

O O  
O **O**P**T**I**N  
O O**



# CloudGate SDK reference manual

Publication date: February 11, 2019  
Revision: V035ext

# Contents

<b>1</b>	<b>Module Index</b>	<b>3</b>
1.1	Modules . . . . .	3
<b>2</b>	<b>Data Structure Index</b>	<b>5</b>
2.1	Data Structures . . . . .	5
<b>3</b>	<b>File Index</b>	<b>7</b>
3.1	File List . . . . .	7
<b>4</b>	<b>Module Documentation</b>	<b>9</b>
4.1	Configuration functions . . . . .	9
4.1.1	Detailed Description . . . . .	9
4.1.2	Function Documentation . . . . .	9
4.1.2.1	<code>cg_conf_open</code> . . . . .	9
4.1.2.2	<code>cg_conf_close</code> . . . . .	10
4.1.2.3	<code>cg_conf_revert</code> . . . . .	10
4.1.2.4	<code>cg_conf_set</code> . . . . .	10
4.1.2.5	<code>cg_conf_get</code> . . . . .	11
4.1.2.6	<code>cg_conf_add_section</code> . . . . .	11
4.1.2.7	<code>cg_conf_del</code> . . . . .	11
4.1.2.8	<code>cg_conf_get_sections</code> . . . . .	12
4.1.2.9	<code>cg_conf_get_options</code> . . . . .	12
4.2	Device Management . . . . .	13
4.2.1	Detailed Description . . . . .	13
4.2.2	Typedef Documentation . . . . .	14
4.2.2.1	<code>device_notification_cb_t</code> . . . . .	14
4.2.2.2	<code>device_claim_cb_t</code> . . . . .	14
4.2.3	Enumeration Type Documentation . . . . .	14
4.2.3.1	<code>cg_device_status_t</code> . . . . .	14
4.2.3.2	<code>cg_device_type_t</code> . . . . .	14
4.2.3.3	<code>cg_location_t</code> . . . . .	15
4.2.3.4	<code>cg_device_event_t</code> . . . . .	15

4.2.4.1	cg_device_list . . . . .	15
4.2.4.2	cg_device_list_by_location . . . . .	15
4.2.4.3	cg_device_claim . . . . .	16
4.2.4.4	cg_device_release . . . . .	16
4.2.4.5	cg_device_register_notification . . . . .	17
4.2.4.6	cg_device_deregister_notification . . . . .	17
4.2.4.7	cg_device_board_list . . . . .	17
4.2.4.8	cg_device_board_device_list . . . . .	17
4.3	DUST_SENSOR . . . . .	19
4.3.1	Detailed Description . . . . .	19
4.3.2	Typedef Documentation . . . . .	19
4.3.2.1	cg_dust_sensor_data_cb_t . . . . .	19
4.3.3	Enumeration Type Documentation . . . . .	20
4.3.3.1	cg_dust_sensor_power_t . . . . .	20
4.3.3.2	cg_dust_sensor_status_t . . . . .	20
4.3.4	Function Documentation . . . . .	20
4.3.4.1	cg_dust_sensor_start . . . . .	20
4.3.4.2	cg_dust_sensor_stop . . . . .	20
4.3.4.3	cg_dust_sensor_set_power . . . . .	21
4.3.4.4	cg_dust_sensor_register_data_callback . . . . .	21
4.3.4.5	cg_dust_sensor_deregister_data_callback . . . . .	21
4.4	General functions . . . . .	22
4.4.1	Detailed Description . . . . .	23
4.4.2	Macro Definition Documentation . . . . .	23
4.4.2.1	CG_SDK_API_LEVEL . . . . .	23
4.4.2.2	CG_MAX_SLOT_INFO_LEN . . . . .	23
4.4.3	Typedef Documentation . . . . .	24
4.4.3.1	cg_prov_kvpair_update_cb_t . . . . .	24
4.4.3.2	cg_system_sync_ntp_cb_t . . . . .	24
4.4.4	Enumeration Type Documentation . . . . .	24
4.4.4.1	cg_prov_status_t . . . . .	24
4.4.4.2	cg_slot_type_t . . . . .	24
4.4.4.3	cg_dev_feature_t . . . . .	25
4.4.4.4	cg_system_boot_condition_t . . . . .	25
4.4.5	Function Documentation . . . . .	25
4.4.5.1	cg_init . . . . .	25
4.4.5.2	cg_deinit . . . . .	25
4.4.5.3	cg_get_api_level . . . . .	25
4.4.5.4	cg_get_serial_number . . . . .	26

4.4.5.5	cg_prov_get_status . . . . .	26
4.4.5.6	cg_prov_check_update . . . . .	26
4.4.5.7	cg_prov_set_automatic . . . . .	26
4.4.5.8	cg_prov_kvpair_get . . . . .	27
4.4.5.9	cg_prov_kvpair_set . . . . .	27
4.4.5.10	cg_prov_kvpair_del . . . . .	27
4.4.5.11	cg_prov_kvpair_list . . . . .	28
4.4.5.12	cg_prov_kvpair_register_update_callback . . . . .	28
4.4.5.13	cg_prov_kvpair_deregister_update_callback . . . . .	28
4.4.5.14	cg_get_slot_list . . . . .	28
4.4.5.15	cg_reset_system . . . . .	29
4.4.5.16	cg_reset_peripherals . . . . .	29
4.4.5.17	cg_system_log . . . . .	29
4.4.5.18	cg_set_dev_mode . . . . .	29
4.4.5.19	cg_system_power_down . . . . .	30
4.4.5.20	cg_system_set_timed_wakeup . . . . .	30
4.4.5.21	cg_system_set_date_time . . . . .	30
4.4.5.22	cg_system_get_date_time . . . . .	31
4.4.5.23	cg_system_set_timezone . . . . .	31
4.4.5.24	cg_system_set_ntp_server . . . . .	31
4.4.5.25	cg_system_get_ntp_server . . . . .	31
4.4.5.26	cg_system_sync_ntp . . . . .	32
4.4.5.27	cg_system_sync_ntp_register_callback . . . . .	32
4.4.5.28	cg_system_sync_ntp_deregister_callback . . . . .	32
4.5	GPS functions . . . . .	33
4.5.1	Detailed Description . . . . .	33
4.5.2	Macro Definition Documentation . . . . .	33
4.5.2.1	CG_MAX_CERTIFICATES . . . . .	33
4.5.2.2	CG_MAX_LEN_CERTIFICATE . . . . .	34
4.5.3	Typedef Documentation . . . . .	34
4.5.3.1	cg_gps_nmea_cb_t . . . . .	34
4.5.3.2	cg_gps_assisted_cb_t . . . . .	34
4.5.4	Enumeration Type Documentation . . . . .	34
4.5.4.1	cg_gps_status_t . . . . .	34
4.5.4.2	cg_gps_assisted_connection_status_t . . . . .	34
4.5.4.3	cg_gps_assisted_function_t . . . . .	35
4.5.5	Function Documentation . . . . .	35
4.5.5.1	cg_gps_get_status . . . . .	35
4.5.5.2	cg_gps_set_enabled . . . . .	35
4.5.5.3	cg_gps_set_reporting_interval . . . . .	35

4.5.5.4	cg_gps_set_assisted . . . . .	36
4.5.5.5	cg_gps_get_assisted . . . . .	36
4.5.5.6	cg_gps_register_nmea_callback . . . . .	37
4.5.5.7	cg_gps_deregister_nmea_callback . . . . .	37
4.5.5.8	cg_gps_register_supl_connection_status_callback . . . . .	37
4.5.5.9	cg_gps_deregister_supl_connection_status_callback . . . . .	37
4.6	HAL . . . . .	39
4.6.1	Detailed Description . . . . .	39
4.6.2	Enumeration Type Documentation . . . . .	39
4.6.2.1	cg_hal_serial_mode_t . . . . .	39
4.6.3	Function Documentation . . . . .	39
4.6.3.1	cg_hal_serial_set_config . . . . .	39
4.7	Networking . . . . .	40
4.7.1	Detailed Description . . . . .	41
4.7.2	Typedef Documentation . . . . .	42
4.7.2.1	cg_dev_name_t . . . . .	42
4.7.2.2	itf_status_callback_t . . . . .	42
4.7.2.3	internet_status_callback_t . . . . .	42
4.7.3	Enumeration Type Documentation . . . . .	42
4.7.3.1	cg_net_if_status_t . . . . .	42
4.7.3.2	cg_net_internet_status_t . . . . .	42
4.7.3.3	cg_net_zone_t . . . . .	43
4.7.3.4	cg_net_mode_t . . . . .	43
4.7.3.5	cg_net_if_type_t . . . . .	43
4.7.3.6	cg_net_ip_config_type_t . . . . .	43
4.7.3.7	cg_net_conn_strat_t . . . . .	43
4.7.3.8	cg_net_lease_unit_t . . . . .	44
4.7.4	Function Documentation . . . . .	44
4.7.4.1	cg_net_get_interface_list . . . . .	44
4.7.4.2	cg_net_get_interface . . . . .	44
4.7.4.3	cg_net_get_interface_by_type . . . . .	44
4.7.4.4	cg_net_interface_set_enabled . . . . .	45
4.7.4.5	cg_net_set_config . . . . .	45
4.7.4.6	cg_net_register_interface_events . . . . .	45
4.7.4.7	cg_net_deregister_interface_events . . . . .	46
4.7.4.8	cg_net_set_manual_conn_strat . . . . .	46
4.7.4.9	cg_net_get_manual_conn_device . . . . .	46
4.7.4.10	cg_net_set_priority_conn_strat . . . . .	46
4.7.4.11	cg_net_get_conn_priority_list . . . . .	47
4.7.4.12	cg_net_get_conn_strat . . . . .	47

4.7.4.13 <code>cg_net_connect</code>	47
4.7.4.14 <code>cg_net_disconnect</code>	47
4.7.4.15 <code>cg_net_get_internet_status</code>	48
4.7.4.16 <code>cg_net_register_internet_events</code>	48
4.7.4.17 <code>cg_net_deregister_internet_events</code>	48
4.7.4.18 <code>cg_net_set_watchdog</code>	48
4.7.4.19 <code>cg_net_get_watchdog</code>	48
4.7.4.20 <code>cg_net_datacounter_set_enabled</code>	49
4.7.4.21 <code>cg_net_datacounter_get_enabled</code>	49
4.7.4.22 <code>cg_net_datacounter_reset_stats</code>	49
4.7.4.23 <code>cg_net_datacounter_get_stats</code>	50
4.7.4.24 <code>cg_net_set_mtu</code>	50
4.7.4.25 <code>cg_net_get_mtu</code>	50
4.7.4.26 <code>cg_net_set_disabled_at_boot</code>	51
4.7.4.27 <code>cg_net_get_disabled_at_boot</code>	51
4.8 SMS	52
4.8.1 Detailed Description	52
4.8.2 Typedef Documentation	52
4.8.2.1 <code>cg_sms_cb_t</code>	52
4.8.3 Function Documentation	52
4.8.3.1 <code>cg_sms_set_smsc</code>	52
4.8.3.2 <code>cg_sms_get_smsc</code>	53
4.8.3.3 <code>cg_sms_register_new_sms</code>	53
4.8.3.4 <code>cg_sms_deregister_new_sms</code>	53
4.8.3.5 <code>cg_sms_send</code>	53
4.9 UI	55
4.9.1 Detailed Description	55
4.9.2 Typedef Documentation	56
4.9.2.1 <code>cg_ui_json_cb_t</code>	56
4.9.2.2 <code>cg_ui_get_cb_t</code>	57
4.9.2.3 <code>cg_ui_session_cb</code>	57
4.9.3 Function Documentation	57
4.9.3.1 <code>cg_ui_register_page</code>	57
4.9.3.2 <code>cg_ui_deregister_page</code>	57
4.9.3.3 <code>cg_ui_register_json_callback</code>	58
4.9.3.4 <code>cg_ui_deregister_json_callback</code>	58
4.9.3.5 <code>cg_ui_register_get_callback</code>	58
4.9.3.6 <code>cg_ui_deregister_get_callback</code>	59
4.9.3.7 <code>cg_ui_register_session_callback</code>	59
4.9.3.8 <code>cg_ui_deregister_session_callback</code>	59

4.10 Upgrade API . . . . .	60
4.10.1 Detailed Description . . . . .	60
4.10.2 Typedef Documentation . . . . .	60
4.10.2.1 <code>cg_upgrade_t</code> . . . . .	60
4.10.3 Function Documentation . . . . .	60
4.10.3.1 <code>cg_upgrade_start</code> . . . . .	60
4.10.3.2 <code>cg_upgrade_data</code> . . . . .	60
4.10.3.3 <code>cg_upgrade_stop</code> . . . . .	61
4.11 WLAN . . . . .	62
4.11.1 Detailed Description . . . . .	63
4.11.2 Macro Definition Documentation . . . . .	63
4.11.2.1 <code>CG_SSID_MAX_LENGTH</code> . . . . .	63
4.11.2.2 <code>CG_PASSWORD_MAX_LENGTH</code> . . . . .	63
4.11.3 Typedef Documentation . . . . .	63
4.11.3.1 <code>cg_wlan_network_list_cb_t</code> . . . . .	63
4.11.4 Enumeration Type Documentation . . . . .	63
4.11.4.1 <code>cg_wlan_auth_t</code> . . . . .	63
4.11.5 Function Documentation . . . . .	63
4.11.5.1 <code>cg_wlan_sta_scan_networks</code> . . . . .	63
4.11.5.2 <code>cg_wlan_sta_set_network</code> . . . . .	64
4.11.5.3 <code>cg_wlan_sta_get_connected_network</code> . . . . .	64
4.11.5.4 <code>cg_wlan_sta_get_network_list</code> . . . . .	65
4.11.5.5 <code>cg_wlan_sta_remove_network</code> . . . . .	65
4.11.5.6 <code>cg_wlan_ap_set_ssid</code> . . . . .	65
4.11.5.7 <code>cg_wlan_ap_get_ssid</code> . . . . .	66
4.11.5.8 <code>cg_wlan_ap_set_channel</code> . . . . .	66
4.11.5.9 <code>cg_wlan_ap_get_channel</code> . . . . .	66
4.11.5.10 <code>cg_wlan_ap_set_auth_params</code> . . . . .	67
4.11.5.11 <code>cg_wlan_ap_get_auth_params</code> . . . . .	67
4.12 WLAN_SNIFFER . . . . .	68
4.12.1 Detailed Description . . . . .	68
4.12.2 Macro Definition Documentation . . . . .	68
4.12.2.1 <code>CG_MONITORING_FILE_PATH_MAX_LEN</code> . . . . .	68
4.12.3 Function Documentation . . . . .	68
4.12.3.1 <code>cg_wlan_sniffer_start</code> . . . . .	68
4.12.3.2 <code>cg_wlan_sniffer_stop</code> . . . . .	69
4.13 WWAN . . . . .	70
4.13.1 Detailed Description . . . . .	72
4.13.2 Macro Definition Documentation . . . . .	72
4.13.2.1 <code>CG_NETWORK_NAME_SIZE</code> . . . . .	72

4.13.2.2 CG_MAX_IMAGE_NAME . . . . .	73
4.13.2.3 GOBI_DEVICE . . . . .	73
4.13.3 Typedef Documentation . . . . .	73
4.13.3.1 cg_wwan_network_list_cb_t . . . . .	73
4.13.3.2 cg_wwan_reg_state_cb_t . . . . .	73
4.13.4 Enumeration Type Documentation . . . . .	73
4.13.4.1 cg_wwan_pin_type_t . . . . .	73
4.13.4.2 cg_wwan_mode_t . . . . .	74
4.13.4.3 cg_wwan_creg_status_t . . . . .	74
4.13.4.4 cg_wwan_act_state_t . . . . .	74
4.13.4.5 cg_wwan_sim_switch_mode_t . . . . .	75
4.13.4.6 cg_wwan_sim_switch_state_t . . . . .	75
4.13.4.7 cg_wwan_auth_type_t . . . . .	75
4.13.5 Function Documentation . . . . .	75
4.13.5.1 cg_wwan_get_active_primary_dev . . . . .	75
4.13.5.2 cg_wwan_get_mode . . . . .	76
4.13.5.3 cg_wwan_signal_strength . . . . .	76
4.13.5.4 cg_wwan_set_radio . . . . .	76
4.13.5.5 cg_wwan_get_radio . . . . .	77
4.13.5.6 cg_wwan_get_reg_state . . . . .	77
4.13.5.7 cg_wwan_register_reg_state_notification . . . . .	77
4.13.5.8 cg_wwan_deregister_reg_state_notification . . . . .	78
4.13.5.9 cg_wwan_get_serial . . . . .	78
4.13.5.10 cg_wwan_get_diagnostics . . . . .	78
4.13.5.11 cg_wwan_set_network_settings . . . . .	79
4.13.5.12 cg_wwan_get_network_settings . . . . .	79
4.13.5.13 cg_wwan_get_phone_number . . . . .	79
4.13.5.14 cg_wwan_search_networks . . . . .	80
4.13.5.15 cg_wwan_get_imei . . . . .	80
4.13.5.16 cg_wwan_get_imsi . . . . .	80
4.13.5.17 cg_wwan_get_iccid . . . . .	81
4.13.5.18 cg_wwan_pin_get_state . . . . .	81
4.13.5.19 cg_wwan_save_pin . . . . .	81
4.13.5.20 cg_wwan_submit_pin . . . . .	81
4.13.5.21 cg_wwan_change_pin . . . . .	82
4.13.5.22 cg_wwan_pin_set_enabled . . . . .	82
4.13.5.23 cg_wwan_get_nw_selection_mode . . . . .	82
4.13.5.24 cg_wwan_set_nw_selection_mode . . . . .	83
4.13.5.25 cg_wwan_get_meid . . . . .	83
4.13.5.26 cg_wwan_get_activation_state . . . . .	83

4.13.5.27cg_wwan_activate . . . . .	84
4.13.5.28cg_wwan_get_prl_version . . . . .	84
4.13.5.29cg_wwan_upload_prl . . . . .	84
4.13.5.30cg_wwan_set_prl_network_update . . . . .	84
4.13.5.31cg_wwan_get_prl_network_update . . . . .	85
4.13.5.32cg_wwan_gobi_get_active_image . . . . .	85
4.13.5.33cg_wwan_gobi_get_image_list . . . . .	85
4.13.5.34cg_wwan_gobi_set_active_image . . . . .	86
4.13.5.35cg_wwan_gobi_get_image_type . . . . .	86
4.13.5.36cg_wwan_modem_reboot . . . . .	86
4.13.5.37cg_wwan_set_sim_switch . . . . .	87
4.13.5.38cg_wwan_get_sim_switch . . . . .	87
4.13.5.39cg_wwan_set_connection_hunting . . . . .	87
4.13.5.40cg_wwan_get_location_info . . . . .	87
<b>5 Data Structure Documentation . . . . .</b>	<b>89</b>
5.1 cg_board_t Struct Reference . . . . .	89
5.1.1 Detailed Description . . . . .	89
5.1.2 Field Documentation . . . . .	89
5.1.2.1 slot_id . . . . .	89
5.1.2.2 board_id . . . . .	89
5.1.2.3 hw_ver . . . . .	89
5.2 cg_conf_list Struct Reference . . . . .	90
5.2.1 Field Documentation . . . . .	90
5.2.1.1 number . . . . .	90
5.2.1.2 values . . . . .	90
5.3 cg_conf_list_t Struct Reference . . . . .	90
5.3.1 Detailed Description . . . . .	90
5.4 cg_config_t Struct Reference . . . . .	90
5.4.1 Detailed Description . . . . .	90
5.5 cg_date_time_t Struct Reference . . . . .	91
5.5.1 Detailed Description . . . . .	91
5.6 cg_device_t Struct Reference . . . . .	91
5.6.1 Detailed Description . . . . .	91
5.6.2 Field Documentation . . . . .	91
5.6.2.1 friendly_name . . . . .	91
5.6.2.2 device_name . . . . .	91
5.6.2.3 device_type . . . . .	92
5.6.2.4 device_location . . . . .	92
5.6.2.5 status . . . . .	92

5.7	cg_dust_sensor_data_t Struct Reference . . . . .	92
5.7.1	Detailed Description . . . . .	92
5.7.2	Field Documentation . . . . .	92
5.7.2.1	byte . . . . .	92
5.7.2.2	status . . . . .	92
5.8	cg_gps_certificate_t Struct Reference . . . . .	93
5.8.1	Detailed Description . . . . .	93
5.8.2	Field Documentation . . . . .	93
5.8.2.1	len . . . . .	93
5.8.2.2	cert . . . . .	93
5.9	cg_gps_t Struct Reference . . . . .	93
5.9.1	Detailed Description . . . . .	93
5.9.2	Field Documentation . . . . .	94
5.9.2.1	status . . . . .	94
5.9.2.2	reporting_interval . . . . .	94
5.10	cg_hal_serial_cfg_t Struct Reference . . . . .	94
5.10.1	Detailed Description . . . . .	94
5.10.2	Field Documentation . . . . .	94
5.10.2.1	serial_mode . . . . .	94
5.10.2.2	rs485 . . . . .	94
5.10.2.3	param . . . . .	94
5.11	cg_hal_serial_rs485_cfg_t Struct Reference . . . . .	95
5.11.1	Detailed Description . . . . .	95
5.11.2	Field Documentation . . . . .	95
5.11.2.1	enable_termination . . . . .	95
5.12	cg_ip_addr_t Struct Reference . . . . .	95
5.12.1	Detailed Description . . . . .	95
5.13	cg_key_t Struct Reference . . . . .	96
5.13.1	Detailed Description . . . . .	96
5.14	cg_kvpair_t Struct Reference . . . . .	96
5.14.1	Detailed Description . . . . .	96
5.15	cg_net_cfg_t Struct Reference . . . . .	96
5.15.1	Detailed Description . . . . .	97
5.15.2	Field Documentation . . . . .	97
5.15.2.1	ip_config . . . . .	97
5.15.2.2	zone . . . . .	97
5.15.2.3	conf . . . . .	97
5.16	cg_net_dhcp_config_t Struct Reference . . . . .	97
5.16.1	Detailed Description . . . . .	97
5.16.2	Field Documentation . . . . .	98

5.16.2.1 enabled . . . . .	98
5.16.2.2 ip_start . . . . .	98
5.16.2.3 ip_end . . . . .	98
5.16.2.4 dns1 . . . . .	98
5.16.2.5 dns2 . . . . .	98
5.16.2.6 leasetime . . . . .	98
5.17 cg_net_if_t Struct Reference . . . . .	98
5.17.1 Detailed Description . . . . .	98
5.17.2 Field Documentation . . . . .	99
5.17.2.1 dev_name . . . . .	99
5.17.2.2 status . . . . .	99
5.17.2.3 type . . . . .	99
5.17.2.4 config . . . . .	99
5.18 cg_net_ip_config_t Struct Reference . . . . .	99
5.18.1 Detailed Description . . . . .	99
5.18.2 Field Documentation . . . . .	99
5.18.2.1 ip_addr . . . . .	99
5.18.2.2 netmask . . . . .	99
5.18.2.3 mtu . . . . .	100
5.19 cg_net_lan_cfg_t Struct Reference . . . . .	100
5.19.1 Detailed Description . . . . .	100
5.19.2 Field Documentation . . . . .	100
5.19.2.1 dhcp_config . . . . .	100
5.19.2.2 bridge_dev_name . . . . .	100
5.20 cg_net_leasetime_t Struct Reference . . . . .	100
5.20.1 Detailed Description . . . . .	101
5.20.2 Field Documentation . . . . .	101
5.20.2.1 amount . . . . .	101
5.20.2.2 unit . . . . .	101
5.21 cg_net_stats_t Struct Reference . . . . .	101
5.21.1 Detailed Description . . . . .	101
5.21.2 Field Documentation . . . . .	101
5.21.2.1 rx_packets . . . . .	101
5.21.2.2 rx_bytes . . . . .	101
5.21.2.3 tx_packets . . . . .	101
5.21.2.4 tx_bytes . . . . .	102
5.22 cg_net_wan_cfg_t Struct Reference . . . . .	102
5.22.1 Detailed Description . . . . .	102
5.22.2 Field Documentation . . . . .	102
5.22.2.1 mode . . . . .	102

5.22.2.2 data_timeout . . . . .	102
5.22.2.3 ip_config_type . . . . .	102
5.22.2.4 gw . . . . .	102
5.22.2.5 dns1 . . . . .	103
5.22.2.6 dns2 . . . . .	103
5.23 cg_net_watchdog_settings_t Struct Reference . . . . .	103
5.23.1 Detailed Description . . . . .	103
5.23.2 Field Documentation . . . . .	103
5.23.2.1 enabled . . . . .	103
5.23.2.2 use_ping . . . . .	103
5.23.2.3 interval . . . . .	103
5.23.2.4 action . . . . .	104
5.23.2.5 num_addresses . . . . .	104
5.23.2.6 addresses . . . . .	104
5.24 cg_slot_list_t Struct Reference . . . . .	104
5.24.1 Detailed Description . . . . .	104
5.24.2 Field Documentation . . . . .	104
5.24.2.1 num_slots . . . . .	104
5.24.2.2 slots . . . . .	104
5.25 cg_slot_t Struct Reference . . . . .	105
5.25.1 Detailed Description . . . . .	105
5.25.2 Field Documentation . . . . .	105
5.25.2.1 type . . . . .	105
5.25.2.2 active . . . . .	105
5.25.2.3 size . . . . .	105
5.25.2.4 version . . . . .	105
5.25.2.5 uid . . . . .	105
5.26 cg_timezone_t Struct Reference . . . . .	105
5.26.1 Detailed Description . . . . .	106
5.27 cg_wlan_auth_params_t Struct Reference . . . . .	106
5.27.1 Detailed Description . . . . .	106
5.27.2 Field Documentation . . . . .	106
5.27.2.1 type . . . . .	106
5.27.2.2 password . . . . .	106
5.28 cg_wlan_network_t Struct Reference . . . . .	107
5.28.1 Detailed Description . . . . .	107
5.28.2 Field Documentation . . . . .	107
5.28.2.1 ssid . . . . .	107
5.28.2.2 auth_type . . . . .	107
5.28.2.3 channel . . . . .	107

5.28.2.4 signal_strength . . . . .	107
5.29 cg_wlan_sniffer_params Struct Reference . . . . .	107
5.29.1 Field Documentation . . . . .	108
5.29.1.1 window_size . . . . .	108
5.29.1.2 channel . . . . .	108
5.29.1.3 max_file_size . . . . .	108
5.29.1.4 max_nbr_files . . . . .	108
5.29.1.5 monitoring_file_path . . . . .	108
5.29.1.6 detailed_monitoring . . . . .	108
5.30 cg_wwan_diag_param_t Struct Reference . . . . .	108
5.30.1 Detailed Description . . . . .	108
5.30.2 Field Documentation . . . . .	109
5.30.2.1 name . . . . .	109
5.30.2.2 value . . . . .	109
5.31 cg_wwan_img_list_t Struct Reference . . . . .	109
5.31.1 Detailed Description . . . . .	109
5.31.2 Field Documentation . . . . .	109
5.31.2.1 entries . . . . .	109
5.31.2.2 current . . . . .	109
5.31.2.3 images . . . . .	109
5.32 cg_wwan_img_t Struct Reference . . . . .	110
5.32.1 Detailed Description . . . . .	110
5.32.2 Field Documentation . . . . .	110
5.32.2.1 name . . . . .	110
5.33 cg_wwan_location_info_t Struct Reference . . . . .	110
5.33.1 Detailed Description . . . . .	110
5.33.2 Field Documentation . . . . .	110
5.33.2.1 lac . . . . .	110
5.33.2.2 tac . . . . .	111
5.33.2.3 cell_id . . . . .	111
5.34 cg_wwan_network_list_t Struct Reference . . . . .	111
5.34.1 Detailed Description . . . . .	111
5.34.2 Field Documentation . . . . .	111
5.34.2.1 entries . . . . .	111
5.34.2.2 networks . . . . .	111
5.35 cg_wwan_network_settings_t Struct Reference . . . . .	111
5.35.1 Detailed Description . . . . .	112
5.35.2 Field Documentation . . . . .	112
5.35.2.1 apn . . . . .	112
5.35.2.2 username . . . . .	112

5.35.2.3 password . . . . .	112
5.35.2.4 auth_type . . . . .	112
5.36 cg_wwan_network.t Struct Reference . . . . .	112
5.36.1 Detailed Description . . . . .	113
5.36.2 Field Documentation . . . . .	113
5.36.2.1 name . . . . .	113
5.36.2.2 system_type . . . . .	113
5.36.2.3 id . . . . .	113
5.36.2.4 preferred . . . . .	113
5.36.2.5 roaming . . . . .	113
5.36.2.6 forbidden . . . . .	113
5.36.2.7 home . . . . .	113
5.37 cg_wwan_pin_state.t Struct Reference . . . . .	113
5.37.1 Detailed Description . . . . .	114
5.37.2 Field Documentation . . . . .	114
5.37.2.1 pin_type . . . . .	114
5.37.2.2 pin_retries . . . . .	114
5.37.2.3 puk_retries . . . . .	114
5.37.2.4 pin_enabled . . . . .	114
5.38 cg_wwan_reg_state.t Struct Reference . . . . .	114
5.38.1 Detailed Description . . . . .	114
5.38.2 Field Documentation . . . . .	115
5.38.2.1 cs_state . . . . .	115
5.38.2.2 ps_state . . . . .	115
5.38.2.3 system_type . . . . .	115
5.38.2.4 network_name . . . . .	115
<b>6 File Documentation</b> . . . . .	<b>117</b>
6.1 libcg/cg_conf.h File Reference . . . . .	117
6.1.1 Detailed Description . . . . .	117
6.2 libcg/cg_device.h File Reference . . . . .	118
6.2.1 Detailed Description . . . . .	119
6.3 libcg/cg_dust_sensor.h File Reference . . . . .	119
6.3.1 Detailed Description . . . . .	120
6.4 libcg/cg_general.h File Reference . . . . .	120
6.4.1 Detailed Description . . . . .	122
6.5 libcg/cg_gps.h File Reference . . . . .	122
6.5.1 Detailed Description . . . . .	123
6.6 libcg/cg_hal.h File Reference . . . . .	123
6.6.1 Detailed Description . . . . .	123

6.7 libcg/cg_net.h File Reference . . . . .	123
6.7.1 Detailed Description . . . . .	125
6.7.2 Macro Definition Documentation . . . . .	126
6.7.2.1 CG_MAX_PING_ADDRESSES . . . . .	126
6.7.2.2 CG_MAX_ADDRESS_LENGTH . . . . .	126
6.8 libcg/cg_sms.h File Reference . . . . .	126
6.8.1 Detailed Description . . . . .	126
6.9 libcg/cg_ui.h File Reference . . . . .	126
6.9.1 Detailed Description . . . . .	127
6.10 libcg/cg_upgrade.h File Reference . . . . .	127
6.10.1 Detailed Description . . . . .	127
6.11 libcg/cg_wlan.h File Reference . . . . .	127
6.11.1 Detailed Description . . . . .	128
6.12 libcg/cg_wlan_sniffer.h File Reference . . . . .	129
6.12.1 Detailed Description . . . . .	129
6.13 libcg/cg_wwan.h File Reference . . . . .	129
6.13.1 Detailed Description . . . . .	132
6.14 libcg/cgate.h File Reference . . . . .	132
6.14.1 Detailed Description . . . . .	133
6.14.2 Enumeration Type Documentation . . . . .	133
6.14.2.1 cg_status_t . . . . .	133
6.14.3 Function Documentation . . . . .	133
6.14.3.1 cg_get_last_error . . . . .	133



# Chapter 1

## Module Index

### 1.1 Modules

Here is a list of all modules:

Configuration functions . . . . .	9
Device Management . . . . .	13
DUST_SENSOR . . . . .	19
General functions . . . . .	22
GPS functions . . . . .	33
HAL . . . . .	39
Networking . . . . .	40
SMS . . . . .	52
UI . . . . .	55
Upgrade API . . . . .	60
WLAN . . . . .	62
WLAN_SNIFFER . . . . .	68
WWAN . . . . .	70



# Chapter 2

## Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">cg_board_t</a>	Description of a pluggable expansion board . . . . .	89
<a href="#">cg_conf_list</a>	.....	90
<a href="#">cg_conf_list_t</a>	Structure to return a list of values from the configuration . . . . .	90
<a href="#">cg_config_t</a>	Structure to reference the configuration context . . . . .	90
<a href="#">cg_date_time_t</a>	Structure holding local or UTC time value . . . . .	91
<a href="#">cg_device_t</a>	Basic properties of a hardware device present in the system . . . . .	91
<a href="#">cg_dust_sensor_data_t</a>	Output parameters of the dust sensor logger . . . . .	92
<a href="#">cg_gps_certificate_t</a>	Root certificate to validate the SUPL server . . . . .	93
<a href="#">cg_gps_t</a>	Structure containing all GPS properties . . . . .	93
<a href="#">cg_hal_serial_cfg_t</a>	Generic configuration settings for a serial interface . . . . .	94
<a href="#">cg_hal_serial_rs485_cfg_t</a>	Serial RS485-specific configuration settings . . . . .	95
<a href="#">cg_ip_addr_t</a>	Structure describing a IP address . . . . .	95
<a href="#">cg_key_t</a>	A single key of a key/value pair . . . . .	96
<a href="#">cg_kvpair_t</a>	Key/value pair . . . . .	96
<a href="#">cg_net_cfg_t</a>	Generic configuration settings for a network interface . . . . .	96
<a href="#">cg_net_dhcp_config_t</a>	Structure used to configure DHCP server functionality . . . . .	97
<a href="#">cg_net_if_t</a>	Data structure describing a network interface in the system . . . . .	98
<a href="#">cg_net_ip_config_t</a>	Structure describing IP configuration of an interface . . . . .	99
<a href="#">cg_net_lan_cfg_t</a>	LAN-specific configuration settings for a network interface . . . . .	100

<code>cg_net_leasetime_t</code>	Structure describing a leasetime . . . . .	100
<code>cg_net_stats_t</code>	Structure that contains the data usage statistics for an interface . . . . .	101
<code>cg_net_wan_cfg_t</code>	WAN-specific configuration settings for a network interface . . . . .	102
<code>cg_net_watchdog_settings_t</code>	Settings structure for the connection watchdog . . . . .	103
<code>cg_slot_list_t</code>	List of present slots . . . . .	104
<code>cg_slot_t</code>	Device firmware slot description . . . . .	105
<code>cg_timezone_t</code>	Time zone splitted up in its components. <a href="https://www.gnu.org/software/libc/manual/html-node/TZ-Variable.html">https://www.gnu.org/software/libc/manual/html-node/TZ-Variable.html</a> . . . . .	105
<code>cg_wlan_auth_params_t</code>	Authentication parameters for a WLAN client/access point . . . . .	106
<code>cg_wlan_network_t</code>	Description of a WLAN network . . . . .	107
<code>cg_wlan_sniffer_params</code>	. . . . .	107
<code>cg_wwan_diag_param_t</code>	Diagnostic parameter information . . . . .	108
<code>cg_wwan_img_list_t</code>	Structure containing a list of available images . . . . .	109
<code>cg_wwan_img_t</code>	Image name . . . . .	110
<code>cg_wwan_location_info_t</code>	Struct defining the location of the base transceiver station . . . . .	110
<code>cg_wwan_network_list_t</code>	Structure for a list of networks . . . . .	111
<code>cg_wwan_network_settings_t</code>	Struct used to set/get the mobile network settings . . . . .	111
<code>cg_wwan_network_t</code>	Structure defining a network . . . . .	112
<code>cg_wwan_pin_state_t</code>	PIN code status of the SIM inserted in the device . . . . .	113
<code>cg_wwan_reg_state_t</code>	Network registration state . . . . .	114

# Chapter 3

## File Index

### 3.1 File List

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

<a href="#">libcg/cg_conf.h</a>	Header file containing definitions for functions related to setting/getting configuration items . . . . .	117
<a href="#">libcg/cg_device.h</a>	Header file containing definition of all device management related function calls . . . . .	118
<a href="#">libcg/cg_dust_sensor.h</a>	Header file containing all definitions/functions related to dust sensor logger . .	119
<a href="#">libcg/cg_general.h</a>	Header file containing definitions for miscellaneous functions . . . . .	120
<a href="#">libcg/cg_gps.h</a>	Header file containing definitions for all GPS-related functionality . . . . .	122
<a href="#">libcg/cg_hal.h</a>	Header file containing all definitions/functions related specific to the Hardware Abstraction Layer . . . . .	123
<a href="#">libcg/cg_net.h</a>	Header file containing all definitions related to working with network interfaces, connecting, disconnecting.. . . . .	123
<a href="#">libcg/cg_sms.h</a>	Header file containing all definitions/functions related specific to SMS . . . . .	126
<a href="#">libcg/cg_ui.h</a>	Header file containing definitions for all UI-related functionality . . . . .	126
<a href="#">libcg/cg_upgrade.h</a>	Header file containing definitions for upgrade API . . . . .	127
<a href="#">libcg/cg_wlan.h</a>	Header file containing all definitions/functions related specific to WLAN networking . . . . .	127
<a href="#">libcg/cg_wlan_sniffer.h</a>	Header file containing all definitions/functions related to wlan sniffing . . . . .	129
<a href="#">libcg/cg_wwan.h</a>	Header file containing all definitions/functions related specific to wwan (3G) networking . . . . .	129
<a href="#">libcg/cgate.h</a>	Main SDK header file . . . . .	132



# 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	<i>ctx</i>	The configuration context previously opened with cg_conf_open.
in	<i>section</i>	The name of the section where we are setting a value. If this section does not exist, it will be created.
in	<i>option</i>	The name of the option for which we are setting a value.
in	<i>value</i>	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	<i>ctx</i>	The configuration context previously opened with cg_conf_open.
in	<i>section</i>	The name of the section where we are getting a value from.
in	<i>option</i>	The name of the option from which we are getting a value.
out	<i>value</i>	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	<i>ctx</i>	The configuration context previously opened with cg_conf_open.
in	<i>name</i>	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-NUL.

**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 <a href="#">cg_device_register_notification</a>

### 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 <a href="#">cg_device_claim</a>

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	<i>location_mask</i>	A mask of cg_location_t values to indicate the locations of the devices to list
in	<i>device_mask</i>	A mask of cg_device_type_t values to indicate the types of devices to list
out	<i>num_devices</i>	Number of device structures returned
out	<i>devices</i>	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	<i>device</i>	The device to claim
in	<i>cb</i>	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	<i>context</i>	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	<i>device</i>	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 <code>cg_device_type_t</code> 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 or `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 <code>boards</code> . Caller needs to free the the array.

Returns

`CG_STATUS_OK` if successful or `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 <code>devices</code> . Caller needs to free the the array.

Returns

CG\_STATUS\_OK if successful or 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	<code>dust</code>	sensor data
in	<code>context</code>	Context as set in <code>cg_dust_sensor_register_data_callback</code>

### 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 occurred

**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](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_ERROR`,  
`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(2,0)`
- `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 <code>cg_prov_kvpair_register_update_callback</code>
in	update	Pointer to a <code>cg_kvpair_t</code> containing the new/updated or deleted key/value pair. If deleted, <code>value_len</code> is 0. This <code>cg_kvpair_t</code> must be freed()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 <code>cg_system_sync_ntp</code> 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

<code>name</code>	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

<code>out</code>	<code>level</code>	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 successful, 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	<code>request_in</code>	A <code>cg_kvpair_t</code> with the key and ns members filled in.
out	<code>If</code>	successful, the key, ns, value_len and value fields are filled in in a <code>cg_kvpair_t</code> 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	<code>request</code>	A <code>cg_kvpair_t</code> 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	<code>request</code>	A <code>cg_kvpair_t</code> 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 <a href="#">cg_key_t</a> , 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 successful. 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 parameter 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 parameter 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 parameter 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 parameter 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 parameter 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 <code>cg_gps_register_nmea_callback</code>
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,: url,: cert_count: cert	disable, enable secure or enable unsecure. URL from SUPL server. cert_count: number of root certificates that follow (maximum C_G_MAX_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 contacted 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

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

example using google supl 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	<code>dev_name</code>	SERIAL device name
in	<code>config</code>	parameters
out	<code>reboot_needed</code>	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( `if_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_WWWAN` = 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 ifts)`
- `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( iff\_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 <a href="#">cg.net_if_status_t</a> 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	<code>num_interfaces</code>	Integer which will be set to the number of interfaces
out	<code>interfaces</code>	interfaces will be set to an array of <code>cg.net_if_t</code> . The number of elements in the array equal to <code>num_interfaces</code> . 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	<code>dev_name</code>	The interface name for which to retrieve the interface info.
out	<code>interface</code>	interface will point to a <code>cg.net_if_t</code> object when the device is found, NULL otherwise. Caller needs to free the <code>cg.net_if_t</code> 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 ifts )`

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	ifds	ifds will be set to an array of <a href="#">cg.net.if.t</a> . 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 <a href="#">cg.net.if.t</a> 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 <a href="#">cg.net.if.t</a> 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, iff.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	<code>dev_name</code>	The interface name for which the data counter needs to be reset.
in	<code>disabled</code>	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	<code>dev_name</code>	The interface name for which the data counter needs to be reset.
out	<code>disabled</code>	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 parameter 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_P-URGE }`

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 <a href="#">cg_ui_register_json_callback</a>
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 <a href="#">cg_ui_register_get_callback</a>
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 is 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 <a href="http://cloudgate/api/libcg/identifier">http://cloudgate/api/libcg/identifier</a>
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 <a href="http://cloudgate/api/libcg/identifier">http://cloudgate/api/libcg/identifier</a>
----	------------	--

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 <a href="http://cloudgate/api/libcg/identifier">http://cloudgate/api/libcg/identifier</a>
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	<code>identifier</code>	Unique identifier used as the last part of the URL <code>http://cloudgate/api/libcg/identifier</code>
----	-------------------------	---

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	<code>Call</code>	function of type <code>cg_ui_session_cb</code>
----	-------------------	--

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_upgrade_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	<code>handle</code> :	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 committed 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	<code>status</code>	Status of the operation. CG_STATUS_SUCCESS or CG_STATUS_ERROR.
in	<code>dev_name</code>	WLAN STA device name.
in	<code>num_entries</code>	Number of entries in <code>networks</code> .
in	<code>networks</code>	Array of <code>cg_wlan_network_t</code> structures.
in	<code>context</code>	Context parameter passed on to <code>cg_wlan_sta_scan_networks</code> .

#### 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 <a href="#">cg_wlan_sta_scan_networks</a> . 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	<code>dev_name</code>	WLAN STA device name.
out	<code>num_entries</code>	When successful, the number of entries in <code>networks</code> .
out	<code>networks</code>	When successful, will be set to an array of <code>cg_wlan_network_t</code> , 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	<code>dev_name</code>	WLAN STA device name.
in	<code>ssid</code>	SSID of the network.
in	<code>auth</code>	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	<code>dev_name</code>	WLAN AP device name.
in	<code>ssid</code>	The SSID to set for the access point.
in	<code>broadcast</code>	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	<code>dev_name</code>	WLAN AP device name.
in	<code>auth</code>	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	<code>dev_name</code>	WLAN AP device name.
out	<code>auth</code>	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	<code>window_size,: </code>	value in seconds.
in	<code>channel,: </code>	wifi channel to log.
in	<code>max_file_size,: </code>	maximum file size in bytes.
in	<code>max_nbr_files,: </code>	maximum number of files created.
in	<code>monitoring_file_path,: </code>	file path for monitoring log files.
in	<code>detailed_monitoring,: </code>	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_PIN`, `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_EVDORELO` = 0x4000, `CG_SYS_EVDOREVA` = 0x8000 }

Enumeration describing the possible mobile system 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\_UTMS)*
- 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_COMMITED` = 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	<i>status</i>	Status of network scan. CG_STATUS_OK or CG_STATUS_ERROR.
in	<i>list</i>	The network list. This list is allocated by the caller of the callback and will be freed when the callback returns.
in	<i>context</i>	The context which was passed on in <code>cg_wwan_search_networks</code>

#### 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	<i>dev_name</i>	The device for which the registration state is updated
in	<i>state</i>	The updated state. Structure allocated by caller and will be free when the callback returns.
in	<i>context</i>	The context which was passed on in <code>cg_wwan_register_reg_state_notification</code>

### 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\_COMMITED** 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, <sup>97</sup>

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 <a href="#">cg_wwan_diag_param_t</a> , allocated by the callee. The caller will be responsible for freeing the array only, not memory pointed to by members of <a href="#">cg_wwan_diag_param_t</a> .

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	<code>dev_name</code>	WWAN device name
in	<code>wwan_network_settings</code>	Pointer to struct of type <code>cg_wwan_network_settings_t</code> 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	<code>dev_name</code>	WWAN device name
out	<code>wwan_network_settings</code>	Callee will set <code>wwan_network_settings</code> to a <code>cg_wwan_network_settings_t</code> 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 ( <a href="#">cg_wwan_search_networks</a> ).

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	<i>dev_name</i>	WWAN device name
in	<i>enabled</i>	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	<i>dev_name</i>	WWAN device name
out	<i>enabled</i>	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	<i>dev_name</i>	WWAN device name
out	<i>image_name</i>	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	<i>dev_name</i>	WWAN device name
out	<i>image_list</i>	List of images. <i>image_list</i> 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	<i>dev_name</i>	WWAN device name.
out	<i>location_info</i>	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 deleted length 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/libcg.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](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_ERROR`,  
`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(2,0)`
- `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( `if_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 ifts)`
- `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_P-`  
`URGE` }

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_EVDORELO` = 0x4000, `CG_SYS_EVDOREVA` = 0x8000 }

*Enumeration describing the possible mobile system 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_COMMITED` = 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

action  
    cg\_net\_watchdog\_settings\_t, 103

active  
    cg\_slot\_t, 105

addresses  
    cg\_net\_watchdog\_settings\_t, 104

amount  
    cg\_net\_leasetime\_t, 101

apn  
    cg\_wwan\_network\_settings\_t, 112

auth\_type  
    cg\_wlan\_network\_t, 107  
    cg\_wwan\_network\_settings\_t, 112

board\_id  
    cg\_board\_t, 89

bridge\_dev\_name  
    cg\_net\_lan\_cfg\_t, 100

byte  
    cg\_dust\_sensor\_data\_t, 92

CG\_DEV\_EVT\_ARRIVED  
    Device Management, 15

CG\_DEV\_EVT CLAIMED  
    Device Management, 15

CG\_DEV\_EVT\_RELEASED  
    Device Management, 15

CG\_DEV\_EVT REMOVED  
    Device Management, 15

CG\_DEV\_MINIVAN\_DEBUG  
    General functions, 25

CG\_DEV\_ROOT\_ACCESS  
    General functions, 25

CG\_DEVICE CLAIMED  
    Device Management, 14

CG\_DEVICE\_PRESENT  
    Device Management, 14

CG\_DEVTYPE\_ADC  
    Device Management, 14

CG\_DEVTYPE\_DAC  
    Device Management, 14

CG\_DEVTYPE\_GPIO  
    Device Management, 14

CG\_DEVTYPE\_NETWORK  
    Device Management, 14

CG\_DEVTYPE\_SERIAL  
    Device Management, 14

CG\_DHCP  
    Networking, 43

CG\_DUST\_SENSOR\_CONNECT  
    DUST\_SENSOR, 20

CG\_DUST\_SENSOR\_DISCONNECT  
    DUST\_SENSOR, 20

CG\_DUST\_SENSOR\_INVALID\_PARAMETER  
    DUST\_SENSOR, 20

CG\_DUST\_SENSOR\_NOT\_AVAILABLE  
    DUST\_SENSOR, 20

CG\_DUST\_SENSOR\_NOT\_AWAKE  
    DUST\_SENSOR, 20

CG\_DUST\_SENSOR\_POWER\_FAN\_ON  
    DUST\_SENSOR, 20

CG\_DUST\_SENSOR\_POWER\_LASER\_ON  
    DUST\_SENSOR, 20

CG\_DUST\_SENSOR\_POWER\_MAX  
    DUST\_SENSOR, 20

CG\_DUST\_SENSOR\_POWER\_OFF  
    DUST\_SENSOR, 20

CG\_DUST\_SENSOR\_RW\_ERROR  
    DUST\_SENSOR, 20

CG\_DUST\_SENSOR\_SPI\_ERROR  
    DUST\_SENSOR, 20

CG\_DUST\_SENSOR\_THREAD\_CREATE\_ERROR  
    DUST\_SENSOR, 20

CG\_DUST\_SENSOR\_VALID\_DATA  
    DUST\_SENSOR, 20

CG\_GPS\_ASSISTED\_CONNECTION\_FAILURE  
    GPS functions, 34

CG\_GPS\_ASSISTED\_CONNECTION\_SUCCESS  
    GPS functions, 34

CG\_GPS\_ASSISTED\_DISABLE  
    GPS functions, 35

CG\_GPS\_ASSISTED\_NON\_SECURE\_SUPPL  
    GPS functions, 35

CG\_GPS\_ASSISTED\_SECURE\_SUPPL  
    GPS functions, 35

CG\_GPS\_ENABLED  
    GPS functions, 34

CG\_GPS\_PRESENT  
    GPS functions, 34

CG\_HAL\_SERIAL\_RS232  
    HAL, 39

CG\_HAL\_SERIAL\_RS485  
    HAL, 39

CG\_LOC\_ALL  
    Device Management, 15

CG\_LOC\_MAIN  
    Device Management, 15

CG\_LOC\_SLOT1  
    Device Management, 15

CG.LOC\_SLOT2  
    Device Management, 15  
CG.LOC\_UNKNOWN  
    Device Management, 15  
CG.MAX\_ADDRESS.LENGTH  
    cg\_net.h, 126  
CG.MAX\_CERTIFICATES  
    GPS functions, 33  
CG.MAX\_IMAGE\_NAME  
    WWAN, 72  
CG.MAX\_LEN\_CERTIFICATE  
    GPS functions, 34  
CG.MAX\_PING\_ADDRESSES  
    cg\_net.h, 126  
CG.MAX\_SLOT\_INFO\_LEN  
    General functions, 23  
CG.MONITORING\_FILE\_PATH\_MAX\_LEN  
    WLAN\_SNIFTER, 68  
CG.NET\_INTERFACE\_CAN\_CONNECT  
    Networking, 42  
CG.NET\_INTERFACE\_CONFIGURED  
    Networking, 42  
CG.NET\_INTERFACE\_CONNECTED  
    Networking, 42  
CG.NET\_INTERFACE\_ENABLED  
    Networking, 42  
CG.NET\_INTERNET\_CONNECTED  
    Networking, 42  
CG.NET\_INTERNET\_CONNECTING  
    Networking, 42  
CG.NET\_INTERNET\_DISCONNECTED  
    Networking, 42  
CG.NET\_INTERNET\_ON\_DEMAND  
    Networking, 42  
CG.NET\_MODE\_ALWAYS\_ON  
    Networking, 43  
CG.NET\_MODE\_ON\_DEMAND  
    Networking, 43  
CG.NET\_STRATEGY\_MANUAL  
    Networking, 43  
CG.NET\_STRATEGY\_PRIORITY  
    Networking, 43  
CG.NET\_TYPE\_BRIDGE  
    Networking, 43  
CG.NET\_TYPE\_IPSEC  
    Networking, 43  
CG.NET\_TYPE\_NIC  
    Networking, 43  
CG.NET\_TYPE\_WLAN  
    Networking, 43  
CG.NET\_TYPE\_WLAN\_AP  
    Networking, 43  
CG.NET\_TYPE\_WWAN  
    Networking, 43  
CG.NET\_UNIT\_DAY  
    Networking, 44  
CG.NET\_UNIT\_HOUR  
    Networking, 44  
CG.NET\_UNIT\_MINUTE  
    Networking, 44  
CG.NET\_ZONE\_LAN  
    Networking, 43  
CG.NET\_ZONE\_WAN  
    Networking, 43  
CG.NETWORK\_NAME\_SIZE  
    WWAN, 72  
CG.PASSWORD\_MAX\_LENGTH  
    WLAN, 63  
CG.PROV\_CHECKING  
    General functions, 24  
CG.PROV\_ERROR  
    General functions, 24  
CG.PROV\_FLASHING  
    General functions, 24  
CG.PROV\_OK  
    General functions, 24  
CG.PROV\_UNPROVISIONED  
    General functions, 24  
CG.PROV\_UNREGISTERED  
    General functions, 24  
CG.SDK\_API\_LEVEL  
    General functions, 23  
CG.SIM\_SWITCH\_MODE\_AUTO  
    WWAN, 75  
CG.SIM\_SWITCH\_MODE\_SIM1  
    WWAN, 75  
CG.SIM\_SWITCH\_MODE\_SIM2  
    WWAN, 75  
CG.SIM\_SWITCH\_STATE\_SIM1  
    WWAN, 75  
CG.SIM\_SWITCH\_STATE\_SIM2  
    WWAN, 75  
CG.SLOT\_BASE\_CONFIG  
    General functions, 24  
CG.SLOT\_BASE\_IMAGE  
    General functions, 24  
CG.SLOT\_BOOTLOADER  
    General functions, 24  
CG.SLOT\_CUSTOMER  
    General functions, 24  
CG.SLOT\_CUSTOMER\_CONFIG  
    General functions, 24  
CG.SLOT\_CUSTOMER\_KEY  
    General functions, 24  
CG.SSID\_MAX\_LENGTH  
    WLAN, 63  
CGSTATIC  
    Networking, 43  
CG.STATUS\_DOWNGRADE\_PREVENTION  
    cgate.h, 133  
CG.STATUS\_ERROR  
    cgate.h, 133  
CG.STATUS\_INVALID\_PARAMETER  
    cgate.h, 133  
CG.STATUS\_OK  
    cgate.h, 133

CG\_STATUS\_RESOURCE\_BUSY  
cgate.h, 133

CG\_STATUS\_RESOURCE\_UNAVAILABLE  
cgate.h, 133

CG\_SYSTEM\_ARM\_IGNITION\_SENSE  
General functions, 25

CG\_SYSTEM\_TIMED\_WAKEUP  
General functions, 25

CG\_WLAN\_AUTH\_OPEN  
WLAN, 63

CG\_WLAN\_AUTH\_WEP  
WLAN, 63

CG\_WLAN\_AUTH\_WPA2\_ENT  
WLAN, 63

CG\_WLAN\_AUTH\_WPA2\_PSK  
WLAN, 63

CG\_WLAN\_AUTH\_WPA\_ENT  
WLAN, 63

CG\_WLAN\_AUTH\_WPA\_PSK  
WLAN, 63

CG\_WLAN\_AUTH\_WPA\_WPA2\_ENT  
WLAN, 63

CG\_WLAN\_AUTH\_WPA\_WPA2\_PSK  
WLAN, 63

CG\_WWAN\_ACT\_STATUS\_ACTIVATED  
WWAN, 74

CG\_WWAN\_ACT\_STATUS\_CONNECTED  
WWAN, 74

CG\_WWAN\_ACT\_STATUS\_CONNECTION  
WWAN, 74

CG\_WWAN\_ACT\_STATUS\_NOT\_ACTIVATED  
WWAN, 74

CG\_WWAN\_ACT\_STATUS\_OTASP\_AUTH  
WWAN, 74

CG\_WWAN\_ACT\_STATUS\_OTASP\_COMMITED  
WWAN, 74

CG\_WWAN\_ACT\_STATUS\_OTASP\_IMSI  
WWAN, 74

CG\_WWAN\_ACT\_STATUS\_OTASP\_MDN  
WWAN, 74

CG\_WWAN\_ACT\_STATUS\_OTASP\_NAM  
WWAN, 74

CG\_WWAN\_ACT\_STATUS\_OTASP\_PRL  
WWAN, 74

CG\_WWAN\_ACT\_STATUS\_OTASP\_SPC  
WWAN, 74

CG\_WWAN\_AUTH\_TYPE\_AUTO  
WWAN, 75

CG\_WWAN\_AUTH\_TYPE\_CHAP  
WWAN, 75

CG\_WWAN\_AUTH\_TYPE\_NONE  
WWAN, 75

CG\_WWAN\_AUTH\_TYPE\_PAP  
WWAN, 75

CG\_WWAN\_CREG\_DENIED  
WWAN, 74

CG\_WWAN\_CREG\_NOT\_REG  
WWAN, 74

CG\_WWAN\_CREG\_REG  
WWAN, 74

CG\_WWAN\_CREG\_ROAMING  
WWAN, 74

CG\_WWAN\_CREG\_SEARCHING  
WWAN, 74

CG\_WWAN\_MODE\_CDMA  
WWAN, 74

CG\_WWAN\_MODE\_UMTS  
WWAN, 74

CG\_WWAN\_PIN\_LOCKED\_PN  
WWAN, 73

CG\_WWAN\_PIN\_READY  
WWAN, 73

CG\_WWAN\_PIN\_SIM\_FAILURE  
WWAN, 73

CG\_WWAN\_PIN\_SIM\_NOT\_INSERTED  
WWAN, 73

CG\_WWAN\_PIN\_SIM\_NOT\_SUPPORTED  
WWAN, 73

CG\_WWAN\_PIN\_SIM\_PIN  
WWAN, 73

CG\_WWAN\_PIN\_SIM\_PUK  
WWAN, 73

CG\_WWAN\_PIN\_UNKNOWN  
WWAN, 73

cell\_id  
cg\_wwan\_location\_info\_t, 111

cert  
cg\_gps\_certificate\_t, 93

cg\_board.t, 89  
board\_id, 89  
hw\_ver, 89  
slot\_id, 89

cg\_conf\_add\_section  
Configuration functions, 11

cg\_conf\_close  
Configuration functions, 10

cg\_conf\_del  
Configuration functions, 11

cg\_conf\_get  
Configuration functions, 11

cg\_conf\_get\_options  
Configuration functions, 12

cg\_conf\_get\_sections  
Configuration functions, 12

cg\_conf\_list, 90  
number, 90  
values, 90

cg\_conf\_list.t, 90

cg\_conf\_open  
Configuration functions, 9

cg\_conf\_revert  
Configuration functions, 10

cg\_conf\_set  
Configuration functions, 10

cg\_config.t, 90

cg\_date\_time.t, 91

cg\_deinit  
    General functions, 25

cg\_dev\_feature\_t  
    General functions, 24

cg\_dev\_name\_t  
    Networking, 42

cg\_device\_board\_device\_list  
    Device Management, 17

cg\_device\_board\_list  
    Device Management, 17

cg\_device\_claim  
    Device Management, 16

cg\_device\_deregister\_notification  
    Device Management, 17

cg\_device\_event\_t  
    Device Management, 15

cg\_device\_list  
    Device Management, 15

cg\_device\_list\_by\_location  
    Device Management, 15

cg\_device\_register\_notification  
    Device Management, 16

cg\_device\_release  
    Device Management, 16

cg\_device\_status\_t  
    Device Management, 14

cg\_device\_t, 91

- device\_location, 92
- device\_name, 91
- device\_type, 91
- friendly\_name, 91
- status, 92

cg\_device\_type\_t  
    Device Management, 14

cg\_dust\_sensor\_data\_cb\_t  
    DUST\_SENSOR, 19

cg\_dust\_sensor\_data\_t, 92

- byte, 92
- status, 92

cg\_dust\_sensor\_deregister\_data\_callback  
    DUST\_SENSOR, 21

cg\_dust\_sensor\_power\_t  
    DUST\_SENSOR, 20

cg\_dust\_sensor\_register\_data\_callback  
    DUST\_SENSOR, 21

cg\_dust\_sensor\_set\_power  
    DUST\_SENSOR, 20

cg\_dust\_sensor\_start  
    DUST\_SENSOR, 20

cg\_dust\_sensor\_status\_t  
    DUST\_SENSOR, 20

cg\_dust\_sensor\_stop  
    DUST\_SENSOR, 20

cg\_get\_api\_level  
    General functions, 25

cg\_get\_last\_error  
    cgate.h, 133

cg\_get\_serial\_number

General functions, 25

cg\_get\_slot\_list  
    General functions, 28

cg\_gps\_assisted\_cb\_t  
    GPS functions, 34

cg\_gps\_assisted\_connection\_status\_t  
    GPS functions, 34

cg\_gps\_assisted\_function\_t  
    GPS functions, 34

cg\_gps\_certificate\_t, 93

- cert, 93
- len, 93

cg\_gps\_deregister\_nmea\_callback  
    GPS functions, 37

cg\_gps\_deregister\_supl\_connection\_status\_callback  
    GPS functions, 37

cg\_gps\_get\_assisted  
    GPS functions, 36

cg\_gps\_get\_status  
    GPS functions, 35

cg\_gps\_nmea\_cb\_t  
    GPS functions, 34

cg\_gps\_register\_nmea\_callback  
    GPS functions, 37

cg\_gps\_register\_supl\_connection\_status\_callback  
    GPS functions, 37

cg\_gps\_set\_assisted  
    GPS functions, 36

cg\_gps\_set\_enabled  
    GPS functions, 35

cg\_gps\_set\_reporting\_interval  
    GPS functions, 35

cg\_gps\_status\_t  
    GPS functions, 34

cg\_gps\_t, 93

- reporting\_interval, 94
- status, 94

cg\_hal\_serial\_cfg\_t, 94

- param, 94
- rs485, 94
- serial\_mode, 94

cg\_hal\_serial\_mode\_t  
    HAL, 39

cg\_hal\_serial\_rs485\_cfg\_t, 95

- enable\_termination, 95

cg\_hal\_serial\_set\_config  
    HAL, 39

cg\_init  
    General functions, 25

cg\_ip\_addr\_t, 95

cg\_key\_t, 96

cg\_kvpair\_t, 96

cg\_location\_t  
    Device Management, 14

cg\_net.h

- CG\_MAX\_ADDRESS\_LENGTH, 126
- CG\_MAX\_PING\_ADDRESSES, 126

cg\_net\_cfg\_t, 96  
  conf, 97  
  ip\_config, 97  
  zone, 97  
cg\_net\_conn\_strat\_t  
  Networking, 43  
cg\_net\_connect  
  Networking, 47  
cg\_net\_datacounter\_get\_enabled  
  Networking, 49  
cg\_net\_datacounter\_get\_stats  
  Networking, 50  
cg\_net\_datacounter\_reset\_stats  
  Networking, 49  
cg\_net\_datacounter\_set\_enabled  
  Networking, 49  
cg\_net\_deregister\_interface\_events  
  Networking, 46  
cg\_net\_deregister\_internet\_events  
  Networking, 48  
cg\_net\_dhcp\_config\_t, 97  
  dns1, 98  
  dns2, 98  
  enabled, 98  
  ip\_end, 98  
  ip\_start, 98  
  leasetime, 98  
cg\_net\_disconnect  
  Networking, 47  
cg\_net\_get\_conn\_priority\_list  
  Networking, 47  
cg\_net\_get\_conn\_strat  
  Networking, 47  
cg\_net\_get\_disabled\_at\_boot  
  Networking, 51  
cg\_net\_get\_interface  
  Networking, 44  
cg\_net\_get\_interface\_by\_type  
  Networking, 44  
cg\_net\_get\_interface\_list  
  Networking, 44  
cg\_net\_get\_internet\_status  
  Networking, 48  
cg\_net\_get\_manual\_conn\_device  
  Networking, 46  
cg\_net\_get\_mtu  
  Networking, 50  
cg\_net\_get\_watchdog  
  Networking, 48  
cg\_net\_if\_status\_t  
  Networking, 42  
cg\_net\_if\_t, 98  
  config, 99  
  dev\_name, 99  
  status, 99  
  type, 99  
cg\_net\_if\_type\_t  
  Networking, 43  
cg\_net\_interface\_set\_enabled  
  Networking, 45  
cg\_net\_internet\_status\_t  
  Networking, 42  
cg\_net\_ip\_config\_t, 99  
  ip\_addr, 99  
  mtu, 99  
  netmask, 99  
cg\_net\_ip\_config\_type\_t  
  Networking, 43  
cg\_net\_lan\_cfg\_t, 100  
  bridge\_dev\_name, 100  
  dhcp\_config, 100  
cg\_net\_lease\_unit\_t  
  Networking, 43  
cg\_net\_leasetime\_t, 100  
  amount, 101  
  unit, 101  
cg\_net\_mode\_t  
  Networking, 43  
cg\_net\_register\_interface\_events  
  Networking, 45  
cg\_net\_register\_internet\_events  
  Networking, 48  
cg\_net\_set\_config  
  Networking, 45  
cg\_net\_set\_disabled\_at\_boot  
  Networking, 50  
cg\_net\_set\_manual\_conn\_strat  
  Networking, 46  
cg\_net\_set\_mtu  
  Networking, 50  
cg\_net\_set\_priority\_conn\_strat  
  Networking, 46  
cg\_net\_set\_watchdog  
  Networking, 48  
cg\_net\_stats\_t, 101  
  rx\_bytes, 101  
  rx\_packets, 101  
  tx\_bytes, 101  
  tx\_packets, 101  
cg\_net\_wan\_cfg\_t, 102  
  data\_timeout, 102  
  dns1, 102  
  dns2, 103  
  gw, 102  
  ip\_config\_type, 102  
  mode, 102  
cg\_net\_watchdog\_settings\_t, 103  
  action, 103  
  addresses, 104  
  enabled, 103  
  interval, 103  
  num\_addresses, 104  
  use\_ping, 103  
cg\_net\_zone\_t  
  Networking, 42  
cg\_prov\_check\_update

General functions, 26  
cg\_prov\_get\_status  
    General functions, 26  
cg\_prov\_kvpair\_del  
    General functions, 27  
cg\_prov\_kvpair\_deregister\_update\_callback  
    General functions, 28  
cg\_prov\_kvpair\_get  
    General functions, 27  
cg\_prov\_kvpair\_list  
    General functions, 27  
cg\_prov\_kvpair\_register\_update\_callback  
    General functions, 28  
cg\_prov\_kvpair\_set  
    General functions, 27  
cg\_prov\_kvpair\_update\_cb\_t  
    General functions, 24  
cg\_prov\_set\_automatic  
    General functions, 26  
cg\_prov\_status\_t  
    General functions, 24  
cg\_reset\_peripherals  
    General functions, 29  
cg\_reset\_system  
    General functions, 29  
cg\_set\_dev\_mode  
    General functions, 29  
cg\_slot\_list\_t, 104  
    num\_slots, 104  
    slots, 104  
cg\_slot\_t, 105  
    active, 105  
    size, 105  
    type, 105  
    uid, 105  
    version, 105  
cg\_slot\_type\_t  
    General functions, 24  
cg\_sms\_cb\_t  
    SMS, 52  
cg\_sms\_deregister\_new\_sms  
    SMS, 53  
cg\_sms\_get\_smsc  
    SMS, 52  
cg\_sms\_register\_new\_sms  
    SMS, 53  
cg\_sms\_send  
    SMS, 53  
cg\_sms\_set\_smsc  
    SMS, 52  
cg\_status\_t  
    cgate.h, 133  
cg\_system\_boot\_condition\_t  
    General functions, 25  
cg\_system\_get\_date\_time  
    General functions, 31  
cg\_system\_get\_ntp\_server  
    General functions, 31  
cg\_system\_log  
    General functions, 29  
cg\_system\_power\_down  
    General functions, 30  
cg\_system\_set\_date\_time  
    General functions, 30  
cg\_system\_set\_ntp\_server  
    General functions, 31  
cg\_system\_set\_timed\_wakeup  
    General functions, 30  
cg\_system\_set\_timezone  
    General functions, 31  
cg\_system\_sync\_ntp  
    General functions, 32  
cg\_system\_sync\_ntp\_cb\_t  
    General functions, 24  
cg\_system\_sync\_ntp\_deregister\_callback  
    General functions, 32  
cg\_system\_sync\_ntp\_register\_callback  
    General functions, 32  
cg\_timezone\_t, 105  
cg\_ui\_deregister\_get\_callback  
    UI, 59  
cg\_ui\_deregister\_json\_callback  
    UI, 58  
cg\_ui\_deregister\_page  
    UI, 57  
cg\_ui\_deregister\_session\_callback  
    UI, 59  
cg\_ui\_get\_cb\_t  
    UI, 57  
cg\_ui\_json\_cb\_t  
    UI, 56  
cg\_ui\_register\_get\_callback  
    UI, 58  
cg\_ui\_register\_json\_callback  
    UI, 58  
cg\_ui\_register\_page  
    UI, 57  
cg\_ui\_register\_session\_callback  
    UI, 59  
cg\_ui\_session\_cb  
    UI, 57  
cg\_upgrade\_data  
    Upgrade API, 60  
cg\_upgrade\_start  
    Upgrade API, 60  
cg\_upgrade\_stop  
    Upgrade API, 61  
cg\_upgrade\_t  
    Upgrade API, 60  
cg\_wlan\_ap\_get\_auth\_params  
    WLAN, 67  
cg\_wlan\_ap\_get\_channel  
    WLAN, 66  
cg\_wlan\_ap\_get\_ssid  
    WLAN, 66  
cg\_wlan\_ap\_set\_auth\_params

WLAN, 67  
cg\_wlan\_ap\_set\_channel  
    WLAN, 66  
cg\_wlan\_ap\_set\_ssid  
    WLAN, 65  
cg\_wlan\_auth\_params\_t, 106  
    password, 106  
    type, 106  
cg\_wlan\_auth\_t  
    WLAN, 63  
cg\_wlan\_network\_list\_cb\_t  
    WLAN, 63  
cg\_wlan\_network\_t, 107  
    auth\_type, 107  
    channel, 107  
    signal\_strength, 107  
    ssid, 107  
cg\_wlan\_sniffer\_params, 107  
    channel, 108  
    detailed\_monitoring, 108  
    max\_file\_size, 108  
    max\_nbr\_files, 108  
    monitoring\_file\_path, 108  
    window\_size, 108  
cg\_wlan\_sniffer\_start  
    WLAN\_SNIFTER, 68  
cg\_wlan\_sniffer\_stop  
    WLAN\_SNIFTER, 69  
cg\_wlan\_sta\_get\_connected\_network  
    WLAN, 64  
cg\_wlan\_sta\_get\_network\_list  
    WLAN, 65  
cg\_wlan\_sta\_remove\_network  
    WLAN, 65  
cg\_wlan\_sta\_scan\_networks  
    WLAN, 63  
cg\_wlan\_sta\_set\_network  
    WLAN, 64  
cg\_wwan\_act\_state\_t  
    WWAN, 74  
cg\_wwan\_activate  
    WWAN, 84  
cg\_wwan\_auth\_type\_t  
    WWAN, 75  
cg\_wwan\_change\_pin  
    WWAN, 82  
cg\_wwan\_creg\_status\_t  
    WWAN, 74  
cg\_wwan\_deregister\_reg\_state\_notification  
    WWAN, 78  
cg\_wwan\_diag\_param\_t, 108  
    name, 109  
    value, 109  
cg\_wwan\_get\_activation\_state  
    WWAN, 83  
cg\_wwan\_get\_active\_primary\_dev  
    WWAN, 75  
cg\_wwan\_get\_diagnostics  
    WWAN, 78  
cg\_wwan\_get\_iccid  
    WWAN, 81  
cg\_wwan\_get\_imei  
    WWAN, 80  
cg\_wwan\_get\_imsi  
    WWAN, 80  
cg\_wwan\_get\_location\_info  
    WWAN, 87  
cg\_wwan\_get\_meid  
    WWAN, 83  
cg\_wwan\_get\_mode  
    WWAN, 76  
cg\_wwan\_get\_network\_settings  
    WWAN, 79  
cg\_wwan\_get\_nw\_selection\_mode  
    WWAN, 82  
cg\_wwan\_get\_phone\_number  
    WWAN, 79  
cg\_wwan\_get\_prl\_network\_update  
    WWAN, 85  
cg\_wwan\_get\_prl\_version  
    WWAN, 84  
cg\_wwan\_get\_radio  
    WWAN, 77  
cg\_wwan\_get\_reg\_state  
    WWAN, 77  
cg\_wwan\_get\_serial  
    WWAN, 78  
cg\_wwan\_get\_sim\_switch  
    WWAN, 87  
cg\_wwan\_gobi\_get\_active\_image  
    WWAN, 85  
cg\_wwan\_gobi\_get\_image\_list  
    WWAN, 85  
cg\_wwan\_gobi\_get\_image\_type  
    WWAN, 86  
cg\_wwan\_gobi\_set\_active\_image  
    WWAN, 86  
cg\_wwan\_img\_list\_t, 109  
    current, 109  
    entries, 109  
    images, 109  
cg\_wwan\_img\_t, 110  
    name, 110  
cg\_wwan\_location\_info\_t, 110  
    cell\_id, 111  
    lac, 110  
    tac, 110  
cg\_wwan\_mode\_t  
    WWAN, 73  
cg\_wwan\_modem\_reboot  
    WWAN, 86  
cg\_wwan\_network\_list\_cb\_t  
    WWAN, 73  
cg\_wwan\_network\_list\_t, 111  
    entries, 111  
    networks, 111

cg\_wwan\_network\_settings\_t, 111  
apn, 112  
auth\_type, 112  
password, 112  
username, 112  
cg\_wwan\_network\_t, 112  
forbidden, 113  
home, 113  
id, 113  
name, 113  
preferred, 113  
roaming, 113  
system\_type, 113  
cg\_wwan\_pin\_get\_state  
WWAN, 81  
cg\_wwan\_pin\_set\_enabled  
WWAN, 82  
cg\_wwan\_pin\_state\_t, 113  
pin\_enabled, 114  
pin\_retries, 114  
pin\_type, 114  
puk\_retries, 114  
cg\_wwan\_pin\_type\_t  
WWAN, 73  
cg\_wwan\_reg\_state\_cb\_t  
WWAN, 73  
cg\_wwan\_reg\_state\_t, 114  
cs\_state, 115  
network\_name, 115  
ps\_state, 115  
system\_type, 115  
cg\_wwan\_register\_reg\_state\_notification  
WWAN, 77  
cg\_wwan\_save\_pin  
WWAN, 81  
cg\_wwan\_search\_networks  
WWAN, 80  
cg\_wwan\_set\_connection\_hunting  
WWAN, 87  
cg\_wwan\_set\_network\_settings  
WWAN, 79  
cg\_wwan\_set\_nw\_selection\_mode  
WWAN, 83  
cg\_wwan\_set\_prl\_network\_update  
WWAN, 84  
cg\_wwan\_set\_radio  
WWAN, 76  
cg\_wwan\_set\_sim\_switch  
WWAN, 86  
cg\_wwan\_signal\_strength  
WWAN, 76  
cg\_wwan\_sim\_switch\_mode\_t  
WWAN, 74  
cg\_wwan\_sim\_switch\_state\_t  
WWAN, 75  
cg\_wwan\_submit\_pin  
WWAN, 81  
cg\_wwan\_upload\_prl  
WWAN, 84  
cgate.h  
CG\_STATUS\_DOWNGRADE\_PREVENTION,  
133  
CG\_STATUS\_ERROR, 133  
CG\_STATUS\_INVALID\_PARAMETER, 133  
CG\_STATUS\_OK, 133  
CG\_STATUS\_RESOURCE\_BUSY, 133  
CG\_STATUS\_RESOURCE\_UNAVAILABLE, 133  
cg\_get\_last\_error, 133  
cg\_status\_t, 133  
channel  
cg\_wlan\_network\_t, 107  
cg\_wlan\_sniffer\_params, 108  
conf  
cg\_net\_cfg\_t, 97  
config  
cg\_net\_if\_t, 99  
Configuration functions, 9  
cg\_conf\_add\_section, 11  
cg\_conf\_close, 10  
cg\_conf\_del, 11  
cg\_conf\_get, 11  
cg\_conf\_get\_options, 12  
cg\_conf\_get\_sections, 12  
cg\_conf\_open, 9  
cg\_conf\_revert, 10  
cg\_conf\_set, 10  
cs\_state  
cg\_wwan\_reg\_state\_t, 115  
current  
cg\_wwan\_img\_list\_t, 109  
DUST\_SENSOR, 19  
CG\_DUST\_SENSOR\_CONNECT, 20  
CG\_DUST\_SENSOR\_DISCONNECT, 20  
CG\_DUST\_SENSOR\_INVALID\_PARAMETER,  
20  
CG\_DUST\_SENSOR\_NOT\_AVAILABLE, 20  
CG\_DUST\_SENSOR\_NOT\_AWAKE, 20  
CG\_DUST\_SENSOR\_POWER\_FAN\_ON, 20  
CG\_DUST\_SENSOR\_POWER\_LASER\_ON, 20  
CG\_DUST\_SENSOR\_POWER\_MAX, 20  
CG\_DUST\_SENSOR\_POWER\_OFF, 20  
CG\_DUST\_SENSOR\_RW\_ERROR, 20  
CG\_DUST\_SENSOR\_SPI\_ERROR, 20  
CG\_DUST\_SENSOR\_THREAD\_CREATE\_ERRO-  
R, 20  
CG\_DUST\_SENSOR\_VALID\_DATA, 20  
cg\_dust\_sensor\_data\_cb\_t, 19  
cg\_dust\_sensor\_deregister\_data\_callback,  
21  
cg\_dust\_sensor\_power\_t, 20  
cg\_dust\_sensor\_register\_data\_callback, 21  
cg\_dust\_sensor\_set\_power, 20  
cg\_dust\_sensor\_start, 20  
cg\_dust\_sensor\_status\_t, 20  
cg\_dust\_sensor\_stop, 20  
data\_timeout

cg\_net\_wan\_cfg\_t, 102  
detailed\_monitoring  
  cg\_wlan\_sniffer\_params, 108  
dev\_name  
  cg\_net\_if\_t, 99  
Device Management, 13  
  CG\_DEV\_EVT\_ARRIVED, 15  
  CG\_DEV\_EVT CLAIMED, 15  
  CG\_DEV\_EVT\_RELEASED, 15  
  CG\_DEV\_EVT\_REMOVED, 15  
  CG\_DEVICE CLAIMED, 14  
  CG\_DEVICE\_PRESENT, 14  
  CG\_DEVTYPE\_ADC, 14  
  CG\_DEVTYPE\_DAC, 14  
  CG\_DEVTYPE\_GPIO, 14  
  CG\_DEVTYPE\_NETWORK, 14  
  CG\_DEVTYPE\_SERIAL, 14  
  CG\_LOC\_ALL, 15  
  CG\_LOC\_MAIN, 15  
  CG\_LOC\_SLOT1, 15  
  CG\_LOC\_SLOT2, 15  
  CG\_LOC\_UNKNOWN, 15  
  cg\_device\_board\_device\_list, 17  
  cg\_device\_board\_list, 17  
  cg\_device\_claim, 16  
  cg\_device\_deregister\_notification, 17  
  cg\_device\_event\_t, 15  
  cg\_device\_list, 15  
  cg\_device\_list\_by\_location, 15  
  cg\_device\_register\_notification, 16  
  cg\_device\_release, 16  
  cg\_device\_status\_t, 14  
  cg\_device\_type\_t, 14  
  cg\_location\_t, 14  
  device\_claim\_cb\_t, 14  
  device\_notification\_cb\_t, 14  
device\_claim\_cb\_t  
  Device Management, 14  
device\_location  
  cg\_device\_t, 92  
device\_name  
  cg\_device\_t, 91  
device\_notification\_cb\_t  
  Device Management, 14  
device\_type  
  cg\_device\_t, 91  
dhcp\_config  
  cg\_net\_lan\_cfg\_t, 100  
dns1  
  cg\_net\_dhcp\_config\_t, 98  
  cg\_net\_wan\_cfg\_t, 102  
dns2  
  cg\_net\_dhcp\_config\_t, 98  
  cg\_net\_wan\_cfg\_t, 103  
enable\_termination  
  cg\_hal\_serial\_rs485\_cfg\_t, 95  
enabled  
  cg\_net\_dhcp\_config\_t, 98  
  cg\_net\_watchdog\_settings\_t, 103  
entries  
  cg\_wwan\_img\_list\_t, 109  
  cg\_wwan\_network\_list\_t, 111  
forbidden  
  cg\_wwan\_network\_t, 113  
friendly\_name  
  cg\_device\_t, 91  
GOBI\_DEVICE  
  WWAN, 73  
GPS functions, 33  
  CG\_GPS\_ASSISTED\_CONNECTION\_FAILURE, 34  
  CG\_GPS\_ASSISTED\_CONNECTION\_SUCCESS, 34  
  CG\_GPS\_ASSISTED\_DISABLE, 35  
  CG\_GPS\_ASSISTED\_NON\_SECURE\_SUPL, 35  
  CG\_GPS\_ASSISTED\_SECURE\_SUPL, 35  
  CG\_GPS\_ENABLED, 34  
  CG\_GPS\_PRESENT, 34  
  CG\_MAX\_CERTIFICATES, 33  
  CG\_MAX\_LEN\_CERTIFICATE, 34  
  cg\_gps\_assisted\_cb\_t, 34  
  cg\_gps\_assisted\_connection\_status\_t, 34  
  cg\_gps\_assisted\_function\_t, 34  
  cg\_gps\_deregister\_nmea\_callback, 37  
  cg\_gps\_deregister\_supl\_connection\_status\_callback, 37  
  cg\_gps\_get\_assisted, 36  
  cg\_gps\_get\_status, 35  
  cg\_gps\_nmea\_cb\_t, 34  
  cg\_gps\_register\_nmea\_callback, 37  
  cg\_gps\_register\_supl\_connection\_status\_callback, 37  
  cg\_gps\_set\_assisted, 36  
  cg\_gps\_set\_enabled, 35  
  cg\_gps\_set\_reporting\_interval, 35  
  cg\_gps\_status\_t, 34  
General functions, 22  
  CG\_DEV\_MINIVAN\_DEBUG, 25  
  CG\_DEV\_ROOT\_ACCESS, 25  
  CG\_MAX\_SLOT\_INFO\_LEN, 23  
  CG\_PROV\_CHECKING, 24  
  CG\_PROV\_ERROR, 24  
  CG\_PROV\_FLASHING, 24  
  CG\_PROV\_OK, 24  
  CG\_PROV\_UNPROVISIONED, 24  
  CG\_PROV\_UNREGISTERED, 24  
  CG\_SDK\_APILEVEL, 23  
  CG\_SLOT\_BASE\_CONFIG, 24  
  CG\_SLOT\_BASE\_IMAGE, 24  
  CG\_SLOT\_BOOTLOADER, 24  
  CG\_SLOT\_CUSTOMER, 24  
  CG\_SLOT\_CUSTOMER\_CONFIG, 24  
  CG\_SLOT\_CUSTOMER\_KEY, 24  
  CG\_SYSTEM\_ARMIGNITION\_SENSE, 25  
  CG\_SYSTEM\_TIMED\_WAKEUP, 25

cg\_deinit, 25  
cg\_dev\_feature\_t, 24  
cg\_get\_api\_level, 25  
cg\_get\_serial\_number, 25  
cg\_get\_slot\_list, 28  
cg\_init, 25  
cg\_prov\_check\_update, 26  
cg\_prov\_get\_status, 26  
cg\_prov\_kvpair\_del, 27  
cg\_prov\_kvpair\_deregister\_update\_callback, 28  
cg\_prov\_kvpair\_get, 27  
cg\_prov\_kvpair\_list, 27  
cg\_prov\_kvpair\_register\_update\_callback, 28  
cg\_prov\_kvpair\_set, 27  
cg\_prov\_kvpair\_update\_cb\_t, 24  
cg\_prov\_set\_automatic, 26  
cg\_prov\_status\_t, 24  
cg\_reset\_peripherals, 29  
cg\_reset\_system, 29  
cg\_set\_dev\_mode, 29  
cg\_slot\_type\_t, 24  
cg\_system\_boot\_condition\_t, 25  
cg\_system\_get\_date\_time, 31  
cg\_system\_get\_ntp\_server, 31  
cg\_system\_log, 29  
cg\_system\_power\_down, 30  
cg\_system\_set\_date\_time, 30  
cg\_system\_set\_ntp\_server, 31  
cg\_system\_set\_timed\_wakeup, 30  
cg\_system\_set\_timezone, 31  
cg\_system\_sync\_ntp, 32  
cg\_system\_sync\_ntp\_cb\_t, 24  
cg\_system\_sync\_ntp\_deregister\_callback, 32  
cg\_system\_sync\_ntp\_register\_callback, 32  
gw  
cg\_net\_wan\_cfg\_t, 102  
HAL, 39  
CG\_HAL\_SERIAL\_RS232, 39  
CG\_HAL\_SERIAL\_RS485, 39  
cg\_hal\_serial\_mode\_t, 39  
cg\_hal\_serial\_set\_config, 39  
home  
cg\_wwan\_network\_t, 113  
hw\_ver  
cg\_board\_t, 89  
id  
cg\_wwan\_network\_t, 113  
images  
cg\_wwan\_img\_list\_t, 109  
internet\_status\_callback\_t  
Networking, 42  
interval  
cg\_net\_watchdog\_settings\_t, 103  
ip\_addr  
cg\_net\_ip\_config\_t, 99  
ip\_config  
cg\_net\_cfg\_t, 97  
ip\_config\_type  
cg\_net\_wan\_cfg\_t, 102  
ip\_end  
cg\_net\_dhcp\_config\_t, 98  
ip\_start  
cg\_net\_dhcp\_config\_t, 98  
cg\_prov\_kvpair\_deregister\_update\_callback, ift\_status\_callback\_t  
Networking, 42  
lac  
cg\_wwan\_location\_info\_t, 110  
leasetime  
cg\_net\_dhcp\_config\_t, 98  
len  
cg\_gps\_certificate\_t, 93  
libcg/cg\_conf.h, 117  
libcg/cg\_device.h, 118  
libcg/cg\_dust\_sensor.h, 119  
libcg/cg\_general.h, 120  
libcg/cg\_gps.h, 122  
libcg/cg\_hal.h, 123  
libcg/cg\_net.h, 123  
libcg/cg\_sms.h, 126  
libcg/cg\_ui.h, 126  
libcg/cg\_upgrade.h, 127  
libcg/cg\_wlan.h, 127  
libcg/cg\_wlan\_sniffer.h, 129  
libcg/cg\_wwan.h, 129  
libcg/cgate.h, 132  
max\_file\_size  
cg\_wlan\_sniffer\_params, 108  
max\_nbr\_files  
cg\_wlan\_sniffer\_params, 108  
mode  
cg\_net\_wan\_cfg\_t, 102  
monitoring\_file\_path  
cg\_wlan\_sniffer\_params, 108  
mtu  
cg\_net\_ip\_config\_t, 99  
name  
cg\_wwan\_diag\_param\_t, 109  
cg\_wwan\_img\_t, 110  
cg\_wwan\_network\_t, 113  
netmask  
cg\_net\_ip\_config\_t, 99  
network\_name  
cg\_wwan\_reg\_state\_t, 115  
Networking, 40  
CG\_DHCP, 43  
CG\_NET\_INTERFACE\_CAN\_CONNECT, 42  
CG\_NET\_INTERFACE\_CONFIGURED, 42  
CG\_NET\_INTERFACE\_CONNECTED, 42  
CG\_NET\_INTERFACE\_ENABLED, 42  
CG\_NET\_INTERNET\_CONNECTED, 42

CG\_NET\_INTERNET\_CONNECTING, 42  
CG\_NET\_INTERNET\_DISCONNECTED, 42  
CG\_NET\_INTERNET\_ON\_DEMAND, 42  
CG\_NET\_MODE\_ALWAYS\_ON, 43  
CG\_NET\_MODE\_ON\_DEMAND, 43  
CG\_NET\_STRATEGY\_MANUAL, 43  
CG\_NET\_STRATEGY\_PRIORITY, 43  
CG\_NET\_TYPE\_BRIDGE, 43  
CG\_NET\_TYPE\_IPSEC, 43  
CG\_NET\_TYPE\_NIC, 43  
CG\_NET\_TYPE\_WLAN, 43  
CG\_NET\_TYPE\_WLAN\_AP, 43  
CG\_NET\_TYPE\_WWAN, 43  
CG\_NET\_UNIT\_DAY, 44  
CG\_NET\_UNIT\_HOUR, 44  
CG\_NET\_UNIT\_MINUTE, 44  
CG\_NET\_ZONE\_LAN, 43  
CG\_NET\_ZONE\_WAN, 43  
CG\_STATIC, 43  
cg\_dev\_name\_t, 42  
cg\_net\_conn\_strat\_t, 43  
cg\_net\_connect, 47  
cg\_net\_datacounter\_get\_enabled, 49  
cg\_net\_datacounter\_get\_stats, 50  
cg\_net\_datacounter\_reset\_stats, 49  
cg\_net\_datacounter\_set\_enabled, 49  
cg\_net\_deregister\_interface\_events, 46  
cg\_net\_deregister\_internet\_events, 48  
cg\_net\_disconnect, 47  
cg\_net\_get\_conn\_priority\_list, 47  
cg\_net\_get\_conn\_strat, 47  
cg\_net\_get\_disabled\_at\_boot, 51  
cg\_net\_get\_interface, 44  
cg\_net\_get\_interface\_by\_type, 44  
cg\_net\_get\_interface\_list, 44  
cg\_net\_get\_internet\_status, 48  
cg\_net\_get\_manual\_conn\_device, 46  
cg\_net\_get\_mtu, 50  
cg\_net\_get\_watchdog, 48  
cg\_net\_if\_status\_t, 42  
cg\_net\_if\_type\_t, 43  
cg\_net\_interface\_set\_enabled, 45  
cg\_net\_internet\_status\_t, 42  
cg\_net\_ip\_config\_type\_t, 43  
cg\_net\_lease\_unit\_t, 43  
cg\_net\_mode\_t, 43  
cg\_net\_register\_interface\_events, 45  
cg\_net\_register\_internet\_events, 48  
cg\_net\_set\_config, 45  
cg\_net\_set\_disabled\_at\_boot, 50  
cg\_net\_set\_manual\_conn\_strat, 46  
cg\_net\_set\_mtu, 50  
cg\_net\_set\_priority\_conn\_strat, 46  
cg\_net\_set\_watchdog, 48  
cg\_net\_zone\_t, 42  
internet\_status\_callback\_t, 42  
itf\_status\_callback\_t, 42  
networks  
cg\_wwan\_network\_list\_t, 111  
num\_addresses  
cg\_net\_watchdog\_settings\_t, 104  
num\_slots  
cg\_slot\_list\_t, 104  
number  
cg\_conf\_list, 90  
param  
cg\_hal\_serial\_cfg\_t, 94  
password  
cg\_wlan\_auth\_params\_t, 106  
cg\_wwan\_network\_settings\_t, 112  
pin\_enabled  
cg\_wwan\_pin\_state\_t, 114  
pin\_retries  
cg\_wwan\_pin\_state\_t, 114  
pin\_type  
cg\_wwan\_pin\_state\_t, 114  
preferred  
cg\_wwan\_network\_t, 113  
ps\_state  
cg\_wwan\_reg\_state\_t, 115  
puk\_retries  
cg\_wwan\_pin\_state\_t, 114  
reporting\_interval  
cg\_gps\_t, 94  
roaming  
cg\_wwan\_network\_t, 113  
rs485  
cg\_hal\_serial\_cfg\_t, 94  
rx\_bytes  
cg\_net\_stats\_t, 101  
rx\_packets  
cg\_net\_stats\_t, 101  
SMS, 52  
cg\_sms\_cb\_t, 52  
cg\_sms\_deregister\_new\_sms, 53  
cg\_sms\_get\_smsc, 52  
cg\_sms\_register\_new\_sms, 53  
cg\_sms\_send, 53  
cg\_sms\_set\_smsc, 52  
serial\_mode  
cg\_hal\_serial\_cfg\_t, 94  
signal\_strength  
cg\_wlan\_network\_t, 107  
size  
cg\_slot\_t, 105  
slot\_id  
cg\_board\_t, 89  
slots  
cg\_slot\_list\_t, 104  
ssid  
cg\_wlan\_network\_t, 107  
status  
cg\_device\_t, 92  
cg\_dust\_sensor\_data\_t, 92

cg\_gps\_t, 94  
cg\_net\_if\_t, 99  
system\_type  
cg\_wwan\_network\_t, 113  
cg\_wwan\_reg\_state\_t, 115

tac  
cg\_wwan\_location\_info\_t, 110

tx\_bytes  
cg\_net\_stats\_t, 101

tx\_packets  
cg\_net\_stats\_t, 101

type  
cg\_net\_if\_t, 99  
cg\_slot\_t, 105  
cg\_wlan\_auth\_params\_t, 106

UI, 55  
cg\_ui\_deregister\_get\_callback, 59  
cg\_ui\_deregister\_json\_callback, 58  
cg\_ui\_deregister\_page, 57  
cg\_ui\_deregister\_session\_callback, 59  
cg\_ui\_get\_cb\_t, 57  
cg\_ui\_json\_cb\_t, 56  
cg\_ui\_register\_get\_callback, 58  
cg\_ui\_register\_json\_callback, 58  
cg\_ui\_register\_page, 57  
cg\_ui\_register\_session\_callback, 59  
cg\_ui\_session\_cb, 57

uid  
cg\_slot\_t, 105

unit  
cg\_net\_leasetime\_t, 101

Upgrade API, 60  
cg\_upgrade\_data, 60  
cg\_upgrade\_start, 60  
cg\_upgrade\_stop, 61  
cg\_upgrade\_t, 60

use\_ping  
cg\_net\_watchdog\_settings\_t, 103

username  
cg\_wwan\_network\_settings\_t, 112

value  
cg\_wwan\_diag\_param\_t, 109

values  
cg\_conf\_list, 90

version  
cg\_slot\_t, 105

WLAN, 62  
CG\_PASSWORD\_MAX\_LENGTH, 63  
CG\_SSID\_MAX\_LENGTH, 63  
CG\_WLAN\_AUTH\_OPEN, 63  
CG\_WLAN\_AUTH\_WEP, 63  
CG\_WLAN\_AUTH\_WPA2\_ENT, 63  
CG\_WLAN\_AUTH\_WPA2\_PSK, 63  
CG\_WLAN\_AUTH\_WPA\_ENT, 63  
CG\_WLAN\_AUTH\_WPA\_PSK, 63

CG\_WLAN\_AUTH\_WPA2\_ENT, 63  
CG\_WLAN\_AUTH\_WPA2\_PSK, 63  
cg\_wlan\_ap\_get\_auth\_params, 67  
cg\_wlan\_ap\_get\_channel, 66  
cg\_wlan\_ap\_get\_ssid, 66  
cg\_wlan\_ap\_set\_auth\_params, 67  
cg\_wlan\_ap\_set\_channel, 66  
cg\_wlan\_ap\_set\_ssid, 65  
cg\_wlan\_auth\_t, 63  
cg\_wlan\_network\_list\_cb\_t, 63  
cg\_wlan\_sta\_get\_connected\_network, 64  
cg\_wlan\_sta\_get\_network\_list, 65  
cg\_wlan\_sta\_remove\_network, 65  
cg\_wlan\_sta\_scan\_networks, 63  
cg\_wlan\_sta\_set\_network, 64

WLAN\_SNIFFER, 68  
CG\_MONITORING\_FILE\_PATH\_MAX\_LEN, 68  
cg\_wlan\_sniffer\_start, 68  
cg\_wlan\_sniffer\_stop, 69

WWAN, 70  
CG\_MAX\_IMAGE\_NAME, 72  
CG\_NETWORK\_NAME\_SIZE, 72  
CG\_SIM\_SWITCH\_MODE\_AUTO, 75  
CG\_SIM\_SWITCH\_MODE\_SIM1, 75  
CG\_SIM\_SWITCH\_MODE\_SIM2, 75  
CG\_SIM\_SWITCH\_STATE\_SIM1, 75  
CG\_SIM\_SWITCH\_STATE\_SIM2, 75  
CG\_WWAN\_ACT\_STATUS\_ACTIVATED, 74  
CG\_WWAN\_ACT\_STATUS\_CONNECTED, 74  
CG\_WWAN\_ACT\_STATUS\_CONNECTION, 74  
CG\_WWAN\_ACT\_STATUS\_NOT\_ACTIVATED, 74  
CG\_WWAN\_ACT\_STATUS\_OTASP\_AUTH, 74  
CG\_WWAN\_ACT\_STATUS\_OTASP\_COMMITTED, 74  
CG\_WWAN\_ACT\_STATUS\_OTASP\_IMSI, 74  
CG\_WWAN\_ACT\_STATUS\_OTASP\_MDN, 74  
CG\_WWAN\_ACT\_STATUS\_OTASP\_NAM, 74  
CG\_WWAN\_ACT\_STATUS\_OTASP\_PRL, 74  
CG\_WWAN\_ACT\_STATUS\_OTASP\_SPC, 74  
CG\_WWAN\_AUTH\_TYPE\_AUTO, 75  
CG\_WWAN\_AUTH\_TYPE\_CHAP, 75  
CG\_WWAN\_AUTH\_TYPE\_NONE, 75  
CG\_WWAN\_AUTH\_TYPE\_PAP, 75  
CG\_WWAN\_CREG\_DENIED, 74  
CG\_WWAN\_CREG\_NOT\_REG, 74  
CG\_WWAN\_CREG\_REG, 74  
CG\_WWAN\_CREG\_ROAMING, 74  
CG\_WWAN\_CREG\_SEARCHING, 74  
CG\_WWAN\_MODE\_CDMA, 74  
CG\_WWAN\_MODE\_UTMS, 74  
CG\_WWAN\_PIN\_LOCKED\_PN, 73  
CG\_WWAN\_PIN\_READY, 73  
CG\_WWAN\_PIN\_SIM\_FAILURE, 73  
CG\_WWAN\_PIN\_SIM\_NOT\_INSERTED, 73  
CG\_WWAN\_PIN\_SIM\_NOT\_SUPPORTED, 73  
CG\_WWAN\_PIN\_SIM\_PIN, 73  
CG\_WWAN\_PIN\_SIM\_PUK, 73

CG\_WWAN\_PIN\_UNKNOWN, 73  
cg\_wwan\_act\_state\_t, 74  
cg\_wwan\_activate, 84  
cg\_wwan\_auth\_type\_t, 75  
cg\_wwan\_change\_pin, 82  
cg\_wwan\_creg\_status\_t, 74  
cg\_wwan\_deregister\_reg\_state\_notification,  
    78  
cg\_wwan\_get\_activation\_state, 83  
cg\_wwan\_get\_active\_primary\_dev, 75  
cg\_wwan\_get\_diagnostics, 78  
cg\_wwan\_get\_iccid, 81  
cg\_wwan\_get\_imei, 80  
cg\_wwan\_get\_imsi, 80  
cg\_wwan\_get\_location\_info, 87  
cg\_wwan\_get\_meid, 83  
cg\_wwan\_get\_mode, 76  
cg\_wwan\_get\_network\_settings, 79  
cg\_wwan\_get\_nw\_selection\_mode, 82  
cg\_wwan\_get\_phone\_number, 79  
cg\_wwan\_get\_prl\_network\_update, 85  
cg\_wwan\_get\_prl\_version, 84  
cg\_wwan\_get\_radio, 77  
cg\_wwan\_get\_reg\_state, 77  
cg\_wwan\_get\_serial, 78  
cg\_wwan\_get\_sim\_switch, 87  
cg\_wwan\_gobi\_get\_active\_image, 85  
cg\_wwan\_gobi\_get\_image\_list, 85  
cg\_wwan\_gobi\_get\_image\_type, 86  
cg\_wwan\_gobi\_set\_active\_image, 86  
cg\_wwan\_mode\_t, 73  
cg\_wwan\_modem\_reboot, 86  
cg\_wwan\_network\_list\_cb\_t, 73  
cg\_wwan\_pin\_get\_state, 81  
cg\_wwan\_pin\_set\_enabled, 82  
cg\_wwan\_pin\_type\_t, 73  
cg\_wwan\_register\_reg\_state\_notification,  
    77  
cg\_wwan\_save\_pin, 81  
cg\_wwan\_search\_networks, 80  
cg\_wwan\_set\_connection\_hunting, 87  
cg\_wwan\_set\_network\_settings, 79  
cg\_wwan\_set\_nw\_selection\_mode, 83  
cg\_wwan\_set\_prl\_network\_update, 84  
cg\_wwan\_set\_radio, 76  
cg\_wwan\_set\_sim\_switch, 86  
cg\_wwan\_signal\_strength, 76  
cg\_wwan\_sim\_switch\_mode\_t, 74  
cg\_wwan\_sim\_switch\_state\_t, 75  
cg\_wwan\_submit\_pin, 81  
cg\_wwan\_upload\_prl, 84  
GOBI\_DEVICE, 73  
window\_size  
    cg\_wlan\_sniffer\_params, 108  
  
zone  
    cg\_net\_cfg\_t, 97

