CloudGate SDK reference manual

Publication date: February 11, 2019
Revision: V035ext

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Module Documentation

4.1 Configuration functions

Data Structures

- struct cg_conf_list
- struct cg_config_t
  Structure to reference the configuration context.
- struct cg_conf_list_t
  Structure to return a list of values from the configuration.

Typedefs

- typedef struct cg_config **cg_config_t**
- typedef struct cg_conf_list **cg_conf_list_t**

Functions

- cg_status_t cg_conf_open (const char file, int state, cg_config_t ctx)
- cg_status_t cg_conf_close (cg_config_t ctx)
- cg_status_t cg_conf_revert (cg_config_t ctx, const char section, const char option)
- cg_status_t cg_conf_set (cg_config_t ctx, const char section, const char option, const char value)
- cg_status_t cg_conf_get (cg_config_t ctx, const char section, const char option, const char value)
- cg_status_t cg_conf_add_section (cg_config_t ctx, const char name)
- cg_status_t cg_conf_del (cg_config_t ctx, const char section, const char option)
- cg_status_t cg_conf_get_sections (cg_config_t ctx, cg_conf_list_t sections)
- cg_status_t cg_conf_get_options (cg_config_t ctx, const char section, cg_conf_list_t options)

4.1.1 Detailed Description

4.1.2 Function Documentation

4.1.2.1 cg_status_t cg_conf_open (const char *file*, int *state*, cg_config_t *ctx*)

Open a configuration context for a specific configuration file

Parameters

in	*file*	The name of the configuration file to open.
in	*state*	If TRUE any changes made to this configuration file will only kept as a temporary state. These settings will only be visible if this file is opened with state set to TRUE. Upon reboot these settings are lost. Setting state to FALSE will persist all changes. Persistent configurations are available for reading when opening in a temporary state.
out	*ctx*	The context pointer for this configuration file. This pointer should be used with every cg_conf call that operates on this file.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.1.2.2 cg_status_t cg_conf_close (cg_config_t *ctx*)

Close the provided configuration context. All changes will be saved persistently unless cg_conf_open was called with state set to TRUE. The ctx pointer will be freed upon return.

Parameters

in	*ctx*	The configuration context previously opened with cg_conf_open.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.1.2.3 cg_status_t cg_conf_revert (cg_config_t *ctx*, const char *section*, const char *option*)

Revert all changes made to the configuration context since cg_conf_open. The other parameters can optionally be used to only revert a specific section or option.

Parameters

in	*ctx*	The configuration context previously opened with cg_conf_open.
in	*section*	The name of the section that needs to be reverted. Can be left NULL.
in	*option*	The name of the option that needs to be reverted. Can be left NULL. If not NULL, section MUST also be non-NULL.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.1.2.4 cg_status_t cg_conf_set (cg_config_t *ctx*, const char *section*, const char *option*, const char *value*)

Set a section and an option value in the configuration.

Parameters

in	*ctx*	The configuration context previously opened with cg_conf_open.
in	*section*	The name of the section where we are setting a value. If this section does not exist, it will be created.
in	*option*	The name of the option for which we are setting a value.
in	*value*	The value which we want to set.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.1.2.5 cg_status_t cg_conf_get (cg_config_t *ctx*, const char *section*, const char *option*, const char *value*)

Get a section type or an option value from the configuration.

Parameters

in	*ctx*	The configuration context previously opened with cg_conf_open.
in	*section*	The name of the section where we are getting a value from.
in	*option*	The name of the option from which we are getting a value.
out	*value*	The value that is read from the configuration is stored here. This value is only usable until cg_conf_close is called. If a longer scope is necessary the user is responsible for making a copy of the value.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.1.2.6 cg_status_t cg_conf_add_section (cg_config_t *ctx*, const char *name*)

Add a section to the configuration. If a section with the same name already exists, this will return CG_STATUS_ERROR.

Parameters

in	*ctx*	The configuration context previously opened with cg_conf_open.
in	*name*	The name for the new section.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.1.2.7 cg_status_t cg_conf_del (cg_config_t *ctx*, const char *section*, const char *option*)

Delete a section or option from the configuration.

Parameters

in	*ctx*	The configuration context previously opened with cg_conf_open.
in	*section*	The name of the section you wish to delete.
in	*option*	The name of the option you wish to delete. Can be left NULL. When left NULL cg_conf_del deletes the entire section. If not NULL, section MUST also be non-NULL.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.1.2.8 cg_status_t cg_conf_get_sections (cg_config_t *ctx*, cg_conf_list_t *sections*)

Get all sections from the configuration file.

Parameters

in	*ctx*	The configuration context previously opened with cg_conf_open.
out	*sections*	The names of the sections in this configuration file. sections has to be freed by the user.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.1.2.9 cg_status_t cg_conf_get_options (cg_config_t *ctx*, const char *section*, cg_conf_list_t *options*)

List all options of the specified section from the configuration file.

Parameters

in	*ctx*	The configuration context previously opened with cg_conf_open.
in	*section*	The section name for which you wish to retrieve the options.
out	*options*	The names of the options in this section. options has to be freed by the user.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.2 Device Management

Data Structures

- struct cg_device_t

  Basic properties of a hardware device present in the system.
- struct cg_board_t

  Description of a pluggable expansion board.

Typedefs

- typedef void(device_notification_cb_t)(cg_device_t device, cg_device_event_t evt, void context)
- typedef cg_status_t(device_claim_cb_t)(cg_device_t device, void context)

Enumerations

- enum cg_device_status_t { CG_DEVICE_PRESENT = 0x1, CG_DEVICE_CLAIMED = 0x2 }
- enum cg_device_type_t {
  CG_DEVTYPE_NETWORK = 0x1, CG_DEVTYPE_SERIAL = 0x2, CG_DEVTYPE_DAC = 0x4, CG_DEVTYPE_GPIO = 0x8,
  CG_DEVTYPE_ADC = 0x10 }
- enum cg_location_t {
  CG_LOC_UNKNOWN = 0x0, CG_LOC_SLOT1 = 0x1, CG_LOC_SLOT2 = 0x2, CG_LOC_MAIN = 0x4,
  CG_LOC_ALL = 0x7 }
- enum cg_device_event_t { CG_DEV_EVT_ARRIVED, CG_DEV_EVT_REMOVED, CG_DEV_EVT_CLAIMED, CG_DEV_EVT_RELEASED }

Functions

- cg_status_t cg_device_list (uint32_t device_mask, uint32_t num_devices, cg_device_t devices)
- cg_status_t cg_device_list_by_location (uint32_t location_mask, uint32_t device_mask, uint32_t num_devices, cg_device_t devices)
- cg_status_t cg_device_claim (cg_device_t device, device_claim_cb_t cb, void context)
- cg_status_t cg_device_release (cg_device_t device)
- cg_status_t cg_device_register_notification (uint32_t device_mask, device_notification_cb_t cb, void context)
- cg_status_t cg_device_deregister_notification (void)
- cg_status_t cg_device_board_list (uint8_t num_boards, cg_board_t boards)
- cg_status_t cg_device_board_device_list (cg_board_t board, uint32_t num_devices, cg_device_t devices)

4.2.1 Detailed Description

Note

TODO:

- Registering/deregistering of board definitions ?

4.2.2 Typedef Documentation

4.2.2.1 typedef void(device_notification_cb_t)(cg_device_t device, cg_device_event_t evt, void context)

Device notification callback

Parameters

in	*device*	Device for which notification is received.
in	*evt*	Received event
in	*context*	The context parameter which was passed on to cg_device_-register_notification

4.2.2.2 typedef cg_status_t(device_claim_cb_t)(cg_device_t device, void context)

Device claim callback

Parameters

in	*device*	The device for which a claim request is received.
in	*context*	The context parameter that was passed on to cg_device_claim

Returns

CG_STATUS_OK to release the device. CG_STATUS_ERROR to hold on to the device.

4.2.3 Enumeration Type Documentation

4.2.3.1 enum cg_device_status_t

Various device status flags

Enumerator

CG_DEVICE_PRESENT Device is physically present in the system

CG_DEVICE_CLAIMED Device has been claimed by a process

4.2.3.2 enum cg_device_type_t

List of possible device types

Enumerator

CG_DEVTYPE_NETWORK Network interface (WLAN/LAN/3G/...)

CG_DEVTYPE_SERIAL Serial port

CG_DEVTYPE_DAC Digital-Analog Converter

CG_DEVTYPE_GPIO GPIO pin

CG_DEVTYPE_ADC Analog-Digital Converter

4.2.3.3 enum cg_location_t

List of possible locations

Enumerator

CG_LOC_UNKNOWN The location is unknown

CG_LOC_SLOT1 The first slot, located below the front side with the leds

CG_LOC_SLOT2 The second slot, located below the back side with the power supply

CG_LOC_MAIN Located on the mainboard

CG_LOC_ALL All locations

4.2.3.4 enum cg_device_event_t

List of all possible device events

Enumerator

CG_DEV_EVT_ARRIVED Event to indicate that a device has arrived

CG_DEV_EVT_REMOVED Event to indicate that a device has been removed

CG_DEV_EVT_CLAIMED Event to indicate that a device has been claimed by a process

CG_DEV_EVT_RELEASED Event to indicate that a device has been released by a process and can now be claimed again

4.2.4 Function Documentation

4.2.4.1 cg_status_t cg_device_list (uint32_t *device_mask*, uint32_t *num_devices*, cg_device_t *devices*)

Returns a list of present accessible hardware devices

Parameters

in	*device_mask*	A mask of cg_device_type_t values to indicate the types of devices to list
out	*num_devices*	Number of device structures returned
out	*devices*	List of device structures allocated by callee and assigned to devices

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of invalid parameter (e.g. device_mask cannot be '0') or CG_STATUS_ERROR in case of an error.

4.2.4.2 cg_status_t cg_device_list_by_location (uint32_t *location_mask*, uint32_t *device_mask*, uint32_t *num_devices*, cg_device_t *devices*)

Returns a list of present accessible hardware devices at a given location

Parameters

in	*location_mask*	A mask of cg_location_t values to indicate the locations of the devices to list
in	*device_mask*	A mask of cg_device_type_t values to indicate the types of devices to list
out	*num_devices*	Number of device structures returned
out	*devices*	List of device structures allocated by callee and assigned to devices

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of invalid parameter (e.g. device_mask cannot be '0') or CG_STATUS_ERROR in case of an error.

4.2.4.3 cg_status_t cg_device_claim (cg_device_t *device*, device_claim_cb_t *cb*, void *context*)

Marks the device as used by the calling process. It is possible that the device is currently being used by another entity. In that case that entity is asked to release the device. If the entity agrees, the device is assigned to the caller. If the entity does not agree, an error is returned and you cannot claim the device. On the other hand: if another entity wants to use the device, the callback cb is called. If cb returns CG_STATUS_OK, it indicates that you allow to release the device so the other entity can claim it. When returning CG_STATUS_ERROR, you indicate that you wish to keep hold of the device.

Parameters

in	*device*	The device to claim
in	*cb*	Callback when another device wants to use the device you are using right now. return CG_STATUS_OK if you agree to release the device. Return CG_STATUS_ERROR if you want to keep on using the device. If you do not provide a callback, all other claims will be responded with CG_STATUS_ERROR.
in	*context*	Context parameter passed along to cb when called.

Returns

CG_STATUS_OK in case of success. CG_STATUS_RESOURCE_BUSY when the device is kept in use. CG_STATUS_INVALID_PARAMETER for invalid parameters (e.g. device = NULL), CG_STATUS_ERROR otherwise.

4.2.4.4 cg_status_t cg_device_release (cg_device_t *device*)

Releases the device so it is free to be used by another process

Parameters

in	*device*	The device to release

Returns

CG_STATUS_OK in normal case. GG_STATUS_INVALID_PARAMETER for invalid parameter (e.g. device == NULL), CG_STATUS_ERROR otherwise.

4.2.4.5 cg_status_t cg_device_register_notification (uint32_t *device_mask*, device_notification_cb_t *cb*, void *context*)

Registers a callback for notifications about device types. Calling this function multiple times will cause the previous registration to be overwritten.

Parameters

in	*device_mask*	A mask of cg_device_type_t types.
in	*cb*	This callback will be called when a notification is received.
in	*context*	A parameter which will be passed on to the callback when it is called.

Returns

CG_STATUS_OK if successful or CG_STATUS_ERROR in case of an error.

4.2.4.6 cg_status_t cg_device_deregister_notification (void)

Deregisters a callback for notifications about a specific device or device type.

Returns

CG_STATUS_OK if successful of CG_STATUS_ERROR in case of error.

4.2.4.7 cg_status_t cg_device_board_list (uint8_t *num_boards*, cg_board_t *boards*)

Requests a list of expansion boards currently plugged into the system

Parameters

out	*num_boards*	Gets filled in by callee with the number of boards currently inserted.
out	*boards*	List of boards in the system. Callee will allocate array of boards and will assign it to boards. Caller needs to free the the array.

Returns

CG_STATUS_OK if successful of CG_STATUS_ERROR in case of error.

4.2.4.8 cg_status_t cg_device_board_device_list (cg_board_t *board*, uint32_t *num_devices*, cg_device_t *devices*)

Requests a list of devices available on an expansion board currently plugged into the system

Parameters

in	*board*	details of board the caller want the device list from
out	*num_devices*	Gets filled in by callee with the number of devices on the board.
out	*devices*	List of devices on a boards in the system. Callee will allocate array of devices and will assign it to devices. Caller needs to free the the array.

Returns

CG_STATUS_OK if successful of CG_STATUS_ERROR in case of error.

4.3 DUST_SENSOR

Data Structures

- struct cg_dust_sensor_data_t
  Output parameters of the dust sensor logger.

Typedefs

- typedef void(cg_dust_sensor_data_cb_t)(const cg_dust_sensor_data_t data, void context)

Enumerations

- enum cg_dust_sensor_power_t { CG_DUST_SENSOR_POWER_OFF = 0, CG_DUST_SENSOR_POWER_FAN_ON = 1, CG_DUST_SENSOR_POWER_LASER_ON = 2, CG_DUST_SENSOR_POWER_MAX = 3 }
- enum cg_dust_sensor_status_t { CG_DUST_SENSOR_VALID_DATA, CG_DUST_SENSOR_RW_ERROR, CG_DUST_SENSOR_DISCONNECT, CG_DUST_SENSOR_CONNECT, CG_DUST_SENSOR_NOT_AVAILABLE, CG_DUST_SENSOR_THREAD_CREATE_ERROR, CG_DUST_SENSOR_INVALID_PARAMETER, CG_DUST_SENSOR_NOT_AWAKE, CG_DUST_SENSOR_SPI_ERROR }

Functions

- cg_status_t cg_dust_sensor_start (uint32_t time_interval)
- cg_status_t cg_dust_sensor_stop (void)
- cg_status_t cg_dust_sensor_set_power (uint32_t power)
- cg_status_t cg_dust_sensor_register_data_callback (cg_dust_sensor_data_cb_t cb, void context)
- cg_status_t cg_dust_sensor_deregister_data_callback (cg_dust_sensor_data_cb_t cb, void context)

4.3.1 Detailed Description

4.3.2 Typedef Documentation

4.3.2.1 typedef void(cg_dust_sensor_data_cb_t)(const cg_dust_sensor_data_t data, void context)

Callback for reporting dust sensor data

Parameters

in	*dust*	sensor data
in	*context*	Context as set in cg_dust_sensor_register_data_callback

4.3.3 Enumeration Type Documentation

4.3.3.1 enum cg_dust_sensor_power_t

Enumerator

CG_DUST_SENSOR_POWER_OFF When nor fan nor laser is powered
CG_DUST_SENSOR_POWER_FAN_ON Bit to set fan power
CG_DUST_SENSOR_POWER_LASER_ON Bit to set laser power
CG_DUST_SENSOR_POWER_MAX Max value dust sensor's power

4.3.3.2 enum cg_dust_sensor_status_t

- The status of dust sensor's monitoring

Enumerator

CG_DUST_SENSOR_VALID_DATA When valid data is available
CG_DUST_SENSOR_RW_ERROR When a read/write error occured
CG_DUST_SENSOR_DISCONNECT when the dust sensor disconnects
CG_DUST_SENSOR_CONNECT When the dust sensor connects
CG_DUST_SENSOR_NOT_AVAILABLE Dust sensor not available
CG_DUST_SENSOR_THREAD_CREATE_ERROR Can't create thread
CG_DUST_SENSOR_INVALID_PARAMETER Invalid parameter
CG_DUST_SENSOR_NOT_AWAKE Dust sensor is not ready
CG_DUST_SENSOR_SPI_ERROR Error setting spi mode

4.3.4 Function Documentation

4.3.4.1 cg_status_t cg_dust_sensor_start (uint32_t *time_interval*)

Initiate dust sensor's logging. The function sets the supplied parameters and starts dust sensor logging with a given time interval.

Parameters

in	*time_interval*	value in seconds.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter, CG_STATUS_RESOURCE_BUSY in case no dust sensor found or CG_STATUS_ERROR in case of an error.

4.3.4.2 cg_status_t cg_dust_sensor_stop (void)

Stop dust sensor monitoring

Returns

CG_STATUS_OK if successful or CG_STATUS_ERROR in case of an error.

4.3.4.3 cg_status_t cg_dust_sensor_set_power (uint32_t *power*)

Sets dust sensor's power The function set the fan power and/or the laser power.

Parameters

in	*power*	0 power off, 1 fan power on, 2 laser power on, 3 laser and fan power on

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter, CG_STATUS_RESOURCE_BUSY in case no dust sensor found or CG_STATUS_ERROR in case of an error.

4.3.4.4 cg_status_t cg_dust_sensor_register_data_callback (cg_dust_sensor_data_cb_t *cb*, void *context*)

Register a callback for dust_sensor data. It's possible to register multiple callbacks as long as either the callback or the context is different.

Parameters

in	*cb*	Callback for data.
in	*context*	Context data passed to cb when called.

Returns

CG_STATUS_OK if successful. CG_STATUS_INVALID_PARAMETER if no callback has been provided or CG_STATUS_ERROR if failure.

4.3.4.5 cg_status_t cg_dust_sensor_deregister_data_callback (cg_dust_sensor_data_cb_t *cb*, void *context*)

Deregister a previously registered callback

Parameters

in	*cb*	Callback to be called when logger data is received
in	*context*	Context data which was passed on to the register call

Returns

CG_STATUS_OK if successful. CG_STATUS_INVALID_PARAMETER if no matching callback and context has been found. CG_STATUS_ERROR (unknown error)

4.4 General functions

Data Structures

- struct cg_slot_t

  Device firmware slot description.
- struct cg_slot_list_t

  List of present slots.
- struct cg_kvpair_t

  Key/value pair.
- struct cg_key_t

  A single key of a key/value pair.
- struct cg_timezone_t

  Time zone splitted up in its components. https://www.gnu.org/software/libc/manual/html_node/-TZ-Variable.html.
- struct cg_date_time_t

  Structure holding local or UTC time value.

Macros

- #define CG_SDK_API_LEVEL 01
- #define CG_MAX_SLOT_INFO_LEN 32
- #define **CG_MAX_SLOT_UID_LEN** 64
- #define **CG_KVPAIR_MAX_NS_LEN** 64
- #define **CG_KVPAIR_MAX_KEY_LEN** 64
- #define **CG_KVPAIR_MAX_VALUE_LEN** 4096

Typedefs

- typedef void(cg_prov_kvpair_update_cb_t)(cg_kvpair_t update, void context)
- typedef void(cg_system_sync_ntp_cb_t)(cg_system_sync_ntp_state_t state, void context)

Enumerations

- enum cg_prov_status_t { CG_PROV_UNPROVISIONED, CG_PROV_CHECKING, CG_PROV_FLASHING, CG_PROV_ERRO-R, CG_PROV_OK, CG_PROV_UNREGISTERED }

  Provisioning status.
- enum cg_slot_type_t { CG_SLOT_BOOTLOADER, CG_SLOT_BASE_IMAGE, CG_SLOT_BASE_CONFIG, CG_SLOT_CUST-OMER, CG_SLOT_CUSTOMER_CONFIG, CG_SLOT_CUSTOMER_KEY }

  Different firmware slots.
- enum cg_dev_feature_t { CG_DEV_ROOT_ACCESS, CG_DEV_MINIVAN_DEBUG }

  Various development features.
- enum **cg_log_level_t** { **CG_LOG_DEBUG** = 0x1, **CG_LOG_INFO** = 0x2, **CG_LOG_WARN** = 0x4, **CG_LOG_ERR** = 0x8 }
- enum cg_system_boot_condition_t { CG_SYSTEM_ARM_IGNITION_SENSE = 0x1, CG_SYSTEM_-TIMED_WAKEUP = 0x2 }

  System boot condition.

- enum **cg_time_format_t** { **CG_LOCAL_TIME**, **CG_UTC_TIME** }
- enum **cg_system_sync_ntp_state_t** { **CG_NTP_SYNC_SUCCESS**, **CG_NTP_SYNC_FAILURE**, **CG_NTP_SYNC_NO_SERVER** }

Functions

- cg_status_t cg_init (const char name)
- cg_status_t cg_deinit (void)
- cg_status_t cg_get_api_level (int32_t level)
- cg_status_t cg_get_serial_number (char serial_number)
- cg_status_t cg_prov_get_status (cg_prov_status_t status)
- cg_status_t cg_prov_check_update (uint8_t force)
- cg_status_t cg_prov_set_automatic (uint32_t onoff)
- cg_status_t cg_prov_kvpair_get (cg_kvpair_t request_in, cg_kvpair_t request_out)
- cg_status_t cg_prov_kvpair_set (cg_kvpair_t request)
- cg_status_t cg_prov_kvpair_del (cg_kvpair_t request)
- cg_status_t cg_prov_kvpair_list (char ns, int n_keys, cg_key_t keys)
- cg_status_t cg_prov_kvpair_register_update_callback (cg_prov_kvpair_update_cb_t cb, void context, const char ns)
- cg_status_t cg_prov_kvpair_deregister_update_callback (cg_prov_kvpair_update_cb_t cb, void context, const char ns)
- cg_status_t cg_get_slot_list (cg_slot_list_t slot)
- cg_status_t cg_reset_system (uint8_t fact_restore, const char reason)
- cg_status_t cg_reset_peripherals (void)
- cg_status_t cg_system_log (uint16_t log_level, const char format,...) CG_GNUC_PRINTF(2
- cg_status_t cg_status_t cg_set_dev_mode (uint8_t enable, cg_dev_feature_t feature, void options)
- cg_status_t cg_system_power_down (const cg_system_boot_condition_t system_boot_condition)
- cg_status_t cg_system_set_timed_wakeup (uint32_t seconds)
- cg_status_t cg_system_set_date_time (cg_date_time_t date_time)
- cg_status_t cg_system_get_date_time (cg_time_format_t time_format, cg_date_time_t date_time)
- cg_status_t cg_system_set_timezone (cg_timezone_t timezone)
- cg_status_t cg_system_set_ntp_server (char ntp_server_url)
- cg_status_t cg_system_get_ntp_server (char ntp_server_url)
- cg_status_t cg_system_sync_ntp (void)
- cg_status_t cg_system_sync_ntp_register_callback (cg_system_sync_ntp_cb_t cb, void context)
- cg_status_t cg_system_sync_ntp_deregister_callback (cg_system_sync_ntp_cb_t cb, void context)

4.4.1 Detailed Description

4.4.2 Macro Definition Documentation

4.4.2.1 #define CG_SDK_API_LEVEL 01

Current API level

4.4.2.2 #define CG_MAX_SLOT_INFO_LEN 32

Maximum length of a slot info string

4.4.3 Typedef Documentation

4.4.3.1 typedef void(cg_prov_kvpair_update_cb_t)(cg_kvpair_t update, void context)

Callback for key/value pair updates

Parameters

in	*context*	Context as set in cg_prov_kvpair_register_update_callback
in	*update*	Pointer to a cg_kvpair_t containing the new/updated or deleted key/value pair. If deleted, value_len is 0. This cg_kvpair_t must be free()d by the user-supplied callback.

4.4.3.2 typedef void(cg_system_sync_ntp_cb_t)(cg_system_sync_ntp_state_t state, void context)

Callback for status code ntp sync

Parameters

in	*state*	The updated state.
in	*context*	The context which was passed on in cg_system_sync_ntp call

4.4.4 Enumeration Type Documentation

4.4.4.1 enum cg_prov_status_t

Provisioning status.

Enumerator

CG_PROV_UNPROVISIONED Device is currently not provisioned
CG_PROV_CHECKING Device is checking for provisioning data
CG_PROV_FLASHING Device is flashing new software
CG_PROV_ERROR Last provisioning attempt returned an error
CG_PROV_OK Device is provisioned
CG_PROV_UNREGISTERED Device is unregistered/not activated

4.4.4.2 enum cg_slot_type_t

Different firmware slots.

o

Enumerator

CG_SLOT_BOOTLOADER Bootloader type
CG_SLOT_BASE_IMAGE Base image type (kernel + root filesystem)
CG_SLOT_BASE_CONFIG Base configuration
CG_SLOT_CUSTOMER Customer/developer firmware
CG_SLOT_CUSTOMER_CONFIG Customer/developer configuration
CG_SLOT_CUSTOMER_KEY Customer/developer public key

4.4.4.3 enum cg_dev_feature_t

Various development features.

Enumerator

CG_DEV_ROOT_ACCESS Root access

CG_DEV_MINIVAN_DEBUG Set the minivan debug level

4.4.4.4 enum cg_system_boot_condition_t

System boot condition.

Enumerator

CG_SYSTEM_ARM_IGNITION_SENSE if this maskbit is set the system will boot when the ignition sense is detected

CG_SYSTEM_TIMED_WAKEUP if this maskbit is set the system will boot when the "timed_-wakeup" time has elapsed

4.4.5 Function Documentation

4.4.5.1 cg_status_t cg_init (const char *name*)

Initialize the SDK.

Parameters

name	The name of the program/process that wants to use the SDK

Returns

CG_STATUS_OK if successful or CG_STATUS_ERROR upon error.

4.4.5.2 cg_status_t cg_deinit (void)

Uninitialize the SDK.

4.4.5.3 cg_status_t cg_get_api_level (int32_t *level*)

Get the API level that the device supports. A user will need to compare this with the SDK API level and make a decision to go ahead or not.

Parameters

out	*level*	Integer indicating the API version

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER when provided with an invalid parameter or CG_STATUS_ERROR upon error.

4.4.5.4 cg_status_t cg_get_serial_number (char *serial_number*)

Get the serial number of the device.

Parameters

out	*serial_number*	When succesful, will be set to the serial number of the device, allocated by the callee. The caller will be responsible for freeing the allocated memory.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER when provided with an invalid parameter or CG_STATUS_ERROR upon error.

4.4.5.5 cg_status_t cg_prov_get_status (cg_prov_status_t *status*)

Get the current provisioning status

Parameters

out	*status*	Provisioning status

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER when provided with an invalid parameter or CG_STATUS_ERROR upon error.

4.4.5.6 cg_status_t cg_prov_check_update (uint8_t *force*)

Trigger a check for new provisioning. The device will contact the provisioning server.

Parameters

in	*force*	Setting this flag to TRUE will force the device to re-provisioning regardless if it already has the correct firmware.

Returns

CG_STATUS_OK if successful or CG_STATUS_ERROR upon error.
CG_STATUS_INVALID_PARAMETER if supplied parameter is invalid.

4.4.5.7 cg_status_t cg_prov_set_automatic (uint32_t *onoff*)

Enables or disables automatic provisioning by the provisioning server.

Parameters

in	*onoff*	Setting this flag to TRUE will enable automatic provisioning

Returns

CG_STATUS_OK if successful or CG_STATUS_ERROR upon error.

4.4.5.8 cg_status_t cg_prov_kvpair_get (cg_kvpair_t *request_in*, cg_kvpair_t *request_out*)

Get the value corresponding to the key from the key/value passing system.

Parameters

in	*request_in*	A cg_kvpair_t with the key and ns members filled in.
out	*If*	successful, the key, ns, value_len and value fields are filled in in a cg_kvpair_t structure allocated by callee. Caller is responsible for freeing the allocated memory.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER if the ns/key (or combination) is empty. On any error, CG_STATUS_ERROR is returned.

4.4.5.9 cg_status_t cg_prov_kvpair_set (cg_kvpair_t *request*)

Set the value corresponding to the key in the key/value passing system.

Parameters

in	*request*	A cg_kvpair_t with the key, ns, value and value_len members filled in, value_len must be positive.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER if the kvpair (or parts of it) is empty. On any error, CG_STATUS_ERROR is returned. If CG_STATUS_RESOURCE_UNAVAILABLE is returned, the caller must retry at a later time, since a sync is in progress with the CloudGate Universe. The supplied cg_kvpair_t is not freed, that is the caller's responsibility.

4.4.5.10 cg_status_t cg_prov_kvpair_del (cg_kvpair_t *request*)

Delete the entry corresponding to the key from the key/value passing system.

Parameters

in	*request*	A cg_kvpair_t with the key and ns members filled in, value and value_len are ignored.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER if the ns/key (or combination) is empty. On any error, CG_STATUS_ERROR is returned. If CG_STATUS_RESOURCE_UNAVAILABLE is returned, the caller must retry at a later time, since a sync is in progress with the CloudGate Universe. We treat the cg_kvpair_t as readonly, caller is responsible for freeing.

4.4.5.11 cg_status_t cg_prov_kvpair_list (char *ns*, int *n_keys*, cg_key_t *keys*)

Get a list of keys currently known in a given namespace.

Parameters

in	*ns*	The namespace in which to look for keys.
out	*keys*	Will contain a pointer to a list of cg_key_t, caller must free.
out	*n_keys*	Will contain the number of keys.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER if any of the parameters are 0 or malformed (ns too long, ...). CG_STATUS_RESOURCE_UNAVAILABLE if the key/value store is currently unavailable (f.e. being synced). CG_STATUS_ERROR on misc errors. The keys parameter will only contain something allocated on CG_STATUS_OK.

4.4.5.12 cg_status_t cg_prov_kvpair_register_update_callback (cg_prov_kvpair_update_cb_t *cb*, void *context*, const char *ns*)

Register a callback for key/value updates.

Parameters

in	*cb*	A cg_prov_kvpair_update_cb_t callback.
in	*context*	Context data passed to the callback (untouched).
in	*ns*	The namespace to watch for updates.

Returns

CG_STATUS_OK if successful. CG_STATUS_INVALID_PARAMETER if no cb provided. CG_STATUS_ERROR for unspecified errors.

4.4.5.13 cg_status_t cg_prov_kvpair_deregister_update_callback (cg_prov_kvpair_update_cb_t *cb*, void *context*, const char *ns*)

Deregister a previously registered callback

Parameters

in	*cb*	Callback (previously registered) to remove
in	*context*	Context pointer which was previously registered
in	*ns*	The namespace being watched by the callback to deregister.

Returns

CG_STATUS_OK if succesful. CG_STATUS_INVALID_PARAMETER if no matching callback/context combination was found to remove. CG_STATUS_ERROR for unspecified errors.

4.4.5.14 cg_status_t cg_get_slot_list (cg_slot_list_t *slot*)

Get the list of slots in the device

Parameters

out	*slot*	List of slots. Caller will need to free slot.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER when provided with an invalid parameter or CG_STATUS_ERROR upon error.

4.4.5.15 cg_status_t cg_reset_system (uint8_t *fact_restore*, const char *reason*)

Request a device reset. Optionally specify if this reset should include a factory restore or not.

Parameters

in	*fact_restore*	Request (0): a normal reset or (1): a factory restore which is done during the reboot.
in	*reason*	Specify a reason for the reset. This reason will be logged.

Returns

CG_STATUS_OK if successful or CG_STATUS_ERROR upon error. If the device is unable to comply at that moment, CG_STATUS_RESOURCE_BUSY will be returned.

4.4.5.16 cg_status_t cg_reset_peripherals (void)

Power cycle all peripherals.

Returns

CG_STATUS_OK if successful, CG_STATUS_RESOURCE_BUSY if the device is unable to comply or CG_STATUS_ERROR upon error.

4.4.5.17 cg_status_t cg_system_log (uint16_t *log_level*, const char *format*, ...)

Log a line to the system log

Parameters

in	*level*	Log level
in	*format*	Log format

Returns

CG_STATUS_OK

4.4.5.18 cg_status_t cg_status_t cg_set_dev_mode (uint8_t *enable*, cg_dev_feature_t *feature*, void *options*)

Enable/disable a development feature

Parameters

in	*enable*	If true, the feature is enabled, if false, the feature is disabled again
in	*feature*	the feature to enable/disable
in	*options*	various configuration options of the different development features: char requested_root_passwd in case of CG_DEV_ROOT_ACCESS, int level in case of CG_DEV_MINIVAN_DEBUG, ...

Returns

CG_STATUS_OK on success, CG_STATUS_ERROR upon error, CG_STATUS_INVALID_PARAMETER when feature is not supported or out of range

4.4.5.19 cg_status_t cg_system_power_down (const cg_system_boot_condition_t *system_boot_condition*)

Power down the complete system. If the bit CG_SYSTEM_ARM_IGNITION_SENSE of the parameter system_boot_condition is set the system will power down at function call. When the ignition sense line is armed the system will boot. In case the mask bit is not set, the system will reboot at function call If the bit CG_SYSTEM_TIMED_WAKEUP of the parameter system_boot_condition is set the system will power down at function call. When the "timed_wakeup" time has elapsed the system will wake up.

Parameters

in	*system_boot_condition*	: The condition needed to initiate system boot

Returns

CG_STATUS_OK on success, CG_STATUS_ERROR upon error, CG_STATUS_INVALID_PARAMETER if parameter is invalid or CG_STATUS_RESOURCE_BUSY if already called

4.4.5.20 cg_status_t cg_system_set_timed_wakeup (uint32_t *seconds*)

Sets the time in seconds to wake up the Cloudgate after a shutdown

Parameters

in	*seconds*	: Time specified in seconds

Returns

CG_STATUS_OK on success, CG_STATUS_INVALID_PARAMETER if parameter is invalid and CG_STATUS_ERROR upon error.

4.4.5.21 cg_status_t cg_system_set_date_time (cg_date_time_t *date_time*)

Sets Calendar date and time

Parameters

in	*date_time,:*	Holds calendar date and time, format can be local or UTC. Calendar date and time components are declared in "struct tm" (see "time.h" for details).

Returns

CG_STATUS_OK on success, CG_STATUS_ERROR upon error or CG_STATUS_INVALID_PARAMETER if paramater is invalid

4.4.5.22 cg_status_t cg_system_get_date_time (cg_time_format_t *time_format*, cg_date_time_t *date_time*)

Retrieves Calender date and time

Parameters

in	*time_format*	: Specifies local time or UTC time.
out	*date_time,:*	Holds calendar date and time, format can be local or UTC. Parameter is allocated by the callee. The caller will be responsible for freeing the allocated memory. Calendar date and time components are declared in "struct tm" (see "time.h" for details).

Returns

CG_STATUS_OK on success, CG_STATUS_ERROR upon error, CG_STATUS_INVALID_PARAMETER if paramater is invalid

4.4.5.23 cg_status_t cg_system_set_timezone (cg_timezone_t *timezone*)

Sets time zone

Parameters

in	*timezone,:*	at least utc_offset_h, utc_offset_m and desc must be filled in

Returns

CG_STATUS_OK on success, CG_STATUS_ERROR upon error or CG_STATUS_INVALID_PARAMETER if paramater is invalid

4.4.5.24 cg_status_t cg_system_set_ntp_server (char *ntp_server_url*)

Sets NTP server url

Parameters

in	*ntp_server_url,:*	is a fully qualified domain name.

Returns

CG_STATUS_OK on success, CG_STATUS_ERROR upon error or CG_STATUS_INVALID_PARAMETER if paramater is invalid

4.4.5.25 cg_status_t cg_system_get_ntp_server (char *ntp_server_url*)

Retrieves NTP server url

Parameters

out	*ntp_server_url,:*	Is a fully qualified domain name and allocated by the callee. The caller will be responsible for freeing the allocated memory.

Returns

CG_STATUS_OK on success, CG_STATUS_ERROR upon error, CG_STATUS_INVALID_PARAMETER if paramater is invalid

4.4.5.26 cg_status_t cg_system_sync_ntp (void)

Synchronizes time with Network Time Server and sets Real Time Clock if present

Returns

CG_STATUS_OK on success or CG_STATUS_ERROR upon error.

4.4.5.27 cg_status_t cg_system_sync_ntp_register_callback (cg_system_sync_ntp_cb_t *cb*, void *context*)

Registers callback for ntp synchronisation

Parameters

in	*callback*	: called when ntp synchronisation status changes
in	*context*	Context parameter passed on to the callback when called.

Returns

CG_STATUS_OK on success, CG_STATUS_INVALID_PARAMETER if parameter is invalid, or CG_STATUS_ERROR upon error.

4.4.5.28 cg_status_t cg_system_sync_ntp_deregister_callback (cg_system_sync_ntp_cb_t *cb*, void *context*)

Deregisters callback for ntp synchronisation

Parameters

in	*callback*	: called when ntp synchronisation status changes
in	*context*	Context parameter passed on to the callback when called.

Returns

CG_STATUS_OK on success, CG_STATUS_INVALID_PARAMETER if parameter is invalid, or CG_STATUS_ERROR upon error.

4.5 GPS functions

Data Structures

- struct cg_gps_t

  Structure containing all GPS properties.
- struct cg_gps_certificate_t

  root certificate to validate the SUPL server.

Macros

- #define CG_MAX_CERTIFICATES 5
- #define CG_MAX_LEN_CERTIFICATE 2000

Typedefs

- typedef void(cg_gps_nmea_cb_t)(const char nmea_data, void context)
- typedef void(cg_gps_assisted_cb_t)(cg_gps_assisted_connection_status_t status)

Enumerations

- enum cg_gps_status_t { CG_GPS_PRESENT = 0x1, CG_GPS_ENABLED = 0x2 }

  bitfield describing various status-flags for GPS
- enum cg_gps_assisted_connection_status_t { CG_GPS_ASSISTED_CONNECTION_SUCCESS, CG_GPS_ASSISTED_CONNECTION_FAILURE }

  latest SUPL server connection indication
- enum cg_gps_assisted_function_t { CG_GPS_ASSISTED_DISABLE, CG_GPS_ASSISTED_NON_SECURE_SUPL, CG_GPS_ASSISTED_SECURE_SUPL }

  disable or enable AGPS with a secure or non secure connection to the SUPL server.

Functions

- cg_status_t cg_gps_get_status (cg_gps_t gps)
- cg_status_t cg_gps_set_enabled (int enabled)
- cg_status_t cg_gps_set_reporting_interval (uint32_t interval)
- cg_status_t cg_gps_set_assisted (cg_gps_assisted_function_t function, const char url, uint32_t cert_count, const cg_gps_certificate_t cert)
- cg_status_t cg_gps_get_assisted (cg_gps_assisted_function_t function, char url)
- cg_status_t cg_gps_register_nmea_callback (cg_gps_nmea_cb_t cb, void context)
- cg_status_t cg_gps_deregister_nmea_callback (cg_gps_nmea_cb_t cb, void context)
- cg_status_t cg_gps_register_supl_connection_status_callback (cg_gps_assisted_cb_t cb)
- cg_status_t cg_gps_deregister_supl_connection_status_callback (void)

4.5.1 Detailed Description

4.5.2 Macro Definition Documentation

4.5.2.1 #define CG_MAX_CERTIFICATES 5

Note

These two methods can be used to receive GPS NMEA data:

- Using the GPS SDK NMEA callback (cg_gps_nmea_cb_t): multiple applications can receive simultaneous GPS NMEA data
- Using the GPS serial port ttyUSB2 after claiming the GPS port with cg_device_claim().The maximum number of certificates that can be written

4.5.2.2 #define CG_MAX_LEN_CERTIFICATE 2000

The maximum length of one certificate

4.5.3 Typedef Documentation

4.5.3.1 typedef void(cg_gps_nmea_cb_t)(const char nmea_data, void context)

Callback for reporting NMEA data

Parameters

in	*context*	Context as set in cg_gps_register_nmea_callback
in	*nmea_data*	NMEA character data

4.5.3.2 typedef void(cg_gps_assisted_cb_t)(cg_gps_assisted_connection_status_t status)

Callback for reporting assisted GPS SUPL server connection status

Parameters

in	*connection*	status

4.5.4 Enumeration Type Documentation

4.5.4.1 enum cg_gps_status_t

bitfield describing various status-flags for GPS

Enumerator

CG_GPS_PRESENT Flag indicating if GPS functionality is physically present on the device

CG_GPS_ENABLED Flag indicating if GPS functionality is currently enabled

4.5.4.2 enum cg_gps_assisted_connection_status_t

latest SUPL server connection indication

Enumerator

CG_GPS_ASSISTED_CONNECTION_SUCCESS Indication that the SUPL server connection passed and that data was successfully retrieved

CG_GPS_ASSISTED_CONNECTION_FAILURE Indication that the SUPL server connection was started and an error occurred

4.5.4.3 enum cg_gps_assisted_function_t

disable or enable AGPS with a secure or non secure connection to the SUPL server.

Enumerator

CG_GPS_ASSISTED_DISABLE disable AGPS: GPS will work standalone

CG_GPS_ASSISTED_NON_SECURE_SUPL enable AGPS non secure connection to SUPL server.

CG_GPS_ASSISTED_SECURE_SUPL enable AGPS. A secure connection will be made to the SUPL server. (root certificate must be provided)

4.5.5 Function Documentation

4.5.5.1 cg_status_t cg_gps_get_status (cg_gps_t *gps*)

Returns the current GPS status Note: even if GPS functionality is present, it could still be disabled.

Parameters

out	*gps*	Callee-allocated structure which is filled in with the GPS parameters.

Returns

CG_STATUS_OK.

4.5.5.2 cg_status_t cg_gps_set_enabled (int *enabled*)

Enables or disables GPS functionality

Parameters

in	*enabled*	Set to TRUE to enable GPS. Set to FALSE to disable GPS.

Returns

CG_STATUS_OK if setting was successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter, CG_STATUS_ERROR (failure or no GPS device present)

Note

The CloudGate 3G rev 3 does not allow combined diversity and GPS. If diversity is enabled then this function will return CG_STATUS_ERROR.

4.5.5.3 cg_status_t cg_gps_set_reporting_interval (uint32_t *interval*)

Set GPS data reporting interval

Parameters

in	*interval*	Reporting interval in seconds

Returns

CG_STATUS_OK if setting was successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter, CG_STATUS_ERROR (failure or no GPS device present)

4.5.5.4 cg_status_t cg_gps_set_assisted (cg_gps_assisted_function_t *function*, const char *url*, uint32_t *cert_count*, const cg_gps_certificate_t *cert*)

Set AGPS connection parameters

Parameters

in	*function,:*	disable, enable secure or enable unsecure.
in	*url,:*	URL from SUPL server.
	i]	cert_count: number of root certificates that follow (maximum CG_MAX_CERTIFICATES)
in	*certificates,:*	for validating the SUPL server certificate (CG_GPS_ASSISTED_SECURE_SUPL only).

Returns

CG_STATUS_OK if setting was successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter, CG_STATUS_ERROR (failure or no GPS device present)

Note

The SUPL server will only be conntacted if a valid PIN and APN is present, the device is registered, the GPS data is invalid and needs updating.
This function will first stop GPS tracking (if already tracking) and then store the settings. Stopping GPS tracking could take about 60 seconds! It is therefore recommended to init agps before calling cg_gps_set_enabled(TRUE).

Warning

Assited gps will not respect the "don't connect when roaming" setting.

example using google suple server secure url: "supl.google.com:7275" unsecure url: "supl.google.com:7276"

4.5.5.5 cg_status_t cg_gps_get_assisted (cg_gps_assisted_function_t *function*, char *url*)

Get AGPS connection parameters

Parameters

out	*pointer*	to function that will be set to: disable, enable secure or enable unsecure.
out	*gps*	Callee-allocated string with supl addr URL when ASSISTED GPS is enabled

Returns

CG_STATUS_OK if setting was successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter, CG_STATUS_ERROR (failure or no GPS device present)

Note

certificates can not be read from the gps device.

4.5.5.6 cg_status_t cg_gps_register_nmea_callback (cg_gps_nmea_cb_t *cb*, void *context*)

Register a callback for NMEA data. It's possible to register multiple callbacks as long as either the callback or the context is different.

Parameters

in	*cb*	Callback for NMEA data.
in	*context*	Context data passed to cb when called.

Returns

CG_STATUS_OK if successful. CG_STATUS_INVALID_PARAMETER if no callback has been provided. CG_STATUS_ERROR (failure or no GPS device present)

4.5.5.7 cg_status_t cg_gps_deregister_nmea_callback (cg_gps_nmea_cb_t *cb*, void *context*)

Deregister a previously registered callback

Parameters

in	*cb*	Callback to be called when NMEA data is received
in	*context*	Context data which was passed on to the register call

Returns

CG_STATUS_OK if successful. CG_STATUS_INVALID_PARAMETER if no matching callback and context has been found. CG_STATUS_ERROR (unknown error)

4.5.5.8 cg_status_t cg_gps_register_supl_connection_status_callback (cg_gps_assisted_cb_t *cb*)

Register a callback for SUPL server connection status. Only one callback can be registered and must be cleared afterwards.

Parameters

in	*cb*	Callback for SUPL connection status.

Returns

CG_STATUS_OK if successful. CG_STATUS_INVALID_PARAMETER if no callback has been provided. CG_STATUS_ERROR (failure or no GPS device present)

4.5.5.9 cg_status_t cg_gps_deregister_supl_connection_status_callback (void)

Deregister a previously registered callback

Returns

CG_STATUS_OK if successful. CG_STATUS_ERROR (unknown error)

4.6 HAL

Data Structures

- struct cg_hal_serial_rs485_cfg_t
  Serial RS485-specific configuration settings.
- struct cg_hal_serial_cfg_t
  Generic configuration settings for a serial interface.

Enumerations

- enum cg_hal_serial_mode_t { CG_HAL_SERIAL_RS232 = 0x1, CG_HAL_SERIAL_RS485 = 0x2 }
  modes for RS485 or RS232

SERIAL - specific functions

- cg_status_t cg_hal_serial_set_config (const char dev_name, const cg_hal_serial_cfg_t config, int reboot_needed)

4.6.1 Detailed Description

4.6.2 Enumeration Type Documentation

4.6.2.1 enum cg_hal_serial_mode_t

modes for RS485 or RS232

Enumerator

CG_HAL_SERIAL_RS232 RS232 mode

CG_HAL_SERIAL_RS485 RS485 mode

4.6.3 Function Documentation

4.6.3.1 cg_status_t cg_hal_serial_set_config (const char *dev_name*, const cg_hal_serial_cfg_t *config*, int *reboot_needed*)

Set the serial interface configuration for RS485 or RS232

Parameters

in	*dev_name*	SERIAL device name
in	*config*	parameters
out	*reboot_needed*	indication that a reboot is needed before the new configuration is used.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or device name or CG_STATUS_ERROR in case of an error.

4.7 Networking

Data Structures

- struct cg_ip_addr_t
  Structure describing a IP address.
- struct cg_net_ip_config_t
  Structure describing IP configuration of an interface.
- struct cg_net_leasetime_t
  Structure describing a leasetime.
- struct cg_net_dhcp_config_t
  Structure used to configure DHCP server functionality.
- struct cg_net_lan_cfg_t
  LAN-specific configuration settings for a network interface.
- struct cg_net_wan_cfg_t
  WAN-specific configuration settings for a network interface.
- struct cg_net_cfg_t
  Generic configuration settings for a network interface.
- struct cg_net_if_t
  Data structure describing a network interface in the system.
- struct cg_net_watchdog_settings_t
  Settings structure for the connection watchdog.
- struct cg_net_stats_t
  Structure that contains the data usage statistics for an interface.

Typedefs

- typedef char cg_dev_name_t [CG_DEV_NAME_MAX]
- typedef void(itf_status_callback_t)(const char dev_name, cg_net_if_status_t status)
- typedef void(internet_status_callback_t)(cg_net_internet_status_t status)

Enumerations

- enum cg_net_if_status_t { CG_NET_INTERFACE_ENABLED = 0x1, CG_NET_INTERFACE_CONFIGURED = 0x2, CG_NET_INTERFACE_CONNECTED = 0x4, CG_NET_INTERFACE_CAN_CONNECT = 0x8 }
  Interface status bitmasks.
- enum cg_net_internet_status_t { CG_NET_INTERNET_DISCONNECTED = 0, CG_NET_INTERNET_CONNECTING = 2, CG_NET_INTERNET_CONNECTED, CG_NET_INTERNET_ON_DEMAND }
  The internet connection status. Not interface-specific but shows the general state of internet (WAN) connectivity of the device. For clients who just want to know if it is possible to reach the internet, without having to look at all the WAN interfaces to know if one is connected or not.
- enum cg_net_zone_t { CG_NET_ZONE_LAN = 0, CG_NET_ZONE_WAN }
  Enumeration to describe the different zones (LAN, WAN)
- enum cg_net_mode_t { CG_NET_MODE_ALWAYS_ON = 0, CG_NET_MODE_ON_DEMAND }
  Describes the different connection modes.
- enum cg_net_if_type_t { CG_NET_TYPE_WWAN = 0, CG_NET_TYPE_NIC, CG_NET_TYPE_WLAN_AP, CG_NET_TYPE_WLAN, CG_NET_TYPE_BRIDGE, CG_NET_TYPE_IPSEC }
  Enumeration listing all the different types of possible interfaces.
- enum cg_net_ip_config_type_t { CG_STATIC = 0, CG_DHCP }

Type of IP address configuration.

- enum cg_net_conn_strat_t { CG_NET_STRATEGY_MANUAL = 0, CG_NET_STRATEGY_PRIORITY }

  Enumeration of the different connection strategies. The device is able to execute different strategies to ensure connectivity automatically.

- enum cg_net_lease_unit_t { CG_NET_UNIT_MINUTE = 0, CG_NET_UNIT_HOUR, CG_NET_UNIT_DAY }

  Enumeration listing all the different units of the leasetime.

- enum cg_net_watchdog_action_t { **CG_CONN_WD_ACTION_RECONNECT** = 0, **CG_CONN_WD_ACTION_RESET** }

  Enumeration of the different watchdog actions.

Functions

- cg_status_t cg_net_get_interface_list (uint32_t num_interfaces, cg_net_if_t interfaces)
- cg_status_t cg_net_get_interface (const char dev_name, cg_net_if_t interface)
- cg_status_t cg_net_get_interface_by_type (cg_net_if_type_t type, uint32_t num_interfaces, cg_net_if_t itfs)
- cg_status_t cg_net_interface_set_enabled (const char dev_name, int enabled)
- cg_status_t cg_net_set_config (const char dev_name, const cg_net_cfg_t config)
- cg_status_t cg_net_register_interface_events (const char dev_name, itf_status_callback_t cb)
- cg_status_t cg_net_deregister_interface_events (const char dev_name)
- cg_status_t cg_net_set_manual_conn_strat (const char dev_name)
- cg_status_t cg_net_get_manual_conn_device (char dev_name)
- cg_status_t cg_net_set_priority_conn_strat (uint32_t num_entries, cg_dev_name_t priority_list)
- cg_status_t cg_net_get_conn_priority_list (uint32_t num_entries, cg_dev_name_t priority_list)
- cg_status_t cg_net_get_conn_strat (cg_net_conn_strat_t strat)
- cg_status_t cg_net_connect (void)
- cg_status_t cg_net_disconnect (void)
- cg_status_t cg_net_get_internet_status (cg_net_internet_status_t status)
- cg_status_t cg_net_register_internet_events (internet_status_callback_t cb)
- cg_status_t cg_net_deregister_internet_events (void)
- cg_status_t cg_net_set_watchdog (cg_net_watchdog_settings_t settings)
- cg_status_t cg_net_get_watchdog (cg_net_watchdog_settings_t settings)
- cg_status_t cg_net_datacounter_set_enabled (const char dev_name, int enabled)
- cg_status_t cg_net_datacounter_get_enabled (const char dev_name, int enabled)
- cg_status_t cg_net_datacounter_reset_stats (const char dev_name)
- cg_status_t cg_net_datacounter_get_stats (const char dev_name, cg_net_stats_t stats)
- cg_status_t cg_net_set_mtu (const char dev_name, int mtu)
- cg_status_t cg_net_get_mtu (const char dev_name, int mtu)
- cg_status_t cg_net_set_disabled_at_boot (const char dev_name, int disabled)
- cg_status_t cg_net_get_disabled_at_boot (const char dev_name, int disabled)

4.7.1 Detailed Description

Note

TODO:

- Add tracking of number of clients connected to LAN interfaces

4.7.2 Typedef Documentation

4.7.2.1 typedef char cg_dev_name_t[CG_DEV_NAME_MAX]

Data type that contains a string of fixed length CG_DEV_NAME_MAX. Used in cg_net_get_conn-_priority_list and cg_net_set_priority_conn_strat.

4.7.2.2 typedef void(itf_status_callback_t)(const char dev_name, cg_net_if_status_t status)

Interface status callback type. Registered in cg_net_register_interface_events.

Parameters

in	*dev_name*	The interface name for which the event is received.
in	*status*	New cg_net_if_status_t status bits

4.7.2.3 typedef void(internet_status_callback_t)(cg_net_internet_status_t status)

Internet connection status callback. Registered in cg_net_register_internet_events

Parameters

in	*status*	The current internet status.

4.7.3 Enumeration Type Documentation

4.7.3.1 enum cg_net_if_status_t

Interface status bitmasks.

Enumerator

CG_NET_INTERFACE_ENABLED Interface is enabled

CG_NET_INTERFACE_CONFIGURED Interface has proper IP address configuration

CG_NET_INTERFACE_CONNECTED Interface is connected

CG_NET_INTERFACE_CAN_CONNECT Possible to make a connection on interface

4.7.3.2 enum cg_net_internet_status_t

The internet connection status. Not interface-specific but shows the general state of internet (WAN) connectivity of the device. For clients who just want to know if it is possible to reach the internet, without having to look at all the WAN interfaces to know if one is connected or not.

Enumerator

CG_NET_INTERNET_DISCONNECTED The device does not have an internet connection

CG_NET_INTERNET_CONNECTING The device is trying to set up an internet connection on an interface

CG_NET_INTERNET_CONNECTED The device is connected to the internet

CG_NET_INTERNET_ON_DEMAND The device has found a WAN interface to connect to, but the interface is configured to connect in on-demand mode, which means that the interface will only be connected when there is traffic outgoing

4.7.3.3 enum cg_net_zone_t

Enumeration to describe the different zones (LAN, WAN)

Enumerator

CG_NET_ZONE_LAN LAN interface (pointed to internal network)
CG_NET_ZONE_WAN WAN interface (pointed to internet)

4.7.3.4 enum cg_net_mode_t

Describes the different connection modes.

Enumerator

CG_NET_MODE_ALWAYS_ON The interface is always active
CG_NET_MODE_ON_DEMAND The interface is only brought up when there is traffic wanting to get out on the interface. This is particularly useful for interfaces where charges are incurred based on connection duration (phone lines, some 3G subscriptions)

4.7.3.5 enum cg_net_if_type_t

Enumeration listing all the different types of possible interfaces.

Enumerator

CG_NET_TYPE_WWAN Mobile 3G data interface
CG_NET_TYPE_NIC Regular ethernet interface
CG_NET_TYPE_WLAN_AP Wireless LAN Access Point interface
CG_NET_TYPE_WLAN Wireless LAN Station interface
CG_NET_TYPE_BRIDGE Bridge interface
CG_NET_TYPE_IPSEC IPsec VPN interface

4.7.3.6 enum cg_net_ip_config_type_t

Type of IP address configuration.

Enumerator

CG_STATIC Static IP address configuration
CG_DHCP Dynamic DHCP IP address configuration

4.7.3.7 enum cg_net_conn_strat_t

Enumeration of the different connection strategies. The device is able to execute different strategies to ensure connectivity automatically.

Enumerator

CG_NET_STRATEGY_MANUAL Manual strategy. All automatic behaviour is disabled.
CG_NET_STRATEGY_PRIORITY Priority-based strategy. The device will try to maintain internet connectivity according to a priority list of interfaces. If a certain interface connection fails, it will move to the next interface in the priority list. If a higher-ranked interface becomes available again, it will try to move back to the higher-ranked interface.

4.7.3.8 enum cg_net_lease_unit_t

Enumeration listing all the different units of the leasetime.

Enumerator

CG_NET_UNIT_MINUTE leasetime unit minute

CG_NET_UNIT_HOUR leasetime unit hour

CG_NET_UNIT_DAY leasetime unit day

4.7.4 Function Documentation

4.7.4.1 cg_status_t cg_net_get_interface_list (uint32_t *num_interfaces*, cg_net_if_t *interfaces*)

Get a list of interfaces in the system

Parameters

out	*num_interfaces*	Integer which will be set to the number of interfaces
out	*interfaces*	interfaces will be set to an array of cg_net_if_t. The number of elements in the array equal to num_interfaces. Caller needs to free the array.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.7.4.2 cg_status_t cg_net_get_interface (const char *dev_name*, cg_net_if_t *interface*)

Get a single interface in the system

Parameters

in	*dev_name*	The interface name for which to retrieve the interface info.
out	*interface*	interface will point to a cg_net_if_t object when the device is found, NULL otherwise. Caller needs to free the cg_net_if_t object.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter, CG_STATUS_RESOURCE_BUSY if a device is too cold or hot or CG_STATUS_ERROR in case of an error.

4.7.4.3 cg_status_t cg_net_get_interface_by_type (cg_net_if_type_t *type*, uint32_t *num_interfaces*, cg_net_if_t *itfs*)

Get a list of interfaces in the system from a certain type.

Parameters

in	*type*	The type of interface to list.
out	*num_interfaces*	Integer which will be set to the number of interfaces
out	*itfs*	itfs will be set to an array of cg_net_if_t. The number of elements in the array equal to num_interfaces. Caller needs to free the array.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.7.4.4 cg_status_t cg_net_interface_set_enabled (const char *dev_name*, int *enabled*)

Enable/disable a particular interface

Parameters

in	*dev_name*	The interface name for which the configuration is set. This is the dev_name member in the cg_net_if_t structure.
in	*enabled*	Set to TRUE to enable the interface. Set to FALSE to disable interface.

Returns

CG_STATUS_OK if successful. CG_STATUS_ERROR in case of error.

4.7.4.5 cg_status_t cg_net_set_config (const char *dev_name*, const cg_net_cfg_t *config*)

Set the configuration for an interface

Parameters

in	*dev_name*	The interface name for which the configuration is set. This is the dev_name member in the cg_net_if_t structure.
in	*config*	Caller-allocated configuration settings

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER if dev_name or config contains incorrect data, CG_STATUS_ERROR if failed to set the new config, CG_STATUS_RESOURCE_BUSY if a device is too cold or hot.

4.7.4.6 cg_status_t cg_net_register_interface_events (const char *dev_name*, itf_status_callback_t *cb*)

Register to receive events concerning interface state changes

Parameters

in	*dev_name*	The interface name for which to receive interface status changes
in	*cb*	Callback which gets called whenever the interface changes status.

Returns

CG_STATUS_OK if successful. CG_STATUS_ERROR in case of error.

4.7.4.7 cg_status_t cg_net_deregister_interface_events (const char *dev_name*)

Deregister to receive interface events.

Parameters

in	*dev_name*	The interface name for which to deregister callbacks.

4.7.4.8 cg_status_t cg_net_set_manual_conn_strat (const char *dev_name*)

Set the internet connection strategy to manual. If there was an active internet connection before, a new connection will be set up using the newly selected interface. Otherwise a separate connect call is necessary.

Parameters

in	*dev_name*	The WAN interface to use for internet connection.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER if dev_name contains an invalid device or CG_STATUS_ERROR in case of an error.

4.7.4.9 cg_status_t cg_net_get_manual_conn_device (char *dev_name*)

Get the interface which is used when using manual connection strategy. Returns an error when the current connection strategy is not manual.

Parameters

out	*dev_name*	Is filled in with the name of the interface that is being used to connect. Caller has to free dev_name.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER if dev_name is an invalid pointer or CG_STATUS_ERROR in case of an error.

4.7.4.10 cg_status_t cg_net_set_priority_conn_strat (uint32_t *num_entries*, cg_dev_name_t *priority_list*)

Set the internet connection strategy to priority

Parameters

in	*num_entries*	The number of interface names in priority_list
in	*priority_list*	An array of cg_dev_name_t containing network interface names. The order in which the interface names are put in the array determines the priority. Priority decreases with array index increments.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER if num_entries or priority_list contains an invalid value or CG_STATUS_ERROR in case of an error.

4.7.4.11 cg_status_t cg_net_get_conn_priority_list (uint32_t *num_entries*, cg_dev_name_t *priority_list*)

Get the priority list which is used when using priority-based connection strategy. Returns error when the current connection strategy is not priority-based.

Parameters

out	*num_entries*	Is filled in with the number of entries in priority_list.
out	*priority_list*	is set to an array of cg_dev_name_t. The caller is responsible for freeing priority_list.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER if num_entries or priority_list contains an invalid pointer or CG_STATUS_ERROR in case of an error.

4.7.4.12 cg_status_t cg_net_get_conn_strat (cg_net_conn_strat_t *strat*)

Get the current configured connection strategy

Parameters

out	*strat*	Will be filled in with the current connection strategy.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER if strat is an invalid pointer or CG_STATUS_ERROR in case of an error.

4.7.4.13 cg_status_t cg_net_connect (void)

Instructs the system to set up internet connection. Register for internet connection events to monitor the progress. If the device is configured for manual connection to a specific interface, the device will try to connect to that interface. If the device is configured for priority-based connection, the system will try to connect a WAN interface according to the priority list. This is a persistent setting across power cycles.

Returns

CG_STATUS_OK if successful. CG_STATUS_ERROR in case no connection could be setup within 60 seconds although the device will keep on trying to connect.

4.7.4.14 cg_status_t cg_net_disconnect (void)

Disconnection internet connection. This will return immediately & should put the internet connection state to CG_NET_INTERNET_DISCONNECTED. This is a persistent setting across power cycles.

Returns

CG_STATUS_OK if successful. CG_STATUS_ERROR in case of error.

4.7.4.15 cg_status_t cg_net_get_internet_status (cg_net_internet_status_t *status*)

Returns the current internet connection state.

Parameters

out	*status*	Internet connection status

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.7.4.16 cg_status_t cg_net_register_internet_events (internet_status_callback_t *cb*)

Register for internet connection status events. These are not interface-specific events, but events to describe the general internet connectivity status of the device.

Returns

CG_STATUS_OK

4.7.4.17 cg_status_t cg_net_deregister_internet_events (void)

Deregister for internet connection status events

Returns

CG_STATUS_OK

4.7.4.18 cg_status_t cg_net_set_watchdog (cg_net_watchdog_settings_t *settings*)

Set the connection watchdog settings. If the enabled flag in the settings is set to FALSE, the connection watchdog will be disabled but the other settings will not be changed.

Parameters

in	*settings*	Watchdog settings

Returns

CG_STATUS_OK if successful. CG_STATUS_ERROR in case of error. CG_STATUS_INVALID_PARAMETER in case of wrong parameters.

4.7.4.19 cg_status_t cg_net_get_watchdog (cg_net_watchdog_settings_t *settings*)

Query the connection watchdog settings

Parameters

out	*settings*	Double pointer to a structure that will be allocated and filled in with the connection watchdog settings.

Returns

CG_STATUS_OK if successful, CG_STATUS_ERROR in case of error or CG_STATUS_INVALID_PARAMETER if setting is a null pointer.

4.7.4.20 cg_status_t cg_net_datacounter_set_enabled (const char *dev_name*, int *enabled*)

Start or stop data usage tracking for a specific interface. This setting will be persistent across reboots. Enabling, even when already enabled, ensures all usage tracking data is saved to flash at least every 10 minutes. This more frequent writing is not persistent across reboot. Disabling will cause the "Data Counters" section on the CloudGate webpage for this interface to stop.

Parameters

in	*dev_name*	The interface name for which the data counter needs to be enabled or disabled.
in	*enabled*	TRUE if data usage tracking needs to be enabled for this interface, FALSE if it needs to be disabled.

Returns

CG_STATUS_OK if successful, CG_STATUS_ERROR when an unexpected situation arises and CG_STATUS_INVALID_PARAMETER if dev_name is a NULL pointer or if dev_name does not exist.

4.7.4.21 cg_status_t cg_net_datacounter_get_enabled (const char *dev_name*, int *enabled*)

Get the data usage tracking state for a specific interface.

Parameters

in	*dev_name*	The interface name for which the data usage tracking state needs to be retrieved.
out	*enabled*	TRUE if data usage tracking is enabled for this interface, FALSE if disabled.

Returns

CG_STATUS_OK if successful, CG_STATUS_ERROR when an unexpected situation arises , CG_STATUS_INVALID_PARAMETER if dev_name or enabled is a NULL pointer or if dev_name does not exist, CG_STATUS_RESOURCE_BUSY if a device is too cold or hot.

4.7.4.22 cg_status_t cg_net_datacounter_reset_stats (const char *dev_name*)

Reset the data usage statistics for a specific interface.

Parameters

in	*dev_name*	The interface name for which the data counter needs to be reset.

Returns

CG_STATUS_OK if successful, CG_STATUS_ERROR when an unexpected situation arises and CG_STATUS_INVALID_PARAMETER if dev_name is a NULL pointer or if dev_name does not exist.

4.7.4.23 cg_status_t cg_net_datacounter_get_stats (const char *dev_name*, cg_net_stats_t *stats*)

Get the data usage statistics for a specific interface.

Parameters

in	*dev_name*	The interface name for which the data usage statistics need to be retrieved.
out	*stats*	Caller-allocated struct that will be filled with current data usage statistics for this interface.

Returns

CG_STATUS_OK if successful, CG_STATUS_ERROR when an unexpected situation arises, CG_STATUS_INVALID_PARAMETER if dev_name or stats is a NULL pointer or if dev_name does not exist, CG_STATUS_RESOURCE_BUSY if a device is too cold or hot.

4.7.4.24 cg_status_t cg_net_set_mtu (const char *dev_name*, int *mtu*)

Set a new MTU for a specific interface.

Parameters

in	*dev_name*	The interface name for which the MTU needs to be set.
in	*mtu*	The new MTU value, this must be a number from 68 to 1500.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER if dev_name or mtu contains incorrect values, CG_STATUS_ERROR if failed to set the new MTU, CG_STATUS_RESOURCE_BUSY if a device is too cold or hot.

4.7.4.25 cg_status_t cg_net_get_mtu (const char *dev_name*, int *mtu*)

Get the current MTU for a specific interface.

Parameters

in	*dev_name*	The interface name for which the MTU needs to be retrieved.
out	*mtu*	The current MTU value

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER if dev_name or mtu contains incorrect values, CG_STATUS_ERROR if failed to get the current MTU, CG_STATUS_RESOURCE_BUSY if a device is too cold or hot.

4.7.4.26 cg_status_t cg_net_set_disabled_at_boot (const char *dev_name*, int *disabled*)

Set a device disabled at boot. This will allow software to take full control of a device before enabling it. The device will always start disabled and must be started by cg_net_interface_set_enabled()

Parameters

in	*dev_name*	The interface name for which the data counter needs to be reset.
in	*disabled*	TRUE: interface will be disabled at boot, FALSE: current interface state will remain unmodified

Returns

CG_STATUS_OK if successful, CG_STATUS_ERROR when an unexpected situation arises and CG_STATUS_INVALID_PARAMETER if dev_name is a NULL pointer or if dev_name does not exist. Note: When clearing this feature the interface will not become enabled automatically. It will fall back to the last known state.

4.7.4.27 cg_status_t cg_net_get_disabled_at_boot (const char *dev_name*, int *disabled*)

Get the device disabled at boot status. The device will always start disabled and must be started by cg_net_interface_set_enabled()

Parameters

in	*dev_name*	The interface name for which the data counter needs to be reset.
out	*disabled*	TRUE: interface will be disabled at boot, FALSE: current interface state will remain unmodified

Returns

CG_STATUS_OK if successful, CG_STATUS_ERROR when an unexpected situation arises and CG_STATUS_INVALID_PARAMETER if dev_name is a NULL pointer or if dev_name does not exist.

4.8 SMS

Typedefs

- typedef void(cg_sms_cb_t)(const char dev_name, uint16_t sms_size, unsigned char sms, void context)

Functions

- cg_status_t cg_sms_set_smsc (const char dev_name, const char number)
- cg_status_t cg_sms_get_smsc (const char dev_name, char number)
- cg_status_t cg_sms_register_new_sms (const char dev_name, cg_sms_cb_t cb, void context)
- cg_status_t cg_sms_deregister_new_sms (const char dev_name, cg_sms_cb_t cb, void context)
- cg_status_t cg_sms_send (const char dev_name, uint16_t sms_size, const unsigned char sms)

4.8.1 Detailed Description

4.8.2 Typedef Documentation

4.8.2.1 typedef void(cg_sms_cb_t)(const char dev_name, uint16_t sms_size, unsigned char sms, void context)

Callback for receiving new sms's

Parameters

in	*dev_name*	WWAN device name
in	*sms_size*	size in bytes of the sms
in	*sms*	The SMS, in layer3 format for 3GPP2(CDMA) and in PDU format for 3GPP (GSM/UMTS)
in	*context*	User-defined context.

4.8.3 Function Documentation

4.8.3.1 cg_status_t cg_sms_set_smsc (const char *dev_name*, const char *number*)

Set the SMS Message Center number this is invalid for CDMA

Parameters

in	*dev_name*	WWAN device name
in	*number*	the SMSC number, eg. +3212345678. An empty smsc ("") will clear the configured smsc and go back to the SIM's default.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error. Invalid paramater include a too long smsc number (>20 digits) and invalid dev_names.

4.8.3.2 cg_status_t cg_sms_get_smsc (const char *dev_name*, char *number*)

Get the SMS Message Center number

Parameters

in	*dev_name*	WWAN device name
out	*number*	The SMSC number, eg. +3212345678. If no smsc is configured and the device isn't ready reading it from the SIM, "unknown" will be returned as smsc.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.8.3.3 cg_status_t cg_sms_register_new_sms (const char *dev_name*, cg_sms_cb_t *cb*, void *context*)

Register a callback for new SMS notifications

Parameters

in	*dev_name*	WWAN device name
in	*cb*	New SMS callback.
in	*context*	User-defined context passed on to cb when called.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.8.3.4 cg_status_t cg_sms_deregister_new_sms (const char *dev_name*, cg_sms_cb_t *cb*, void *context*)

Deregister a callback for new SMS notifications

Parameters

in	*dev_name*	WWAN device name
in	*cb*	Callback.
in	*context*	User-defined context passed on to cb when called.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.8.3.5 cg_status_t cg_sms_send (const char *dev_name*, uint16_t *sms_size*, const unsigned char *sms*)

Send an SMS. The function will cut the SMS in parts if it is too large to fit in a single part.

Parameters

in	*dev_name*	WWAN device name
in	*sms_size*	size in bytes of the sms
in	*sms*	The SMS, in layer3 format for 3GPP2(CDMA) and in PDU format for 3GPP (GSM/UMTS) For 3GPP the raw message in PDU format must include the SMSC address length identifier as the first byte of the message. If this byte is set to zero, the SMSC set with cg_sms-_set_smsc is used, or the default (SIM) otherwise. Otherwise, the first byte indicates the length, in bytes, of the SMSC address that is included after the first byte, but before the start of the actual PDU message.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.9 UI

Typedefs

- typedef char (cg_ui_json_cb_t)(char json_data, int logged_in, void context)
- typedef char (cg_ui_get_cb_t)(char query_string, int logged_in, void context)
- typedef void(cg_ui_session_cb)(unsigned int session_id, cg_ui_session_action_t action)

Enumerations

- enum cg_ui_session_action_t { **CG_UI_SESSION_ADD**, **CG_UI_SESSION_DEL**, **CG_UI_SESSION_PURGE** }

  Action to be taken on a session event.

Functions

- cg_status_t cg_ui_register_page (const char name, const char url)
- cg_status_t cg_ui_deregister_page (const char name)
- cg_status_t cg_ui_register_json_callback (const char identifier, cg_ui_json_cb_t cb, void context)
- cg_status_t cg_ui_deregister_json_callback (const char identifier)
- cg_status_t cg_ui_register_get_callback (const char identifier, cg_ui_get_cb_t cb, void context)
- cg_status_t cg_ui_deregister_get_callback (const char identifier)
- cg_status_t cg_ui_register_session_callback (cg_ui_session_cb cb)
- cg_status_t cg_ui_deregister_session_callback (void)

4.9.1 Detailed Description

Consider the following example to extend the CloudGate web UI.

In your SDK directory, save below configuration settings as config/config/m2mweb:

```
package m2mweb

config default 'customers'
   option path '/rom/mnt/cust/examples'

config replacement_ui 'ui'
   option url 'customers/install.html'

config replacement_logo 'logo'
   option url 'http://www.option.com/wp-content/uploads/2014/09/logo.png'

config ui 'custom_ui'
   option brand_logo 'http://www.option.com/wp-content/uploads/2014/09/logo.png'
   option company_logo 'http://www.option.com/wp-content/uploads/2014/09/logo.png'
   option favicon 'http://www.option.com/wp-content/uploads/2014/09/logo.png'
   option tagline '<font color="blue"><b>My new <u>tagline</u> in blue bold text</b></font>'
   option tabtitle 'My new browser tab title'
```

The first part shows how a new document root can be added:

- **customers** : the symlink which will be created in /www.
- **path** : absolute path to the new document root.

NOTE 1: the path needs to be absolute and it must exist.

NOTE 2: you can add multiple document roots.

NOTE 3: you cannot create a document root with the name 'api'.

The second part shows how the web UI, shown after logging in, can be replaced:

- **replacement_ui** : there can only be one replacement UI. The name is always 'ui'.
- **path** : relative or absolute path to the replacement page.

DEPRECATED: The third part shows how the CloudGate logo, at the topleft of the web UI, can be replaced.

This part is deprecated. Please use the custom_ui part.

- **replacement_logo** : there can only be one replacement logo. The name is always 'logo'.
- **url** : relative or absolute url to the new logo.

The last part shows how the GUI can be extended. You can change the brand logo, company logo, favicon, tagline and browser tab title.

The brand logo, company logo, favicon and tagline can also be hidden from the UI.

- **brand_logo** : the CloudGate logo, at the topleft of the web UI, can be replaced. Optimal size: 250 x 40 pixels.
- **company_logo** : the Option logo, at the topright of the web UI, can be replaced. Optimal size: 120 x 80 pixels.
- **tagline** : the 'Connecting THINGS to the cloud' text, at the topcenter of the web UI, can be replaced. HTML tags are supported. (Inline) CSS is not supported.
- **favicon** : the browser tab favicon can be replaced.
- **tabtitle** : the browser tab title can be replaced.

  To hide an item, set the value of an option to 'hidden'. For example, to hide the brand_logo do:

  config ui 'custom_ui'

  option brand_logo 'hidden'

4.9.2 Typedef Documentation

4.9.2.1 typedef char (cg_ui_json_cb_t)(char json_data, int logged_in, void context)

Callback for receiving JSON data. This function is not called for GET requests if a GET request handler is registered for the same identifier.

Parameters

in	*context*	Context as set in cg_ui_register_json_callback
in	*json_data*	JSON character data or an empty string if the request was not a POST request with content-type application/json.
in	*logged_in*	FALSE if not logged in.

Returns

A string that will be sent as a reply with content-type application/json. If the return value is NULL 'Status: 400 Bad Request' will be sent. This value will be freed by the caller.

4.9.2.2 typedef char (cg_ui_get_cb_t)(char query_string, int logged_in, void context)

Callback for receiving GET request.

Parameters

in	*context*	Context as set in cg_ui_register_get_callback
in	*query_string*	Any data following the ? in the requested URL
in	*logged_in*	FALSE if not logged in.

Returns

A string that will be sent as a reply with content-type application/json. If the return value is NULL 'Status: 400 Bad Request' will be sent. This value will be freed by the caller.

4.9.2.3 typedef void(cg_ui_session_cb)(unsigned int session_id, cg_ui_session_action_t action)

Callback for receiving session information when a user, for example, logs in via the web GUI.

Parameters

in	*session_id*	The session_id of the logged in/out user. Session id on only valid if action is CG_UI_SESSION_ADD or CG_UI_SESSION_DEL.
in	*action*	A session can be added, deleted or becomes invalid. On CG_UI_SESSION_PURGE, all active sessions are invalid.

4.9.3 Function Documentation

4.9.3.1 cg_status_t cg_ui_register_page (const char *name*, const char *url*)

Register a new link in the custom menu of the Option CloudGate webUI.

Parameters

in	*name*	The text displayed in the menu.
in	*url*	The destination URL of this link. This must be a RELATIVE url. It will be mapped to /www/your_relative_url. The path must exist!

Returns

CG_STATUS_OK if successful else CG_STATUS_ERROR.

4.9.3.2 cg_status_t cg_ui_deregister_page (const char *name*)

Deregister a link in the custom menu of the Option CloudGate webUI.

Parameters

in	*name*	The text displayed in the menu.

Returns

CG_STATUS_OK if successful else CG_STATUS_ERROR.

4.9.3.3 cg_status_t cg_ui_register_json_callback (const char *identifier*, cg_ui_json_cb_t *cb*, void *context*)

Register a callback for HTTP requests.

Parameters

in	*identifier*	Unique identifier used as the last part of the URL http://cloudgate/api/libcg/identifier
in	*cb*	Callback for JSON data.
in	*context*	Context data passed to cb when called.

Returns

CG_STATUS_OK if successful. CG_STATUS_INVALID_PARAMETER if no callback has been provided. CG_STATUS_ERROR if there is a JSON callback registered for this identifier or when another failure occurs.

4.9.3.4 cg_status_t cg_ui_deregister_json_callback (const char *identifier*)

Deregister a callback for HTTP requests.

Parameters

in	*identifier*	Unique identifier used as the last part of the URL http://cloudgate/api/libcg/identifier

Returns

CG_STATUS_OK if successful. CG_STATUS_ERROR in case of failure.

4.9.3.5 cg_status_t cg_ui_register_get_callback (const char *identifier*, cg_ui_get_cb_t *cb*, void *context*)

Register a callback for HTTP GET requests.

Parameters

in	*identifier*	Unique identifier used as the last part of the URL http://cloudgate/api/libcg/identifier
in	*cb*	Callback for GET data.
in	*context*	Context data passed to cb when called.

Returns

CG_STATUS_OK if successful. CG_STATUS_INVALID_PARAMETER if no callback has been provided. CG_STATUS_ERROR if there is a GET callback registered for this identifier or when another failure occurs.

4.9.3.6 cg_status_t cg_ui_deregister_get_callback (const char *identifier*)

Deregister a callback for HTTP GET requests.

Parameters

in	*identifier*	Unique identifier used as the last part of the URL http://cloudgate/api/libcg/identifier

Returns

CG_STATUS_OK if successful. CG_STATUS_ERROR in case of failure.

4.9.3.7 cg_status_t cg_ui_register_session_callback (cg_ui_session_cb *cb*)

Register a callback to receive session notifications when a user logs in/out to/from the web GUI

Parameters

in	*Call*	function of type cg_ui_session_cb

Returns

CG_STATUS_OK if successful. CG_STATUS_INVALID_PARAMETER if no callback has been provided. CG_STATUS_ERROR if there is a session callback registered for this identifier or when another failure occurs.

4.9.3.8 cg_status_t cg_ui_deregister_session_callback (void)

Deregister the callback for receiving session notifications

Returns

CG_STATUS_OK if successful. CG_STATUS_ERROR if a failure occurs.

4.10 Upgrade API

Typedefs

- typedef struct cg_upgrade cg_upgrade_t

Functions

- cg_status_t cg_upgrade_start (cg_upgrade_t handle)
- cg_status_t cg_upgrade_data (cg_upgrade_t handle, char buffer, uint32_t len)
- cg_status_t cg_upgrade_stop (cg_upgrade_t handle)

4.10.1 Detailed Description

4.10.2 Typedef Documentation

4.10.2.1 typedef struct cg_upgrade cg_upgrade_t

handle for upgrade API

Note

created by cg_uprade_start, used by cg_upgrade_data & cg_upgrade_stop

4.10.3 Function Documentation

4.10.3.1 cg_status_t cg_upgrade_start (cg_upgrade_t *handle*)

Prepares a device upgrade

Note

Switch off auto upgrade by the provisioning server to be sure the manual provisioning will not be overruled by an automatic provisioning from the provisioning server. After a successful call of cg_upgrade_start, call cg_upgrade_data to transfer data and finish the upgrade by calling cg_upgrade_stop, even if the upgrading process is stopped by a failing upgrade data transfer.

Parameters

out	*handle,:*	handle needed by other upgrade functions

Returns

CG_STATUS_OK if function call was successful, CG_STATUS_ERROR in case of error, CG_INVALID_PARAMETER if handle is NULL, CG_STATUS_RESOURCE_BUSY if there is an ongoing device upgrade busy

4.10.3.2 cg_status_t cg_upgrade_data (cg_upgrade_t *handle*, char *buffer*, uint32_t *len*)

Performs the upgrade based on data chunks

Once the last chunk of data has been successfully received by the gateway, the new image is commited and can't be reversed.

cg_upgrade_stop() indicates whether the upgrade was successful.

Note

Please limit the size of the data chunks to 16384 bytes, in order not to exhaust memory resources during upgrade.

Parameters

in	*handle,:*	handle returned by cg_upgrade_start
in	*buffer,:*	character buffer with a chunk of data
in	*length,:*	actual length of that chunk of data in bytes

Returns

CG_STATUS_OK if no errors were encountered, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter submitted, CG_STATUS_ERROR in case an upgrade failure was encountered

4.10.3.3 cg_status_t cg_upgrade_stop (cg_upgrade_t *handle*)

Destroys internal context and returns the overall status of the device upgrade

If the upgrade was successful then the gateway will boot with the new image after a reset.

cg_reset_system() can be called to reboot the gateway.

Parameters

in	*handle,:*	handle returned by cg_upgrade_start

Returns

CG_STATUS_OK if device upgrade was successful, CG_STATUS_INVALID_PARAMETER when a parameter is invalid, CG_STATUS_ERROR if device upgrade errors were encountered

4.11 WLAN

Data Structures

- struct cg_wlan_auth_params_t

  Authentication parameters for a WLAN client/access point.
- struct cg_wlan_network_t

  Description of a WLAN network.

Macros

- #define CG_SSID_MAX_LENGTH 32
- #define CG_PASSWORD_MAX_LENGTH 64

Typedefs

- typedef void(cg_wlan_network_list_cb_t)(cg_status_t status, const char dev_name, uint32_t num_entries, cg_wlan_network_t networks, void context)

Enumerations

- enum cg_wlan_auth_t { CG_WLAN_AUTH_OPEN = 0, CG_WLAN_AUTH_WEP, CG_WLAN_AUTH_WPA_PSK, CG_WLAN_AUTH_WPA2_PSK, CG_WLAN_AUTH_WPA_WPA2_PSK, CG_WLAN_AUTH_WPA_ENT, CG_WLAN_AUTH_WPA2_ENT, CG_WLAN_AUTH_WPA_WPA2_ENT }

  Different WLAN authentication type.

Functions

- cg_status_t cg_wlan_sta_scan_networks (const char dev_name, cg_wlan_network_list_cb_t cb, void context)
- cg_status_t cg_wlan_sta_set_network (const char dev_name, const char ssid, cg_wlan_auth_params_t auth)
- cg_status_t cg_wlan_sta_get_connected_network (const char dev_name, cg_wlan_network_t network)
- cg_status_t cg_wlan_sta_get_network_list (const char dev_name, uint32_t num_entries, cg_wlan_network_t networks)
- cg_status_t cg_wlan_sta_remove_network (const char dev_name, const char ssid, cg_wlan_auth_t auth)
- cg_status_t cg_wlan_ap_set_ssid (const char dev_name, const char ssid, uint8_t broadcast)
- cg_status_t cg_wlan_ap_get_ssid (const char dev_name, char ssid, uint8_t broadcast)
- cg_status_t cg_wlan_ap_set_channel (const char dev_name, uint16_t channel)
- cg_status_t cg_wlan_ap_get_channel (const char dev_name, uint16_t channel)
- cg_status_t cg_wlan_ap_set_auth_params (const char dev_name, cg_wlan_auth_params_t auth)
- cg_status_t cg_wlan_ap_get_auth_params (const char dev_name, cg_wlan_auth_params_t auth)

4.11.1 Detailed Description

4.11.2 Macro Definition Documentation

4.11.2.1 #define CG_SSID_MAX_LENGTH 32

Maximum length of the SSID

4.11.2.2 #define CG_PASSWORD_MAX_LENGTH 64

Maximum length of the password

4.11.3 Typedef Documentation

4.11.3.1 typedef void(cg_wlan_network_list_cb_t)(cg_status_t status, const char dev_name, uint32_t num_entries, cg_wlan_network_t networks, void context)

Callback for cg_wlan_sta_scan_networks. Will be called with a list of scanned networks. cg_wlan_network_t will not contain any valid channel numbers.

Parameters

in	*status*	Status of the operation. CG_STATUS_SUCCESS or CG_STATUS_ERROR.
in	*dev_name*	WLAN STA device name.
in	*num_entries*	Number of entries in *networks*.
in	*networks*	Array of cg_wlan_network_t structures.
in	*context*	Context parameter passed on to cg_wlan_sta_scan_networks.

4.11.4 Enumeration Type Documentation

4.11.4.1 enum cg_wlan_auth_t

Different WLAN authentication type.

Enumerator

CG_WLAN_AUTH_OPEN Open authentication (= no authentication)
CG_WLAN_AUTH_WEP WEP PSK authentication
CG_WLAN_AUTH_WPA_PSK WPA-PSK authentication
CG_WLAN_AUTH_WPA2_PSK WPA2-PSK authentication
CG_WLAN_AUTH_WPA_WPA2_PSK WPA-PSK or WPA2-PSK authentication
CG_WLAN_AUTH_WPA_ENT WPA-Enterprise authentication
CG_WLAN_AUTH_WPA2_ENT WPA2-Enterprise authentication
CG_WLAN_AUTH_WPA_WPA2_ENT WPA-Enterprise or WPA2-Enterprise authentication

4.11.5 Function Documentation

4.11.5.1 cg_status_t cg_wlan_sta_scan_networks (const char *dev_name*, cg_wlan_network_list_cb_t *cb*, void *context*)

Initiate a scan for WLAN networks.

Parameters

in	*dev_name*	WLAN STA device name.
in	*cb*	Callback function that is called when the network scan has finished.
in	*context*	User-defined context that is passed to the callback function.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.11.5.2 cg_status_t cg_wlan_sta_set_network (const char *dev_name*, const char *ssid*, cg_wlan_auth_params_t *auth*)

Add a network to the list of saved access points. An attempt will be made to make a network connection. The network with the best signal quality will be selected. The list of available networks can be queried with cg_wlan_sta_scan_networks. This function can also be used to change the password for the specified WLAN network.

Parameters

in	*dev_name*	WLAN STA device name.
in	*ssid*	The SSID of the network.
in	*auth*	The authentication parameters. If it is not known which authentication methods are used by the access point, you can query the list of available networks with cg_wlan_sta_scan_networks The result of that scan will contain the type of authentication which is necessary.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.11.5.3 cg_status_t cg_wlan_sta_get_connected_network (const char *dev_name*, cg_wlan_network_t *network*)

Get information regarding the currently connected WLAN network. If we aren't connected to a WLAN access point, the SSID will not be filled in.

Parameters

in	*dev_name*	WLAN STA device name.
out	*network*	Caller-allocated struct that will contain the relevant information on return.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.11.5.4 cg_status_t cg_wlan_sta_get_network_list (const char *dev_name*, uint32_t *num_entries*, cg_wlan_network_t *networks*)

Get the list of saved access points. cg_wlan_network_t entries in the list will not contain valid channel numbers and signal quality values.

Parameters

in	*dev_name*	WLAN STA device name.
out	*num_entries*	When successful, the number of entries in *networks*.
out	*networks*	When successful, will be set to an array of cg_wlan_network_t, allocated by the callee. The caller will be responsible for freeing the array.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.11.5.5 cg_status_t cg_wlan_sta_remove_network (const char *dev_name*, const char *ssid*, cg_wlan_auth_t *auth*)

Remove a network from the list of saved access points.

Parameters

in	*dev_name*	WLAN STA device name.
in	*ssid*	SSID of the network.
in	*auth*	Authentication type of the network.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.11.5.6 cg_status_t cg_wlan_ap_set_ssid (const char *dev_name*, const char *ssid*, uint8_t *broadcast*)

Set the SSID for the WLAN access point.

Parameters

in	*dev_name*	WLAN AP device name.
in	*ssid*	The SSID to set for the access point.
in	*broadcast*	A flag to indicate if the SSID should be broadcast or not. Set to FALSE to disable broadcast. In this case the network will be hidden & not visible for clients. In order to connect a client would need to know the SSID upfront. Set to TRUE to enable broadcast.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.11.5.7 cg_status_t cg_wlan_ap_get_ssid (const char *dev_name*, char *ssid*, uint8_t *broadcast*)

Get the SSID for the WLAN access point.

Parameters

in	*dev_name*	WLAN AP device name.
out	*ssid*	SSID for the access point. Callee will set ssid to the SSID string. Caller will need to free.
out	*broadcast*	A flag to indicate if the SSID is broadcasted or not.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.11.5.8 cg_status_t cg_wlan_ap_set_channel (const char *dev_name*, uint16_t *channel*)

Set the WLAN channel for a WLAN access point interface. Changing the channel of a WLAN access point interface will change the channel for all access point interfaces on the same WLAN radio. If the WLAN station is enabled, changing the channel will not be possible since the access points have to follow the station's channel.

Parameters

in	*dev_name*	WLAN AP device name.
in	*channel*	The WLAN channel. If set to zero, automatic channel selection is enabled.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter, CG_STATUS_ERROR in case of an error or CG_STATUS_RESOURCE_BUSY when the WLAN station is enabled.

4.11.5.9 cg_status_t cg_wlan_ap_get_channel (const char *dev_name*, uint16_t *channel*)

Get the WLAN channel for a WLAN access point interface.

Parameters

in	*dev_name*	WLAN AP device name.
in	*channel*	WLAN channel to get. If channel is zero, automatic channel selection is enabled.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.11.5.10 cg_status_t cg_wlan_ap_set_auth_params (const char *dev_name*, cg_wlan_auth_params_t *auth*)

Set the authentication parameters for a WLAN access point interface. CG_WLAN_AUTH_WEP isn't a valid authentication type for access point.

Parameters

in	*dev_name*	WLAN AP device name.
in	*auth*	Authentication parameters.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.11.5.11 cg_status_t cg_wlan_ap_get_auth_params (const char *dev_name*, cg_wlan_auth_params_t *auth*)

Get the authentication params for a WLAN access point interface.

Parameters

in	*dev_name*	WLAN AP device name.
out	*auth*	Current authentication parameters of a WLAN access point interface.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.12 WLAN_SNIFFER

Data Structures

- struct cg_wlan_sniffer_params

Macros

- #define CG_MONITORING_FILE_PATH_MAX_LEN 64

Typedefs

- typedef struct cg_wlan_sniffer_params **cg_wlan_sniffer_params_t**

Functions

- cg_status_t cg_wlan_sniffer_start (uint32_t window_size, uint32_t channel, uint32_t max_file_size, uint32_t max_nbr_files, const char monitoring_file_path, uint32_t detailed_monitoring)
- cg_status_t cg_wlan_sniffer_stop (void)

4.12.1 Detailed Description

4.12.2 Macro Definition Documentation

4.12.2.1 #define CG_MONITORING_FILE_PATH_MAX_LEN 64

Maximum length of the string containing the file monitoring path

4.12.3 Function Documentation

4.12.3.1 cg_status_t cg_wlan_sniffer_start (uint32_t *window_size*, uint32_t *channel*, uint32_t *max_file_size*, uint32_t *max_nbr_files*, const char *monitoring_file_path*, uint32_t *detailed_monitoring*)

Initiate wlan sniffing. The function sets the supplied parameters and starts wlan logging probe requests for a time span given by the window size. The logging result is filtered and written to an output file. As soon the size of the output file reaches "max_file_size", the file is closed and a new file with a different name is created. The maximum number of files created is given by the parameter "max_nbr_files".

Parameters

in	*window_size,:*	value in seconds.
in	*channel,:*	wifi channel to log.
in	*max_file_size,:*	maximum file size in bytes.
in	*max_nbr_files,:*	maximum number of files created.
in	*monitoring_file_path,:*	file path for monitoring log files.
in	*detailed_monitoring,:*	0 simple monitoring , 1 detailed monitoring.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.12.3.2 cg_status_t cg_wlan_sniffer_stop (void)

stop wlan sniffing. The functions stops wlan logging and stops also wlan_snifferd daemon.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13 WWAN

Data Structures

- struct cg_wwan_network_t
  Structure defining a network.
- struct cg_wwan_network_list_t
  Structure for a list of networks.
- struct cg_wwan_reg_state_t
  Network registration state.
- struct cg_wwan_pin_state_t
  PIN code status of the SIM inserted in the device.
- struct cg_wwan_img_t
  Image name.
- struct cg_wwan_img_list_t
  Structure containing a list of available images.
- struct cg_wwan_diag_param_t
  Diagnostic parameter information.
- struct cg_wwan_network_settings_t
  Struct used to set/get the mobile network settings.
- struct cg_wwan_location_info_t
  Struct defining the location of the base transceiver station.

Macros

- #define CG_NETWORK_NAME_SIZE 64
- #define CG_MAX_IMAGE_NAME 64
- #define GOBI_DEVICE "usb0"
- #define **TELIT_DEVICE** "wwan0"
- #define **HPEX_DEVICE** "hso0"

Typedefs

- typedef void(cg_wwan_network_list_cb_t)(cg_status_t status, cg_wwan_network_list_t list, void context)
- typedef void(cg_wwan_reg_state_cb_t)(const char dev_name, cg_wwan_reg_state_t state, void context)

Enumerations

- enum cg_wwan_pin_type_t {
  CG_WWAN_PIN_UNKNOWN = 0, CG_WWAN_PIN_SIM_NOT_INSERTED, CG_WWAN_PIN_READY, CG_WWAN_PIN_SIM_PIN,
  CG_WWAN_PIN_SIM_PUK, CG_WWAN_PIN_SIM_FAILURE, CG_WWAN_PIN_LOCKED_PN, CG_WWAN_PIN_SIM_NOT_SUPPORTED }
- enum cg_wwan_system_t {
  CG_SYS_NONE = 0x1, **CG_SYS_GSM** = 0x2, **CG_SYS_GPRS** = 0x4, **CG_SYS_EDGE** = 0x8,
  CG_SYS_WCDMA = 0x10, **CG_SYS_HSDPA** = 0x20, **CG_SYS_HSUPA** = 0x40, **CG_SYS_HSPA** = 0x80,
  CG_SYS_HSDPAPLUS = 0x100, **CG_SYS_HSPAPLUS** = 0x200, **CG_SYS_LTE** = 0x400, **CG_SYS_CDMA** = 0x1000,
  CG_SYS_EVDO = 0x2000, **CG_SYS_EVDOREL0** = 0x4000, **CG_SYS_EVDOREVA** = 0x8000 }

Enumeration describing the possible mobile sytem technologies offered by the serving cell. Cells can indicate combinations of technologies , eg CG_SYS_WCDMA & CG_SYS_HSDPA & CG_SYS_HSUPA in general all represent 3G.

- enum cg_wwan_mode_t { CG_WWAN_MODE_CDMA = 0, CG_WWAN_MODE_UMTS }

  Indicate in what mode the WWAN device is. In case of CG_WWAN_TYPE_CDMA, you can use the CDMA-specific calls, but not the UMTS-specific calls. (Other way around for CG_WWAN_TYPE_UMTS)
- enum cg_wwan_creg_status_t { CG_WWAN_CREG_NOT_REG = 0, CG_WWAN_CREG_REG = 1, CG_WWAN_CREG_SEARCHING = 2, CG_WWAN_CREG_DENIED = 3, CG_WWAN_CREG_ROAMING = 5 }

  Indicate in what network registration status the WWAN device is.
- enum cg_wwan_act_state_t { CG_WWAN_ACT_STATUS_NOT_ACTIVATED = 0x00, CG_WWAN_ACT_STATUS_ACTIVATED = 0x01, CG_WWAN_ACT_STATUS_CONNECTION = 0x02, CG_WWAN_ACT_STATUS_CONNECTED = 0x03, CG_WWAN_ACT_STATUS_OTASP_AUTH = 0x04, CG_WWAN_ACT_STATUS_OTASP_NAM = 0x05, CG_WWAN_ACT_STATUS_OTASP_MDN = 0x06, CG_WWAN_ACT_STATUS_OTASP_IMSI = 0x07, CG_WWAN_ACT_STATUS_OTASP_PRL = 0x08, CG_WWAN_ACT_STATUS_OTASP_SPC = 0x09, CG_WWAN_ACT_STATUS_OTASP_COMMITTED = 0x0A }
- enum cg_wwan_sim_switch_mode_t { CG_SIM_SWITCH_MODE_AUTO, CG_SIM_SWITCH_MODE_SIM1, CG_SIM_SWITCH_MODE_SIM2 }
- enum cg_wwan_sim_switch_state_t { CG_SIM_SWITCH_STATE_SIM1 = 1, CG_SIM_SWITCH_STATE_SIM2 = 2 }
- enum cg_wwan_auth_type_t { CG_WWAN_AUTH_TYPE_NONE, CG_WWAN_AUTH_TYPE_PAP, CG_WWAN_AUTH_TYPE_CHAP, CG_WWAN_AUTH_TYPE_AUTO }

  The authentication type for the mobile connection.

Generic functions

- cg_status_t cg_wwan_get_active_primary_dev (char dev_name)
- cg_status_t cg_wwan_get_mode (const char dev_name, cg_wwan_mode_t mode)
- cg_status_t cg_wwan_signal_strength (const char dev_name, int8_t strength, int8_t ecio)
- cg_status_t cg_wwan_set_radio (const char dev_name, int enabled)
- cg_status_t cg_wwan_get_radio (const char dev_name, int enabled)
- cg_status_t cg_wwan_get_reg_state (const char dev_name, cg_wwan_reg_state_t state)
- cg_status_t cg_wwan_register_reg_state_notification (const char dev_name, cg_wwan_reg_state_cb_t cb, void context)
- cg_status_t cg_wwan_deregister_reg_state_notification (const char dev_name, cg_wwan_reg_state_cb_t cb, void context)
- cg_status_t cg_wwan_get_serial (const char dev_name, char serial)
- cg_status_t cg_wwan_get_diagnostics (const char dev_name, uint32_t num_params, cg_wwan_diag_param_t params)
- cg_status_t cg_wwan_set_network_settings (const char dev_name, cg_wwan_network_settings_t wwan_network_settings)
- cg_status_t cg_wwan_get_network_settings (const char dev_name, cg_wwan_network_settings_t wwan_network_settings)
- cg_status_t cg_wwan_get_phone_number (const char dev_name, char phone_number)

UMTS - specific functions

- cg_status_t cg_wwan_search_networks (const char dev_name, cg_wwan_network_list_cb_t cb, void context)
- cg_status_t cg_wwan_get_imei (const char dev_name, char imei)

- cg_status_t cg_wwan_get_imsi (const char dev_name, char imsi)
- cg_status_t cg_wwan_get_iccid (const char dev_name, char iccid)
- cg_status_t cg_wwan_pin_get_state (const char dev_name, cg_wwan_pin_state_t pin_state)
- cg_status_t cg_wwan_save_pin (const char dev_name, uint8_t enabled)
- cg_status_t cg_wwan_submit_pin (const char dev_name, const char pin)
- cg_status_t cg_wwan_change_pin (const char dev_name, const char old_pin_puk, const char new_pin)
- cg_status_t cg_wwan_pin_set_enabled (const char dev_name, uint8_t enabled, const char pin)
- cg_status_t cg_wwan_get_nw_selection_mode (const char dev_name, uint8_t automatic, cg_wwan_network_t network)
- cg_status_t cg_wwan_set_nw_selection_mode (const char dev_name, uint8_t automatic, cg_wwan_network_t network)

CDMA - specific functions

- cg_status_t cg_wwan_get_meid (const char dev_name, char meid)
- cg_status_t cg_wwan_get_activation_state (const char dev_name, cg_wwan_act_state_t state)
- cg_status_t cg_wwan_activate (const char dev_name, const char act_string)
- cg_status_t cg_wwan_get_prl_version (const char dev_name, uint16_t prl_version)
- cg_status_t cg_wwan_upload_prl (const char dev_name)
- cg_status_t cg_wwan_set_prl_network_update (const char dev_name, uint8_t enabled)
- cg_status_t cg_wwan_get_prl_network_update (const char dev_name, uint8_t enabled)

Device - specific functions.

- cg_status_t cg_wwan_gobi_get_active_image (const char dev_name, char image_name)
- cg_status_t cg_wwan_gobi_get_image_list (const char dev_name, cg_wwan_img_list_t image_list)
- cg_status_t cg_wwan_gobi_set_active_image (const char dev_name, const char image_name)
- cg_status_t cg_wwan_gobi_get_image_type (const char dev_name, const char image_name, char image_type)
- cg_status_t cg_wwan_modem_reboot ()
- cg_status_t cg_wwan_set_sim_switch (cg_wwan_sim_switch_mode_t mode)
- cg_status_t cg_wwan_get_sim_switch (cg_wwan_sim_switch_mode_t mode, cg_wwan_sim_switch_state_t current)
- cg_status_t cg_wwan_set_connection_hunting (const char dev_name, int enabled, int fallback_time)
- cg_status_t cg_wwan_get_location_info (const char dev_name, cg_wwan_location_info_t location_info)

4.13.1 Detailed Description

4.13.2 Macro Definition Documentation

4.13.2.1 #define CG_NETWORK_NAME_SIZE 64

Maximum size of network names

4.13.2.2 #define CG_MAX_IMAGE_NAME 64

Maximum size of image names

4.13.2.3 #define GOBI_DEVICE "usb0"

Definition of the modem device names

4.13.3 Typedef Documentation

4.13.3.1 typedef void(cg_wwan_network_list_cb_t)(cg_status_t status, cg_wwan_network_list_t list, void context)

Callback when requesting a network scan

Parameters

in	*status*	Status of network scan. CG_STATUS_OK or CG_STATUS_ERROR.
in	*list*	The network list. This list is allocated by the caller of the callback and will be freed when the callback returns.
in	*context*	The context which was passed on in cg_wwan_search_networks

4.13.3.2 typedef void(cg_wwan_reg_state_cb_t)(const char dev_name, cg_wwan_reg_state_t state, void context)

Callback for registration state updates

Parameters

in	*dev_name*	The device for which the registration state is updated
in	*state*	The updated state. Structure allocated by caller and will be free when the callback returns.
in	*context*	The context which was passed on in cg_wwan_register_reg_state-_notification

4.13.4 Enumeration Type Documentation

4.13.4.1 enum cg_wwan_pin_type_t

Different PIN states

Enumerator

CG_WWAN_PIN_UNKNOWN Unknown PIN state

CG_WWAN_PIN_SIM_NOT_INSERTED SIM is not inserted

CG_WWAN_PIN_READY PIN is ready & no further action required

CG_WWAN_PIN_SIM_PIN PIN code needs to be entered

CG_WWAN_PIN_SIM_PUK PUK code needs to be entered

CG_WWAN_PIN_SIM_FAILURE An error occurred while trying to initialize the SIM

CG_WWAN_PIN_LOCKED_PN The SIM is locked to a different network, an unlock code is required

CG_WWAN_PIN_SIM_NOT_SUPPORTED A SIM was inserted which is not supported

4.13.4.2 enum cg_wwan_mode_t

Indicate in what mode the WWAN device is. In case of CG_WWAN_TYPE_CDMA, you can use the CDMA-specific calls, but not the UMTS-specific calls. (Other way around for CG_WWAN_TYPE_UMTS)

Enumerator

CG_WWAN_MODE_CDMA CDMA mode

CG_WWAN_MODE_UMTS UMTS mode

4.13.4.3 enum cg_wwan_creg_status_t

Indicate in what network registration status the WWAN device is.

Enumerator

CG_WWAN_CREG_NOT_REG Not registered, WWAN device is not currently searching a new operator to register to.

CG_WWAN_CREG_REG Registered, home network

CG_WWAN_CREG_SEARCHING Not registered, but WWAN device is currently searching a new operator to register to.

CG_WWAN_CREG_DENIED Registration denied.

CG_WWAN_CREG_ROAMING Registered, roaming.

4.13.4.4 enum cg_wwan_act_state_t

The state of CDMA activation

Enumerator

CG_WWAN_ACT_STATUS_NOT_ACTIVATED Service is not activated

CG_WWAN_ACT_STATUS_ACTIVATED Service is activated

CG_WWAN_ACT_STATUS_CONNECTION Activation connecting – Network connection is in progress for automatic activation of service

CG_WWAN_ACT_STATUS_CONNECTED Activation connected – Network connection is connected for automatic activation of service

CG_WWAN_ACT_STATUS_OTASP_AUTH OTASP security is authenticated

CG_WWAN_ACT_STATUS_OTASP_NAM OTASP NAM is downloaded

CG_WWAN_ACT_STATUS_OTASP_MDN OTASP MDN is downloaded

CG_WWAN_ACT_STATUS_OTASP_IMSI OTASP IMSI downloaded

CG_WWAN_ACT_STATUS_OTASP_PRL OTASP PRL is downloaded

CG_WWAN_ACT_STATUS_OTASP_SPC OTASP SPC is downloaded

CG_WWAN_ACT_STATUS_OTASP_COMMITTED OTASP settings are committed

4.13.4.5 enum cg_wwan_sim_switch_mode_t

The sim mode used for sims in the Telematics expansion card

Enumerator

CG_SIM_SWITCH_MODE_AUTO The sim slot with the best functionality is chosen automatically

CG_SIM_SWITCH_MODE_SIM1 The sim in slot 1 will be used as active sim This is the sim closest to the PCB

CG_SIM_SWITCH_MODE_SIM2 The sim in slot 2 will be used as active sim This is the sim farthest from the PCB

4.13.4.6 enum cg_wwan_sim_switch_state_t

Enumerator

CG_SIM_SWITCH_STATE_SIM1 The sim in slot 1 is currently the active sim This is the sim closest to the PCB

CG_SIM_SWITCH_STATE_SIM2 The sim in slot 2 is currently the active sim This is the sim farthest from the PCB

4.13.4.7 enum cg_wwan_auth_type_t

The authentication type for the mobile connection.

Enumerator

CG_WWAN_AUTH_TYPE_NONE No authentication protocol is used

CG_WWAN_AUTH_TYPE_PAP Password Authentication Protocol is used

CG_WWAN_AUTH_TYPE_CHAP Challenge Handshake Authentication Protocol is used

CG_WWAN_AUTH_TYPE_AUTO Automatic authentication is used

4.13.5 Function Documentation

4.13.5.1 cg_status_t cg_wwan_get_active_primary_dev (char *dev_name*)

Get the name of the primary active device

Parameters

out	*dev_name*	will be filled in by the callee in case of success. Caller will need to free the string.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case an invalid parameter is provided or CG_STATUS_ERROR in case no primary WWAN device present.

4.13.5.2 cg_status_t cg_wwan_get_mode (const char *dev_name*, cg_wwan_mode_t *mode*)

Get the mode which the WWAN device is currently in

Parameters

in	*dev_name*	WWAN device name.
out	*mode*	Device mode

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.3 cg_status_t cg_wwan_signal_strength (const char *dev_name*, int8_t *strength*, int8_t *ecio*)

Get the current signal strength

Parameters

in	*dev_name*	WWAN device name
out	*strength*	Signal strength represented in dBm. Always negative, 0 when no signal present. The exact meaning depends on the WWAN technology of the selected cell: GSM/GPRS/EDGE (2G): RSSI (Received Signal Strength Indication) WCDMA/HSDPA/HSUPA/HSPA/HSDPA+/HSPA+ (3G): RSCP (Received Signal Code Power) LTE (4G): RSRP (Reference Signal Received Power) CDMA/EVDO: Signal strength of forward link pilot
out	*ecio*	ECIO represented in dB. Always negative, 0 when no signal present.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.4 cg_status_t cg_wwan_set_radio (const char *dev_name*, int *enabled*)

Enable or disable the WWAN radio. The transition to disabled can take a little time as the modem is detaching from the network. cg_wwan_get_radio can be polled to double check when the transition has completed.

Parameters

in	*dev_name*	WWAN device name
in	*enabled*	Set to TRUE to enable radio. FALSE to disable.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.5 cg_status_t cg_wwan_get_radio (const char *dev_name*, int *enabled*)

Get the current WWAN radio state.

Parameters

in	*dev_name*	WWAN device name
out	*enabled*	Returns TRUE if radio is enabled. FALSE if disabled.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.6 cg_status_t cg_wwan_get_reg_state (const char *dev_name*, cg_wwan_reg_state_t *state*)

Get the current registration state

Parameters

in	*dev_name*	WWAN device name
out	*state*	The registration state

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.7 cg_status_t cg_wwan_register_reg_state_notification (const char *dev_name*, cg_wwan_reg_state_cb_t *cb*, void *context*)

Register for registration state notification for a WWAN device.

Parameters

in	*dev_name*	WWAN device name
in	*cb*	Callback which is called when the registration state changes
in	*context*	Context parameter passed on to the callback when called.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.8 cg_status_t cg_wwan_deregister_reg_state_notification (const char *dev_name*, cg_wwan_reg_state_cb_t *cb*, void *context*)

Deregister for registration state notifications

Parameters

in	*dev_name*	WWAN device name
in	*cb*	Callback to deregister
in	*context*	Context parameter,97

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.9 cg_status_t cg_wwan_get_serial (const char *dev_name*, char *serial*)

Get the serial number of the WWAN device.

Parameters

in	*dev_name*	WWAN device name
out	*serial*	The device serial. serial will be filled in by the callee. Caller will need to free the string.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.10 cg_status_t cg_wwan_get_diagnostics (const char *dev_name*, uint32_t *num_params*, cg_wwan_diag_param_t *params*)

Get a list containing diagnostic information

Only supported by the GOBI_DEVICE

Parameters

in	*dev_name*	WWAN device name
out	*num_params*	When successful, will be set to the number of diagnostic parameters in the array
out	*params*	When successful, will be set to an array of cg_wwan_diag_param_t, allocated by the callee. The caller will be responsible for freeing the array only, not memory pointed to by members of cg_wwan_diag_param_t.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.11 cg_status_t cg_wwan_set_network_settings (const char *dev_name*, cg_wwan_network_settings_t *wwan_network_settings*)

Sets the mobile network settings.

Parameters

in	*dev_name*	WWAN device name
in	*wwan_network_settings*	Pointer to struct of type cg_wwan_network_settings_t including the APN, username, password, authentication method.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

Note

The sim must be present, otherwise you get a CG_STATUS_ERROR.

4.13.5.12 cg_status_t cg_wwan_get_network_settings (const char *dev_name*, cg_wwan_network_settings_t *wwan_network_settings*)

Gets the mobile network settings.

Parameters

in	*dev_name*	WWAN device name
out	*wwan_network_settings*	Callee will set wwan_network_settings to a cg_wwan_network_settings_t pointer. Caller needs to free this pointer.

Note

: The password is always an empty string. It is not readable.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

Note

The sim must be present, otherwise you get a CG_STATUS_ERROR.

4.13.5.13 cg_status_t cg_wwan_get_phone_number (const char *dev_name*, char *phone_number*)

Get the phone number from the SIM or MDN

Parameters

in	*dev_name*	The device name.
out	*phone_-number*	The phone number from the SIM or MDN. If no phone number is present, "unknown" is returned. Caller needs to free this pointer.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.14 cg_status_t cg_wwan_search_networks (const char *dev_name*, cg_wwan_network_list_cb_t *cb*, void *context*)

Search for UMTS networks

Parameters

in	*dev_name*	WWAN device name
in	*cb*	Callback called when network scan is completed.
in	*context*	This parameter will be passed on to the callback

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter, CG_STATUS_ERROR in case of an error or CG_STATUS_RESOURCE_UNAVAILABLE when the WWAN device is connected to a network.

4.13.5.15 cg_status_t cg_wwan_get_imei (const char *dev_name*, char *imei*)

Get the IMEI of the WWAN device

Parameters

in	*dev_name*	WWAN device name
out	*imei*	The device IMEI. imei will be filled in the callee. Called will need to free.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.16 cg_status_t cg_wwan_get_imsi (const char *dev_name*, char *imsi*)

Get the IMSI of the SIM inserted in the device

Parameters

in	*dev_name*	WWAN device name
out	*imsi*	IMSI of the device. imsi will be filled in the callee. Called will need to free.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.17 cg_status_t cg_wwan_get_iccid (const char *dev_name*, char *iccid*)

Get the ICCID of the SIM inserted in the device

Parameters

in	*dev_name*	WWAN device name
out	*iccid*	ICCID of the device. iccid will be filled in the callee. Called will need to free.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.18 cg_status_t cg_wwan_pin_get_state (const char *dev_name*, cg_wwan_pin_state_t *pin_state*)

Parameters

in	*dev_name*	WWAN device name
out	*pin_state*	PIN state of the device.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.19 cg_status_t cg_wwan_save_pin (const char *dev_name*, uint8_t *enabled*)

Save the pin (on next submission), or disable (and delete the saved pin)

Parameters

in	*dev_name*	WWAN device name
in	*enabled*	Set to TRUE if you want the device to save the PIN so it doesn't need to be entered every time. Set to FALSE if you want the device to always ask for the PIN code.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.20 cg_status_t cg_wwan_submit_pin (const char *dev_name*, const char *pin*)

Parameters

in	*dev_name*	WWAN device name
in	*pin*	Pin code

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.21 cg_status_t cg_wwan_change_pin (const char *dev_name*, const char *old_pin_puk*, const char *new_pin*)

Parameters

in	*dev_name*	WWAN device name
in	*old_pin_puk*	The old PIN code if you want to change your current PIN. If the PIN is blocked because a wrong PIN code was entered too many times, this field should contain a PUK code instead.
in	*new_pin*	The new PIN code

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.22 cg_status_t cg_wwan_pin_set_enabled (const char *dev_name*, uint8_t *enabled*, const char *pin*)

Parameters

in	*dev_name*	WWAN device name
in	*enabled*	Set to TRUE to enable the PIN code. FALSE to disable.
in	*pin*	The current PIN code.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.23 cg_status_t cg_wwan_get_nw_selection_mode (const char *dev_name*, uint8_t *automatic*, cg_wwan_network_t *network*)

Get the WWAN UMTS network selection mode

Parameters

in	*dev_name*	WWAN device name
out	*automatic*	Is set to 1 in case of automatic network selection, set to 0 in case of manual network selection.
out	*network*	The network on which the device is registered or tries to register. Note: The network settings preferred, roaming, forbidden and home are set to 0xFF if they are unknown.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.24 cg_status_t cg_wwan_set_nw_selection_mode (const char *dev_name*, uint8_t *automatic*, cg_wwan_network_t *network*)

Set the WWAN UMTS network selection mode

Parameters

in	*dev_name*	WWAN device name
in	*automatic*	Set to TRUE to set network selection mode to Automatic. In this case the device will search for a proper network himself and the network parameter will be ignored. If auto is set to FALSE, a network will need to be specified.
in	*network*	The network on which to manually register. This is usually an entry which was returned in the network scan (cg_wwan_search_networks).

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.25 cg_status_t cg_wwan_get_meid (const char *dev_name*, char *meid*)

Note

: TODO Add Sprint functions Get the CDMA MEID

Parameters

in	*dev_name*	WWAN device name
out	*meid*	The MEID. meid will be filled in the callee. Called will need to free.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.26 cg_status_t cg_wwan_get_activation_state (const char *dev_name*, cg_wwan_act_state_t *state*)

Get the current state of the CDMA activation

Parameters

in	*dev_name*	WWAN device name
out	*state*	The current state of the activation

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.27 cg_status_t cg_wwan_activate (const char *dev_name*, const char *act_string*)

Initiate CDMA service activation

Parameters

in	*dev_name*	WWAN device name
in	*act_string*	The activation string. Check your CDMA operator to see what it should be. Set to NULL in case of automatic activation.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.28 cg_status_t cg_wwan_get_prl_version (const char *dev_name*, uint16_t *prl_version*)

Get the PRL version

Parameters

in	*dev_name*	WWAN device name
out	*prl_version*	The PRL version.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.29 cg_status_t cg_wwan_upload_prl (const char *dev_name*)

Upload new PRL file. This file should be freely accessible and has to be located at /tmp/temp.prl. The user can remove this file afterwards.

Parameters

in	*dev_name*	WWAN device name

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.30 cg_status_t cg_wwan_set_prl_network_update (const char *dev_name*, uint8_t *enabled*)

Set whether the PRL file can be updated by the network. This only works for Sprint.

Parameters

in	*dev_name*	WWAN device name
in	*enabled*	Set to TRUE to enable the network to push PRL updates. Set to FALSE to disable this

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.31 cg_status_t cg_wwan_get_prl_network_update (const char *dev_name*, uint8_t *enabled*)

Check whether the PRL file can be updated by the network. This only works for Sprint.

Parameters

in	*dev_name*	WWAN device name
out	*enabled*	If enabled, the PRL file can be updated by the network.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.32 cg_status_t cg_wwan_gobi_get_active_image (const char *dev_name*, char *image_name*)

Get the name of the currently running image on the Gobi WWAN device

Parameters

in	*dev_name*	WWAN device name
out	*image_name*	Name of the image

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.33 cg_status_t cg_wwan_gobi_get_image_list (const char *dev_name*, cg_wwan_img_list_t *image_list*)

Get the list of images on the Gobi WWAN device

Parameters

in	*dev_name*	WWAN device name
out	*image_list*	List of images. image_list will be allocated and set by callee. Caller will need to free

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.34 cg_status_t cg_wwan_gobi_set_active_image (const char *dev_name*, const char *image_name*)

Set an image to be the active image on the Gobi device. The list of images can be queried through cg_wwan_gobi_get_image_list. WARNING: The modem will be reset after calling this function. It can take more than 30 seconds for the modem to be completely initialized after this.

Parameters

in	*dev_name*	WWAN device name
in	*image_name*	The name of the image to set active.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.35 cg_status_t cg_wwan_gobi_get_image_type (const char *dev_name*, const char *image_name*, char *image_type*)

Gets the type of a Gobi image given the image name

Parameters

in	*dev_name*	WWAN device name
in	*image_name*	Name of the image
out	*image_type*	Callee will set image_type to a string pointer. Caller needs to free this pointer.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.36 cg_status_t cg_wwan_modem_reboot ( )

Reboots the current modem. Tip: The function cg_device_register_notification can be used to receive an update when the modem is present again. For this the device_mask CG_DEVTYPE_NETWORK should be used. In the callback we should look for the event CG_DEV_EVT_ARRIVED of the device with name "usb0", "hso0" or "wwan0", depending on the type of modem.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter, CG_STATUS_BUSY if the modem is busy or CG_STATUS_ERROR in case of an error.

4.13.5.37 cg_status_t cg_wwan_set_sim_switch (cg_wwan_sim_switch_mode_t *mode*)

Sets the sim switch mode of the sim slots on the Telematics expansion card. To apply this sim mode, a modem reboot is necessary. (cg_wwan_modem_reboot). You need hardware version 1.3 or newer.

Parameters

in	*mode*	The sim switch mode to set.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.38 cg_status_t cg_wwan_get_sim_switch (cg_wwan_sim_switch_mode_t *mode*, cg_wwan_sim_switch_state_t *current*)

Gets the sim switch status of the sim slots on the Telematics expansion card. You need hardware version 1.3 or newer.

Parameters

out	*mode*	Gets the sim switch mode.
out	*current*	Gets the current sim switch state (which sim is currently active).

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.39 cg_status_t cg_wwan_set_connection_hunting (const char *dev_name*, int *enabled*, int *fallback_time*)

Sets connection hunting for the given device.

Parameters

in	*dev_name*	The device name.
in	*enabled*	Defines whether to enable/disable connection hunting.
in	*fallback_time*	Defines the fallback time of the connection hunting.

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error.

4.13.5.40 cg_status_t cg_wwan_get_location_info (const char *dev_name*, cg_wwan_location_info_t *location_info*)

Gets the location information of the base transceiver station.

Parameters

in	*dev_name*	WWAN device name.
out	*location_info*	The location information, some attributes can be zero if not supported

Returns

CG_STATUS_OK if successful, CG_STATUS_INVALID_PARAMETER in case of an invalid parameter or CG_STATUS_ERROR in case of an error or function not supported or modem not registered.

Chapter 5

Data Structure Documentation

5.1 cg_board_t Struct Reference

Description of a pluggable expansion board.

#include <cg_device.h>

Data Fields

- uint8_t slot_id
- char board_id [15]
- char hw_ver [5]

5.1.1 Detailed Description

Description of a pluggable expansion board.

5.1.2 Field Documentation

5.1.2.1 uint8_t cg_board_t::slot_id

The slot in which the board is inserted

5.1.2.2 char cg_board_t::board_id[15]

The ID of the board as read out from the EEPROM

5.1.2.3 char cg_board_t::hw_ver[5]

The hardware version of the expansion board

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

- libcg/cg_device.h

5.2 cg_conf_list Struct Reference

Data Fields

- int number
- const char values []

5.2.1 Field Documentation

5.2.1.1 int cg_conf_list::number

Number of values in this list

5.2.1.2 const char cg_conf_list::values[]

Array of pointers to the values in the list
The documentation for this struct was generated from the following file:

- libcg/cg_conf.h

5.3 cg_conf_list_t Struct Reference

Structure to return a list of values from the configuration.
#include <cg_conf.h>

5.3.1 Detailed Description

Structure to return a list of values from the configuration.
The documentation for this struct was generated from the following file:

- libcg/cg_conf.h

5.4 cg_config_t Struct Reference

Structure to reference the configuration context.
#include <cg_conf.h>

5.4.1 Detailed Description

Structure to reference the configuration context.
The documentation for this struct was generated from the following file:

- libcg/cg_conf.h

5.5 cg_date_time_t Struct Reference

Structure holding local or UTC time value.

#include <cg_general.h>

Data Fields

- struct tm **date_time**
- cg_time_format_t **time_format**

5.5.1 Detailed Description

Structure holding local or UTC time value.

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

- libcg/cg_general.h

5.6 cg_device_t Struct Reference

Basic properties of a hardware device present in the system.

#include <cg_device.h>

Data Fields

- char friendly_name [52]
- char device_name [100]
- cg_device_type_t device_type
- cg_location_t device_location
- cg_device_status_t status

5.6.1 Detailed Description

Basic properties of a hardware device present in the system.

5.6.2 Field Documentation

5.6.2.1 char cg_device_t::friendly_name[52]

A user-friendly name for the hardware device, e.g. "Gobi 0"

5.6.2.2 char cg_device_t::device_name[100]

The actual device path in order to access the device, e.g. "/dev/qmi0". For network devices this is the name as can be used with the cg_wwan and cg_net SDK functions.

5.6.2.3 cg_device_type_t cg_device_t::device_type

The type of device

5.6.2.4 cg_location_t cg_device_t::device_location

The location of the device

5.6.2.5 cg_device_status_t cg_device_t::status

Indicates various device status flags
The documentation for this struct was generated from the following file:

- libcg/cg_device.h

5.7 cg_dust_sensor_data_t Struct Reference

Output parameters of the dust sensor logger.
#include <cg_dust_sensor.h>

Data Fields

- unsigned char byte [63]
- uint32_t status

5.7.1 Detailed Description

Output parameters of the dust sensor logger.
Structure describing logging interface.

5.7.2 Field Documentation

5.7.2.1 unsigned char cg_dust_sensor_data_t::byte[63]

63 bytes raw logging data

5.7.2.2 uint32_t cg_dust_sensor_data_t::status

4 byte status number see_dust_sensor_status_t for meaning
The documentation for this struct was generated from the following file:

- libcg/cg_dust_sensor.h

5.8 cg_gps_certificate_t Struct Reference

root certificate to validate the SUPL server.

#include <cg_gps.h>

Data Fields

- uint32_t len
- uint8_t cert [CG_MAX_LEN_CERTIFICATE]

5.8.1 Detailed Description

root certificate to validate the SUPL server.

5.8.2 Field Documentation

5.8.2.1 uint32_t cg_gps_certificate_t::len

root certificate container

Note

each time new certificates are written the existing certificates will be deletedlength of each certificate

5.8.2.2 uint8_t cg_gps_certificate_t::cert[CG_MAX_LEN_CERTIFICATE]

certificate in DER format

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

- libcg/cg_gps.h

5.9 cg_gps_t Struct Reference

Structure containing all GPS properties.

#include <cg_gps.h>

Data Fields

- uint8_t status
- uint32_t reporting_interval

5.9.1 Detailed Description

Structure containing all GPS properties.

5.9.2 Field Documentation

5.9.2.1 uint8_t cg_gps_t::status

Status fields containing cg_gps_status_t flags

5.9.2.2 uint32_t cg_gps_t::reporting_interval

Reporting interval for GPS data

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

- libcg/cg_gps.h

5.10 cg_hal_serial_cfg_t Struct Reference

Generic configuration settings for a serial interface.

#include <cg_hal.h>

Data Fields

- cg_hal_serial_mode_t serial_mode
- union {
  cg_hal_serial_rs485_cfg_t rs485
  } param

5.10.1 Detailed Description

Generic configuration settings for a serial interface.

5.10.2 Field Documentation

5.10.2.1 cg_hal_serial_mode_t cg_hal_serial_cfg_t::serial_mode

Serial mode selection between RS485 or RS232

5.10.2.2 cg_hal_serial_rs485_cfg_t cg_hal_serial_cfg_t::rs485

RS232 has no additional parameters RS485 configuration parameters

5.10.2.3 union { ... } cg_hal_serial_cfg_t::param

Configuration specific to the selected serial mode (if required)

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

- libcg/cg_hal.h

5.11 cg_hal_serial_rs485_cfg_t Struct Reference

Serial RS485-specific configuration settings.

#include <cg_hal.h>

Data Fields

- uint32_t enable_termination

5.11.1 Detailed Description

Serial RS485-specific configuration settings.

5.11.2 Field Documentation

5.11.2.1 uint32_t cg_hal_serial_rs485_cfg_t::enable_termination

Enable the termination resistors on the RS485 wires

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

- libcg/cg_hal.h

5.12 cg_ip_addr_t Struct Reference

Structure describing a IP address.

#include <cg_net.h>

Data Fields

- uint8_t **ipv6**
- union {
  in_addr_t **ipv4**
  struct in6_addr **ipv6**
  } **ip**

5.12.1 Detailed Description

Structure describing a IP address.

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

- libcg/cg_net.h

5.13 cg_key_t Struct Reference

A single key of a key/value pair.

#include <cg_general.h>

Data Fields

- char **key** [CG_KVPAIR_MAX_KEY_LEN]

5.13.1 Detailed Description

A single key of a key/value pair.

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

- libcg/cg_general.h

5.14 cg_kvpair_t Struct Reference

Key/value pair.

#include <cg_general.h>

Data Fields

- char **ns** [CG_KVPAIR_MAX_NS_LEN]
- char **key** [CG_KVPAIR_MAX_KEY_LEN]
- int **value_len**
- uint8_t **value** [CG_KVPAIR_MAX_VALUE_LEN]

5.14.1 Detailed Description

Key/value pair.

Remarks

The 'com.option. ' namespace is reserved for use by Option NV.

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

- libcg/cg_general.h

5.15 cg_net_cfg_t Struct Reference

Generic configuration settings for a network interface.

#include <cg_net.h>

Data Fields

- cg_net_ip_config_t ip_config
- cg_net_zone_t zone
- union {
  cg_net_lan_cfg_t **lan**
  cg_net_wan_cfg_t **wan**
  } conf

5.15.1 Detailed Description

Generic configuration settings for a network interface.

5.15.2 Field Documentation

5.15.2.1 cg_net_ip_config_t cg_net_cfg_t::ip_config

The interface IP address

5.15.2.2 cg_net_zone_t cg_net_cfg_t::zone

The zone where this interface belongs to (WAN, LAN)

5.15.2.3 union { ... } cg_net_cfg_t::conf

Configuration specific to the zone to which this interface belongs
The documentation for this struct was generated from the following file:

- libcg/cg_net.h

5.16 cg_net_dhcp_config_t Struct Reference

Structure used to configure DHCP server functionality.
#include <cg_net.h>

Data Fields

- int enabled
- cg_ip_addr_t ip_start
- cg_ip_addr_t ip_end
- cg_ip_addr_t dns1
- cg_ip_addr_t dns2
- cg_net_leasetime_t leasetime

5.16.1 Detailed Description

Structure used to configure DHCP server functionality.

5.16.2 Field Documentation

5.16.2.1 int cg_net_dhcp_config_t::enabled

Set to TRUE to enable DHCP server. FALSE to disable

5.16.2.2 cg_ip_addr_t cg_net_dhcp_config_t::ip_start

Starting address for the DHCP IP range

5.16.2.3 cg_ip_addr_t cg_net_dhcp_config_t::ip_end

End address for the DHCP IP range

5.16.2.4 cg_ip_addr_t cg_net_dhcp_config_t::dns1

Fill this in to override the first DNS address which is passed on to DHCP clients

5.16.2.5 cg_ip_addr_t cg_net_dhcp_config_t::dns2

Fill this in to override the second DNS address which is passed on to DHCP clients

5.16.2.6 cg_net_leasetime_t cg_net_dhcp_config_t::leasetime

Fill this in to override the leasetime to DHCP clients
The documentation for this struct was generated from the following file:

- libcg/cg_net.h

5.17 cg_net_if_t Struct Reference

Data structure describing a network interface in the system.
#include <cg_net.h>

Data Fields

- char dev_name [CG_DEV_NAME_MAX]
- cg_net_if_status_t status
- cg_net_if_type_t type
- cg_net_cfg_t config

5.17.1 Detailed Description

Data structure describing a network interface in the system.

5.17.2 Field Documentation

5.17.2.1 char cg_net_if_t::dev_name[CG_DEV_NAME_MAX]

Name of the network interface, eg. eth0, uap0,...

5.17.2.2 cg_net_if_status_t cg_net_if_t::status

Interface status

5.17.2.3 cg_net_if_type_t cg_net_if_t::type

The type of network interface

5.17.2.4 cg_net_cfg_t cg_net_if_t::config

IP configuration for a WAN or LAN interface.
The documentation for this struct was generated from the following file:

- libcg/cg_net.h

5.18 cg_net_ip_config_t Struct Reference

Structure describing IP configuration of an interface.
#include <cg_net.h>

Data Fields

- cg_ip_addr_t ip_addr
- cg_ip_addr_t netmask
- int mtu

5.18.1 Detailed Description

Structure describing IP configuration of an interface.

5.18.2 Field Documentation

5.18.2.1 cg_ip_addr_t cg_net_ip_config_t::ip_addr

IPv4 interface address

5.18.2.2 cg_ip_addr_t cg_net_ip_config_t::netmask

IPv4 netmask

5.18.2.3 int cg_net_ip_config_t::mtu

MTU of the interface

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

- libcg/cg_net.h

5.19 cg_net_lan_cfg_t Struct Reference

LAN-specific configuration settings for a network interface.

#include <cg_net.h>

Data Fields

- cg_net_dhcp_config_t dhcp_config
- char bridge_dev_name [CG_DEV_NAME_MAX]

5.19.1 Detailed Description

LAN-specific configuration settings for a network interface.

5.19.2 Field Documentation

5.19.2.1 cg_net_dhcp_config_t cg_net_lan_cfg_t::dhcp_config

DHCP server configuration

5.19.2.2 char cg_net_lan_cfg_t::bridge_dev_name[CG_DEV_NAME_MAX]

Reference to bridge interface. If this parameter is non-NULL, it means the interface is part of a bridge. This voids all previous IP config settings because the bridge IP configuration will determine the interface configuration.Name of the bridge network interface

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

- libcg/cg_net.h

5.20 cg_net_leasetime_t Struct Reference

Structure describing a leasetime.

#include <cg_net.h>

Data Fields

- unsigned int amount
- cg_net_lease_unit_t unit

5.20.1 Detailed Description

Structure describing a leasetime.

5.20.2 Field Documentation

5.20.2.1 unsigned int cg_net_leasetime_t::amount

Fill this in to override the leasetime amount which is passed on to DHCP clients

5.20.2.2 cg_net_lease_unit_t cg_net_leasetime_t::unit

Fill this in to override the leasetime unit which is passed on to DHCP clients
The documentation for this struct was generated from the following file:

- libcg/cg_net.h

5.21 cg_net_stats_t Struct Reference

Structure that contains the data usage statistics for an interface.
#include <cg_net.h>

Data Fields

- uint64_t rx_packets
- uint64_t rx_bytes
- uint64_t tx_packets
- uint64_t tx_bytes

5.21.1 Detailed Description

Structure that contains the data usage statistics for an interface.

5.21.2 Field Documentation

5.21.2.1 uint64_t cg_net_stats_t::rx_packets

The amount of received packets.

5.21.2.2 uint64_t cg_net_stats_t::rx_bytes

The amount of received bytes.

5.21.2.3 uint64_t cg_net_stats_t::tx_packets

The amount of transmitted packets.

5.21.2.4 uint64_t cg_net_stats_t::tx_bytes

The amount of transmitted bytes.

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

- libcg/cg_net.h

5.22 cg_net_wan_cfg_t Struct Reference

WAN-specific configuration settings for a network interface.

#include <cg_net.h>

Data Fields

- cg_net_mode_t mode
- uint32_t data_timeout
- cg_net_ip_config_type_t ip_config_type
- cg_ip_addr_t gw
- cg_ip_addr_t dns1
- cg_ip_addr_t dns2

5.22.1 Detailed Description

WAN-specific configuration settings for a network interface.

5.22.2 Field Documentation

5.22.2.1 cg_net_mode_t cg_net_wan_cfg_t::mode

Connection mode (always-on, on-demand,...) In order for 'on-demand' to work, 'automatic' connecting must be enabled by calling cg_net_connect() first

5.22.2.2 uint32_t cg_net_wan_cfg_t::data_timeout

If the interface is set to on-demand mode (see mode), this value indicates the interval (in minutes) within which data must be received in order to keep the interface connected. If no data was received within this interval, the interface will disconnect and wait-for-data again.

5.22.2.3 cg_net_ip_config_type_t cg_net_wan_cfg_t::ip_config_type

The type of IP address configuration

5.22.2.4 cg_ip_addr_t cg_net_wan_cfg_t::gw

IPv4 gateway

5.22.2.5 cg_ip_addr_t cg_net_wan_cfg_t::dns1

First DNS server

5.22.2.6 cg_ip_addr_t cg_net_wan_cfg_t::dns2

Second DNS server

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

- libcg/cg_net.h

5.23 cg_net_watchdog_settings_t Struct Reference

Settings structure for the connection watchdog.

#include <cg_net.h>

Data Fields

- int enabled
- int use_ping
- uint32_t interval
- cg_net_watchdog_action_t action
- uint32_t num_addresses
- char addresses [CG_MAX_PING_ADDRESSES][CG_MAX_ADDRESS_LENGTH]

5.23.1 Detailed Description

Settings structure for the connection watchdog.

5.23.2 Field Documentation

5.23.2.1 int cg_net_watchdog_settings_t::enabled

Set to TRUE to enable connection watchdog. FALSE to disable

5.23.2.2 int cg_net_watchdog_settings_t::use_ping

The device will perform DNS lookups by default on addresses, unless an IP address was specified. By setting this variable to TRUE, the device will perform an additional ping.

5.23.2.3 uint32_t cg_net_watchdog_settings_t::interval

The frequency in seconds with which to perform the connection check. Eg. if an interval of 60 seconds was specified, the connection test will be started every 60 seconds. TODO: specify minimum value to cover maximum execution time of test.

5.23.2.4 cg_net_watchdog_action_t cg_net_watchdog_settings_t::action

The action that the watchdog should undertake when it determines that there is no connectivity anymore

5.23.2.5 uint32_t cg_net_watchdog_settings_t::num_addresses

The number of valid addresses in addresses

5.23.2.6 char cg_net_watchdog_settings_t::addresses[CG_MAX_PING_ADDRESSES][CG_MAX_ADDRESS_LENGTH]

Array of IP/host addresses

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

- libcg/cg_net.h

5.24 cg_slot_list_t Struct Reference

List of present slots.

#include <cg_general.h>

Data Fields

- uint16_t num_slots
- cg_slot_t slots [0]

5.24.1 Detailed Description

List of present slots.

5.24.2 Field Documentation

5.24.2.1 uint16_t cg_slot_list_t::num_slots

Number of slots in *slots*

5.24.2.2 cg_slot_t cg_slot_list_t::slots[0]

Array of slots

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

- libcg/cg_general.h

5.25 cg_slot_t Struct Reference

Device firmware slot description.

#include <cg_general.h>

Data Fields

- cg_slot_type_t type
- uint8_t active
- uint32_t size
- char version [CG_MAX_SLOT_INFO_LEN]
- char uid [CG_MAX_SLOT_UID_LEN+1]

5.25.1 Detailed Description

Device firmware slot description.

5.25.2 Field Documentation

5.25.2.1 cg_slot_type_t cg_slot_t::type

Type of slot

5.25.2.2 uint8_t cg_slot_t::active

Flag to indicate if this slot is currently active or not

5.25.2.3 uint32_t cg_slot_t::size

Size of the slot (in kilobytes)

5.25.2.4 char cg_slot_t::version[CG_MAX_SLOT_INFO_LEN]

Version of the current image in that slot

5.25.2.5 char cg_slot_t::uid[CG_MAX_SLOT_UID_LEN+1]

Unique ID of the slot

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

- libcg/cg_general.h

5.26 cg_timezone_t Struct Reference

Time zone splitted up in its components. https://www.gnu.org/software/libc/manual/html_node/TZ-Variable.html.

#include <cg_general.h>

Data Fields

- int **utc_offset_h**
- int **utc_offset_m**
- int **dst_start_m**
- int **dst_start_n**
- int **dst_start_d**
- int **dst_end_m**
- int **dst_end_n**
- int **dst_end_d**
- char **desc** [256]

5.26.1 Detailed Description

Time zone splitted up in its components. https://www.gnu.org/software/libc/manual/html_node/TZ-Variable.html.

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

- libcg/cg_general.h

5.27 cg_wlan_auth_params_t Struct Reference

Authentication parameters for a WLAN client/access point.

#include <cg_wlan.h>

Data Fields

- cg_wlan_auth_t type
- char password [CG_PASSWORD_MAX_LENGTH+1]

5.27.1 Detailed Description

Authentication parameters for a WLAN client/access point.

5.27.2 Field Documentation

5.27.2.1 cg_wlan_auth_t cg_wlan_auth_params_t::type

Type of authentication

5.27.2.2 char cg_wlan_auth_params_t::password[CG_PASSWORD_MAX_LENGTH+1]

Password used for PSK authentication

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

- libcg/cg_wlan.h

5.28 cg_wlan_network_t Struct Reference

Description of a WLAN network.

#include <cg_wlan.h>

Data Fields

- char ssid [CG_SSID_MAX_LENGTH+1]
- cg_wlan_auth_t auth_type
- uint16_t channel
- uint16_t signal_strength

5.28.1 Detailed Description

Description of a WLAN network.

5.28.2 Field Documentation

5.28.2.1 char cg_wlan_network_t::ssid[CG_SSID_MAX_LENGTH+1]

The SSID of the WLAN network

5.28.2.2 cg_wlan_auth_t cg_wlan_network_t::auth_type

Authentication type of the WLAN network

5.28.2.3 uint16_t cg_wlan_network_t::channel

Configured channel of the WLAN network. If set to zero, automatic channel selection is enabled

5.28.2.4 uint16_t cg_wlan_network_t::signal_strength

Signal strength of the WLAN network

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

- libcg/cg_wlan.h

5.29 cg_wlan_sniffer_params Struct Reference

Data Fields

- uint32_t window_size
- uint32_t channel
- uint32_t max_file_size
- uint32_t max_nbr_files
- char monitoring_file_path [CG_MONITORING_FILE_PATH_MAX_LEN+1]
- uint32_t detailed_monitoring

5.29.1 Field Documentation

5.29.1.1 uint32_t cg_wlan_sniffer_params::window_size

window size in seconds

5.29.1.2 uint32_t cg_wlan_sniffer_params::channel

wifi channel number

5.29.1.3 uint32_t cg_wlan_sniffer_params::max_file_size

max file size in bytes

5.29.1.4 uint32_t cg_wlan_sniffer_params::max_nbr_files

maximum number of files created

5.29.1.5 char cg_wlan_sniffer_params::monitoring_file_path[CG_MONITORING_FILE_PATH_MAX_-LEN+1]

monitoring file path

5.29.1.6 uint32_t cg_wlan_sniffer_params::detailed_monitoring

detailed or simple monitoring
The documentation for this struct was generated from the following file:

- libcg/cg_wlan_sniffer.h

5.30 cg_wwan_diag_param_t Struct Reference

Diagnostic parameter information.
#include <cg_wwan.h>

Data Fields

- const char name
- const char value

5.30.1 Detailed Description

Diagnostic parameter information.

5.30.2 Field Documentation

5.30.2.1 const char cg_wwan_diag_param_t::name

Parameter name

5.30.2.2 const char cg_wwan_diag_param_t::value

Parameter value

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

- libcg/cg_wwan.h

5.31 cg_wwan_img_list_t Struct Reference

Structure containing a list of available images.

#include <cg_wwan.h>

Data Fields

- uint32_t entries
- uint32_t current
- cg_wwan_img_t images []

5.31.1 Detailed Description

Structure containing a list of available images.

5.31.2 Field Documentation

5.31.2.1 uint32_t cg_wwan_img_list_t::entries

Number of entries

5.31.2.2 uint32_t cg_wwan_img_list_t::current

Current active entry

5.31.2.3 cg_wwan_img_t cg_wwan_img_list_t::images[]

Array of images names

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

- libcg/cg_wwan.h

5.32 cg_wwan_img_t Struct Reference

Image name.

#include <cg_wwan.h>

Data Fields

- char name [CG_MAX_IMAGE_NAME]

5.32.1 Detailed Description

Image name.

5.32.2 Field Documentation

5.32.2.1 char cg_wwan_img_t::name[CG_MAX_IMAGE_NAME]

Name of the image

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

- libcg/cg_wwan.h

5.33 cg_wwan_location_info_t Struct Reference

Struct defining the location of the base transceiver station.

#include <cg_wwan.h>

Data Fields

- uint32_t lac
- uint32_t tac
- uint32_t cell_id

5.33.1 Detailed Description

Struct defining the location of the base transceiver station.

5.33.2 Field Documentation

5.33.2.1 uint32_t cg_wwan_location_info_t::lac

Location Area Code is a unique number to identify location area of a base station (for UMTS and GSM networks) (equals zero if not supplied).

5.33.2.2 uint32_t cg_wwan_location_info_t::tac

Tracking Area Code identifies a tracking area within a particular network (for LTE networks) (equals zero if not supplied).

5.33.2.3 uint32_t cg_wwan_location_info_t::cell_id

A unique number to identify base transceiver station (BTS) or sector of a BTS (equals zero if not supplied).

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

- libcg/cg_wwan.h

5.34 cg_wwan_network_list_t Struct Reference

Structure for a list of networks.

#include <cg_wwan.h>

Data Fields

- uint32_t entries
- cg_wwan_network_t networks [0]

5.34.1 Detailed Description

Structure for a list of networks.

5.34.2 Field Documentation

5.34.2.1 uint32_t cg_wwan_network_list_t::entries

The number of entries in networks

5.34.2.2 cg_wwan_network_t cg_wwan_network_list_t::networks[0]

The actual networks

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

- libcg/cg_wwan.h

5.35 cg_wwan_network_settings_t Struct Reference

Struct used to set/get the mobile network settings.

#include <cg_wwan.h>

Data Fields

- char apn [100+1]
- char username [50+1]
- char password [50+1]
- cg_wwan_auth_type_t auth_type

5.35.1 Detailed Description

Struct used to set/get the mobile network settings.

5.35.2 Field Documentation

5.35.2.1 char cg_wwan_network_settings_t::apn[100+1]

Access Point Name

5.35.2.2 char cg_wwan_network_settings_t::username[50+1]

The username

5.35.2.3 char cg_wwan_network_settings_t::password[50+1]

The password

5.35.2.4 cg_wwan_auth_type_t cg_wwan_network_settings_t::auth_type

The authentication type

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

- libcg/cg_wwan.h

5.36 cg_wwan_network_t Struct Reference

Structure defining a network.

#include <cg_wwan.h>

Data Fields

- char name [CG_NETWORK_NAME_SIZE]
- uint16_t system_type
- uint32_t id
- uint8_t preferred
- uint8_t roaming
- uint8_t forbidden
- uint8_t home

5.36.1 Detailed Description

Structure defining a network.

5.36.2 Field Documentation

5.36.2.1 char cg_wwan_network_t::name[CG_NETWORK_NAME_SIZE]

Name of the network

5.36.2.2 uint16_t cg_wwan_network_t::system_type

Type of the system. Mask of cg_wwan_system_t

5.36.2.3 uint32_t cg_wwan_network_t::id

Network ID: MCC / MNC concatenation

5.36.2.4 uint8_t cg_wwan_network_t::preferred

Byte to indicate if this is a preferred network.

5.36.2.5 uint8_t cg_wwan_network_t::roaming

Byte to indicate that the device can roam on that network

5.36.2.6 uint8_t cg_wwan_network_t::forbidden

Byte to indicate that it is forbidden to roam on that network

5.36.2.7 uint8_t cg_wwan_network_t::home

byte to indicate that it is a home network
The documentation for this struct was generated from the following file:

- libcg/cg_wwan.h

5.37 cg_wwan_pin_state_t Struct Reference

PIN code status of the SIM inserted in the device.
#include <cg_wwan.h>

Data Fields

- cg_wwan_pin_type_t pin_type
- uint32_t pin_retries

- uint32_t puk_retries
- uint8_t pin_enabled

5.37.1 Detailed Description

PIN code status of the SIM inserted in the device.

5.37.2 Field Documentation

5.37.2.1 cg_wwan_pin_type_t cg_wwan_pin_state_t::pin_type

The current PIN state

5.37.2.2 uint32_t cg_wwan_pin_state_t::pin_retries

The number of PIN retries remaining on the SIM

5.37.2.3 uint32_t cg_wwan_pin_state_t::puk_retries

The number of PUK retries remaining on the SIM

5.37.2.4 uint8_t cg_wwan_pin_state_t::pin_enabled

Set to 1 if PIN code is enabled, 0 is PIN code is disabled
The documentation for this struct was generated from the following file:

- libcg/cg_wwan.h

5.38 cg_wwan_reg_state_t Struct Reference

Network registration state.
#include <cg_wwan.h>

Data Fields

- cg_wwan_creg_status_t cs_state
- cg_wwan_creg_status_t ps_state
- cg_wwan_system_t system_type
- char network_name [CG_NETWORK_NAME_SIZE]

5.38.1 Detailed Description

Network registration state.

5.38.2 Field Documentation

5.38.2.1 cg_wwan_creg_status_t cg_wwan_reg_state_t::cs_state

Circuit-switched registration state

5.38.2.2 cg_wwan_creg_status_t cg_wwan_reg_state_t::ps_state

Packet-switched registration state

5.38.2.3 cg_wwan_system_t cg_wwan_reg_state_t::system_type

Type of system on which the device is registered.

5.38.2.4 char cg_wwan_reg_state_t::network_name[CG_NETWORK_NAME_SIZE]

The name of the network on which we're registered
The documentation for this struct was generated from the following file:

- libcg/cg_wwan.h

Chapter 6

File Documentation

6.1 libcg/cg_conf.h File Reference

Header file containing definitions for functions related to setting/getting configuration items.

#include <libcg/cgate.h>

Data Structures

- struct cg_conf_list

Typedefs

- typedef struct cg_config **cg_config_t**
- typedef struct cg_conf_list **cg_conf_list_t**

Functions

- cg_status_t cg_conf_open (const char file, int state, cg_config_t ctx)
- cg_status_t cg_conf_close (cg_config_t ctx)
- cg_status_t cg_conf_revert (cg_config_t ctx, const char section, const char option)
- cg_status_t cg_conf_set (cg_config_t ctx, const char section, const char option, const char value)
- cg_status_t cg_conf_get (cg_config_t ctx, const char section, const char option, const char value)
- cg_status_t cg_conf_add_section (cg_config_t ctx, const char name)
- cg_status_t cg_conf_del (cg_config_t ctx, const char section, const char option)
- cg_status_t cg_conf_get_sections (cg_config_t ctx, cg_conf_list_t sections)
- cg_status_t cg_conf_get_options (cg_config_t ctx, const char section, cg_conf_list_t options)

6.1.1 Detailed Description

Header file containing definitions for functions related to setting/getting configuration items. This part of the API can be used to manage configuration items. All configuration items are divided in to files. A file is divided in to sections. Each section has a name and a type. Each section can contain multiple options. Options are key-value pairs.

These functions return CG_STATUS_INVALID_PARAMETER if a provided parameter is not valid (e.g. NULL pointer where only valid pointers are allowed) or if a configuration item does not exist and should exist to be able to perform the operation (e.g. you want to get the option 'enabled' in the section 'general', but the section 'general' does not exist). If a function does not specify that a parameter is optional, you should assume the parameter is mandatory. Names for section or options are limited to alphanumeric and '_'. Values are not restricted to any characters. CG_STATUS_ERROR is returned when an unexpected situation arises (e.g. failure to open a file).

6.2 libcg/cg_device.h File Reference

Header file containing definition of all device management related function calls.

#include <libcg/cgate.h>

Data Structures

- struct cg_device_t
  Basic properties of a hardware device present in the system.
- struct cg_board_t
  Description of a pluggable expansion board.

Typedefs

- typedef void(device_notification_cb_t)(cg_device_t device, cg_device_event_t evt, void context)
- typedef cg_status_t(device_claim_cb_t)(cg_device_t device, void context)

Enumerations

- enum cg_device_status_t { CG_DEVICE_PRESENT = 0x1, CG_DEVICE_CLAIMED = 0x2 }
- enum cg_device_type_t {
  CG_DEVTYPE_NETWORK = 0x1, CG_DEVTYPE_SERIAL = 0x2, CG_DEVTYPE_DAC = 0x4, CG_DEVTYPE_GPIO = 0x8,
  CG_DEVTYPE_ADC = 0x10 }
- enum cg_location_t {
  CG_LOC_UNKNOWN = 0x0, CG_LOC_SLOT1 = 0x1, CG_LOC_SLOT2 = 0x2, CG_LOC_MAIN = 0x4,
  CG_LOC_ALL = 0x7 }
- enum cg_device_event_t { CG_DEV_EVT_ARRIVED, CG_DEV_EVT_REMOVED, CG_DEV_EVT_CLAIMED, CG_DEV_EVT_RELEASED }

Functions

- cg_status_t cg_device_list (uint32_t device_mask, uint32_t num_devices, cg_device_t devices)
- cg_status_t cg_device_list_by_location (uint32_t location_mask, uint32_t device_mask, uint32_t num_devices, cg_device_t devices)
- cg_status_t cg_device_claim (cg_device_t device, device_claim_cb_t cb, void context)
- cg_status_t cg_device_release (cg_device_t device)
- cg_status_t cg_device_register_notification (uint32_t device_mask, device_notification_cb_t cb, void context)

- cg_status_t cg_device_deregister_notification (void)
- cg_status_t cg_device_board_list (uint8_t num_boards, cg_board_t boards)
- cg_status_t cg_device_board_device_list (cg_board_t board, uint32_t num_devices, cg_device_t devices)

6.2.1 Detailed Description

Header file containing definition of all device management related function calls.

6.3 libcg/cg_dust_sensor.h File Reference

Header file containing all definitions/functions related to dust sensor logger.

#include <libcg/cgate.h>

Data Structures

- struct cg_dust_sensor_data_t

  Output parameters of the dust sensor logger.

Typedefs

- typedef void(cg_dust_sensor_data_cb_t)(const cg_dust_sensor_data_t data, void context)

Enumerations

- enum cg_dust_sensor_power_t { CG_DUST_SENSOR_POWER_OFF = 0, CG_DUST_SENSOR_POWER_FAN_ON = 1, CG_DUST_SENSOR_POWER_LASER_ON = 2, CG_DUST_SENSOR_POWER_MAX = 3 }
- enum cg_dust_sensor_status_t { CG_DUST_SENSOR_VALID_DATA, CG_DUST_SENSOR_RW_ERROR, CG_DUST_SENSOR_DISCONNECT, CG_DUST_SENSOR_CONNECT, CG_DUST_SENSOR_NOT_AVAILABLE, CG_DUST_SENSOR_THREAD_CREATE_ERROR, CG_DUST_SENSOR_INVALID_PARAMETER, CG_DUST_SENSOR_NOT_AWAKE, CG_DUST_SENSOR_SPI_ERROR }

Functions

- cg_status_t cg_dust_sensor_start (uint32_t time_interval)
- cg_status_t cg_dust_sensor_stop (void)
- cg_status_t cg_dust_sensor_set_power (uint32_t power)
- cg_status_t cg_dust_sensor_register_data_callback (cg_dust_sensor_data_cb_t cb, void context)
- cg_status_t cg_dust_sensor_deregister_data_callback (cg_dust_sensor_data_cb_t cb, void context)

6.3.1 Detailed Description

Header file containing all definitions/functions related to dust sensor logger.

6.4 libcg/cg_general.h File Reference

Header file containing definitions for miscellaneous functions.

#include <time.h>
#include <libcg/cgate.h>
#include <sys/time.h>

Data Structures

- struct cg_slot_t
  Device firmware slot description.
- struct cg_slot_list_t
  List of present slots.
- struct cg_kvpair_t
  Key/value pair.
- struct cg_key_t
  A single key of a key/value pair.
- struct cg_timezone_t
  Time zone splitted up in its components. https://www.gnu.org/software/libc/manual/html_node/-TZ-Variable.html.
- struct cg_date_time_t
  Structure holding local or UTC time value.

Macros

- #define CG_SDK_API_LEVEL 01
- #define CG_MAX_SLOT_INFO_LEN 32
- #define **CG_MAX_SLOT_UID_LEN** 64
- #define **CG_KVPAIR_MAX_NS_LEN** 64
- #define **CG_KVPAIR_MAX_KEY_LEN** 64
- #define **CG_KVPAIR_MAX_VALUE_LEN** 4096

Typedefs

- typedef void(cg_prov_kvpair_update_cb_t)(cg_kvpair_t update, void context)
- typedef void(cg_system_sync_ntp_cb_t)(cg_system_sync_ntp_state_t state, void context)

Enumerations

- enum cg_prov_status_t {
  CG_PROV_UNPROVISIONED, CG_PROV_CHECKING, CG_PROV_FLASHING, CG_PROV_ERRO-R,
  CG_PROV_OK, CG_PROV_UNREGISTERED }
  Provisioning status.

- enum cg_slot_type_t { CG_SLOT_BOOTLOADER, CG_SLOT_BASE_IMAGE, CG_SLOT_BASE_CONFIG, CG_SLOT_CUSTOMER, CG_SLOT_CUSTOMER_CONFIG, CG_SLOT_CUSTOMER_KEY }

  Different firmware slots.
- enum cg_dev_feature_t { CG_DEV_ROOT_ACCESS, CG_DEV_MINIVAN_DEBUG }

  Various development features.
- enum **cg_log_level_t** { **CG_LOG_DEBUG** = 0x1, **CG_LOG_INFO** = 0x2, **CG_LOG_WARN** = 0x4, **CG_LOG_ERR** = 0x8 }
- enum cg_system_boot_condition_t { CG_SYSTEM_ARM_IGNITION_SENSE = 0x1, CG_SYSTEM_TIMED_WAKEUP = 0x2 }

  System boot condition.
- enum **cg_time_format_t** { **CG_LOCAL_TIME**, **CG_UTC_TIME** }
- enum **cg_system_sync_ntp_state_t** { **CG_NTP_SYNC_SUCCESS**, **CG_NTP_SYNC_FAILURE**, **CG_NTP_SYNC_NO_SERVER** }

Functions

- cg_status_t cg_init (const char name)
- cg_status_t cg_deinit (void)
- cg_status_t cg_get_api_level (int32_t level)
- cg_status_t cg_get_serial_number (char serial_number)
- cg_status_t cg_prov_get_status (cg_prov_status_t status)
- cg_status_t cg_prov_check_update (uint8_t force)
- cg_status_t cg_prov_set_automatic (uint32_t onoff)
- cg_status_t cg_prov_kvpair_get (cg_kvpair_t request_in, cg_kvpair_t request_out)
- cg_status_t cg_prov_kvpair_set (cg_kvpair_t request)
- cg_status_t cg_prov_kvpair_del (cg_kvpair_t request)
- cg_status_t cg_prov_kvpair_list (char ns, int n_keys, cg_key_t keys)
- cg_status_t cg_prov_kvpair_register_update_callback (cg_prov_kvpair_update_cb_t cb, void context, const char ns)
- cg_status_t cg_prov_kvpair_deregister_update_callback (cg_prov_kvpair_update_cb_t cb, void context, const char ns)
- cg_status_t cg_get_slot_list (cg_slot_list_t slot)
- cg_status_t cg_reset_system (uint8_t fact_restore, const char reason)
- cg_status_t cg_reset_peripherals (void)
- cg_status_t cg_system_log (uint16_t log_level, const char format,...) CG_GNUC_PRINTF(2
- cg_status_t cg_status_t cg_set_dev_mode (uint8_t enable, cg_dev_feature_t feature, void options)
- cg_status_t cg_system_power_down (const cg_system_boot_condition_t system_boot_condition)
- cg_status_t cg_system_set_timed_wakeup (uint32_t seconds)
- cg_status_t cg_system_set_date_time (cg_date_time_t date_time)
- cg_status_t cg_system_get_date_time (cg_time_format_t time_format, cg_date_time_t date_time)
- cg_status_t cg_system_set_timezone (cg_timezone_t timezone)
- cg_status_t cg_system_set_ntp_server (char ntp_server_url)
- cg_status_t cg_system_get_ntp_server (char ntp_server_url)
- cg_status_t cg_system_sync_ntp (void)
- cg_status_t cg_system_sync_ntp_register_callback (cg_system_sync_ntp_cb_t cb, void context)
- cg_status_t cg_system_sync_ntp_deregister_callback (cg_system_sync_ntp_cb_t cb, void context)

6.4.1 Detailed Description

Header file containing definitions for miscellaneous functions.

6.5 libcg/cg_gps.h File Reference

Header file containing definitions for all GPS-related functionality.

#include <libcg/cgate.h>
#include <libcg/cg_net.h>

Data Structures

- struct cg_gps_t
  Structure containing all GPS properties.
- struct cg_gps_certificate_t
  root certificate to validate the SUPL server.

Macros

- #define CG_MAX_CERTIFICATES 5
- #define CG_MAX_LEN_CERTIFICATE 2000

Typedefs

- typedef void(cg_gps_nmea_cb_t)(const char nmea_data, void context)
- typedef void(cg_gps_assisted_cb_t)(cg_gps_assisted_connection_status_t status)

Enumerations

- enum cg_gps_status_t { CG_GPS_PRESENT = 0x1, CG_GPS_ENABLED = 0x2 }
  bitfield describing various status-flags for GPS
- enum cg_gps_assisted_connection_status_t { CG_GPS_ASSISTED_CONNECTION_SUCCESS, CG_GPS_ASSISTED_CONNECTION_FAILURE }
  latest SUPL server connection indication
- enum cg_gps_assisted_function_t { CG_GPS_ASSISTED_DISABLE, CG_GPS_ASSISTED_NON_SECURE_SUPL, CG_GPS_ASSISTED_SECURE_SUPL }
  disable or enable AGPS with a secure or non secure connection to the SUPL server.

Functions

- cg_status_t cg_gps_get_status (cg_gps_t gps)
- cg_status_t cg_gps_set_enabled (int enabled)
- cg_status_t cg_gps_set_reporting_interval (uint32_t interval)
- cg_status_t cg_gps_set_assisted (cg_gps_assisted_function_t function, const char url, uint32_t cert_count, const cg_gps_certificate_t cert)
- cg_status_t cg_gps_get_assisted (cg_gps_assisted_function_t function, char url)
- cg_status_t cg_gps_register_nmea_callback (cg_gps_nmea_cb_t cb, void context)

- cg_status_t cg_gps_deregister_nmea_callback (cg_gps_nmea_cb_t cb, void context)
- cg_status_t cg_gps_register_supl_connection_status_callback (cg_gps_assisted_cb_t cb)
- cg_status_t cg_gps_deregister_supl_connection_status_callback (void)

6.5.1 Detailed Description

Header file containing definitions for all GPS-related functionality.

6.6 libcg/cg_hal.h File Reference

Header file containing all definitions/functions related specific to the Hardware Abstraction Layer.

#include <libcg/cgate.h>

Data Structures

- struct cg_hal_serial_rs485_cfg_t
  Serial RS485-specific configuration settings.
- struct cg_hal_serial_cfg_t
  Generic configuration settings for a serial interface.

Enumerations

- enum cg_hal_serial_mode_t { CG_HAL_SERIAL_RS232 = 0x1, CG_HAL_SERIAL_RS485 = 0x2 }
  modes for RS485 or RS232

Functions

SERIAL - specific functions

- cg_status_t cg_hal_serial_set_config (const char dev_name, const cg_hal_serial_cfg_t config, int reboot_needed)

6.6.1 Detailed Description

Header file containing all definitions/functions related specific to the Hardware Abstraction Layer.

6.7 libcg/cg_net.h File Reference

Header file containing all definitions related to working with network interfaces, connecting, disconnecting...

#include <libcg/cgate.h>
#include <sys/socket.h>
#include <netinet/in.h>

Data Structures

- struct cg_ip_addr_t
  Structure describing a IP address.
- struct cg_net_ip_config_t
  Structure describing IP configuration of an interface.
- struct cg_net_leasetime_t
  Structure describing a leasetime.
- struct cg_net_dhcp_config_t
  Structure used to configure DHCP server functionality.
- struct cg_net_lan_cfg_t
  LAN-specific configuration settings for a network interface.
- struct cg_net_wan_cfg_t
  WAN-specific configuration settings for a network interface.
- struct cg_net_cfg_t
  Generic configuration settings for a network interface.
- struct cg_net_if_t
  Data structure describing a network interface in the system.
- struct cg_net_watchdog_settings_t
  Settings structure for the connection watchdog.
- struct cg_net_stats_t
  Structure that contains the data usage statistics for an interface.

Macros

- #define CG_MAX_PING_ADDRESSES 5
- #define CG_MAX_ADDRESS_LENGTH 64
- #define **CG_DEV_NAME_MAX** 32

Typedefs

- typedef char cg_dev_name_t [CG_DEV_NAME_MAX]
- typedef void(itf_status_callback_t)(const char dev_name, cg_net_if_status_t status)
- typedef void(internet_status_callback_t)(cg_net_internet_status_t status)

Enumerations

- enum cg_net_if_status_t { CG_NET_INTERFACE_ENABLED = 0x1, CG_NET_INTERFACE_CONFIGURED = 0x2, CG_NET_INTERFACE_CONNECTED = 0x4, CG_NET_INTERFACE_CAN_CONNECT = 0x8 }
  Interface status bitmasks.
- enum cg_net_internet_status_t { CG_NET_INTERNET_DISCONNECTED = 0, CG_NET_INTERNET_CONNECTING = 2, CG_NET_INTERNET_CONNECTED, CG_NET_INTERNET_ON_DEMAND }
  The internet connection status. Not interface-specific but shows the general state of internet (WAN) connectivity of the device. For clients who just want to know if it is possible to reach the internet, without having to look at all the WAN interfaces to know if one is connected or not.
- enum cg_net_zone_t { CG_NET_ZONE_LAN = 0, CG_NET_ZONE_WAN }
  Enumeration to describe the different zones (LAN, WAN)
- enum cg_net_mode_t { CG_NET_MODE_ALWAYS_ON = 0, CG_NET_MODE_ON_DEMAND }
  Describes the different connection modes.

- enum cg_net_if_type_t { CG_NET_TYPE_WWAN = 0, CG_NET_TYPE_NIC, CG_NET_TYPE_WLAN_AP, CG_NET_TYPE_WLAN, CG_NET_TYPE_BRIDGE, CG_NET_TYPE_IPSEC }

  Enumeration listing all the different types of possible interfaces.
- enum cg_net_ip_config_type_t { CG_STATIC = 0, CG_DHCP }

  Type of IP address configuration.
- enum cg_net_conn_strat_t { CG_NET_STRATEGY_MANUAL = 0, CG_NET_STRATEGY_PRIORITY }

  Enumeration of the different connection strategies. The device is able to execute different strategies to ensure connectivity automatically.
- enum cg_net_lease_unit_t { CG_NET_UNIT_MINUTE = 0, CG_NET_UNIT_HOUR, CG_NET_UNIT_DAY }

  Enumeration listing all the different units of the leasetime.
- enum cg_net_watchdog_action_t { **CG_CONN_WD_ACTION_RECONNECT** = 0, **CG_CONN_WD_ACTION_RESET** }

  Enumeration of the different watchdog actions.

Functions

- cg_status_t cg_net_get_interface_list (uint32_t num_interfaces, cg_net_if_t interfaces)
- cg_status_t cg_net_get_interface (const char dev_name, cg_net_if_t interface)
- cg_status_t cg_net_get_interface_by_type (cg_net_if_type_t type, uint32_t num_interfaces, cg_net_if_t itfs)
- cg_status_t cg_net_interface_set_enabled (const char dev_name, int enabled)
- cg_status_t cg_net_set_config (const char dev_name, const cg_net_cfg_t config)
- cg_status_t cg_net_register_interface_events (const char dev_name, itf_status_callback_t cb)
- cg_status_t cg_net_deregister_interface_events (const char dev_name)
- cg_status_t cg_net_set_manual_conn_strat (const char dev_name)
- cg_status_t cg_net_get_manual_conn_device (char dev_name)
- cg_status_t cg_net_set_priority_conn_strat (uint32_t num_entries, cg_dev_name_t priority_list)
- cg_status_t cg_net_get_conn_priority_list (uint32_t num_entries, cg_dev_name_t priority_list)
- cg_status_t cg_net_get_conn_strat (cg_net_conn_strat_t strat)
- cg_status_t cg_net_connect (void)
- cg_status_t cg_net_disconnect (void)
- cg_status_t cg_net_get_internet_status (cg_net_internet_status_t status)
- cg_status_t cg_net_register_internet_events (internet_status_callback_t cb)
- cg_status_t cg_net_deregister_internet_events (void)
- cg_status_t cg_net_set_watchdog (cg_net_watchdog_settings_t settings)
- cg_status_t cg_net_get_watchdog (cg_net_watchdog_settings_t settings)
- cg_status_t cg_net_datacounter_set_enabled (const char dev_name, int enabled)
- cg_status_t cg_net_datacounter_get_enabled (const char dev_name, int enabled)
- cg_status_t cg_net_datacounter_reset_stats (const char dev_name)
- cg_status_t cg_net_datacounter_get_stats (const char dev_name, cg_net_stats_t stats)
- cg_status_t cg_net_set_mtu (const char dev_name, int mtu)
- cg_status_t cg_net_get_mtu (const char dev_name, int mtu)
- cg_status_t cg_net_set_disabled_at_boot (const char dev_name, int disabled)
- cg_status_t cg_net_get_disabled_at_boot (const char dev_name, int disabled)

6.7.1 Detailed Description

Header file containing all definitions related to working with network interfaces, connecting, disconnecting...

6.7.2 Macro Definition Documentation

6.7.2.1 #define CG_MAX_PING_ADDRESSES 5

The maximum number of addresses which can be used for the connection watchdog

6.7.2.2 #define CG_MAX_ADDRESS_LENGTH 64

The maximum length of the addresses in the connection watchdog

6.8 libcg/cg_sms.h File Reference

Header file containing all definitions/functions related specific to SMS.

#include <libcg/cgate.h>

Typedefs

- typedef void(cg_sms_cb_t)(const char dev_name, uint16_t sms_size, unsigned char sms, void context)

Functions

- cg_status_t cg_sms_set_smsc (const char dev_name, const char number)
- cg_status_t cg_sms_get_smsc (const char dev_name, char number)
- cg_status_t cg_sms_register_new_sms (const char dev_name, cg_sms_cb_t cb, void context)
- cg_status_t cg_sms_deregister_new_sms (const char dev_name, cg_sms_cb_t cb, void context)
- cg_status_t cg_sms_send (const char dev_name, uint16_t sms_size, const unsigned char sms)

6.8.1 Detailed Description

Header file containing all definitions/functions related specific to SMS.

6.9 libcg/cg_ui.h File Reference

Header file containing definitions for all UI-related functionality.

#include <libcg/cgate.h>

Typedefs

- typedef char (cg_ui_json_cb_t)(char json_data, int logged_in, void context)
- typedef char (cg_ui_get_cb_t)(char query_string, int logged_in, void context)
- typedef void(cg_ui_session_cb)(unsigned int session_id, cg_ui_session_action_t action)

Enumerations

- enum cg_ui_session_action_t { **CG_UI_SESSION_ADD**, **CG_UI_SESSION_DEL**, **CG_UI_SESSION_PURGE** }
  Action to be taken on a session event.

Functions

- cg_status_t cg_ui_register_page (const char name, const char url)
- cg_status_t cg_ui_deregister_page (const char name)
- cg_status_t cg_ui_register_json_callback (const char identifier, cg_ui_json_cb_t cb, void context)
- cg_status_t cg_ui_deregister_json_callback (const char identifier)
- cg_status_t cg_ui_register_get_callback (const char identifier, cg_ui_get_cb_t cb, void context)
- cg_status_t cg_ui_deregister_get_callback (const char identifier)
- cg_status_t cg_ui_register_session_callback (cg_ui_session_cb cb)
- cg_status_t cg_ui_deregister_session_callback (void)

6.9.1 Detailed Description

Header file containing definitions for all UI-related functionality.

6.10 libcg/cg_upgrade.h File Reference

Header file containing definitions for upgrade API.

#include <stdint.h>

Typedefs

- typedef struct cg_upgrade cg_upgrade_t

Functions

- cg_status_t cg_upgrade_start (cg_upgrade_t handle)
- cg_status_t cg_upgrade_data (cg_upgrade_t handle, char buffer, uint32_t len)
- cg_status_t cg_upgrade_stop (cg_upgrade_t handle)

6.10.1 Detailed Description

Header file containing definitions for upgrade API.

6.11 libcg/cg_wlan.h File Reference

Header file containing all definitions/functions related specific to WLAN networking.

#include <libcg/cgate.h>

Data Structures

- struct cg_wlan_auth_params_t

  Authentication parameters for a WLAN client/access point.
- struct cg_wlan_network_t

  Description of a WLAN network.

Macros

- #define CG_SSID_MAX_LENGTH 32
- #define CG_PASSWORD_MAX_LENGTH 64

Typedefs

- typedef void(cg_wlan_network_list_cb_t)(cg_status_t status, const char dev_name, uint32_t num_entries, cg_wlan_network_t networks, void context)

Enumerations

- enum cg_wlan_auth_t {
  CG_WLAN_AUTH_OPEN = 0, CG_WLAN_AUTH_WEP, CG_WLAN_AUTH_WPA_PSK, CG_WLAN_AUTH_WPA2_PSK,
  CG_WLAN_AUTH_WPA_WPA2_PSK, CG_WLAN_AUTH_WPA_ENT, CG_WLAN_AUTH_WPA2_ENT,
  CG_WLAN_AUTH_WPA_WPA2_ENT }

  Different WLAN authentication type.

Functions

- cg_status_t cg_wlan_sta_scan_networks (const char dev_name, cg_wlan_network_list_cb_t cb, void context)
- cg_status_t cg_wlan_sta_set_network (const char dev_name, const char ssid, cg_wlan_auth_params_t auth)
- cg_status_t cg_wlan_sta_get_connected_network (const char dev_name, cg_wlan_network_t network)
- cg_status_t cg_wlan_sta_get_network_list (const char dev_name, uint32_t num_entries, cg_wlan_network_t networks)
- cg_status_t cg_wlan_sta_remove_network (const char dev_name, const char ssid, cg_wlan_auth_t auth)
- cg_status_t cg_wlan_ap_set_ssid (const char dev_name, const char ssid, uint8_t broadcast)
- cg_status_t cg_wlan_ap_get_ssid (const char dev_name, char ssid, uint8_t broadcast)
- cg_status_t cg_wlan_ap_set_channel (const char dev_name, uint16_t channel)
- cg_status_t cg_wlan_ap_get_channel (const char dev_name, uint16_t channel)
- cg_status_t cg_wlan_ap_set_auth_params (const char dev_name, cg_wlan_auth_params_t auth)
- cg_status_t cg_wlan_ap_get_auth_params (const char dev_name, cg_wlan_auth_params_t auth)

6.11.1 Detailed Description

Header file containing all definitions/functions related specific to WLAN networking.

6.12 libcg/cg_wlan_sniffer.h File Reference

Header file containing all definitions/functions related to wlan sniffing.

#include <libcg/cgate.h>

Data Structures

- struct cg_wlan_sniffer_params

Macros

- #define CG_MONITORING_FILE_PATH_MAX_LEN 64

Typedefs

- typedef struct cg_wlan_sniffer_params **cg_wlan_sniffer_params_t**

Functions

- cg_status_t cg_wlan_sniffer_start (uint32_t window_size, uint32_t channel, uint32_t max_file_size, uint32_t max_nbr_files, const char monitoring_file_path, uint32_t detailed_monitoring)
- cg_status_t cg_wlan_sniffer_stop (void)

6.12.1 Detailed Description

Header file containing all definitions/functions related to wlan sniffing.

6.13 libcg/cg_wwan.h File Reference

Header file containing all definitions/functions related specific to wwan (3G) networking.

#include <time.h>
#include <libcg/cgate.h>

Data Structures

- struct cg_wwan_network_t
  Structure defining a network.
- struct cg_wwan_network_list_t
  Structure for a list of networks.
- struct cg_wwan_reg_state_t
  Network registration state.
- struct cg_wwan_pin_state_t
  PIN code status of the SIM inserted in the device.
- struct cg_wwan_img_t

Image name.

- struct cg_wwan_img_list_t

  Structure containing a list of available images.
- struct cg_wwan_diag_param_t

  Diagnostic parameter information.
- struct cg_wwan_network_settings_t

  Struct used to set/get the mobile network settings.
- struct cg_wwan_location_info_t

  Struct defining the location of the base transceiver station.

Macros

- #define CG_NETWORK_NAME_SIZE 64
- #define CG_MAX_IMAGE_NAME 64
- #define GOBI_DEVICE "usb0"
- #define **TELIT_DEVICE** "wwan0"
- #define **HPEX_DEVICE** "hso0"

Typedefs

- typedef void(cg_wwan_network_list_cb_t)(cg_status_t status, cg_wwan_network_list_t list, void context)
- typedef void(cg_wwan_reg_state_cb_t)(const char dev_name, cg_wwan_reg_state_t state, void context)

Enumerations

- enum cg_wwan_pin_type_t { CG_WWAN_PIN_UNKNOWN = 0, CG_WWAN_PIN_SIM_NOT_INSERTED, CG_WWAN_PIN_READY, CG_WWAN_PIN_SIM_PIN, CG_WWAN_PIN_SIM_PUK, CG_WWAN_PIN_SIM_FAILURE, CG_WWAN_PIN_LOCKED_PN, CG_WWAN_PIN_SIM_NOT_SUPPORTED }
- enum cg_wwan_system_t { **CG_SYS_NONE** = 0x1, **CG_SYS_GSM** = 0x2, **CG_SYS_GPRS** = 0x4, **CG_SYS_EDGE** = 0x8, **CG_SYS_WCDMA** = 0x10, **CG_SYS_HSDPA** = 0x20, **CG_SYS_HSUPA** = 0x40, **CG_SYS_HSPA** = 0x80, **CG_SYS_HSDPAPLUS** = 0x100, **CG_SYS_HSPAPLUS** = 0x200, **CG_SYS_LTE** = 0x400, **CG_SYS_CDMA** = 0x1000, **CG_SYS_EVDO** = 0x2000, **CG_SYS_EVDOREL0** = 0x4000, **CG_SYS_EVDOREVA** = 0x8000 }

  Enumeration describing the possible mobile sytem technologies offered by the serving cell. Cells can indicate combinations of technologies , eg CG_SYS_WCDMA & CG_SYS_HSDPA & CG_SYS_HSUPA in general all represent 3G.
- enum cg_wwan_mode_t { CG_WWAN_MODE_CDMA = 0, CG_WWAN_MODE_UMTS }

  Indicate in what mode the WWAN device is. In case of CG_WWAN_TYPE_CDMA, you can use the CDMA-specific calls, but not the UMTS-specific calls. (Other way around for CG_WWAN_TYPE_UMTS)
- enum cg_wwan_creg_status_t { CG_WWAN_CREG_NOT_REG = 0, CG_WWAN_CREG_REG = 1, CG_WWAN_CREG_SEARCHING = 2, CG_WWAN_CREG_DENIED = 3, CG_WWAN_CREG_ROAMING = 5 }

  Indicate in what network registration status the WWAN device is.

- enum cg_wwan_act_state_t { CG_WWAN_ACT_STATUS_NOT_ACTIVATED = 0x00, CG_WWAN_ACT_STATUS_ACTIVATED = 0x01, CG_WWAN_ACT_STATUS_CONNECTION = 0x02, CG_WWAN_ACT_STATUS_CONNECTED = 0x03, CG_WWAN_ACT_STATUS_OTASP_AUTH = 0x04, CG_WWAN_ACT_STATUS_OTASP_NAM = 0x05, CG_WWAN_ACT_STATUS_OTASP_MDN = 0x06, CG_WWAN_ACT_STATUS_OTASP_IMSI = 0x07, CG_WWAN_ACT_STATUS_OTASP_PRL = 0x08, CG_WWAN_ACT_STATUS_OTASP_SPC = 0x09, CG_WWAN_ACT_STATUS_OTASP_COMMITTED = 0x0A }
- enum cg_wwan_sim_switch_mode_t { CG_SIM_SWITCH_MODE_AUTO, CG_SIM_SWITCH_MODE_SIM1, CG_SIM_SWITCH_MODE_SIM2 }
- enum cg_wwan_sim_switch_state_t { CG_SIM_SWITCH_STATE_SIM1 = 1, CG_SIM_SWITCH_STATE_SIM2 = 2 }
- enum cg_wwan_auth_type_t { CG_WWAN_AUTH_TYPE_NONE, CG_WWAN_AUTH_TYPE_PAP, CG_WWAN_AUTH_TYPE_CHAP, CG_WWAN_AUTH_TYPE_AUTO }

  The authentication type for the mobile connection.

Functions

Generic functions

- cg_status_t cg_wwan_get_active_primary_dev (char dev_name)
- cg_status_t cg_wwan_get_mode (const char dev_name, cg_wwan_mode_t mode)
- cg_status_t cg_wwan_signal_strength (const char dev_name, int8_t strength, int8_t ecio)
- cg_status_t cg_wwan_set_radio (const char dev_name, int enabled)
- cg_status_t cg_wwan_get_radio (const char dev_name, int enabled)
- cg_status_t cg_wwan_get_reg_state (const char dev_name, cg_wwan_reg_state_t state)
- cg_status_t cg_wwan_register_reg_state_notification (const char dev_name, cg_wwan_reg_state_cb_t cb, void context)
- cg_status_t cg_wwan_deregister_reg_state_notification (const char dev_name, cg_wwan_reg_state_cb_t cb, void context)
- cg_status_t cg_wwan_get_serial (const char dev_name, char serial)
- cg_status_t cg_wwan_get_diagnostics (const char dev_name, uint32_t num_params, cg_wwan_diag_param_t params)
- cg_status_t cg_wwan_set_network_settings (const char dev_name, cg_wwan_network_settings_t wwan_network_settings)
- cg_status_t cg_wwan_get_network_settings (const char dev_name, cg_wwan_network_settings_t wwan_network_settings)
- cg_status_t cg_wwan_get_phone_number (const char dev_name, char phone_number)

UMTS - specific functions

- cg_status_t cg_wwan_search_networks (const char dev_name, cg_wwan_network_list_cb_t cb, void context)
- cg_status_t cg_wwan_get_imei (const char dev_name, char imei)
- cg_status_t cg_wwan_get_imsi (const char dev_name, char imsi)
- cg_status_t cg_wwan_get_iccid (const char dev_name, char iccid)
- cg_status_t cg_wwan_pin_get_state (const char dev_name, cg_wwan_pin_state_t pin_state)
- cg_status_t cg_wwan_save_pin (const char dev_name, uint8_t enabled)
- cg_status_t cg_wwan_submit_pin (const char dev_name, const char pin)
- cg_status_t cg_wwan_change_pin (const char dev_name, const char old_pin_puk, const char new_pin)
- cg_status_t cg_wwan_pin_set_enabled (const char dev_name, uint8_t enabled, const char pin)
- cg_status_t cg_wwan_get_nw_selection_mode (const char dev_name, uint8_t automatic, cg_wwan_network_t network)

- cg_status_t cg_wwan_set_nw_selection_mode (const char dev_name, uint8_t automatic, cg_wwan_network_t network)

CDMA - specific functions

- cg_status_t cg_wwan_get_meid (const char dev_name, char meid)
- cg_status_t cg_wwan_get_activation_state (const char dev_name, cg_wwan_act_state_t state)
- cg_status_t cg_wwan_activate (const char dev_name, const char act_string)
- cg_status_t cg_wwan_get_prl_version (const char dev_name, uint16_t prl_version)
- cg_status_t cg_wwan_upload_prl (const char dev_name)
- cg_status_t cg_wwan_set_prl_network_update (const char dev_name, uint8_t enabled)
- cg_status_t cg_wwan_get_prl_network_update (const char dev_name, uint8_t enabled)

Device - specific functions.

- cg_status_t cg_wwan_gobi_get_active_image (const char dev_name, char image_name)
- cg_status_t cg_wwan_gobi_get_image_list (const char dev_name, cg_wwan_img_list_t image_list)
- cg_status_t cg_wwan_gobi_set_active_image (const char dev_name, const char image_name)
- cg_status_t cg_wwan_gobi_get_image_type (const char dev_name, const char image_name, char image_type)
- cg_status_t cg_wwan_modem_reboot ()
- cg_status_t cg_wwan_set_sim_switch (cg_wwan_sim_switch_mode_t mode)
- cg_status_t cg_wwan_get_sim_switch (cg_wwan_sim_switch_mode_t mode, cg_wwan_sim_switch_state_t current)
- cg_status_t cg_wwan_set_connection_hunting (const char dev_name, int enabled, int fallback_time)
- cg_status_t cg_wwan_get_location_info (const char dev_name, cg_wwan_location_info_t location_info)

6.13.1 Detailed Description

Header file containing all definitions/functions related specific to wwan (3G) networking.

6.14 libcg/cgate.h File Reference

Main SDK header file.

#include <stdint.h>

Macros

- #define **CG_BEGIN_DECLS**
- #define **CG_END_DECLS**
- #define **NULL** ((void) 0)
- #define **FALSE** (0)
- #define **TRUE** (!FALSE)
- #define **CG_GNUC_PRINTF**(format_idx, arg_idx)

Enumerations

- enum cg_status_t {
  CG_STATUS_OK = 0, CG_STATUS_ERROR, CG_STATUS_RESOURCE_BUSY, CG_STATUS_INVALID_-
  PARAMETER,
  CG_STATUS_RESOURCE_UNAVAILABLE, CG_STATUS_DOWNGRADE_PREVENTION }
  List of possible return statuses used throughout the CloudGate SDK.

Functions

- char cg_get_last_error (void)

6.14.1 Detailed Description

Main SDK header file.

6.14.2 Enumeration Type Documentation

6.14.2.1 enum cg_status_t

List of possible return statuses used throughout the CloudGate SDK.

Enumerator

CG_STATUS_OK The SDK function completed successfully.

CG_STATUS_ERROR The SDK function completed with an error.

CG_STATUS_RESOURCE_BUSY The SDK function could not be completed because a device was busy. I.e. a device is too cold (and is heating up) or too hot (and is cooling down).

CG_STATUS_INVALID_PARAMETER The SDK function could not be completed because an invalid parameter was provided. I.e. a NULL pointer, wrong device name, ...

CG_STATUS_RESOURCE_UNAVAILABLE The SDK function could not be completed because a device or resource is temporary unavailable.

CG_STATUS_DOWNGRADE_PREVENTION The SDK function could not be completed because of device downgrade prevention.

6.14.3 Function Documentation

6.14.3.1 char cg_get_last_error (void)

Get a description of the last error encountered by the SDK

Returns

A string describing the error

Index