_SESSION_FAILED: Sesión falló

3.23.2.2 generate_unique_identity()

```
static char * generate_unique_identity ( )  [static]
```

3.23.2.3 generate_unique_psk_key()

```
static char * generate_unique_psk_key ( )  [static]
```

3.23.2.4 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() (p. **??**)
coap_config.h (p. **??**)

3.23.2.5 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() (p. **??**)
cargar_datos_simulacion() (p. **??**)
mi_simulador_recibe_can_gw() (p. **??**)

3.23.2.6 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.

3.23.2.7 simular_eventos_ascensor()

```
void simular_eventos_ascensor (
            void  )
```

Ejecuta una secuencia de eventos simulados de ascensor desde JSON.

Esta función reemplaza la simulación hardcodeada anterior. Ahora carga datos de simulación desde un archivo JSON, selecciona un edificio aleatorio y ejecuta todas sus peticiones secuencialmente.

Proceso de simulación:

1. Verifica si hay datos de simulación cargados
2. Selecciona un edificio aleatorio de los 100 disponibles
3. Configura el ID del edificio en el sistema
4. Ejecuta las 10 peticiones del edificio secuencialmente
5. Procesa I/O CoAP entre cada petición

Fallback: Si no se pueden cargar los datos JSON, ejecuta la simulación básica de 2 peticiones hardcodeadas como antes.

Características:

- Procesamiento de I/O CoAP entre eventos
- Logging detallado de cada paso
- Sincronización para evitar condiciones de carrera
- Configuración automática del ID de edificio

See also

cargar_datos_simulacion() (p. **??**)
seleccionar_edificio_aleatorio() (p. **??**)
simular_llamada_de_piso_via_can() (p. **??**)
simular_solicitud_cabina_via_can() (p. **??**)
coap_io_process()

3.23.2.8 simulate_elevator_group_step()

```
static void simulate_elevator_group_step (
            coap_context_t * ctx,
             elevator_group_state_t * group )  [static]
```

Simula un paso de movimiento para todos los ascensores del grupo.

Parameters

ctx	Contexto CoAP para logging y operaciones
group	Grupo de ascensores a simular

Esta función implementa la lógica de simulación de movimiento de ascensores. Se ejecuta periódicamente para actualizar el estado de todos los ascensores que tienen tareas asignadas.

Funcionalidades implementadas:

- **Cierre de puertas**: Cierra puertas antes del movimiento
- **Determinación de dirección**: Calcula dirección basada en destino
- **Simulación de movimiento**: Mueve ascensores piso por piso
- **Detección de llegada**: Detecta cuando un ascensor llega a su destino
- **Completado de tareas**: Notifica llegadas y libera ascensores

La simulación maneja los siguientes estados:

- Ascensores ocupados con destino válido
- Movimiento hacia arriba (MOVING_UP)
- Movimiento hacia abajo (MOVING_DOWN)
- Llegada a destino y liberación

See also

elevator_group_state_t (p. **??**)
elevator_status_t (p. **??**)

3.23.3 Variable Documentation

3.23.3.1 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.

3.23.3.2 g_coap_context_for_session_mgnt

```
coap_context_t* g_coap_context_for_session_mgnt = NULL  [static]
```

Contexto CoAP para la gestión de sesiones DTLS.

Contexto específico utilizado para el manejo de eventos de sesiones DTLS con el servidor central.

3.23.3.3 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-PSK con el servidor central para el envío de solicitudes de asignación de ascensores. Se reutiliza para múltiples solicitudes para eficiencia.

3.23.3.4 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 (p. **??**)

elevator_state_manager.h (p. **??**)

3.23.3.5 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() (p. **??**)

3.24 src/main_dynamic_port.c File Reference

Versión del API Gateway que acepta puerto como parámetro.

Functions

- static char * **generate_unique_identity** ()
- static char * **generate_unique_psk_key** ()
- static void **simulate_elevator_group_step** (coap_context_t *ctx, **elevator_group_state_t** *group)
- 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.
- static int **event_handler_gw** (coap_session_t *session, coap_event_t event)
- 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[])

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.
- static coap_context_t * **g_coap_context_for_session_mgnt** = NULL
- static int **dynamic_listen_port** = 5683

3.24.1 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.

3.24.2 Function Documentation

3.24.2.1 event_handler_gw()

```
static int event_handler_gw (
            coap_session_t * session,
            coap_event_t event )  [static]
```

3.24.2.2 generate_unique_identity()

```
static char * generate_unique_identity ( )  [static]
```

3.24.2.3 generate_unique_psk_key()

```
static char * generate_unique_psk_key ( )  [static]
```

3.24.2.4 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() (p. **??**)
coap_config.h (p. **??**)

3.24.2.5 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() (p. **??**)
cargar_datos_simulacion() (p. **??**)
mi_simulador_recibe_can_gw() (p. **??**)

3.24.2.6 main()

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

3.24.2.7 simular_eventos_ascensor()

```
void simular_eventos_ascensor (
            void  )
```

Ejecuta una secuencia de eventos simulados de ascensor desde JSON.

Esta función reemplaza la simulación hardcodeada anterior. Ahora carga datos de simulación desde un archivo JSON, selecciona un edificio aleatorio y ejecuta todas sus peticiones secuencialmente.

Proceso de simulación:

1. Verifica si hay datos de simulación cargados
2. Selecciona un edificio aleatorio de los 100 disponibles
3. Configura el ID del edificio en el sistema
4. Ejecuta las 10 peticiones del edificio secuencialmente
5. Procesa I/O CoAP entre cada petición

Fallback: Si no se pueden cargar los datos JSON, ejecuta la simulación básica de 2 peticiones hardcodeadas como antes.

Características:

- Procesamiento de I/O CoAP entre eventos
- Logging detallado de cada paso
- Sincronización para evitar condiciones de carrera
- Configuración automática del ID de edificio

See also

cargar_datos_simulacion() (p. **??**)
seleccionar_edificio_aleatorio() (p. **??**)
simular_llamada_de_piso_via_can() (p. **??**)
simular_solicitud_cabina_via_can() (p. **??**)
coap_io_process()

3.24.2.8 simulate_elevator_group_step()

```
static void simulate_elevator_group_step (
             coap_context_t * ctx,
              elevator_group_state_t * group )  [static]
```

3.24.3 Variable Documentation

3.24.3.1 dynamic_listen_port

```
int dynamic_listen_port = 5683  [static]
```

3.24.3.2 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** (p. **??**) 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.

3.24.3.3 g_coap_context_for_session_mgnt

```
coap_context_t* g_coap_context_for_session_mgnt = NULL  [static]
```

3.24.3.4 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** (p. **??**). Se utiliza para enviar solicitudes al servidor central de manera eficiente reutilizando la misma conexión segura.

See also

main.c (p. **??**)

get_or_create_central_server_dtls_session() (p. **??**)

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

3.24.3.5 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** (p. **??**). Contiene información completa de todos los ascensores del edificio.

See also

elevator_group_state_t (p. **??**)

main.c (p. **??**)

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 (p. **??**)

elevator_state_manager.h (p. **??**)

3.24.3.6 quit_main_loop

```
volatile sig_atomic_t quit_main_loop = 0
```

Bandera para indicar si el bucle principal debe terminar.

Bandera global para controlar el bucle principal de la aplicación.

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

See also

handle_sigint_gw() (p. **??**)

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() (p. **??**)

3.25 src/mi_simulador_ascensor.c File Reference

Simulador de Ascensores para Testing del API Gateway.

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.

Variables

- coap_context_t ∗ **g_coap_context**
  Contexto CoAP global del API Gateway.
- **elevator_group_state_t managed_elevator_group**
  Grupo de ascensores gestionado.
- static **datos_simulacion_t datos_simulacion_global**
  Datos de simulación cargados desde JSON.

3.25.1 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 (p. **??**)
elevator_state_manager.h (p. **??**)
simulation_loader.h (p. **??**)
api_common_defs.h (p. **??**)

3.25.2 Function Documentation

3.25.2.1 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() (p. **??**)
cargar_datos_simulacion() (p. **??**)
mi_simulador_recibe_can_gw() (p. **??**)

3.25.2.2 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() (p. **??**)
simulated_can_frame_t (p. **??**)

3.25.2.3 simular_eventos_ascensor()

```
void simular_eventos_ascensor (
            void  )
```

Ejecuta una secuencia de eventos simulados de ascensor desde JSON.

Esta función reemplaza la simulación hardcodeada anterior. Ahora carga datos de simulación desde un archivo JSON, selecciona un edificio aleatorio y ejecuta todas sus peticiones secuencialmente.

Proceso de simulación:

1. Verifica si hay datos de simulación cargados
2. Selecciona un edificio aleatorio de los 100 disponibles
3. Configura el ID del edificio en el sistema
4. Ejecuta las 10 peticiones del edificio secuencialmente
5. Procesa I/O CoAP entre cada petición

Fallback: Si no se pueden cargar los datos JSON, ejecuta la simulación básica de 2 peticiones hardcodeadas como antes.

Características:

- Procesamiento de I/O CoAP entre eventos
- Logging detallado de cada paso
- Sincronización para evitar condiciones de carrera
- Configuración automática del ID de edificio

See also

cargar_datos_simulacion() (p. **??**)
seleccionar_edificio_aleatorio() (p. **??**)
simular_llamada_de_piso_via_can() (p. **??**)
simular_solicitud_cabina_via_can() (p. **??**)
coap_io_process()

3.25.2.4 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() (p. **??**)
movement_direction_enum_t (p. **??**)

3.25.2.5 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() (p. **??**)
simulated_can_frame_t (p. **??**)

3.25.3 Variable Documentation

3.25.3.1 datos_simulacion_global

```
datos_simulacion_t datos_simulacion_global  [static]
```

Datos de simulación cargados desde JSON.

Variable global que almacena todos los edificios y peticiones cargados desde el archivo de simulación JSON.

3.25.3.2 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** (p. **??**) 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.

3.25.3.3 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** (p. **??**). Se utiliza para configurar el ID del edificio durante la simulación.

See also

elevator_group_state_t (p. **??**)

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 (p. **??**)
elevator_state_manager.h (p. **??**)

Grupo de ascensores gestionado.

Estado del grupo de ascensores gestionado.

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

See also

elevator_group_state_t (p. **??**)
main.c (p. **??**)

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 (p. **??**)
elevator_state_manager.h (p. **??**)

3.26 src/psk_manager.c File Reference

Implementación del gestor de claves PSK.

Data Structures

- struct **psk_keys_t**

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.

Variables

- static **psk_keys_t g_psk_keys** = {NULL, 0, 0}

3.26.1 Detailed Description

Implementación del gestor de claves PSK.

Author

Sistema de Control de Ascensores

Date

2025

Version

1.0

3.26.2 Function Documentation

3.26.2.1 psk_manager_cleanup()

```
void psk_manager_cleanup (
            void  )
```

Libera los recursos del gestor de claves PSK.

3.26.2.2 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

3.26.2.3 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

3.26.2.4 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

3.26.2.5 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

3.26.3 Variable Documentation

3.26.3.1 g_psk_keys

```
psk_keys_t g_psk_keys = {NULL, 0, 0}  [static]
```

3.27 src/psk_manager.h File Reference

Gestor de claves PSK para API Gateway.

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.

3.27.1 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.

3.27.2 Function Documentation

3.27.2.1 psk_manager_cleanup()

```
void psk_manager_cleanup (
            void  )
```

Libera los recursos del gestor de claves PSK.

3.27.2.2 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

3.27.2.3 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

3.27.2.4 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

3.27.2.5 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

3.28 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
```

3.29 src/simulation_loader.c File Reference

Implementación del Sistema de Carga y Ejecución de Simulaciones.

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 utilizada por los datos de simulación.
- **edificio_simulacion_t** * **seleccionar_edificio_aleatorio** (**datos_simulacion_t** *datos)
  Selecciona un edificio aleatorio de los datos cargados.
- int **convertir_direccion_string** (const char *direccion_str)
  Convierte una cadena de dirección a enum.
- int **ejecutar_peticiones_edificio** (**edificio_simulacion_t** *edificio, coap_context_t *ctx)
  Ejecuta todas las peticiones de un edificio.

Variables

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

3.29.1 Detailed Description

Implementación del Sistema de Carga y Ejecución de Simulaciones.

Author

Sistema de Control de Ascensores

Date

2025

Version

1.0

3.29.2 Variable Documentation

3.29.2.1 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 (p. **??**)

elevator_state_manager.h (p. **??**)

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** (p. **??**). Contiene información completa de todos los ascensores del edificio.

See also

elevator_group_state_t (p. **??**)

main.c (p. **??**)

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 (p. **??**)

elevator_state_manager.h (p. **??**)