WMSDK Reference Manual

C API Reference

Draft

© Marvell International, 2009-2015

Confidential

August 21, 2015 17:48
Generated by Doxygen 1.8.1.1

Confidential

Contents

1 Main Page **1**

1.1 Introduction . . . 1

1.1.1 Developer Documentation . . . 1

1.1.1.1 Application Framework . . . 1

1.1.1.2 CMSIS Library . . . 1

1.1.1.3 SDK Modules . . . 2

1.1.1.4 SDK Drivers . . . 2

1.1.1.5 SDK Porting . . . 3

1.1.1.6 Third-Party Cloud Providers . . . 3

2 Deprecated List **5**

3 Module Index **7**

3.1 Modules . . . 7

4 Data Structure Index **9**

4.1 Data Structures . . . 9

5 File Index **11**

5.1 File List . . . 11

6 Module Documentation **15**

6.1 Wi-Fi Region Configuration . . . 15

6.1.1 Detailed Description . . . 15

6.1.2 Function Documentation . . . 15

6.1.2.1 wlan_enable_11d . . . 15

6.1.2.2 wlan_get_country . . . 16

6.1.2.3 wlan_set_country . . . 16

6.2 Custom Wi-Fi Region Configuration . . . 17

6.3 Low level RFGET APIs . . . 20

6.3.1 Detailed Description . . . 20

6.3.2 Typedef Documentation . . . 20

6.3.2.1 update_desc_t . . . 20

Confidential

MARVELL

Proprietary Information.
August 21, 2015 17:48

Copyright © 2015 Marvell

6.3.3 Function Documentation . . . 20
6.3.3.1 rfget_init . . . 20
6.3.3.2 rfget_update_begin . . . 21
6.3.3.3 rfget_update_data . . . 21
6.3.3.4 rfget_update_complete . . . 21
6.3.3.5 rfget_update_abort . . . 21
6.3.3.6 rfget_is_update_in_progress . . . 22
6.3.3.7 verify_load_firmware . . . 22
6.3.3.8 rfget_get_passive_firmware . . . 22
6.3.3.9 rfget_get_passive_wifi_firmware . . . 22
6.4 Middle level RFGET APIs . . . 23
6.4.1 Detailed Description . . . 23
6.4.2 Typedef Documentation . . . 23
6.4.2.1 data_fetch . . . 23
6.4.2.2 data_validate . . . 24
6.4.3 Function Documentation . . . 24
6.4.3.1 update_and_validate_firmware . . . 24
6.4.3.2 update_and_validate_wifi_firmware . . . 24
6.4.3.3 update_and_validate_fs . . . 25
6.4.3.4 update_firmware . . . 25
6.4.3.5 update_wifi_firmware . . . 26
6.5 High level RFGET APIs . . . 27
6.5.1 Detailed Description . . . 27
6.5.2 Function Documentation . . . 27
6.5.2.1 rfget_server_mode_update . . . 27
6.5.2.2 rfget_client_mode_update . . . 28
6.6 Other RFGET APIs . . . 29
6.6.1 Detailed Description . . . 29
6.6.2 Function Documentation . . . 29
6.6.2.1 rfget_cli_init . . . 29
6.6.2.2 rfget_ftfs_init . . . 29

7 Data Structure Documentation 31
7.1 _wifi_cal_data_t Struct Reference . . . 31
7.1.1 Detailed Description . . . 31
7.1.2 Field Documentation . . . 31
7.1.2.1 data_len . . . 31
7.1.2.2 data . . . 31
7.2 _wifi_mgmt_frame_tx Struct Reference . . . 31
7.2.1 Detailed Description . . . 32

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.2.2 Field Documentation . . . 32
7.2.2.1 frm_len . . . 32
7.2.2.2 frm_ctl . . . 32
7.2.2.3 duration_id . . . 32
7.2.2.4 addr1 . . . 32
7.2.2.5 addr2 . . . 32
7.2.2.6 addr3 . . . 32
7.2.2.7 seq_ctl . . . 32
7.2.2.8 addr4 . . . 32
7.2.2.9 payload . . . 33
7.3 app_init_state Struct Reference . . . 33
7.3.1 Detailed Description . . . 33
7.3.2 Field Documentation . . . 33
7.3.2.1 factory_reset . . . 33
7.3.2.2 backup_fw . . . 33
7.3.2.3 rst_cause . . . 33
7.4 app_p2p_role_data Struct Reference . . . 33
7.4.1 Detailed Description . . . 34
7.4.2 Field Documentation . . . 34
7.4.2.1 role . . . 34
7.4.2.2 data . . . 34
7.5 cli_command Struct Reference . . . 34
7.5.1 Detailed Description . . . 34
7.5.2 Field Documentation . . . 34
7.5.2.1 name . . . 34
7.5.2.2 help . . . 34
7.5.2.3 function . . . 34
7.6 DH_PG_PARAMS Struct Reference . . . 35
7.6.1 Detailed Description . . . 35
7.6.2 Field Documentation . . . 35
7.6.2.1 prime . . . 35
7.6.2.2 primeLen . . . 35
7.6.2.3 generator . . . 35
7.6.2.4 generatorLen . . . 35
7.7 dma_config_type Struct Reference . . . 35
7.7.1 Detailed Description . . . 35
7.7.2 Field Documentation . . . 36
7.7.2.1 dma_cfg . . . 36
7.7.2.2 perDmaInter . . . 36
7.8 flash_desc Struct Reference . . . 36

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.8.1 Detailed Description . . . 36
7.8.2 Field Documentation . . . 36
7.8.2.1 fl_dev . . . 36
7.8.2.2 fl_start . . . 36
7.8.2.3 fl_size . . . 36
7.9 healthmon_handler Struct Reference . . . 37
7.9.1 Detailed Description . . . 37
7.9.2 Field Documentation . . . 37
7.9.2.1 name . . . 37
7.9.2.2 is_sick . . . 37
7.9.2.3 about_to_die . . . 37
7.9.2.4 check_interval . . . 37
7.9.2.5 consecutive_failures . . . 38
7.10 http_req_t Struct Reference . . . 38
7.10.1 Detailed Description . . . 38
7.10.2 Field Documentation . . . 38
7.10.2.1 type . . . 38
7.10.2.2 resource . . . 38
7.10.2.3 version . . . 38
7.10.2.4 content . . . 38
7.10.2.5 content_len . . . 39
7.11 http_resp_t Struct Reference . . . 39
7.11.1 Detailed Description . . . 39
7.11.2 Field Documentation . . . 39
7.11.2.1 protocol . . . 39
7.11.2.2 version . . . 39
7.11.2.3 status_code . . . 39
7.11.2.4 reason_phrase . . . 39
7.11.2.5 server . . . 40
7.11.2.6 content_type . . . 40
7.11.2.7 content_encoding . . . 40
7.11.2.8 keep_alive_ack . . . 40
7.11.2.9 keep_alive_timeout . . . 40
7.11.2.10 keep_alive_max . . . 40
7.11.2.11 chunked . . . 40
7.11.2.12 content_length_field_present . . . 40
7.11.2.13 content_length . . . 40
7.12 httpc_cfg_t Struct Reference . . . 40
7.12.1 Detailed Description . . . 41
7.12.2 Field Documentation . . . 41

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.12.2.1 flags . . . 41
7.12.2.2 retry_cnt . . . 41
7.13 httpd_request_t Struct Reference . . . 41
7.13.1 Detailed Description . . . 41
7.13.2 Field Documentation . . . 42
7.13.2.1 type . . . 42
7.13.2.2 filename . . . 42
7.13.2.3 sock . . . 42
7.13.2.4 hdr_parsed . . . 42
7.13.2.5 remaining_bytes . . . 42
7.13.2.6 body_nbytes . . . 42
7.13.2.7 wsgi . . . 42
7.13.2.8 chunked . . . 42
7.13.2.9 agent . . . 42
7.13.2.10 content_type . . . 42
7.13.2.11 if_none_match . . . 42
7.13.2.12 etag_val . . . 42
7.13.2.13 httpd_priv . . . 43
7.13.2.14 free_priv_data_fn . . . 43
7.14 httpd_ssi_call Struct Reference . . . 43
7.14.1 Detailed Description . . . 43
7.14.2 Field Documentation . . . 43
7.14.2.1 name . . . 43
7.14.2.2 function . . . 43
7.15 httpd_tls_certs_t Struct Reference . . . 43
7.15.1 Detailed Description . . . 44
7.15.2 Field Documentation . . . 44
7.15.2.1 server_cert . . . 44
7.15.2.2 server_cert_chained . . . 44
7.15.2.3 server_cert_size . . . 44
7.15.2.4 server_key . . . 44
7.15.2.5 server_key_size . . . 44
7.15.2.6 client_cert . . . 44
7.15.2.7 client_cert_size . . . 44
7.16 httpd_useragent_t Struct Reference . . . 44
7.16.1 Detailed Description . . . 45
7.16.2 Field Documentation . . . 45
7.16.2.1 product . . . 45
7.16.2.2 version . . . 45
7.17 httpd_wsgi_call Struct Reference . . . 45

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.17.1 Detailed Description . . . 45
7.17.2 Field Documentation . . . 45
7.17.2.1 uri . . . 45
7.17.2.2 hdr_fields . . . 45
7.17.2.3 http_flags . . . 46
7.17.2.4 get_handler . . . 46
7.17.2.5 set_handler . . . 46
7.17.2.6 put_handler . . . 46
7.17.2.7 delete_handler . . . 46
7.18 input_gpio_cfg_t Struct Reference . . . 46
7.18.1 Detailed Description . . . 46
7.18.2 Field Documentation . . . 46
7.18.2.1 gpio . . . 46
7.18.2.2 type . . . 46
7.19 ipv4_config Struct Reference . . . 47
7.19.1 Detailed Description . . . 47
7.19.2 Field Documentation . . . 47
7.19.2.1 addr_type . . . 47
7.19.2.2 address . . . 47
7.19.2.3 gw . . . 47
7.19.2.4 netmask . . . 47
7.19.2.5 dns1 . . . 47
7.19.2.6 dns2 . . . 47
7.20 jobj_t Struct Reference . . . 48
7.20.1 Detailed Description . . . 48
7.20.2 Field Documentation . . . 48
7.20.2.1 parser . . . 48
7.20.2.2 js . . . 48
7.20.2.3 tokens . . . 48
7.20.2.4 cur . . . 48
7.20.2.5 num_tokens . . . 48
7.21 jsontok_t Struct Reference . . . 48
7.21.1 Detailed Description . . . 49
7.21.2 Field Documentation . . . 49
7.21.2.1 type . . . 49
7.21.2.2 size . . . 49
7.21.2.3 start . . . 49
7.21.2.4 end . . . 49
7.21.2.5 parent . . . 49
7.22 mdns_service Struct Reference . . . 49

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.22.1 Detailed Description . . . 50
7.22.2 Field Documentation . . . 50
7.22.2.1 servname . . . 50
7.22.2.2 servtype . . . 50
7.22.2.3 domain . . . 51
7.22.2.4 port . . . 51
7.22.2.5 proto . . . 51
7.22.2.6 keyvals . . . 51
7.22.2.7 ipaddr . . . 51
7.22.2.8 fqsn . . . 51
7.22.2.9 ptrname . . . 51
7.22.2.10 kvlen . . . 51
7.22.2.11 flags . . . 51
7.23 os_queue_pool Struct Reference . . . 51
7.23.1 Detailed Description . . . 52
7.23.2 Field Documentation . . . 52
7.23.2.1 size . . . 52
7.24 os_thread_stack Struct Reference . . . 52
7.24.1 Detailed Description . . . 52
7.24.2 Field Documentation . . . 52
7.24.2.1 size . . . 52
7.25 output_gpio_cfg_t Struct Reference . . . 52
7.25.1 Detailed Description . . . 52
7.25.2 Field Documentation . . . 53
7.25.2.1 gpio . . . 53
7.25.2.2 type . . . 53
7.26 overlay Struct Reference . . . 53
7.26.1 Detailed Description . . . 53
7.26.2 Field Documentation . . . 53
7.26.2.1 o_name . . . 53
7.26.2.2 o_ram_start . . . 53
7.26.2.3 o_no_parts . . . 53
7.26.2.4 o_range . . . 53
7.27 overlay_ranges Struct Reference . . . 54
7.27.1 Detailed Description . . . 54
7.27.2 Field Documentation . . . 54
7.27.2.1 or_flash_start . . . 54
7.27.2.2 or_flash_end . . . 54
7.27.2.3 o_bss_start . . . 54
7.27.2.4 o_bss_end . . . 54

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.28 p2p_config Struct Reference . . . 54
7.28.1 Detailed Description . . . 55
7.28.2 Field Documentation . . . 55
7.28.2.1 role . . . 55
7.28.2.2 auto_go . . . 55
7.28.2.3 pin_generator . . . 55
7.28.2.4 version . . . 55
7.28.2.5 version2 . . . 55
7.28.2.6 device_name . . . 55
7.28.2.7 manufacture . . . 55
7.28.2.8 model_name . . . 55
7.28.2.9 model_number . . . 56
7.28.2.10 serial_number . . . 56
7.28.2.11 config_methods . . . 56
7.28.2.12 primary_dev_category . . . 56
7.28.2.13 primary_dev_subcategory . . . 56
7.28.2.14 rf_bands . . . 56
7.28.2.15 os_version . . . 56
7.28.2.16 p2p_msg_max_retry . . . 56
7.28.2.17 p2p_msg_timeout . . . 56
7.28.2.18 pin_len . . . 56
7.28.2.19 p2p_callback . . . 56
7.29 p2p_scan_result Struct Reference . . . 57
7.29.1 Detailed Description . . . 57
7.29.2 Field Documentation . . . 57
7.29.2.1 group_capability . . . 57
7.29.2.2 device_id . . . 57
7.29.2.3 device_name . . . 57
7.29.2.4 device_address . . . 57
7.29.2.5 go_ssid . . . 57
7.29.2.6 intended_address . . . 57
7.29.2.7 op_channel . . . 57
7.29.2.8 listen_channel . . . 58
7.30 partition_entry Struct Reference . . . 58
7.30.1 Detailed Description . . . 58
7.30.2 Field Documentation . . . 58
7.30.2.1 type . . . 58
7.30.2.2 device . . . 58
7.30.2.3 name . . . 58
7.30.2.4 start . . . 58

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.30.2.5 size . . . 58
7.30.2.6 gen_level . . . 59
7.31 partition_table Struct Reference . . . 59
7.31.1 Detailed Description . . . 59
7.31.2 Field Documentation . . . 59
7.31.2.1 magic . . . 59
7.31.2.2 version . . . 59
7.31.2.3 partition_entries_no . . . 59
7.31.2.4 gen_level . . . 59
7.31.2.5 crc . . . 59
7.32 prov_scan_config Struct Reference . . . 60
7.32.1 Detailed Description . . . 60
7.32.2 Field Documentation . . . 60
7.32.2.1 wifi_scan_params . . . 60
7.32.2.2 scan_interval . . . 60
7.33 provisioning_config Struct Reference . . . 60
7.33.1 Detailed Description . . . 60
7.33.2 Field Documentation . . . 60
7.33.2.1 prov_mode . . . 60
7.33.2.2 provisioning_event_handler . . . 61
7.34 psm_cfg_t Struct Reference . . . 61
7.34.1 Detailed Description . . . 61
7.34.2 Field Documentation . . . 61
7.34.2.1 read_only . . . 61
7.34.2.2 secure . . . 61
7.35 psm_objattrib_t Struct Reference . . . 61
7.35.1 Detailed Description . . . 62
7.35.2 Field Documentation . . . 62
7.35.2.1 flags . . . 62
7.36 tls_client_t Struct Reference . . . 62
7.36.1 Detailed Description . . . 62
7.36.2 Field Documentation . . . 62
7.36.2.1 ca_cert . . . 62
7.36.2.2 ca_cert_size . . . 62
7.36.2.3 client_cert . . . 62
7.36.2.4 client_cert_is_chained . . . 63
7.36.2.5 client_cert_size . . . 63
7.36.2.6 client_key . . . 63
7.36.2.7 client_key_size . . . 63
7.37 tls_init_config_t Struct Reference . . . 63

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.37.1 Detailed Description . . . 63
7.37.2 Field Documentation . . . 64
7.37.2.1 flags . . . 64
7.37.2.2 ca_cert . . . 64
7.37.2.3 ca_cert_size . . . 64
7.37.2.4 client_cert . . . 64
7.37.2.5 client_cert_size . . . 64
7.37.2.6 client_key . . . 64
7.37.2.7 client_key_size . . . 64
7.37.2.8 client . . . 64
7.37.2.9 server_cert . . . 64
7.37.2.10 server_cert_size . . . 64
7.37.2.11 server_key . . . 65
7.37.2.12 server_key_size . . . 65
7.37.2.13 server . . . 65
7.37.2.14 tls . . . 65
7.38 tm Struct Reference . . . 65
7.38.1 Detailed Description . . . 65
7.38.2 Field Documentation . . . 65
7.38.2.1 tm_sec . . . 65
7.38.2.2 tm_min . . . 65
7.38.2.3 tm_hour . . . 66
7.38.2.4 tm_mday . . . 66
7.38.2.5 tm_mon . . . 66
7.38.2.6 tm_year . . . 66
7.38.2.7 tm_wday . . . 66
7.39 update_desc Struct Reference . . . 66
7.39.1 Detailed Description . . . 66
7.39.2 Field Documentation . . . 66
7.39.2.1 pe . . . 66
7.39.2.2 dev . . . 66
7.39.2.3 addr . . . 67
7.39.2.4 remaining . . . 67
7.40 upgrade_struct_t Struct Reference . . . 67
7.40.1 Detailed Description . . . 67
7.40.2 Field Documentation . . . 67
7.40.2.1 decrypt_init . . . 67
7.40.2.2 decrypt . . . 67
7.40.2.3 signature_verify . . . 68
7.40.2.4 hash_update . . . 68

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.40.2.5 hash_finish . . . 68
7.40.2.6 fw_data_read_fn . . . 68
7.40.2.7 decrypt_ctx . . . 68
7.40.2.8 hash_ctx . . . 68
7.40.2.9 iv . . . 68
7.40.2.10 iv_len . . . 68
7.40.2.11 signature . . . 68
7.40.2.12 signature_len . . . 68
7.40.2.13 hash . . . 68
7.40.2.14 hash_len . . . 68
7.40.2.15 pk . . . 69
7.40.2.16 decrypt_key . . . 69
7.40.2.17 session_priv_data . . . 69
7.40.2.18 bytes_to_read . . . 69
7.41 wifi_domain_param_t Struct Reference . . . 69
7.41.1 Detailed Description . . . 69
7.41.2 Field Documentation . . . 69
7.41.2.1 country_code . . . 69
7.41.2.2 no_of_sub_band . . . 69
7.41.2.3 sub_band . . . 70
7.42 wifi_sub_band_set_t Struct Reference . . . 70
7.42.1 Detailed Description . . . 70
7.42.2 Field Documentation . . . 70
7.42.2.1 first_chan . . . 70
7.42.2.2 no_of_chan . . . 70
7.42.2.3 max_tx_pwr . . . 70
7.43 wlan_ip_config Struct Reference . . . 70
7.43.1 Detailed Description . . . 71
7.43.2 Field Documentation . . . 71
7.43.2.1 ipv4 . . . 71
7.44 wlan_network Struct Reference . . . 71
7.44.1 Detailed Description . . . 71
7.44.2 Field Documentation . . . 72
7.44.2.1 name . . . 72
7.44.2.2 ssid . . . 72
7.44.2.3 bssid . . . 72
7.44.2.4 channel . . . 72
7.44.2.5 type . . . 72
7.44.2.6 role . . . 72
7.44.2.7 security . . . 72

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.44.2.8 ip . . . 72
7.44.2.9 ssid_specific . . . 72
7.44.2.10 bssid_specific . . . 73
7.44.2.11 channel_specific . . . 73
7.44.2.12 security_specific . . . 73
7.45 wlan_network_security Struct Reference . . . 73
7.45.1 Detailed Description . . . 73
7.45.2 Field Documentation . . . 74
7.45.2.1 type . . . 74
7.45.2.2 mcstCipher . . . 74
7.45.2.3 ucstCipher . . . 74
7.45.2.4 is_pmf_required . . . 74
7.45.2.5 psk . . . 74
7.45.2.6 psk_len . . . 74
7.45.2.7 pmk . . . 74
7.45.2.8 pmk_valid . . . 74
7.46 wlan_scan_result Struct Reference . . . 74
7.46.1 Detailed Description . . . 75
7.46.2 Field Documentation . . . 75
7.46.2.1 ssid . . . 75
7.46.2.2 ssid_len . . . 75
7.46.2.3 bssid . . . 75
7.46.2.4 channel . . . 75
7.46.2.5 type . . . 75
7.46.2.6 role . . . 75
7.46.2.7 wmm . . . 75
7.46.2.8 wep . . . 76
7.46.2.9 wpa . . . 76
7.46.2.10 wpa2 . . . 76
7.46.2.11 rssi . . . 76
7.47 wps_config Struct Reference . . . 76
7.47.1 Detailed Description . . . 76
7.47.2 Field Documentation . . . 77
7.47.2.1 role . . . 77
7.47.2.2 pin_generator . . . 77
7.47.2.3 version . . . 77
7.47.2.4 version2 . . . 77
7.47.2.5 device_name . . . 77
7.47.2.6 manufacture . . . 77
7.47.2.7 model_name . . . 77

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.47.2.8 model_number . . . 77
7.47.2.9 serial_number . . . 77
7.47.2.10 config_methods . . . 77
7.47.2.11 primary_dev_category . . . 77
7.47.2.12 primary_dev_subcategory . . . 77
7.47.2.13 rf_bands . . . 78
7.47.2.14 os_version . . . 78
7.47.2.15 wps_msg_max_retry . . . 78
7.47.2.16 wps_msg_timeout . . . 78
7.47.2.17 pin_len . . . 78
7.47.2.18 wps_callback . . . 78
7.48 wslay_frame Struct Reference . . . 78
7.48.1 Detailed Description . . . 78
7.48.2 Field Documentation . . . 78
7.48.2.1 fin . . . 78
7.48.2.2 opcode . . . 79
7.48.2.3 data . . . 79
7.48.2.4 data_len . . . 79

8 File Documentation **81**
8.1 app_framework.h File Reference . . . 81
8.1.1 Detailed Description . . . 81
8.1.2 Application Framework Features . . . 81
8.1.3 Usage . . . 82
8.1.4 State Machine . . . 82
8.1.5 Function Documentation . . . 83
8.1.5.1 app_framework_start . . . 83
8.1.5.2 app_configure_uap_ip . . . 83
8.1.5.3 app_uap_start_with_dhcp . . . 84
8.1.5.4 app_uap_start_on_channel_with_dhcp . . . 84
8.1.5.5 app_uap_start . . . 85
8.1.5.6 app_uap_start_on_channel . . . 85
8.1.5.7 app_uap_start_by_network . . . 86
8.1.5.8 app_uap_stop . . . 86
8.1.5.9 app_sta_start . . . 86
8.1.5.10 app_sta_save_network . . . 87
8.1.5.11 app_sta_start_by_network . . . 87
8.1.5.12 app_sta_stop . . . 87
8.1.5.13 app_sta_save_network_and_start . . . 88
8.1.5.14 app_reset_saved_network . . . 88

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.1.5.15 app_provisioning_start . . . 88
8.1.5.16 app_provisioning_only_start . . . 89
8.1.5.17 app_ezconnect_provisioning_start . . . 89
8.1.5.18 app_prov_wps_session_start . . . 89
8.1.5.19 app_provisioning_stop . . . 90
8.1.5.20 app_provisioning_only_stop . . . 90
8.1.5.21 app_ezconnect_provisioning_stop . . . 90
8.1.5.22 app_ftfs_init . . . 90
8.1.5.23 app_httpd_with_fs_start . . . 91
8.1.5.24 app_httpd_start . . . 91
8.1.5.25 app_httpd_only_start . . . 91
8.1.5.26 app_sys_register_upgrade_handler . . . 91
8.1.5.27 app_sys_register_diag_handler . . . 92
8.1.5.28 app_sys_register_perf_handler . . . 92
8.1.5.29 app_httpd_stop . . . 92
8.1.5.30 app_mdns_start . . . 92
8.1.5.31 app_mdns_stop . . . 93
8.1.5.32 app_mdns_refresh_hostname . . . 93
8.1.5.33 app_mdns_add_service . . . 93
8.1.5.34 app_mdns_add_service_arr . . . 93
8.1.5.35 app_mdns_remove_service . . . 94
8.1.5.36 app_mdns_remove_service_arr . . . 94
8.1.5.37 app_mdns_remove_service_all . . . 94
8.1.5.38 app_mdns_iface_state_change . . . 95
8.1.5.39 app_init_device_name . . . 95
8.1.5.40 app_get_device_name . . . 95
8.1.5.41 app_set_device_name . . . 96
8.1.5.42 app_sys_get_uuid . . . 96
8.1.5.43 app_network_ip_get . . . 96
8.1.5.44 app_reboot . . . 96
8.1.5.45 app_ctrl_notify_application_event . . . 97
8.1.5.46 app_sys_http_update_all . . . 97
8.1.5.47 app_p2p_start . . . 97
8.1.5.48 app_p2p_stop . . . 98
8.1.5.49 app_p2p_session_start . . . 98
8.1.5.50 app_p2p_disconnect . . . 98
8.1.5.51 app_p2p_scan . . . 98
8.1.5.52 app_p2p_ip_get . . . 99
8.1.5.53 app_ctrl_event_name . . . 99
8.1.5.54 app_psm_init . . . 99

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.1.5.55 app_get_connection_status . . . 99
8.1.5.56 app_ctrl_getmode . . . 100
8.1.6 Macro Documentation . . . 100
8.1.6.1 APP_NETWORK_NOT_PROVISIONED . . . 100
8.1.6.2 APP_NETWORK_PROVISIONED . . . 100
8.1.6.3 APP_NETWORK_CONF_INVALID . . . 100
8.1.7 Typedef Documentation . . . 100
8.1.7.1 app_event_handler_t . . . 100
8.1.7.2 app_p2p_role_data_t . . . 100
8.1.8 Enumeration Type Documentation . . . 101
8.1.8.1 wm_af_errno . . . 101
8.1.8.2 app_conn_status_t . . . 101
8.1.8.3 app_conn_failure_reason_t . . . 101
8.1.8.4 app_ctrl_event_t . . . 101
8.1.8.5 app_p2p_role_t . . . 103
8.1.9 Variable Documentation . . . 103
8.1.9.1 app_psm_hnd . . . 103
8.2 block_alloc.h File Reference . . . 103
8.2.1 Detailed Description . . . 103
8.2.2 Introduction . . . 103
8.2.3 Function Documentation . . . 104
8.2.3.1 ba_pool_create . . . 104
8.2.3.2 ba_pool_delete . . . 105
8.2.3.3 ba_block_allocate . . . 105
8.2.3.4 ba_block_release . . . 105
8.2.3.5 ba_cli_init . . . 105
8.3 board.h File Reference . . . 106
8.3.1 Detailed Description . . . 106
8.3.2 Function Documentation . . . 106
8.3.2.1 board_main_xtal . . . 106
8.3.2.2 board_main_osc . . . 106
8.3.2.3 board_cpu_freq . . . 106
8.3.2.4 board_32k_xtal . . . 107
8.3.2.5 board_32k_osc . . . 107
8.3.2.6 board_rc32k_calib . . . 107
8.3.2.7 board_card_detect . . . 107
8.3.2.8 board_sdio_pdn . . . 107
8.3.2.9 board_sdio_pwr . . . 107
8.3.2.10 board_sdio_reset . . . 108
8.3.2.11 board_sdio_pdn_support . . . 108

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.3.2.12 board_button_3 . . . 108
8.3.2.13 board_button_pressed . . . 108
8.3.2.14 board_gpio_power_on . . . 108
8.3.2.15 board_uart_pin_config . . . 108
8.3.2.16 board_i2c_pin_config . . . 109
8.3.2.17 board_ssp_pin_config . . . 109
8.3.2.18 board_usb_pin_config . . . 109
8.3.2.19 board_sdio_pin_config . . . 109
8.3.2.20 board_led_1 . . . 109
8.3.2.21 board_led_2 . . . 109
8.3.2.22 board_led_3 . . . 110
8.3.2.23 board_led_4 . . . 110
8.3.2.24 board_led_on . . . 110
8.3.2.25 board_led_off . . . 110
8.3.2.26 board_button_1 . . . 111
8.3.2.27 board_button_2 . . . 111
8.3.2.28 board_lcd_backlight_off . . . 111
8.3.2.29 board_lcd_backlight_on . . . 111
8.3.2.30 board_lcd_reset . . . 111
8.3.2.31 board_wakeup0_functional . . . 111
8.3.2.32 board_wakeup1_functional . . . 111
8.3.2.33 board_wakeup0_wifi . . . 112
8.3.2.34 board_wakeup1_wifi . . . 112
8.4 cli.h File Reference . . . 112
8.4.1 Detailed Description . . . 112
8.4.2 Usage . . . 112
8.4.3 Function Documentation . . . 112
8.4.3.1 cli_register_command . . . 112
8.4.3.2 cli_unregister_command . . . 112
8.4.3.3 cli_init . . . 113
8.4.3.4 cli_stop . . . 113
8.4.3.5 cli_register_commands . . . 113
8.4.3.6 cli_unregister_commands . . . 113
8.5 dhcp-server.h File Reference . . . 114
8.5.1 Detailed Description . . . 114
8.5.2 Function Documentation . . . 114
8.5.2.1 dhcpd_cli_init . . . 114
8.5.2.2 dhcp_server_start . . . 114
8.5.2.3 dhcp_server_stop . . . 114
8.5.2.4 dhcp_server_lease_timeout . . . 114

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.5.2.5 dhcp_get_ip_from_mac . . . 115
8.5.3 Enumeration Type Documentation . . . 115
8.5.3.1 wm_dhcpd_errno . . . 115
8.6 ezxml.h File Reference . . . 116
8.6.1 Detailed Description . . . 116
8.6.2 Usage . . . 116
8.6.3 Function Documentation . . . 116
8.6.3.1 ezxml_parse_str . . . 116
8.6.3.2 ezxml_child . . . 117
8.6.3.3 ezxml_idx . . . 117
8.6.3.4 ezxml_attr . . . 117
8.6.3.5 ezxml_get . . . 117
8.6.3.6 ezxml_toxml . . . 118
8.6.3.7 ezxml_pi . . . 118
8.6.3.8 ezxml_free . . . 118
8.6.3.9 ezxml_error . . . 118
8.6.3.10 ezxml_new . . . 119
8.6.3.11 ezxml_add_child . . . 119
8.6.3.12 ezxml_set_txt . . . 119
8.6.3.13 ezxml_set_attr . . . 119
8.6.3.14 ezxml_set_flag . . . 120
8.6.3.15 ezxml_cut . . . 120
8.6.3.16 ezxml_insert . . . 120
8.6.4 Macro Documentation . . . 120
8.6.4.1 ezxml_next . . . 120
8.6.4.2 ezxml_name . . . 120
8.6.4.3 ezxml_txt . . . 121
8.6.4.4 ezxml_new_d . . . 121
8.6.4.5 ezxml_add_child_d . . . 121
8.6.4.6 ezxml_set_txt_d . . . 121
8.6.4.7 ezxml_set_attr_d . . . 121
8.6.4.8 ezxml_move . . . 121
8.6.4.9 ezxml_remove . . . 121
8.6.5 Enumeration Type Documentation . . . 121
8.6.5.1 wm_exml_errno . . . 122
8.7 flash.h File Reference . . . 122
8.7.1 Detailed Description . . . 122
8.7.2 Function Documentation . . . 123
8.7.2.1 flash_drv_init . . . 123
8.7.2.2 flash_drv_open . . . 123

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.7.2.3 flash_drv_close . . . 123
8.7.2.4 flash_drv_write . . . 123
8.7.2.5 flash_drv_read . . . 124
8.7.2.6 flash_drv_erase . . . 124
8.7.2.7 flash_drv_eraseall . . . 124
8.7.2.8 flash_drv_get_id . . . 124
8.7.3 Typedef Documentation . . . 125
8.7.3.1 flash_desc_t . . . 125
8.8 ftfs.h File Reference . . . 125
8.8.1 Detailed Description . . . 125
8.8.2 Function Documentation . . . 125
8.8.2.1 ftfs_init . . . 125
8.8.2.2 ft_fopen . . . 125
8.8.2.3 ft_fclose . . . 126
8.8.2.4 ft_fread . . . 126
8.8.2.5 ft_ftell . . . 126
8.8.2.6 ft_fseek . . . 126
8.8.2.7 ft_read_header . . . 127
8.8.2.8 ftfs_cli_init . . . 127
8.8.3 Enumeration Type Documentation . . . 127
8.8.3.1 wm_ftfs_errno . . . 127
8.9 fw_upgrade_ed_chacha.h File Reference . . . 128
8.9.1 Detailed Description . . . 128
8.9.2 Function Documentation . . . 128
8.9.2.1 ed_chacha_client_mode_upgrade_fw . . . 128
8.9.2.2 ed_chacha_server_mode_upgrade_fw . . . 129
8.9.3 Macro Documentation . . . 129
8.9.3.1 ED_CHACHA_DECRYPT_KEY_LEN . . . 129
8.9.3.2 ED_CHACHA_VERIFICATION_KEY_LEN . . . 129
8.10 fw_upgrade_rsa_aes.h File Reference . . . 129
8.10.1 Detailed Description . . . 129
8.10.2 Function Documentation . . . 130
8.10.2.1 rsa2048_aes128_client_mode_upgrade_fw . . . 130
8.10.2.2 rsa2048_aes128_server_mode_upgrade_fw . . . 130
8.10.3 Macro Documentation . . . 131
8.10.3.1 RSA_PUBLIC_KEY_LEN . . . 131
8.10.3.2 AES_KEY_LEN . . . 131
8.11 generic_io.h File Reference . . . 131
8.11.1 Macro Documentation . . . 131
8.11.1.1 GPIO_ACTIVE_LOW . . . 131

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.11.1.2 GPIO_ACTIVE_HIGH . . . 131
8.12 healthmon.h File Reference . . . 131
8.12.1 Detailed Description . . . 131
8.12.2 Usage . . . 131
8.12.3 Function Documentation . . . 132
8.12.3.1 healthmon_init . . . 132
8.12.3.2 healthmon_start . . . 132
8.12.3.3 healthmon_register_handler . . . 132
8.12.3.4 healthmon_unregister_handler . . . 132
8.12.3.5 healthmon_change_check_interval . . . 133
8.12.3.6 healthmon_set_final_about_to_die_handler . . . 133
8.12.3.7 healthmon_register_cli . . . 133
8.12.4 Macro Documentation . . . 134
8.12.4.1 HM_NAME_MAX . . . 134
8.12.4.2 MAX_HM_HANDLERS . . . 134
8.12.4.3 WD_STROBE_INTERVAL . . . 134
8.12.4.4 WKUP_INTERVAL . . . 134
8.13 http_parse.h File Reference . . . 134
8.13.1 Detailed Description . . . 134
8.13.2 Function Documentation . . . 134
8.13.2.1 httpd_get_tag_from_post_data . . . 134
8.13.2.2 httpd_get_tag_from_url . . . 134
8.14 httpc.h File Reference . . . 135
8.14.1 Detailed Description . . . 135
8.14.2 Usage . . . 135
8.14.3 Function Documentation . . . 136
8.14.3.1 http_open_session . . . 136
8.14.3.2 http_prepare_req . . . 136
8.14.3.3 http_add_header . . . 136
8.14.3.4 http_send_request . . . 137
8.14.3.5 http_get_response_hdr . . . 137
8.14.3.6 http_get_response_hdr_value . . . 138
8.14.3.7 http_get_response_hdr_all . . . 139
8.14.3.8 http_read_content . . . 140
8.14.3.9 http_parse_URL . . . 140
8.14.3.10 http_lowlevel_read . . . 141
8.14.3.11 http_lowlevel_write . . . 141
8.14.3.12 httpc_write_chunked . . . 142
8.14.3.13 http_close_session . . . 142
8.14.3.14 http_setsockopt . . . 143

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.14.3.15 http_getsockopt . . . 143
8.14.3.16 http_get_sockfd_from_handle . . . 143
8.14.3.17 httpc_validate_url . . . 144
8.14.3.18 httpc_get . . . 144
8.14.4 Macro Documentation . . . 144
8.14.4.1 DEFAULT_RETRY_COUNT . . . 144
8.14.5 Enumeration Type Documentation . . . 144
8.14.5.1 http_hdr_field_sel_t . . . 145
8.14.5.2 wm_httpc_errno . . . 145
8.14.5.3 http_open_flags_t . . . 145
8.15 httpd.h File Reference . . . 145
8.15.1 Detailed Description . . . 145
8.15.2 Usage . . . 145
8.15.3 Web Services Gateway Interface (WSGI) . . . 146
8.15.4 Static File Handling . . . 147
8.15.4.1 Compressed Files . . . 147
8.15.5 Server Side Include (SSI) . . . 147
8.15.6 Other Considerations . . . 148
8.15.7 HTTPD Call Graph . . . 148
8.15.8 Function Documentation . . . 149
8.15.8.1 httpd_init . . . 149
8.15.8.2 httpd_shutdown . . . 149
8.15.8.3 httpd_start . . . 149
8.15.8.4 httpd_stop . . . 150
8.15.8.5 httpd_is_running . . . 150
8.15.8.6 httpd_send . . . 150
8.15.8.7 httpd_send_chunk . . . 150
8.15.8.8 httpd_send_chunk_begin . . . 151
8.15.8.9 httpd_get_priv_data . . . 151
8.15.8.10 httpd_clear_priv_data . . . 151
8.15.8.11 httpd_register_wsgi_handler . . . 152
8.15.8.12 httpd_register_wsgi_handlers . . . 152
8.15.8.13 httpd_unregister_wsgi_handler . . . 152
8.15.8.14 httpd_unregister_wsgi_handlers . . . 152
8.15.8.15 httpd_handle_file . . . 153
8.15.8.16 httpd_register_ssi . . . 153
8.15.8.17 httpd_unregister_ssi . . . 153
8.15.8.18 httpd_cli_init . . . 154
8.15.8.19 httpd_send_hdr_from_code . . . 154
8.15.8.20 httpd_validate_uri . . . 154

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.15.8.21 httpd_recv . . . 154
8.15.8.22 httpd_send_response . . . 155
8.15.8.23 httpd_send_response_header . . . 155
8.15.8.24 httpd_send_response_301 . . . 156
8.15.8.25 httpd_send_header . . . 156
8.15.8.26 httpd_send_default_headers . . . 157
8.15.8.27 httpd_purge_headers . . . 157
8.15.8.28 httpd_parse_hdr_tags_with_cb . . . 157
8.15.8.29 httpd_parse_hdr_tags . . . 158
8.15.8.30 httpd_get_data . . . 158
8.15.8.31 httpd_use_tls_certificates . . . 159
8.15.8.32 httpd_send_override . . . 159
8.15.8.33 httpd_recv_override . . . 159
8.15.8.34 httpd_purge_all_socket_data . . . 159
8.15.9 Macro Documentation . . . 159
8.15.9.1 HTTP_PORT . . . 159
8.15.9.2 HTTPS_PORT . . . 159
8.15.9.3 APP_HTTP_FLAGS_NO_EXACT_MATCH . . . 160
8.15.9.4 HTTPD_REQ_TYPE_UNKNOWN . . . 160
8.15.9.5 HTTPD_REQ_TYPE_POST . . . 160
8.15.9.6 HTTPD_REQ_TYPE_GET . . . 160
8.15.9.7 HTTPD_REQ_TYPE_PUT . . . 160
8.15.9.8 HTTPD_REQ_TYPE_DELETE . . . 160
8.15.9.9 HTTPD_REQ_TYPE_HEAD . . . 160
8.15.9.10 HTTPD_DONE . . . 160
8.15.9.11 HTTPD_MAX_MESSAGE . . . 160
8.15.9.12 HTTPD_MAX_URI_LENGTH . . . 161
8.15.9.13 HTTPD_MAX_VAL_LENGTH . . . 161
8.15.9.14 HTTPD_MAX_CONTENT_TYPE_LENGTH . . . 161
8.15.9.15 HTTP_CONTENT_JSON_STR . . . 161
8.15.9.16 HTTP_CONTENT_XML_STR . . . 161
8.15.9.17 HTTP_CONTENT_PLAIN_TEXT_STR . . . 161
8.15.9.18 HTTP_RES_200 . . . 162
8.15.9.19 HTTP_RES_204 . . . 162
8.15.9.20 HTTP_RES_207 . . . 162
8.15.9.21 HTTP_RES_301 . . . 162
8.15.9.22 HTTP_RES_304 . . . 162
8.15.9.23 HTTP_RES_400 . . . 162
8.15.9.24 HTTP_RES_401 . . . 162
8.15.9.25 HTTP_RES_404 . . . 162

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.15.9.26 HTTP_RES_405 . . . 162
8.15.9.27 HTTP_RES_429 . . . 162
8.15.9.28 HTTP_RES_470 . . . 162
8.15.9.29 HTTP_RES_500 . . . 162
8.15.9.30 HTTP_RES_505 . . . 163
8.15.9.31 HTTP_EVENT_200 . . . 163
8.15.9.32 httpd_send_crlf . . . 163
8.15.9.33 HTTPD_MAX_SSI_ARGS . . . 163
8.15.9.34 HTTPD_SSI_CALL . . . 163
8.15.9.35 MAX_REGISTERED_SSIS . . . 163
8.15.9.36 HTTPD_DEFAULT_HDR_FLAGS . . . 164
8.15.10 Typedef Documentation . . . 164
8.15.10.1 httpd_free_priv_data_fn_t . . . 164
8.15.10.2 httpd_parse_hdr_cb_t . . . 164
8.15.10.3 httpd_ssifunction . . . 164
8.15.10.4 httpd_send_func_t . . . 164
8.15.10.5 httpd_recv_func_t . . . 164
8.15.11 Enumeration Type Documentation . . . 165
8.15.11.1 wm_httpd_errno . . . 165
8.15.11.2 http_content_type . . . 165
8.15.11.3 httpd_hdr_field_sel_t . . . 165
8.16 json_generator.h File Reference . . . 165
8.16.1 Detailed Description . . . 165
8.16.2 Usage . . . 166
8.16.3 Function Documentation . . . 166
8.16.3.1 json_str_init . . . 166
8.16.3.2 json_start_object . . . 167
8.16.3.3 json_push_object . . . 167
8.16.3.4 json_push_array_object . . . 167
8.16.3.5 json_pop_array_object . . . 167
8.16.3.6 json_close_object . . . 168
8.16.3.7 json_start_array . . . 168
8.16.3.8 json_close_array . . . 168
8.16.4 Macro Documentation . . . 168
8.16.4.1 json_pop_object . . . 168
8.16.4.2 json_set_val_str . . . 169
8.16.4.3 json_set_val_int . . . 169
8.16.4.4 json_set_val_uint . . . 169
8.16.4.5 json_set_val_uint_64 . . . 169
8.16.4.6 json_set_val_float . . . 170

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.16.4.7 json_set_val_float_precision . . . 170
8.16.4.8 json_set_val_bool . . . 170
8.16.4.9 json_set_val_null . . . 171
8.16.4.10 json_set_array_str . . . 171
8.16.4.11 json_set_array_int . . . 171
8.16.4.12 json_set_array_float . . . 171
8.16.4.13 json_set_array_bool . . . 172
8.17 jsonv2.h File Reference . . . 172
8.17.1 Detailed Description . . . 172
8.17.2 Usage . . . 172
8.17.2.1 Search inside an Object . . . 172
8.17.2.2 Search inside an array . . . 173
8.17.3 Function Documentation . . . 174
8.17.3.1 json_init . . . 174
8.17.3.2 json_is_object . . . 174
8.17.3.3 json_is_array . . . 174
8.17.3.4 json_get_val_bool . . . 175
8.17.3.5 json_get_val_int . . . 175
8.17.3.6 json_get_val_int64 . . . 175
8.17.3.7 json_get_val_float . . . 176
8.17.3.8 json_get_val_str . . . 176
8.17.3.9 json_get_val_str_len . . . 177
8.17.3.10 json_get_composite_object . . . 177
8.17.3.11 json_release_composite_object . . . 177
8.17.3.12 json_get_array_object . . . 178
8.17.3.13 json_release_array_object . . . 178
8.17.3.14 json_array_get_num_elements . . . 178
8.17.3.15 json_array_get_bool . . . 179
8.17.3.16 json_array_get_int . . . 179
8.17.3.17 json_array_get_int64 . . . 179
8.17.3.18 json_array_get_float . . . 180
8.17.3.19 json_array_get_str . . . 180
8.17.3.20 json_array_get_str_len . . . 181
8.17.3.21 json_array_get_composite_object . . . 181
8.17.3.22 json_array_release_composite_object . . . 181
8.17.3.23 json_array_get_array_object . . . 182
8.17.3.24 json_array_release_array_object . . . 182
8.17.4 Enumeration Type Documentation . . . 182
8.17.4.1 wm_json_errno . . . 182
8.18 keystore.h File Reference . . . 183

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.18.1 Detailed Description . . . 183
8.18.2 Usage . . . 183
8.18.3 Function Documentation . . . 184
8.18.3.1 keystore_get . . . 184
8.18.3.2 keystore_get_ref . . . 184
8.18.3.3 keystore_iterate_start . . . 184
8.18.3.4 keystore_iterate . . . 184
8.18.3.5 keystore_iterate_ref . . . 185
8.18.4 Enumeration Type Documentation . . . 185
8.18.4.1 ks_tlv_type . . . 185
8.19 led_indicator.h File Reference . . . 186
8.19.1 Function Documentation . . . 186
8.19.1.1 led_on . . . 186
8.19.1.2 led_off . . . 186
8.19.1.3 led_blink . . . 186
8.19.1.4 led_get_state . . . 186
8.19.1.5 led_fast_blink . . . 186
8.19.1.6 led_slow_blink . . . 187
8.19.2 Enumeration Type Documentation . . . 187
8.19.2.1 led_state . . . 187
8.20 mdev.h File Reference . . . 187
8.20.1 Detailed Description . . . 187
8.20.2 Usage . . . 187
8.20.3 Function Documentation . . . 187
8.20.3.1 mdev_register . . . 187
8.20.3.2 mdev_deregister . . . 188
8.20.3.3 mdev_get_handle . . . 188
8.21 mdev_acomp.h File Reference . . . 188
8.21.1 Detailed Description . . . 188
8.21.2 Function Documentation . . . 189
8.21.2.1 acomp_drv_init . . . 189
8.21.2.2 acomp_drv_open . . . 189
8.21.2.3 acomp_drv_result . . . 189
8.21.2.4 acomp_drv_close . . . 190
8.21.2.5 acomp_drv_deinit . . . 190
8.22 mdev_adc.h File Reference . . . 190
8.22.1 Detailed Description . . . 190
8.22.2 Function Documentation . . . 191
8.22.2.1 adc_drv_init . . . 191
8.22.2.2 adc_get_config . . . 191

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.22.2.3 adc_modify_default_config . . . 192
8.22.2.4 adc_drv_open . . . 192
8.22.2.5 adc_drv_result . . . 192
8.22.2.6 adc_drv_timeout . . . 192
8.22.2.7 adc_drv_get_samples . . . 193
8.22.2.8 adc_drv_close . . . 193
8.22.2.9 adc_drv_deinit . . . 193
8.23 mdev_aes.h File Reference . . . 193
8.23.1 Detailed Description . . . 193
8.23.2 Function Documentation . . . 195
8.23.2.1 aes_drv_init . . . 195
8.23.2.2 aes_drv_open . . . 195
8.23.2.3 aes_drv_setkey . . . 195
8.23.2.4 aes_drv_ccm_setkey . . . 196
8.23.2.5 aes_drv_encrypt . . . 196
8.23.2.6 aes_drv_ccm_encrypt . . . 196
8.23.2.7 aes_drv_decrypt . . . 197
8.23.2.8 aes_drv_ccm_decrypt . . . 197
8.23.2.9 aes_ctr_final . . . 198
8.23.2.10 aes_drv_close . . . 198
8.23.3 Enumeration Type Documentation . . . 198
8.23.3.1 hw_aes_mode_t . . . 198
8.24 mdev_crc.h File Reference . . . 198
8.24.1 Detailed Description . . . 198
8.24.2 Function Documentation . . . 200
8.24.2.1 crc_drv_init . . . 200
8.24.2.2 crc_drv_open . . . 200
8.24.2.3 crc_drv_block_calc . . . 200
8.24.2.4 crc_drv_stream_setlen . . . 201
8.24.2.5 crc_drv_stream_feed . . . 201
8.24.2.6 crc_drv_stream_get_result . . . 201
8.24.2.7 crc_drv_close . . . 201
8.24.3 Enumeration Type Documentation . . . 202
8.24.3.1 crc_mode_t . . . 202
8.25 mdev_dac.h File Reference . . . 202
8.25.1 Detailed Description . . . 202
8.25.2 Function Documentation . . . 203
8.25.2.1 dac_drv_init . . . 203
8.25.2.2 dac_modify_default_config . . . 203
8.25.2.3 dac_drv_open . . . 203

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.25.2.4 dac_drv_output . . . 204
8.25.2.5 dac_drv_close . . . 204
8.25.2.6 dac_drv_deinit . . . 204
8.26 mdev_dma.h File Reference . . . 204
8.26.1 Detailed Description . . . 204
8.26.2 Function Documentation . . . 205
8.26.2.1 dma_drv_init . . . 205
8.26.2.2 dma_drv_open . . . 206
8.26.2.3 dma_drv_set_cb . . . 206
8.26.2.4 dma_drv_transfer . . . 206
8.26.2.5 dma_drv_wait_for_transfer_complete . . . 207
8.26.2.6 dma_drv_close . . . 207
8.26.2.7 dma_drv_deinit . . . 208
8.27 mdev_fpu.h File Reference . . . 208
8.27.1 Macro Documentation . . . 208
8.27.1.1 fpu_init . . . 208
8.27.1.2 fpu_deinit . . . 208
8.28 mdev_gpio.h File Reference . . . 208
8.28.1 Detailed Description . . . 208
8.28.2 Usage . . . 208
8.28.3 Function Documentation . . . 210
8.28.3.1 gpio_drv_init . . . 210
8.28.3.2 gpio_drv_open . . . 210
8.28.3.3 gpio_drv_close . . . 210
8.28.3.4 gpio_drv_setdir . . . 211
8.28.3.5 gpio_drv_write . . . 211
8.28.3.6 gpio_drv_read . . . 211
8.28.3.7 gpio_drv_set_cb . . . 212
8.28.3.8 gpio_drv_get_io_domain . . . 212
8.28.4 Typedef Documentation . . . 212
8.28.4.1 gpio_irq_cb . . . 212
8.29 mdev_gpt.h File Reference . . . 213
8.29.1 Detailed Description . . . 213
8.29.2 Usage . . . 213
8.29.3 Function Documentation . . . 213
8.29.3.1 gpt_drv_init . . . 213
8.29.3.2 gpt_drv_open . . . 214
8.29.3.3 gpt_drv_close . . . 214
8.29.3.4 gpt_drv_set . . . 214
8.29.3.5 gpt_drv_setcb . . . 215

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.29.3.6 gpt_drv_start . . . 215
8.29.3.7 gpt_drv_stop . . . 215
8.29.3.8 gpt_drv_pwm . . . 215
8.30 mdev_i2c.h File Reference . . . 216
8.30.1 Detailed Description . . . 216
8.30.2 Usage . . . 216
8.30.3 Function Documentation . . . 217
8.30.3.1 i2c_drv_init . . . 217
8.30.3.2 i2c_drv_set_clkcnt . . . 217
8.30.3.3 i2c_drv_rxbuf_size . . . 218
8.30.3.4 i2c_drv_xfer_mode . . . 218
8.30.3.5 i2c_drv_open . . . 218
8.30.4 of flags passed to i2c_drv_set_address: . . . 219
8.30.4.1 i2c_drv_set_callback . . . 219
8.30.4.2 i2c_drv_set_address . . . 220
8.30.5 of flags passed to i2c_drv_set_address: . . . 220
8.30.5.1 i2c_drv_wait_till_inactivity . . . 220
8.30.5.2 i2c_drv_close . . . 220
8.30.5.3 i2c_drv_read . . . 221
8.30.5.4 i2c_drv_slv_dma_read . . . 221
8.30.5.5 i2c_drv_write . . . 221
8.30.5.6 i2c_drv_dma_write . . . 222
8.30.5.7 i2c_drv_dma_read . . . 222
8.30.5.8 i2c_drv_timeout . . . 223
8.30.5.9 i2c_drv_deinit . . . 223
8.30.6 Macro Documentation . . . 223
8.30.6.1 NUM_I2C_PORTS . . . 223
8.30.6.2 BIT10 . . . 223
8.30.6.3 I2C_CLK_100KHZ . . . 223
8.30.6.4 I2C_CLK_10KHZ . . . 224
8.30.6.5 I2C_CLK_400KHZ . . . 224
8.30.6.6 I2C_DEVICE_SLAVE . . . 224
8.30.6.7 I2C_SLAVEADR . . . 224
8.30.6.8 I2C_10_BIT_ADDR . . . 224
8.30.7 Enumeration Type Documentation . . . 224
8.30.7.1 I2C_DMA_MODE . . . 224
8.30.7.2 i2c_status . . . 224
8.31 mdev_pinmux.h File Reference . . . 224
8.31.1 Detailed Description . . . 224
8.31.2 Usage . . . 225

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.31.3 Function Documentation . . . 225
8.31.3.1 pinmux_drv_init . . . 225
8.31.3.2 pinmux_drv_open . . . 225
8.31.3.3 pinmux_drv_close . . . 226
8.31.3.4 pinmux_drv_setfunc . . . 226
8.31.3.5 pinmux_drv_get_gpio_func . . . 226
8.32 mdev_pm.h File Reference . . . 226
8.32.1 Detailed Description . . . 226
8.32.2 MCU Power Modes relationship Table. . . . 227
8.32.3 Function Documentation . . . 228
8.32.3.1 pm_mcu_cfg . . . 228
8.32.3.2 pm_mcu_state . . . 228
8.32.3.3 pm_wakeup_source . . . 229
8.32.3.4 pm_mcu_is_enabled . . . 229
8.32.3.5 pm_mcu_get_mode . . . 229
8.32.3.6 pm_mcu_io_cfg . . . 229
8.32.3.7 arch_pm_init . . . 229
8.32.3.8 arch_reboot . . . 230
8.32.4 Macro Documentation . . . 230
8.32.4.1 pmu_drv_poweron_vddio . . . 230
8.32.4.2 pmu_drv_poweroff_vddio . . . 230
8.32.4.3 pmu_drv_powerdeepoff_vddio . . . 231
8.32.4.4 pmu_drv_powernormal_vddio . . . 231
8.32.5 Enumeration Type Documentation . . . 231
8.32.5.1 wakeup_cause . . . 231
8.32.5.2 io_domain_t . . . 232
8.32.5.3 power_state_t . . . 232
8.33 mdev_rtc.h File Reference . . . 232
8.33.1 Detailed Description . . . 232
8.33.2 Function Documentation . . . 233
8.33.2.1 rtc_drv_init . . . 233
8.33.2.2 rtc_drv_open . . . 234
8.33.2.3 rtc_drv_set . . . 234
8.33.2.4 rtc_drv_set_cb . . . 234
8.33.2.5 rtc_drv_set_alarm_cb . . . 234
8.33.2.6 rtc_drv_reset . . . 234
8.33.2.7 rtc_drv_start . . . 235
8.33.2.8 rtc_drv_get . . . 235
8.33.2.9 rtc_drv_get_uppval . . . 235
8.33.2.10 rtc_drv_set_alarm . . . 235

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.33.2.11 rtc_drv_stop . . . 236
8.33.2.12 rtc_drv_close . . . 236
8.34 mdev_sdio.h File Reference . . . 236
8.34.1 Detailed Description . . . 236
8.34.2 Usage . . . 236
8.34.3 Function Documentation . . . 237
8.34.3.1 sdio_drv_creg_read . . . 237
8.34.3.2 sdio_drv_creg_write . . . 237
8.34.3.3 sdio_drv_read . . . 237
8.34.3.4 sdio_drv_write . . . 238
8.34.3.5 sdio_drv_open . . . 238
8.34.3.6 sdio_drv_init . . . 238
8.34.3.7 sdio_drv_deinit . . . 238
8.34.3.8 sdio_drv_set_cb1 . . . 239
8.35 mdev_ssp.h File Reference . . . 239
8.35.1 Detailed Description . . . 239
8.35.2 Usage . . . 239
8.35.3 Default Parameter Configuration . . . 240
8.35.4 Function Documentation . . . 240
8.35.4.1 ssp_drv_init . . . 240
8.35.4.2 ssp_drv_deinit . . . 240
8.35.4.3 ssp_drv_open . . . 241
8.35.4.4 ssp_drv_close . . . 241
8.35.4.5 ssp_drv_write . . . 241
8.35.4.6 ssp_drv_read . . . 242
8.35.4.7 ssp_drv_cs_activate . . . 242
8.35.4.8 ssp_drv_cs_deactivate . . . 242
8.35.4.9 ssp_drv_set_clk . . . 243
8.35.4.10 ssp_drv_rxbuf_size . . . 243
8.35.4.11 ssp_drv_timeout . . . 243
8.35.5 Enumeration Type Documentation . . . 244
8.35.5.1 SSP_DMA . . . 244
8.36 mdev_uart.h File Reference . . . 244
8.36.1 Detailed Description . . . 244
8.36.2 Usage . . . 244
8.36.3 Function Documentation . . . 245
8.36.3.1 uart_drv_init . . . 245
8.36.3.2 uart_drv_deinit . . . 245
8.36.3.3 uart_drv_xfer_mode . . . 245
8.36.3.4 uart_drv_open . . . 246

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.36.3.5 uart_drv_close . . . 246
8.36.3.6 uart_drv_read . . . 246
8.36.3.7 uart_drv_write . . . 247
8.36.3.8 uart_drv_get_portid . . . 247
8.36.3.9 uart_drv_tx_flush . . . 247
8.36.3.10 uart_drv_rxbuf_size . . . 247
8.36.3.11 uart_drv_dma_rd_blk_size . . . 248
8.36.3.12 uart_drv_set_opts . . . 248
8.36.3.13 uart_drv_blocking_read . . . 248
8.36.3.14 uart_drv_rx_buf_reset . . . 249
8.36.4 Macro Documentation . . . 249
8.36.4.1 UART_8BIT . . . 249
8.36.4.2 UART_9BIT . . . 249
8.36.4.3 UARTDRV_SIZE_RX_BUF . . . 249
8.36.4.4 UARTDRV_THRESHOLD_RX_BUF . . . 249
8.36.5 Enumeration Type Documentation . . . 249
8.36.5.1 UART_DMA_MODE . . . 249
8.36.5.2 flow_control_t . . . 250
8.37 mdev_usb.h File Reference . . . 250
8.37.1 Detailed Description . . . 250
8.37.2 Usage . . . 250
8.37.3 Function Documentation . . . 251
8.37.3.1 usb_drv_init . . . 251
8.37.3.2 usb_drv_open . . . 251
8.37.3.3 usb_drv_close . . . 251
8.37.3.4 usb_drv_read . . . 251
8.37.3.5 usb_drv_write . . . 252
8.37.4 Macro Documentation . . . 252
8.37.4.1 WM_E_USB_TIMEOUT_ERROR . . . 252
8.37.4.2 WM_E_USB_CONNECTION_ERROR . . . 252
8.37.5 Enumeration Type Documentation . . . 252
8.37.5.1 usb_class . . . 252
8.38 mdev_usb_host.h File Reference . . . 253
8.38.1 Detailed Description . . . 253
8.38.2 Usage . . . 253
8.38.3 Function Documentation . . . 254
8.38.3.1 usb_host_uac_drv_init . . . 254
8.38.3.2 usb_host_uvc_drv_init . . . 254
8.38.3.3 usb_host_cdc_drv_init . . . 254
8.38.3.4 usb_host_msc_drv_init . . . 254

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.38.3.5 usb_host_drv_open . . . 255
8.38.3.6 usb_host_drv_start . . . 255
8.38.3.7 usb_host_drv_stop . . . 255
8.38.3.8 usb_host_drv_info . . . 256
8.38.3.9 usb_host_drv_read . . . 256
8.38.3.10 usb_host_drv_write . . . 256
8.38.3.11 usb_host_drv_block_read . . . 257
8.38.3.12 usb_host_drv_block_write . . . 257
8.38.3.13 usb_host_drv_close . . . 257
8.38.4 Macro Documentation . . . 257
8.38.4.1 USB_UVC_DEBUG . . . 258
8.38.4.2 USB_CDC_DEBUG . . . 258
8.39 mdev_wdt.h File Reference . . . 258
8.39.1 Detailed Description . . . 258
8.39.2 Usage . . . 258
8.39.3 Function Documentation . . . 258
8.39.3.1 wdt_drv_init . . . 258
8.39.3.2 wdt_drv_open . . . 259
8.39.3.3 wdt_drv_set_timeout . . . 259
8.39.3.4 wdt_drv_start . . . 259
8.39.3.5 wdt_drv_strobe . . . 259
8.40 mdns.h File Reference . . . 260
8.40.1 Detailed Description . . . 260
8.40.2 Example Usage: . . . 260
8.40.2.1 Working with mDNS Responder (Announcing the service) . . . 260
8.40.2.2 Working with mDNS Querier (Querying the service) . . . 261
8.40.2.3 Build Options . . . 262
8.40.3 Function Documentation . . . 262
8.40.3.1 mdns_start . . . 262
8.40.3.2 mdns_stop . . . 263
8.40.3.3 mdns_query_monitor . . . 263
8.40.3.4 mdns_query_unmonitor . . . 263
8.40.3.5 mdns_announce_service . . . 264
8.40.3.6 mdns_deannounce_service . . . 264
8.40.3.7 mdns_iface_state_change . . . 265
8.40.3.8 mdns_set_txt_rec . . . 265
8.40.4 Macro Documentation . . . 266
8.40.4.1 MDNS_MAX_LABEL_LEN . . . 266
8.40.4.2 MDNS_MAX_NAME_LEN . . . 266
8.40.4.3 MDNS_MAX_KEYVAL_LEN . . . 266

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.40.4.4 MAX_MDNS_LST . . . 266
8.40.4.5 MAX_INTERFACE . . . 266
8.40.4.6 MDNS_PROTO_TCP . . . 266
8.40.4.7 MDNS_PROTO_UDP . . . 267
8.40.5 Typedef Documentation . . . 267
8.40.5.1 mdns_query_cb . . . 267
8.40.6 Enumeration Type Documentation . . . 268
8.40.6.1 iface_state . . . 268
8.40.6.2 wm_mdns_errno . . . 268
8.41 nw_utils.h File Reference . . . 269
8.41.1 Detailed Description . . . 269
8.41.2 Function Documentation . . . 269
8.41.2.1 nw_utils_cli_init . . . 269
8.41.2.2 nw_utils_cli_deinit . . . 269
8.42 overlays.h File Reference . . . 269
8.42.1 Detailed Description . . . 269
8.42.2 Function Documentation . . . 270
8.42.2.1 overlay_load . . . 270
8.43 p2p.h File Reference . . . 270
8.43.1 Detailed Description . . . 270
8.43.2 Usage . . . 270
8.43.3 Function Documentation . . . 271
8.43.3.1 p2p_start . . . 271
8.43.3.2 p2p_stop . . . 271
8.43.3.3 p2p_scan . . . 271
8.43.3.4 p2p_connect . . . 271
8.43.3.5 p2p_disconnect . . . 272
8.43.3.6 p2p_generate_pin . . . 272
8.43.3.7 p2p_validate_pin . . . 272
8.43.4 Macro Documentation . . . 272
8.43.4.1 P2P_LOG . . . 272
8.43.4.2 P2P_GROUP_OWNER_MASK . . . 273
8.43.4.3 MAX_DEVICE_NAME_LEN . . . 273
8.43.5 Enumeration Type Documentation . . . 273
8.43.5.1 p2p_session_command . . . 273
8.43.5.2 p2p_event . . . 273
8.44 partition.h File Reference . . . 273
8.44.1 Detailed Description . . . 273
8.44.2 Usage . . . 274
8.44.3 Function Documentation . . . 274

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.44.3.1 part_init . . . 274
8.44.3.2 part_to_flash_desc . . . 274
8.44.3.3 part_get_layout_by_id . . . 274
8.44.3.4 part_get_layout_by_name . . . 274
8.44.3.5 part_get_active_partition . . . 275
8.44.3.6 part_set_active_partition . . . 275
8.44.4 Enumeration Type Documentation . . . 275
8.44.4.1 flash_comp . . . 275
8.45 provisioning.h File Reference . . . 276
8.45.1 Detailed Description . . . 276
8.45.2 Introduction . . . 276
8.45.2.1 WLAN network (micro-AP) based provisioning . . . 276
8.45.2.2 WPS based provisioning . . . 276
8.45.2.3 EZConnect provisioning . . . 277
8.45.3 Function Documentation . . . 277
8.45.3.1 prov_start . . . 277
8.45.3.2 prov_finish . . . 278
8.45.3.3 prov_ezconnect_start . . . 278
8.45.3.4 prov_ezconnect_finish . . . 278
8.45.3.5 prov_get_scan_config . . . 278
8.45.3.6 prov_set_scan_config . . . 278
8.45.3.7 prov_set_scan_config_no_cb . . . 279
8.45.3.8 verify_scan_interval_value . . . 279
8.45.3.9 register_provisioning_web_handlers . . . 279
8.45.3.10 unregister_provisioning_web_handlers . . . 279
8.45.3.11 register_secure_provisioning_web_handlers . . . 280
8.45.3.12 unregister_secure_provisioning_web_handlers . . . 280
8.45.3.13 prov_register_secure_provisioning_pin . . . 280
8.45.3.14 prov_unregister_secure_provisioning_pin . . . 280
8.45.3.15 register_ezconnect_provisioning_ssid . . . 280
8.45.3.16 sys_wq_enqueue_job_wlan_scan . . . 281
8.45.3.17 sys_wq_dequeue_job_wlan_scan . . . 281
8.45.3.18 set_network_cb_register . . . 281
8.45.3.19 prov_set_scan_policy . . . 281
8.45.3.20 prov_get_lock_scan_results . . . 282
8.45.3.21 prov_put_lock_scan_results . . . 282
8.45.3.22 prov_get_scan_count . . . 282
8.45.3.23 prov_get_scan_result . . . 282
8.45.3.24 prov_set_network . . . 282
8.45.4 Macro Documentation . . . 283

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.45.4.1 PROVISIONING_WLANNW . . . 283
8.45.4.2 PROVISIONING_WPS . . . 283
8.45.4.3 PROVISIONING_EZCONNECT . . . 283
8.45.5 Enumeration Type Documentation . . . 283
8.45.5.1 prov_scan_policy . . . 283
8.45.5.2 prov_event . . . 283
8.45.5.3 provisioning_state . . . 284
8.46 psm-v2.h File Reference . . . 284
8.46.1 Detailed Description . . . 285
8.46.2 Compaction process . . . 285
8.46.3 Terminology . . . 285
8.46.4 Memory usage of PSM module . . . 285
8.46.5 Thread safety . . . 286
8.46.6 Read only usage . . . 286
8.46.7 Usage . . . 286
8.46.8 Indexing and caching . . . 287
8.46.9 PSM Encryption (Secure PSM) . . . 287
8.46.10 Function Documentation . . . 287
8.46.10.1 psm_module_init . . . 287
8.46.10.2 psm_object_open . . . 288
8.46.10.3 psm_object_read . . . 289
8.46.10.4 psm_object_write . . . 289
8.46.10.5 psm_object_close . . . 290
8.46.10.6 psm_object_delete . . . 290
8.46.10.7 psm_format . . . 291
8.46.10.8 psm_objects_dump . . . 291
8.46.10.9 psm_set_variable . . . 291
8.46.10.10 psm_set_variable_str . . . 292
8.46.10.11 psm_set_variable_int . . . 292
8.46.10.12 psm_get_variable . . . 292
8.46.10.13 psm_get_variable_str . . . 293
8.46.10.14 psm_get_variable_int . . . 293
8.46.10.15 psm_module_deinit . . . 294
8.46.11 Typedef Documentation . . . 294
8.46.11.1 psm_hnd_t . . . 294
8.46.11.2 psm_object_handle_t . . . 294
8.46.12 Enumeration Type Documentation . . . 294
8.46.12.1 psm_mode_t . . . 294
8.46.12.2 psm_flags_t . . . 294
8.47 psm.h File Reference . . . 295

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.47.1 Detailed Description . . . 295
8.47.2 Macro Documentation . . . 295
8.47.2.1 PSM_CREAT . . . 295
8.48 push_button.h File Reference . . . 295
8.48.1 Function Documentation . . . 295
8.48.1.1 push_button_set_cb . . . 295
8.48.1.2 push_button_reset_cb . . . 296
8.48.2 Macro Documentation . . . 297
8.48.2.1 MAX_PB_CB . . . 297
8.48.2.2 PB_DEF_DEBOUNCE_TIME . . . 297
8.49 pwrmgr.h File Reference . . . 297
8.49.1 Detailed Description . . . 297
8.49.2 Usage . . . 297
8.49.3 Function Documentation . . . 297
8.49.3.1 pm_register_cb . . . 297
8.49.3.2 pm_deregister_cb . . . 298
8.49.3.3 pm_change_event . . . 298
8.49.3.4 pm_init . . . 298
8.49.3.5 is_pm_init_done . . . 299
8.49.3.6 pm_reboot_soc . . . 299
8.49.3.7 wakelock_get . . . 299
8.49.3.8 wakelock_put . . . 300
8.49.3.9 wakelock_isheld . . . 300
8.49.3.10 pm_cli_init . . . 300
8.49.3.11 pm_mcu_cli_init . . . 300
8.49.4 Typedef Documentation . . . 301
8.49.4.1 power_save_state_change_cb_fn . . . 301
8.49.5 Enumeration Type Documentation . . . 301
8.49.5.1 power_save_event_t . . . 301
8.50 rfget.h File Reference . . . 301
8.50.1 Detailed Description . . . 301
8.50.2 Enumeration Type Documentation . . . 302
8.50.2.1 wm_rfget_code . . . 302
8.51 secure_upgrade.h File Reference . . . 302
8.51.1 Detailed Description . . . 302
8.51.2 Secure Firmware upgrade image format . . . 303
8.51.3 Firmware Upgrade Algorithm . . . 303
8.51.4 Function Documentation . . . 304
8.51.4.1 client_mode_secure_upgrade . . . 304
8.51.4.2 server_mode_secure_upgrade . . . 304

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.51.5 Typedef Documentation . . . 304
8.51.5.1 decrypt_init_func_t . . . 304
8.51.5.2 decrypt_func_t . . . 305
8.51.5.3 signature_verify_func_t . . . 305
8.51.5.4 hash_update_func_t . . . 305
8.51.5.5 hash_finish_func_t . . . 306
8.51.5.6 fw_data_read_fn_t . . . 306
8.52 telnetd.h File Reference . . . 306
8.52.1 Function Documentation . . . 306
8.52.1.1 telnetd_start . . . 306
8.52.1.2 telnetd_stop . . . 306
8.52.2 Enumeration Type Documentation . . . 307
8.52.2.1 wm_telnetd_errno . . . 307
8.53 websockets.h File Reference . . . 307
8.53.1 Detailed Description . . . 307
8.53.2 Function Documentation . . . 307
8.53.2.1 ws_upgrade_socket . . . 307
8.53.2.2 ws_frame_send . . . 307
8.53.2.3 ws_frame_recv . . . 308
8.53.2.4 ws_close_ctx . . . 308
8.53.2.5 ws_set_genmask_callback . . . 308
8.53.3 Typedef Documentation . . . 308
8.53.3.1 ws_frame_t . . . 308
8.53.4 Enumeration Type Documentation . . . 308
8.53.4.1 ws_event_t . . . 309
8.54 wifi-decl.h File Reference . . . 309
8.54.1 Macro Documentation . . . 309
8.54.1.1 MLAN_MAX_VER_STR_LEN . . . 309
8.54.1.2 BSS_TYPE_STA . . . 309
8.54.1.3 BSS_TYPE_UAP . . . 309
8.54.1.4 MLAN_MAX_SSID_LENGTH . . . 309
8.54.2 Typedef Documentation . . . 309
8.54.2.1 wifi_channel_desc_t . . . 309
8.54.2.2 wifi_chan_trpc_entry_t . . . 309
8.54.2.3 chan_trpc_config_t . . . 309
8.54.2.4 chan_trpc_t . . . 310
8.54.3 Variable Documentation . . . 310
8.54.3.1 start_freq . . . 310
8.54.3.2 chan_width . . . 310
8.54.3.3 chan_num . . . 310

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.54.3.4 mod_group . . . 310
8.54.3.5 tx_power . . . 310
8.54.3.6 num_mod_grps . . . 310
8.54.3.7 chan_desc . . . 310
8.54.3.8 chan_trpc_entry . . . 310
8.54.3.9 num_chans . . . 310
8.54.3.10 chan_trpc_config . . . 310
8.55 wifi_events.h File Reference . . . 311
8.55.1 Enumeration Type Documentation . . . 311
8.55.1.1 wifi_event . . . 311
8.55.1.2 wlan_bss_type . . . 311
8.55.1.3 wlan_bss_role . . . 311
8.55.1.4 wifi_wakeup_event_t . . . 311
8.55.1.5 wifi_state . . . 311
8.56 wlan.h File Reference . . . 312
8.56.1 Detailed Description . . . 312
8.56.2 Usage . . . 312
8.56.3 Station State Machine . . . 312
8.56.4 Micro-AP State Machine . . . 313
8.56.5 Function Documentation . . . 314
8.56.5.1 wlan_init . . . 314
8.56.5.2 wlan_start . . . 314
8.56.5.3 wlan_stop . . . 315
8.56.5.4 wlan_deinit . . . 315
8.56.5.5 wlan_initialize_uap_network . . . 315
8.56.5.6 wlan_add_network . . . 315
8.56.5.7 wlan_remove_network . . . 316
8.56.5.8 wlan_connect . . . 316
8.56.5.9 wlan_disconnect . . . 317
8.56.5.10 wlan_start_network . . . 317
8.56.5.11 wlan_stop_network . . . 318
8.56.5.12 wlan_get_mac_address . . . 318
8.56.5.13 wlan_get_wfd_mac_address . . . 318
8.56.5.14 wlan_get_address . . . 319
8.56.5.15 wlan_get_uap_address . . . 319
8.56.5.16 wlan_get_wfd_address . . . 320
8.56.5.17 wlan_get_current_network . . . 320
8.56.5.18 wlan_get_current_uap_network . . . 320
8.56.5.19 wlan_get_current_wfd_network . . . 321
8.56.5.20 is_uap_started . . . 321

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.56.5.21 is_sta_connected . . . 321
8.56.5.22 is_sta_ipv4_connected . . . 321
8.56.5.23 is_wfd_started . . . 322
8.56.5.24 wlan_get_network . . . 322
8.56.5.25 wlan_get_network_byname . . . 322
8.56.5.26 wlan_get_network_count . . . 323
8.56.5.27 wlan_get_connection_state . . . 323
8.56.5.28 wlan_get_uap_connection_state . . . 323
8.56.5.29 wlan_get_wfd_connection_state . . . 324
8.56.5.30 wlan_scan . . . 324
8.56.5.31 wlan_get_scan_result . . . 324
8.56.5.32 wlan_enable_ed_mac_mode . . . 325
8.56.5.33 wlan_set_cal_data . . . 325
8.56.5.34 wlan_set_mac_addr . . . 325
8.56.5.35 wlan_configure_listen_interval . . . 326
8.56.5.36 wlan_configure_null_pkt_interval . . . 326
8.56.5.37 wlan_deepsleepps_on . . . 327
8.56.5.38 wlan_deepsleepps_off . . . 327
8.56.5.39 wlan_pdnps_on . . . 327
8.56.5.40 wlan_pdnps_off . . . 327
8.56.5.41 wlan_get_current_ant . . . 328
8.56.5.42 wlan_set_current_ant . . . 328
8.56.5.43 wlan_get_firmware_version_ext . . . 328
8.56.5.44 wlan_uap_inact_sleep_ps_on . . . 328
8.56.5.45 wlan_uap_inact_sleep_ps_off . . . 329
8.56.5.46 wlan_get_ps_mode . . . 329
8.56.5.47 wlan_get_uap_ps_mode . . . 329
8.56.5.48 wlan_get_wfd_ps_mode . . . 329
8.56.5.49 wlan_wlcmgr_send_msg . . . 330
8.56.5.50 wlan_basic_cli_init . . . 330
8.56.5.51 wlan_cli_init . . . 330
8.56.5.52 wlan_iw_cli_init . . . 331
8.56.5.53 wlan_uaputl_cli_init . . . 331
8.56.5.54 wlan_enhanced_cli_init . . . 331
8.56.5.55 wlan_pm_cli_init . . . 332
8.56.5.56 pm_ieeeps_hs_cfg . . . 332
8.56.5.57 load_wep_key . . . 332
8.56.5.58 wlan_sniffer_start . . . 333
8.56.5.59 wlan_sniffer_status . . . 333
8.56.5.60 wlan_sniffer_stop . . . 334

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.56.5.61 wlan_get_uap_supported_max_clients . . . 334
8.56.5.62 wlan_get_uap_max_clients . . . 334
8.56.5.63 wlan_set_uap_max_clients . . . 334
8.56.5.64 wlan_get_sta_tx_power . . . 335
8.56.5.65 wlan_set_sta_tx_power . . . 335
8.56.5.66 wlan_get_sta_tx_rate_index . . . 335
8.56.5.67 wlan_set_sta_tx_rate_index . . . 335
8.56.5.68 wlan_get_mgmt_ie . . . 335
8.56.5.69 wlan_set_mgmt_ie . . . 336
8.56.5.70 wlan_clear_mgmt_ie . . . 336
8.56.5.71 wlan_get_11d_enable_status . . . 336
8.56.5.72 wlan_get_current_signal_strength . . . 336
8.56.5.73 wlan_inject_frame . . . 337
8.56.5.74 wlan_remain_on_channel . . . 337
8.56.5.75 wlan_get_otp_user_data . . . 337
8.56.5.76 wlan_get_cal_data . . . 338
8.56.5.77 wlan_set_trpc . . . 338
8.56.5.78 wlan_auto_reconnect_enable . . . 338
8.56.5.79 wlan_auto_reconnect_disable . . . 339
8.56.5.80 wlan_get_auto_reconnect_config . . . 339
8.56.6 Macro Documentation . . . 339
8.56.6.1 WLAN_RESCAN_LIMIT . . . 339
8.56.6.2 WLAN_NETWORK_NAME_MIN_LENGTH . . . 339
8.56.6.3 WLAN_NETWORK_NAME_MAX_LENGTH . . . 340
8.56.6.4 WLAN_PSK_MAX_LENGTH . . . 340
8.56.6.5 WLAN_PMK_LENGTH . . . 340
8.56.6.6 MAX_IPV6_ADDRESSES . . . 340
8.56.6.7 WLAN_ERROR_NONE . . . 340
8.56.6.8 WLAN_ERROR_PARAM . . . 340
8.56.6.9 WLAN_ERROR_NOMEM . . . 340
8.56.6.10 WLAN_ERROR_STATE . . . 340
8.56.6.11 WLAN_ERROR_ACTION . . . 340
8.56.6.12 WLAN_ERROR_PS_ACTION . . . 340
8.56.6.13 WLAN_ERROR_NOT_SUPPORTED . . . 340
8.56.7 Typedef Documentation . . . 340
8.56.7.1 wlan_beacon_info_t . . . 340
8.56.7.2 wlan_probe_req_info_t . . . 341
8.56.7.3 wlan_auth_info_t . . . 341
8.56.7.4 wlan_data_info_t . . . 341
8.56.7.5 wlan_frame_t . . . 341

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.56.8 Enumeration Type Documentation . . . 341
8.56.8.1 wm_wlan_errno . . . 341
8.56.8.2 wlan_event_reason . . . 341
8.56.8.3 wlan_wakeup_event_t . . . 342
8.56.8.4 wlan_connection_state . . . 342
8.56.8.5 wlan_ps_mode . . . 343
8.56.8.6 wlan_uap_ps_mode . . . 343
8.56.8.7 wlan_wfd_ps_mode . . . 343
8.56.8.8 wlan_security_type . . . 344
8.56.8.9 anonymous enum . . . 344
8.56.8.10 wlan_frame_type_t . . . 344
8.56.9 Variable Documentation . . . 344
8.56.9.1 frame_ctrl_flags . . . 344
8.56.9.2 duration . . . 344
8.56.9.3 dest . . . 344
8.56.9.4 src . . . 345
8.56.9.5 bssid . . . 345
8.56.9.6 seq_frag_num . . . 345
8.56.9.7 timestamp . . . 345
8.56.9.8 beacon_interval . . . 345
8.56.9.9 cap_info . . . 345
8.56.9.10 ssid_element_id . . . 345
8.56.9.11 ssid_len . . . 345
8.56.9.12 ssid . . . 345
8.56.9.13 qos_ctrl . . . 345
8.56.9.14 frame_type . . . 345
8.56.9.15 beacon_info . . . 345
8.56.9.16 probe_req_info . . . 346
8.56.9.17 auth_info . . . 346
8.56.9.18 data_info . . . 346
8.57 wlan_11d.h File Reference . . . 346
8.57.1 Function Documentation . . . 346
8.57.1.1 wlan_set_domain_params . . . 346
8.57.1.2 wlan_set_region_code . . . 347
8.57.1.3 wlan_11d_country_index_2_string . . . 347
8.57.2 Typedef Documentation . . . 347
8.57.2.1 country_code_t . . . 347
8.57.3 Enumeration Type Documentation . . . 347
8.57.3.1 anonymous enum . . . 347
8.58 wm-tls.h File Reference . . . 347

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.58.1 Detailed Description . . . 348
8.58.2 Function Documentation . . . 348
8.58.2.1 ctaocrypt_lib_init . . . 348
8.58.2.2 tls_lib_init . . . 348
8.58.2.3 tls_session_init . . . 348
8.58.2.4 tls_create_server_context . . . 348
8.58.2.5 tls_purge_server_context . . . 349
8.58.2.6 tls_create_client_context . . . 349
8.58.2.7 tls_purge_client_context . . . 349
8.58.2.8 tls_send . . . 350
8.58.2.9 tls_recv . . . 350
8.58.2.10 tls_close . . . 350
8.58.2.11 tls_pending . . . 350
8.58.2.12 tls_get_error . . . 351
8.58.3 Typedef Documentation . . . 351
8.58.3.1 tls_handle_t . . . 351
8.58.4 Enumeration Type Documentation . . . 351
8.58.4.1 tls_flags_t . . . 351
8.59 wm_net.h File Reference . . . 351
8.59.1 Detailed Description . . . 351
8.59.2 Function Documentation . . . 351
8.59.2.1 net_gethostbyname . . . 351
8.59.2.2 netif_add_udp_broadcast_filter . . . 352
8.59.2.3 netif_remove_udp_broadcast_filter . . . 352
8.59.2.4 net_sock_to_interface . . . 352
8.59.2.5 net_get_sta_handle . . . 353
8.59.2.6 net_get_uap_handle . . . 353
8.59.2.7 net_get_wfd_handle . . . 353
8.59.2.8 net_interface_up . . . 353
8.59.2.9 net_interface_down . . . 354
8.59.2.10 net_interface_dhcp_stop . . . 354
8.59.2.11 net_get_if_addr . . . 354
8.59.2.12 net_get_if_ip_addr . . . 354
8.59.2.13 net_get_if_ip_mask . . . 355
8.59.2.14 net_ipv4stack_init . . . 355
8.59.2.15 raw_low_level_output . . . 355
8.60 wm_os.h File Reference . . . 356
8.60.1 Detailed Description . . . 356
8.60.2 Usage . . . 356
8.60.3 Function Documentation . . . 356

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.60.3.1 os_get_usec_counter . . . 356
8.60.3.2 os_get_timestamp . . . 357
8.60.3.3 os_thread_create . . . 357
8.60.3.4 os_thread_delete . . . 357
8.60.3.5 os_thread_sleep . . . 358
8.60.3.6 os_thread_wait_abort . . . 358
8.60.3.7 os_msec_to_ticks . . . 358
8.60.3.8 os_ticks_to_msec . . . 358
8.60.3.9 os_thread_self_complete . . . 359
8.60.3.10 os_queue_create . . . 359
8.60.3.11 os_queue_send . . . 359
8.60.3.12 os_queue_recv . . . 360
8.60.3.13 os_queue_delete . . . 360
8.60.3.14 os_queue_get_msgs_waiting . . . 361
8.60.3.15 os_setup_idle_function . . . 361
8.60.3.16 os_setup_tick_function . . . 361
8.60.3.17 os_remove_idle_function . . . 361
8.60.3.18 os_remove_tick_function . . . 362
8.60.3.19 os_mutex_create . . . 362
8.60.3.20 os_mutex_get . . . 362
8.60.3.21 os_mutex_put . . . 363
8.60.3.22 os_recursive_mutex_create . . . 363
8.60.3.23 os_recursive_mutex_get . . . 364
8.60.3.24 os_recursive_mutex_put . . . 364
8.60.3.25 os_mutex_delete . . . 364
8.60.3.26 os_semaphore_create . . . 365
8.60.3.27 os_semaphore_create_counting . . . 365
8.60.3.28 os_semaphore_get . . . 365
8.60.3.29 os_semaphore_put . . . 366
8.60.3.30 os_semaphore_getcount . . . 366
8.60.3.31 os_semaphore_delete . . . 366
8.60.3.32 os_rwlock_create . . . 367
8.60.3.33 os_rwlock_delete . . . 367
8.60.3.34 os_rwlock_write_lock . . . 367
8.60.3.35 os_rwlock_write_unlock . . . 368
8.60.3.36 os_rwlock_read_lock . . . 368
8.60.3.37 os_rwlock_read_unlock . . . 368
8.60.3.38 os_timer_create . . . 368
8.60.3.39 os_timer_activate . . . 369
8.60.3.40 os_timer_change . . . 369

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.60.3.41 os_timer_is_running . . . 369
8.60.3.42 os_timer_get_context . . . 370
8.60.3.43 os_timer_reset . . . 370
8.60.3.44 os_timer_deactivate . . . 370
8.60.3.45 os_timer_delete . . . 371
8.60.3.46 os_mem_calloc . . . 371
8.60.3.47 os_heap_add_bank . . . 371
8.60.3.48 os_dump_mem_stats . . . 371
8.60.4 Macro Documentation . . . 372
8.60.4.1 os_thread_relinquish . . . 372
8.60.4.2 os_ticks_get . . . 372
8.60.4.3 os_total_ticks_get . . . 372
8.60.4.4 os_ticks_to_unblock . . . 372
8.60.4.5 os_thread_stack_define . . . 372
8.60.4.6 os_queue_pool_define . . . 372
8.60.4.7 OS_WAIT_FOREVER . . . 372
8.60.4.8 OS_NO_WAIT . . . 372
8.60.4.9 OS_MUTEX_INHERIT . . . 372
8.60.4.10 OS_MUTEX_NO_INHERIT . . . 372
8.60.4.11 os_mem_alloc . . . 373
8.60.4.12 os_mem_realloc . . . 373
8.60.4.13 os_mem_free . . . 373
8.60.4.14 os_get_runtime_stats . . . 373
8.60.5 Typedef Documentation . . . 373
8.60.5.1 os_thread_stack_t . . . 373
8.60.5.2 os_queue_pool_t . . . 374
8.60.5.3 cb_fn . . . 374
8.60.5.4 os_timer_reload_t . . . 374
8.60.5.5 os_timer_activate_t . . . 374
8.60.6 Enumeration Type Documentation . . . 374
8.60.6.1 os_timer_reload . . . 374
8.60.6.2 os_timer_activate . . . 374
8.61 wm_supplicant.h File Reference . . . 374
8.61.1 Function Documentation . . . 374
8.61.1.1 supplicant_start . . . 374
8.61.1.2 supplicant_send_cmd . . . 375
8.61.1.3 supplicant_finish . . . 375
8.61.2 Enumeration Type Documentation . . . 375
8.61.2.1 supp_event . . . 375
8.62 wm_utils.h File Reference . . . 375

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.62.1 Detailed Description . . . 375
8.62.2 Function Documentation . . . 376
8.62.2.1 hex2bin . . . 376
8.62.2.2 bin2hex . . . 376
8.62.2.3 random_register_handler . . . 376
8.62.2.4 random_unregister_handler . . . 377
8.62.2.5 random_register_seed_handler . . . 377
8.62.2.6 random_unregister_seed_handler . . . 377
8.62.2.7 random_initialize_seed . . . 377
8.62.2.8 sample_initialise_random_seed . . . 378
8.62.2.9 get_random_sequence . . . 378
8.62.2.10 strdup . . . 378
8.62.2.11 soft_crc32 . . . 378
8.62.3 Macro Documentation . . . 379
8.62.3.1 container_of . . . 379
8.62.4 Typedef Documentation . . . 379
8.62.4.1 random_hdlr_t . . . 379
8.63 wm_wlan.h File Reference . . . 379
8.63.1 Detailed Description . . . 379
8.63.2 Function Documentation . . . 379
8.63.2.1 wm_wlan_init . . . 379
8.63.2.2 wm_wlan_deinit . . . 380
8.63.2.3 wm_core_and_wlan_init . . . 380
8.64 wmcrypto.h File Reference . . . 380
8.64.1 Detailed Description . . . 380
8.64.2 Function Documentation . . . 380
8.64.2.1 mrvl_dh_setup_key . . . 380
8.64.2.2 mrvl_dh_compute_key . . . 380
8.64.2.3 mrvl_dh_free . . . 381
8.64.2.4 mrvl_sha1_vector . . . 381
8.64.2.5 mrvl_sha256_vector . . . 381
8.64.2.6 mrvl_sha256 . . . 381
8.64.2.7 mrvl_hmac_sha256 . . . 382
8.64.2.8 mrvl_kdf . . . 382
8.64.2.9 mrvl_aes_wrap . . . 382
8.64.2.10 mrvl_aes_unwrap . . . 383
8.64.2.11 mrvl_aes_wrap_ext . . . 383
8.64.2.12 mrvl_aes_unwrap_ext . . . 383
8.64.2.13 mrvl_crypto_hmac_md5 . . . 383
8.64.2.14 mrvl_crypto_md5 . . . 384

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.64.2.15 mrvl_crypto_pass_to_key . . . 384
8.64.3 Macro Documentation . . . 384
8.64.3.1 u8 . . . 384
8.64.3.2 u32 . . . 384
8.64.3.3 SHA256_DIGEST_SIZE . . . 384
8.64.3.4 SHA256_BLOCK_SIZE . . . 384
8.65 wmerrno.h File Reference . . . 385
8.65.1 Detailed Description . . . 385
8.65.2 Macro Documentation . . . 385
8.65.2.1 MOD_UNUSED_3 . . . 385
8.65.2.2 MOD_HTTPD . . . 385
8.65.2.3 MOD_AF . . . 385
8.65.2.4 MOD_FTFS . . . 385
8.65.2.5 MOD_RFGET . . . 385
8.65.2.6 MOD_JSON . . . 385
8.65.2.7 MOD_TELNETD . . . 385
8.65.2.8 MOD_SMDNS . . . 385
8.65.2.9 MOD_EXML . . . 385
8.65.2.10 MOD_DHCPD . . . 385
8.65.2.11 MOD_MDNS . . . 386
8.65.2.12 MOD_SYSINFO . . . 386
8.65.2.13 MOD_UNUSED_1 . . . 386
8.65.2.14 MOD_CRYPTO . . . 386
8.65.2.15 MOD_HTTPC . . . 386
8.65.2.16 MOD_PROV . . . 386
8.65.2.17 MOD_SPI . . . 386
8.65.2.18 MOD_PSM . . . 386
8.65.2.19 MOD_TTCP . . . 386
8.65.2.20 MOD_DIAG . . . 386
8.65.2.21 MOD_UNUSED_2 . . . 386
8.65.2.22 MOD_WPS . . . 386
8.65.2.23 MOD_WLAN . . . 387
8.65.2.24 MOD_USB . . . 387
8.65.2.25 MOD_WIFI . . . 387
8.65.2.26 MOD_CRIT_ERR . . . 387
8.65.2.27 MOD_ERR_LAST . . . 387
8.66 wmstats.h File Reference . . . 387
8.66.1 Enumeration Type Documentation . . . 387
8.66.1.1 wm_reboot_reason_t . . . 387
8.67 wmstdio.h File Reference . . . 387

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.67.1 Function Documentation . . . 387
8.67.1.1 wmprintf . . . 387
8.67.1.2 wmstdio_putchar . . . 388
8.67.1.3 ll_printf . . . 388
8.67.1.4 wmstdio_flush . . . 388
8.67.1.5 wmstdio_getchar . . . 388
8.67.1.6 wmstdio_getconsole_port . . . 388
8.67.1.7 wmstdio_init . . . 389
8.67.2 Macro Documentation . . . 389
8.67.2.1 MAX_MSG_LEN . . . 389
8.68 wmsysinfo.h File Reference . . . 389
8.68.1 Detailed Description . . . 389
8.68.2 Function Documentation . . . 390
8.68.2.1 sysinfo_init . . . 390
8.69 wmtime.h File Reference . . . 390
8.69.1 Detailed Description . . . 390
8.69.2 Function Documentation . . . 390
8.69.2.1 http_date_to_time . . . 390
8.69.2.2 wmtime_time_set . . . 390
8.69.2.3 wmtime_time_get . . . 390
8.69.2.4 wmtime_time_set_posix . . . 391
8.69.2.5 wmtime_time_get_posix . . . 391
8.69.2.6 gmtime_r . . . 391
8.69.2.7 mktime . . . 391
8.69.2.8 asctime . . . 392
8.69.2.9 wmtime_init . . . 392
8.69.2.10 wmtime_cli_init . . . 392
8.70 wps.h File Reference . . . 392
8.70.1 Detailed Description . . . 392
8.70.2 Usage . . . 393
8.70.3 Function Documentation . . . 393
8.70.3.1 wps_start . . . 393
8.70.3.2 wps_connect . . . 393
8.70.3.3 wps_generate_pin . . . 393
8.70.3.4 wps_validate_pin . . . 394
8.70.3.5 wps_stop . . . 394
8.70.4 Macro Documentation . . . 394
8.70.4.1 WPS_LOG . . . 394
8.70.5 Enumeration Type Documentation . . . 394
8.70.5.1 wps_session_command . . . 394

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.70.5.2 wps_event 394
8.70.5.3 wps_session_types 395

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Confidential

Chapter 1

Main Page

1.1 Introduction

Marvell's Wireless Microcontroller solution enables customers to quickly develop applications of interest to add connectivity to different sensors and appliances. The WMSDK supports rich set of features required for connectivity. The WMSDK also relieves developers by providing re-usable libraries and sample applications.

1.1.1 Developer Documentation

This manual provides developer reference documentation for building custom applications with the Wireless Microcontroller SDK.

Please consult the SDK User Guide for an overview of the SDK and user-centric documentation.

The Software Development Kit may be used with compliant Wireless Microcontroller platforms. Please consult the documentation for your specific platform for any platform details.

In addition to the reference documentation in this manual, you can also explore the source code for the various software components (distributed as part of the SDK) for more details.

1.1.1.1 Application Framework

The SDK is equipped with an application framework that may be used to develop applications. The application framework provides implementation for most of the common activities required by a network application. This relieves the developer from the management of these common activities and facilitates quicker application development.

The application framework utilizes the same set of SDK modules provided by the SDK. Developers may choose not to use the application framework and implement similar functionality using the rest of the SDK. The documentation for the application framework is available here: Application Framework documentation .

After going through the Application Framework documentation it would be worthwhile to look at the implementation of the wlan based sample applications in the sample_apps/ folder. This will demonstrate how the application framework can effectively be used by applications.

Once you have a good understanding of this piece, you may start digging deeper within individual SDK modules.

1.1.1.2 CMSIS Library

The SDK has support for CMSIS DSP Library V1.4.5. Refer the link for API details: `http://www.keil.com/cmsis/dsp`

MARVELL
Copyright © 2015 Marvell
Proprietary Information.
August 21, 2015 17:48

Note

1. Support is NOT THREAD-SAFE. Only single thread can access the library at a time. Use standard mutual exclusion mechanism if multiple threads access the API's.
2. For 88MW30X, applications must call fpu_init() before using dsp library. Otherwise hard fault will be generated. Please refer to main.c of any dsp_demo application for reference.
3. To link against CMSIS DSP library, applications must include /dsprules.mk in their Makefiles. Please refer to Makefile of any dsp_demo application.

1.1.1.3 SDK Modules

The following modules are available in the SDK:

- WLAN Connection Manager
- Persistent Storage Manager
- Time Management
- Provisioning Support
- WPS 2.0 Support
- Wi-Fi Direct (P2P) Support
- HTTP Web Server API
- HTTP Client
- TLS API
- ezXML: Easy XML Parsing C Library
- mDNS/DNS-SD responder-querier
- Remote Firmware Upgrade
- Health Monitor
- Power Manager
- Crypto Support
- Partition Management
- Overlay Manager
- JSON Generation and Parser

1.1.1.4 SDK Drivers

The following drivers are available in the SDK:

- mdev-based ACOMP Driver
- mdev-based ADC Driver
- mdev-based AES Driver
- mdev-based CRC Driver
- mdev-based DAC Driver
- mdev-based GPIO Driver

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

- mdev-based GPT Driver
- mdev-based I2C Driver
- mdev-based Poiwer Management
- mdev-based FPU driver
- mdev-based PINMUX Driver
- mdev-based RTC Driver
- mdev-based UART Driver
- mdev-based SSP Driver
- mdev-based WATCHDOG Driver
- mdev-based SDIO Driver
- mdev-based USB Host Driver
- mdev-based Flash Driver

1.1.1.5 SDK Porting

- Board Specific APIs

1.1.1.6 Third-Party Cloud Providers

The Files section provides documentation for all the APIs that are available in this SDK.

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Chapter 2

Deprecated List

Global update_firmware (data_fetch data_fetch_cb, void ∗priv, size_t firmware_length, struct partition_entry ∗passive)

Update firmware image

Global update_wifi_firmware (data_fetch data_fetch_cb, void ∗priv, size_t firmware_length, struct partition_entry ∗passive)

Update Wi-Fi firmware image

Confidential

Proprietary Information.
August 21, 2015 17:48

Copyright © 2015 Marvell

Confidential

Proprietary Information.
August 21, 2015 17:48

Copyright © 2015 Marvell

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Wi-Fi Region Configuration 15
Custom Wi-Fi Region Configuration 17
Low level RFGET APIs 20
Middle level RFGET APIs 23
High level RFGET APIs 27
Other RFGET APIs 29

Confidential

MARVELL
Copyright © 2015 Marvell
Proprietary Information.
August 21, 2015 17:48

Confidential

Proprietary Information.
August 21, 2015 17:48

Copyright © 2015 Marvell

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

_wifi_cal_data_t . . . 31
_wifi_mgmt_frame_tx . . . 31
app_init_state . . . 33
app_p2p_role_data . . . 33
cli_command . . . 34
DH_PG_PARAMS . . . 35
dma_config_type . . . 35
flash_desc . . . 36
healthmon_handler . . . 37
http_req_t . . . 38
http_resp_t . . . 39
httpc_cfg_t . . . 40
httpd_request_t . . . 41
httpd_ssi_call . . . 43
httpd_tls_certs_t . . . 43
httpd_useragent_t . . . 44
httpd_wsgi_call . . . 45
input_gpio_cfg_t . . . 46
ipv4_config . . . 47
jobj_t . . . 48
jsontok_t . . . 48
mdns_service . . . 49
os_queue_pool . . . 51
os_thread_stack . . . 52
output_gpio_cfg_t . . . 52
overlay . . . 53
overlay_ranges . . . 54
p2p_config . . . 54
p2p_scan_result . . . 57
partition_entry . . . 58
partition_table . . . 59
prov_scan_config . . . 60
provisioning_config . . . 60
psm_cfg_t . . . 61
psm_objattrib_t . . . 61
tls_client_t . . . 62
tls_init_config_t . . . 63
tm . . . 65

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

update_desc 66
upgrade_struct_t 67
wifi_domain_param_t 69
wifi_sub_band_set_t 70
wlan_ip_config 70
wlan_network 71
wlan_network_security 73
wlan_scan_result 74
wps_config 76
wslay_frame 78

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Chapter 5

File Index

5.1 File List

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

app_framework.h
The Application Framework 81
block_alloc.h
Slab/Block Allocator 103
board.h
Board Specific APIs 106
cli.h
CLI module 112
dhcp-server.h
DHCP server 114
ezxml.h
EzXML - Easy XML Parsing C Library 116
flash.h
Flash Driver 122
ftfs.h
File Table Filesystem (FTFS) 125
fw_upgrade_ed_chacha.h
Firmware Upgrades using ED25519 and Chacha20 128
fw_upgrade_rsa_aes.h
Firmware Upgrades using RSA and AES 129
generic_io.h
Input/Output GPIO related data structures 131
healthmon.h
Health Monitor 131
http_parse.h
Common HTTP functions 134
httpc.h
HTTP Client Module 135
httpd.h
HTTP Web Server 145
json_generator.h
JSON generator module 165
jsonv2.h
JSON parser module 172
keystore.h
Keystore access module (This module is available only for MW30X devices) 183
led_indicator.h
LED indicator helper functions 186

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

mdev.h
Marvell Device Driver API . . . 187
mdev_acomp.h
Analog Comparator Driver . . . 188
mdev_adc.h
Analog to Digital Converter Driver . . . 190
mdev_aes.h
AES Driver . . . 193
mdev_crc.h
CRC Driver . . . 198
mdev_dac.h
Digital Analog Converter Driver . . . 202
mdev_dma.h
DMA Module . . . 204
mdev_fpu.h
Cortex M4 Floating Point Unit . . . 208
mdev_gpio.h
General Purpose Input Output (GPIO) Driver . . . 208
mdev_gpt.h
General Purpose Timer (GPT) Driver . . . 213
mdev_i2c.h
Inter-Integrated Circuit (I2C) Driver (TWSI) . . . 216
mdev_pinmux.h
GPIO Pin Multiplexer Driver . . . 224
mdev_pm.h
MCU power management module . . . 226
mdev_rtc.h
RTC driver . . . 232
mdev_sdio.h
SDIO Driver . . . 236
mdev_ssp.h
Synchronous Serial Protocol (SSP) Driver . . . 239
mdev_uart.h
Universal Asynchronous Receiver Transmitter (UART) driver . . . 244
mdev_usb.h
Universal Serial Bus (USB) Driver . . . 250
mdev_usb_host.h
Universal Serial Bus (USB) Host driver . . . 253
mdev_wdt.h
WatchDog Timer (WDT) driver . . . 258
mdns.h
The mDNS/DNS-SD Responder-Querier . . . 260
nw_utils.h
Networking CLIs . . . 269
overlays.h
Overlay Manager . . . 269
p2p.h
Wi-Fi Direct/P2P . . . 270
partition.h
Partition Management . . . 273
provisioning.h
Provisioning Support . . . 276
psm-v2.h
Persistent storage module version 2 . . . 284
psm.h
Persistent storage module version 1 (Deprecated) . . . 295
push_button.h
Push button helper functions . . . 295

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

pwrmgr.h
Power Manager Framework . . . 297
rfget.h
Remote Firmware Upgrade . . . 301
secure_upgrade.h
Secure firmware upgrades . . . 302
telnetd.h
The Telnet Server . . . 306
websockets.h
Websocket Implementation in WMSDK using the WSLAY websocket library . . . 307
wifi-decl.h
Wifi structure declarations . . . 309
wifi_events.h
Wi-Fi events . . . 311
wlan.h
WLAN Connection Manager . . . 312
wlan_11d.h
WLAN module 11d API . . . 346
wm-tls.h
TLS API's . . . 347
wm_net.h
Network Abstraction Layer . . . 351
wm_os.h
WMSDK OS Abstraction Layer . . . 356
wm_supplicant.h
WMSDK Host Supplicant . . . 374
wm_utils.h
WMSDK Utility functions . . . 375
wm_wlan.h
Architecture specific system include file . . . 379
wmcrypto.h
Crypto Functions . . . 380
wmerrno.h
Error Management . . . 385
wmstats.h
Wireless Microcontroller statistics . . . 387
wmstdio.h
Standard Input Output . . . 387
wmsysinfo.h
System Info Utilities Here is the sysinfo usage: sysinfo [options] service name [command] where service name can be one of the following: [thread|socket|netstat|heap|stats|mdns|-h] . . . 389
wmtime.h
Time Management Subsystem . . . 390
wps.h
WPS - WiFi Protected Setup . . . 392

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Confidential

Proprietary Information.
August 21, 2015 17:48

Copyright © 2015 Marvell

Chapter 6

Module Documentation

6.1 Wi-Fi Region Configuration

Functions

- static int wlan_enable_11d ()
- static int wlan_get_country ()
- static int wlan_set_country (country_code_t country)

6.1.1 Detailed Description

By default, the SDK builds applications that are compliant with the US region configuration. This implies that the module obeys the US regulations for Wi-Fi transmissions on certified frequency bands. The SDK provides mechanism for configuring various region codes in the applications. This can be performed in one of the following two ways:

I) Specifying Country Code

In this method of configuration, the application defines up-front what is the country code that the MC200/MW300 is going to be deployed in. Once configured the Wi-Fi firmware obeys the configured countries regulations. This configuration can be set by making a call to the wlan_set_country() API. This API should be called after WLAN is initialized but before starting uAP or making any connection attempts on station interface.

For example: wlan_set_country(COUNTRY_CN);

II) Using 802.11D

Note

The FCC does not allow the use of 802.11D in the US starting Jan 1, 2015. In this method of configuration, the Wi-Fi driver of the SDK will scan for Access Points in the vicinity and accordingly configure itself to operate in the available frequency bands. This configuration can be set by making a call to the wlan_enable_11d() API. This API should be called after WLAN is initialized but before starting uAP or making any connection attempts on station interface.

For example: wlan_enable_11d();

6.1.2 Function Documentation

6.1.2.1 static int wlan_enable_11d ( ) [inline], [static]

Enable 11D support in WLAN Driver.

MARVELL
Copyright © 2015 Marvell
Proprietary Information.
August 21, 2015 17:48

Note

This API should be called after WLAN is initialized but before starting uAP or making any connection attempts on station interface.
Either this function or wlan_set_country() should be used at a time. If both functions are called in the application, then WLAN Driver properties will be set as per the wlan_set_country() function.

Returns

-WM_FAIL if operation was failed.
WM_SUCCESS if operation was successful.

6.1.2.2 static int wlan_get_country ( ) [inline],[static]

Get country code from WLAN Driver.

Note

This API should be called after WLAN is initialized but before starting uAP or making any connection attempts on station interface.

Returns

Country code. Refer to country_code_t.

6.1.2.3 static int wlan_set_country (country_code_t *country*) [inline],[static]

Set country code in WLAN Driver.

Note

This API should be called after WLAN is initialized but before starting uAP or making any connection attempts on station interface.
Either this function or wlan_enable_11d() should be used at a time. If both functions are called in the application, then WLAN Driver properties will be set as per the wlan_set_country() function.

Parameters

in	*country*	Country code. Refer to country_code_t.

Returns

-WM_FAIL if operation was failed.
WM_SUCCESS if operation was successful.

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

6.2 Custom Wi-Fi Region Configuration

Ideally applications should use either wlan_enable_11d() or wlan_set_country() APIs to have standard 802.11d functionality as per regulations of Wi-Fi transmissions on certified frequency bands.

But If application wants to configure custom 802.11d configurations then wlan_uap_set_domain_params() and wlan-_set_domain_params APIs can be used for that.

If applications just want to set a particular region then wlan_set_region_code() API can be used for the purpose.

Supported region code values are given in mlan_11d.c file.

Sets the domain parameters for the uAP.

```
@note This API should be called after WLAN is initialized
but before starting uAP
```

To use this API you will need to fill up the structure wifi_domain_param_t with correct parameters.

The below section lists all the arrays that can be passed individually or in combination to the API wlan_uap_set_-domain_params(). These are the sub band sets to be part of the Country Info IE in the uAP beacon. One of them is to be selected according to your region. Please have a look at the example given in the documentation below for reference.

Supported Country Codes: "US" : USA, "CA" : Canada, "SG" : Singapore, "EU" : Europe, "AU" : Australia, "KR" : Republic of Korea, "CN" : China, "FR" : France, "JP" : Japan

```
Region: US(US) or Singapore(SG) 5 GHz
wifi_sub_band_set_t subband_US_SG_5GHz[] = {
        {36, 1, 20},
        {40, 1, 20},
        {44, 1, 20},
        {48, 1, 20},
        {52, 1, 20},
        {56, 1, 20},
        {60, 1, 20},
        {64, 1, 20},
        {100, 1, 20},
        {104, 1, 20},
        {108, 1, 20},
        {112, 1, 20},
        {116, 1, 20},
        {120, 1, 20},
        {124, 1, 20},
        {128, 1, 20},
        {132, 1, 20},
        {136, 1, 20},
        {140, 1, 20},
        {149, 1, 20},
        {153, 1, 20},
        {157, 1, 20},
        {161, 1, 20},
        {165, 1, 20}
};

Region: Canada(CA) 5 GHz
wifi_sub_band_set_t subband_CA_5GHz[] = {
        {36, 1, 20},
        {40, 1, 20},
        {44, 1, 20},
        {48, 1, 20},
        {52, 1, 20},
        {56, 1, 20},
        {60, 1, 20},
        {64, 1, 20},
        {100, 1, 20},
        {104, 1, 20},
        {108, 1, 20},
        {112, 1, 20},
        {116, 1, 20},
        {132, 1, 20},
        {136, 1, 20},
        {140, 1, 20},
        {149, 1, 20},
        {153, 1, 20},
        {157, 1, 20},
        {161, 1, 20},
        {165, 1, 20}
};
```

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

```
Region: Europe/ETSI(EU), Australia(AU), Republic of Korea(KR) 5 GHz
wifi_sub_band_set_t subband_EU_AU_KR_5GHz[] = {
        {36, 1, 20},
        {40, 1, 20},
        {44, 1, 20},
        {48, 1, 20},
        {52, 1, 20},
        {56, 1, 20},
        {60, 1, 20},
        {64, 1, 20},
        {100, 1, 20},
        {104, 1, 20},
        {108, 1, 20},
        {112, 1, 20},
        {116, 1, 20},
        {120, 1, 20},
        {124, 1, 20},
        {128, 1, 20},
        {132, 1, 20},
        {136, 1, 20},
        {140, 1, 20}
};

Region: China(CN) 5 GHz
wifi_sub_band_set_t subband_CN_5GHz[] = {
        {149, 1, 33},
        {153, 1, 33},
        {157, 1, 33},
        {161, 1, 33},
        {165, 1, 33},
};

Region: France(FR) 5 GHz
wifi_sub_band_set_t subband_FR_5GHz[] = {
        {36, 1, 20},
        {40, 1, 20},
        {44, 1, 20},
        {48, 1, 20},
        {52, 1, 20},
        {56, 1, 20},
        {60, 1, 20},
        {64, 1, 20},
        {100, 1, 20},
        {104, 1, 20},
        {108, 1, 20},
        {112, 1, 20},
        {116, 1, 20},
        {120, 1, 20},
        {124, 1, 20},
        {128, 1, 20},
        {132, 1, 20},
        {136, 1, 20},
        {140, 1, 20},
        {149, 1, 20},
        {153, 1, 20},
        {157, 1, 20},
        {161, 1, 20},
        {165, 1, 20}
};

Region: Japan(JP) 5 GHz
wifi_sub_band_set_t subband_JP_5_GHz[] = {
        {8, 1, 23},
        {12, 1, 23},
        {16, 1, 23},
        {36, 1, 23},
        {40, 1, 23},
        {44, 1, 23},
        {48, 1, 23},
        {52, 1, 23},
        {56, 1, 23},
        {60, 1, 23},
        {64, 1, 23},
        {100, 1, 23},
        {104, 1, 23},
        {108, 1, 23},
        {112, 1, 23},
        {116, 1, 23},
        {120, 1, 23},
        {124, 1, 23},
        {128, 1, 23},
        {132, 1, 23},
        {136, 1, 23},
        {140, 1, 23}
};
```

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

```
// We will be using the KR 2.4 and 5 GHz bands for this example

int nr_sb = (sizeof(subband_EU_AU_KR_CN_2_4GHz)
             + sizeof(subband_EU_AU_KR_5GHz))
             / sizeof(wifi_sub_band_set_t);

// We already have space for first sub band info entry in
// wifi_domain_param_t
wifi_domain_param_t *dp =
        os_mem_alloc(sizeof(wifi_domain_param_t) +
                     (sizeof(wifi_sub_band_set_t) * (nr_sb - 1)));

// COUNTRY_CODE_LEN is 3. Add extra ' ' as country code is 2 characters
memcpy(dp->country_code, "KR ", COUNTRY_CODE_LEN);

dp->no_of_sub_band = nr_sb;
memcpy(&dp->sub_band[0], &subband_EU_AU_KR_CN_2_4GHz[0],
       1 * sizeof(wifi_sub_band_set_t));
memcpy(&dp->sub_band[1], &subband_EU_AU_KR_5GHz,
       (nr_sb - 1) * sizeof(wifi_sub_band_set_t));

wlan_uap_set_domain_params(dp);
os_mem_free(dp);
```

Parameters

in	*dp*	The wifi domain parameters

Returns

-WM_E_INVAL if invalid parameters were passed.
-WM_E_NOMEM if memory allocation failed.
WM_SUCCESS if operation was successful.

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

6.3 Low level RFGET APIs

Data Structures

- struct update_desc

Typedefs

- typedef struct update_desc update_desc_t

Functions

- int rfget_init ()
- int rfget_update_begin (update_desc_t ∗ud, struct partition_entry ∗p)
- int rfget_update_data (update_desc_t ∗ud, const char ∗data, size_t datalen)
- int rfget_update_complete (update_desc_t ∗ud)
- int rfget_update_abort (update_desc_t ∗ud)
- bool rfget_is_update_in_progress (void)
- int verify_load_firmware (uint32_t start, int size)
- struct partition_entry ∗ rfget_get_passive_firmware (void)
- struct partition_entry ∗ rfget_get_passive_wifi_firmware (void)

6.3.1 Detailed Description

These APIs can be used for granular control on the upgrades or to upgrade any application specific partition. A simple flow will be in this order:

```
rfget_init();
struct partition_entry *p = get_passive_partition(); //Write own function
update_desc_t ud;
rfget_update_begin(&ud, p);
rfget_update_data(&ud, data, data_len); //Loop to write all data
rfget_update_complete(&ud);
```

Applications should use appropriate mechanism to verify that the written image is valid. Various convenience APIs are also provided by this module to help in upgrading the MCU Firmware or Wi-Fi firmware.

6.3.2 Typedef Documentation

6.3.2.1 typedef struct update_desc update_desc_t

Update Description Structure

This is referenced when using the rfget_update_ APIs. The contents of this structure are populated and used by the rfget_update_ APIs.

6.3.3 Function Documentation

6.3.3.1 int rfget_init ( )

Initialize the RFGET module

This API should be called before using any other low level upgrade APIs

Returns

WM_SUCCESS on success, error otherwise

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

6.3.3.2 int rfget_update_begin (update_desc_t * *ud*, struct partition_entry * *p*)

Begin an update

This function erases the associated flash sector for the partition entry that is passed to it. The ud is used to maintain information across the various calls to the rfget_update_ functions.

Parameters

out	*ud*	the update descriptor structure
in	*p*	the partition entry of the passive partition

Returns

WM_SUCCESS on success, error otherwise

6.3.3.3 int rfget_update_data (update_desc_t * *ud*, const char * *data*, size_t *datalen*)

Write data to partition

This should be called after a call to the rfget_update_begin(). Successive calls to this function write data to the flash.

Parameters

in	*ud*	the update descriptor
in	*data*	the data to be written
in	*datalen*	the length of the data to be written

Returns

WM_SUCCESS on success, error otherwise

6.3.3.4 int rfget_update_complete (update_desc_t * *ud*)

Finish update

Carrying out subsequent reboot is not handled by this API and is left to the application.

Mark this partition as the active partition and finish the update process.

Parameters

in	*ud*	the update descriptor

Returns

WM_SUCCESS on success, error otherwise

6.3.3.5 int rfget_update_abort (update_desc_t * *ud*)

Abort update in case of some error

This function does not mark this partition as the active partition but only cleans up the update process.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*ud*	the update descriptor

Returns

WM_SUCCESS on success, error otherwise

6.3.3.6 bool rfget_is_update_in_progress (void)

This function tells whether system image update is in progress.

Returns

If update is in progress returns TRUE else returns FALSE.

6.3.3.7 int verify_load_firmware (uint32_t *start*, int *size*)

Verify if the MCU Firmware written to flash is valid

Parameters

in	*start*	The start address from which the firmware data starts
in	*size*	of the firmware image on flash

Returns

WM_SUCCESS on success
-WM_E_RFGET_FFW_LOAD if an error is encountered

6.3.3.8 struct partition_entry* rfget_get_passive_firmware (void)

Convenience function to get the passive firmware partition

Returns

Pointer of type partition_entry for the passive firmware partition.

6.3.3.9 struct partition_entry* rfget_get_passive_wifi_firmware (void)

Convenience function to get the passive wifi firmware partition

Returns

Pointer of type partition_entry for the passive wifi firmware partition.

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

6.4 Middle level RFGET APIs

Typedefs

- typedef size_t(∗ data_fetch)(void ∗priv, void ∗buf, size_t len)
- typedef int(∗ data_validate)(void ∗priv)

Functions

- int update_and_validate_firmware (data_fetch data_fetch_cb, void ∗priv, size_t firmware_length, struct partition_entry ∗passive, data_validate data_validate_cb)
- int update_and_validate_wifi_firmware (data_fetch data_fetch_cb, void ∗priv, size_t firmware_length, struct partition_entry ∗passive, data_validate data_validate_cb)
- int update_and_validate_fs (data_fetch data_fetch_cb, void ∗priv, size_t fs_length, struct partition_entry ∗passive, data_validate data_validate_cb)
- int update_firmware (data_fetch data_fetch_cb, void ∗priv, size_t firmware_length, struct partition_entry ∗passive)
- int update_wifi_firmware (data_fetch data_fetch_cb, void ∗priv, size_t firmware_length, struct partition_entry ∗passive)

6.4.1 Detailed Description

The middle level APIs provide APIs for MCU firmware, Wi-Fi firmware and File system upgrades by abstracting out the low level details. These APIs do not put any restriction on the method/protocol used to fetch the data.

The convenience APIs rfget_get_passive_firmware() and rfget_get_passive_wifi_firmware() can be used to get the passive partitions.

Note

Carrying out subsequent reboot is not handled by any of the APIs and is left to the application.

6.4.2 Typedef Documentation

6.4.2.1 typedef size_t(∗ data_fetch)(void ∗priv, void ∗buf, size_t len)

Data Fetch callback function

Callback function pointer passed to the update_and_validate_firmware(), update_and_validate_wifi_firmware() or update_and_validate_fs() functions. This is needed because the update file may be too big to fit in a buffer at a time. The rfget module will call this API repeatedly to fetch the data. Applications should write appropriate function for this.

Parameters

in,out	*priv*	A private data pointer that will be used/modified by this function to maintain a context. This will be initialized with the pointer passed to update_and_validate_firmware(), update_and_validate_wifi_firmware() or update_and_validate_fs()
out	*buf*	Pointer to a buffer that should be used by this function to read the data
in	*len*	Length of the data requested

Returns

Length of the data actually read
-WM_FAIL if any error is encountered

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

6.4.2.2 typedef int(∗ data_validate)(void ∗priv)

Data validate callback function

Optional callback function pointer passed to the update_and_validate_firmware(), update_and_validate_wifi_firmware() or update_and_validate_fs() function. This may be required as a security measure to validate the received data. Applications should write appropriate function for this.

Parameters

in	*priv*	The private data pointer used/modified by data_fetch callback

Returns

WM_SUCCESS if validation succeeds
-WM_FAIL if validation fails.

6.4.3 Function Documentation

6.4.3.1 int update_and_validate_firmware (data_fetch *data_fetch_cb*, void ∗ *priv*, size_t *firmware_length*, struct partition_entry ∗ *passive*, data_validate *data_validate_cb*)

Update and validate firmware image

This API can be used to update and validate the MCU firmware without worrying about any lower level details.

Parameters

in	*data_fetch_cb*	A callback of type data_fetch that will internally be used by this API to fetch the firmware data.
in	*priv*	Pointer to private data that will be passed to the data_fetch and data_validate functions.
in	*firmware_length*	Length of the firmware image that needs to be written to flash.
in	*passive*	Pointer of type partition_entry to the passive partition to be updated
in	*data_validate_cb*	An optional pointer to a function of type data_validate that will be called at the end, to validate the written data. This can be kept NULL if validation is not required.

Returns

WM_SUCCESS on success
-WM_E_RFGET_INVLEN if the passed firmware_length is invalid (either 0 or more than the partition size)
-WM_E_RFGET_INV_IMG if the received image is not valid firmware image
-WM_E_RFGET_FWR if the passive partition write/erase failed
-WM_E_RFGET_FFW_LOAD if firmware verification failed
-WM_FAIL if any other error encountered
any other error returned by the data_validate function

6.4.3.2 int update_and_validate_wifi_firmware (data_fetch *data_fetch_cb*, void ∗ *priv*, size_t *firmware_length*, struct partition_entry ∗ *passive*, data_validate *data_validate_cb*)

Update and validate Wi-Fi firmware image

This API can be used to update and validate the Wi-Fi firmware without worrying about any lower level details.

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*data_fetch_cb*	A callback of type data_fetch that will internally be used by this API to fetch the wifi firmware data.
in	*priv*	Pointer to private data that will be passed to the data_fetch and data_validate functions.
in	*firmware_length*	Length of the wifi firmware image that needs to be written to flash.
in	*passive*	Pointer of type partition_entry to the passive wifi partition to be updated
in	*data_validate_cb*	An optional pointer to a function of type data_validate that will be called at the end, to validate the written data. This can be kept NULL if validation is not required.

Returns

WM_SUCCESS on success
-WM_E_RFGET_INVLEN if the passed firmware_length is invalid (either 0 or more than the partition size)
-WM_E_RFGET_FWR if the passive partition write/erase failed
-WM_FAIL if any other error encountered
any other error returned by the data_validate function

6.4.3.3 int update_and_validate_fs (data_fetch *data_fetch_cb*, void * *priv*, size_t *fs_length*, struct partition_entry * *passive*, data_validate *data_validate_cb*)

Update and validate filesystem image

This API can be used to update and validate the Filesystem without worrying about any lower level details.

Parameters

in	*data_fetch_cb*	A callback of type data_fetch that will internally be used by this API to fetch the filesystem data.
in	*priv*	Pointer to private data that will be passed to the data_fetch and data_validate functions.
in	*fs_length*	Length of the filesystem image that needs to be written to flash.
in	*passive*	Pointer of type partition_entry to the passive partition to be updated
in	*data_validate_cb*	An optional pointer to a function of type data_validate that will be called at the end, to validate the written data. This can be kept NULL if validation is not required.

Returns

WM_SUCCESS on success
-WM_E_RFGET_INVLEN if the passed fs_length is invalid (either 0 or more than the partition size)
-WM_E_RFGET_INV_IMG if the received image is not valid filesystem image
-WM_E_RFGET_FWR if the passive partition write/erase failed
-WM_FAIL if any other error encountered
any other error returned by the data_validate function

6.4.3.4 int update_firmware (data_fetch *data_fetch_cb*, void * *priv*, size_t *firmware_length*, struct partition_entry * *passive*)

Deprecated Update firmware image

Please use update_and_validate_firmware()

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*data_fetch_cb*	A callback of type data_fetch that will internally be used by this API to fetch the firmware data.
in	*priv*	Pointer to private data that will be passed to the data_fetch and data_validate functions.
in	*firmware_length*	Length of the firmware image that needs to be written to flash.
in	*passive*	Pointer of type partition_entry to the passive partition to be updated

Returns

Please refer update_and_validate_firmware()

6.4.3.5 int update_wifi_firmware (data_fetch *data_fetch_cb*, void * *priv*, size_t *firmware_length*, struct partition_entry * *passive*)

Deprecated Update Wi-Fi firmware image

Please use update_and_validate_wifi_firmware()

Parameters

in	*data_fetch_cb*	A callback of type data_fetch that will internally be used by this API to fetch the wifi firmware data.
in	*priv*	Pointer to private data that will be passed to the data_fetch and data_validate functions.
in	*firmware_length*	Length of the wifi firmware image that needs to be written to flash.
in	*passive*	Pointer of type partition_entry to the passive wifi partition to be updated

Returns

Please refer update_and_validate_wifi_firmware()

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

6.5 High level RFGET APIs

Functions

- int rfget_server_mode_update (int sockfd, size_t filelength, struct partition_entry ∗passive)
- int rfget_client_mode_update (const char ∗url_str, struct partition_entry ∗passive, const httpc_cfg_t ∗cfg)

6.5.1 Detailed Description

There are two different High level APIs provided by this module:

- Server mode upgrade: In this method rfget module receives the image data through an already opened socket connection with a peer. This mode is agnostic to image delivery protocol. This is covered by rfget_server_-mode_update API.
- Client mode upgrade: In this method wireless microcontroller device acts as a HTTP client to receive the image from other web server using specified URL. rfget_client_mode_update API covers this functionality.

In both the cases the image is written to flash using following process:

1. Receive firmware image header and verify signature
2. Identify passive partition using partition table and erase it. Convenience APIs rfget_get_passive_firmware() and rfget_get_passive_wifi_firmware() have been provided to get the passive MCU Firmware and Wi-Fi firmware partitions.
3. Receive image data and write it as received in parts
4. Update partition table

As of now these functions support the MCU firmware, Wi-Fi firmware and filesystem upgrades. If an application specific component needs to be upgraded, the lower level APIs given in Low level RFGET APIs should be used.

Note

Carrying out subsequent reboot is not handled by any of the APIs and is left to the application.

6.5.2 Function Documentation

6.5.2.1 int rfget_server_mode_update (int *sockfd,* size_t *filelength,* struct partition_entry ∗ *passive*)

Update a system image in server mode using http server.

This function updates a system image. i.e. a firmware or filesystem image. This function reads the file from the HTTP server side socket 'sockfd'

Parameters

in	*sockfd*	Server side socket file descriptor. This function will read 'filelength' bytes from this socket and write it to appropriate location depending on the partition
in	*filelength*	Number of bytes to be read from socket and written to flash.
in	*passive*	the passive partition that should be upgraded.

Returns

On success returns WM_SUCCESS. On error returns appropriate error code.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Note

This API should be called only after stripping off HTTP header from received packets. The sockfd should contain the data starting with image header.

6.5.2.2 int rfget_client_mode_update (const char * *url_str*, struct partition_entry * *passive*, const httpc_cfg_t * *cfg*)

Update a system image in client mode using http client.

This function updates a system image. for e.g. a firmware image. This function reads the resource given as a part of the URL.

Parameters

in	*url_str*	URL to the image.
in	*passive*	the passive partition that should be upgraded.
in	*cfg*	is a pointer of the type httpc_cfg_t. Please refer to file httpc.h for more information.

Returns

On success returns WM_SUCCESS. On error returns appropriate error code.

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

6.6 Other RFGET APIs

Functions

- int rfget_cli_init (void)
- struct fs ∗ rfget_ftfs_init (struct ftfs_super ∗sb, int be_ver, const char ∗name, struct partition_entry ∗∗passive)

6.6.1 Detailed Description

This section provides any additional convenience APIs related to RFGET.

6.6.2 Function Documentation

6.6.2.1 int rfget_cli_init (void)

This function registers the CLI for the rfget application.

Returns

On success returns WM_SUCCESS. On error returns appropriate error code.

6.6.2.2 struct fs∗ rfget_ftfs_init (struct ftfs_super ∗ *sb,* int *be_ver,* const char ∗ *name,* struct partition_entry ∗∗ *passive*)

Initialize Upgradeable FTFS

If you wish to use upgradeable FTFS you should use this function. If you have FTFS partitions that do not require upgradability, please use ftfs_init() instead.

Given the backend API version this function will mount the appropriate FTFS from the active-passive pair. The FTFS with the highest generation number that matches the be_ver is mounted.

Parameters

out	*sb*	pointer to a ftfs_super structure
in	*be_ver*	backend API version
in	*name*	name of the FTFS component in the layout
out	*passive*	a pointer to the passive component

Returns

a pointer to struct fs. The pointer points to an entry within the 'sb' parameter that is passed as an argument. NULL in case of an error

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Confidential

Proprietary Information.
August 21, 2015 17:48

Copyright © 2015 Marvell

Chapter 7

Data Structure Documentation

7.1 _wifi_cal_data_t Struct Reference

```
#include <wifi-decl.h>
```

Data Fields

- t_u16 data_len
- t_u8 * data

7.1.1 Detailed Description

Calibration Data

7.1.2 Field Documentation

7.1.2.1 t_u16 _wifi_cal_data_t::data_len

Calibration data length

7.1.2.2 t_u8* _wifi_cal_data_t::data

Calibration data

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

- wifi-decl.h

7.2 _wifi_mgmt_frame_tx Struct Reference

```
#include <wifi-decl.h>
```

Data Fields

- t_u16 frm_len
- t_u16 frm_ctl

Proprietary Information.
August 21, 2015 17:48

Copyright © 2015 Marvell

- t_u16 duration_id
- t_u8 addr1 [MLAN_MAC_ADDR_LENGTH]
- t_u8 addr2 [MLAN_MAC_ADDR_LENGTH]
- t_u8 addr3 [MLAN_MAC_ADDR_LENGTH]
- t_u16 seq_ctl
- t_u8 addr4 [MLAN_MAC_ADDR_LENGTH]
- t_u8 payload [0]

7.2.1 Detailed Description

802_11_header packet

7.2.2 Field Documentation

7.2.2.1 t_u16 _wifi_mgmt_frame_tx::frm_len

Packet Length

7.2.2.2 t_u16 _wifi_mgmt_frame_tx::frm_ctl

Frame Control

7.2.2.3 t_u16 _wifi_mgmt_frame_tx::duration_id

Duration ID

7.2.2.4 t_u8 _wifi_mgmt_frame_tx::addr1[MLAN_MAC_ADDR_LENGTH]

Address1

7.2.2.5 t_u8 _wifi_mgmt_frame_tx::addr2[MLAN_MAC_ADDR_LENGTH]

Address2

7.2.2.6 t_u8 _wifi_mgmt_frame_tx::addr3[MLAN_MAC_ADDR_LENGTH]

Address3

7.2.2.7 t_u16 _wifi_mgmt_frame_tx::seq_ctl

Sequence Control

7.2.2.8 t_u8 _wifi_mgmt_frame_tx::addr4[MLAN_MAC_ADDR_LENGTH]

Address4

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.2.2.9 t_u8 _wifi_mgmt_frame_tx::payload[0]

Frame payload

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

- wifi-decl.h

7.3 app_init_state Struct Reference

```
#include <app_framework.h>
```

Data Fields

- char factory_reset
- char backup_fw
- uint32_t rst_cause

7.3.1 Detailed Description

This structure is passed along with the AF_EVT_INIT_DONE event to indicate the initial state.

7.3.2 Field Documentation

7.3.2.1 char app_init_state::factory_reset

Set to 1 if we are booting up after a factory reset

7.3.2.2 char app_init_state::backup_fw

Set to 1 if we are booting from backup firmware

7.3.2.3 uint32_t app_init_state::rst_cause

Previous reset cause

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

- app_framework.h

7.4 app_p2p_role_data Struct Reference

```
#include <app_framework.h>
```

Data Fields

- app_p2p_role_t role
- void * data

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.4.1 Detailed Description

This structure is passed along with the AF_EVT_P2P_STARTED and the AF_EVT_P2P_ROLE_NEGOTIATED events to indicate the current role. Refer app_p2p_role_t for details.

7.4.2 Field Documentation

7.4.2.1 app_p2p_role_t app_p2p_role_data::role

The P2P node type (client/GO)

7.4.2.2 void∗ app_p2p_role_data::data

Points to the name of the device

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

- app_framework.h

7.5 cli_command Struct Reference

```
#include <cli.h>
```

Data Fields

- const char ∗ name
- const char ∗ help
- void(∗ function)(int argc, char ∗∗argv)

7.5.1 Detailed Description

Structure for registering CLI commands

7.5.2 Field Documentation

7.5.2.1 const char∗ cli_command::name

The name of the CLI command

7.5.2.2 const char∗ cli_command::help

The help text associated with the command

7.5.2.3 void(∗ cli_command::function)(int argc, char ∗∗argv)

The function that should be invoked for this command.

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

- cli.h

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.6 DH_PG_PARAMS Struct Reference

```
#include <wmcrypto.h>
```

Data Fields

- unsigned char * prime
- unsigned int primeLen
- unsigned char * generator
- unsigned int generatorLen

7.6.1 Detailed Description

- Diffie-Hellman parameters.

7.6.2 Field Documentation

7.6.2.1 unsigned char* DH_PG_PARAMS::prime

prime

7.6.2.2 unsigned int DH_PG_PARAMS::primeLen

length of prime

7.6.2.3 unsigned char* DH_PG_PARAMS::generator

generator

7.6.2.4 unsigned int DH_PG_PARAMS::generatorLen

length of generator

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

- wmcrypto.h

7.7 dma_config_type Struct Reference

```
#include <mdev_dma.h>
```

Data Fields

- DMA_CFG_Type dma_cfg
- DMA_PerMapping_Type perDmaInter

7.7.1 Detailed Description

DMA configuration structure

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.7.2 Field Documentation

7.7.2.1 DMA_CFG_Type dma_config_type::dma_cfg

dma_cfg is the configuration structure that stores all relevant parameters corresponding to the transfer

7.7.2.2 DMA_PerMapping_Type dma_config_type::perDmaInter

Dma Peripheral mapping or handshaking interface is required for memory to/from peripheral transfer. It would not be required in case the transfer is of type MEM-MEM

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

- mdev_dma.h

7.8 flash_desc Struct Reference

```
#include <flash.h>
```

Data Fields

- uint8_t fl_dev
- uint32_t fl_start
- uint32_t fl_size

7.8.1 Detailed Description

The flash descriptor

All components that work with flash refer to the flash descriptor. The flash descriptor captures the details of the component resident in flash.

7.8.2 Field Documentation

7.8.2.1 uint8_t flash_desc::fl_dev

The flash device

7.8.2.2 uint32_t flash_desc::fl_start

The start address on flash

7.8.2.3 uint32_t flash_desc::fl_size

The size on flash

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

- flash.h

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.9 healthmon_handler Struct Reference

```
#include <healthmon.h>
```

Data Fields

- char name [HM_NAME_MAX+1]
- bool(∗ is_sick)(unsigned int cur_msec)
- void(∗ about_to_die)(bool is_failed_module)
- unsigned int check_interval
- unsigned char consecutive_failures

7.9.1 Detailed Description

Healthmon handler that is used for registration with healthmon thread. This structure instance's memory can be reclaimed immediately after successful registration.

7.9.2 Field Documentation

7.9.2.1 char healthmon_handler::name[HM_NAME_MAX+1]

Name of the module. This is used as identifier for handler

7.9.2.2 bool(∗ healthmon_handler::is_sick)(unsigned int cur_msec)

Function pointer to be registered for periodic call back to check module health

Parameters

cur_msec	Milli-seconds since device bootup

Returns

Should return FALSE if module is healthy Should return TRUE if module is not healthy

7.9.2.3 void(∗ healthmon_handler::about_to_die)(bool is_failed_module)

Function pointer to be registered that will be invoked before system dies as any of the registered module is not healthy.

Parameters

is_failed_module	Flag to indicate if this module is found unhealthy

7.9.2.4 unsigned int healthmon_handler::check_interval

Check interval for module represented in seconds

Note

check interval should be a multiple of WKUP_INTERVAL

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.9.2.5 unsigned char healthmon_handler::consecutive_failures

Number of consecutive unhealthy probes after which system should die

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

- healthmon.h

7.10 http_req_t Struct Reference

```
#include <httpc.h>
```

Data Fields

- http_method_t type
- const char * resource
- http_ver_t version
- const char * content
- int content_len

7.10.1 Detailed Description

The HTTP Client's Request structure

The application should fill up this structure and pass it to the http_prepare_req() API while making an HTTP request. For HTTP POST, the fields content and content_len of this structure have to be set as well. More name-value fields can be added to HTTP header using http_add_header().

7.10.2 Field Documentation

7.10.2.1 http_method_t http_req_t::type

The Type of HTTP Request

7.10.2.2 const char* http_req_t::resource

The target resource for the HTTP Request. A complete URL is also accepted.

7.10.2.3 http_ver_t http_req_t::version

The HTTP Protocol Version

7.10.2.4 const char* http_req_t::content

Pointer to data buffer. NULL if GET request

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.10.2.5 int http_req_t::content_len

The length of the data pointed to by *content* above. This is don't-care if the content is set to NULL

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

- httpc.h

7.11 http_resp_t Struct Reference

```
#include <httpc.h>
```

Data Fields

- const char * protocol
- http_ver_t version
- int status_code
- const char * reason_phrase
- const char * server
- const char * content_type
- const char * content_encoding
- bool keep_alive_ack
- int keep_alive_timeout
- int keep_alive_max
- bool chunked
- bool content_length_field_present
- uint32_t content_length

7.11.1 Detailed Description

Structure used to give back http header response fields to the caller.

7.11.2 Field Documentation

7.11.2.1 const char* http_resp_t::protocol

The value of the protocol field in the first line of the HTTP header response. e.g. "HTTP".

7.11.2.2 http_ver_t http_resp_t::version

HTTP version

7.11.2.3 int http_resp_t::status_code

The status code returned as a part of the first line of the HTTP response e.g. 200 if success

7.11.2.4 const char* http_resp_t::reason_phrase

The ASCII string present in the first line of HTTP response. It is the verbose representation of status code. e.g. "OK" if status_code is 200

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.11.2.5 const char* http_resp_t::server

HTTP "Server" header field value

7.11.2.6 const char* http_resp_t::content_type

The value of "Content-Type" header field. e.g. "text/html"

7.11.2.7 const char* http_resp_t::content_encoding

The value of "Content-Encoding" header field e.g. "gzip"

7.11.2.8 bool http_resp_t::keep_alive_ack

If "Keep-Alive" field is present or if the value of "Connection" field is "Keep-Alive" then this member is set to 'true'. It is set to 'false' in other cases

7.11.2.9 int http_resp_t::keep_alive_timeout

If "Keep-Alive" field is present in the response, this member contains the value of the "timeout" sub-field of this field. This is the time the server will allow an idle connection to remain open before it is closed.

7.11.2.10 int http_resp_t::keep_alive_max

If "Keep-Alive" field is present in the response, this member contains the value of the "max" sub-field of this field. The max parameter indicates the maximum number of requests that a client will make, or that a server will allow to be made on the persistent connection.

7.11.2.11 bool http_resp_t::chunked

This will be 'true' if "Transfer-Encoding" field is set to "chunked". Note that this is only for information and the API http_read_content() transparently handles chunked reads.

7.11.2.12 bool http_resp_t::content_length_field_present

Set to 'true' when "Content-Length" header field is present in server response.

7.11.2.13 uint32_t http_resp_t::content_length

Value of the "Content-Length" field. If "Transfer-Encoding" is set to chunked then this value will be zero. If 'content-_length_present' is 'false' ignore this field value.

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

- httpc.h

7.12 httpc_cfg_t Struct Reference

```
#include <httpc.h>
```


Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Data Fields

- int flags
- int retry_cnt

7.12.1 Detailed Description

HTTP session configuration. Passed to http_open_session()

7.12.2 Field Documentation

7.12.2.1 int httpc_cfg_t::flags

This is the bitwise OR of the members in http_hdr_field_sel_t. Use macro STANDARD_HDR_FLAGS if not sure

7.12.2.2 int httpc_cfg_t::retry_cnt

If 0 is passed default value is used

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

- httpc.h

7.13 httpd_request_t Struct Reference

```
#include <httpd.h>
```

Data Fields

- int type
- char filename [HTTPD_MAX_URI_LENGTH+1]
- int sock
- unsigned char hdr_parsed
- int remaining_bytes
- int body_nbytes
- struct httpd_wsgi_call * wsgi
- unsigned char chunked
- httpd_useragent_t agent
- char content_type [HTTPD_MAX_CONTENT_TYPE_LENGTH]
- bool if_none_match
- unsigned etag_val
- void * httpd_priv
- httpd_free_priv_data_fn_t free_priv_data_fn

7.13.1 Detailed Description

Request structure representing various properties of an HTTP request

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.13.2 Field Documentation

7.13.2.1 int httpd_request_t::type

HTTP Request type: GET, HEAD, POST, PUT, DELETE

7.13.2.2 char httpd_request_t::filename[HTTPD_MAX_URI_LENGTH+1]

The incoming URI

7.13.2.3 int httpd_request_t::sock

The socket of the incoming HTTP Request

7.13.2.4 unsigned char httpd_request_t::hdr_parsed

Indicator notifying whether the HTTP headers are parsed or not

7.13.2.5 int httpd_request_t::remaining_bytes

The number of data bytes not yet parsed

7.13.2.6 int httpd_request_t::body_nbytes

The size of data in the incoming HTTP Request

7.13.2.7 struct httpd_wsgi_call* httpd_request_t::wsgi

Pointer to the corresponding wsgi structure

7.13.2.8 unsigned char httpd_request_t::chunked

Set to 1 if chunked data is used

7.13.2.9 httpd_useragent_t httpd_request_t::agent

User-agent: Product and Version

7.13.2.10 char httpd_request_t::content_type[HTTPD_MAX_CONTENT_TYPE_LENGTH]

The content type of the incoming HTTP Request

7.13.2.11 bool httpd_request_t::if_none_match

True if "If-None-Match" header is present in the incoming HTTP Request

7.13.2.12 unsigned httpd_request_t::etag_val

Used for storing the etag of an URI

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.13.2.13 void* httpd_request_t::httpd_priv

Pointer for private data associated with an HTTP session. This will be NULL by default and can be set by applications if required.

7.13.2.14 httpd_free_priv_data_fn_t httpd_request_t::free_priv_data_fn

Pointer to a function to free the httpd_priv data. If not set, the default action is to free the httpd_priv data while closing the socket.

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

- httpd.h

7.14 httpd_ssi_call Struct Reference

```
#include <httpd.h>
```

Data Fields

- const char * name
- const httpd_ssifunction function

7.14.1 Detailed Description

Opaque structure representing a virtual SSI handler

Users should use the macro HTTPD_SSI_CALL to initialize this struct statically.

7.14.2 Field Documentation

7.14.2.1 const char* httpd_ssi_call::name

The name of the function, used in the script file

7.14.2.2 const httpd_ssifunction httpd_ssi_call::function

The pointer to the function that implements the handler

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

- httpd.h

7.15 httpd_tls_certs_t Struct Reference

```
#include <httpd.h>
```

Data Fields

- const unsigned char * server_cert
- bool server_cert_chained

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

- int server_cert_size
- const unsigned char * server_key
- int server_key_size
- const unsigned char * client_cert
- int client_cert_size

7.15.1 Detailed Description

Structure to be passed to httpd_use_tls_certificates()

7.15.2 Field Documentation

7.15.2.1 const unsigned char* httpd_tls_certs_t::server_cert

Mandatory. Will be sent to the client

7.15.2.2 bool httpd_tls_certs_t::server_cert_chained

Pass 'true' if certificate is chained

7.15.2.3 int httpd_tls_certs_t::server_cert_size

Size of server_cert

7.15.2.4 const unsigned char* httpd_tls_certs_t::server_key

Server private key. Mandatory. For the perusal of the server

7.15.2.5 int httpd_tls_certs_t::server_key_size

Size of server_key

7.15.2.6 const unsigned char* httpd_tls_certs_t::client_cert

CA cert to verify client certificate. Needed if the server wants to verify client certificate. Otherwise set to NULL.

7.15.2.7 int httpd_tls_certs_t::client_cert_size

Size of client_cert

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

- httpd.h

7.16 httpd_useragent_t Struct Reference

```
#include <httpd.h>
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Data Fields

- char product [HTTPD_MAX_VAL_LENGTH+1]
- char version [HTTPD_MAX_VAL_LENGTH+1]

7.16.1 Detailed Description

httpd_useragent_t reports the product and version of the client accessing the httpd.

7.16.2 Field Documentation

7.16.2.1 char httpd_useragent_t::product[HTTPD_MAX_VAL_LENGTH+1]

Product of the client

7.16.2.2 char httpd_useragent_t::version[HTTPD_MAX_VAL_LENGTH+1]

Version of the client

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

- httpd.h

7.17 httpd_wsgi_call Struct Reference

```
#include <httpd.h>
```

Data Fields

- const char * uri
- int hdr_fields
- int http_flags
- int(* get_handler)(httpd_request_t *req)
- int(* set_handler)(httpd_request_t *req)
- int(* put_handler)(httpd_request_t *req)
- int(* delete_handler)(httpd_request_t *req)

7.17.1 Detailed Description

Structure representing a WSGI handler

7.17.2 Field Documentation

7.17.2.1 const char* httpd_wsgi_call::uri

URI of the WSGI

7.17.2.2 int httpd_wsgi_call::hdr_fields

Indicator for HTTP headers to be sent in the response. This should be logical OR of the flags defined in httpd_hdr_field_sel_t. HTTPD_DEFAULT_HDR_FLAGS is typically used.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.17.2.3 int httpd_wsgi_call::http_flags

Flag indicating if exact match of the URI is required or not

7.17.2.4 int(* httpd_wsgi_call::get_handler)(httpd_request_t *req)

HTTP GET or HEAD Handler

7.17.2.5 int(* httpd_wsgi_call::set_handler)(httpd_request_t *req)

HTTP POST Handler

7.17.2.6 int(* httpd_wsgi_call::put_handler)(httpd_request_t *req)

HTTP PUT Handler

7.17.2.7 int(* httpd_wsgi_call::delete_handler)(httpd_request_t *req)

HTTP DELETE Handler

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

- httpd.h

7.18 input_gpio_cfg_t Struct Reference

```
#include <generic_io.h>
```

Data Fields

- int gpio
- bool type

7.18.1 Detailed Description

Input GPIO Configuration type. Normally used for push buttons.

7.18.2 Field Documentation

7.18.2.1 int input_gpio_cfg_t::gpio

The GPIO number associated with this input.

7.18.2.2 bool input_gpio_cfg_t::type

The type of the input. Can be either GPIO_ACTIVE_LOW or GPIO_ACTIVE_HIGH. This indicates what the state of the GPIO will be when the button is pressed.

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

- generic_io.h

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.19 ipv4_config Struct Reference

```
#include <wlan.h>
```

Data Fields

- unsigned addr_type:2
- unsigned address
- unsigned gw
- unsigned netmask
- unsigned dns1
- unsigned dns2

7.19.1 Detailed Description

This data structure represents an IPv4 address

7.19.2 Field Documentation

7.19.2.1 unsigned ipv4_config::addr_type

Set to ADDR_TYPE_DHCP to use DHCP to obtain the IP address or ADDR_TYPE_STATIC to use a static IP. In case of static IP Address ip, gw, netmask and dns members must be specified. When using DHCP, the ip, gw, netmask and dns are overwritten by the values obtained from the DHCP Server. They should be zeroed out if not used.

7.19.2.2 unsigned ipv4_config::address

The system's IP address in network order.

7.19.2.3 unsigned ipv4_config::gw

The system's default gateway in network order.

7.19.2.4 unsigned ipv4_config::netmask

The system's subnet mask in network order.

7.19.2.5 unsigned ipv4_config::dns1

The system's primary dns server in network order.

7.19.2.6 unsigned ipv4_config::dns2

The system's secondary dns server in network order.

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

- wlan.h

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.20 jobj_t Struct Reference

```
#include <jsonv2.h>
```

Data Fields

- json_parser_t parser
- char * js
- jsontok_t * tokens
- jsontok_t * cur
- int num_tokens

7.20.1 Detailed Description

Object used by the JSON parser internally. This object needs to be defined by the application but all the elements will be set internally by the JSON parser. Please refer Usage for more information.

Note

Please do not modify any elements of this object

7.20.2 Field Documentation

7.20.2.1 json_parser_t jobj_t::parser

Internal JSON parser object

7.20.2.2 char* jobj_t::js

Pointer to JSON string

7.20.2.3 jsontok_t* jobj_t::tokens

Pointer to JSON tokens array

7.20.2.4 jsontok_t* jobj_t::cur

Pointer to current active JSON token

7.20.2.5 int jobj_t::num_tokens

Number of tokens available for parser

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

- jsonv2.h

7.21 jsontok_t Struct Reference

```
#include <jsonv2.h>
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Data Fields

- uint8_t type
- uint8_t size
- int16_t start
- int16_t end
- int16_t parent

7.21.1 Detailed Description

JSON token. Application just needs to define an array of these tokens and pass it to json_init().

7.21.2 Field Documentation

7.21.2.1 uint8_t jsontok_t::type

Type of the token, like object, string, array, etc.

7.21.2.2 uint8_t jsontok_t::size

Number of children of this token. Valid for objects and arrays.

7.21.2.3 int16_t jsontok_t::start

Start position in JSON data string.

7.21.2.4 int16_t jsontok_t::end

End position in JSON data string.

7.21.2.5 int16_t jsontok_t::parent

Parent token's index in the token's arrays.

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

- jsonv2.h

7.22 mdns_service Struct Reference

```
#include <mdns.h>
```

Data Fields

- const char * servname
- const char * servtype
- const char * domain
- uint16_t port
- int proto
- char * keyvals

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

- uint32_t ipaddr
- uint8_t fqsn [MDNS_MAX_NAME_LEN]
- uint8_t * ptrname
- uint16_t kvlen
- uint32_t flags

7.22.1 Detailed Description

service descriptor

Central to mdns is the notion of a service. Hosts advertise service types such as a website, a printer, some custom service, etc. Network users can use an mdns browser to discover services on the network. Internally, this mdns implementation uses the following struct to describe a service. These structs can be created by a user, populated, and passed to mdns announce functions to specify services that are to be advertised. When a user starts a query for services, the discovered services are passed back to the user in this struct.

The members include:

servname: string that is the service instance name that will be advertised. It could be something like "Brian's Website" or "Special Service on Device #123". This is the name that is typically presented to users browsing for your service. The servname must not exceed MDNS_MAX_LABEL_LEN bytes. The MDNS specification allows servname to be a UTF8 string. However, only the ascii subset of UTF-8 has been tested.

servtype: string that represents the service type. This should be a type registered at `http://dns-sd.org/-ServiceTypes.html`. For example, "http" is the service type for a web server and "ssh" is for an ssh server. You may use an unregisterd service type during development, but not in released products. Consider registering any new service types at the aforementioned webpage. servtype must be non-NULL.

domain: string that represents the domain. If this value is NULL, domain name ".local" will be used. The domain must not exceed MDNS_MAX_LABEL_LEN bytes.

port: the tcp or udp port on which the service named servname is available in network byte order.

proto: Either MDNS_PROTO_TCP or MDNS_PROTO_UDP depending on what protocol clients should use to connect to the service servtype.

keyvals: NULL-terminated string of colon-separated key=value pairs. These are the key/value pairs for the TXT record associated with a service type. For example, the servtype "http" defines the TXT keys "u", "p", and "path" for the username, password, and path to a document. If you supplied all of these, the keyvals string would be:

"u=myusername:p=mypassword:path=/index.html"

If keyvals is NULL, no TXT record will be advertised. If keyvals is ":", a TXT record will appear, but it will not contain any key/value pairs. The key must be present (i.e., two contiguous ':' characters should never appear in the keyvals string.) A key may appear with no value. The interpretation of this depends on the nature of the service. The length of a single key/value pair cannot exceed MDNS_MAX_KEYVAL_LEN bytes. It is the responsibility of the application to verify that the keyval string is a valid string. The keyval string buffer is used by the mDNS module internally and it can modify it. Hence, during subsequent calls to the mDNS module, it is possible that the original string has been messed up and needs to be recreated.

ipaddr, fqsn, ptrname, kvlen and flags are for internal use only and should not be dereferenced by the user.

7.22.2 Field Documentation

7.22.2.1 const char* mdns_service::servname

Name of MDNS service

7.22.2.2 const char* mdns_service::servtype

Type of MDNS service

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.22.2.3 const char* mdns_service::domain

Domain for MDNS service

7.22.2.4 uint16_t mdns_service::port

Port number

7.22.2.5 int mdns_service::proto

Protocol used

7.22.2.6 char* mdns_service::keyvals

Key value pairs for TXT records

7.22.2.7 uint32_t mdns_service::ipaddr

IP Address of device

7.22.2.8 uint8_t mdns_service::fqsn[MDNS_MAX_NAME_LEN]

The following members are for internal use only and should not be dereferenced by the user.

7.22.2.9 uint8_t* mdns_service::ptrname

PTR record name

7.22.2.10 uint16_t mdns_service::kvlen

Length of keyvals string

7.22.2.11 uint32_t mdns_service::flags

MDNS flags

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

- mdns.h

7.23 os_queue_pool Struct Reference

```
#include <wm_os.h>
```

Data Fields

- int size

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.23.1 Detailed Description

Structure used for queue definition

7.23.2 Field Documentation

7.23.2.1 int os_queue_pool::size

Size of the queue

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

- wm_os.h

7.24 os_thread_stack Struct Reference

```
#include <wm_os.h>
```

Data Fields

- int size

7.24.1 Detailed Description

Structure to be used during call to the function os_thread_create(). Please use the macro os_thread_stack_define instead of using this structure directly.

7.24.2 Field Documentation

7.24.2.1 int os_thread_stack::size

Total stack size

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

- wm_os.h

7.25 output_gpio_cfg_t Struct Reference

```
#include <generic_io.h>
```

Data Fields

- int gpio
- bool type

7.25.1 Detailed Description

Output GPIO Configuration type. Normally used for LED indicators or for driving loads.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.25.2 Field Documentation

7.25.2.1 int output_gpio_cfg_t::gpio

The GPIO number associated with this output.

7.25.2.2 bool output_gpio_cfg_t::type

The type of the output. Can be either GPIO_ACTIVE_LOW or GPIO_ACTIVE_HIGH.

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

- generic_io.h

7.26 overlay Struct Reference

```
#include <overlays.h>
```

Data Fields

- char * o_name
- char * o_ram_start
- unsigned char o_no_parts
- struct overlay_ranges * o_range

7.26.1 Detailed Description

The overlay structure

7.26.2 Field Documentation

7.26.2.1 char* overlay::o_name

A user-friendly name for the overlay

7.26.2.2 char* overlay::o_ram_start

The start address of the overlay in RAM

7.26.2.3 unsigned char overlay::o_no_parts

The number of parts that can be overlaid on top of each other

7.26.2.4 struct overlay_ranges* overlay::o_range

A pointer to an array that contains the flash parts

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

- overlays.h

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.27 overlay_ranges Struct Reference

```
#include <overlays.h>
```

Data Fields

- char * or_flash_start
- char * or_flash_end
- char * o_bss_start
- char * o_bss_end

7.27.1 Detailed Description

Overlay Range

7.27.2 Field Documentation

7.27.2.1 char* overlay_ranges::or_flash_start

The start address in flash of one range

7.27.2.2 char* overlay_ranges::or_flash_end

The end address in flash of one range

7.27.2.3 char* overlay_ranges::o_bss_start

The start address of the bss section that should be zeroed out

7.27.2.4 char* overlay_ranges::o_bss_end

The end address of the bss section, before which the zeroing will stop

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

- overlays.h

7.28 p2p_config Struct Reference

```
#include <p2p.h>
```

Data Fields

- uint8_t role
- uint8_t auto_go
- uint8_t pin_generator
- uint8_t version
- uint8_t version2
- uint8_t device_name [32]
- uint8_t manufacture [64]

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

- uint8_t model_name [32]
- uint8_t model_number [32]
- uint8_t serial_number [32]
- uint16_t config_methods
- uint16_t primary_dev_category
- uint16_t primary_dev_subcategory
- uint8_t rf_bands
- uint32_t os_version
- uint8_t p2p_msg_max_retry
- uint32_t p2p_msg_timeout
- uint16_t pin_len
- int(∗ p2p_callback)(enum p2p_event event, void ∗data, uint16_t len)

7.28.1 Detailed Description

This struct is passed to p2p_start(). The user must initialize it with parameters as described inline.

7.28.2 Field Documentation

7.28.2.1 uint8_t p2p_config::role

WiFi Direct mode:4

7.28.2.2 uint8_t p2p_config::auto_go

Auto Group Owner

7.28.2.3 uint8_t p2p_config::pin_generator

Pin Generator. Wfd Auto Display PIN:3, Wfd Auto Enter PIN:4

7.28.2.4 uint8_t p2p_config::version

Version: 0x10: Version 1.0

7.28.2.5 uint8_t p2p_config::version2

Version: 0x20: Version 2.0

7.28.2.6 uint8_t p2p_config::device_name[32]

Device name: describes the product

7.28.2.7 uint8_t p2p_config::manufacture[64]

Manufacture: identifies the manufacturer of the device

7.28.2.8 uint8_t p2p_config::model_name[32]

Model name: identifies the model of the device

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.28.2.9 uint8_t p2p_config::model_number[32]

Model number: provides additional description of the device

7.28.2.10 uint8_t p2p_config::serial_number[32]

Serial number: identifies the serial number of the Enrollee or Registrar

7.28.2.11 uint16_t p2p_config::config_methods

Configuration Methods: 0x0001 : USBA 0x0002 : Ethernet 0x0004 : Label 0x0008 : Display 0x0010 : Ethernet NFC Token 0x0020 : Integrated NFC Token 0x0040 : NFC Interface 0x0080 : Push Button 0x0100 : Keypad

7.28.2.12 uint16_t p2p_config::primary_dev_category

Primary Device category: identifies device type category of the Enrollee or Registrar

7.28.2.13 uint16_t p2p_config::primary_dev_subcategory

Primary Device subcategory: identifies device type subcategory of the Enrollee or Registrar

7.28.2.14 uint8_t p2p_config::rf_bands

RF bands

7.28.2.15 uint32_t p2p_config::os_version

OS Version

7.28.2.16 uint8_t p2p_config::p2p_msg_max_retry

P2P message max retry

7.28.2.17 uint32_t p2p_config::p2p_msg_timeout

P2P message timeout: Value for Message transmit, timeout unit is msec

7.28.2.18 uint16_t p2p_config::pin_len

PIN length: Length of the P2P PIN: 8 digits

7.28.2.19 int(* p2p_config::p2p_callback)(enum p2p_event event, void *data, uint16_t len)

P2P callback: Callback function to report events back

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

- p2p.h

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.29 p2p_scan_result Struct Reference

```
#include <p2p.h>
```

Data Fields

- uint8_t group_capability
- uint8_t device_id [IEEEtypes_ADDRESS_SIZE]
- char device_name [MAX_DEVICE_NAME_LEN+1]
- uint8_t device_address [IEEEtypes_ADDRESS_SIZE]
- uint8_t go_ssid [IEEEtypes_SSID_SIZE+1]
- uint8_t intended_address [IEEEtypes_ADDRESS_SIZE]
- int op_channel
- int listen_channel

7.29.1 Detailed Description

P2P Device Information to display

7.29.2 Field Documentation

7.29.2.1 uint8_t p2p_scan_result::group_capability

Group capability

7.29.2.2 uint8_t p2p_scan_result::device_id[IEEEtypes_ADDRESS_SIZE]

Device Identifier ; Interface address

7.29.2.3 char p2p_scan_result::device_name[MAX_DEVICE_NAME_LEN+1]

Device Name

7.29.2.4 uint8_t p2p_scan_result::device_address[IEEEtypes_ADDRESS_SIZE]

Device address in deviceInfo

7.29.2.5 uint8_t p2p_scan_result::go_ssid[IEEEtypes_SSID_SIZE+1]

P2P GO SSID

7.29.2.6 uint8_t p2p_scan_result::intended_address[IEEEtypes_ADDRESS_SIZE]

Intended address from Peer

7.29.2.7 int p2p_scan_result::op_channel

Operating channel

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.29.2.8 int p2p_scan_result::listen_channel

Listen channel

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

- p2p.h

7.30 partition_entry Struct Reference

```
#include <partition.h>
```

Data Fields

- uint8_t type
- uint8_t device
- char name [MAX_NAME]
- uint32_t start
- uint32_t size
- uint32_t gen_level

7.30.1 Detailed Description

Partition Entry

7.30.2 Field Documentation

7.30.2.1 uint8_t partition_entry::type

The type of the flash component

7.30.2.2 uint8_t partition_entry::device

The device id, internal flash is always id 0

7.30.2.3 char partition_entry::name[MAX_NAME]

A descriptive component name

7.30.2.4 uint32_t partition_entry::start

Start address on the given device

7.30.2.5 uint32_t partition_entry::size

Size on the given device

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.30.2.6 uint32_t partition_entry::gen_level

Generation level

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

- partition.h

7.31 partition_table Struct Reference

```
#include <partition.h>
```

Data Fields

- uint32_t magic
- uint16_t version
- uint16_t partition_entries_no
- uint32_t gen_level
- uint32_t crc

7.31.1 Detailed Description

Partition Table

7.31.2 Field Documentation

7.31.2.1 uint32_t partition_table::magic

The magic identifying the start of the partition table

7.31.2.2 uint16_t partition_table::version

The version number of this partition table

7.31.2.3 uint16_t partition_table::partition_entries_no

The number of partition entries that follow this

7.31.2.4 uint32_t partition_table::gen_level

Generation level

7.31.2.5 uint32_t partition_table::crc

The CRC of all the above components

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

- partition.h

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.32 prov_scan_config Struct Reference

```
#include <provisioning.h>
```

Data Fields

- struct wifi_scan_params_t wifi_scan_params
- int scan_interval

7.32.1 Detailed Description

This structure is used to represent scan configuration

7.32.2 Field Documentation

7.32.2.1 struct wifi_scan_params_t prov_scan_config::wifi_scan_params

Wireless scan parameters as defined at struct wifi_scan_params_t

7.32.2.2 int prov_scan_config::scan_interval

Interval between consecutive scans represented in seconds. This value should not be less than 2 seconds and should not exceed 60 seconds.

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

- provisioning.h

7.33 provisioning_config Struct Reference

```
#include <provisioning.h>
```

Data Fields

- unsigned int prov_mode
- int(∗ provisioning_event_handler)(enum prov_event event, void ∗arg, int len)

7.33.1 Detailed Description

This structure is used to configure provisioning module with application specific settings.

7.33.2 Field Documentation

7.33.2.1 unsigned int provisioning_config::prov_mode

Provisioning mode; could be PROVISIONING_WLANNW and/or PROVISIONING_WPS or PROVISIONING_EZCONNECT. Note that while PROVISIONING_WLANNW and PROVISIONING_WPS can be simultaneously used, PROVISIONING_EZCONNECT can only be used by itself.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.33.2.2 int(∗ provisioning_config::provisioning_event_handler)(enum prov_event event, void ∗arg, int len)

Callback function pointer that will be invoked for any events to be given to the application. These events are listed in enum prov_event. arg is a generic pointer that may be valid when data is to be received along with event. In such case len contains length of data passed along.

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

- provisioning.h

7.34 psm_cfg_t Struct Reference

```
#include <psm-v2.h>
```

Data Fields

- bool read_only
- bool secure

7.34.1 Detailed Description

PSM Configuration

This structure is passed to psm_module_init(). PSMv2 won't need the user copy of this structure after call to psm_module_init() returns.

7.34.2 Field Documentation

7.34.2.1 bool psm_cfg_t::read_only

```
If read-only is enabled no writes will be possible to the
```

partition

7.34.2.2 bool psm_cfg_t::secure

If 'secure' is set, the object will be encrypted before writing, and object will be decrypted before reading. If secure PSM is enabled KEY_PSM_ENCRYPT_KEY and KEY_PSM_NONCE needs to be present in the keystore. Without these keys the PSM initialization will fail. This variable should not be set in case of mc200.

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

- psm-v2.h

7.35 psm_objattrib_t Struct Reference

```
#include <psm-v2.h>
```

Data Fields

- uint32_t flags

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.35.1 Detailed Description

PSM Object attributes

This structure is accepted by psm_object_open()

7.35.2 Field Documentation

7.35.2.1 uint32_t psm_objattrib_t::flags

OR of the flags defined in psm_flags_t

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

- psm-v2.h

7.36 tls_client_t Struct Reference

```
#include <wm-tls.h>
```

Data Fields

- const unsigned char * ca_cert
- int ca_cert_size
- const unsigned char * client_cert
- bool client_cert_is_chained
- int client_cert_size
- const unsigned char * client_key
- int client_key_size

7.36.1 Detailed Description

All the below certificates and keys have to be encoded in the PEM format.

7.36.2 Field Documentation

7.36.2.1 const unsigned char* tls_client_t::ca_cert

Needed if the client wishes to verify server certificate, Otherwise set to NULL. Chained buffers are automatically handled.

7.36.2.2 int tls_client_t::ca_cert_size

Size of CA_cert

7.36.2.3 const unsigned char* tls_client_t::client_cert

Needed if the server (the other end) mandates verification of client certificate. i.e. the server wants to verify if we are authentic. Set to NULL if not required. Typically, to connect to any public server (over SSL) like wikipedia, google this does not need to be set. This is for those special cases where the server wishes to ensure that it is talking with the correct client (us).

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.36.2.4 bool tls_client_t::client_cert_is_chained

Set this to true if client certificate is chained

7.36.2.5 int tls_client_t::client_cert_size

Size of client_cert

7.36.2.6 const unsigned char* tls_client_t::client_key

Client private key

7.36.2.7 int tls_client_t::client_key_size

Size of client key

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

- wm-tls.h

7.37 tls_init_config_t Struct Reference

```
#include <wm-tls.h>
```

Data Fields

- int flags
- union {
  - struct {
    - const unsigned char * ca_cert
    - int ca_cert_size
    - const unsigned char * client_cert
    - int client_cert_size
    - const unsigned char * client_key
    - int client_key_size
  - } client
  - struct {
    - const unsigned char * server_cert
    - int server_cert_size
    - const unsigned char * server_key
    - int server_key_size
    - const unsigned char * client_cert
    - int client_cert_size
  - } server
- } tls

7.37.1 Detailed Description

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Note

: When a certificate is passed to tls function its size parameter should be 1 less that sizeof(array) in which the certificate is present.
: The user buffers containing certificate and private key need not be available after return from tls_session_init(). CyaSSL maintains own copy.

7.37.2 Field Documentation

7.37.2.1 int tls_init_config_t::flags

OR of flags defined in tls_flags_t

7.37.2.2 const unsigned char* tls_init_config_t::ca_cert

Needed if the RADIUS server mandates verification of CA certificate. Otherwise set to NULL.

7.37.2.3 int tls_init_config_t::ca_cert_size

Size of CA_cert

7.37.2.4 const unsigned char* tls_init_config_t::client_cert

Needed if the server mandates verification of client certificate. Otherwise set to NULL. In the former case please OR the flag TLS_USE_CLIENT_CERT to flags variable in tls_init_config_t passed to tls_session_init()

Needed if the server wants to verify client certificate. Otherwise set to NULL.

7.37.2.5 int tls_init_config_t::client_cert_size

Size of client_cert

7.37.2.6 const unsigned char* tls_init_config_t::client_key

Client private key

7.37.2.7 int tls_init_config_t::client_key_size

Size of client key

7.37.2.8 struct { ... } tls_init_config_t::client

Structure for client TLS configuration

7.37.2.9 const unsigned char* tls_init_config_t::server_cert

Mandatory. Will be sent to the client

7.37.2.10 int tls_init_config_t::server_cert_size

Size of server_cert

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.37.2.11 const unsigned char* tls_init_config_t::server_key

Server private key. Mandatory. For the perusal of the server

7.37.2.12 int tls_init_config_t::server_key_size

Size of server_key

7.37.2.13 struct { ... } tls_init_config_t::server

Structure for server TLS configuration

7.37.2.14 union { ... } tls_init_config_t::tls

Either a client or a server can be configured at a time through tls_session_init(). Fill up appropriate structure from the below union depending on your requirement.

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

- wm-tls.h

7.38 tm Struct Reference

```
#include <wmtime.h>
```

Data Fields

- int tm_sec
- int tm_min
- int tm_hour
- int tm_mday
- int tm_mon
- int tm_year
- int tm_wday

7.38.1 Detailed Description

Abridged version of struct tm (refer 'man gmtime' found in standard C library

7.38.2 Field Documentation

7.38.2.1 int tm::tm_sec

seconds [0-59]

7.38.2.2 int tm::tm_min

minutes [0-59]

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.38.2.3 int tm::tm_hour

hours [0-23]

7.38.2.4 int tm::tm_mday

day of the month [1-31]

7.38.2.5 int tm::tm_mon

month [0-11]

7.38.2.6 int tm::tm_year

year. The number of years since 1900

7.38.2.7 int tm::tm_wday

day of the week [0-6] 0-Sunday...6-Saturday

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

- wmtime.h

7.39 update_desc Struct Reference

```
#include <rfget.h>
```

Data Fields

- struct partition_entry * pe
- mdev_t * dev
- uint32_t addr
- uint32_t remaining

7.39.1 Detailed Description

Update Description Structure

This is referenced when using the rfget_update_ APIs. The contents of this structure are populated and used by the rfget_update_ APIs.

7.39.2 Field Documentation

7.39.2.1 struct partition_entry* update_desc::pe

Pointer to partition entry structure this is the partition being updated

7.39.2.2 mdev_t* update_desc::dev

MDEV handle to flash driver

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.39.2.3 uint32_t update_desc::addr

Address in flash where update occurs

7.39.2.4 uint32_t update_desc::remaining

Remaining space in partition being updated

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

- rfget.h

7.40 upgrade_struct_t Struct Reference

```
#include <secure_upgrade.h>
```

Data Fields

- decrypt_init_func_t decrypt_init
- decrypt_func_t decrypt
- signature_verify_func_t signature_verify
- hash_update_func_t hash_update
- hash_finish_func_t hash_finish
- fw_data_read_fn_t fw_data_read_fn
- void * decrypt_ctx
- void * hash_ctx
- void * iv
- int iv_len
- void * signature
- int signature_len
- void * hash
- int hash_len
- void * pk
- void * decrypt_key
- void * session_priv_data
- int bytes_to_read

7.40.1 Detailed Description

Upgrade structure

Structure to be populated by the application and passed to client_mode_secure_upgrade() or server_mode_secure_upgrade()

7.40.2 Field Documentation

7.40.2.1 decrypt_init_func_t upgrade_struct_t::decrypt_init

Pointer to the decryption context initialization function of type decrypt_init_func_t

7.40.2.2 decrypt_func_t upgrade_struct_t::decrypt

Pointer to the decryption function of type decrypt_func_t

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.40.2.3 signature_verify_func_t upgrade_struct_t::signature_verify

Pointer to the signature verification function of type signature_verify_func_t

7.40.2.4 hash_update_func_t upgrade_struct_t::hash_update

Pointer to the hash update function of type hash_update_func_t

7.40.2.5 hash_finish_func_t upgrade_struct_t::hash_finish

Pointer to a hash finish function of type hash_finish_func_t

7.40.2.6 fw_data_read_fn_t upgrade_struct_t::fw_data_read_fn

Pointer to a data read function of type fw_data_read_fn_t that will be used to fetch the firmware data.

7.40.2.7 void∗ upgrade_struct_t::decrypt_ctx

Pointer to the decrypt context initialized by the application before using the secure_upgrade APIs

7.40.2.8 void∗ upgrade_struct_t::hash_ctx

Pointer to the hash context initialized by the application before using the secure_upgrade APIs

7.40.2.9 void∗ upgrade_struct_t::iv

Pointer to a buffer which will hold the decryption context initialization vector received from the server. This must point to an allocated buffer of size equal to the IV required by the decryption algorithm being used.

7.40.2.10 int upgrade_struct_t::iv_len

Length of the IV as per the decryption algorithm being used. This must be initialized by the application.

7.40.2.11 void∗ upgrade_struct_t::signature

Pointer to a buffer which holds the received signature. This must point to an allocated buffer which holds the actual decrypted signature received from the server.

7.40.2.12 int upgrade_struct_t::signature_len

Length of the signature as per the algorithm being used. This must be initialized by the application.

7.40.2.13 void∗ upgrade_struct_t::hash

Pointer to a buffer which holds the hash value. The memory must be allocated by the application.

7.40.2.14 int upgrade_struct_t::hash_len

Length of the hash as per the hashing algorithm being used.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.40.2.15 void* upgrade_struct_t::pk

Pointer to the server's public key for verification.

7.40.2.16 void* upgrade_struct_t::decrypt_key

Pointer to the decryption key which will later be passed to the decrypt_init function.

7.40.2.17 void* upgrade_struct_t::session_priv_data

Private data associated with a firmware upgrade session. This will be passed as an argument to fw_data_read_fn_t.

7.40.2.18 int upgrade_struct_t::bytes_to_read

Number of bytes remaining to read. For internal use only. Do not modify.

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

- secure_upgrade.h

7.41 wifi_domain_param_t Struct Reference

```
#include <wifi-decl.h>
```

Data Fields

- t_u8 country_code [COUNTRY_CODE_LEN]
- int no_of_sub_band
- wifi_sub_band_set_t sub_band [1]

7.41.1 Detailed Description

Data structure for domain parameters

This structure is accepted by wlan_uap_set_domain_params() API. This information is used to generate the country info IE.

7.41.2 Field Documentation

7.41.2.1 t_u8 wifi_domain_param_t::country_code[COUNTRY_CODE_LEN]

Country code

7.41.2.2 int wifi_domain_param_t::no_of_sub_band

subbands count

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.41.2.3 wifi_sub_band_set_t wifi_domain_param_t::sub_band[1]

Set of subbands of no_of_sub_band number of elements

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

- wifi-decl.h

7.42 wifi_sub_band_set_t Struct Reference

```
#include <wifi-decl.h>
```

Data Fields

- t_u8 first_chan
- t_u8 no_of_chan
- t_u8 max_tx_pwr

7.42.1 Detailed Description

Data structure for subband set

For uAP 11d support

7.42.2 Field Documentation

7.42.2.1 t_u8 wifi_sub_band_set_t::first_chan

First channel

7.42.2.2 t_u8 wifi_sub_band_set_t::no_of_chan

Number of channels

7.42.2.3 t_u8 wifi_sub_band_set_t::max_tx_pwr

Maximum Tx power in dBm

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

- wifi-decl.h

7.43 wlan_ip_config Struct Reference

```
#include <wlan.h>
```

Data Fields

- struct ipv4_config ipv4

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.43.1 Detailed Description

Network IP configuration.

This data structure represents the network IP configuration for IPv4 as well as IPv6 addresses

7.43.2 Field Documentation

7.43.2.1 struct ipv4_config wlan_ip_config::ipv4

The network IPv4 address configuration that should be associated with this interface.

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

- wlan.h

7.44 wlan_network Struct Reference

```
#include <wlan.h>
```

Data Fields

- char name [WLAN_NETWORK_NAME_MAX_LENGTH]
- char ssid [IEEEtypes_SSID_SIZE+1]
- char bssid [IEEEtypes_ADDRESS_SIZE]
- unsigned int channel
- enum wlan_bss_type type
- enum wlan_bss_role role
- struct wlan_network_security security
- struct wlan_ip_config ip
- unsigned ssid_specific:1
- unsigned bssid_specific:1
- unsigned channel_specific:1
- unsigned security_specific:1

7.44.1 Detailed Description

WLAN Network Profile

This data structure represents a WLAN network profile. It consists of an arbitrary name, WiFi configuration, and IP address configuration.

Every network profile is associated with one of the two interfaces. The network profile can be used for the station interface (i.e. to connect to an Access Point) by setting the role field to WLAN_BSS_ROLE_STA. The network profile can be used for the micro-AP interface (i.e. to start a network of our own.) by setting the mode field to WLAN_BSS_ROLE_UAP.

If the mode field is WLAN_BSS_ROLE_STA, either of the SSID or BSSID fields are used to identify the network, while the other members like channel and security settings characterize the network.

If the mode field is WLAN_BSS_ROLE_UAP, the SSID, channel and security fields are used to define the network to be started.

In both the above cases, the address field is used to determine the type of address assignment to be used for this interface.

Proprietary Information.
August 21, 2015 17:48

Copyright © 2015 Marvell

7.44.2 Field Documentation

7.44.2.1 char wlan_network::name[WLAN_NETWORK_NAME_MAX_LENGTH]

The name of this network profile. Each network profile that is added to the WLAN Connection Manager must have a unique name.

7.44.2.2 char wlan_network::ssid[IEEEtypes_SSID_SIZE+1]

The network SSID, represented as a C string of up to 32 characters in length. If this profile is used in the micro-AP mode, this field is used as the SSID of the network. If this profile is used in the station mode, this field is used to identify the network. Set the first byte of the SSID to NULL (a 0-length string) to use only the BSSID to find the network.

7.44.2.3 char wlan_network::bssid[IEEEtypes_ADDRESS_SIZE]

The network BSSID, represented as a 6-byte array. If this profile is used in the micro-AP mode, this field is ignored. If this profile is used in the station mode, this field is used to identify the network. Set all 6 bytes to 0 to use any BSSID, in which case only the SSID will be used to find the network.

7.44.2.4 unsigned int wlan_network::channel

The channel for this network.

If this profile is used in micro-AP mode, this field specifies the channel to start the micro-AP interface on. Set this to 0 for auto channel selection.

If this profile is used in the station mode, this constrains the channel on which the network to connect should be present. Set this to 0 to allow the network to be found on any channel.

7.44.2.5 enum wlan_bss_type wlan_network::type

BSS type

7.44.2.6 enum wlan_bss_role wlan_network::role

The network wireless mode enum wlan_bss_role. Set this to specify what type of wireless network mode to use. This can either be WLAN_BSS_ROLE_STA for use in the station mode, or it can be WLAN_BSS_ROLE_UAP for use in the micro-AP mode.

7.44.2.7 struct wlan_network_security wlan_network::security

The network security configuration specified by struct wlan_network_security for the network.

7.44.2.8 struct wlan_ip_config wlan_network::ip

The network IP address configuration specified by struct wlan_ip_config that should be associated with this interface.

7.44.2.9 unsigned wlan_network::ssid_specific

If set to 1, the ssid field contains the specific SSID for this network. The WLAN Connection Manager will only connect to networks whose SSID matches. If set to 0, the ssid field contents are not used when deciding whether

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

to connect to a network, the BSSID field is used instead and any network whose BSSID matches is accepted.

This field will be set to 1 if the network is added with the SSID specified (not an empty string), otherwise it is set to 0.

7.44.2.10 unsigned wlan_network::bssid_specific

If set to 1, the bssid field contains the specific BSSID for this network. The WLAN Connection Manager will not connect to any other network with the same SSID unless the BSSID matches. If set to 0, the WLAN Connection Manager will connect to any network whose SSID matches.

This field will be set to 1 if the network is added with the BSSID specified (not set to all zeroes), otherwise it is set to 0.

7.44.2.11 unsigned wlan_network::channel_specific

If set to 1, the channel field contains the specific channel for this network. The WLAN Connection Manager will not look for this network on any other channel. If set to 0, the WLAN Connection Manager will look for this network on any available channel.

This field will be set to 1 if the network is added with the channel specified (not set to 0), otherwise it is set to 0.

7.44.2.12 unsigned wlan_network::security_specific

If set to 0, any security that matches is used. This field is internally set when the security type parameter above is set to WLAN_SECURITY_WILDCARD.

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

- wlan.h

7.45 wlan_network_security Struct Reference

```
#include <wlan.h>
```

Data Fields

- enum wlan_security_type type
- struct wlan_cipher mcstCipher
- struct wlan_cipher ucstCipher
- bool is_pmf_required
- char psk [WLAN_PSK_MAX_LENGTH]
- char psk_len
- char pmk [WLAN_PMK_LENGTH]
- char pmk_valid

7.45.1 Detailed Description

Network security configuration

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

7.45.2 Field Documentation

7.45.2.1 enum wlan_security_type wlan_network_security::type

Type of network security to use specified by enum wlan_security_type.

7.45.2.2 struct wlan_cipher wlan_network_security::mcstCipher

Type of network security Group Cipher suite used internally

7.45.2.3 struct wlan_cipher wlan_network_security::ucstCipher

Type of network security Pairwise Cipher suite used internally

7.45.2.4 bool wlan_network_security::is_pmf_required

Is PMF required

7.45.2.5 char wlan_network_security::psk[WLAN_PSK_MAX_LENGTH]

Pre-shared key (network password). For WEP networks this is a hex byte sequence of length psk_len, for WPA and WPA2 networks this is an ASCII pass-phrase of length psk_len. This field is ignored for networks with no security.

7.45.2.6 char wlan_network_security::psk_len

Length of the WEP key or WPA/WPA2 pass phrase, 0 to WLAN_PSK_MAX_LENGTH. Ignored for networks with no security.

7.45.2.7 char wlan_network_security::pmk[WLAN_PMK_LENGTH]

Pairwise Master Key. When pmk_valid is set, this is the PMK calculated from the PSK for WPA/PSK networks. If pmk_valid is not set, this field is not valid. When adding networks with wlan_add_network, users can initialize pmk and set pmk_valid in lieu of setting the psk. After successfully connecting to a WPA/PSK network, users can call wlan_get_current_network to inspect pmk_valid and pmk. Thus, the pmk value can be populated in subsequent calls to wlan_add_network. This saves the CPU time required to otherwise calculate the PMK.

7.45.2.8 char wlan_network_security::pmk_valid

Flag reporting whether pmk is valid or not.

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

- wlan.h

7.46 wlan_scan_result Struct Reference

```
#include <wlan.h>
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Data Fields

- char ssid [33]
- unsigned int ssid_len
- char bssid [6]
- unsigned int channel
- enum wlan_bss_type type
- enum wlan_bss_role role
- unsigned wmm:1
- unsigned wep:1
- unsigned wpa:1
- unsigned wpa2:1
- unsigned char rssi

7.46.1 Detailed Description

Scan Result

7.46.2 Field Documentation

7.46.2.1 char wlan_scan_result::ssid[33]

The network SSID, represented as a NULL-terminated C string of 0 to 32 characters. If the network has a hidden SSID, this will be the empty string.

7.46.2.2 unsigned int wlan_scan_result::ssid_len

SSID length

7.46.2.3 char wlan_scan_result::bssid[6]

The network BSSID, represented as a 6-byte array.

7.46.2.4 unsigned int wlan_scan_result::channel

The network channel.

7.46.2.5 enum wlan_bss_type wlan_scan_result::type

The network wireless type.

7.46.2.6 enum wlan_bss_role wlan_scan_result::role

The network wireless mode.

7.46.2.7 unsigned wlan_scan_result::wmm

The network supports WMM. This is set to 0 if the network does not support WMM or if the system does not have WMM support enabled.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.46.2.8 unsigned wlan_scan_result::wep

The network uses WEP security.

7.46.2.9 unsigned wlan_scan_result::wpa

The network uses WPA security.

7.46.2.10 unsigned wlan_scan_result::wpa2

The network uses WPA2 security

7.46.2.11 unsigned char wlan_scan_result::rssi

The signal strength of the beacon

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

- wlan.h

7.47 wps_config Struct Reference

```
#include <wps.h>
```

Data Fields

- uint8_t role
- uint8_t pin_generator
- uint8_t version
- uint8_t version2
- uint8_t device_name [32]
- uint8_t manufacture [64]
- uint8_t model_name [32]
- uint8_t model_number [32]
- uint8_t serial_number [32]
- uint16_t config_methods
- uint16_t primary_dev_category
- uint16_t primary_dev_subcategory
- uint8_t rf_bands
- uint32_t os_version
- uint8_t wps_msg_max_retry
- uint32_t wps_msg_timeout
- uint16_t pin_len
- int(∗ wps_callback)(enum wps_event event, void ∗data, uint16_t len)

7.47.1 Detailed Description

This struct is passed to wps_start(). The user must initialize it with parameters as described inline.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.47.2 Field Documentation

7.47.2.1 uint8_t wps_config::role

Enrollee: 1, Registrar: 2, WiFi Direct mode:4

7.47.2.2 uint8_t wps_config::pin_generator

PIN Generator - Enrollee or Registrar

7.47.2.3 uint8_t wps_config::version

version

7.47.2.4 uint8_t wps_config::version2

version

7.47.2.5 uint8_t wps_config::device_name[32]

Device name

7.47.2.6 uint8_t wps_config::manufacture[64]

Manufacture

7.47.2.7 uint8_t wps_config::model_name[32]

Model name

7.47.2.8 uint8_t wps_config::model_number[32]

Model number

7.47.2.9 uint8_t wps_config::serial_number[32]

Serial number

7.47.2.10 uint16_t wps_config::config_methods

Config methods

7.47.2.11 uint16_t wps_config::primary_dev_category

Primary Device category

7.47.2.12 uint16_t wps_config::primary_dev_subcategory

Primary Device subcategory

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.47.2.13 uint8_t wps_config::rf_bands

RF bands

7.47.2.14 uint32_t wps_config::os_version

OS Version

7.47.2.15 uint8_t wps_config::wps_msg_max_retry

WPS message max retry

7.47.2.16 uint32_t wps_config::wps_msg_timeout

WPS message timeout

7.47.2.17 uint16_t wps_config::pin_len

PIN length

7.47.2.18 int(* wps_config::wps_callback)(enum wps_event event, void *data, uint16_t len)

WPS callback

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

- wps.h

7.48 wslay_frame Struct Reference

```
#include <websockets.h>
```

Data Fields

- uint8_t fin
- uint8_t opcode
- const uint8_t * data
- size_t data_len

7.48.1 Detailed Description

Web Sockets Transfer Frame

7.48.2 Field Documentation

7.48.2.1 uint8_t wslay_frame::fin

Frame Finish. Set to 1 if this is the final fragment of the frame

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

7.48.2.2 uint8_t wslay_frame::opcode

The Opcode of this data

7.48.2.3 const uint8_t* wslay_frame::data

Pointer to data

7.48.2.4 size_t wslay_frame::data_len

Length of the data

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

- websockets.h

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Confidential

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Chapter 8

File Documentation

8.1 app_framework.h File Reference

The Application Framework.

8.1.1 Detailed Description

The application framework (AF) provides a core state machine that is commonly required by applications. It manages two wireless network interfaces, micro-AP and station. The station interface can either be in the unconfigured or the configured mode. The AF is so designed that it enables a wide variety of use cases and pulls in only the code for the used APIs.

8.1.2 Application Framework Features

The AF provides wrappers for other modules that provide an easy to use interface. The AF provides the following features:

a) Micro-AP Mode: State machine for starting and stopping the Micro-AP interface.

b) Station Mode: State machine for starting and stopping the station interface. Typically the station interface requires information about the wireless network to connect to. This information includes parameters like Network name, security, passphrase among other things. The process of configuring this information in the device is called provisioning. AF allows a variety of ways to provision the device. Once the network information is configured, AF maintains this information in Persistent Storage Manager (PSM) so that the information is maintained across power down events.

c) Marvell Provisioning Service: AF provides wrappers for Marvell Provisioning service. This service allows two ways for network provisioning:

1. Micro-AP based Provisioning: Users can connect to the Micro-AP network. The provisioning services provides HTTP handlers that can be used to perform network provisioning. Applications can use a webapp-based wizard to guide the user through the provisioning process.
2. WPS based Provisioning: Users press a button on the Access Point (AP) and a button on this device. The WPS protocol is initiated which takes care of transferring the network credentials to the device.

The Marvell Provisioning Service is optional. Applications may choose to use their own mechanism of provisioning.

d) HTTP Service: AF provides convenience wrappers for the HTTP Web Service. These wrappers implement the following commonly required functionality of the HTTP Web service:

1. Filesystem: A method to optionally attach a filesystem at the '/' location of the Web Server. This is useful when the Web Service has to service files like HTML/JS/JPEG.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

2. /sys/ Handlers: A default set of HTTP handlers is available at the '/sys/' location of the Web Server. These handlers provide basic information/configuration of the device

e) mDNS Service: AF provides convenience wrappers for the mDNS service.

8.1.3 Usage

1. Applications can use the AF by calling the app_framework_start() function. A callback is registered with the call the app_framework_start(). The AF invokes this callback handler on occurrence of various events in the system. The set of events are documented in the app_ctrl_event_t data type. No other functions of the AF should be called before the AF delivers the AF_EVT_INIT_DONE event to the callback handler.

2. Applications can control the Micro-AP interface by making calls to the app_uap_start_with_dhcp() and app_uap_stop() functions.

3. Applications can control the station interface by making calls to the app_sta_start(), app_sta_start_by_network() and app_sta_stop() functions. Once the station interface is started, the AF checks for events like link-loss and deauth that cause a network disconnection. The AF automatically reinitiates connection attempts on occurrence of these events.

4. Applications can use Marvell Provisioning service by calling app_provisioning_start() and app_provisioning_stop() functions. Applications also have an option to use their own provisioning mechanism. If applications do so, they should inform the application framework of the configured network using the app_sta_save_network_and_start() function. This will inform the AF of the network configuration so that the information can be stored in PSM for future use. Applications can request the AF to forget the network configuration by using the app_reset_saved_network() call.

5. Applications can control the Web Server using the app_httpd_start(), app_httpd_with_fs_start() and app_httpd_stop() functions. Applications can control what HTTP handlers to expose by using the functions app_sys_register_upgrade_handler(), app_sys_register_diag_handler() and app_sys_register_perf_handler().

6. Applications can control the mDNS service using the app_mdns_start() and app_mdns_stop() calls.

8.1.4 State Machine

The application framework state diagram is as shown below. The Yellow boxes indicate the various states. The top half is the name of the state, and the bottom half consists of any code that is executed on entering that state. Labels on the transitions indicate when that particular transition is taken.

```
... (skipping 22 lines) ...
  [*] --> AF_UNCONFIGURED : station_configured = 0
  [*] --> AF_CONFIGURED : station_configured = 1
 AF_UNCONFIGURED --> AF_NORMAL_INIT : app_sta_save_network_and_start()
 OR \n app_sta_start_by_network();
Syntax Error?
```

There is an independent P2P state machine as well, as shown below.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.1.5 Function Documentation

8.1.5.1 int app_framework_start (app_event_handler_t *event_handler*)

Start Application Framework

This is the main function that starts the application framework

If you wish to use the application framework, call this from the application's main() function. This function will:

1. Start the application framework thread
2. Initialize the Wireless Connection Manager
3. Initialize the Persistent State Manager (PSM)
4. Read the network configuration state from the psm
5. Informs the application's callback handler with events AF_EVT_INIT_DONE followed by AF_EVT_WLAN_INIT_DONE

Any functions related to the WiFi like app_sta_start() or app_uap_start_with_dhcp() should be called only after the AF_EVT_WLAN_INIT_DONE event. Any other application framework functions should be called only after the AF_EVT_INIT_DONE event.

Parameters

in	*event_handler*	the application's event handling function app_event_handler_t

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.2 int app_configure_uap_ip (char * *ip*)

App Framework configure uAP IP address

This function provides a facility to configure the IP address of the uAP interface if uAP is started using the app_uap_start_with_dhcp() or app_uap_start() function.

Note that this function is optional and needs to be used before calling the app_uap_start_with_dhcp() or app_uap_start() functions, else the default IP will be used.

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*ip*	the IP address string of the form a.b.c.d (Eg. 192.168.2.1)

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.1.5.3 int app_uap_start_with_dhcp (char * *ssid*, char * *wpa2_passphrase*)

App Framework Start micro-AP Network with DHCP Server

This provides the facility to start the micro-AP network.

This starts a default micro-AP network with WPA2 security and start a DHCP Server on this interface. The IP Address of the micro-AP interface is 192.168.10.1/255.255.255.0 This also starts the DHCP service on that network.

If NULL is passed as the passphrase, an Open, unsecured micro-AP network is started.

For a finer grained control on the network configuration use app_uap_start_by_network() instead.

Note

This function should not be called before the AF_EVT_WLAN_INIT_DONE event is received by the application.

Parameters

in	*ssid*	the micro-AP SSID
in	*wpa2_passphrase*	the WPA2 passphrase of the micro-AP. Pass NULL for an Open network.

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.4 int app_uap_start_on_channel_with_dhcp (char * *ssid*, char * *wpa2_passphrase*, int *channel*)

App Framework Start micro-AP Network on channel with DHCP Server

This provides the facility to start the micro-AP network.

This starts a default micro-AP network on pre-configured channel with WPA2 security and start a DHCP Server on this interface. The IP Address of the micro-AP interface is 192.168.10.1/255.255.255.0 This also starts the DHCP service on that network.

If NULL is passed as the passphrase, an Open, unsecured micro-AP network is started.

If 0 is passed as the channel, channel selection is set to auto.

For a finer grained control on the network configuration use app_uap_start_by_network() instead.

Note

This function should not be called before the AF_EVT_WLAN_INIT_DONE event is received by the application.

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*ssid*	the micro-AP SSID
in	*wpa2_passphrase*	the WPA2 passphrase of the micro-AP. Pass NULL for an Open network.
in	*channel*	the micro-AP channel

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.5 int app_uap_start (char * *ssid*, char * *wpa2_passphrase*)

App Framework Start micro-AP Network

This provides the facility to start the micro-AP network.

This starts a default micro-AP network with WPA2 security, and the IP Address of the micro-AP interface is 192.168.10.1/255.255.255.0. Please note that the application should start the DHCP server on its own by calling the dhcp_server_start() function.

If NULL is passed as the passphrase, an Open, unsecured micro-AP network is started

For a finer grained control, like using a different security settings, or different IP Address range, or a particular channel, use app_uap_start_by_network() instead.

Note

This function should not be called before the AF_EVT_WLAN_INIT_DONE event is received by the application.

Parameters

in	*ssid*	the micro-AP SSID
in	*wpa2_passphrase*	the WPA2 passphrase of the micro-AP. Pass NULL for an Open network.

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.6 int app_uap_start_on_channel (char * *ssid*, char * *wpa2_passphrase*, int *channel*)

App Framework Start micro-AP Network

This provides the facility to start the micro-AP network.

This starts a default micro-AP network on pre-configured channel with WPA2 security and start a DHCP Server on this interface. The IP Address of the micro-AP interface is 192.168.10.1/255.255.255.0 This also starts the DHCP service on that network.

This starts a default micro-AP network on pre-configured channel with WPA2 security, and the IP Address of the micro-AP interface is 192.168.10.1/255.255.255.0. Please note that the application should start the DHCP server on its own by calling the dhcp_server_start() function.

If NULL is passed as the passphrase, an Open, unsecured micro-AP network is started If 0 is passed as the channel, channel selection is set to auto.

For a finer grained control, like using a different security settings, or different IP Address range, or a particular channel, use app_uap_start_by_network() instead.

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Note

This function should not be called before the AF_EVT_WLAN_INIT_DONE event is received by the application.

Parameters

in	*ssid*	the micro-AP SSID
in	*wpa2_passphrase*	the WPA2 passphrase of the micro-AP. Pass NULL for an Open network.
in	*channel*	the micro-AP channel

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.7 int app_uap_start_by_network (struct wlan_network * *net*)

App Framework Start micro-AP Network

This function is similar to the app_uap_start() function, but is more configurable since it accepts the entire struct wlan_network as a parameter.

Note

This function should not be called before the AF_EVT_WLAN_INIT_DONE event is received by the application.

Parameters

in	*net*	the micro-AP network details in struct wlan_network format

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.8 int app_uap_stop ( )

App Framework Stop micro-AP Network

This stops the micro-AP network that was started using the app_uap_start() function (or its variants). If app_uap_start_with_dhcp() was used to start the micro-AP network, then this call also stops the DHCP service.

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.9 int app_sta_start ( )

App Framework Start Station Interface

Connect the station interface to the configured network. A configured network should exist in the psm for this call to function properly. If Marvell provisioning is performed then the network configuration is written to psm when provisioning is complete. If Marvell provisioning is not used, the application should have informed the application framework about the configured network using the app_sta_save_network_and_start().

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Note

This function should not be called before the AF_EVT_WLAN_INIT_DONE event is received by the application.

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.10 int app_sta_save_network (struct wlan_network * *net*)

App Framework Save Network

This api saves the Network Configuration in Persistent Storage. The app_sta_start() api can be used further to make a connection to the stored network.

Parameters

in	*net*	pointer to wlan_network object to be saved in Persistent Storage

Returns

WM_SUCCESS on success
-WM_FAIL otherwise

8.1.5.11 int app_sta_start_by_network (struct wlan_network * *net*)

App Framework Start Station Interface with given Network

Connect the station interface to the network provided. This network is not stored to PSM by the application framework.

Note

This function should not be called before the AF_EVT_WLAN_INIT_DONE event is received by the application.

Parameters

in	*net*	network details in the form struct wlan_network, to be used as the target network to connect to

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.12 int app_sta_stop ( )

App Framework Stop Station Interface

This stops the station interface that was started using the app_sta_start() function (or its variants). After this, the application framework goes back to the disconnected state.

Returns

WM_SUCCESS on success
error code otherwise

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.1.5.13 int app_sta_save_network_and_start (struct wlan_network * *net*)

App Framework Configure Network

Configure a network to connect the station interface to, and connect to this configured network.

If Marvell provisioning (using app_provisioning_* calls) is being used, then this function is not required. The provisioning module takes care of configuring the network, that the user has selected, within the application framework.

If any other mechanism is being used for querying the network configuration information from the user, then this call should be used to inform the application framework of the configured network.

The application framework makes a note of this configuration in the psm. This is the WiFi network that app_sta_start() initiates connections to, even across reboots.

Note

This function should not be called before the AF_EVT_WLAN_INIT_DONE event is received by the application.

Parameters

in	*net*	the network to connect the station interface to in the struct wlan_network format. The memory location pointed to by this pointer should be available until the network connection is successfully established.

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.14 int app_reset_saved_network ( )

App Framework Reset Configured Network

This resets the configured network such that the application framework from now onwards comes up in the provisioning/unconfigured mode. This api also erases the configured network from PSM

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.15 int app_provisioning_start (char *prov_mode*)

App Framework Start Marvell Provisioning Service

Start the network provisioning process. If the network provisioning is desired, the application should first start the micro-AP before calling app_provisioning_start().

This will start the provisioning thread that does periodic WiFi scans for the networks in the vicinity. Also if WPS provisioning is desired, its state machine is initialized.

Parameters

in	*prov_mode*	The modes of provisioning that should be started. This could be either PROVISIONING_WLANNW or PROVISIONING_WPS or PROVISIONING_WLANNW | PROVISIONING_WPS

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.16 int app_provisioning_only_start (char *prov_mode*)

App Framework Start Marvell Provisioning Service without

Start the network provisioning process. If the network provisioning is desired, the application should first start the micro-AP before calling app_provisioning_start(). This does not start the standard web services such as the /sys/network, /sys/scan and /sys/scan-config URI

This will start the provisioning thread that does periodic WiFi scans for the networks in the vicinity. Also if WPS provisioning is desired, its state machine is initialized.

Parameters

in	*prov_mode*	The modes of provisioning that should be started. This could be either PROVISIONING_WLANNW or PROVISIONING_WPS or PROVISIONING_WLANNW | PROVISIONING_WPS

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.17 int app_ezconnect_provisioning_start (char * *ssid*, uint8_t * *prov_key*, int *prov_key_len*)

App Framework Start Marvell EZConnect Provisioning Service

Start the network EZConnect provisioning process. If the EZConnect provisioning is desired, then there is no need to start micro-AP. If a prov_key is specified then the network credentials are encrypted using this key thus ensuring the network credentials are transmitted securely over the air.

Note

This provisioning is supported only on SD8782 and SD8801 WiFi chipsets.

A Smart Phone app should be used to send network configuration information to this device.

Parameters

in	*ssid*	Pointer to micro-AP ssid string
in	*prov_key*	Pointer to provisioning key string for encryption. This must be 16-bytes in length.
in	*prov_key_len*	length of provisioning key. This value must be 16-bytes.

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.18 int app_prov_wps_session_start (char * *pin*)

App Framework Start WPS Session

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Instruct the application framework to start a WPS session.

It is mandatory that app_provisioning_start() is called before this function. This call will indicate the provisioning module to start a WPS session with the specified pin or push-button.

Parameters

in	*pin*	In case of push-button based WPS this should be NULL. While for pin-based WPS, it should include a null-terminated string for the WPS PIN.

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.19 void app_provisioning_stop (void)

App Framework Stop Provisioning

This stops the provisioning service and clears any resources allocated to it.

On successful provisioning, this should be called in response to the AF_EVT_PROV_DONE event.

8.1.5.20 void app_provisioning_only_stop ( )

App Framework Stop Provisioning

This stops the provisioning service and clears any resources allocated to it. This API should be specifically used in case if app_provisioning_only_start() is being used to start the provisioning module. This is because the provisioning web-handlers are not available in this case.

On successful provisioning, this should be called in response to the AF_EVT_PROV_DONE event.

8.1.5.21 void app_ezconnect_provisioning_stop (void)

App Framework Stop EZConnect Provisioning

This stops the ezconnect provisioning service and clears any resources allocated to it.

On successful provisioning, this should be called in response to the AF_EVT_PROV_DONE event.

8.1.5.22 void app_ftfs_init (int *fs_ver*, const char * *part_name*, struct fs ** *fs*)

App Framework initializes the filesystem

This function initializes the filesystem that will be used by the HTTP Server for serving files. This also registers the filesystem with the HTTP Server. This API should be called after the call to app_httpd_start().

Parameters

in	*fs_ver*	The FTFS API version that should be mounted. When active-passive (redundant) filesystems are used, this will mount the active filesystem that matches the specified fs_ver.
in	*part_name*	The FTFS partition name to mount.
out	*fs*	A pointer to a pointer of fs structure which stores filesystem handle of filesystem in use (struct fs).

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.1.5.23 int app_httpd_with_fs_start (int *fs_ver,* const char * *part_name,* struct fs ** *fs*)

App Framework Start HTTP Web Service with filesystem

This starts the HTTPD server similar to app_httpd_start(). But in addition to it, this also initializes the filesystem that is specified as FC_COMP_FTFS in the flash layout and mounts it at '/' of the webserver.

Parameters

in	*fs_ver*	The FTFS API version that should be mounted. When active-passive (redundant) filesystems are used, this will mount the active filesystem that matches the specified fs_ver.
in	*part_name*	The FTFS partition name to mount.
out	*fs*	A pointer to a pointer of fs structure which stores filesystem handle of filesystem in use (struct fs).

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.24 int app_httpd_start ( )

App Framework Start HTTP Web Service

Start the HTTP Web Service. This will start the HTTP server which will listen for HTTP connections on port 80 (http) or port 443 (https) as configured in the build configuration. This also starts the standard web services with the /sys URI prefix.

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.25 int app_httpd_only_start ( )

App Framework Start HTTP Web Service

Start the HTTP Web Service. This will start the HTTP server which will listen for HTTP connections on port 80 (http) or port 443 (https) as configured in the build configuration. This does not start the standard web services with the /sys URI prefix.

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.26 int app_sys_register_upgrade_handler ( )

App Framework Register Upgrade Services

This call registers the HTTP upgrade handlers /sys/firmware, /sys/filesystem and /sys/updater that allow upgrades on the system.

Returns

WM_SUCCESS on success
error code otherwise

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.1.5.27 int app_sys_register_diag_handler ( )

App Framework Register Diagnostics Services

This call registers the HTTP diagnostics handlers available under /sys/diag/

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.28 int app_sys_register_perf_handler ( )

App Framework Register Performance Measurement Services

This call registers the HTTP performance measurement API /sys/perf. HTTP Clients can use this API to get a quick sense of the performance of the device. Please note that this provides only an approximation, and specific performance measurement APIs should be used to get an accurate characterization of the performance. A sample way to get a performance assessment is:

for TCP RX: curl -F "filename=@/tmp/testfile" wmdemo-acca.local/sys/perf

for TCP TX: curl wmdemo-acca.local/sys/perf?size=10485760 -o /tmp/ignore.txt

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.29 int app_httpd_stop ( )

App Framework Stop HTTP Web Service

Stop the HTTP Web Service

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.30 void app_mdns_start (char * *hostname*)

App Framework Start mDNS Service

The mDNS service is used to announce services over the network. mDNS also exposes a hostname that can be resolved using mDNS. For further details about mDNS, please refer to mDNS-SD Responder/Querier .

Services can be associated with a wireless network interface using app_mdns_add_service() or app_mdns_add_service_arr() and can be disassociate using app_mdns_remove_service() or app_mdns_remove_service_arr() or app_mdns_remove_service_all().

Parameters

in	*hostname*	mDNS hostname

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.1.5.31 void app_mdns_stop (void)

App Framework Stop mDNS Service

8.1.5.32 void app_mdns_refresh_hostname (char * *hostname*)

App Framework Refresh mDNS hostname

This refreshes the hostname that is being used by the mDNS service.

Parameters

in	*hostname*	new mDNS hostname

8.1.5.33 int app_mdns_add_service (struct **mdns_service** * *srv*, void * *iface*)

App Framework Add single mDNS service instance

This call is used to associate a service on a network interface. This call announces the service on a given interface and adds it to a list of announced services.

Note

This call is used to add a single service. Please use the app_mdns_add_service_arr() call to add more than one services.

Parameters

in	*srv*	pointer to global service structure (mdns_service)
in	*iface*	pointer to interface handler as retrieved using net_get_uap_handle() or net_get_sta_handle()

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.34 int app_mdns_add_service_arr (struct **mdns_service** * *services[]*, void * *iface*)

App Framework Add multiple mDNS service instances

This call is used to associate a set of services on a network interface. This call announces given set of services on a given interface and adds them to a list of announced services.

Parameters

in	*services*	array of pointers to global service structure (mdns_service)
in	*iface*	pointer to interface handler as retrieved using net_get_uap_handle() or net_get_sta_handle()

Returns

WM_SUCCESS on success
error code otherwise

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.1.5.35 int app_mdns_remove_service (struct mdns_service * *service*, void * *iface*)

App Framework Remove single mDNS service instance

This call is used to disassociate a set of services from a network interface. This call remove service from list of announced services and sends BYE packet for the service on a given interface.

Note

This call is used to remove a single service. Please use the app_mdns_remove_service_arr() call to remove more than one services.

Parameters

in	*service*	pointer to global service structure (mdns_service)
in	*iface*	pointer to interface handler as retrieved using net_get_uap_handle() or net_get_sta_handle()

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.36 int app_mdns_remove_service_arr (struct mdns_service * *services[]*, void * *iface*)

App Framework Remove multiple mDNS service instances

This call is used to disassociate a service from a network interface. This call remove services from list of announced services and sends BYE packet for set of services on a given interface.

Parameters

in	*services*	array of pointers to global service structures (mdns_service)
in	*iface*	pointer to interface handler as retrieved using net_get_uap_handle() or net_get_sta_handle()

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.37 int app_mdns_remove_service_all (void * *iface*)

App Framework Remove all mDNS services from Interface

This clears any services that were associated with a network interface.

Parameters

in	*iface*	pointer to interface handler as retrieved using net_get_uap_handle() or net_get_sta_handle()

Returns

On success 0 is returned. Or error -1 is returned.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.1.5.38 void app_mdns_iface_state_change (void * *iface,* enum iface_state *state*)

App Framework mDNS Service Announcement Control

This controls the mDNS service announcement on the network interface. The state of the interface can be configured as UP, DOWN or REANNOUNCE. The mDNS/ module refers to the value of the state to appropriately announce, unannounce or reannounce the services.

Generally, the application can use UP state to announce associated services on a network interface. Services can be associated with an interface using app_mdns_add_service() or app_mdns_add_service_arr() function calls. UP will announce associated services only when interface is in running state.

Application can use DOWN state to send bye packet on mDNS group so that the mDNS clients can remove service entries from their cache.

In events like link-lost or dhcp renew, network interface is down for some time during which it gives a chance to other devices to announce services with same name. Also in some cases application may change the hostname or service name. After such events, application should repeat the initial probing sequence. This can be achieved by calling this function with DOWN state, followed by calling it again with UP state. This would help to resolve service name conflicts if introduced by other devices. Also the previous service entry in mDNS clients will be removed and new updated entry will be added.

If application need to change TXT records, it should call mdns_set_txt_rec() to set updated TXT records followed by call to this function with the state parameter equal to REANNOUNCE. This will re-announce the mDNS service with the updated TXT records.

Parameters

in	*iface*	pointer to interface handler as retrieved using net_get_uap_handle() or net_get_sta_handle()
in	*state*	valid interface state from iface_state

8.1.5.39 int app_init_device_name (char * *dev_name*)

Initialize Device name in psm

This function sets a default device name in the PSM. The application framework does not use the device name in anyway by itself. Applications can retrieve and use this for their purposes.

Parameters

in	*dev_name*	A NULL-terminated string that stores the default device name.

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.40 int app_get_device_name (char * *dev_name,* int *len*)

Get current device name

This function retrieves the device name that is stored in the psm.

Parameters

out	*dev_name*	Pointer to a buffer to store the device name
in	*len*	The length of the buffer pointed to by dev_name above

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.41 int app_set_device_name (char * *dev_name*)

Set device name

This function updates the device name stored in the PSM. The application framework does not use the device name in anyway by itself. Applications can retrieve and use this for their purposes.

Parameters

in	*dev_name*	A NULL-terminated string that stores the default device name.

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.42 void app_sys_get_uuid (char * *output*, int *len*)

App Framework GET UUID (Universal Unique Identifier) This function is used to get the UUID of the system in a string format.

Parameters

in	*output*	A character buffer which will be populated with the UUID.
in	*len*	Length of the buffer.

8.1.5.43 int app_network_ip_get (char * *ipaddr*)

Query the device's IP Address

This function retrieves the device's IP Address in a NULL-terminated string format.

Parameters

out	*ipaddr*	Pointer to a buffer to write the NULL-terminated IP Address to. The buffer should be at least 16 bytes in size.

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.44 int app_reboot (wm_reboot_reason_t *reason*)

Soft reboot the System

Parameters

in	*reason*	The reason of type wm_reboot_reason_t for the reboot.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

This function does not return

8.1.5.45 int app_ctrl_notify_application_event (app_ctrl_event_t *event*, void * *data*)

Application Framework notify application event

This function allows the applications to generate their own application specific events. The event will pass through the app_framework's event handling mechanism and then the appropriate application registered event callback will be triggered.

This avoids the need of creating one more thread context and enables the application to do some things in the context of the app_framework's thread. This simplifies asynchronous event handling for the applications.

The application needs to define its own events. However, care must be taken that the events do not clash with the application framework's events. So, it would be best to start the enumeration of the application specific events from APP_CTRL_EVT_MAX + 1.

Eg. typedef enum { APP_EVT_APP1 = APP_CTRL_EVT_MAX + 1, APP_EVT_APP2, } app_event_t;

Parameters

in	*event*	The application specific event value (app_ctrl_event_t)
in	*data*	Data if any that the event needs to pass (can be NULL)

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.46 int app_sys_http_update_all (jobj_t * *obj*, short * *fs_done*, short * *fw_done*, short * *wififw_done*)

Upgrade the filesystem, firmware and/or wifi-firmware

This function is used to upgrade filesystem, firmware and/or wifi-firmware provided the input JSON contains appropriate values for elements fs_url, fw_url, wififw_url.

Parameters

in	*obj*	JSON object containing the JSON data for upgrades
out	*fs_done*	Parameter indicating if filesystem upgrade was successful
out	*fw_done*	Parameter indicating if firmware upgrade was successful
out	*wififw_done*	Parameter indicating if wifi firmware upgrade was successful

Returns

WM_SUCCESS on success
-WM_FAIL otherwise

8.1.5.47 int app_p2p_start (app_p2p_role_t *role*)

Start P2P module

This function starts the P2P module. If the operations succeeds, the AF_EVT_P2P_STARTED event is generated. Else, the AF_EVT_P2P_FAILED event is generated.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*role*	can be ROLE_AUTO_GO or ROLE_UNDECIDED only

Returns

WM_SUCCESS on success. Actual result is notified through the events AF_EVT_P2P_STARTED or AF_EVT_P2P_FAILED

8.1.5.48 int app_p2p_stop ( )

Stop the P2P Module

This function stops the P2P module. The AF_EVT_P2P_FINISHED event is generated on success.

Returns

WM_SUCCESS on success. Actual status is notified through the event AF_EVT_P2P_FINISHED

8.1.5.49 int app_p2p_session_start (char * *pin*, struct **p2p_scan_result** * *remote*)

Start a P2P session

This function initiates a connection with the node specified in argument. In AUTO_GO mode, this function just means that the device is ready to accept connections.

Parameters

in	*pin*	The pin to be used for the session. A NULL value means that it is a push button session. Currently, only push button is supported.
in	*remote*	A pointer to struct p2p_scan_result. A session with the node pointed to by this is pointer is started.

Returns

WM_SUCCESS on success. Further results are notified by various P2P events.

8.1.5.50 int app_p2p_disconnect ( )

Disconnect from a P2P connection

This function disconnects a connected P2P session.

Returns

WM_SUCCESS on success.
-WM_FAIL on failure.

8.1.5.51 struct **p2p_scan_result** * app_p2p_scan (int * *no*)

Get P2P scan results.

This function returns a pointer to a list if P2P nodes. The P2P module should be started prior to using this function.

Parameters

out	*no*	Number of P2P nodes in the scan list.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

A pointer to a struct p2p_scan_result array which holds a list of scanned P2P nodes.

8.1.5.52 int app_p2p_ip_get (char * *ipaddr*)

Query the P2P interface's IP address

This function retrieves the P2P interface's IP Address in a NULL-terminated string format.

Parameters

out	*ipaddr*	Pointer to a buffer to write the NULL-terminated IP Address to. The buffer should be at least 16 bytes in size.

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.53 const char* app_ctrl_event_name (app_ctrl_event_t *event*)

Convert App Framework event id to event string

This is useful for debugging.

Parameters

in	*event*	App Framework event ID

Returns

String name corresponding to the App Framework event ID

8.1.5.54 int app_psm_init (void)

Initialize the PSM subsystem

This initialization is normally done by the application framework when it is started using app_framework_start However, some applications may need PSM even before starting application framework. Such applications can use this API.

Returns

WM_SUCCESS on success
error code otherwise

8.1.5.55 void app_get_connection_status (app_conn_status_t * *state*, app_conn_failure_reason_t * *reason*, int * *attempts*)

Get the WLAN station interface connection status

This function retrieves the WLAN connection state, failure reason in case of connection failure and no. of attempts of connection made.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

out	*state*	is a pointer of type app_conn_status_t which is one of CONN_STATE_NOTCONNECTED, CONN_STATE_CONNECTING or CONN_STATE_CONNECTED
out	*reason*	is a pointer of type app_conn_failure_reason_t specifying the failure reason. -1 in case of no failure.

Note

reason parameter is valid only in CONN_STATE_CONNECTING state

Parameters

out	*attempts*	specifying the number of connection attempts made to a network

8.1.5.56 int app_ctrl_getmode ( )

Get the current provisioning status of the device.

This function returns if the device is provisioned or not.

Returns

1 if the device is provisioned 0 if not provisioned

8.1.6 Macro Documentation

8.1.6.1 #define APP_NETWORK_NOT_PROVISIONED 0

Station configuration not found

8.1.6.2 #define APP_NETWORK_PROVISIONED 1

Station configuration found

8.1.6.3 #define APP_NETWORK_CONF_INVALID 3

Station configuration is invalid

8.1.7 Typedef Documentation

8.1.7.1 typedef int(∗ app_event_handler_t)(int event, void ∗data)

Application Framework Event Handler

This is the event handler that applications register with the application framework for handling any events. This function is executed in the context of the application framework's thread.

8.1.7.2 typedef struct app_p2p_role_data app_p2p_role_data_t

This structure is passed along with the AF_EVT_P2P_STARTED and the AF_EVT_P2P_ROLE_NEGOTIATED events to indicate the current role. Refer app_p2p_role_t for details.

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.1.8 Enumeration Type Documentation

8.1.8.1 enum wm_af_errno

Application Framework Error Codes

Enumerator

WM_E_AF_NW_RD Error while reading network configuration data

WM_E_AF_NW_WR Error while writing network configuration data

WM_E_AF_REB_TMR_WR Error in updating reboot timer settings

WM_E_AF_NW_ADD Network add failed

WM_E_AF_NW_DEL Network remove failed

WM_E_AF_NW_CONN Network connect failed

WM_E_AF_WLCMGR_FAIL Failed to start wlcmgr

WM_E_AF_PSM_INIT PSM Initialization failed

WM_E_AF_WD_INIT Watchdog Initialization failed

8.1.8.2 enum app_conn_status_t

App-framework connection status of the station interface

Enumerator

CONN_STATE_NOTCONNECTED In case station interface is not connected

CONN_STATE_CONNECTING In case if station inteface is in connecting state

CONN_STATE_CONNECTED In case if station inteface is in connected state

8.1.8.3 enum app_conn_failure_reason_t

App-framework failure reason codes

Enumerator

NETWORK_NOT_FOUND In case AP-is down or network unavailable

AUTH_FAILED In case of authentication failure

DHCP_FAILED In case of failure in assigning IP address

P2P_CONNECT_FAILED In case of a P2P failure after starting a session

8.1.8.4 enum app_ctrl_event_t

Events processed by application controller.

These are the events that are processed by the application controller. The application is notified of these events so that it can show appropriate indicators to the user or take appropriate actions.

Enumerator

AF_EVT_INIT_DONE The WM is up and all basic modules have been initialized, perform any notification-cleanups/inits if required. The structure app_init_state is passed as data along with this event.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

AF_EVT_WLAN_INIT_DONE The WLAN subsystem has been initialized. The data that is passed along with this is either APP_NETWORK_NOT_PROVISIONED or APP_NETWORK_PROVISIONED. Applications can decide whether to start the micro-AP or station interface using app_uap_start_with_dhcp() or app_sta_start() calls respectively.

Application can call wlan_enable_11d(), wlan_set_country() APIs after reception of this event to enable 11d support or set sub band information for particular country.

AF_EVT_NORMAL_INIT We are now starting in normal mode

AF_EVT_NORMAL_CONNECTING Attempting to connect in normal mode

AF_EVT_NORMAL_CONNECTED We have successfully connected to the network. Note that we can get here after normal bootup or after successful provisioning.

AF_EVT_NORMAL_CONNECT_FAILED One connection attempt to the network has failed. Note that we can get here after normal bootup or after an unsuccessful provisioning. A reference to app_conn_failure_reason_t object gets propagated which specifies the failure reason.

app_event_handler_t function should return WM_SUCCESS if it wishes to attempt reconnection upon receiving this event. Returning non-zero value shall cause the application framework to stop reconnection attempts and it moves the station into disconnected state. In this state app_sta_start() (or its variants) can be used to attempt reconnection.

AF_EVT_NORMAL_LINK_LOST We were connected to the network, but the link was lost intermittently.

app_event_handler_t function should return WM_SUCCESS if it wishes to attempt reconnection upon receiving this event. Returning non-zero value shall cause the application framework to stop reconnection attempts and it moves the station into disconnected state. In this state app_sta_start() (or its variants) can be used to attempt reconnection.

AF_EVT_NORMAL_CHAN_SWITCH We were connected to the network, but the channel switch announcement is received

AF_EVT_NORMAL_WPS_DISCONNECT We were connected to the WPS network, the WPS module asked to be disconnected. This event is delivered regardless of whether WPS is successful or not.

AF_EVT_NORMAL_USER_DISCONNECT We were connected to the network, the user (application) asked to be disconnected.

AF_EVT_NW_SET Network settings have been modified and updated. Attempt to connect to that network now.

AF_EVT_NORMAL_DHCP_RENEW DHCP lease was expired and hence the DHCP address was renewed.

AF_EVT_PROV_DONE Provisioning is complete, attempting to connect to network. Include some progress indicators while the connection attempt either succeeds or fails.

AF_EVT_PROV_WPS_SSID_SELECT_REQ An SSID with an active WPS session is found. WPS negotiation will now be started with this AP.

AF_EVT_PROV_WPS_START The WPS session started.

AF_EVT_PROV_WPS_REQ_TIMEOUT The WPS session failed due to timeout

AF_EVT_PROV_WPS_SUCCESSFUL The WPS session has completed successfully

AF_EVT_PROV_WPS_UNSUCCESSFUL The WPS session has completed unsuccessfully

AF_EVT_NORMAL_PRE_RESET_PROV This indicates to the user the application framework has received the reset-to-provisioning request in a valid state. The station interface is about to be taken down.

AF_EVT_NORMAL_RESET_PROV This indicates to the user the application framework has received the reset-to-provisioning request and the station interface has been taken down.

AF_EVT_UAP_STARTED This indicates to the user that the microAP interface has been started.

AF_EVT_UAP_CLIENT_ASSOC This indicates to the user that a wireless client is associated with the micro-AP interface.

AF_EVT_UAP_CLIENT_DISSOC This indicates to the user that a wireless client is dissociated from the microAP interface.

AF_EVT_UAP_STOPPED This indicates to the user that the microAP interface has been stopped.

AF_EVT_PS_ENTER This indicates to the user that WIFI has entered Power save mode.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

AF_EVT_PS_EXIT This indicates to the user that WIFI has exited Power save mode.

AF_EVT_PROV_CLIENT_DONE This event is sent when an indication is received from the http provisioning client that it has concluded that the provisioning of the device is successful.

AF_EVT_P2P_STARTED This event indicates that the P2P module has started. The data field is of type app_p2p_role_data_t and tells if the device has started in the Auto GO role or undecided role. If the role is undecided, it will be decided when a session is initiated and the negotiation has completed. The role in that case is notified in the data of the AF_EVT_P2P_ROLE_NEGOTIATED event. The data field also gives the name of the device.

AF_EVT_P2P_SESSION_STARTED This indicates that the P2P session has started

AF_EVT_P2P_ROLE_NEGOTIATED This indicates that the P2P device's role has been negotiated. It can be either client or GO (Group Owner). This information is available in the data field which is of type app_p2p_role_data_t. The status of the connection is indicated by other events like AF_EVT_NORMAL-_CONNECTED, AF_EVT_NORMAL_CONNECT_FAILED, etc.

AF_EVT_P2P_FAILED This indicates that the P2P thread failed to start.

AF_EVT_P2P_FINISHED This indicates that the P2P module has been shutdown successfully.

APP_CTRL_EVT_INTERNAL Events from APP_CTRL_EVT_INTERNAL to APP_CTRL_EVT_MAX are for internal use and should not be used by applications.

8.1.8.5 enum app_p2p_role_t

Role of the P2P application

Enumerator

ROLE_UNDECIDED The P2P application has started, but the role is undecided. The actual role will be decided after negotiation.

ROLE_AUTO_GO The P2P application has started in the auto GO mode. This means that the device will be the group owner (GO).

ROLE_CLIENT The P2P device has taken up the role of a client after the negotiation.

ROLE_GO The P2P device has taken up the role of a Group Owner (GO) after the negotiation.

8.1.9 Variable Documentation

8.1.9.1 psm_hnd_t app_psm_hnd

Applications wanting to use PSM should use this handle. This handle is associated with PSM after app framework is initialized

8.2 block_alloc.h File Reference

Slab/Block Allocator.

8.2.1 Detailed Description

Copyright 2008-2015, Marvell International Ltd. All Rights Reserved.

8.2.2 Introduction

There are some packages which perform a large number of fixed-size allocations. But, the maximum number of allocations that are active at any given point in time is very less. It would be desirable to have a slab allocator which

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

can reduce the fragmentation that may potentially be caused by the use of such large number of allocations. The slab/block allocator aims at allocating fixed size blocks as per the requirement of the user, subject to the total size specified by the user when initializing the pool of that specific block size.

Example: If a user wishes to create a pool with restricted access and having 10 blocks of size say 20 bytes, then he can first initialize the pool.

For that he can use:

- struct pool_info ∗pool_ptr;
- char buffer[20 ∗ 10 + 20];
- char ∗block_ptr;

Steps:

- Initialize Pool:

pool_ptr = ba_pool_create(20, buffer, sizeof(buffer));

Further, he may allocate or deallocate blocks using the pool pointer.

- Allocation

block_ptr = ba_block_allocate(pool_ptr);

- Deallocation: ba_block_release(pool_ptr, block_ptr); Further, if a user wishes to delete the memory pool, he may do so by:
- Delete pool

ba_pool_delete(pool_ptr);

Note

A pool can be deleted only if all the blocks in it are free.

8.2.3 Function Documentation

8.2.3.1 struct pool_info∗ ba_pool_create (unsigned long *block_size,* void ∗ *buf,* unsigned long *pool_size*)

Create a memory pool of specified block size.

This function creates and initializes a pool with blocks of specified size.

Parameters

block_size	The required block size for each block.
buf	A pointer to the caller allocated buffer. This buffer needs to be 4 byte aligned. All allocations are satisfied from this pool.
pool_size	Total size of the buffer.

Returns

pointer to the created pool if successful.
NULL if unsuccessful.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.2.3.2 int ba_pool_delete (struct pool_info * *b*)

Delete already initialized memory pool.

This function deletes an initialized memory pool.

Parameters

b	A pointer of structure pool_info to be deleted.

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

8.2.3.3 void* ba_block_allocate (struct pool_info * *b*)

Allocate a block from the memory pool.

This function allocates a block of the specified size.

Parameters

b	An already initialized pointer of structure pool_info corresponding to the required size.

Returns

the pointer if successful.
NULL if unsuccessful.

8.2.3.4 int ba_block_release (struct pool_info * *b*, void * *alloc*)

Free a block from the memory pool.

This function frees a block of the specified size.

Parameters

b	An already initialized pointer of structure pool_info whose block is to be freed.
alloc	The actual pointer to be deallocated.

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

8.2.3.5 int ba_cli_init (void)

Register the CLI commands for Block/Slab Allocator.

Returns

WM_SUCCESS on success
-WM_FAIL if unsuccessful.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.3 board.h File Reference

Board Specific APIs.

8.3.1 Detailed Description

The board specific APIs are provided to help with porting the SDK onto new boards. These APIs are board specific and they need to be modified/implemented according to the actual board specifications.

Please verify that board.c file found under wmsdk_bundle-x.y.z/wmsdk/src/boards/board_file.c reflects the correct hardware configuration for the project.

8.3.2 Function Documentation

8.3.2.1 int board_main_xtal ( )

Frequency of Main Crystal

The MC200/MW300 processor can be clocked through either of the following 3 sources:

1) Externally connected crystal.

2) Externally connected oscillator.

3) Internal RC oscillator.

If the board has an external crystal for main cpu clocking, mention it's frequency here.

Returns

Frequency of external crystal if it is present, -WM_FAIL otherwise.

8.3.2.2 int board_main_osc ( )

Frequency of Main Oscillator

The MC200/MW300 processor can be clocked through either of the following 3 sources:

1) Externally connected crystal.

2) Externally connected oscillator.

3) Internal RC oscillator.

If the board has an external oscillator for main cpu clocking, mention it's frequency here.

Returns

Frequency of external oscillator if it is present, -WM_FAIL otherwise.

8.3.2.3 int board_cpu_freq ( )

Frequency at which CPU should run

Mention the frequency at which the CPU should run here. If this frequency is greater than the source clock, internal PLL will be configured.

NOTE: Valid frequencies depend upon the source clock. Please refer to the datasheet for further details.

Returns

Frequency at which CPU should operate.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.3.2.4 int board_32k_xtal ( )

32 KHz crystal

If 32 KHz crystal is present on the board and is functional, return true here.

Returns

true if 32 KHz crystal is present and functional, false otherwise.

8.3.2.5 int board_32k_osc ( )

32 KHz Oscillator

If 32 KHz oscillator is present on the board and is functional, return true here.

Returns

true if 32 KHz oscillator is present and functional, false otherwise.

8.3.2.6 int board_rc32k_calib ( )

32 KHz Internal RC Calibration

If 32 KHz crystal or oscillator is not present. The internal rc32k can be calibrated using GPT. NOTE1: On MW300 this needs the GPIO24 and GPIO25 to be connected. NOTE2: On MC200 this will work only if 32MHz crystal is present on the board.

Returns

true if RC32K should be calibrated using GPT

8.3.2.7 int board_card_detect ( )

Detect the SDIO card

If there is a mechanism to detect the SDIO card on the board, use it here, or else, always return the default condition.

Returns

TRUE if card is present, FALSE otherwise

8.3.2.8 void board_sdio_pdn ( )

Powerdown the SDIO card

If there is a GPIO connected to PDn of the SDIO card, ASSERT it in this function.

Leave blank if not connected.

8.3.2.9 void board_sdio_pwr ( )

Powerup the SDIO card

If there is a GPIO connected to PDn of the SDIO card, DE-ASSERT it in this function.

Leave blank if not connected.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.3.2.10 void board_sdio_reset ( )

Reset the SDIO card

If there is a GPIO connected to RST of the SDIO card, ASSERT it for some duration and DE-ASSERT it again.

Leave blank if not connected.

8.3.2.11 int board_sdio_pdn_support ( )

Check if PDn pin of the SDIO card can be controlled by HOST MCU

Returns

TRUE if it can be controlled, FALSE otherwise

8.3.2.12 int board_button_3 ( )

Boot Override Pushbutton

Returns

GPIO pin number connected to the pushbutton to be used for Boot override mode functionality or -WM_FAIL if such pushbutton is not implemented on the board.

8.3.2.13 int board_button_pressed (int *pin*)

Push Button Pressed

Depending upon how the push button is connected, corresponding GPIO needs to be checked for either set or reset.

If no pushbuttons are implemented on the board, always return FALSE (i.e. button not pressed).

Parameters

in	*pin*	GPIO pin to be used

Returns

TRUE if pressed, FALSE otherwise

8.3.2.14 void board_gpio_power_on ( )

Power On GPIO settings

Do board specific power GPIO settings here.

8.3.2.15 void board_uart_pin_config (int *id*)

UART pin config

Select which pins will be used for different UART ports

Leave Blank if UART is not implemented on the board.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*id*	UART port number

8.3.2.16 void board_i2c_pin_config (int *id*)

I2C pin config

Select which pins will be used for different I2C ports

Leave Blank if I2C is not implemented on the board.

Parameters

in	*id*	I2C port number

8.3.2.17 void board_ssp_pin_config (int *id*, bool *cs*)

SSP pin config

Select which pins will be used for different SSP ports

Leave Blank if SSP is not implemented on the board.

Parameters

in	*id*	SSP port number
in	*cs*	If internal SSPx_FRM (cs=1) should be activated or not.

8.3.2.18 void board_usb_pin_config ( )

Select pin configuration for USB

8.3.2.19 void board_sdio_pin_config ( )

SDIO pin config

Select which pins will be used for SDIO controller

Leave Blank if SDIO is not connected on the board.

8.3.2.20 output_gpio_cfg_t board_led_1 ( )

Pin number and configuration of GPIO connected to LED 1

Note

If LED is not present on the board gpio member in output_gpio_cfg_t is set to -1 by this API.

Returns

Output GPIO Configuration type output_gpio_cfg_t

8.3.2.21 output_gpio_cfg_t board_led_2 ( )

Pin number and configuration of GPIO connected to LED 2

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Note

If LED is not present on the board gpio member in output_gpio_cfg_t is set to -1 by this API.

Returns

Output GPIO Configuration type output_gpio_cfg_t

8.3.2.22 output_gpio_cfg_t board_led_3 ( )

Pin number and configuration of GPIO connected to LED 3

Note

If LED is not present on the board gpio member in output_gpio_cfg_t is set to -1 by this API.

Returns

Output GPIO Configuration type output_gpio_cfg_t

8.3.2.23 output_gpio_cfg_t board_led_4 ( )

Pin number and configuration of GPIO connected to LED 4

Note

If LED is not present on the board gpio member in output_gpio_cfg_t is set to -1 by this API.

Returns

Output GPIO Configuration type output_gpio_cfg_t

8.3.2.24 void board_led_on (int *pin*)

Turn LED on

Depending upon how the LED is connected, corresponding GPIO needs to be either set or reset.

Leave Blank if LEDs are not implemented on the board.

Parameters

in	*pin*	GPIO pin to be used

8.3.2.25 void board_led_off (int *pin*)

Turn LED off

Depending upon how the LED is connected, corresponding GPIO needs to be either set or reset.

Leave Blank if LEDs are not implemented on the board.

Parameters

in	*pin*	GPIO pin to be used

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.3.2.26 int board_button_1 ( )

WPS Pushbutton

Returns

GPIO connected to the pushbutton to be used for WPS functionality or -WM_FAIL if such pushbutton is not implemented on the board.

8.3.2.27 int board_button_2 ( )

Reset to Provisioning Pushbutton

Returns

GPIO connected to the pushbutton to be used for Reset to Provisioning mode functionality or -WM_FAIL if such pushbutton is not implemented on the board.

8.3.2.28 void board_lcd_backlight_off ( )

Turn LCD backlight off

Depending upon how the LCD backlight is controlled corresponding GPIO is to be toggled. Leave Blank if LCD backlight control is not present on the board.

8.3.2.29 void board_lcd_backlight_on ( )

Turn LCD backlight on

Depending upon how the LCD backlight is controlled corresponding GPIO is to be toggled. Leave Blank if LCD backlight control is not present on the board.

8.3.2.30 void board_lcd_reset ( )

Reset LCD

Reset the on-board LCD. Leave Blank if LCD is not present on the board.

8.3.2.31 int board_wakeup0_functional ( )

This function indicates whether wakeup 0 pin is functional or not on this board

Returns

True or False

8.3.2.32 int board_wakeup1_functional ( )

This function indicates whether wakeup 1 pin is functional or not on this board

Returns

True or False

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.3.2.33 int board_wakeup0_wifi ( )

This function indicates whether wakeup 0 pin is connected to the WiFi chip's host wakeup pin.

Returns

True or False

8.3.2.34 int board_wakeup1_wifi ( )

This function indicates whether wakeup 1 pin is connected to the WiFi chip's host wakeup pin.

Returns

True or False

8.4 cli.h File Reference

CLI module.

8.4.1 Detailed Description

8.4.2 Usage

The CLI module lets you register commands with the CLI interface. Modules that wish to register the commands should initialize the struct cli_command structure and pass this to cli_register_command(). These commands will then be available on the CLI.

8.4.3 Function Documentation

8.4.3.1 int cli_register_command (const struct cli_command * *command*)

Register a CLI command

This function registers a command with the command-line interface.

Parameters

in	*command*	The structure to register one CLI command

Returns

0 on success
1 on failure

8.4.3.2 int cli_unregister_command (const struct cli_command * *command*)

Unregister a CLI command

This function unregisters a command from the command-line interface.

Parameters

in	*command*	The structure to unregister one CLI command

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

0 on success
1 on failure

8.4.3.3 int cli_init (void)

Initialize the CLI module

Returns

WM_SUCCESS on success
error code otherwise.

Note

This function is called by the wm_core_init function. Applications need not explicitly call this function if they have already called wm_core_init().

8.4.3.4 int cli_stop (void)

Stop the CLI thread and carry out the cleanup

Returns

WM_SUCCESS on success
error code otherwise.

8.4.3.5 int cli_register_commands (const struct cli_command * *commands*, int *num_commands*)

Register a batch of CLI commands

Often, a module will want to register several commands.

Parameters

in	*commands*	Pointer to an array of commands.
in	*num_commands*	Number of commands in the array.

Returns

0 on success
1 on failure

8.4.3.6 int cli_unregister_commands (const struct cli_command * *commands*, int *num_commands*)

Unregister a batch of CLI commands

Parameters

in	*commands*	Pointer to an array of commands.
in	*num_commands*	Number of commands in the array.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

0 on success
1 on failure

8.5 dhcp-server.h File Reference

DHCP server.

8.5.1 Detailed Description

The DHCP Server is required in the provisioning mode of the application to assign IP Address to Wireless Clients that connect to the WM.

8.5.2 Function Documentation

8.5.2.1 int dhcpd_cli_init (void)

Register DHCP server commands

This function registers the CLI(s) for the DHCP server. If the test build option is enabled this also registers CLI(s) for the testing of the DHCP Server.

Returns

-WM_E_DHCPD_REGISTER_CMDS if cli init operation failed.
WM_SUCCESS if cli init operation success.

8.5.2.2 int dhcp_server_start (void * *intrfc_handle*)

Start DHCP server

This starts the DHCP server on the interface specified. Typically DHCP server should be running on the micro-AP interface but it can also run on wifi direct interface if configured as group owner. Use net_get_uap_handle() to get micro-AP interface handle and net_get_wfd_handle() to get wifi-direct interface handle.

Parameters

in	*intrfc_handle*	The interface handle on which DHCP server will start

Returns

WM_SUCCESS on success or error code

8.5.2.3 void dhcp_server_stop (void)

Stop DHCP server

8.5.2.4 int dhcp_server_lease_timeout (uint32_t *val*)

Configure the DHCP dynamic IP lease time

This API configures the dynamic IP lease time, which should be invoked before DHCP server initialization

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*val*	Number of seconds, use (60U∗60U∗number of hours) for clarity. Max value is (60U∗60U∗24U∗49700U)

Returns

Error status code

8.5.2.5 int dhcp_get_ip_from_mac (uint8_t ∗ *client_mac*, uint32_t ∗ *client_ip*)

Get IP address corresponding to MAC address from dhcpd ip-mac mapping

This API returns IP address mapping to the MAC address present in cache. IP-MAC cache stores MAC to IP mapping of previously or currently connected clients.

Parameters

in	*client_mac*	Pointer to a six byte array containing the MAC address of the client
out	*client_ip*	Pointer to IP address of the client

Returns

WM_SUCCESS on success or -WM_FAIL.

8.5.3 Enumeration Type Documentation

8.5.3.1 enum wm_dhcpd_errno

DHCPD Error Codes

Enumerator

WM_E_DHCPD_SERVER_RUNNING Dhcp server is already running

WM_E_DHCPD_THREAD_CREATE Failed to create dhcp thread

WM_E_DHCPD_MUTEX_CREATE Failed to create dhcp mutex

WM_E_DHCPD_REGISTER_CMDS Failed to register dhcp commands

WM_E_DHCPD_RESP_SEND Failed to send dhcp response

WM_E_DHCPD_DNS_IGNORE Ignore as msg is not a valid dns query

WM_E_DHCPD_BUFFER_FULL Buffer overflow occurred

WM_E_DHCPD_INVALID_INPUT The input message is NULL or has incorrect length

WM_E_DHCPD_INVALID_OPCODE Invalid opcode in the dhcp message

WM_E_DHCPD_INCORRECT_HEADER Invalid header type or incorrect header length

WM_E_DHCPD_SPOOF_NAME Spoof length is either NULL or it exceeds max length

WM_E_DHCPD_BCAST_ADDR Failed to get broadcast address

WM_E_DHCPD_IP_ADDR Failed to look up requested IP address from the interface

WM_E_DHCPD_NETMASK Failed to look up requested netmask from the interface

WM_E_DHCPD_SOCKET Failed to create the socket

WM_E_DHCPD_ARP_SEND Failed to send Gratuitous ARP

WM_E_DHCPD_IOCTL_CALL Error in ioctl call

WM_E_DHCPD_INIT Failed to init dhcp server

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.6 ezxml.h File Reference

ezXML - Easy XML Parsing C Library

8.6.1 Detailed Description

ezXML is a C library for parsing XML documents. It's easy to use. It's ideal for parsing XML configuration files or REST web service responses. It's also fast and lightweight (less than 20k compiled).

8.6.2 Usage

Given the following example XML string stored in *xml_buf*:

```
<?xml version="1.0"?>
<formula1>
  <team name="McLaren">
      <driver>
            <name>Kimi Raikkonen</name>
            <points>112</points>
      </driver>
      <driver>
            <name>Juan Pablo Montoya</name>
            <points>60</points>
      </driver>
  </team>
</formula1>
```

This code snippet prints out a list of drivers, which team they drive for, and how many championship points they have:

```
ezxml_t f1 = ezxml_parse_str(xml_buf, strlen(xml_buf));
const char *teamname, *team, *driver;

for (team = ezxml_child(f1, "team"); team; team = team->next) {
    teamname = ezxml_attr(team, "name");
    for (driver = ezxml_child(team, "driver"); driver; driver = driver->next) {
        wmprintf("%s, %s: %s\n", ezxml_child(driver, "name")->txt, teamname,
                ezxml_child(driver, "points")->txt);
    }
}

ezxml_free(f1);
```

Alternately, the following would print out the name of the second driver on the first team:

```
ezxml_t f1 = ezxml_parse_str(xml_buf, strlen(xml_buf));

wmprintf("%s\n", ezxml_get(f1, "team", 0, "driver", 1, "name", -1)->txt);
ezxml_free(f1);
```

The -1 indicates the end of the argument list.

8.6.3 Function Documentation

8.6.3.1 ezxml_t ezxml_parse_str (char * *s*, size_t *len*)

Given a string of xml data and its length, parses it and creates an ezxml structure. For efficiency, modifies the data by adding null terminators and decoding ampersand sequences. If you don't want this, copy the data and pass the copy. Returns NULL on failure.

Parameters

s	String of xml data
len	Length of the xml data string

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

ezxml node if successful, NULL otherwise

8.6.3.2 ezxml_t ezxml_child (ezxml_t *xml*, const char * *name*)

Get first child tag with the given name.

Parameters

xml	ezxml node
name	Name of the child tag

Returns

First child tag (one level deeper) with the given name or NULL if not found

8.6.3.3 ezxml_t ezxml_idx (ezxml_t *xml*, int *idx*)

Parameters

in	*xml*	The xml node.
in	*idx*	The current index.

Returns

Nth tag with the same name in the same section at the same depth or NULL if not found. An index of 0 returns the tag given

8.6.3.4 const char* ezxml_attr (ezxml_t *xml*, const char * *attr*)

Traverses the ezxml structure to retrieve the value of requested tag attribute.

Parameters

xml	ezxml node
attr	Attribute name

Returns

Value of the requested tag attribute, or NULL if not found

8.6.3.5 ezxml_t ezxml_get (ezxml_t *xml*, ...)

Traverses the ezxml structure to retrieve a specific subtag. Takes a variable length list of tag names and indices. The argument list must be terminated by either an index of -1 or an empty string tag name.

Note

Example: title = ezxml_get(library, "shelf", 0, "book", 2, "title", -1); This retrieves the title of the 3rd book on the 1st shelf of library.

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

xml	ezxml node

Returns

Specific subtag, NULL if not found

8.6.3.6 char* ezxml_toxml (ezxml_t *xml*)

Converts an ezxml structure back to xml.

Parameters

xml	ezxml node

Returns

A string of xml data that must be freed

8.6.3.7 const char** ezxml_pi (ezxml_t *xml*, const char * *target*)

Parameters

in	*xml*	The xml node
in	*target*	The target string.

Returns

A NULL terminated array of processing instructions for the given target.

8.6.3.8 void ezxml_free (ezxml_t *xml*)

Frees the memory allocated to an ezxml node and it's children.

Parameters

xml	ezxml node

Returns

No value

8.6.3.9 const char* ezxml_error (ezxml_t *xml*)

Parameters

in	*xml*	The xml node

Returns

Parser error message or empty string if none

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.6.3.10 ezxml_t ezxml_new (const char * *name*)

Creates a new empty ezxml node with the given root tag name.

Parameters

in	*name*	Name for the ezxml node.

Returns

A new empty ezxml node

8.6.3.11 ezxml_t ezxml_add_child (ezxml_t *xml*, const char * *name*, size_t *off*)

Adds a child ezxml node.

Parameters

xml	parent ezxml node
name	Name of the child node
off	Offset of the child tag relative to the start of the parent tag's character content

Returns

Added ezxml node

8.6.3.12 ezxml_t ezxml_set_txt (ezxml_t *xml*, const char * *txt*)

Sets the character content for the given ezxml node.

Parameters

xml	ezxml node
txt	Character content to be set

Returns

Modified ezxml node

8.6.3.13 ezxml_t ezxml_set_attr (ezxml_t *xml*, const char * *name*, const char * *value*)

Sets the given tag attribute or adds a new attribute if not found. A value of NULL will remove the specified attribute.

Parameters

xml	ezxml node
name	Attribute name
value	Attribute value

Returns

ezxml node with modified attribute

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.6.3.14 ezxml_t ezxml_set_flag (ezxml_t *xml*, short *flag*)

Sets a flag for the given ezxml node.

Parameters

xml	ezxml node
flag	Flag to be set

Returns

Modified ezxml node

8.6.3.15 ezxml_t ezxml_cut (ezxml_t *xml*)

Removes a tag along with its subtags without freeing its memory.

Parameters

xml	ezxml node

Returns

Removed ezxml node

8.6.3.16 ezxml_t ezxml_insert (ezxml_t *xml*, ezxml_t *dest*, size_t *off*)

Inserts an existing tag into an ezxml structure.

Parameters

xml	ezxml node
dest	ezxml node to be inserted
off	Offset of the tag relative to the start of the dest tag's character content

Returns

Inserted ezxml node

8.6.4 Macro Documentation

8.6.4.1 #define ezxml_next(*xml*) ((xml) ? xml->next : NULL)

Returns

Next tag of the same name in the same section and depth or NULL if not found

8.6.4.2 #define ezxml_name(*xml*) ((xml) ? xml->name : NULL)

Returns

Name of the given tag

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.6.4.3 #define ezxml_txt(*xml*) ((xml) ? xml->txt : "")

Returns

Given tag's character content or empty string if none

8.6.4.4 #define ezxml_new_d(*name*) ezxml_set_flag(ezxml_new(ezxml_mem_strdup(name)), EZXML_NAMEM)

Wrapper for ezxml_new() that strdups name.

8.6.4.5 #define ezxml_add_child_d(*xml, name, off*)

Value:

```
ezxml_set_flag(ezxml_add_child(xml, ezxml_mem_strdup(name), off), \
                    EZXML_NAMEM)
```

Wrapper for ezxml_add_child() that strdups name.

8.6.4.6 #define ezxml_set_txt_d(*xml, txt*) ezxml_set_flag(ezxml_set_txt(xml, ezxml_mem_strdup(txt)), EZXML_TXTM)

Wrapper for ezxml_set_txt() that strdups txt.

8.6.4.7 #define ezxml_set_attr_d(*xml, name, value*)

Value:

```
ezxml_set_attr(ezxml_set_flag(xml, EZXML_DUP), \
                      ezxml_mem_strdup(name), ezxml_mem_strdup(value))
```

Wrapper for ezxml_set_attr() that strdups name/value. value cannot be NULL.

8.6.4.8 #define ezxml_move(*xml,* dest, *off*) ezxml_insert(ezxml_cut(xml), dest, off)

Moves an existing tag to become a subtag of dest at the given offset from the start of dest's character content.

Parameters

xml	ezxml node
dest	ezxml node to be moved
off	Offset of the tag relative to the start of the dest tag's character content

Returns

Moved ezxml node

8.6.4.9 #define ezxml_remove(*xml*) ezxml_free(ezxml_cut(xml))

Removes a tag along with all its subtags.

8.6.5 Enumeration Type Documentation

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.6.5.1 enum wm_exml_errno

Ezxml Error Codes

Enumerator

WM_E_EXML_INBUF EXML buffer null

WM_E_EXML_INLEN Invalid length

8.7 flash.h File Reference

Flash Driver.

8.7.1 Detailed Description

This is a wrapper driver which abstracts the internal flash driver (iflash) and the external flash driver (xflash) (if configured) into one.

Steps to use flash with mdev

1. Flash driver initialization

Call flash_drv_init from your application initialization function. This will initialize and register both internal and external flash drivers with mdev interface.

1. Open Flash device for use

This should be performed once before using Flash hardware. Here is example code to open mdev flash handler:

```
{
    mdev_t *fl_dev;
       fl_dev = flash_drv_open(FL_FTFS_DEV);
}
```

Note: The parameter must be either FL_INT or FL_EXT depending upon which flash you want to use.

1. Data read/write on Flash

flash_drv_read() and flash_drv_write() can be used to read and write data to/from the Flash device. Please note that you should pass the handle returned by flash_drv_open() to the flash_drv_read() and flash_drv_write() functions.

1. Erase Flash

flash_drv_erase() amd flash_drv_eraseall are used to erase the flash chip.

1. Closing Flash device.

flash_drv_close() call is used to close the flash driver.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.7.2 Function Documentation

8.7.2.1 void flash_drv_init (void)

Initialize the Flash Driver

This will initialize the internal as well as external (if defined) flash driver and register it with the mdev interface.

Returns

void

8.7.2.2 mdev_t* flash_drv_open (int *fl_dev*)

Open the Flash Driver

This will open the flash driver for the flash device specified using fl_dev.

Parameters

fl_dev	Flash device to be opened

Returns

mdev handle if successful, NULL otherwise

8.7.2.3 int flash_drv_close (mdev_t * *dev*)

Close the flash driver

This will close the flash driver

Parameters

dev	mdev handle to the driver

Returns

void

8.7.2.4 int flash_drv_write (mdev_t * *dev*, const uint8_t * *buf*, uint32_t *len*, uint32_t *addr*)

Write data to flash

This will write specified number of bytes to flash. Note that flash needs to be erased before writing.

Parameters

dev	mdev handle to the driver
buf	Pointer to the data to be written
len	Length of data to be written
addr	Write address in flash

Returns

0 on success

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.7.2.5 int flash_drv_read (mdev_t * *dev*, uint8_t * *buf*, uint32_t *len*, uint32_t *addr*)

Read data from flash

This will read specified number of bytes from flash.

Parameters

dev	mdev handle to the driver
buf	Pointer to store the read data
len	Length of data to be read
addr	Read address in flash

Returns

0 on success

8.7.2.6 int flash_drv_erase (mdev_t * *dev*, unsigned long *start*, unsigned long *size*)

Erase flash

This will erase the flash for the specified start address and size

Parameters

dev	mdev handle to the driver
start	Start address in flash
size	Number of bytes to be erased

Returns

0 on success

8.7.2.7 int flash_drv_eraseall (mdev_t * *dev*)

Erase entire flash

This will erase the entire flash chip

Parameters

dev	mdev handle to the driver

Returns

0 on success

8.7.2.8 int flash_drv_get_id (mdev_t * *dev*, uint64_t * *id*)

Get the unique id for flash This will get the unique id for the internal flash

Parameters

dev	mdev handle to the driver
id	a pointer to the 64 bit unique id

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

0 on success
-WM_FAIL if handle is not for internal flash or id is NULL

8.7.3 Typedef Documentation

8.7.3.1 typedef struct flash_desc flash_desc_t

The flash descriptor

All components that work with flash refer to the flash descriptor. The flash descriptor captures the details of the component resident in flash.

8.8 ftfs.h File Reference

File Table Filesystem (FTFS)

8.8.1 Detailed Description

FTFS is a very simple 'packing format' suitable for packing files into a serial EEPROM or Flash.

This API provides a similar interface to the basic streaming IO routines in stdio.h, except that FTFS is read-only.

8.8.2 Function Documentation

8.8.2.1 struct fs∗ ftfs_init (struct ftfs_super ∗ *sb*, flash_desc_t ∗ *fd*)

Initialize FTFS

This function initializes the File Table Filesystem module.

Parameters

out	*sb*	the ftfs superblock to be used for processing
in	*fd*	the flash descriptor

Returns

fs which is an opaque filesystem handle. This should be passed to subsequent open calls. The points to a member within the 'sb' that is passed as an argument.

8.8.2.2 file∗ ft_fopen (struct fs ∗ *fs*, const char ∗ *path*, const char ∗ *mode*)

Open a file

This function is analogous to the fopen() call.

Parameters

fs	The opaque filesystem handle returned on filesytem initialization
path	The path of the file to open
mode	The mode to open the file in

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

A pointer to a file handle to be used in subsequent calls to fread etc. or NULL in case of error.

8.8.2.3 int ft_fclose (file * f)

Close a file

This function is analogous to the fclose() call.

Parameters

f	The file handle returned in the ft_fopen() call.

Returns

-WM_E_FTFS_INVALID_FILE if invalid or NULL file is passed.
WM_SUCCESS if operation successful.

8.8.2.4 size_t ft_fread (void * *ptr*, size_t *size*, size_t *nmemb*, file * *f*)

Read from a file

This function is analogous to the fread() call.

Parameters

ptr	A pointer to the buffer to read data in.
size	The size of a single data element
nmemb	The number of elements of data to read
f	The file handle returned in the ft_fopen() call

Returns

The length of the data read in bytes.

8.8.2.5 long ft_ftell (file * f)

Get current file position

This function is analogous to the ftell() call.

Parameters

f	The file handle returned in the ft_fopen() call.

Returns

The current read offset in the file

8.8.2.6 int ft_fseek (file * *f*, long *offset*, int *whence*)

Seek to an offset within a file

This function is analogous to the fseek() call.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

f	The file handle returned in the ft_fopen() call.
offset	The number of bytes with which to move the read offset.
whence	Either of SEEK_SET, SEEK_CUR, or SEEK_END, to indicate from the start of the file, from the current offset or from the end of the file respectively.

Returns

-WM_E_FTFS_SEEK_FAIL if seek operation fail.
WM_SUCCESS if seek operation success.

8.8.2.7 static int ft_read_header (FT_HEADER * *fh*, uint32_t *address*, mdev_t * *fl_dev*) [inline],[static]

Read FTFS Header

Parameters

fh	Pointer to the FT_HEADER that should be filled
address	The address where the FTFS starts
fl_dev	The mdev_t pointer for the flash device

Returns

-WM_FAIL if header read operation fails.
WM_SUCCESS if header read operation is successful.

8.8.2.8 int ftfs_cli_init (struct fs * *fs*)

Enable FTFS CLI

The CLI can be enabled for only one of the FTFS's in the system.

Parameters

in	*fs*	the filesystem to use for ftfs commands

Note

This function can be directly called by the application after ftfs is initialized.

Returns

-WM_E_FTFS_INIT_FAIL if cli registration failed.
WM_SUCCESS if cli registration is successful.

8.8.3 Enumeration Type Documentation

8.8.3.1 enum wm_ftfs_errno

FTFS Error Codes

Enumerator

WM_E_FTFS_INIT_FAIL Error while performing ftfs test init

WM_E_FTFS_INVALID_FILE Invalid file pointer

WM_E_FTFS_SEEK_FAIL Ftfs seek failed

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.9 fw_upgrade_ed_chacha.h File Reference

Firmware Upgrades using ED25519 and Chacha20.

8.9.1 Detailed Description

The WMSDK provides convenience APIs to enable secure firmware upgrades using Chacha20 encryption and signing using ED25519 by taking a SHA512 checksum of the firmware image. This has to be enabled using the CONFIG_FWUPG_ED_CHACHA config option.

Please refer secure_upgrade.h for details about the upgrade process and the format of the firmware image.

Please also have a look at the README for fw_generator utility in tools/

A short summary of steps to follow for using this module are:

- Go to the wmsdk/tools/src/fw_generator
- Execute make. This will generate a executable named fw_generator
- Create a config file fw_generator.config using

```
~# ./fw_generator config > fw_generator.config
```

- Create an encrypted binary using

```
~# ./fw_generator <ipfile> <opfile> fw_generator.config
```

  The input file should be the .bin firmware image that is normally used to flash on the board.

- Use the fwupg_verification_key and fwupg_encrypt_decrypt_key as the keys for the functions ed_chacha_client_mode_upgrade_fw() or ed_chacha_server_mode_upgrade_fw()
- The mechanism to use for providing these keys to the functions is left to the applications. A common method is to store the keys in a manufacturing psm partition and then use it from there.

8.9.2 Function Documentation

8.9.2.1 int ed_chacha_client_mode_upgrade_fw (enum flash_comp *flash_comp_type*, char * *fw_upgrade_url*, uint8_t * *verification_pk*, uint8_t * *decrypt_key*)

Upgrade Firmware in client mode (HTTPC)

Parameters

in	*flash_comp_type*	Flash component to be updated. Valid values are FC_COMP_FW, and FC_COMP_WLAN_FW.
in	*fw_upgrade_url*	URL from which the firmware image should be fetched.
in	*verification_pk*	Pointer to the server's ED25519 public key which will be used for verification of the image. The key must be of length ED_CHACHA_VERIFICATION_KEY_LEN.
in	*decrypt_key*	Pointer to the CHACHA20 key which will be used for decryption of the firmware image. The key must be of length ED_CHACHA_DECRYPT_KEY_LEN.

Returns

WM_SUCCESS if the upgrade succeeds.
-WM_FAIL if the upgrade fails.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.9.2.2 int ed_chacha_server_mode_upgrade_fw (enum flash_comp *flash_comp_type*, uint8_t * *verification_pk*, uint8_t * *decrypt_key*, fw_data_read_fn_t *fw_data_read_fn*, void * *priv*)

Upgrade Firmware in Server mode (HTTPD)

Parameters

in	*flash_comp_type*	Flash component to be updated. Valid values are FC_COMP_FW, and FC_COMP_WLAN_FW.
in	*verification_pk*	Pointer to the server's ED25519 public key which will be used for verification of the image. The key must be of length ED_CHACHA_VERIFICATION_KEY_LEN.
in	*decrypt_key*	Pointer to the CHACHA20 key which will be used for decryption of the firmware image. The key must be of length ED_CHACHA_DECRYPT_KEY_LEN.
in	*fw_data_read_fn*	Pointer to a callback function of type fw_data_read_fn_t to be used by the secure_upgrade module to read the firmware image data
in	*priv*	Private data that will be passed to fw_data_read_fn as an argument. This can be used to maintain some context of the session.

Returns

WM_SUCCESS if the upgrade succeeds.
-WM_FAIL if the upgrade fails.

8.9.3 Macro Documentation

8.9.3.1 #define ED_CHACHA_DECRYPT_KEY_LEN 32

Length of the decryption key required by the ed_chacha_client_mode_upgrade_fw() function

8.9.3.2 #define ED_CHACHA_VERIFICATION_KEY_LEN 32

Length of the verification key required by the ed_chacha_client_mode_upgrade_fw() function

8.10 fw_upgrade_rsa_aes.h File Reference

Firmware Upgrades using RSA and AES.

8.10.1 Detailed Description

The WMSDK Provides an API to enable secure remote firmware upgrades. It uses AES CTR 128 encryption and signing using RSA by taking a SHA512 checksum of the firmware image. This has to be enabled using the CONFIG_FWUPG_RSA_AES config option.

Please refer secure_upgrade.h for details about the upgrade process and the format of the firmware image.

Please also have a look at the README for fw_generator utility in tools/

A short summary of steps to follow for using this module are:

- Go to the wmsdk/tools/src/fw_generator
- Execute make. ~# make CONFIG_OPTION=rsa_aesctr CYASSL_PATH=/path/to/cyassl-3.1.0 This will generate a executable named fw_generator
- Create a config file fw_generator.config using

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

```
~# ./fw_generator config > fw_generator.config
```

- Create an encrypted binary using

```
~# ./fw_generator <ipfile> <opfile> fw_generator.config
```

  The input file should be the .bin firmware image that is normally used to flash on the board.

- Use the fwupg_verification_key and fwupg_encrypt_decrypt_key as the keys for the functions rsa2048_aes128_client_mode_upgrade_fw() or rsa2048_aes128_server_mode_upgrade_fw()
- The mechanism to use for providing these keys to the functions is left to the applications. A common method is to store the keys in a manufacturing psm partition and then use it from there.

8.10.2 Function Documentation

8.10.2.1 int rsa2048_aes128_client_mode_upgrade_fw (enum flash_comp *flash_comp_type*, char * *fw_upgrade_url*, uint8_t * *verification_pk*, uint8_t * *decrypt_key*, const tls_client_t * *cfg*)

Upgrade Firmware in client mode (HTTPC)

Parameters

in	*flash_comp_type*	Flash component to be updated. Valid values are FC_COMP_FW, and FC_COMP_WLAN_FW.
in	*fw_upgrade_url*	URL from which the firmware image should be fetched.
in	*verification_pk*	Pointer to the server's RSA public key which will be used for verification of the image. The key must be of length RSA_PUBLIC_KEY_LEN.
in	*decrypt_key*	Pointer to the AES CTR key which will be used for decryption of the firmware image. The key must be of length AES_KEY_LEN.
in	*cfg*	Pointer to the tls_client_t structure variable giving TLS certificate details. This should be specified as "NULL" when TLS is not being used.

Note

tls_client_t paramater can be specified as NULL in case of HTTP. For HTTPS, the HTTPS support needs to be enabled from HTTP Client option in make menuconfig.

Returns

WM_SUCCESS if the upgrade succeeds.
-WM_FAIL if the upgrade fails.

8.10.2.2 int rsa2048_aes128_server_mode_upgrade_fw (enum flash_comp *flash_comp_type*, uint8_t * *verification_pk*, uint8_t * *decrypt_key*, fw_data_read_fn_t *fw_data_read_fn*, void * *priv*)

Upgrade Firmware in Server mode (HTTPD)

Parameters

in	*flash_comp_type*	Flash component to be updated. Valid values are FC_COMP_FW, and FC_COMP_WLAN_FW.
in	*verification_pk*	Pointer to the server's RSA public key which will be used for verification of the image. The key must be of length RSA_PUBLIC_KEY_LEN.
in	*decrypt_key*	Pointer to the AES CTR key which will be used for decryption of the firmware image. The key must be of length AES_KEY_LEN.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

in	*fw_data_read_fn*	Pointer to a callback function of type fw_data_read_fn_t to be used by the secure_upgrade module to read the firmware image data
in	*priv*	Private data that will be passed to fw_data_read_fn as an argument. This can be used to maintain some context of the session.

Returns

WM_SUCCESS if the upgrade succeeds.
-WM_FAIL if the upgrade fails.

8.10.3 Macro Documentation

8.10.3.1 #define RSA_PUBLIC_KEY_LEN 294

Length of the decryption key

8.10.3.2 #define AES_KEY_LEN 16

Length of the verification key required

8.11 generic_io.h File Reference

Input/Output GPIO related data structures.

8.11.1 Macro Documentation

8.11.1.1 #define GPIO_ACTIVE_LOW 0

GPIO needs to be pulled low to activate

8.11.1.2 #define GPIO_ACTIVE_HIGH 1

GPIO needs to be pulled high to activate

8.12 healthmon.h File Reference

Health Monitor.

8.12.1 Detailed Description

Health monitor is a system process that monitors the health of the device using. It queries the system services and applications periodically for their health status. It uses the hardware watchdog (or approximate software simulation, if hardware watchdog is not available) to recover if it detects any problems.

8.12.2 Usage

If application wishes to install its own periodic health check function, it needs to instantiate healthmon_handler structure that contains a valid is_sick function. This function will be called by healthmon thread periodically as

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

specified by the handler. The function must return 0 in case of healthy state and non-zero value in case of failure. The healthmon thread obeys consecutive failures registered in handler before declaring system as unhealthy. The *about_to_die* function is optional and gets called when system is found unhealthy and is about to reboot. In case of catastrophic events, even though healthmon thread is unable to run (e.g. hardfault, frozen system) the system reboots because the hardware watchdog is not strobed.

8.12.3 Function Documentation

8.12.3.1 int healthmon_init ( )

Initialize healthmon.

Note

This function is called by the wm_core_init function. Applications need not explicity call this function if they have already called wm_core_init().

Returns

WM_SUCCESS if call succeeds
-WM_FAIL in case of failure

8.12.3.2 int healthmon_start ( )

Starts the healthmon thread

Returns

WM_SUCCESS if thread is successfully started
-WM_FAIL if thread is already running or healthmon is uninitialized

8.12.3.3 int healthmon_register_handler (struct healthmon_handler * *handler*)

Register healthmon handler

Parameters

handler	Pointer to properly initialized healthmon_handler structure

Returns

WM_SUCCESS Successful registration
-WM_FAIL if handler already present with same name or invalid configuration

8.12.3.4 int healthmon_unregister_handler (const char * *name*)

Unregister healthmon handler

Parameters

name	Pointer to the name of the module

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS if handler unregistered
-WM_FAIL if no such handler found

8.12.3.5 int healthmon_change_check_interval (const char * *name*, unsigned int *interval*)

Changes check interval for specified module

Note

check interval should be a multiple of WKUP_INTERVAL

Parameters

name	Name of the module for which check interval is to be changed
interval	New check interval in seconds

Returns

WM_SUCCESS if check interval changed successfully
-WM_FAIL if invalid interval specified or no such module

8.12.3.6 int healthmon_set_final_about_to_die_handler (void(*)(void *data) *fun*)

Final about to die handler

Set the final action handler to be called by the healthmon before device resets due to the watchdog. Every module registered to the healthmon has its own about_to_die_handler which modules use to perform their own actions required before the system dies. The final_about_to_die_handler is optional and is executed after all the modules' about_to_die handlers get called. Typically this can be used to log diagnostics information captured by all the other about_to_die handlers, or any other application specific functions.

Note

It is not guaranteed that the final about the die handler will get called before all watchdog resets. This function is called from the healthmon thread's context, and in some system freeze cases, the healthmon may not get a chance to execute before the hardware watchdog kicks in.

Parameters

fun	Pointer to a function that needs to be executed

Returns

WM_SUCCESS if handler is registered successfully
-WM_E_PERM if this is called before healthmon_init

8.12.3.7 int healthmon_register_cli (void)

Register healthmon cli

Returns

WM_SUCCESS if cli registered successfully
-WM_FAIL in case of error

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.12.4 Macro Documentation

8.12.4.1 #define HM_NAME_MAX 15

Maximum name size for module or application name

8.12.4.2 #define MAX_HM_HANDLERS 10

Maximum number of healthmon handlers that can be registered

8.12.4.3 #define WD_STROBE_INTERVAL 20 /∗ 20 seconds ∗/

Default watchdog strobe interval. Please modify this if the hardware strobe requirement is different

8.12.4.4 #define WKUP_INTERVAL 5 /∗ 5 seconds ∗/

Wakeup interval for healthmon thread.

8.13 http_parse.h File Reference

Common HTTP functions.

8.13.1 Detailed Description

Copyright 2008-2015, Marvell International Ltd. All Rights Reserved.

8.13.2 Function Documentation

8.13.2.1 int httpd_get_tag_from_post_data (char ∗ *inbuf*, const char ∗ *tag*, char ∗ *val*, unsigned *val_len*)

Parse tag/value form elements present in HTTP POST body

Given a tag this function will retrieve its value from the buffer and return it to the caller.

Parameters

in	*inbuf*	Pointer to NULL-terminated buffer that holds POST data
in	*tag*	The tag to look for
out	*val*	Buffer where the value will be copied to
in	*val_len*	The length of the val buffer

Returns

WM_SUCCESS when a valid tag is found, error otherwise

8.13.2.2 int httpd_get_tag_from_url (**httpd_request_t** ∗ *req_p*, const char ∗ *tag*, char ∗ *val*, unsigned *val_len*)

Parse tag/value form elements present in HTTP GET URL

Given a tag this function will retrieve its value from the HTTP URL and return it to the caller.

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*req_p*	pointer to httpd_request_t object
in	*tag*	The tag to look for
out	*val*	Buffer where the value will be copied to
in	*val_len*	The length of the val buffer

Returns

WM_SUCCESS when a valid tag is found, error otherwise

8.14 httpc.h File Reference

HTTP Client Module.

8.14.1 Detailed Description

The HTTP Client module is a software component layered atop basic BSD socket API and CyaSSL (a TLS library). The HTTP Client module (hereafter called *the module*) provides a set of API and data structures to provide an easy way of communication with any web server supporting the HTTP protocol.

Some examples of the HTTP client users (hereafter called *the users*) of this module are: an email client, a web-sockets library, OTA firmware upgrade module, and all the myriad web services users.

The module implements a session based API. The users *open* a *session* with the server which internally sets up a TCP stream with the server. The user gets a handle to this session. All the other operations viz. sending HTTP headers, sending HTTP content, reading HTTP content, etc. are then performed on the same session through the session handle. The server also supports multiple HTTP transactions on the same server (which uses the keep-alive mechanism through persistent connections). A *transaction* is defined as a request and a response activity. Interleaved transactions are also supported. The module also supports chunked encoding transparently to the user.

An important feature supported by the module is HTTPS connections. This allows a user to perform his HTTP transactions securely. To help support this the module works with an external TLS library. During session open, the user *optionally* needs to supply a root certificate to verify the server certificate. The user can also provide the module with its own client certificate if the server is expected to do client verification. Apart from session the initialization API, the remaining API's handle secure and non-secure connections transparently to the user.

Lastly, the module also provides some API's to deal with HTTP session directly. i.e. it provides users access to the socket directly.

8.14.2 Usage

A typical HTTP Client usage scenario is as follows:

1. Initiate a connection with the remote Web Server using a call to http_open_session().
2. Prepare the request with a call the API to http_prepare_req().
3. Send one or more (for interleaved mode) HTTP requests to the server using the APIs http_prepare_req() and http_send_request() pair together.
4. An optional call the API to API http_get_response_hdr() can be made if the application wishes to know certain fields of the HTTP response.
5. Then multiple calls of API http_read_content() are made for reading the response from the remote Web Server.
6. The connection with the remote server is terminated using the API http_close_session()

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.14.3 Function Documentation

8.14.3.1 int http_open_session (http_session_t * *handle*, const char * *hostname*, const **httpc_cfg_t** * *httpc_cfg*)

Start an HTTP session.

This API starts a new HTTP session. It will create a socket and then connect to the server given as a parameter.

Parameters

in,out	*handle*	Pointer to a handle for the session. Will be assigned by the callee.
in	*hostname*	Host name or IP address. The hostname should be in the format [http-://]hostname[:portno][/path/to/resource]. The fields between the square brackets above are optional. The part specifying the path to resource will be ignored as it is of no use during session setup.
in	*httpc_cfg*	HTTP configuration. Pass NULL to allow HTTPC client to use default parameters.

Returns

WM_SUCCESS on success
-WM_E_INVAL if any of the arguments were invalid
-WM_FAIL if the API was unable to initiate an HTTP session with the server.

8.14.3.2 int http_prepare_req (http_session_t *handle*, const **http_req_t** * *req*, **http_hdr_field_sel_t** *field_flags*)

Prepare the HTTP header for sending to the server.

This API will prepare an HTTP request for sending it out through http_send_request(). Calling this API is mandatory before calling http_send_request().

Precondition

http_open_session()

Parameters

in	*handle*	Handle returned from the call to http_open_session()
in	*req*	Allocated and initialized http_req_t structure.
in	*field_flags*	Bitwise OR of flags of enum http_hdr_field_sel_t as required by the caller.

Note

The request structure http_req_t needs to be filled before invoking this API.
The field_flags parameter is a way to request the http client to use default field values for standard fields. If the corresponding bits are not set then the respective fields are not added. The user can add any custom fields including the standard ones using the API http_add_header()

Returns

Standard wmsdk return codes.

8.14.3.3 int http_add_header (http_session_t *handle*, const **http_req_t** * *req*, const char * *name*, const char * *value*)

Add custom http headers to the partially generated header.

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

This API allows the caller to add custom headers to the partial http header generated by earlier call to http_prepare_req()

Note

Calling this API is optional and needs to be called only if custom headers (or standard headers with non-default values) are to be added.

Precondition

http_prepare_req()

Parameters

in	*handle*	Handle returned from the call to http_open_session()
in	*req*	Structure of the type http_req_t passed earlier to http_prepare_req()
in	*name*	The name of the field. e.g User-Agent or If-Modified-Since
in	*value*	Value associated with the name given above.

Returns

Standard wmsdk value.

8.14.3.4 int http_send_request (http_session_t *handle*, const http_req_t * *req*)

Perform an HTTP request.

This API sends the prepared request header and content if applicable.

Note

If this API fails and the caller decides to abandon this session altogether it needs to call the API http_close_session() explicitly. Failure to do this will cause undefined behaviour.

Precondition

http_prepare_req() is mandatory. http_add_header() is optional.

Parameters

in	*handle*	Handle returned from the call to http_open_session()
in	*req*	The http_req_t structure filled up with appropriate parameters.

Returns

WM_SUCCESS on success
-WM_E_IO if failed to send data to network
-WM_E_INVAL for an invalid arguments.

8.14.3.5 int http_get_response_hdr (http_session_t *handle*, http_resp_t ** *resp*)

Get the HTTP response header

This API received the response header from the server and parses it. It then populates the http_resp_t structure and returns the pointer to this structure. The structure http_resp_t is allocated by the callee.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

The caller is expected to analyse this information to find out the result of the earlier request sent through API http_send_request(). If the earlier request was HTTP_GET then the caller will need to call the API http_read_content() to read the data sent by the server.

Note

Calling this API is *optional*. If the caller does not want to read/analyse the header and knows beforehand what to expect, it can directly call http_read_content(). The API http_read_content() will internally read the HTTP response header. However, the API http_read_content() will return an error if the HTTP status code returned is not 200. If you need to read content in cases where the return status code is other than 200 you need to call http_get_response_hdr() first.
After invoking this API, it is mandatory to read the entire content using the http_read_content() API, especially in the case of repeat http transaction on the same session handle (interleaved mode).
If this API fails and the caller decides to abandon this session altogether it needs to call http_close_session() explicitly. Failure to do this will cause undefined behaviour.

Precondition

http_send_request() is called before.

Parameters

in	*handle*	Handle returned from the call to http_new_session()
in,out	*resp*	Pointer to a pointer of type http_resp_t .The structure will be allocated by the callee. Note that the caller does **not** need to free the structure allocated and returned from this API. The allocation and free is done by the callee automatically.

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.14.3.6 int http_get_response_hdr_value (http_session_t *handle*, const char * *header_name*, char ** *value*)

Get a particular HTTP response header value.

This API parses the response header received from the server and returns the value corresponding to a given name. This can help the caller get value of a particular field not parsed automatically in http_resp_t

The caller is not expected to call http_get_response_hdr() before this. However, calling http_get_response_hdr() is recommended so that the user gets to know the exact status of the HTTP operation before he tries to extract any field not present in structure http_resp_t

Note

Calling this API is **optional**. If the caller does not want to read/analyse the header and knows beforehand what to expect, it can directly call http_read_content(). The API http_read_content() will internally read the HTTP response header. However, the API http_read_content() will return an error if the HTTP status code returned is not 200. If you need to read content in cases where the return status code is other than 200 you need to call http_get_response_hdr() or the current API first.
If this API fails and the caller decides to abandon this session altogether it needs to call http_close_session() explicitly. Failure to do this will cause undefined behaviour.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Precondition

http_send_request is called before this call. Calling http_get_response_hdr() is **not** mandatory but recommended.

Parameters

in	*handle*	Handle returned from the call to *http_new_session*
in	*header_name*	The name of the field whose value you wish to know
in,out	*value*	A pointer which will point to the value corresponding to the name passed as second parameter.

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.14.3.7 int http_get_response_hdr_all (http_session_t *handle*, http_header_pair_t * *arr*, int * *count*)

Get requested number of HTTP response header name-value pairs.

This API parses the response header received from the server and returns requested number of HTTP response header name-value pairs.

The caller is not expected to call API http_get_response_hdr() before this. However, calling http_get_response_hdr is recommended so that the user gets to know the exact status of the HTTP operation before he tries to extract any field not present in structure http_resp_t

Note

Calling this API is **optional**. If the caller does not want to read/analyse the header and knows beforehand what to expect, s/he can directly call http_read_content(). The API http_read_content() will internally read the HTTP response header. However, the API http_read_content() will return an error if the HTTP status code returned is not 200. If you need to read content in cases where the return status code is other than 200 you need to call http_get_response_hdr() or the current API first.
If this API fails and the caller decides to abandon this session altogether it needs to call http_close_session() explicitly. Failure to do this will cause undefined behaviour.

Precondition

http_send_request() is called before this call. Calling http_get_response_hdr() is **not** mandatory but recommended.

Parameters

in	*handle*	Handle returned from the call to http_new_session()
in,out	*arr*	A pointer that points to an array of the structure to hold the response header name-value pairs. This array needs to be pre-allocated and its pointer is to be passed, this pointer shall point to the array of requested number of response header name-value pairs.
in,out	*count*	An integer pointer that points to the number of response header name-value pairs that the user wants. This has to be passed by the user and it is later updated to the actual number of response header name-value pairs that were filled in the array mentioned above.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.14.3.8 int http_read_content (http_session_t *handle*, void * *buf*, uint32_t *max_len*)

Read data content from the stream.

This API will read the data sent by the server and pass it back to the caller.

The data transfer from the server to the client may be chunked. This fact will be reported in the response structure http_resp_t returned in a previous call to http_get_response_hdr(). Notwithstanding this information, chunked data handling is transparent to the caller. Thus, irrespective of the transfer type, this API should be called repeatedly till the return value of the API is zero.

Note

Once you start reading the content, it is mandatory to read the entire content before you begin the next HTTP transaction.
If this API fails and the caller decides to abandon this session altogether it needs to call http_close_session() explicitly. Failure to do this will cause undefined behaviour.

Parameters

in	*handle*	Handle returned from the call to http_new_session()
in,out	*buf*	Caller allocated buffer of size max_len
in	*max_len*	Size of the buffer. The data read will be less than or equal to this size, depending on the size of the resource.

Returns

On success, the number of bytes read is returned. Return value of zero is *not* an error case and should be treated as a signal that server is done with sending the data.
-WM_FAIL if this API is called again after the previous call retuned 0 or for any other error.

8.14.3.9 int http_parse_URL (const char * *URL*, char * *tmp_buf*, int *tmp_buf_len*, parsed_url_t * *parsed_url*)

Parse the given string into URL subcomponents.

The hostname string syntax is: [scheme://]hostname[:port][/path_to_resource?query_string::fragment_id] The fields between the square brackets above are optional.

Note

Buffer management: To free the caller of the task of allocating multiple buffers of arbitrary size to store each of the subcomponents we use the following strategy. This API takes a temporary buffer, allocated (static or dynamic) by the caller to store the subcomponents. The caller needs to allocate this buffer with size equal to (strlen(URL) + 10). When this API returns, the pointers in the parsed_url structure will point to appropriate address in this buffer. The caller may free this buffer after usage of the sub-components is complete. Freeing before that will cause undefined behavior.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*URL*	Pointer to the string containing the URL. This API will not modify this argument.
in	*tmp_buf*	A buffer where the strings will be stored after parsing. Needs to be allocated by the caller and should be of size atleast 10 greater than size of the URL string passed as first argument.
in	*tmp_buf_len*	Size of the caller allocated, temporary buffer sent as second argument.
in,out	*parsed_url*	Pointer to an allocated structure. The member pointers will point to the appropriate address in the temporary buffer holding the NULL terminated string corresponding to the subcomponent.

Returns

Standard WMSDK return codes.

8.14.3.10 int http_lowlevel_read (http_session_t *handle*, void * *buf*, unsigned *maxlen*)

HTTP lowlevel read

This API is present to serve clients which want to handle the HTTP stream directly. This is useful for handling websockets for e.g.

Warning

This is not for general users of HTTP client API. Normal HTTP transactions do not need this API.

Note

Once this API used no other API from the HTTP API set, except http_close_session(), can be used. This is because once this API is used the HTTP client loses track of the stream and only the caller using the API can know the state of the stream. The only advantage of using this API over normal socket read/write is that this API transparently takes care of TLS and non-TLS interface to read/write.

Parameters

in	*handle*	Handle returned from the call to http_new_session()
in,out	*buf*	Pointer to an allocated buffer of size equal to or more than the value of the third parameter maxlen
in	*maxlen*	The maximum number of bytes to be read from the network. Note that the actual read bytes can be less than this.

Returns

Number of bytes read. -WM_E_INVAL is returned in case of invalid parameters. Standard libc error codes are returned in case when there other problems.

8.14.3.11 int http_lowlevel_write (http_session_t *handle*, const void * *buf*, unsigned *len*)

HTTP lowlevel write

This API is present to serve clients which want to handle the HTTP stream directly. This is useful for handling websockets for e.g.

Warning

This is not for general users of HTTP client API. Normal HTTP transactions do not need this API.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Note

Once this API used no other API from the HTTP API set, except *http_close_session*, can be used. This is because once this API is used the HTTP client loses track of the stream and only the caller using the API can know the state of the stream. The only advantage of using this API over normal socket read/write is that this API transparently takes care of TLS and non-TLS interface to read/write.

Parameters

in	*handle*	Handle returned from the call to http_new_session()
in,out	*buf*	Pointer to a buffer which has the data to be written out.
in	*len*	The number of bytes to be written to the network.

Returns

Number of bytes written. -WM_E_INVAL is returned in case of invalid parameters. Standard libc error codes are returned in case when there other problems.

8.14.3.12 int httpc_write_chunked (http_session_t *handle*, const char * *data*, int *len*)

HTTP write chunked

This API is used for chunked transfer encoding

Note

To end the chunked data transfer, this API should be called with len=0, buf may be NULL

Parameters

in	*handle*	Handle retrieved from the call to http_open_session()
in	*data*	Pointer to the buffer to be written.
in	*len*	The number of bytes to be written.

Returns

WM_SUCCESS is returned on success.
-WM_E_INVAL is returned in case of invalid parameters.
-WM_FAIL is returned if session is invalid.
Standard libc error codes are returned when there are other problems.

8.14.3.13 void http_close_session (http_session_t * *handle*)

Close the session.

This API will close the session represented by the given handle. The socket associated with this session will be closed and thus the TCP connection with the server will be terminated. No requests on this same session will succeed after this API is called.

Note

The response structure http_resp_t returned earlier from http_get_response_hdr() will be invalid after this call.

Parameters

in	*handle*	Handle returned from the previous call to http_new_session(). The handle will be set to NULL by the callee thus effectively blocking further requests on the same session handle.

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.14.3.14 int http_setsockopt (http_session_t *handle*, int *level*, int *optname*, const void * *optval*, socklen_t *optlen*)

Set socket options for the socket associated with the session

Parameters

in	*handle*	Handle returned from the previous call to http_open_session().
in	*level*	Refer the man page for setsockopt.
in	*optname*	Refer the man page for setsockopt.
in	*optval*	Refer the man page for setsockopt.
in	*optlen*	Refer the man page for setsockopt.

Note

For the description and usage of the parameters, please refer to the man page for setsockopt. Their semantics are exactly same as mentioned in the man page.

Returns

WM_SUCCESS is returned on success.
-WM_E_INVAL is returned in case of invalid parameters.
-WM_FAIL is returned if session is invalid or if setsockopt fails.

8.14.3.15 int http_getsockopt (http_session_t *handle*, int *level*, int *optname*, void * *optval*, socklen_t * *optlen*)

Get socket options for the socket associated with the session

Parameters

in	*handle*	Handle returned from the previous call to http_open_session().
in	*level*	Refer the man page for setsockopt.
in	*optname*	Refer the man page for setsockopt.
in	*optval*	Refer the man page for setsockopt.
in	*optlen*	Refer the man page for setsockopt.

Note

For the description and usage of the parameters, please refer to the man page for setsockopt. Their semantics are exactly same as mentioned in the man page.

Returns

WM_SUCCESS is returned on success.
-WM_E_INVAL is returned in case of invalid parameters.
-WM_FAIL is returned if session is invalid or if setsockopt fails.

8.14.3.16 int http_get_sockfd_from_handle (http_session_t *handle*)

Get the socket fd from the http session handle

Parameters

in	*handle*	Handle returned from the previous call to http_open_session().

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

socket fd is returned on success.
-WM_E_INVAL is returned in case of invalid parameters.

8.14.3.17 int httpc_validate_url (const char * *url_str*)

Validate URL for outgoing HTTP request

This function is used to check if the given URL is valid or not

Parameters

in	*url_str*	URL to be validated

Returns

WM_SUCCESS if URL is valid
error if URL is invalid

8.14.3.18 int httpc_get (const char * *url_str,* http_session_t * *handle,* http_resp_t ** *http_resp,* const httpc_cfg_t * *httpc_cfg*)

Perform HTTP GET operation

This is a helper function which calls different functions of httpc module - httpc_validate_url(), http_open_session(), http_prepare_req(), http_send_request(), http_get_response_hdr() in sequence to perform HTTP GET operation. STANDARD_HDR_FLAGS are passed to http_prepare_req() internally.

This function reads only the HTTP response header. The caller of this function can use http_read_content() function to read the HTTP response. It is responsibility of the caller to close the HTTP session after the required work is done.

Parameters

in	*url_str*	Target URL for HTTP request
in	*handle*	Pointer to a handle for the session
in,out	*http_resp*	Structure used to give back header fields of http response to the caller.
in	*httpc_cfg*	HTTP configuration. Pass NULL to allow HTTPC client to use default parameters.

Returns

WM_SUCCESS is returned on success
other if there is any error returned by httpc_validate_url(), http_open_session(), or http_prepare_req(), or http_send_request(), or http_get_response_hdr() functions which are called by this function

8.14.4 Macro Documentation

8.14.4.1 #define DEFAULT_RETRY_COUNT 3

The number of times HTTP client tries to create a socket before giving up and returning error

8.14.5 Enumeration Type Documentation

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.14.5.1 enum http_hdr_field_sel_t

The OR of zero or more flags below is passed to the API http_prepare_req(). If the flag is passed the corresponding HTTP header field is added to the HTTP header. The values added will be default ones.

8.14.5.2 enum wm_httpc_errno

HTTPC Error Codes

Enumerator

WM_E_HTTPC_TCP_CONNECT_FAIL TCP connection failed (maybe due to unreachable server)

WM_E_HTTPC_FILE_NOT_FOUND HTTP File not found

WM_E_HTTPC_BAD_REQUEST HTTP Bad Request

WM_E_HTTPC_TLS_NOT_ENABLED TLS not enabled

WM_E_HTTPC_SOCKET_ERROR Socket error. Note to applications: Please check 'errno' for more information

WM_E_HTTPC_SOCKET_SHUTDOWN Peer has performed orderly shutdown

8.14.5.3 enum http_open_flags_t

If the given URL has a scheme field, and it is https then the http client will auto-switch to SSL mode, irrespective of the TLS_ENABLE flag below.

Enumerator

TLS_ENABLE Pass this flag when you want the connection to be SSL based

8.15 httpd.h File Reference

HTTP Web Server.

8.15.1 Detailed Description

HTTP Web Server can serve static pages from a file system, dynamic content using a server-side include (SSI) capability, and dynamic content using custom web service gateway interface (WSGI) handlers. This documentation describes:

1. How to initialize, start, stop, and shutdown the server.
2. How static files are served.
3. How to implement WSGI handlers.
4. How the SSI directives "file" and "virtual" are processed. Several limitations that may be unexpected by programmers accustomed to using more sophisticated web servers and frameworks are also discussed.

8.15.2 Usage

httpd must be initialized by calling httpd_init. After httpd_init returns WM_SUCCESS, httpd_start and httpd_stop can be called to start and stop the server respectively. Finally, to perform a final cleanup, httpd_shutdown should be called. After a successful call to httpd_start, the server is listening for connections on HTTP_PORT or HTTPS_PORT.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

The WSGI and SSI handlers can be registered by calling the functions httpd_register_wsgi_handler() and httpd_-register_ssi().

When a client request is made, httpd first checks if the requested URI matches any registered WSGI handlers (see below). If a matching handler is found it gets executed and the processing is terminated. If a file matching the requested URI is found, it is served. If that file is a .shtml file, it is processed for directives as described in the file handling section.

8.15.3 Web Services Gateway Interface (WSGI)

The Web Services Gateway Interface (WSGI) allows developers to take complete control over how HTTP requests are processed. When a request arrives, its request method (or type) and URI are parsed. A table of all registered WSGI handlers is queried for any matches. If an exact match is obtained, the control is transferred to the corresponding WSGI handler. If the WSGI handler returns WM_SUCCESS, no further processing is performed. In case A 500 Internal Server Error is returned to the client:

1. When any other value than WM_SUCCESS is returned by the WSGI handler.

A 404 Not Found Error is returned to the client:

1. If no handler (including the file system) matches the incoming URI
2. If incoming request type is not supported by that WSGI

A WSGI handler typically is executed in response to a HTTP GET or HTTP POST operation. In case of HTTP GET, the handler fetches some data from the device, formats it as desired and sends it to the requester. In case of HTTP POST, the handler reads the data that is sent along with the POST request and takes some action on the device. The WSGI handlers can make use of some of the common httpd helper functions to perform tasks pertaining to the sending/receiving of the data.

A simple WSGI HTTP GET handler is shown below: Example: JSON Time Get

```
int sys_get_time(httpd_request_t *req)
{
      char content[30];
      time_t time_val;

      some_api_to_fetch_time(&time_val)
      snprintf(content, sizeof(content), "{\"epoch\":%d}", time_val);

      return httpd_send_response(req, HTTP_RES_200
              , content, strlen(content), HTTP_CONTENT_JSON_STR);
}
```

A simple WSGI HTTP POST handler is shown below: Example: JSON Time Set

```
#include <jsonv2.h>
#define NUM_TOKENS    20
int sys_set_time(httpd_request_t *req)
{
      char sys_time_req[30];
      int ret;
      time_t time_val;

      ret = httpd_get_data(req, sys_time_req,
              sizeof(sys_time_req));
      if (ret < 0) {
              wmprintf("Failed to get post request data");
              return ret;
      }
      jsontok_t json_tokens[NUM_TOKENS];
      jobj_t jobj;
      json_init(&jobj, json_tokens, NUM_TOKENS);
      ret = json_parse(&jobj, sys_time_req, req->body_nbytes);
      if (ret != WM_SUCCESS)
              return -WM_FAIL;

      json_get_val_int(&jobj, "epoch", (int *)&time_val);
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

```
        ret = Set the time using the value time_val func()
        if (ret == WM_SUCCESS)
                return httpd_send_response(req, HTTP_RES_200
                   , HTTPD_JSON_SUCCESS, strlen(HTTPD_JSON_SUCCESS),
                   HTTP_CONTENT_JSON_STR);
        else
                return httpd_send_response(req, HTTP_RES_200
                   , HTTPD_JSON_ERROR, strlen(HTTPD_JSON_ERROR),
                   HTTP_CONTENT_JSON_STR);
}
```

The above examples being simplest examples use the most convenient functions, namely httpd_send_response() and httpd_get_data() for most of their tasks. An advanced user may prefer to have more flexibility by using functions that provide a greater control. While httpd_get_data() is a basic low level function, httpd_send_response() is a convenience function that performs the following:

- httpd_send_response()
  1. Purge HTTP headers (valid only for GET request) : httpd_purge_headers()
  2. Send response status (200 OK/404 Not Found etc.): httpd_send()
  3. Send the default headers from httpd_hdr_field_sel_t : httpd_send_default_headers()
  4. Send the Content-Type of the response : httpd_send_header()
  5. Send the header terminator : httpd_send_crlf()
  6. Send the chunked data : httpd_send_chunk()
  7. Send the last chunk : httpd_send_chunk() with length = 0

8.15.4 Static File Handling

Static files can be served from a filesystem. A WSGI handler can be registered at '/'. This handler can call httpd_-handle_file() for dealing with file serving requests. If the fs contains a file called /404.html, this file will be returned when httpd receives a request for a URL that does not have a registered handler or page. The following table summarizes the Content-Types with which files of various file extensions are served:

Extension	Content-Type
.html	text/html (with charset=iso8859-1)
.shtml	same as .html but with caching disabled
.css	text/css
.png	image/png
.gif	image/gif
.jpg	image/jpeg
[none]	application/octet-stream
[other]	text/plain

8.15.4.1 Compressed Files

If the request for a file *abc.html* is received, the web server first tries to check if a compressed version of this file *abc.html.gz* exists. If found, this compressed version is served. The *Content-Encoding* field is set to *gzip* for these requests, so that the HTTP clients can handle it properly.

8.15.5 Server Side Include (SSI)

Server Side Include (SSI) directives are mark-up snippets in .shtml files that are detected by HTTPD and replaced with some other content. Two SSI directives are supported: file and virtual.

When the file directive is encountered, it is replaced by the contents of a file. For example, suppose you have the following .shtml files in your filesystem:

index.shtml:

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

```
Hello World
<!--#include file=footer.html-->
```

footer.html:

```
<b>Copyright 2014 your name here<\b>
```

When index.shtml is requested, the SSI file directive will be replaced by the contents of footer.html, and the following will be served: index.shtml:

```
Hello World
<b>Copyright 2008-2015 your name here<\b>
```

Much like a file SSI directive, a virtual SSI directive can also be used in .shtml files. They look something like this:

```
<!--#include virtual=foo 1 2 3-->
```

When a virtual SSI directive is encountered, control is passed to the SSI handler that is registered under the specified name ("foo" in this example). The foo handler would be passed the string "1 2 3" as its argument, along with the open socket to which the response is being sent, and an httpd_request_t that can be queried for the request type (POST or GET) and any url-encoded form data (POST only). The handler is expected to send complete http chunks to the socket (see httpd_send_chunk).

When using .shtml files and SSI directives, keep the following tips and caveats in mind:

- See HTTPD_SSI_CALL, httpd_register_ssi, struct httpd_ssifunction for additional documentation and pointers.
- Placing SSI directives in a file that does not have the .shtml extension has no effect; the file will be served as-is.
- Each SSI directive should be on its own line with no other mark up. If there is any other markup, it will be ignored and will not be served to the client.
- The SSI directives must start with either "<!--#include file=" or "<!--#include virtual=". They must not contain any extraneous spaces.
- File names passed to the "file=" directive must not contain spaces.
- If a virtual directive is encountered for which there is no registered handler, nothing will be output and the page processing will continue.

8.15.6 Other Considerations

HTTPD operates in a highly memory constrained environment. Accordingly, it allocates a limited amount of memory for various elements such as the maximum URL length, maximum number of form elements, etc. Developers writing the handlers and web applications must consider these constraints. Details regarding these are discussed in the Defines section below. If necessary, these values can be changed and HTTPD can be rebuilt.

8.15.7 HTTPD Call Graph

The HTTPD library fundamentally uses the functions httpd_send() and httpd_recv() to send and receive data over the HTTP socket. Additional functionality is built on top of these functions in a hierarchical manner for the user's convenience. The following callgraph shows the layering of these calls.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.15.8 Function Documentation

8.15.8.1 int httpd_init (void)

Initialize the httpd

This function must be called before any of the other API functions. If any of the initialization steps fail, the function will fail.

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.2 int httpd_shutdown (void)

Shutdown the httpd

This function is the opposite of httpd_init(). It cleans up any resources used by the httpd. If the httpd thread is running (i.e., httpd_stop has not been called), it will be halted.

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.3 int httpd_start (void)

Start the httpd

Upon WM_SUCCESS, the user can expect that the server is running and ready to receive connections. httpd_stop() should be called to stop the server. A return value of -WM_FAIL generally indicates that the thread has already been started.

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.15.8.4 int httpd_stop (void)

Stop the httpd

This function is the opposite of httpd_start(). It stops the httpd. If WM_SUCCESS is returned, the server has been stopped and will no longer accept connections. -WM_FAIL is returned if the httpd is uninitialized, or if an internal error occurs (which is not expected). Note that this function forces the httpd thread to stop and closes any open sockets. If a handler is being executed during this time, it will be terminated.

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.5 int httpd_is_running (void)

Check if httpd is running

Returns

0 if httpd is not running
non-zero if it is running

8.15.8.6 int httpd_send (int *sock*, const char * *buf*, int *len*)

Helper function to send a buffer over a connection.

WSGI handlers will often want to send data over a connection. This can be achieved with the regular send() system call, however that call may not send all of the data. This helper function calls send in a loop repeatedly until all data has been sent or until an error occurs. This function internally calls tls_send() if TLS is enabled, else it calls send()

Parameters

in	*sock*	The socket on which the data is to be sent
in	*buf*	Pointer to the buffer
in	*len*	Length of the buffer

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

See Also

httpd_send_chunk

8.15.8.7 int httpd_send_chunk (int *sock*, const char * *buf*, int *len*)

Helper function to send an HTTP chunk of data over a connection

This function sends HTTP chunked data. It includes sending the chunked header (which consists of the chunk size in ascii followed by \r\n characters), then the actual http data and at the end chunk end indicator (\r\n characters) If the len is passed as 0 to this function, it sends the last chunk i.e. 0\r\n

Note that while using chunked data, HTTP Header "Transfer Encoding: Chunked" should be sent in the response

Send either an HTTP chunk of data or the terminating HTTP chunk.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*sock*	The socket on which the data is to be sent
in	*buf*	Pointer to the buffer or NULL for last chunk
in	*len*	Length of the buffer or 0 for last chunk

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.8 int httpd_send_chunk_begin (int *sock*, int *size*)

Helper function to send a HTTP chunk header

Note that this function is not required when httpd_send_chunk is used. httpd_send_chunk internally calls this function

This function can be followed by a call to httpd_send to send the actual content followed by sending the chunk end indicator (\r\n)

Parameters

in	*sock*	The socket over which chunk header is to be sent
in	*size*	Length of the chunk to be sent

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.9 void* httpd_get_priv_data (int *sockfd*)

Function to get pointer to httpd private data

This function returns a pointer to the httpd_priv data for a socket

Parameters

in	*sockfd*	The socket descriptor for which the private data is desired.

Returns

Void pointer to the socket's private data

8.15.8.10 void httpd_clear_priv_data (int *sockfd*)

Function to clear httpd private data

This function clears the httpd_priv data associated with a socket,

Parameters

in	*sockfd*	The socket descriptor whose associated private data needs to be cleared.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.15.8.11 int httpd_register_wsgi_handler (struct **httpd_wsgi_call** * *wsgi_call*)

Register a WSGI handler

WSGI handlers declared must be registered with the httpd by calling this function.

Parameters

in	*wsgi_call*	pointer to a struct httpd_wsgi_call. The memory location pointed to by this pointer should be available until the httpd handler is unregistered.

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.12 int httpd_register_wsgi_handlers (struct **httpd_wsgi_call** * *wsgi_call_list*, int *handler_cnt*)

Register a list of WSGI handlers

WSGI handler list declared must be registered with the httpd by calling this function.

Parameters

in	*wsgi_call_list*	pointer to a list of struct httpd_wsgi_call. The memory location pointed to by this pointer should be available until the httpd handler list is unregistered.
in	*handler_cnt*	number of WSGI handlers in wsgi_call_list

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.13 int httpd_unregister_wsgi_handler (struct **httpd_wsgi_call** * *wsgi_call*)

Unregister a WSGI handler

This call unregisters a WSGI handler.

Parameters

in	*wsgi_call*	pointer to a struct httpd_wsgi_call

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.14 int httpd_unregister_wsgi_handlers (struct **httpd_wsgi_call** * *wsgi_call_list*, int *handler_cnt*)

Unregister a list of WSGI handlers

WSGI handler list declared can be unregistered with the httpd by calling this function.

Parameters

in	*wsgi_call_list*	list of struct httpd_wsgi_call
in	*handler_cnt*	number of WSGI handlers in wsgi_call_list

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.15 int httpd_handle_file (**httpd_request_t** * *req_p*, **struct fs** * *fs*)

Serve a file request

Certain WSGI handlers may be so configured that they serve static files from the FTFS . This function takes care of service these static files.

Note that there can be multiple FTFS images in the system. Since the FTFS image provides a flat file hierarchy, multiple images can be mounted at various locations. For example, ftfs1 can be mounted at /, while ftfs2 can be mounted at /marvell. An HTTP GET for /a.html will serve the file a.html from ftfs1, while an HTTP GET request for /marvell/a.html will service the file a.html from ftfs2.

Parameters

in	*req_p*	The incoming HTTP request httpd_request_t
in	*fs*	the filesystem structure for this ftfs. This structure is returned by the rfget_ftfs_init() or the ftfs_init() calls.

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.16 int httpd_register_ssi (**struct httpd_ssi_call** * *ssi_call*)

Register a virtual SSI handler

virtual SSI handlers declared with HTTPD_SSI_CALL must be registered with the httpd by calling this function.

Parameters

in	*ssi_call*	pointer to the httpd_ssi_call to register

Returns

WM_SUCCESS if successful
-WM_E_HTTPD_SSI_MAX if the maximum number of supported handlers has been reached
-WM_FAIL if the handler is already registered

Note

All SSI handlers are cleared when httpd_init is called.

8.15.8.17 void httpd_unregister_ssi (**struct httpd_ssi_call** * *ssi_call*)

Unregister a virtual SSI handler

Parameters

in	*ssi_call*	pointer to the httpd_ssi_call to unregister

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Note

All SSI handlers are cleared when httpd_init is called.

8.15.8.18 int httpd_cli_init (void)

Register and Enable httpd cli-commands

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

Note

This function can be called by the application after the httpd is initialized.

8.15.8.19 int httpd_send_hdr_from_code (int *sock*, int *stat_code*, enum **http_content_type** *content_type*)

Send HTTP Status Headers

This function sends the corresponding HTTP headers on the socket.

Parameters

in	*sock*	the socket to send headers on
in	*stat_code*	the status code WM_SUCCESS, -WM_E_HTTPD_HANDLER_404 etc.
in	*content_type*	the content type http_content_type

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.20 int httpd_validate_uri (char * *req_uri*, const char * *uri*, int *flags*)

Validate an incoming request with a registered handler

This function is used to match the incoming request URI against the one that is registered with HTTPD

Parameters

in	*req_uri*	URI of the incoming HTTP request
in	*uri*	URI of an handler registered with HTTPD
in	*flags*	Flag indicating if exact match is required or not APP_HTTP_FLAGS_NO_EXACT_MATCH

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.21 int httpd_recv (int *sock*, void * *buf*, size_t *n*, int *flags*)

Helper function to receive data over a socket

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

This function abstracts the TCP/IP recv call. We need the abstraction because recv could be through TLS connection in which case we do not call the TCP/IP stack 'recv' function directly.

This API is only for the httpd handlers who have been passed the socket descriptors by httpd.

Parameters

in	*sock*	The socket on which the data is to be received
out	*buf*	The buffer in which data will be received
in	*n*	Number of bytes to be received
in	*flags*	Same values as that of flag parameter in man page of recv

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.22 int httpd_send_response (**httpd_request_t** ∗ *req,* const char ∗ *first_line,* char ∗ *content,* int *length,* const char ∗ *content_type*)

Send the entire HTTP response

This is a helper function which can be used by the WSGI handlers to send the entire HTTP response.

This function calls the httpd_send_response_header() function to send the response header. Please refer to its description for exactly what this function does. Then, if the request type is HTTP HEAD, only the headers are sent out and this function terminates, else data provided by the user (chunked/non chunked data) follows.

Note that this API is optional and WSGI handlers may prefer to send the response as per their wish using other HTTPD APIs.

Parameters

in	*req*	The incoming HTTP request httpd_request_t
in	*first_line*	First line of the response. for e.g.: To send 200 OK it will be: "HTTP/1.1 200 OK\r\n"
in	*content*	The data to be sent
in	*length*	The length of data to be sent
in	*content_type*	The content type of the response. for e.g.: To send JSON response it will be: "application/json"

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.23 int httpd_send_response_header (**httpd_request_t** ∗ *req,* const char ∗ *first_line,* int *length,* const char ∗ *content_type*)

Send entire HTTP Response Header

This is a helper function which can be used by the WSGI handlers to send the entire HTTP response header. This function first purges the incoming headers (this step is valid only for HTTP GET or HEAD request.) Then it sends the first line of the response i.e. 200 OK, 404 Not Found etc. as specified by the user. Further, it sends the default HTTP headers as per the value of hdr_fields parameter of that WSGI. After that, it sends the Content-Type HTTP header using the content_type sent by the user.

Note that this API is optional and WSGI handlers may prefer to send the response as per their wish using other HTTPD APIs.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*req*	The incoming HTTP request httpd_request_t
in	*first_line*	First line of the response. for e.g.: To send 200 OK it will be: "HTTP/1.1 200 OK\r\n"
in	*length*	The length of data to be sent
in	*content_type*	The content type of the response. for e.g.: To send JSON response it will be: "application/json"

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.24 int httpd_send_response_301 (**httpd_request_t** * *req*, char * *location*, const char * *content_type*, char * *content*, int *content_len*)

Send HTTP response 301: Moved Permanently

This is a helper function which can be used by the WSGI handlers to send HTTP response HTTP 301 Moved Permanently. This function first purges the incoming headers (this step is valid only for HTTP GET or HEAD request.) Then it sends the first line of the response i.e. 301 Moved Permanently. Further, it sends the default HTTP headers as per the value of hdr_fields parameter of that WSGI. After that, it sends the HTTP header Location which is the new link for redirection. Then, it sends the Content-Type HTTP header using the content_type sent by the user. If the request type is HTTP HEAD, only the headers are sent out and this function terminates, else data provided by the user (chunked/non chunked data) follows.

Note that this API is optional and WSGI handlers may prefer to send the response as per their wish using other HTTPD APIs.

Parameters

in	*req*	The incoming HTTP request httpd_request_t
in	*location*	The new link for redirection
in	*content_type*	The content type of the response. for e.g.: To send JSON response it will be: "application/json"
in	*content*	The data to be sent (optional)
in	*content_len*	The length of data to be sent

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.25 int httpd_send_header (int *sock*, const char * *name*, const char * *value*)

Send an HTTP header

This function can be used to send out an HTTP header.

It accepts a name value pair as input and sends it out as: "name: value\r\n" for e.g.: To send header "Content-Type: application/json\r\n" call httpd_send_header(sock, "Content-Type", "application/json");

Parameters

in	*sock*	the socket to send header on
in	*name*	name of the HTTP header
in	*value*	value of the HTTP header 'name'

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.26 int httpd_send_default_headers (int *sock*, int *hdr_fields*)

Send the default HTTP Headers

This function can be used by the WSGI handlers to send out some or all headers present in httpd_hdr_field_sel_t. The WSGI handlers when declared can decide which headers they want to send out in the response and accordingly populate hdr_fields parameter in the httpd_wsgi_call structure

Parameters

in	*sock*	the socket to send headers on
in	*hdr_fields*	The hdr_fields parameter of the matching WSGI for the HTTP request, selected from httpd_hdr_field_sel_t

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.27 int httpd_purge_headers (int *sock*)

Purge the headers of the incoming HTTP request

This function is used to purge the headers of the incoming HTTPD request

Note that it is mandatory to consume all the HTTP headers for the proper functioning of web server

Parameters

in	*sock*	the socket of the incoming HTTP request

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.28 int httpd_parse_hdr_tags_with_cb (**httpd_request_t** ∗ *req*, int *sock*, char ∗ *buffer*, int *len*, **httpd_parse_hdr_cb_t** *cb*)

Parse the HTTP headers into the req httpd_request_t structure and execute a callback, if registered, for each HTTP header

This function will one by one read lines terminated by CR-LF (\r\n) from the socket and parse the headers contained in them. For each HTTP header it will call the callback of type httpd_parse_hdr_cb_t if registered. Some of the headers (at this time only the Content-Length) will be then copied to the req httpd_request_t structure.

After executing this function, the socket will point to the request entity body, if any.

Parameters

in	*req*	The incoming HTTP request httpd_request_t
in	*sock*	The socket of the incoming HTTP request
in	*buffer*	scratch Buffer used to read the HTTP headers

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

in	*len*	Length of the scratch buffer
in	*cb*	The callback of type httpd_parse_hdr_cb_t which is to be called for each HTTP header

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.29 int httpd_parse_hdr_tags (**httpd_request_t** * *req*, int *sock*, char * *scratch*, int *len*)

Parse the HTTP headers into the req httpd_request_t structure.

This function calls httpd_parse_hdr_tags_with_cb with cb as NULL

After executing this function, the socket will point to the request entity body, if any.

Parameters

in	*req*	The incoming HTTP request httpd_request_t
in	*sock*	The socket of the incoming HTTP request
in	*scratch*	Buffer used to read the HTTP headers
in	*len*	Length of the scratch buffer

Returns

WM_SUCCESS if successful
-WM_FAIL otherwise

8.15.8.30 int httpd_get_data (**httpd_request_t** * *req*, char * *content*, int *length*)

Get the incoming data in case of HTTP POST request

This function first parses the HTTP header tags and then receives the data in the buffer provided by the user.

It can be called multiple times if the size of user buffer is less than the actual data that is received. Care has been taken that, in such situations, the HTTP headers are parsed only once. The req->remaining_bytes count can be used to monitor the number of bytes remaining to be read.

Note that this API is optional and WSGI handlers may prefer to parse the headers and get the data as per their wish using other HTTPD APIs.

Parameters

in	*req*	The incoming HTTP request httpd_request_t
out	*content*	The buffer in which the data is to be received
in	*length*	The length of the content buffer

Returns

The number of bytes read from the connection. A zero value will be returned when the client has performed an orderly shutdown.
-WM_FAIL otherwise

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.15.8.31 int httpd_use_tls_certificates (const **httpd_tls_certs_t** * *httpd_tls_certs*)

Call this function to set httpd tls certificates.

Note

The certificate buffers passed can be freed immediately after this function returns.

Parameters

httpd_tls_certs	Certificate details. httpd_tls_certs_t

Returns

WM_SUCCESS TLS certificates set successfully.
-WM_FAIL HTTPS is not enabled in the server configuration.

8.15.8.32 void httpd_send_override (**httpd_send_func_t** *send_func*)

Override HTTPD's Send Function

Parameters

in	*send_func*	The overriding function of type httpd_send_func_t

8.15.8.33 void httpd_recv_override (**httpd_recv_func_t** *recv_func*)

Override HTTPD's Receive Function

Parameters

in	*recv_func*	The overriding function of type httpd_recv_func_t

8.15.8.34 void httpd_purge_all_socket_data (**httpd_request_t** * *req*)

Purge all the data from the HTTP socket. This function is useful when the wsgi PUT/POST handlers need to respond without reading any actual HTTP data.

Parameters

in	*req*	The incoming HTTP request httpd_request_t

8.15.9 Macro Documentation

8.15.9.1 #define HTTP_PORT 80

Port on which the httpd listens for non secure connections.

8.15.9.2 #define HTTPS_PORT 443

Port on which the httpd listens for secure connections.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.15.9.3 #define APP_HTTP_FLAGS_NO_EXACT_MATCH 1

Do not perform an exact match of the URI, but ensure that only the prefix is correct. This is required when the registered URI is /nodes/ while the handler wishes to match with /nodes/0/, /nodes/1/ etc.

8.15.9.4 #define HTTPD_REQ_TYPE_UNKNOWN 0

This REQ_TYPE should not be used externally.

8.15.9.5 #define HTTPD_REQ_TYPE_POST 1

This request type indicates that incoming http request is of type POST.

8.15.9.6 #define HTTPD_REQ_TYPE_GET 2

This request type indicates that incoming http request is of type GET.

Note

GET does not support URL-encoded tag/val pairs.

8.15.9.7 #define HTTPD_REQ_TYPE_PUT 4

This request type indicates that incoming http request is of type PUT.

8.15.9.8 #define HTTPD_REQ_TYPE_DELETE 8

This request type indicates that incoming http request is of type DELETE.

8.15.9.9 #define HTTPD_REQ_TYPE_HEAD 16

This request type indicates that incoming http request is of type HEAD.

8.15.9.10 #define HTTPD_DONE 1

This value is returned when a WSGI handler is successful to indicate that all processing has been completed.

8.15.9.11 #define HTTPD_MAX_MESSAGE 512

Maximum received message length

This value is essentially the longest single line that the HTTPD can handle. This limits programmers and users in some interesting ways. For example:

1. This is the maximum length of any url-encoded data sent as the body of a POST request. If this limit is exceeded, the server will generate a 500 Internal Server Error response. Applications that require a higher capacity for form data must implement their own custom handler to process form data or tune this value.
2. This is the maximum length of a header line that can be processed by httpd_parse_hdr_tags. Header lines that exceed this length will be truncated before being analyzed. Applications that require more capacity must implement their own custom handler to process HTTP header data.
3. This is the maximum length of a single line in a .shtml script.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Developers must ensure that these limits are addressed to avoid unexpected errors in the field.

Note

This value does not include any NULL termination required internally by the httpd.

8.15.9.12 #define HTTPD_MAX_URI_LENGTH 256

Maximum URI length

This is the maximum supported URI length. For example, if a client sends a GET /foobar.html, the length of "/foobar.-html" must be less than this value. Programmers are obliged to ensure that none of the URIs or HTTPD file names exceed this limit. If URIs do end up exceeding this limit, a 500 Internal Server Error response will be generated.

Note

This value does not include any NULL termination required internally by the HTTPD. However it does include the leading "/".

8.15.9.13 #define HTTPD_MAX_VAL_LENGTH 64

Maximum length of the value portion of a tag/value pair

If this length is exceeded, the value will be truncated to this length and no error will be generated. If debugging is enabled, however, a warning will be written to the console.

Note

Truncating the user input passed by client may result in unexpected behavior depending on how a custom handler uses that data. Programmers should take steps to validate and/or limit user input using javascript, input tag properties, or other client-side means.
This value does not include the NULL termination of the value.

8.15.9.14 #define HTTPD_MAX_CONTENT_TYPE_LENGTH 128

Maximum length of Content-Type

httpd_parse_hdr_tags() will parse the headers and store the value of the Content-Type. The maximum value of this field has been arrived keeping in mind the need to support the following type of Content-Type header for working with multipart/form-data for file uploads: Content-Type: multipart/form-data; boundary=<delimiter_max_length_-70> The length of this line is 104 bytes

8.15.9.15 #define HTTP_CONTENT_JSON_STR "application/json"

Content-Type: application/json

8.15.9.16 #define HTTP_CONTENT_XML_STR "text/xml"

Content-Type: text/xml

8.15.9.17 #define HTTP_CONTENT_PLAIN_TEXT_STR "text/plain"

Content-Type: text/plain

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.15.9.18 #define HTTP_RES_200 "HTTP/1.1 200 OK\r\n"

HTTP Response: 200 OK

8.15.9.19 #define HTTP_RES_204 "HTTP/1.1 204 No Content\r\n"

HTTP Response: 204 No Content

8.15.9.20 #define HTTP_RES_207 "HTTP/1.1 207 Multi-Status\r\n"

HTTP Response: 207 Multi-Status

8.15.9.21 #define HTTP_RES_301 "HTTP/1.1 301 Moved Permanently\r\n"

HTTP Response: 301 Moved Permanently

8.15.9.22 #define HTTP_RES_304 "HTTP/1.1 304 Not Modified\r\n"

HTTP Response: 304 Not Modified

8.15.9.23 #define HTTP_RES_400 "HTTP/1.1 400 Bad Request\r\n"

HTTP Response: 400 Bad Request

8.15.9.24 #define HTTP_RES_401 "HTTP/1.1 401 Unauthorized\r\n"

HTTP Response: 401 Unauthorized

8.15.9.25 #define HTTP_RES_404 "HTTP/1.1 404 Not Found\r\n"

HTTP Response: 404 Not Found

8.15.9.26 #define HTTP_RES_405 "HTTP/1.1 405 Method Not Allowed\r\n"

HTTP Response: 405 Method Not Allowed

8.15.9.27 #define HTTP_RES_429 "HTTP/1.1 429 Too Many Requests\r\n"

HTTP Response: 429 Too Many Requests

8.15.9.28 #define HTTP_RES_470 "HTTP/1.1 470 Connection Authorization Required\r\n"

HTTP Response: 470 Connection Authorization Required

8.15.9.29 #define HTTP_RES_500 "HTTP/1.1 500 Internal Server Error\r\n"

HTTP Response: 500 HTTP Internal Server Error

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.15.9.30 #define HTTP_RES_505 "HTTP/1.1 505 HTTP Version Not Supported\r\n"

HTTP Response: 505 HTTP Version Not Supported

8.15.9.31 #define HTTP_EVENT_200 "EVENT/1.0 200 OK\r\n"

HTTP Event: 200 OK

8.15.9.32 #define httpd_send_crlf(_sock_) httpd_send(_sock_, "\r\n", 2);

Send carriage return and new line characters on the socket _sock_

8.15.9.33 #define HTTPD_MAX_SSI_ARGS 32

Maximum length of virtual SSI arguments

The supported format for virtual SSI directives is:

```
<!--#include virtual="foo arg1 arg2 arg3"-->
```

or

```
<!--#include virtual=foo arg1 arg2 arg3-->
```

HTTPD_MAX_SSI_ARGS is the maximum length for "arg1 arg2 arg3" (not including the quotes). If this length is exceeded, the arguments will be truncated.

Note

This does not include any null-termination required internally by the httpd or the virtual SSI handlers.
whitespace between the function name (e.g., foo) and the first argument (e.g. arg1) is ignored and not passed to handlers.
Double quotes are reserved and should only be used as described in the above examples.

8.15.9.34 #define HTTPD_SSI_CALL(*name, str, function*)

httpd virtual SSI handler declaration

This macro is used for declaring a virtual SSI handler struct httpd_ssi_call. The function is then added to the list of handlers with httpd_register_ssi, and removed with httpd_unregister_ssi function.

Parameters

name	Name of the httpd_ssi_call structure object
str	The string name of the function, used in the script file
function	A pointer to the function that implements the handler

8.15.9.35 #define MAX_REGISTERED_SSIS 32

Maximum number of virtual SSI handlers that are supported.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.15.9.36 #define HTTPD_DEFAULT_HDR_FLAGS

Value:

```
(HTTPD_HDR_ADD_SERVER | \
                HTTPD_HDR_ADD_CONN_CLOSE |
      HTTPD_HDR_ADD_TYPE_CHUNKED)
```

Default HTTP headers from httpd_hdr_field_sel_t

8.15.10 Typedef Documentation

8.15.10.1 typedef void(* httpd_free_priv_data_fn_t)(void *priv_data)

Pointer to a function to free private data associated with a socket.

8.15.10.2 typedef void(* httpd_parse_hdr_cb_t)(char *buf, int len, **httpd_request_t** *req)

A callback to be called for every HTTP header that is found in the HTTP request. This callback receives the actual header (buf), its length (len) and the corresponding HTTPD request (req)

8.15.10.3 typedef int(* httpd_ssifunction)(const **httpd_request_t** *req, const char *args, int sock, char *scratch)

SSI virtual include handler function prototype

Parameters

in	*req*	The incoming HTTP request httpd_request_t. Minimally, the type and filename members will be initialized. Handlers are not expected to write to this struct.
in	*args*	Null-terminated string that is the arguments passed to the handler in the include directive.
in	*sock*	Socket to which the handler output should be sent.
in	*scratch*	A scratch buffer that can be used by the handler. It is HTTPD_MAX_MESSAGE bytes long.

Note

SSI virtual include handlers should only send complete chunks i.e. they should send any data using httpd_send_chunk, not httpd_send or any lower-level system calls. These handlers should also not send the final chunk i.e. they should not invoke httpd_send_chunk with a length of 0.

8.15.10.4 typedef int(* httpd_send_func_t)(int conn, const char *buf, int len)

Prototype for the httpd_send function.

Applications can write their own httpd_send function and register it using httpd_send_override() to override the default function. This may be useful if the HTTP data needs to be encrypted/packetized before sending on socket.

8.15.10.5 typedef int(* httpd_recv_func_t)(int fd, void *buf, size_t n, int flags)

Prototype for httpd_recv function.

Applications can write their own httpd_recv function and register it using httpd_recv_override() to override the default function. This may be useful if encrypted/packetized HTTP data is expected on the socket.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.15.11 Enumeration Type Documentation

8.15.11.1 enum wm_httpd_errno

HTTPD Error Codes

Enumerator

WM_E_HTTPD_HDR_FNAME Error parsing filename in HTTP Header
WM_E_HTTPD_NOTSUPP HTTPD doesn't support the http version
WM_E_HTTPD_TOOLONG Post data too long
WM_E_HTTPD_DATA_RD Data read failed
WM_E_HTTPD_SSI_MAX SSI Table full
WM_E_HTTPD_HANDLER_404 Handler generated a HTTP 404 File Not Found error
WM_E_HTTPD_HANDLER_400 Handler generated a HTTP 400 Bad Request error
WM_E_HTTPD_HANDLER_500 Handler generated a HTTP 500 Internal Server error
WM_E_HTTPD_NO_HANDLER No handler was found

8.15.11.2 enum http_content_type

HTTP Content types

Enumerator

HTTP_CONTENT_PLAIN_TEXT Content-Type: text/plain
HTTP_CONTENT_JSON Content-Type: application/json
HTTP_CONTENT_XML Content-Type: text/xml
HTTP_CONTENT_HTML Content-Type: text/html
HTTP_CONTENT_JPEG Content-Type: image/jpeg

8.15.11.3 enum httpd_hdr_field_sel_t

HTTP Headers to be sent in the HTTP response

Enumerator

HTTPD_HDR_ADD_SERVER "Server: Marvell-WM\r\n"
HTTPD_HDR_ADD_CONN_KEEP_ALIVE "Connection: keep-alive\r\n"
HTTPD_HDR_ADD_CONN_CLOSE "Connection: close\r\n"
HTTPD_HDR_ADD_TYPE_CHUNKED "Transfer-Encoding: chunked\r\n"
HTTPD_HDR_ADD_CACHE_CTRL "Cache-Control: no-store, no-cache, must-revalidate\r\n"
HTTPD_HDR_ADD_CACHE_CTRL_NO_CHK "Cache-Control: post-check=0, pre-check=0\r\n"
HTTPD_HDR_ADD_PRAGMA_NO_CACHE "Pragma: no-cache\r\n"

8.16 json_generator.h File Reference

JSON generator module.

8.16.1 Detailed Description

This module is used to generate JSON strings

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.16.2 Usage

The JSON Generator can create various JSON strings. Here is an example:

```
json_str_init(&jstr, buff, sizeof(buff));
// This initializes the json generator to generate string in the buffer buff

json_start_object(&jstr);
json_set_val_str(&jstr, "name", "John Travolta");
json_set_val_int(&jstr, "roll", 123);
json_push_object(&jstr, "address");
json_set_val_str(&jstr, "house", "Atlas");
json_set_val_str(&jstr, "street", "New Avenue");
json_set_val_int(&jstr, "pin", 39065);
json_pop_object(&jstr);
json_close_object(&jstr);
```

will create the above mentioned JSON string.

Similarly arrays can be created as:

```
json_start_object(&jstr);
json_push_array_object(&jstr, "roll");
json_set_array_int(&jstr, 10);
json_set_array_int(&jstr, 20);
json_set_array_int(&jstr, 30);
json_pop_array_object(&jstr);
json_close_object(&jstr);
```

will create the JSON string

```
{ "roll": [ [ 10, 20, 30] ] }
```

Similarly complex use-case like objects within arrays can be created as:

```
json_str_init(&jstr, buff, sizeof(buff));
json_start_object(&jstr);
json_push_array_object(&jstr, "arr");
json_start_object(&jstr);
json_set_val_str(&jstr, "name", "John Travolta");
json_set_val_int(&jstr, "roll", 123);
json_close_object(&jstr);
json_start_object(&jstr);
json_set_val_str(&jstr, "name", "Foo Bar");
json_set_val_int(&jstr, "roll", 112);
json_close_object(&jstr);
json_pop_array_object(&jstr);
json_close_object(&jstr);
```

will create the JSON string

```
{ "arr": [
         { "name": "John Travolta", "roll": 123 },
         { "name": "Foo Bar", "roll": 112 }
     ]
}
```

8.16.3 Function Documentation

8.16.3.1 void json_str_init (struct json_str * *jptr*, char * *buff*, int *len*)

Initialize the JSON generator

This function initializes the JSON generator.

Parameters

out	*jptr*	A pointer to the structure json_str which will be used as a handle for the rest of the json_str operations.
out	*buff*	A pointer to a buffer that will be used to store the generated JSON data.
in	*len*	The length of the buffer pointed to by buff

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.16.3.2 int json_start_object (struct json_str * *jptr*)

Start a new JSON object

This function starts a new JSON object

Parameters

in	*jptr*	Pointer to json_str object.

Returns

-WM_E_JSON_OBUF if buffer available is insufficient or a buffer overflow has already occured.
WM_SUCCESS if operation successful.

8.16.3.3 int json_push_object (struct json_str * *jptr*, const char * *name*)

Start a new composite object

This function starts a new composite object which may have its own set of key-value pairs. Any json_set() calls performed after this will create key-value pairs within this composite object.

Parameters

in	*jptr*	A pointer to the json_str handle returned by the json_str_init() call.
in	*name*	The name of the composite object that should be created.

Returns

-WM_E_JSON_OBUF if buffer available is insufficient or a buffer overflow has already occurred.
WM_SUCCESS if operation successful.

8.16.3.4 int json_push_array_object (struct json_str * *jptr*, const char * *name*)

Start a JSON array object.

This function creates an element that has an array as its value.

Parameters

in	*jptr*	A pointer to the json_str handle returned by the json_str_init() call.
in	*name*	The name of the array object

Returns

-WM_E_JSON_OBUF if buffer available is insufficient or a buffer overflow has already occurred.
WM_SUCCESS if operation successful.

8.16.3.5 int json_pop_array_object (struct json_str * *jptr*)

Close a JSON array object.

This closes a previously pushed JSON object.

Parameters

in	*jptr*	Pointer to json_str object.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

-WM_E_JSON_OBUF if buffer available is insufficient or a buffer overflow has already occurred.
WM_SUCCESS if operation successful.

8.16.3.6 int json_close_object (struct json_str * *jptr*)

Close JSON object

This function closes a JSON object that was started earlier.

Parameters

in	*jptr*	Pointer to json_str object.

Returns

-WM_E_JSON_OBUF if buffer available is insufficient or a buffer overflow has already occurred.
WM_SUCCESS if operation successful.

8.16.3.7 int json_start_array (struct json_str * *jptr*)

Start a JSON array

This function starts a JSON array. This is different than the json_push_arraj_object(), in that json_push_array_object() creates an key-value pair with an array as the value, while this simply starts an array object.

Parameters

in	*jptr*	Pointer to json_str object.

Returns

-WM_E_JSON_OBUF if buffer available is insufficient or a buffer overflow has already occurred.
WM_SUCCESS if operation successful.

8.16.3.8 int json_close_array (struct json_str * *jptr*)

Close a JSON array

This function closes a JSON array started by json_start_array().

Parameters

in	*jptr*	Pointer to json_str object.

Returns

-WM_E_JSON_OBUF if buffer available is insufficient or a buffer overflow has already occurred.
WM_SUCCESS if operation successful.

8.16.4 Macro Documentation

8.16.4.1 #define json_pop_object(*jptr*) **json_close_object((jptr))**

Close a composite object

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

This closes a composite object. Any json_set() calls after this will create key-value pairs at the same depth as the composite object that was closed.

8.16.4.2 #define json_set_val_str(*jptr, name, val*)

Value:

```
json_set_object_value(jptr, name, val, 0, 0.0, 0, \
                                JSON_VAL_STR)
```

Create a key with a string value.

This function adds a key-value pair to the JSON text with string as the value.

Parameters

in	*jptr*	A pointer to the json_str handle returned by the json_str_init() call.
in	*name*	The name of the key
in	*val*	The string value of the key

8.16.4.3 #define json_set_val_int(*jptr, name, val*)

Value:

```
json_set_object_value(jptr, name, NULL, val, 0.0, 0, \
                                JSON_VAL_INT)
```

Create a key with an integer value.

This function adds a key-value pair to the JSON text with an integer as the value.

Parameters

in	*jptr*	A pointer to the json_str handle returned by the json_str_init() call.
in	*name*	The name of the key
in	*val*	The integer value of the key

8.16.4.4 #define json_set_val_uint(*jptr, name, val*) json_set_object_value(jptr, name, NULL, val, 0.0, 0, JSON_VAL_UINT)

Create a key with an unsigned integer value.

This function adds a key-value pair to the JSON text with an unsigned integer as the value.

Parameters

in	*jptr*	A pointer to the json_str handle returned by the json_str_init() call.
in	*name*	The name of the key
in	*val*	The unsigned integer value of the key

8.16.4.5 #define json_set_val_uint_64(*jptr, name, val*)

Value:

```
json_set_object_value(jptr, name, NULL, val, 0.0, 0, \
                        JSON_VAL_UINT_64)
```

Create a key with an unsigned 64 bit integer value.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

This function adds a key-value pair to the JSON text with an unsigned 64 bit integer as the value.

Parameters

in	*jptr*	A pointer to the json_str handle returned by the json_str_init() call.
in	*name*	The name of the key
in	*val*	The unsigned 64 bit integer value of the key

8.16.4.6 #define json_set_val_float(*jptr, name, val*)

Value:

```
json_set_object_value(jptr, name, NULL, 0, val, 2, \
                              JSON_VAL_FLOAT)
```

Create a key with a float value.

This function adds a key-value pair to the JSON text with a float as the value.

Parameters

in	*jptr*	A pointer to the json_str handle returned by the json_str_init() call.
in	*name*	The name of the key
in	*val*	The float value of the key

8.16.4.7 #define json_set_val_float_precision(*jptr, name, val, precision*)

Value:

```
json_set_object_value(jptr, name, NULL, 0, \
                              val, precision, JSON_VAL_FLOAT)
```

Create a key with a float value and decimal precision.

This function adds a key-value pair to the JSON text with a float as the value.

Parameters

	jptr	A pointer to the json_str handle returned by the json_str_init() call.
in	*name*	The name of the key
in	*val*	The float value of the key
in	*precision*	The number of precision digits for float

8.16.4.8 #define json_set_val_bool(*jptr, name, val*)

Value:

```
((val == true) ? \
                (json_set_object_value(jptr, name, NULL, 1, 0.0, 0, \
                              JSON_VAL_BOOL)) : \
                (json_set_object_value(jptr, name, NULL, 0, 0.0, 0, \
                              JSON_VAL_BOOL)))
```

Create a key with an boolean value.

This function adds a key-value pair to the JSON text with an integer as the value.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*jptr*	A pointer to the json_str handle returned by the json_str_init() call.
in	*name*	The name of the key
in	*val*	The boolean value of the key.

8.16.4.9 #define json_set_val_null(*jptr, name*)

Value:

```
(json_set_object_value(jptr, name, NULL, 0, 0.0, 0, \
                                JSON_VAL_NULL))
```

Create a key with a null value.

This function adds a key-value pair to the JSON text with null as the value.

Parameters

in	*jptr*	A pointer to the json_str handle returned by the json_str_init() call.
in	*name*	The name of the key

8.16.4.10 #define json_set_array_str(*jptr, val*) json_set_array_value(jptr, val, 0, 0.0, JSON_VAL_STR)

Create a string array element

This function creates a string array element in a previously started JSON array.

Parameters

in	*jptr*	A pointer to the json_str handle returned by the json_str_init() call.
in	*val*	A pointer to the string value of the array element.

8.16.4.11 #define json_set_array_int(*jptr, val*) json_set_array_value(jptr, NULL, val, 0.0, JSON_VAL_INT)

Create an integer array element

This function creates an integer array element in a previously started JSON array.

Parameters

in	*jptr*	A pointer to the json_str handle returned by the json_str_init() call.
in	*val*	The integer value of the array element.

8.16.4.12 #define json_set_array_float(*jptr, val*) json_set_array_value(jptr, NULL, 0, val, JSON_VAL_FLOAT)

Create a float array element

This function creates a float array element in a previously started JSON array.

Parameters

in	*jptr*	A pointer to the json_str handle returned by the json_str_init() call.
in	*val*	A pointer to the float value of the array element.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.16.4.13 #define json_set_array_bool(*jptr, val*)

Value:

```
((val == true) ? \
                (json_set_array_value(jptr, NULL, 1, 0.0, JSON_VAL_BOOL)) : \
                (json_set_array_value(jptr, NULL, 0, 0.0, JSON_VAL_BOOL)))
```

Create an boolean array element

This function creates an boolean array element in a previously started JSON array.

Parameters

in	*jptr*	A pointer to the json_str handle returned by the json_str_init() call.
in	*val*	The boolean value of the array element.

8.17 jsonv2.h File Reference

JSON parser module.

8.17.1 Detailed Description

This module is used to parse a JSON string For JSON generation, please refer json_generator.h

8.17.2 Usage

The JSON data can be viewed as a tree of key-value pairs. Composite objects indicate special nodes within the tree that anchor other subtrees. Arrays are special nodes that hold a collection of JSON elements.

The JSON parser module provides two sets of APIs, one to search inside a JSON object and the other to search inside a JSON array.

8.17.2.1 Search inside an Object

The APIs in this category are:

- json_get_val_bool()
- json_get_val_int()
- json_get_val_int64()
- json_get_val_float()
- json_get_val_str()
- json_get_composite_object()
- json_release_composite_object()
- json_get_array_object()
- json_release_array_object()

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.17.2.2 Search inside an array

The APIs in this category are:

- json_array_get_bool()
- json_array_get_int()
- json_array_get_in64t()
- json_array_get_float()
- json_array_get_str()
- json_array_get_composite_object()
- json_array_release_composite_object()
- json_array_get_array_object()
- json_array_release_array_object()

A valid JSON string is an object and so the object search APIs should be used first.

The JSON data that looks like the following:

```
{ "name" : "JSON Parser",
  "version" : 2.0,
  "copyright" : 2014,
  "supported_el" : ["bool","int","float","str","object","array"],
  "features" : { "objects":true,
            "arrays":"yes"
            }
}
```

can be thought of as a tree with a root node that has 5 children "name", "version", "copyright", "supported_el" and "features". The node at "supported_el" is an array with 6 elements. The node at "features" further has 2 child nodes "objects", "arrays". The JSON can be much more complex with multiple levels of objects/arrays and with different types of elements inside an array or object.

This JSON parser is based on JSMN which treats the JSON string as a collection of tokens. A token can be an object, array, string or primitive. In the above, "name", "JSON Parser", "vesion", etc. are all strings. 2.0, 2014, true are primitives. The entire thing is a JSON object, as is the portion beginning from "{" after "features" :. The part between [] is an array. So, in all, the above JSON string consists of 21 tokens. This detail is given here so that the application developers have an idea about how large the JSON token array should be, for successful JSON parsing.

Using this JSON parser module you can parse the above elements with the following C code:

```
#define NUM_TOKENS      30
// Define an array of JSON tokens
jsontok_t json_tokens[NUM_TOKENS];
jobj_t jobj;
// Initialise the JSON parser and parse the given string
int err = json_init(&jobj, json_tokens, NUM_TOKENS, json_str_ptr,
        strlen(json_str_ptr));
if (err != WM_SUCCESS)
   return err;
// Get the value of "name" which is a string
json_get_val_str(&jobj, "name", sval, sizeof(sval));
// Get the value of "version" which is a float
json_get_val_float(&jobj, "version", &fval);
// Get the value of "copyright" which is an integer
json_get_val_int(&jobj, "copyright", &ival);
// Get the JSON array object "supported_el". After this function returns
// successfully, only the array search APIs can be used unless the array
// is released.
json_get_array_object(&jobj, "supported_el", &num_elements);
// Get the value at index 2 of the array which is a string value
json_array_get_str(&jobj, 2, sval, sizeof(sval));
 // Release the array object so that the scope of searches returns back
 // to the parent object.
 json_release_array_object(&jobj);
// Get the JSON composite object "features". After this function returns
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

```
// successfully, the scope of the subsequent searches is constrained to
// within this object, unless the object is released.
json_get_composite_object(&jobj, "features");
// Get the value of "objects" which is a boolean
json_get_val_bool(&jobj, "objects", &bval);
// Release the composite object so that the scope of searches returns back
// to the parent object.
json_release_composite_object(&jobj);
```

8.17.3 Function Documentation

8.17.3.1 int json_init (jobj_t * *jobj*, jsontok_t * *tokens*, int *num_tokens*, char * *js*, int *js_len*)

Initialize the JSON Parser and parse the given JSON string.

This function initializes the JSON object that will be used for all the subsequent JSON parser APIs. It also parses the given JSON string and sets up the internal jobj for the subsequent APIs.

Parameters

in,out	*jobj*	Pointer to a JSON object jobj_t assigned by the application. This is set up for further use internally by this API.
in	*tokens*	Pointer to an array of JSON tokens jsontok_t assigned by the application. To get an idea about how large the array should be, please refer Usage section.
in	*num_tokens*	Number of tokens in the tokens array.
in	*js*	Pointer to the JSON string.
in	*js_len*	Length of the JSON string.

Returns

WM_SUCCESS on successful parsing. json_is_object() and/or json_is_array() can then be used to find out if the parsed string is a JSON object or array so that the appropriate parsing APIs can be used.
-WM_E_JSON_NOMEM if insufficient number of tokens are provided in json_init().
-WM_E_JSON_INVAL if there are invalid characters in the given JSON string.
-WM_E_JSON_INCOMPLETE if the given JSON string is incomplete.
-WM_E_JSON_INVALID_JOBJ if the given json string is not a valid JSON object.
-WM_E_JSON_FAIL if any other error was encountered.

8.17.3.2 bool json_is_object (jobj_t * *jobj*)

Find out if the current object is a JSON Object

Parameters

in	*jobj*	The current JSON object pointer of type jobj_t

Returns

true if it is an object
false if it is not an object

8.17.3.3 bool json_is_array (jobj_t * *jobj*)

Find out if the current object is a JSON Array

Parameters

in	*jobj*	The current JSON object pointer of type jobj_t

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

true if it is an array
false if it is not an array

8.17.3.4 int json_get_val_bool (jobj_t * *jobj*, char * *key*, bool * *value*)

Get JSON bool value

Gets the value of a JSON boolean element from an object based on the given key.

Parameters

in	*jobj*	Current JSON search object jobj_t.
in	*key*	Name of the JSON element.
out	*value*	Pointer to a boolean variable assigned by the application. This will hold the value on success.

Returns

WM_SUCCESS on success
-WM_E_JSON_NOT_FOUND if an element with the given key was not found.
-WM_E_JSON_INVALID_TYPE if an element with the given key was found but it was not a boolean.
-WM_E_JSON_INVALID_JOBJ if the search object was not a valid JSON object.
-WM_E_JSON_FAIL if any other error was encountered.

8.17.3.5 int json_get_val_int (jobj_t * *jobj*, char * *key*, int * *value*)

Get JSON integer value

Gets the value of a JSON integer element from an object based on the given key.

Parameters

in	*jobj*	Current JSON search object jobj_t.
in	*key*	Name of the JSON element.
out	*value*	Pointer to an integer variable assigned by the application. This will hold the value on success.

Returns

WM_SUCCESS on success
-WM_E_JSON_NOT_FOUND if an element with the given key was not found.
-WM_E_JSON_INVALID_TYPE if an element with the given key was found but it was not an integer.
-WM_E_JSON_INVALID_JOBJ if the search object was not a valid JSON object.
-WM_E_JSON_FAIL if any other error was encountered.

8.17.3.6 int json_get_val_int64 (jobj_t * *jobj*, char * *key*, int64_t * *value*)

Get 64bit JSON integer value

Gets the value of a JSON integer element from an object based on the given key.

Parameters

in	*jobj*	Current JSON search object jobj_t.
in	*key*	Name of the JSON element.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

out	*value*	Pointer to a 64bit integer variable assigned by the application. This will hold the value on success.

Returns

WM_SUCCESS on success
-WM_E_JSON_NOT_FOUND if an element with the given key was not found.
-WM_E_JSON_INVALID_TYPE if an element with the given key was found but it was not an integer.
-WM_E_JSON_INVALID_JOBJ if the search object was not a valid JSON object.
-WM_E_JSON_FAIL if any other error was encountered.

8.17.3.7 int json_get_val_float (jobj_t * *jobj*, char * *key*, float * *value*)

Get JSON float value

Gets the value of a JSON float element from an object based on the given key.

Parameters

in	*jobj*	Current JSON search object jobj_t.
in	*key*	Name of the JSON element.
out	*value*	Pointer to a float variable assigned by the application. This will hold the value on success.

Returns

WM_SUCCESS on success
-WM_E_JSON_NOT_FOUND if an element with the given key was not found.
-WM_E_JSON_INVALID_TYPE if an element with the given key was found but it was not a float.
-WM_E_JSON_INVALID_JOBJ if the search object was not a valid JSON object.
-WM_E_JSON_FAIL if any other error was encountered.

8.17.3.8 int json_get_val_str (jobj_t * *jobj*, char * *key*, char * *value*, int *max_len*)

Get JSON string value

Gets the value of a JSON string element from an object based on the given key.

Parameters

in	*jobj*	Current JSON search object jobj_t.
in	*key*	Name of the JSON element.
out	*value*	Pointer to a buffer assigned by the application to hold the string. This will hold the null terminated string on success.
in	*max_len*	Length of the buffer passed.

Returns

WM_SUCCESS on success
-WM_E_JSON_NOT_FOUND if an element with the given key was not found.
-WM_E_JSON_INVALID_TYPE if an element with the given key was found but it was not a string.
-WM_E_JSON_INVALID_JOBJ if the search object was not a valid JSON object.
-WM_E_JSON_NOMEM if the buffer provided was insufficient.
-WM_E_JSON_FAIL if any other error was encountered.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.17.3.9 int json_get_val_str_len (jobj_t * jobj, char * key, int * len)

Get JSON string length

Gets the length of a JSON string element from an object based on the given key. This is useful when the application does not know what size of buffer to use for json_get_val_str()

Parameters

in	*jobj*	Current JSON search object jobj_t.
in	*key*	Name of the JSON element.
out	*len*	Pointer to an integer assigned by the application to hold the length of string. Applications will have to use a buffer of size atleast 1 byte more than this length

Returns

WM_SUCCESS on success
-WM_E_JSON_NOT_FOUND if an element with the given key was not found.
-WM_E_JSON_INVALID_TYPE if an element with the given key was found but it was not a string.
-WM_E_JSON_INVALID_JOBJ if the search object was not a valid JSON object.
-WM_E_JSON_FAIL if any other error was encountered.

8.17.3.10 int json_get_composite_object (jobj_t * jobj, char * key)

Get JSON composite object

Gets a composite JSON object from another object based on the given key.

Parameters

in,out	*jobj*	Current JSON search object jobj_t. On success, the jobj will be modified such that the scope of subsequent searches will be limited to this composite object. Use json_release_composite_object() to expand the scope back to the parent object.
in	*key*	Name of the JSON element.

Returns

WM_SUCCESS on success
-WM_E_JSON_NOT_FOUND if an element with the given key was not found.
-WM_E_JSON_INVALID_TYPE if an element with the given key was found but it was not an object.
-WM_E_JSON_INVALID_JOBJ if the search object was not a valid JSON object or the object found was itself not a valid JSON object.
-WM_E_JSON_FAIL if any other error was encountered.

8.17.3.11 int json_release_composite_object (jobj_t * jobj)

Release a JSON composite object

This function expands the scope of searches back to the parent object, if it was previously constrained using json_-get_composite_object().

Parameters

in,out	*jobj*	Current JSON search object jobj_t. On success, this will be modified such that the scope of future searches will be expanded back to the parent object.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS on success.
-WM_E_JSON_FAIL if any error was encountered.

8.17.3.12 int json_get_array_object (jobj_t * *jobj*, char * *key*, int * *num_elements*)

Get JSON array object

Gets a JSON array object from another object based on the given key.

Parameters

in,out	*jobj*	Current JSON search object jobj_t. On success, the jobj will be modified such that the scope of subsequent searches will be limited to this array. Use json_release_array_object() to expand the scope back to the parent object.
in	*key*	Name of the JSON element.
out	*num_elements*	Pointer to an integer assigned by the application. On sucess, this will hold the number of elements in the array found.

Returns

WM_SUCCESS on success
-WM_E_JSON_NOT_FOUND if an element with the given key was not found.
-WM_E_JSON_INVALID_TYPE if an element with the given key was found but it was not an array.
-WM_E_JSON_INVALID_JOBJ if the search object was not a valid JSON object.
-WM_E_JSON_FAIL if any other error was encountered.

8.17.3.13 int json_release_array_object (jobj_t * *jobj*)

Release a JSON array object

This function expands the scope of searches back to the parent object, if it was previously constrained using json_get_array_object().

Parameters

in,out	*jobj*	Current JSON search object jobj_t. On success, this will be modified such that the scope of future searches will be expanded back to the parent object.

Returns

WM_SUCCESS on success.
-WM_E_JSON_FAIL if any error was encountered.

8.17.3.14 int json_array_get_num_elements (jobj_t * *jobj*)

Get number of elements in an array

This API is valid only if the current scope is an array

Parameters

in	*jobj*	The current JSON object pointer.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

Number of elements in the JSON Array
-WM_FAIL if the current scope is not an array

8.17.3.15 int json_array_get_bool (jobj_t * *jobj*, uint16_t *index*, bool * *value*)

Get JSON bool value from array

Gets the value of a JSON boolean element from an array based on the given index.

Parameters

in	*jobj*	Current JSON search object jobj_t.
in	*index*	Index in the array (beginning from 0).
out	*value*	Pointer to a boolean variable assigned by the application. This will hold the value on success.

Returns

WM_SUCCESS on success
-WM_E_JSON_INVALID_INDEX if the index provided was invalid.
-WM_E_JSON_INVALID_TYPE if an element was found at the given index, but it was not a boolean.
-WM_E_JSON_INVALID_JARRAY if the search object was not a valid JSON array.
-WM_E_JSON_FAIL if any other error was encountered.

8.17.3.16 int json_array_get_int (jobj_t * *jobj*, uint16_t *index*, int * *value*)

Get JSON integer value from array

Gets the value of a JSON integer element from an array based on the given index.

Parameters

in	*jobj*	Current JSON search object jobj_t.
in	*index*	Index in the array (beginning from 0).
out	*value*	Pointer to an integer variable assigned by the application. This will hold the value on success.

Returns

WM_SUCCESS on success
-WM_E_JSON_INVALID_INDEX if the index provided was invalid.
-WM_E_JSON_INVALID_TYPE if an element was found at the given index, but it was not an integer.
-WM_E_JSON_INVALID_JARRAY if the search object was not a valid JSON array.
-WM_E_JSON_FAIL if any other error was encountered.

8.17.3.17 int json_array_get_int64 (jobj_t * *jobj*, uint16_t *index*, int64_t * *value*)

Get 64bit JSON integer value from array

Gets the value of a JSON integer element from an array based on the given index.

Parameters

in	*jobj*	Current JSON search object jobj_t.
in	*index*	Index in the array (beginning from 0).

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

out	*value*	Pointer to a 64bit integer variable assigned by the application. This will hold the value on success.

Returns

WM_SUCCESS on success
-WM_E_JSON_INVALID_INDEX if the index provided was invalid.
-WM_E_JSON_INVALID_TYPE if an element was found at the given index, but it was not an integer.
-WM_E_JSON_INVALID_JARRAY if the search object was not a valid JSON array.
-WM_E_JSON_FAIL if any other error was encountered.

8.17.3.18 int json_array_get_float (jobj_t * *jobj*, uint16_t *index*, float * *value*)

Get JSON float value from array

Gets the value of a JSON float element from an array based on the given index.

Parameters

in	*jobj*	Current JSON search object jobj_t.
in	*index*	Index in the array (beginning from 0).
out	*value*	Pointer to a float variable assigned by the application. This will hold the value on success.

Returns

WM_SUCCESS on success
-WM_E_JSON_INVALID_INDEX if the index provided was invalid.
-WM_E_JSON_INVALID_TYPE if an element was found at the given index, but it was not a float.
-WM_E_JSON_INVALID_JARRAY if the search object was not a valid JSON array.
-WM_E_JSON_FAIL if any other error was encountered.

8.17.3.19 int json_array_get_str (jobj_t * *jobj*, uint16_t *index*, char * *value*, int *maxlen*)

Get JSON string value from array

Gets the value of a JSON string element from an array based on the given index.

Parameters

in	*jobj*	Current JSON search object jobj_t.
in	*index*	Index in the array (beginning from 0).
out	*value*	Pointer to a buffer assigned by the application to hold the string. This will hold the null terminated string on success.
in	*maxlen*	Length of the buffer passed.

Returns

WM_SUCCESS on success
-WM_E_JSON_INVALID_INDEX if the index provided was invalid.
-WM_E_JSON_INVALID_TYPE if an element was found at the given index, but it was not a string.
-WM_E_JSON_INVALID_JARRAY if the search object was not a valid JSON array.
-WM_E_JSON_NOMEM if the buffer provided was insufficient.
-WM_E_JSON_FAIL if any other error was encountered.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.17.3.20 int json_array_get_str_len (jobj_t * *jobj*, uint16_t *index*, int * *len*)

Get JSON string length from array

Gets the length of a JSON string element from an array based on the given index. This is useful when the application does not know what size of buffer to use for json_array_get_str()

Parameters

in	*jobj*	Current JSON search object jobj_t.
in	*index*	Index in the array (beginning from 0).
out	*len*	Pointer to an integer assigned by the application to hold the length of string. Applications will have to use a buffer of size atleast 1 byte more than this length

Returns

WM_SUCCESS on success
-WM_E_JSON_INVALID_INDEX if the index provided was invalid.
-WM_E_JSON_INVALID_TYPE if an element was found at the given index, but it was not a string.
-WM_E_JSON_INVALID_JARRAY if the search object was not a valid JSON array.
-WM_E_JSON_FAIL if any other error was encountered.

8.17.3.21 int json_array_get_composite_object (jobj_t * *jobj*, uint16_t *index*)

Get JSON composite object from array

Gets a composite JSON object from an array based on the given index.

Parameters

in,out	*jobj*	Current JSON search object jobj_t. On success, the jobj will be modified such that the scope of subsequent searches will be limited to this composite object. Use json_array_release_composite_object() to expand the scope back to the parent array.
in	*index*	Index in the array (beginning from 0).

Returns

WM_SUCCESS on success
-WM_E_JSON_INVALID_INDEX if the index provided was invalid.
-WM_E_JSON_INVALID_TYPE if an element was found at the given index, but it was not an object.
-WM_E_JSON_INVALID_JARRAY if the search object was not a valid JSON array.
-WM_E_JSON_INVALID_JOBJ if the object found was not a valid JSON object.
-WM_E_JSON_FAIL if any other error was encountered.

8.17.3.22 int json_array_release_composite_object (jobj_t * *jobj*)

Release a JSON composite object from an array

This function expands the scope of searches back to the parent array, if it was previously constrained using json_array_get_composite_object().

Parameters

in,out	*jobj*	Current JSON search object jobj_t. On success, this will be modified such that the scope of future searches will be expanded back to the parent array.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS on success.
-WM_E_JSON_FAIL if any error was encountered.

8.17.3.23 int json_array_get_array_object (jobj_t * *jobj*, uint16_t *index*, int * *num_elements*)

Get JSON array object from array

Gets a JSON array object from another array based on the given index.

Parameters

in,out	*jobj*	Current JSON search object jobj_t. On success, the jobj will be modified such that the scope of subsequent searches will be limited to this composite object. Use json_array_release_composite_object() to expand the scope back to the parent array.
in	*index*	Index in the array (beginning from 0).
out	*num_elements*	Pointer to an integer assigned by the application. On sucess, this will hold the number of elements in the array found.

Returns

WM_SUCCESS on success
-WM_E_JSON_INVALID_INDEX if the index provided was invalid.
-WM_E_JSON_INVALID_TYPE if an element was found at the given index, but it was not an array.
-WM_E_JSON_INVALID_JARRAY if the search object was not a valid JSON array.
-WM_E_JSON_FAIL if any other error was encountered.

8.17.3.24 int json_array_release_array_object (jobj_t * *jobj*)

Release a JSON array object from an array

This function expands the scope of searches back to the parent array, if it was previously constrained using json_array_get_array_object().

Parameters

in,out	*jobj*	Current JSON search object jobj_t. On success, this will be modified such that the scope of future searches will be expanded back to the parent array.

Returns

WM_SUCCESS on success.
-WM_E_JSON_FAIL if any error was encountered.

8.17.4 Enumeration Type Documentation

8.17.4.1 enum wm_json_errno

Json Error Codes

Enumerator

WM_E_JSON_FAIL Generic JSON parse failure

WM_E_JSON_NOMEM Insufficient memory to hold the results

WM_E_JSON_INVAL Invalid characters in JSON

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

WM_E_JSON_INCOMPLETE Incomplete JSON string

WM_E_JSON_INVALID_JOBJ Invalid JSON object

WM_E_JSON_NOT_FOUND JSON element not found

WM_E_JSON_INVALID_TYPE JSON element of invalid type

WM_E_JSON_INVALID_INDEX Invalid array index given

WM_E_JSON_INVALID_JARRAY Invalid search array provided

WM_E_JSON_INVSTART Invalid json start

WM_E_JSON_OBUF JSON buffer overflow

8.18 keystore.h File Reference

Keystore access module (This module is available only for MW30X devices)

8.18.1 Detailed Description

Keystore is a component of bootloader that is passed by it to application firmware while booting. Usually, bootloader will be encrypted and hence keystore acts as read-only secure storage on flash to store important information such as encryption/decryption keys for secure PSM or any application specific secure information. Bootloader passes on this keystore to application firmware in decrypted form in RAM.

Keystore is basically a packed array of TLV (Type-Length-Value) triplets. There is a header at the start of keystore which is used to validate and traverse keystore. Header is followed by one or more TLV triplet, each of different type and different lengths. TLVs of keystore are divided into two catagories, TLV Types 0-95 are reserved for secure boot process and TLV Types 96-254 are reserved for applications for custom use, including that for secure PSM.

This module provides a simple interface to access keystore from application firmware.

Note

Keystore is a read-only storage and cannot be updated in memory or on flash by application firmware. It can only be done by flashing new bootloader with updated keystore.

8.18.2 Usage

1. Use keystore_get to get a copy of TLV value stored in keystore
2. Use keystore_get_ref to get in-place pointer to TLV value in keystore
3. To iterate through all TLV's stored in keystore

```
uint8_t val[256];
uint8_t type;
uint16_t len;
int ret;

keystore_iterate_start();
while(1) {
    ret = keystore_iterate(&type, sizeof(val), val);
    if (ret == ERR_TLV_NOT_FOUND)
        break;
    if (ret == ERR_TLV_BAD_BUF_LEN) {
        log("Use_larger_val_buffer");
        break;
    }
    len = ret;
    use_tlv(type, len, val);
}
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.18.3 Function Documentation

8.18.3.1 int keystore_get (uint8_t *type*, uint16_t *len*, uint8_t * *value*)

This function retrives a TLV from keystore.

Parameters

in	*type*	TLV type to retrieve. Refer to enum ks_tlv_type for pre-defined values
in	*len*	Length of the buffer in which to copy TLV value
out	*value*	Buffer in which to copy TLV value. Can be NULL to retrive the the length of TLV value

Returns

Length of TLV value if TLV type found (can be 0)
ERR_TLV_BAD_BUF_LEN if buffer length is smaller than TLV value size
ERR_TLV_NOT_FOUND if TLV type not found

8.18.3.2 int keystore_get_ref (uint8_t *type*, uint16_t * *plen*, const uint8_t ** *pvalue*)

This function retrives a TLV from keystore. Unlike keystore_get() this function returns reference of TLV value, rather than copy into a separate buffer.

Note

Application must not modify this TLV value.

Parameters

in	*type*	TLV type to retrieve. Refer to enum ks_tlv_type for pre-defined values
out	*plen*	Length of TLV value (can be 0)
out	*pvalue*	Pointer within keystore to TLV value of size *plen. Can be NULL to retrive the the length of TLV value

Returns

Length of TLV value if TLV type found
ERR_TLV_NOT_FOUND if TLV type not found

8.18.3.3 void keystore_iterate_start ( )

This function enables to iterate over each TLV stored in keystore from beginning. This must be called once before keystore_iterate() or keystore_iterate_ref().

Note

This function also enables to restart iteration from beginning.

8.18.3.4 int keystore_iterate (uint8_t * *ptype*, uint16_t *len*, uint8_t * *value*)

This function retrives each TLV from keystore in the sequence in which they are stored. TLV is copied into the buffer supplied.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Precondition

keystore_iterate_start()

Parameters

out	*ptype*	TLV type retrived. Refer to enum ks_tlv_type for pre-defined values
in	*len*	Length of the buffer in which to copy TLV value
out	*value*	Buffer in which to copy TLV value

Returns

Length of TLV value if TLV type found
ERR_TLV_BAD_BUF_LEN if buffer length is smaller than TLV value size
ERR_TLV_NOT_FOUND if end of keystore reached

8.18.3.5 int keystore_iterate_ref (uint8_t * *ptype*, uint16_t * *plen*, const uint8_t ** *pvalue*)

This function is keystore_get_ref() equivalent of keystore_iterate(). TLV value is returned as reference into keystore rather than separate buffer copy.

Note

Application must not modify this TLV value.

Precondition

keystore_iterate_start()

Parameters

out	*ptype*	TLV type retrived. Refer to enum ks_tlv_type for pre-defined values
out	*plen*	Length of TLV value (can be 0)
out	*pvalue*	Pointer within keystore to TLV value of size *plen

Returns

Length of TLV value if TLV type found
ERR_TLV_NOT_FOUND if end of keystore reached

8.18.4 Enumeration Type Documentation

8.18.4.1 enum ks_tlv_type

Predefined TLV key types for keystore. Note that TLV types must be unique throughout the system to avoid any confusion. Applications must use a unique integer in the range KEY_APP_DATA_START and KEY_APP_DATA_-END as TLV type for its use.

Enumerator

KEY_APP_DATA_START Application specific TLV type must be greater than or equal to this value (= 96)

KEY_APP_DATA_END Application specific TLV type must be less than or equal to this value (= 254)

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.19 led_indicator.h File Reference

LED indicator helper functions.

8.19.1 Function Documentation

8.19.1.1 void led_on (output_gpio_cfg_t *led*)

Switch ON LED

Parameters

in	*led*	The output LED GPIO configuration of type output_gpio_cfg_t

8.19.1.2 void led_off (output_gpio_cfg_t *led*)

Switch OFF LED

Parameters

in	*led*	The output LED GPIO configuration of type output_gpio_cfg_t

8.19.1.3 void led_blink (output_gpio_cfg_t *led*, int *on_duty_cycle*, int *off_duty_cycle*)

Blink the LED with given duty cycle

Parameters

in	*led*	The output LED GPIO configuration of type output_gpio_cfg_t
in	*on_duty_cycle*	Time in millisec for which LED will be ON in blinking cycle
in	*off_duty_cycle*	Time in millisec for which LED will be OFF in blinking cycle

8.19.1.4 int led_get_state (output_gpio_cfg_t *led*)

Get the current LED state

Parameters

in	*led*	The output LED GPIO configuration of type output_gpio_cfg_t

Returns

LED_ON if the LED is On
LED_OFF if the LED is Off
-WM_FAIL if invalid parameter specified

8.19.1.5 static void led_fast_blink (output_gpio_cfg_t *led*) [inline], [static]

Fast Blink the LED

Blink LED with on_duty_cycle = 200ms and off_duty_cycle = 200ms

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*led*	The output LED GPIO configuration of type output_gpio_cfg_t

8.19.1.6 static void led_slow_blink (output_gpio_cfg_t *led*) [inline], [static]

Slow Blink the LED

Blink LED with on_duty_cycle = 1000ms and off_duty_cycle = 1000ms

Parameters

in	*led*	The output LED GPIO configuration of type output_gpio_cfg_t

8.19.2 Enumeration Type Documentation

8.19.2.1 enum led_state

State of the LED

Enumerator

LED_OFF LED is Off

LED_ON LED is On

8.20 mdev.h File Reference

Marvell Device Driver API.

8.20.1 Detailed Description

This provides the Marvell Device abstraction for communicating with various devices.

8.20.2 Usage

Drivers typically register an mdev API using the mdev_register() call. The applications can use this device as follows:

1. open the device using [driver name]_drv_open()
2. perform read and write on the device using [driver name]_drv_read() and [driver name]_drv_write() respectively.
3. close the device [driver name]_drv_close()

8.20.3 Function Documentation

8.20.3.1 uint32_t mdev_register (mdev_t * *dev*)

Parameters

in	*dev*	device handler

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

PASS/FAIL

8.20.3.2 uint32_t mdev_deregister (const char * *name*)

Parameters

in	*name*	name of the device to deregister

Returns

PASS/FAIL

8.20.3.3 mdev_t* mdev_get_handle (const char * *dev_name*)

Parameters

in	*dev_name*	name of the device to get handle

Returns

dev handle to the device

8.21 mdev_acomp.h File Reference

Analog Comparator Driver.

8.21.1 Detailed Description

ACOMP driver provides mdev interface to use 2 analog comparators. Follow the steps below to use ACOMP driver from applications

1. Include Files

Include mdev.h and mdev_acomp.h in your source file.

1. ACOMP driver initialization

Call acomp_drv_init from your application initialization function. This will initialize and register acomp driver with the mdev interface.

1. Open ACOMP device to use

acomp_drv_open should be called once before using ACOMP hardware. Here you will specify ACOMP number (from 0 to 1). The other parameters that are set by default are -

ACOMP_HYSTER_0MV - Hysteresis 0mV for positive input

ACOMP_HYSTER_0MV - Hysteresis 0mV for negative input

LOGIC_LO - Output LOW during inactive state

ACOMP_PWMODE_1 - BIAS Current (Response time)

ACOMP_WARMTIME_16CLK - Warm-up time 16 clock cycles

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

ACOMP_PIN_OUT_SYN - Synchronous pin output

If these defaults need to be changed they should be changed in acomp_drv_open

```
{
    mdev_t *acomp_dev;
    acomp_dev = acomp_drv_open(MDEV_ACOMP<0-1>);
}
```

1. Get result/output of ACOMP

Call acomp_drv_result to get the output of the mentioned comparator.

1. Closing a device.

A call to acomp_drv_close informs the ACOMP driver to release the resources that it is holding.

8.21.2 Function Documentation

8.21.2.1 int acomp_drv_init (ACOMP_ID_Type *id*)

Register ACOMP Device

This function registers acomp driver with mdev interface.

Parameters

in	*id*	ACOMP device to be used.

Returns

WM_SUCCESS on success, error code otherwise.

8.21.2.2 mdev_t* acomp_drv_open (ACOMP_ID_Type *acomp_id*, ACOMP_PosChannel_Type *pos*, ACOMP_NegChannel_Type *neg*)

Open ACOMP Device

This function opens the device for use and returns a handle. This handle should be used for other calls to the ACOMP driver.

Parameters

in	*acomp_id*	ACOMP device id (ACOMP_ID_Type) to be opened.
in	*pos*	Positive channel to be used.
in	*neg*	Negative channel to be used.

Returns

NULL if error, mdev_t handle otherwise.

8.21.2.3 int acomp_drv_result (mdev_t * *mdev_p*)

Get ACOMP Output

This function gets the output of acomp device.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*mdev_p*	mdev_t handle to the driver.

Returns

The output value 0 or 1

8.21.2.4 int acomp_drv_close (mdev_t * *dev*)

Close ACOMP Device

This function closes the device after use and frees up resources.

Parameters

in	*dev*	mdev handle to the driver to be closed.

Returns

WM_SUCCESS on success, -WM_FAIL on error.

8.21.2.5 int acomp_drv_deinit (ACOMP_ID_Type *acomp_id*)

De-initialize ACOMP driver port

This function de-initializes ACOMP driver port. Note that if an ACOMP device has been opened using acomp_drv_open(), it should be first closed using acomp_drv_close() before calling this function.

Parameters

in	*acomp_id,:*	Port ID of ACOMP.

Returns

-WM_FAIL if operation failed.
WM_SUCCESS if operation was a success.

8.22 mdev_adc.h File Reference

Analog to Digital Converter Driver.

8.22.1 Detailed Description

ADC driver provides mdev interface to use the internal Analog to Digital Converter unit/s. Follow the steps below to use ADC driver from applications

1. Include Files

Include mdev.h and mdev_adc.h in your source file.

1. ADC driver initialization

Call adc_drv_init from your application initialization function. This will initialize and register adc driver with the mdev interface.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

1. Open ADC device to use

adc_drv_open() should be called once before using ADC hardware. Here you will specify ADC number (from 0 to 1). The other parameters that are set by default are -

ADC_RESOLUTION_16BIT - Resolution 16 bit

ADC_VREF_INTERNAL - Internal 1.2V reference

ADC_GAIN_1 - PGA gain set to 1

ADC_CLOCK_DIVIDER_32 - ADC clock divided by 32

ADC_BIAS_FULL - Full biasing mode

If these defaults need to be changed they should be changed in adc_drv_open()

```
{
    mdev_t *adc_dev;
    adc_dev = adc_drv_open(ADC0_ID);
}
```

1. Get result/output of ADC

Call adc_drv_result() to get the output of the mentioned adc.

1. Closing a device.

A call to adc_drv_close informs the ADC driver to release the resources that it is holding.

8.22.2 Function Documentation

8.22.2.1 int adc_drv_init (ADC_ID_Type *adc_id*)

Register ADC Device

This function registers adc driver with mdev interface.

Parameters

in	*adc_id*	The ADC to be initialized.

Returns

WM_SUCCESS on succes, error code otherwise.

8.22.2.2 void adc_get_config (ADC_CFG_Type * *config*)

Get adc driver configuration

This function returns a structure with ADC parameters as its members. If the default parameters are changed by the user, this function returns the structure with current parameters.

Parameters

in	*config*	Pointer to ADC config structure

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.22.2.3 void adc_modify_default_config (adc_cfg_param *config_type*, int *value*)

Modify default ADC configuration

This function allows the user to over-ride the default parameter/s Call to this function is not mandatory and should be made if the user wants to change any of the defaults for ADC. It can change a single parameter in each call. User may choose to modify one or more default parameters. All the un-modified parameter are set to the default value. Default values are as below: Config-Type | Default value adcResolution = ADC_RESOLUTION_16BIT, adcVref-Source = ADC_VREF_INTERNAL, adcGainSel = ADC_GAIN_1, adcClockDivider = ADC_CLOCK_DIVIDER_32, adcBiasMode = ADC_BIAS_FULL

Parameters

in	*config_type*	ADC Config-Type
in	*value*	Value for the chosen Config-Type

8.22.2.4 mdev_t* adc_drv_open (ADC_ID_Type *adc_id*, ADC_ChannelSource_Type *channel*)

Open ADC Device

This function opens the device for use and returns a handle. This handle should be used for other calls to the ADC driver.

Parameters

in	*adc_id*	adc port of the driver to be opened.
in	*channel*	Channel to be used.

Returns

NULL if error, mdev_t handle otherwise.

8.22.2.5 int adc_drv_result (mdev_t * *mdev_p*)

Get ADC Output

This function gets the output of adc device.

Parameters

in	*mdev_p*	mdev_t handle to the driver.

Returns

The output value (max 16 bit)

8.22.2.6 void adc_drv_timeout (ADC_ID_Type *id*, uint32_t *tout*)

Set timeout for ADC driver This function allows user to set timeout for ADC conversion. This time is time required for ADC conversion for a single chunk (i.e. maximum 1023 datapoints) and dma transfer time. If user does not set the timeout, default value is used. By default this function waits till the DMA_TRANS_COMPLETE interrupt occurs. Note: To set timeout this function needs to be called before adc_drv_init() call.

Parameters

in	*id*	ADC ID.
in	*tout*	The timeout value.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.22.2.7 int adc_drv_get_samples (mdev_t * *mdev_p*, uint16_t * *buf*, int *num*)

Get ADC samples for user supplied analog input

This function provides n samples for user supplied analog input (n specified by the user).It internally uses DMA to transfer data to the user buffer.The DMA transfer takes place in chunks of data-points. There is no limit on number of samples that can be transferred. Each ADC sample by default uses 16 bit resolution.

Parameters

in	*mdev_p*	mdev handle to the driver
in,out	*buf*	pointer to user allocated buffer to store sample points
in	*num*	number of samples required

Returns

-WM_FAIL if operation failed.
WM_SUCCESS if operation was a success.

8.22.2.8 int adc_drv_close (mdev_t * *dev*)

Close ADC Device

This function closes the device after use and frees up resources.

Parameters

in	*dev*	mdev handle to the driver to be closed.

Returns

WM_SUCCESS on success, -WM_FAIL on error.

8.22.2.9 void adc_drv_deinit (ADC_ID_Type *adc_id*)

De-initialize ADC Driver

This function de-initializes ADC driver. Note that if an ADC device has been opened using adc_drv_open(), it should be first closed using adc_drv_close() before calling this function.

Parameters

in	*adc_id*	Port ID of ADC.

8.23 mdev_aes.h File Reference

AES Driver.

8.23.1 Detailed Description

The AES driver handles the AES hardware accelerator and provides an API to the user for AES encryption and decryption.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Note

This AES driver does not support interrupt based AES operations.

Follow the steps below to use AES driver from applications

1. AES driver initialization

Call aes_drv_init from your application initialization function. This will initialize and register aes driver with the mdev interface.

1. Open AES device to use

aes_drv_open should be called once before using AES hardware for AES operations. Here you will specify the name of the AES module. There is only one hardware engine. Hence, you should pass MDEV_AES_0 here.

Typical example:

Opening AES device

```
{
    mdev_t *aes_dev;
     aes_dev = aes_drv_open("MDEV_AES_0");
}
```

1. Perform AES operations.

AES API can is used for encryption and decryption of plain text and cipher text respectively. Before any operation first a key, relevant to that operation needs to be set.

Note

Please ensure that the size of the plain/cipher text sent to the encryption/decryption functions are aligned to AES_BLOCK_SIZE

Typical example:

Encryption:

```
{
    unsigned char const key[] = {
     0xc2, 0x86, 0x69, 0x6d,
     0x88, 0x7c, 0x9a, 0xa0,
      0x61, 0x1b, 0xbb, 0x3e,
      0x20, 0x25, 0xa4, 0x5a
  };

unsigned char const initial_vector[] = {
    0x56, 0x2e, 0x17, 0x99,
     0x6d, 0x09, 0x3d, 0x28,
    0xdd, 0xb3, 0xba, 0x69,
    0x5a, 0x2e, 0x6f, 0x58
   };

char plain[1024];
char cipher[1024];

     aes_t enc_aes;
     aes_drv_init();
     mdev_t *aes_dev = aes_drv_open(MDEV_AES_0);
     aes_drv_setkey(aes_dev, &enc_aes, &key, sizeof(key),
           &initial_vector, AES_ENCRYPTION, HW_AES_MODE_CBC);

     aes_drv_encrypt(aes_dev, &enc_aes, plain, cipher, sizeof(plain));
     aes_drv_close(aes_dev);
}
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

1. Closing a device.

A call to aes_drv_close informs the AES driver to release the resources that it is holding. It is always good practice to free the resources after use.

Typical usecase:

```
aes_drv_close(aes_dev);
```

8.23.2 Function Documentation

8.23.2.1 int aes_drv_init (void)

Init AES Driver

This function registers the AES driver with the mdev interface.

Returns

WM_SUCCESS on success, error code otherwise.

8.23.2.2 mdev_t* aes_drv_open (const char * *name*)

Open AES Device

This function opens the device for use and returns a handle. This handle should be used for other calls to the AES driver.

Parameters

in	*name*	Name of the driver to be opened. It should be MDEV_AES_0.

Returns

NULL if error, mdev_t handle otherwise.

8.23.2.3 int aes_drv_setkey (mdev_t * *dev*, aes_t * *aes*, const uint8_t * *userKey*, uint32_t *key_len*, const uint8_t * *IV*, int *dir*, hw_aes_mode_t *hw_mode*)

Set the AES key

Parameters

in	*dev*	pointer returned in earlier call to aes_drv_open
in	*aes*	Pointer to the structure allocated by **caller**. Please do not touch this structure after return from aes_drv_setkey. Pass it unchanged to aes_drv_encrypt or aes_drv_decrypt
in	*userKey*	AES key. Can either is 16 bytes, 24 bytes or 32 bytes. Any other value is invalid.
in	*key_len*	The length of the key in bytes.
in	*IV*	The initialization vector.
in	*dir*	Is either AES_ENCRYPTION or AES_DECRYPTION
in	*hw_mode*	One of the enum hw_aes_mode_t

Returns

Standard SDK return codes.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.23.2.4 int aes_drv_ccm_setkey (mdev_t * *dev*, aes_t * *aes*, const uint8_t * *userKey*, uint32_t *key_len*)

Set the AES key for CCM.

Parameters

in	*dev*	pointer returned in earlier call to aes_drv_open
in	*aes*	Pointer to the structure allocated by **caller**. Please do not touch this structure after return from aes_drv_ccm_setkey. Pass it unchanged to aes_drv_ccm_-encrypt or aes_drv_ccm_decrypt
in	*userKey*	AES key. Can either is 16 bytes, 24 bytes or 32 bytes. Any other value is invalid.
in	*key_len*	The length of the key in bytes.

Returns

Standard SDK return codes.

8.23.2.5 int aes_drv_encrypt (mdev_t * *dev*, aes_t * *aes*, const uint8_t * *plain*, uint8_t * *cipher*, uint32_t *sz*)

Encrypt the given plain text based on the key set earlier.

Parameters

in	*dev*	The pointer returned in earlier call to aes_drv_open
in	*aes*	The same object sent to the function aes_drv_setkey unchanged
in	*plain*	Plain text
out	*cipher*	Cipher text returned by this function.
in	*sz*	The total length of the plain text

Returns

Standard SDK return codes.

8.23.2.6 int aes_drv_ccm_encrypt (mdev_t * *dev*, aes_t * *aes*, const char * *plain*, char * *cipher*, uint32_t *sz*, const char * *nonce*, uint32_t *nonceSz*, char * *authTag*, uint32_t *authTagSz*, const char * *authIn*, uint32_t *authInSz*)

Encrypt the given plain text based on the key set earlier.

Parameters

in	*dev*	The pointer returned in earlier call to aes_drv_open
in	*aes*	The same object sent to the function aes_drv_ccm_setkey unchanged
in	*plain*	Plain text to be encrypted
out	*cipher*	Cipher text.
in	*sz*	The total length of the plain text in bytes
in	*nonce*	The initialization vector
in	*nonceSz*	The size in bytes of the init vector. In CCM mode, length of nonce ranges from 11 bytes to 13 bytes.
out	*authTag*	The MIC will be returned in this buffer
in	*authTagSz*	Size in bytes of MIC required.
in	*authIn*	The additional authenticated data (AAD).
in	*authInSz*	Size in bytes of the AAD.

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

Standard SDK return codes.

Note

Iterative calls for encrypting data in chunked fashion is not supported by this function, i.e. entire input (AAD + Plain Text) should be provided in single call.

8.23.2.7 int aes_drv_decrypt (mdev_t * *dev*, aes_t * *aes*, const uint8_t * *cipher*, uint8_t * *plain*, uint32_t *sz*)

Decrypt the given cipher text based on the key set earlier.

Parameters

in	*dev*	The pointer returned in earlier call to aes_drv_open
in	*aes*	The same object sent to the function aes_drv_setkey unchanged
in	*cipher*	Cipher text.
out	*plain*	Plain text returned by this function.
in	*sz*	The total length of the plain text

Returns

Standard SDK return codes.

8.23.2.8 int aes_drv_ccm_decrypt (mdev_t * *dev*, aes_t * *aes*, const char * *cipher*, char * *plain*, uint32_t *sz*, const char * *nonce*, uint32_t *nonceSz*, const char * *authTag*, uint32_t *authTagSz*, const char * *authIn*, uint32_t *authInSz*)

Decrypt the given cipher text based on the key set earlier.

Parameters

in	*dev*	The pointer returned in earlier call to aes_drv_open
in	*aes*	The same object sent to the function aes_drv_ccm_setkey unchanged
in	*cipher*	Cipher text.
out	*plain*	Plain text returned by this function.
in	*sz*	The total length of the cipher text in bytes
in	*nonce*	The initialization vector
in	*nonceSz*	The size in bytes of the init vector. In CCM mode, length of nonce ranges from 11 bytes to 13 bytes.
in	*authTag*	The input MIC for verification
in	*authTagSz*	Size in bytes of input MIC
in	*authIn*	The additional authenticated data (AAD).
in	*authInSz*	Size in bytes of the AAD.

Returns

Standard SDK return codes.

Note

Iterative calls for decrypting data in chunked fashion is not supported by this function, i.e. entire input (AAD + Cipher Text + MIC) should be provided in single call.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.23.2.9 void aes_ctr_final (aes_t * *aes*)

Function clears any buffered data for unaligned block encryption to start encryption of an entirely new block.

Parameters

in	*aes*	The pointer to the aes_t structure.

8.23.2.10 void aes_drv_close (mdev_t * *dev*)

Close the device instance.

Parameters

in	*dev*	The pointer returned in earlier call to aes_drv_open

Note

Calling this function (after your work with AES engine is over) is essential. Failing to do so will result in failure of the AES driver.

Returns

void

8.23.3 Enumeration Type Documentation

8.23.3.1 enum hw_aes_mode_t

Mode in which HW AES is used

Enumerator

HW_AES_MODE_ECB AES mode: ECB

HW_AES_MODE_CBC AES mode: CBC

HW_AES_MODE_CTR AES mode: CTR

HW_AES_MODE_CCM AES mode: CCM

HW_AES_MODE_MMO AES mode: MMO

HW_AES_MODE_BYPASS AES mode: Bypass

8.24 mdev_crc.h File Reference

CRC Driver.

8.24.1 Detailed Description

The CRC driver handles the CRC hardware engine and provides an API to the user for calculation of CRC.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Note

This CRC driver does not support interrupt based CRC calculations.

Follow the steps below to use CRC driver from applications

1. CRC driver initialization

Call crc_drv_init from your application initialization function. This will initialize and register crc driver with the mdev interface.

1. Open CRC device to use

crc_drv_open should be called once before using CRC hardware for CRC calculations. Here you will specify the name of the CRC module. There is only one hardware engine. Hence, you should pass MDEV_CRC_0 here. Apart from name you also need to pass the CRC mode. The modes are mentioned in the below enum crc_mode_t

Typical example:

Opening CRC device

```
{
    mdev_t *crc_dev;
     crc_dev = crc_drv_open("MDEV_CRC_ENGINE_0", CRC32_IEEE);
}
```

1. Perform CRC calculations.

CRC API can be used in two ways, viz block and stream. Which one to use depends on the data availability. If the entire data (over which CRC is to be calculated) is available, use the block operation API crc_drv_block_calc. This API will do the job in one go. However, if you want to calculate the CRC on your data part by part (maybe because only part of the data is available now) then use the stream based API.

Note

The stream API has a peculiar quirk. It can handle non word (32-bit) aligned data. But, only the first call (among many) to the function crc_drv_stream_feed can have non word aligned length. In all the remaining calls to crc_drv_stream_feed the len should be word aligned.

Typical example:

Block based operation:

```
{
      uint32 crc32;
      char buf[100];
      crc_drv_block_calc(crc_dev, buf, 100, &crc32);
      wmprintf("The CRC32 result is: 0x%x\n\r", crc32);
}
```

Stream based operation (word aligned data):

```
{
     const uint32_t total_data_size = 1024;
     uint32_t total_read_data = 0;
     uint32_t crc32;
     char buf[16];
     int size_read;
     crc_drv_stream_setlen(crc_dev, total_data_size);
     while(total_read_data < total_data_size) {
            xxx_fetch_data(&buf, sizeof(buf));
            crc_drv_stream_feed(crc_dev, buf, sizeof(buf));
            total_read_data += sizeof(buf);
     }
     crc_drv_stream_get_result(crc_dev, &crc32);
     wmprintf("CRC result: 0x%x\n\r", crc32);
}
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Stream based operation (non word aligned data): We know that the data size is non-word aligned. So, we will call crc_drv_stream_get_result first time with non-aligned word data. All the later calls would be word aligned.

```
{
    const uint32_t total_data_size = 17;
    uint32_t remaining_size;
    uint32_t crc32;
    char buf[16];
    int size_read;
    crc_drv_stream_setlen(crc_dev, total_data_size);

    int tail = total_data_size % 4;
    xxx_fetch_data(&buf, tail);
    crc_drv_stream_feed(crc_dev, buf, tail);
    remaining_size -= tail;

    xxx_fetch_data(&buf, remaining_size);
    crc_drv_stream_feed(crc_dev, buf, remaining_size);
    crc_drv_stream_get_result(crc_dev, &crc32);
    wmprintf("CRC result: 0x%x\n\r", crc32);
}
```

1. Closing a device.

A call to crc_drv_close informs the CRC driver to release the resources that it is holding. It is always good practice to free the resources after use.

Typical usecase:

```
crc_drv_close(crc_dev);
```

8.24.2 Function Documentation

8.24.2.1 int crc_drv_init (void)

Init CRC Driver

This function registers the CRC driver with the mdev interface.

Returns

WM_SUCCESS on success
error code otherwise.

8.24.2.2 mdev_t* crc_drv_open (const char * *name*, crc_mode_t *mode*)

Open CRC Device

This function opens the device for use and returns a handle. This handle should be used for other calls to the CRC driver.

Parameters

in	*name*	Name of the driver(MDEV_CRC_ENGINE_0) to be opened.
in	*mode*	The mode of operation of the CRC engine. One of the enum crc_mode_t

Returns

NULL if error, mdev_t handle otherwise.

8.24.2.3 int crc_drv_block_calc (mdev_t * *dev*, uint8_t * *buf*, uint32_t *len*, uint32_t * *result*)

Calculate CRC in a block operation.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*dev*	The pointer returned in earlier call to crc_drv_open
in	*buf*	The buffer holding the data whose CRC is to be calculated.
in	*len*	The length of the data in buffer.
out	*result*	The calculated CRC

Returns

Standard SDK return codes.

8.24.2.4 int crc_drv_stream_setlen (mdev_t * *dev*, uint32_t *len*)

Set length of the stream data. API for CRC stream based operation only.

Parameters

in	*dev*	The pointer returned in earlier call to crc_drv_open
in	*len*	The total length of the stream data

Returns

Standard SDK return codes.

8.24.2.5 int crc_drv_stream_feed (mdev_t * *dev*, uint8_t * *buf*, uint32_t *len*)

Feed stream data to CRC driver. API for CRC stream based operation only.

Parameters

in	*dev*	The pointer returned in earlier call to crc_drv_open
in	*buf*	The buffer holding the data whose CRC is to be calculated.
in	*len*	The length of the data in buffer.

Returns

Standard SDK return codes.

8.24.2.6 int crc_drv_stream_get_result (mdev_t * *dev*, uint32_t * *result*)

Get the result of the stream data CRC. API for CRC stream based operation only.

Parameters

in	*dev*	The pointer returned in earlier call to crc_drv_open
out	*result*	The calculated CRC

Returns

Standard SDK return codes.

8.24.2.7 void crc_drv_close (mdev_t * *dev*)

Close the device instance.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*dev*	The pointer returned in earlier call to crc_drv_open

Note

Calling this function (after your work with CRC engine is over) is essential. Failing to do so will result in failure of the CRC driver.

Returns

void

8.24.3 Enumeration Type Documentation

8.24.3.1 enum crc_mode_t

Emum to maintain various modes of CRC operation.

The below enum members also each have a comment above them mentioning the exact polynomial used by the hardware for CRC computation.

8.25 mdev_dac.h File Reference

Digital Analog Converter Driver.

8.25.1 Detailed Description

DAC driver provides mdev interface to use Digital to Analog Converter unit/s. Follow the steps below to use DAC driver from applications

1. Include Files

Include mdev.h and mdev_dac.h in your source file.

1. DAC driver initialization

Call dac_drv_init from your application initialization function. This will initialize and register dac driver with the mdev interface.

1. Open DAC device to use

dac_drv_open should be called once before using DAC hardware. Here you will specify DAC channel (A or B). The other parameters that are set by default are -

DAC_WAVE_NORMAL - No predefined waveform

DAC_RANGE_LARGE - Output Range Large

DAC_OUTPUT_PAD - Output is enable to pad

DAC_NON_TIMING_CORRELATED - Non timing corelated mode

If these defaults need to be changed they should be changed in dac_drv_open

```
{
    mdev_t *dac_dev;
    dac_dev = dac_drv_open(MDEV_DAC, DAC_CH_A);
}
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

1. Set output of DAC

Call dac_drv_output to get the output of the mentioned dac. Max resolution is 10 bit.

1. Closing a device.

A call to dac_drv_close informs the DAC driver to release the resources that it is holding.

1. De-register the DAC driver

A call to dac_drv_deinit informs DAC driver to de-register the DAC driver.

8.25.2 Function Documentation

8.25.2.1 int dac_drv_init ( )

Register DAC Device

This function registers dac driver with mdev interface.

Returns

WM_SUCCESS on success, error code otherwise.

8.25.2.2 void dac_modify_default_config (dac_cfg_param *config_type*, int *value*)

Modify the default DAC configuration

This function allows user to over-ride the default DAC configuration. A call to thif function is not mandatory and should be used only if the user is aware of its impact.

This function should be used after dac_drv_init() but before dac_drv_open().

Only one paramater can be changed in a single call. Use this function multiple times if multiple parameters need to be changed. All un-modified parameters are set to their default values. The default values are as below: Config-Type | Default value waveMode = DAC_WAVE_NORMAL outMode = DAC_OUTPUT_PAD timingMode = DAC_NON_TIMING_CORRELATED

Parameters

in	*config_type*	The config option to be changed. Can be waveMode, outMode or timingMode.
in	*value*	The desired value for the parameter.

8.25.2.3 mdev_t* dac_drv_open (const char * *name*, DAC_ChannelID_Type *channel*)

Open DAC Device

This function opens the device for use and returns a handle. This handle should be used for other calls to the DAC driver.

Parameters

in	*name*	Name of the driver(MDEV_DAC) to be opened.
in	*channel*	Channel to be used.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

NULL if error, mdev_t handle otherwise.

8.25.2.4 void dac_drv_output (mdev_t * *mdev_p*, int *val*)

Set DAC Output

This function sets the output of dac device.

Parameters

in	*mdev_p*	mdev_t handle to the driver.
in	*val*	Max 10 bit value to be set.

8.25.2.5 int dac_drv_close (mdev_t * *dev*)

Close DAC Device

This function closes the device after use and frees up resources.

Parameters

in	*dev*	mdev handle to the driver to be closed.

Returns

WM_SUCCESS on success, -WM_FAIL on error.

8.25.2.6 void dac_drv_deinit ( )

De-register DAC device This function de-registers the DAC driver.

8.26 mdev_dma.h File Reference

DMA Module.

8.26.1 Detailed Description

mdev_dma is a module that handles DMA driver functionality a layer above the low-level driver code. It establishes a mechanism to deploy DMA capabilities to all peripherals, possibly simultaneously.

The DMAC hardware supports 32 DMA channels. By default, four channels can operate simultaneously. User may configure the value of NUM_DMA_CHANNELS to modify the number of supported channels.

This module provides a set of APIs to use DMA. DMA transfer can be used for the following modes.

1. MEMORY_TO_PERIPHERAL
2. PERIPHERAL_TO_MEMORY
3. MEMORY_TO_MEMORT

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Note

PERIPHERAL_TO_PERIPHERAL transfer is not supported.
DMA driver limitations

1. DMAC supports a maximum of 8191 byte data transfer with the added restriction that SOURCE and DESTINATION ADDRESSES should align with the transfer width.

2. DMA transfers will be asynchronous.

   A typical DMA usage scenario is as follows:

   (a) Initialize the DMA driver with a call dma_drv_init()

   (b) Open a channel for DMA transfer using dma_drv_open()

   (c) User needs to configure dma_config_type.

   (d) [Optional] Call dma_drv_set_cb to set callback for a specific dma channel.

   (e) Call dma_drv_transfer dma_drv_transfer with appropriate parameters.

   (f) Close the DMM channel, on transfer complete.

   (g) de-initialize DMA driver provided no code is using DMA.

Code snippet: Following code demonstrates how to use DMA driver API to generate and handle an interrupt

```
static void dma_cb(DMA_Channel_Type channel, void *data)
{
     process_dma_interrupt();
}

{
     int err = dma_drv_init();
     if (err != WM_SUCCESS)
        err_handling();

     mdev_t *dma_dev = dma_drv_open();
     if (dma_dev == NULL)
        err_handling();

[optional] dma_drv_set_cb(dma_dev, dma_cb);

     err = dma_drv_transfer(dma_dev, &dma_c);
     if (err != WM_SUCCESS)
        err_handling();

     dma_drv_wait_for_transfer_complete();
}
```

8.26.2 Function Documentation

8.26.2.1 int dma_drv_init ( )

DMA driver initialization.

This API initializes the DMA driver.

Returns

-WM_FAIL in case of failure to register the DMA driver
WM_SUCCESS if DMA driver registration is successful.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.26.2.2 mdev_t* dma_drv_open ( )

Open a dma channel.

This API selects a channel for DMA transfer. It will select a free channel and reserve the same till dma_drv_-close() is explicitly called.

Precondition

dma_drv_init()

Returns

handle to dma channel. It is channel number type-casted to pointer of type mdev_t.
-NULL in case the API fails to select a DMA channel

Note

Any available channel can be used for data transfer There is no mapping between DMA channel and peripheral. However correct handshaking interface must be chosen during memory to/from peripheral data transfer.

8.26.2.3 int dma_drv_set_cb (mdev_t * *mdev_dma,* dma_irq_cb *dma_cb,* void * *data*)

Set callback for a specific DMA channel on DMA interrupt.

This API registers a callback on a specific DMA interrupt for a given DMA channel.

Parameters

in	*mdev_dma*	pointer to mdev_dma
in	*dma_cb*	function to be called when DMA interrupt occurs
in	*data*	user's private pointer that is passed as a parameter to the callback function

Returns

WM_SUCCESS on success
-WM_E_INVAL on failure to register callback or in case of invalid parameters.

8.26.2.4 int dma_drv_transfer (mdev_t * *mdev_dma,* dma_config_t * *dmac*)

Configure & enable a particular dma channel.

This API configures a DMA channel and enables it to begin with the data transfer.

Precondition

dma_drv_open()

Parameters

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

in	*mdev_dma*	handle pointer to dma channel
in	*dmac*	pointer to dma_config_type

Returns

-WM_E_INVAL in case of invalid parameters
WM_SUCCESS on success

Note

Since DMA Controller handles transmit & receive in the exact same manner; no separate read/write calls are built. A single API, dma_transfer_start() is maintained for all types of data transfers. The API configures a channel and starts data transfer. In order to check the status of data transfer, user should use the API dma_drv_wait_for_transfer_complete().

8.26.2.5 int dma_drv_wait_for_transfer_complete (mdev_t * *mdev_dma,* uint32_t *timeout*)

Wait for transfer complete on a perticular dma channel.

This API waits for a specified time for DMA transfer to complete. It also notifies the user if an error occurs during DMA transfer. The error may be a DMA bus error or address alignment error.

Parameters

in	*mdev_dma*	handle pinter to dma channel
in	*timeout*	can be a specific time or OS_NO_WAIT OS_NO_WAIT or OS_WAIT_FOREVER OS_WAIT_FOREVER

Returns

-WM_E_INVAL in case of invalid parameters
-WM_E_AGAIN in case of timeout
WM_SUCCESS on success

Note

It is mandatory to call dma_drv_wait_for_transfer_complete() after dma_drv_transfer() and to check the status of data transfer.

8.26.2.6 int dma_drv_close (mdev_t * *mdev_dma*)

Close DMA channel.

This API closes the DMA channel specified by the given handle. The resources associated with the channel are freed up. This channel now can be selected by dma_drv_open for another DMA transfer. This API also de-registers the registered callbacks on the channel.

Parameters

in	*mdev_dma*	handle to DMA channel

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS if channel is closed successfully
-WM_E_INVAL if the parameters are invalid

8.26.2.7 void dma_drv_deinit ( )

De-initialize DMA driver.

This API de-registers the DMA driver. All resources associated with memory and DMA driver are freed up.

8.27 mdev_fpu.h File Reference

Cortex M4 Floating Point Unit.

8.27.1 Macro Documentation

8.27.1.1 #define fpu_init( ) FPU_Init()

Enable Full Access to FPU

The function enables the CP10 and CP11 for full access to FPU

Note

This API should be used only with Mw300 based platform

8.27.1.2 #define fpu_deinit( ) FPU_Deinit()

Disable Access to FPU

The function disables the CP10 and CP11 for any access to FPU. Any attempted access generates a NOCP Usage-Fault.

Note

This API should be used only with Mw300 based platform

8.28 mdev_gpio.h File Reference

General Purpose Input Output (GPIO) Driver.

8.28.1 Detailed Description

The GPIO driver is used to handle I/O requests to read/write GPIO pins. GPIO pins are used for communication with external entities. They are used for sending control signals like chip select and clock.

8.28.2 Usage

A typical GPIO device usage scenario is as follows:

1. Initialize the GPIO driver using gpio_drv_init().

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

2. Open the GPIO device handle using gpio_drv_open() call.
3. Use gpio_drv_setdir() to set direction of GPIO pin (input or output).
4. Use gpio_drv_read() or gpio_drv_write() to read GPIO state or to write GPIO pin state respectively.
5. GPIO pins can be used to generate interrupts and interrupt callback is registered or unregistered using gpio_drv_set_cb().
6. Close the GPIO device using gpio_drv_close() after its use.

   gpio_drv_close() informs the GPIO driver to release the resources that it is holding. It is always good practice to free the resources after use.

Code snippet:

Following code demonstrates how to use GPIO driver APIs to set a pin as output and set it high.

```
{
   gpio_drv_init();

       .
       .
       .
  mdev_t* gpio_dev = gpio_drv_open("MDEV_GPIO");
  if (gpio_dev == NULL)
     err_handling();
  int err = gpio_drv_setdir(gpio_dev, pin, GPIO_IO_OUTPUT);
  if (err != WM_SUCCESS)
     err_handling();

  err = gpio_drv_write(gpio_dev, pin, GPIO_IO_HIGH);
  if (err != WM_SUCCESS)
     err_handling();
  gpio_drv_close(gpio_dev);
  return;
}
```

Following code demonstrates how to use GPIO driver APIs to read a pin state.

```
{
  int err = gpio_drv_init();
  if (err != WM_SUCCESS)
     err_handling();
       .
       .
       .
  mdev_t* gpio_dev = gpio_drv_open("MDEV_GPIO");
  if (gpio_dev == NULL)
     err_handling();
  int state;
  int err = gpio_drv_read(gpio_dev, pin, &state);
  if (err != WM_SUCCESS)
     err_handling();
  gpio_drv_close(gpio_dev);
  return;
}
```

Following code demonstrates how to use GPIO driver APIs to generate and handle an interrupt.

```
static void gpio_cb()
{
    process_interrupt_of_gpio();
}

{
  int err = gpio_drv_init();
  if (err != WM_SUCCESS)
     err_handling();

       .
       .
       .
  mdev_t* gpio_dev = gpio_drv_open("MDEV_GPIO");
  if (gpio_dev == NULL)
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

```
    err_handling();

  int err = gpio_drv_setdir(gpio_dev, pin, GPIO_IO_INPUT);
  if (err != WM_SUCCESS)
     err_handling();

  err = gpio_drv_set_cb(gpio_dev, pin, GPIO_INT_FALLING_EDGE,
                        gpio_cb);
  if (err != WM_SUCCESS)
     err_handling();

  gpio_drv_close(gpio_dev);
  return;
}
```

Note

1. Locking mechanism is implemented to provide atomic access.
2. The APIs assume that caller passed correct GPIO pin number.
3. GPIO_Int_Type enumeration has following possible values GPIO_INT_RISING_EDGE , GPIO_INT_FALLING_EDGE, GPIO_INT_DISABLE, GPIO_INT_DISABLE.
4. A GPIO pin state can have only two possible values GPIO_IO_LOW and GPIO_IO_HIGH.
5. A GPIO pin direction can be either GPIO_INPUT or GPIO_OUTPUT.

8.28.3 Function Documentation

8.28.3.1 int gpio_drv_init (void)

Initialize the GPIO driver

This function initializes GPIO driver and registers it with mdev interface.

Returns

WM_SUCCESS on success
Error code otherwise

8.28.3.2 mdev_t* gpio_drv_open (const char * *name*)

Open GPIO device driver

This function opens the handle to GPIO device and enables application to use the device. This handle should be used in other calls related to GPIO device.

Parameters

name	Name of mdev gpio driver. It should be "MDEV_GPIO" string.

Returns

handle to driver on success
NULL otherwise

8.28.3.3 int gpio_drv_close (mdev_t * *dev*)

Close the GPIO device

This function closes the handle to GPIO device.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*dev*	Handle to the GPIO device returned by gpio_drv_open().

Returns

WM_SUCCESS on Success

8.28.3.4 int gpio_drv_setdir (mdev_t * *dev*, int *pin*, int *dir*)

Set direction of GPIO pin

This function allows caller to set direction of a GPIO pin.

Parameters

in	*dev*	Handle to the GPIO device returned by gpio_drv_open().
in	*pin*	Pin Number(GPIO_XX) of GPIO pin (example: GPIO_12)
in	*dir*	Either GPIO_INPUT or GPIO_OUTPUT

Returns

WM_SUCCESS on Success
Error code otherwise

8.28.3.5 int gpio_drv_write (mdev_t * *dev*, int *pin*, int *val*)

Set GPIO pin state

This function allows caller to set GPIO pin's state to high or low.

Parameters

in	*dev*	Handle to the GPIO device returned by gpio_drv_open().
in	*pin*	Pin Number(GPIO_XX) of GPIO pin
in	*val*	Value to be set. GPIO_IO_LOW(0) or GPIO_IO_HIGH(1)

Returns

WM_SUCCESS on success
Error code otherwise

8.28.3.6 int gpio_drv_read (mdev_t * *dev*, int *pin*, int * *val*)

Read GPIO pin status

This function allows caller to get GPIO pin's state.

Parameters

in	*dev*	Handle to the GPIO device returned by gpio_drv_open().
in	*pin*	Pin Number(GPIO_XX) of GPIO pin (example: GPIO_12)
out	**val*	Value read

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS on success
Error code otherwise

8.28.3.7 int gpio_drv_set_cb (mdev_t * *dev*, int *pin*, GPIO_Int_Type *type*, void * *data*, gpio_irq_cb *gpio_cb*)

Register CallBack for GPIO Pin Interrupt

Parameters

in	*dev*	Handle to the GPIO device returned by gpio_drv_open().
in	*pin*	Pin Number(GPIO_XX) of GPIO pin
in	*type*	Possible values from GPIO_Int_Type
in	*data*	Pointer to a caller's data structuree
in	*gpio_cb*	Function to be called when said interrupt occurs. If NULL the callback will be de-registered.

Note

intCallback_Type is defined as typedef void (intCallback_Type)(void); GPIO pin's direction should be set as input to generate an interrupt.

Returns

WM_SUCCESS on success
-WM_FAIL if invalid parameters

8.28.3.8 int gpio_drv_get_io_domain (int *pin*)

Get the voltage IO domain for a given pin

The different GPIOs of the MCU are part of different voltage domains. These domains have significance in power management wherein some domains can be turned off and only the required ones can be kept on. This API can be used to get the domain for a particular GPIO so that the application can decide whether it wants to keep that domain On or Off.

Parameters

in	*pin*	the GPIO pin number

Returns

The domain number for the GPIO (value >= 0)
-1 if the GPIo is not part of any of the domains.

8.28.4 Typedef Documentation

8.28.4.1 typedef void(* gpio_irq_cb)(int pin, void *data)

Function pointer for callback function called on GPIO interrupt

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.29 mdev_gpt.h File Reference

General Purpose Timer (GPT) Driver.

8.29.1 Detailed Description

General Purpose Timer (GPT) driver provides mdev interface to use 4 general purpose hardware timers.

8.29.2 Usage

A typical GPT device usage scenario is as follows:

1. Initialize the GPT driver using gpt_drv_init().
2. Open the GPT device handle using gpt_drv_open(). Here caller must specify GPT number (from 0 to 3)-
3. Use gpt_drv_set() to set desired interval in micro-seconds.
4. Close the GPT device using gpt_drv_close() after its use. gpt_drv_close() call informs the GPT driver to release the resources that it is holding. It is always good practice to free the resources after use.
5. gpt_drv_setcb() should be used to set GPT callback function. This callback will be called after interval specified in gpt_drv_set() until GPT is explicitly stopped using gpt_drv_stop().
6. Start GPT operation using gpt_drv_start().
7. Stop GPT operation using gpt_drv_stop().

   Note

   gpt_drv_stop must not be called from callback registered using gpt_drv_setcb()or from interrupt context.

   Code snippet:

   Following code demonstrates how to use GPT driver APIs to generate a timeout.

```
static void gpt_cb(void)
{
   process_gpt_timeout_cb();
}

{
  gpt_drv_init(timer_id <0-3>);

        .
        .
        .
  mdev_t* gpt_dev = gpt_drv_open(timer_id<0-3>);
  if (gpt_dev == NULL)
      err_handling();
  gpt_drv_set(gpt_dev,time_in_microsecs);
  gpt_drv_setcb(gpt_dev, gpt_cb);
  gpt_drv_start(gpt_dev);
          .
          .
          .
  gpt_drv_stop(gpt_dev);
  gpt_drv_close(gpt_dev);
  return;
}
```

8.29.3 Function Documentation

8.29.3.1 int gpt_drv_init (GPT_ID_Type *id*)

Initialize the GPT driver

This function initializes GPT driver and registers it with mdev interface.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*id*	GPT device to be used.

Returns

WM_SUCCESS on success
Error code otherwise

8.29.3.2 mdev_t* gpt_drv_open (GPT_ID_Type *gpt_id*)

Open GPT device driver

This function opens the handle to GPT device and enables application to use the device. This handle should be used in other calls related to GPT device.

Parameters

in	*gpt_id*	GPT device to be opened.

Returns

handle to driver on success
NULL otherwise

8.29.3.3 int gpt_drv_close (mdev_t * *dev*)

Close the GPT device

This function closes the handle to GPT device.

Parameters

in	*dev*	Handle to the GPT device returned by gpt_drv_open().

Returns

WM_SUCCESS on Success
-WM_FAIL on error

Note

Failing to call this function will make GPT<n> timer un-usable.

8.29.3.4 void gpt_drv_set (mdev_t * *dev*, uint32_t *interval_us*)

Set GPT Configuration

This function sets configuration for GPT device.

Parameters

in	*dev*	Handle to the GPT device returned by gpt_drv_open().
in	*interval_us*	Interval in micro-seconds.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.29.3.5 void gpt_drv_setcb (mdev_t * *dev*, void(*)(void) *func*)

Set GPT callback handler

This function sets callback handler with GPT driver.

Parameters

in	*dev*	Handle to the GPT device returned by gpt_drv_open().
in	*func*	application registered callback handler. To deregister a callback pass NULL. This callback will be called after interval specified in gpt_drv_set() until GPT is explicitly stopped using gpt_drv_stop().

8.29.3.6 void gpt_drv_start (mdev_t * *dev*)

Start GPT device

This function starts GPT operation.

Parameters

in	*dev*	Handle to the GPT device returned by gpt_drv_open().

8.29.3.7 void gpt_drv_stop (mdev_t * *dev*)

Stop GPT device

This function stops GPT operation.

Parameters

in	*dev*	Handle to the GPT device returned by gpt_drv_open(). @ note gpt_drv_stop must not be called from callback registered using gpt_drv_setcb()or from interrupt context.

8.29.3.8 void gpt_drv_pwm (mdev_t * *dev*, GPT_ChannelNumber_Type *ch*, uint32_t *ton*, uint32_t *toff*)

Enable PWM using GPT

This function configures the PWM functionality using the General Purpose Timer.

Note

The correct PinMux needs to be set using the PINMUX driver before calling this function.

The following parameters are set by default

GPT_CH_FUNC_PWM_EDGE - Pulse starts at edge of cycle

GPT_CH_WAVE_POL_POS - Waveform goes from high to low

Parameters

in	*dev*	Handle to the GPT device returned by gpt_drv_open().
in	*ch*	The Channel to generate PWM waveform on.
in	*ton*	Number of cycles of GPT the output will be On.
in	*toff*	Number of cycles of GPT the output will be Off.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.30 mdev_i2c.h File Reference

Inter-Integrated Circuit (I2C) Driver (TWSI)

8.30.1 Detailed Description

The 2-Wire Serial Interface (TSWI) unit is an interface between a 2-wire serial bus and the CPU. I2C Driver is meant to use TWSI hardware to communicate with external peripheral connected on I2C bus through application.

a. The speed of operation: There are standard two speeds at which I2C peripherals operate. 100KHz (Standard speed) and 400KHz (High speed). We have also provided 10KHz (Low speed) to handle any special case.

b. Slave address definition: Any valid transaction on I2C bus is associated with the device slave address and each I2C peripheral has it's own address. The I2C driver should be informed of the slave address of the peripheral that should be accessed.

c. Interface type: Internally TWSI can be connected to different hardware blocks. This identifies the device connection and specifies the interface type.

Note: The FIFO size of the hardware FIFO is 16 bytes

8.30.2 Usage

Follow the steps below to use I2C driver from applications

1. I2C driver initialization using i2c_drv_init().
2. Open I2C device to handle using i2c_drv_open() call.
3. Use i2c_drv_write or i2c_drv_read() calls to write or read data over I2C bus respectively.
4. Close the I2C port using i2c_drv_close() to release the resources that the driver holds.

Typical example:

Below peice of code demonstrates how to configure a I2C device whose slave address is I2C_ADDR interfaced on I2C1 for read/write operation.

```
{
       mdev_t *i2c_dev;

       i2c_dev = i2c_drv_open(I2C1_PORT,
                       I2C_SLAVEADR(I2C_ADDR >> 1));

 if (ret != WM_SUCCESS)
               return;

  uint8_t read_buf[4];
  uint8_t write_buf[2];
  buf[0] = 0x01;
  buf[1] = 0x02;
        int read_len = i2c_drv_read(i2c_dev, buf, 4);
  int write_len = i2c_drv_write(i2c_dev, buf, 4);
        i2c_drv_close(i2c_dev);
}
```

Note

I2C_LCD_ADDR being a 8 bit address, was right shifted by 1 to make it valid 7 bit I2C address. This is slave dependant operation, please check slave device datasheet for more details.

I2C master start stop generation: The I2C IP automatically handles START and STOP condition generation. When first byte is put in TxFIFO (using the API i2c_drv_write()), a START condition is generated and slave address is automatically sent on the bus. When all bytes from the TxFIFO are out on the bus the IP generates STOP condition

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

on the bus. This point is important to note while designing the I2C application. For instance, the application may want to call i2c_drv_write() multiple times during a single transaction. If the time gap between any successive call exceeds FIFO emptying time then a STOP will be generated automatically and things will not work as expected. A potential pitfall is adding debugging prints in between two calls to i2c_drv_write() which may delay next write and the FIFO will may get emptied generating the unexpected STOP. After all the writes are complete you may call i2c_drv_wait_till_inactivity() to ensure that all the writes are out on the bus an a STOP condition is generated signifying end of current transaction.

Conversely, if the application wishes to have two back to back calls to i2c_drv_write() in separate transactions it needs to ensure that TxFIFO is empty before second call to i2c_drv_write(). The API i2c_drv_wait_till_inactivity() is provided for this purpose. Using this API will ensure the application can after this API returns with a success the earlier write is out on the bus and a STOP is generated. The following example illustrates the scenario:

1. Write a byte to I2C device at an address
   - Write slave address
   - Write destination address
   - Write data to be written
2. Use i2c_drv_wait_till_inactivity() to ensure this transaction is complete.
3. Read a byte from some other address
   - Write slave address
   - Write read address
   - Read data

Note that the API i2c_drv_wait_till_inactivity() is not technically necessary for STOP to be generated. It only helps the application to sync itself with that particular bus event.

8.30.3 Function Documentation

8.30.3.1 int i2c_drv_init (I2C_ID_Type *id*)

Initialize I2C Driver

This function initializes I2C driver. And registers the deivce with the mdev interface.

Parameters

in	*id*	I2C device to be used.

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.30.3.2 void i2c_drv_set_clkcnt (I2C_ID_Type *i2c_id*, int *hightime*, int *lowtime*)

Set I2C driver clock(SCL) configuration

This is optional call to change I2C CLK frequency. By default the clock settings are set for standard loading conditions. I2C clock frequency may deviate from the standard 100K/ 400K due to bus loading. User may use this API to tune the I2C clk frequency.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Note

This call should be made after i2c_drv_init() and before i2c_drv_open() to over-write default configuration.

Parameters

in	*i2c_id*	I2C ID of the driver
in	*hightime*	min high time for I2C SCL (clock line) in nanoseconds
in	*lowtime*	min low time for I2C SCL (clock line) in nanoseconds Reducing the hightime or lowtime would increase the SCL frequency Default values for 100K are HCNT: 4000 LCNT:4700 Default values for 400K are HCNT: 600 LCNT:1300

8.30.3.3 int i2c_drv_rxbuf_size (I2C_ID_Type *i2c_id*, uint32_t *size*)

Set RX Ring-buffer size

This is optional call to change rx ringbuffer size. By default ringbuffer size is set to 128 bytes.

Note

This call should be made after i2c_drv_init() and before i2c_drv_open() to over-write default configuration.

Parameters

in	*i2c_id*	I2C ID of the driver
in	*size*	size of ring buffer in bytes

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.30.3.4 int i2c_drv_xfer_mode (I2C_ID_Type *i2c_id*, I2C_DMA_MODE *dma_mode*)

set I2C transfer mode

This is an optional call to set I2C trasfer mode. By default I2C trasfer mode is set to I2C_DMA_DISABLE. This function should be called after i2c_drv_init() and before i2c_drv_open() call to enable DMA transfer mode. Master write and slave read over I2C bus using DMA are supported. Slave write and master read over DMA is not supported and hence use PIO (non DMA) mode for data transmit/receive.

Parameters

in	*i2c_id*	Port ID of I2C
in	*dma_mode*	I2C_DMA_DISABLE or I2C_DMA_ENABLE

Returns

WM_SUCCESS on success.
-WM_FAIL on error.

8.30.3.5 mdev_t* i2c_drv_open (I2C_ID_Type *i2c_id*, uint32_t *flags*)

Open handle to I2C Device

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

This function opens the handle to I2C device and enables application to use the device. This handle should be used in other calls related to I2C device.

Parameters

in	*i2c_id*	I2C port of the driver to be opened.
in	*flags*	

8.30.4 of flags passed to i2c_drv_set_address:

BIT No	Use
BIT[0:6]	I2C_SLAVE_ADDRESS (7 bit address mode)
BIT[0:9]	I2C_SLAVE_ADDRESS (10 bit address mode)
BIT10	set for 10 bit address mode
BIT11	set for slave mode operation
BIT[12 : 13]	speed of operation

Returns

handle to driver on success
NULL otherwise

8.30.4.1 int i2c_drv_set_callback (mdev_t * *dev*, i2c_int_cb *event_cb*, void * *data*)

Set callback function to receive notifications

These notifications are actually hardware interrupts and are delivered in ISR context. The actual ISR handling is done by the driver. The notification may help the driver to manage its state machine.

Note

Only a few interrupts out of the complete list are delivered to the application. Following is the list of interrupts delivered to the application:

1. I2C_INT_RD_REQ
2. I2C_INT_RX_OVER
3. I2C_INT_RX_DONE
4. I2C_INT_STOP_DET

The description of the above interrupts are given in file {mc200/mw300}_i2c.h in the SDK.

Note

Since the callbacks are delivered in ISR context, appropriate care needs to be taken. Especially, no other i2c_drv_* calls like i2c_drv_write() or i2c_drv_read() should be done from this callback as they may using blocking API's internally.

Parameters

in	*dev*	Handle to I2C device returned by i2c_drv_open()
in	*event_cb*	Callback function defined by caller.
in	*data*	user's private pointer that is passed as a parameter to the callback function

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS if callback registered successfully.
-WM_E_INVAL if invalid argument.

8.30.4.2 int i2c_drv_set_address (mdev_t * *dev*, uint32_t *flags*)

Set the slave address in i2c master mode.

This sets the slave address with which i2c module will perform subsequent IO operations in master mode. It can be optionally called after i2c_drv_open() and before performing any read/write operation. This function does not set the address of i2c module in slave mode.

Parameters

in	*dev*	Handle to I2C device returned by i2c_drv_open()
in	*flags*	

8.30.5 of flags passed to i2c_drv_set_address:

BIT No	Use
BIT[0:6]	I2C_SLAVE_ADDRESS (7 bit address mode)
BIT[0:9]	I2C_SLAVE_ADDRESS (10 bit address mode)
BIT10	set for 10 bit address mode

Returns

WM_SUCCESS I2C address is set successfully.
-WM_FAIL in case of failure.

8.30.5.1 int i2c_drv_wait_till_inactivity (mdev_t * *dev*, uint32_t *timeout_ms*)

Wait for I2C module inactivity.

Checks for and wait till the I2C module is inactive. For e.g. when i2c_drv_write() returns, some bytes are still pending in the TxFIFO. If the application wishes to ensure that TxFIFO is empty before starting a new transaction this API needs to be called. Please check current file documentation for more details.

Parameters

in	*dev*	Handle to I2C device returned by i2c_drv_open()
in	*timeout_ms*	0 if to return immediately else spins for inactivity for max timeout_ms millisec-onds.

Returns

WM_SUCCESS I2C module is inactive.
-WM_FAIL I2C module is still active.

8.30.5.2 int i2c_drv_close (mdev_t * *dev*)

Close handle to I2C Device

This function closes the handle to I2C device and resets the I2C

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*dev*	Handle to the I2C device returned by i2c_drv_open().

Returns

WM_SUCCESS on success, -WM_FAIL on error.

8.30.5.3 int i2c_drv_read (mdev_t * *dev*, void * *buf*, uint32_t *nbytes*)

Read from I2C

This function reads specified number of bytes into the provided buffer from the I2C device. I2C Reads two bytes from interface as [Start][Slave address(read)]Rd Byte0, Rd Byte1[Stop] sequence on I2C

Parameters

in	*dev*	Handle to the I2C device returned by i2c_drv_open().
out	*buf*	Buffer Pointer to an allocated buffer of size equal to or more than the value of the third parameter nbytes.
in	*nbytes*	Number of bytes to be read.

Returns

number of bytes read on success
-WM_E_AGAIN in case of a timeout (if timeout is enabled using the API i2c_drv_timeout) non-zero otherwise.

Note

: read/write timeout is supported for master mode only

8.30.5.4 int i2c_drv_slv_dma_read (mdev_t * *dev*, void * *buf*, uint32_t *nbytes*)

Read from I2C port in slave mode in DMA mode

This function reads specified number of bytes into the user buffer

Parameters

in	*dev*	Handle to the I2C device returned by i2c_drv_open().
out	*buf*	Buffer Pointer to an allocated buffer of size equal to or more than the value of the third parameter nbytes.
in	*nbytes*	Number of bytes to be read.

Returns

number of bytes read on success

8.30.5.5 int i2c_drv_write (mdev_t * *dev*, const void * *buf*, uint32_t *nbytes*)

Write to I2C I2C Reads two bytes from interface as [Start][Slave address(write)]Byte0,Byte1[Stop] sequence on I2C

This function writes specified number of bytes to the I2C device Tx-FIFO.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Note

In slave mode if the user wants to enable low power modes, application should take wakelock in order to prevent slave from entering low power mode while serving the write request.

While operating in slave mdoe, if the master sends RD_REQ to the slave before i2c_drv_write call is made, slave Tx-FIFO remains empty and slave sends out dymmy data byte (i.e.0xff).

Parameters

in	*dev*	mdev_t handle to the driver.
in	*buf*	Buffer containing data to be written.
in	*nbytes*	Number of bytes to be written.

Returns

number of bytes written on success -WM_E_AGAIN in case of a timeout(if timeout is enabled using the API i2c_drv_timeout), -WM_FAIL otherwise.

Note

1. For a slave device waiting on RD_REQ, if time out occurs, no data is sent to master and i2c_drv_write returns 0 in this case.
2. read/write timeout is supported for master mode only

8.30.5.6 int i2c_drv_dma_write (mdev_t * *dev*, const uint8_t * *buf*, uint32_t *nbytes*)

Write to I2C using DMA I2C Reads two bytes from interface as [Start][Slave address(write)]Byte0,Byte1[Stop] sequence on I2C

This function writes specified number of bytes to the I2C device Tx-FIFO; using DMA to copy data from RAM to I2C FIFO.

Parameters

in	*dev*	mdev_t handle to the driver.
in	*buf*	Buffer containing data to be written.
in	*nbytes*	Number of bytes to be written.

Returns

number of bytes written on success -WM_E_AGAIN in case the data length exceeds the max allowed length -WM_FAIL otherwise.

Note

1. This API is supported for master mode only

8.30.5.7 int i2c_drv_dma_read (mdev_t * *dev*, uint8_t * *buf*, uint32_t *nbytes*)

Read from I2C using DMA

This function reads specified number of bytes from the I2C-RX-FIFO using DMA to copy data from I2C FIFO to RAM.

Parameters

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

in	*dev*	mdev_t handle to the driver.
in	*buf*	Buffer containing data to be written.
in	*nbytes*	Number of bytes to be written.

Returns

number of bytes written on success -WM_E_AGAIN in case of a timeout -WM_FAIL otherwise.

Note

1. This API is supported for master mode only

8.30.5.8 void i2c_drv_timeout (I2C_ID_Type *id*, uint32_t *txtout*, uint32_t *rxtout*)

Set timeout for I2C read/write operations for master mode only.

This function sets the timeout for I2C read/write operations. It is disabled by default. This function needs to be called after init and before open for that particular port.

Parameters

in	*id*	I2C port number
in	*txtout*	Tx timeout value in milli seconds, or zero to disable timeout.
in	*rxtout*	Rx timeout value in milli seconds, or zero to disable timeout.

8.30.5.9 void i2c_drv_deinit (I2C_ID_Type *id*)

De-initialize I2C Port

This function de-initializes I2C port. Note that if an I2C device has been opened using i2c_drv_open(), it should be first closed using i2c_drv_close() before calling this function.

Parameters

in	*id*	I2C device to be used.

8.30.6 Macro Documentation

8.30.6.1 #define NUM_I2C_PORTS 2

Maximum number of I2C Hardware Interfaces available

8.30.6.2 #define BIT10 (0x00000001 << 10)

I2C specific parameters required for initialization

8.30.6.3 #define I2C_CLK_100KHZ (~(BIT12 | BIT13) & 0x3000)

I2C Hardware Interface speed of operation -Standard (100Khz)

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.30.6.4 #define I2C_CLK_10KHZ BIT13

I2C Hardware Interface speed of operation -Low (10Khz)

8.30.6.5 #define I2C_CLK_400KHZ BIT12

I2C Hardware Interface speed of operation -High (10Khz)

8.30.6.6 #define I2C_DEVICE_SLAVE BIT11

I2C flag to open driver as slave mode

8.30.6.7 #define I2C_SLAVEADR(*x*) (0x3ff & (x))

Macro to specify slave address of max 10 bits while opening a device the address would be truncated to 7 bits if I2C_10_BIT_ARRD is not set

8.30.6.8 #define I2C_10_BIT_ADDR BIT10

Macro to specify 10 bit address mode

8.30.7 Enumeration Type Documentation

8.30.7.1 enum I2C_DMA_MODE

Enum to enable and disable DMA mode for I2C transfer

Enumerator

I2C_DMA_DISABLE Disable I2C DMA controller

I2C_DMA_ENABLE Enable I2C DMA controller

8.30.7.2 enum i2c_status

Enumeration which indicates status of I2C

Enumerator

I2C_INACTIVE I2C_INACTIVE : indicates that no ongoing I2C activity on the bus

I2C_ACTIVE I2C_ACTIVE : indicates I2C bus is busy.

I2C_ERROR I2C_ERROR : indicates that an error has occurred on I2C bus.

8.31 mdev_pinmux.h File Reference

GPIO Pin Multiplexer Driver.

8.31.1 Detailed Description

Various functionalities are multiplexed on the same GPIO pin as the number of Pins are limited. The functionality to be assigned to any particular pin can be configured using the PINMUX driver.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.31.2 Usage

A typical PINMUX device usage scenario is as follows:

1. Initialize the PINMUX driver using pinmux_drv_init().
2. Open the PINMUX device handle using pinmux_drv_open() call.
3. Set the alternate function using pinmux_drv_setfunc()
4. Close the PINMUX device using pinmux_drv_close() after its use.

   pinmux_drv_close() informs the PINMUX driver to release the resources that it is holding. It is always good practice to free the resources after use.

Code snippet:

Following code demonstrates how to use PINMUX driver APIs to set a pin's alternate function.

```
{
  pinmux_drv_init();
        .
        .
        .
  mdev_t* pinmux_dev = pinmux_drv_open("MDEV_PINMUX");
  if (pinmux_dev == NULL)
      err_handling();

  int err = pinmux_drv_setfunc(pinmux_dev, pin, PINMUX_FUNCTION_0);
  if (err != WM_SUCESS)
      err_handling();
  pinmux_drv_close(pinmux_dev);
  return;
}
```

8.31.3 Function Documentation

8.31.3.1 int pinmux_drv_init (void)

Initialize the PINMUX driver

This function initializes PINMUX driver and registers it with mdev interface.

Returns

WM_SUCCESS on success
Error code otherwise

8.31.3.2 mdev_t* pinmux_drv_open (const char * *name*)

Open PINMUX device driver

This function opens the handle to PINMUX device and enables application to use the device. This handle should be used in other calls related to PINMUX device.

Parameters

in	*name*	Name of mdev pinmux driver. It should be "MDEV_PINMUX" string.

Returns

handle to driver on success
NULL otherwise

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.31.3.3 int pinmux_drv_close (mdev_t * *dev*)

Close the PINMUX device

This function closes the handle to PINMUX device.

Parameters

in	*dev*	Handle to the PINMUX device returned by pinmux_drv_open().

Returns

WM_SUCCESS on success

8.31.3.4 int pinmux_drv_setfunc (mdev_t * *dev*, int *pin*, int *func*)

Set alternate function of a GPIO pin

Sets the alternate function for the specified pin.

Parameters

in	*dev*	Handle to the PINMUX device returned by pinmux_drv_open().
in	*pin*	Pin to be modified
in	*func*	Alternate function number(GPIO_PinMuxFunc_Type)

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.31.3.5 int pinmux_drv_get_gpio_func (int *pin*)

Get the GPIO pinmux function for a given pin

The GPIO pins are multiplexed such that they can be used for various alternate functions. The required function has to be selected by setting the mux appropriately using pinmux_drv_setfunc(). If the pin has to be used as a GPIO, use this function to get the corresponding pinmux function to be passed to pinmux_drv_setfunc()

Parameters

in	*pin*	Pin number

Returns

The GPIO pinmux function for the given pin if successful.
-1 on error.

8.32 mdev_pm.h File Reference

MCU power management module.

8.32.1 Detailed Description

Allows application to put MCU in various low power modes. MCU power modes in brief are :

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

- PM0 (Active Mode): This is the full power state of MCU. Instruction execution takes place only in PM0.
- PM1 (Idle Mode) : In this mode, the Cortex Mx core function clocks are stopped until the occurrence of any interrupt. This consumes lower power than PM0. On exit from this mode, execution resumes from the next instruction in SRAM.
- PM2 (Standby Mode):In this mode, the Cortex Mx core, most of the peripherals & SRAM arrays are in low-power mode. The PMU and RTC are operational. A wakeup can happen by timeout (RTC based) or by asserting the WAKEUP 0/1 lines. This consumes much lower power than PM1. On exit from this mode, execution resumes from the next instruction in SRAM.
- PM3 (Sleep Mode) : This mode further aggressively conserves power. Only 192 KB (160 KB in SRAM0 and 32 KB in SRAM1) out of 512 KB of SRAM is alive. All peripherals are turned off and register config is lost. It is the application's responsbility to restore the peripheral config after exit from PM3. This consumes lower power than in PM2. A wakeup can happen by timeout (RTC based)or by asserting the WAKEUP 0/1 lines. On exit from this mode, execution resumes from the next instruction in SRAM.
- PM4(Shutoff Mode) : This simulates a shutdown condition. A wakeup can happen by timeout (RTC based) or by asserting the WAKEUP 0/1 lines. This is the lowest power state of MCU. On wakeup, execution begins from bootrom as if a fresh bootup has occurred.

  The power management module provides two APIs :

- a power manager framework API, pm_mcu_cfg(), that allows applications to define a power management policy. Once defined, the system will automatically put the MCU into the configured power save modes on detection of idle activity.
- a raw API, pm_mcu_state(), which puts the MCU into the configured power save states immediately.

8.32.2 MCU Power Modes relationship Table.

Cortex Mx state

Run (C0)	Idle (C1)	Standby (C2)	Off (C3)
HCLK On	HCLK On	HCLK Off	Power removed
FCLK On	FCLK Off	FCLK Off	Power removed

SRAM Modes

Run (M0)	Standby (M2)	Off (M3)
On	Low power	Power off

Power Modes with Core , SRAM , Peripherals

	PM0	PM1	PM2	PM3	PM4
Cortex M3	C0	C1	C2	C3	C3
SRAM	M0	M0	M2	M2	M3
Peripherals	On	On	Retained	Off	Off

MCU state transiotions are as shown below:

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.32.3 Function Documentation

8.32.3.1 void pm_mcu_cfg (bool *enabled,* power_state_t *mode,* unsigned int *threshold*)

Configure Power Mgr Framework's MCU Power Save

This function is used to configure the power manager framework's MCU Power Save. When enabled, the power manager framework will detect if the system is idle and put the MCU processor in power save mode. Before entering power save mode, the power manager framework will detect the next scheduled wakeup for any process/timer and configure the sleep duration such that the processor wakes up just in time to service that process. Once enabled the power manager framework will continue to opportunistically put the system into the configured low power mode.

A threshold parameter is also supported. The power manager framework will make the MCU enter power save mode, only if the next scheduled thread wakeup is greater than the configured threshold.

For example, say,

1. the threshold is configured to 10 milliseconds
2. the PM2 exit latency is 3 milliseconds Now if the system is idle, (that is there is no running thread in the system) at time t0, the power management framework will put the MCU into power save mode, only if the next thread is scheduled to wakeup and execute at t0 + 10 + 3 milliseconds. If a thread is scheduled to wakeup and execute before this time, then the MCU doesn't enter power save at time t0.

Parameters

in	*enabled*	true to enable, false to disable power manager framework
in	*mode*	MCU power mode power_state_t.
in	*threshold*	System will not go to power save state if it is going to wake up in less than [threshold] ticks.

Note

This API should be used only for PM2 and PM3. The application should call this function only once to configure and enable power save scheme. Power manager framework takes care of opportunistically entering and exiting power save.
Applications can register callbacks for low power entry and exit using pm_register_cb(). This will ensure that the callback gets called everytime before entering the power save mode or exiting from the power save mode. Applications can perform any state save-restore operations in this callback.

8.32.3.2 int pm_mcu_state (power_state_t *state,* uint32_t *time_dur*)

Enter low power state of MCU

This function puts the core into the requested low power mode.

Parameters

in	*state*	the desired power mode of MCU power_state_t
in	*time_dur*	duration (in milliseconds) for which device should be in requested low power mode. The special value 0 indicates maximum possible timeout that can be configured using RTC.

Returns

wakeup_cause

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Note

The device wakes up automatically after the pre-configured time interval has expired or when the external gpio pin is asserted. For details please refer to the MCU data sheet. MCU by default keeps on toggling between PM0 and PM1.

8.32.3.3 int pm_wakeup_source ( )

This function is used to get last source of wake up.

Returns

wakeup_cause

8.32.3.4 bool pm_mcu_is_enabled ( )

This function is used to check enable status of power manager framework.

Returns

current power manager enable status,
true if enabled
false if disabled

8.32.3.5 int pm_mcu_get_mode ( )

Return target power save mode

This function returns the configured power save mode that the MCU will enter on detection of inactivity.

Returns

power_state_t

8.32.3.6 void pm_mcu_io_cfg (uint32_t *domain_on*)

Manage IO Domains on/off in PM2 This function allows application to determine which GPIO domain (D0, D1, D2, D3 (for MW300)) are to be kept on or off when MCU enters PM2

Note

This API should be used if application wants to keep IO domains on in PM2 else by default they will be turned off while entering PM2 and turned on after exiting PM2.

Parameters

in	*domain_on*	Logical OR of IO domains to be i kept on in PM2.

8.32.3.7 void arch_pm_init (void)

Initialize Power Manager

This is an MCU specific API to initialize Power manager module

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.32.3.8 void arch_reboot ( )

Reboot MCU

This API can be used to reboot the MCU.

8.32.4 Macro Documentation

8.32.4.1 #define pmu_drv_poweron_vddio(*domain*) PMU_PowerOnVDDIO(domain)

Turn on individual IO domains

MC200 MCU has 6 configurable IO Voltage domains.

1. D0 : GPIO 0 to GPIO 17
2. Aon : GPIO 18 to GPIO 27 (Always On)
3. D1 : GPIO 28 to GPIO 50
4. SDIO : GPIO 51 to GPIO 56 (SDIO interface)
5. D2 : GPIO 59 to GPIO 79
6. FL : Internal Flash

MW300 MCU has 5 configurable IO Voltage domains.

1. D0 : GPIO 0 to GPIO 15
2. D1 : GPIO 16 to GPIO 21
3. Aon : GPIO 22 to GPIO 26 (Always On)
4. D2 : GPIO 28 to GPIO 33
5. D3 : GPIO 27, GPIO 34 to GPIO 49

Each domain can be in 3 possible states

- ON/Normal : Full power on.
- Deep OFF : Low power state. Latency to go back to ON state is less.
- OFF : Completely off. Latency to go back to ON state is higher than Deep Off.

When MCU enters PM2, PM3, PM4 using power management APIs all IO voltage domains except Aon are turned off. Aon is kept in deep off. These APIs assist in controlling the IO domains in PM0.

Parameters

in	*domain*	IO domain to be turned on

Returns

none

8.32.4.2 #define pmu_drv_poweroff_vddio(*domain*) PMU_PowerOffVDDIO(domain)

Turn off individual IO domains

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*domain*	IO domain to be turned off

Returns

none

Note

For details of IO domains please see pmu_drv_poweron_vddio()

8.32.4.3 #define pmu_drv_powerdeepoff_vddio(*domain*) PMU_PowerDownVDDIO(domain)

Deep off individual IO domains

Parameters

in	*domain*	IO domain to be set to deep off

Returns

none

Note

For details of IO domains please see pmu_drv_poweron_vddio()

8.32.4.4 #define pmu_drv_powernormal_vddio(*domain*) PMU_PowerNormalVDDIO(domain)

Set individual IO domains in Normal mode This API should be called to get IO domain out of Deep Off mode

Parameters

in	*domain*	IO domain to be set to normal mode

Returns

none

Note

For details of IO domains please see pmu_drv_poweron_vddio()

8.32.5 Enumeration Type Documentation

8.32.5.1 enum wakeup_cause

Indicates the wakeup source It can be any external GPIO pin or RTC.

Enumerator

EGPIO0 Wakeup source is EXT GPIO 0
EGPIO1 Wakeup Source is EXT GPIO 1
ERTC Wakeup source is RTC
EWLAN ONLY for MW300 : Wakeup source is WLAN

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.32.5.2 enum io_domain_t

Enumeration for Voltage IO domains om MCU

Enumerator

IO_DOMAIN_0 voltage domain 0

IO_DOMAIN_1 voltage domain 1

IO_DOMAIN_2 voltage domain 2

IO_DOMAIN_3 voltage domain 3 only for MW300

8.32.5.3 enum power_state_t

Power States of MCU

Enumerator

PM0 (Active Mode): This is the full power state of MCU. Instruction execution takes place only in PM0.

PM1 (Idle Mode): In this mode Cortex M3 core function clocks are stopped until the occurrence of any interrupt. This consumes lower power than PM0.

PM2 (Standby Mode):In this mode, the Cortex M3, most of the peripherals & SRAM arrays are in low-power mode.The PMU and RTC are operational. A wakeup can happen by timeout (RTC based) or by asserting the WAKEUP 0/1 lines.This consumes much lower power than PM1.

PM3 (Sleep Mode): This mode further aggressively conserves power. Only 192 KB (160 KB in SRAM0 and 32 KB in SRAM1) out of 512 KB of SRAM is alive. All peripherals are turned off and register config is lost. Application should restore the peripheral config after exit form PM3. This consumes lower power than in PM2. A wakeup can happen by timeout (RTC based) or by asserting the WAKEUP 0/1 lines.

PM4 (Shutoff Mode): This simulates a shutdown condition. A wakeup can happen by timeout (RTC based) or by asserting the WAKEUP 0/1 lines. This is the lowest power state of MCU. On wakeup execution begins from bootrom as if a fresh bootup has occurred.

8.33 mdev_rtc.h File Reference

RTC driver.

8.33.1 Detailed Description

RTC (Real Time Clock) driver provides mdev interface to access the internal RTC. Follow the steps below to use RTC driver from applications

1. RTC driver initialization

Call rtc_drv_init from your application initialization function. This will initialize and register rtc driver with the mdev interface.

1. Open RTC device to use

rtc_drv_open should be called once before using RTC hardware for communication.

```
{
    mdev_t *rtc_dev;
    rtc_dev = rtc_drv_open("MDEV_RTC");
}
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

1. Set RTC configuration

Call rtc_drv_set from your application to set desired upper count The RTC runs at 1KHz by default (for A0 version only).

1. Set RTC callback function

Call rtc_drv_set_cb from your application to register callback handler with RTC driver. This will be called after clock exceeds the upper value specified in step 3, until RTC is explicitly stopped using rtc_drv_stop. While RTC continues to run , this will be called everytime the counter value wraps around. RTC can be stopped by calling rtc_drv_stop.

1. Reset the RTC counter

Call rtc_drv_reset to reset the rtc counter before starting .

1. Start RTC operation

Call rtc_drv_start to start rtc.

1. Get RTC counter value

Call rtc_drv_get to rtc counter value.

1. Stop RTC operation

Call rtc_drv_stop to stop rtc.

1. Closing a device.

A call to rtc_drv_close informs the RTC driver to release the resources that it is holding.

Note

This must not be called from callback registered in step 4 or any interrupt context.

Typical usecase:

```
rtc_drv_close(rtc_dev);
```

8.33.2 Function Documentation

8.33.2.1 int rtc_drv_init (void)

Register RTC Device

This function registers rtc driver with mdev interface.

Returns

WM_SUCCESS on success, error code otherwise.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.33.2.2 mdev_t* rtc_drv_open (const char * *name*)

Open RTC Device

This function opens the device for use and returns a handle. This handle should be used for other calls to the RTC driver.

Parameters

in	*name*	Name of the driver to be opened. Name should be "MDEV_RTC" string

Returns

NULL if error, mdev_t handle otherwise.

8.33.2.3 void rtc_drv_set (mdev_t * *dev*, uint32_t *cnt_upp*)

Set RTC Configuration

This function sets configuration for rtc device.

Parameters

in	*dev*	mdev_t handle to the driver.
in	*cnt_upp*	Overflow value. Counting starts from 0 and does to to this value. Frequency is 1KHz by default for A0 version of chip and 32KHz for Z1 or earlier chips.

8.33.2.4 void rtc_drv_set_cb (void * *user_cb*)

Set RTC callback handler for UPP Val

This function sets callback handler with RTC driver.

Parameters

in	*user_cb*	application registered callback handler, if func is NULL then the callback is unset.

8.33.2.5 void rtc_drv_set_alarm_cb (void * *user_cb*)

Set RTC callback handler for Alarm

This function sets callback handler with RTC driver.

Parameters

in	*user_cb*	application registered callback handler, if func is NULL then the callback is unset.

8.33.2.6 void rtc_drv_reset (mdev_t * *dev*)

Reset RTC counter

This function resets the RTC counter.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*dev*	mdev_t handle to the driver.

8.33.2.7 void rtc_drv_start (mdev_t * *dev*)

Start RTC device

This function starts RTC operation.

Parameters

in	*dev*	mdev_t handle to the driver.

8.33.2.8 unsigned int rtc_drv_get (mdev_t * *dev*)

Get RTC counter value.

This function gets the value of RTC counter.

Parameters

in	*dev*	mdev_t handle to the driver.

Returns

the counter value.

8.33.2.9 unsigned int rtc_drv_get_uppval (mdev_t * *dev*)

Get RTC upper count value.

This function gets the value of RTC upper count. That is the value till which the RTC up-counts.

Parameters

in	*dev*	mdev_t handle to the driver.

Returns

the upper count value.

8.33.2.10 void rtc_drv_set_alarm (mdev_t * *dev*, uint32_t *cnt_alarm*) [inline]

Set RTC ALARM

This function sets Alarm configuration for rtc device.

Parameters

in	*dev*	mdev_t handle to the driver.
in	*cnt_alarm*	value at which Alarm interrupt will be generated. Frequency is 1KHz by default.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.33.2.11 void rtc_drv_stop (mdev_t * *dev*)

Stop RTC device

This function stops RTC operation.

Parameters

in	*dev*	mdev_t handle to the driver.

8.33.2.12 int rtc_drv_close (mdev_t * *dev*)

Close RTC Device

This function closes the device after use and frees up resources.

Parameters

in	*dev*	mdev handle to the driver to be closed.

Returns

WM_SUCCESS on success, -WM_FAIL on error.

8.34 mdev_sdio.h File Reference

SDIO Driver.

8.34.1 Detailed Description

The SDIO driver is used to configure and do data transfer over the SDIO interface

8.34.2 Usage

Steps to use sdio with mdev:

1. Register gpio using sdio_drv_init()
2. Open the device using sdio_drv_open() call. This will return a handler to gpio.
3. To do various SDIO related operations:
   - sdio_drv_write() : Write using CMD53
   - sdio_drv_read() : Read using CMD53
   - sdio_drv_creg_write() : Write using CMD52
   - sdio_drv_creg_read() : Read using CMD52

   Locking mechanism is implemented to provide atomic access.
4. Close the device using sdio_drv_close() call.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.34.3 Function Documentation

8.34.3.1 int sdio_drv_creg_read (mdev_t * *dev*, int *addr*, int *fn*, uint32_t * *resp*)

Read Card Register

This is used to read card register using CMD52. This is a blocking call.

Parameters

dev	mdev handle returned by sdio_drv_open
addr	Card Register Address
fn	Number of the function with the Card
resp	Response of CMD52

Returns

true on success, false otherwise

8.34.3.2 int sdio_drv_creg_write (mdev_t * *dev*, int *addr*, int *fn*, uint8_t *data*, uint32_t * *resp*)

Write to Card Register

This is used to write to card register using CMD52. This is a blocking call.

Parameters

dev	mdev handle returned by sdio_drv_open
addr	Card Register Address
fn	Number of the function with the Card
data	value to be written on register
resp	Response of CMD52

Returns

true on success, false otherwise

8.34.3.3 int sdio_drv_read (mdev_t * *dev*, uint32_t *addr*, uint32_t *fn*, uint32_t *bcnt*, uint32_t *bsize*, uint8_t * *buf*, uint32_t * *resp*)

Read Data from SDIO

This is used to read data from SDIO card using CMD53. This is a blocking call.

Parameters

dev	mdev handle returned by sdio_drv_open
addr	Card Register Address
fn	Number of the function with the Card
bcnt	Number of blocks to be read
bsize	Size of each block
buf	Buffer to read the data into
resp	Response of CMD53

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

true on success, false otherwise

8.34.3.4 int sdio_drv_write (mdev_t * *dev*, uint32_t *addr*, uint32_t *fn*, uint32_t *bcnt*, uint32_t *bsize*, uint8_t * *buf*, uint32_t * *resp*)

Write Data to SDIO

This is used to write data to SDIO card using CMD53. This is a blocking call.

Parameters

dev	mdev handle returned by sdio_drv_open
addr	Card Register Address
fn	Number of the function with the Card
bcnt	Number of blocks to be written
bsize	Size of each block
buf	Buffer to write the data into
resp	Response of CMD53

Returns

true on success, false otherwise

8.34.3.5 mdev_t* sdio_drv_open (const char * *name*)

Open SDIO Driver

This returns a mdev handle.

Parameters

name	Name of driver to be opened

Returns

handle to the opened driver on success, NULL on failure

8.34.3.6 int sdio_drv_init (void(*)(int) *cd_int*)

Initialize the SDIO Driver

This should be called once before using the driver. The callback for Card Interrupt can be registered using this call.

Parameters

cd_int	Callback for Card Detect Interrupt

Returns

WM_SUCCESS or -WM_FAIL

8.34.3.7 void sdio_drv_deinit (void)

De-initialize the SDIO Driver

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

SDIO will not be usable after this call. All resources are released and driver is deregistered.

8.34.3.8 void sdio_drv_set_cb1 (void(∗)(int) *cd_int*)

Register additional callback

Additional callback for Card Interrupt (other than the one set during sdio_drv_init()) can be registered using this call.

Parameters

cd_int	Callback for Card Detect Interrupt

8.35 mdev_ssp.h File Reference

Synchronous Serial Protocol (SSP) Driver.

8.35.1 Detailed Description

The SSP port is a synchronous serial controller that can be connected to a variety of external devices which use serial protocol for data transfer. There are 3 SSP interfaces viz, SSP0 , SSP1, SSP2. The SSP ports can be configured to operate in Master Mode (wherein attached device is slave) or Slave Mode (wherein external device is master). SSP driver exposes various SSP device features through mdev interface.

8.35.2 Usage

1. Verify that board file API board_ssp_pin_config() located at wmsdk-bundle-x.x.x/wmsdk-x.x.x/src/boards/{board-_file}.c reflects the correct pinmux settings. If found missing, please add the appropriate pinmux configuration corresponding to the SSP_ID_Type.
2. Initialize the SSP driver using ssp_drv_init().
3. Open the SSP device handle using ssp_drv_open(). ssp_drv_open() takes as input frame format (SPI or I2S) and master/slave mode. Optionally it can also accept custom chip select gpio pin for slave device.
4. Use ssp_drv_read() or ssp_drv_write() to read SSP data or
5. ssp_drv_set_clk() can be used to set frequency of SSP port.
6. ssp_drv_rxbuf_size() can be called to set Receive buffer size. to write data to SSP respectively.
7. Close the SSP device using ssp_drv_close() after its use. ssp_drv_close() call informs the SSP driver to release the resources that it is holding. It is always good practice to free the resources after use.

   Code snippet:

   Following code demonstrates SSP0 in master mode transmitting & reading data back from a slave device.

```
{
    int ret;
    uint8_t buf[512];
    uint8_t read_buf[512];
    mdev_t *ssp_dev;
    ssp_drv_init(SSP0_ID);
    ssp_drv_set_clk(SSP0_ID, 13000000);
    ssp_drv_rxbuf_size(SSP0_ID, 1024);
    ssp_dev = ssp_drv_open(SSP0_ID, SSP_FRAME_SPI,
                           SSP_MASTER, DMA_DISABLE,
                           -1, 0);
    unit32_t len = ssp_drv_write(ssp_dev, write_data,
                                 NULL,
                                 512,
                                 0);
                        .
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

```
                    .
                    .
    len = ssp_drv_read(ssp_dev, read_buf, 512);

    ssp_drv_deinit(ssp_dev);
}
```

Note

SSP_ID_Type enumeration has values SSP0_ID, SSP1_ID, SSP2_ID

8.35.3 Default Parameter Configuration

```
Default configuration parameter set for SPI are:<br>
mode : SSP_NORMAL<br>
frameformat : SSP_FRAME_SPI<br>
master or slave : configured by user<br>
tr/rx mode  : SSP_TR_MODE<br>
frame data size : SSP_DATASIZE_8<br>
serial frame polarity type : SSP_SAMEFRM_PSP<br>
slave clock running type : SSP_SLAVECLK_TRANSFER<br>
txd turns to three state type : SSP_TXD3STATE_ELSB (i.e. on clock edge
ends LSB)<br>
Enable or Disable SSP turns txd three state mode: ENABLE<br>
Timeout : 0x3<br>

Default configuration parameters set for I2S are:<br>
mode : SSP_NORMAL<br>
frameformat : SSP_FRAME_PSP<br>
master or slave : configured by user<br>
tr/rx mode  : SSP_TR_MODE<br>
frame data size : SSP_DATASIZE_8<br>
serial frame polarity type : SSP_SAMEFRM_PSP<br>
slave clock running type : SSP_SLAVECLK_TRANSFER<br>
txd turns to three state type : SSP_TXD3STATE_12LSB (i.e. 1/2 clock
cycle after start LSB)<br>
Enable or Disable SSP turns txd three state mode: ENABLE<br>
Timeout : 0x3<br>
```

8.35.4 Function Documentation

8.35.4.1 int ssp_drv_init (SSP_ID_Type *id*)

Initialize SSP Driver

This function initializes SSP driver. And registers the device with the mdev interface.

Parameters

in	*id*	ssp port to be initialized.

Returns

WM_SUCCESS on success
error code otherwise

8.35.4.2 void ssp_drv_deinit (SSP_ID_Type *id*)

De-initialize SSP Driver

This function de-initializes SSP driver. Note that if a SSP device has been opened using ssp_drv_open(), it should be first closed using ssp_drv_close() before calling this function.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*id*	SSP port id

8.35.4.3 mdev_t* ssp_drv_open (SSP_ID_Type *ssp_id*, SSP_FrameFormat_Type *format*, SSP_MS_Type *mode*, SSP_DMA *dma*, int *cs*, bool *level*)

Open handle to SSP Device

This function opens the handle to SSP device and enables application to use the device. This handle should be used in other calls related to SSP device.

Parameters

in	*ssp_id*	SSP ID of the driver to be opened
in	*format*	configure mdev driver for SPI/I2S
in	*mode*	it can be either master or slave
in	*dma*	it can be either enabled or disabled
in	*cs*	chip select gpio number (optional) or -1, for master mode only
in	*level*	chip select gpio level, active low or active high, for master mode only

Returns

handle to driver on success
NULL otherwise

Note

1. If DMA mode is enabled, it will used DMA channel 0 for SSP tx and DMA channel 1 for SSP rx operation.

8.35.4.4 int ssp_drv_close (mdev_t * *dev*)

Close handle to SSP Device

This function closes the handle to SSP device and resets the SSP.

Parameters

in	*dev*	Handle to the SSP device returned by ssp_drv_open().

Returns

WM_SUCCESS on success
-WM_FAIL on error.

8.35.4.5 int ssp_drv_write (mdev_t * *dev*, const uint8_t * *data*, uint8_t * *din*, uint32_t *num*, bool *flag*)

Write data to SSP port

This function is used to write data to the SSP in order to transmit them serially over serial out line. The write call is BLOCKING. It will not return till all the requested number of data elements (8 bit) are transmitted.

Parameters

in	*dev*	Handle to the SSP device returned by ssp_drv_open().
in	*data*	data to be written
in	*din*	buffer address for data to be read (if flag = 1) or null

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

in	*num*	Number of 8 bit elements to be written
in	*flag*	flag = 1, read data from rxfifo

Returns

Number of 8 bit elements actually written (may be less than num if timeout occurs) -WM_FAIL otherwise.

Note

: read/write timeout is supported for dma io only
When SSP pushes out a byte from txfifo, it receives a byte in rxfifo. So, depending on slave device behavior one may want to read out these bytes from rxfifo and store them or discard them. If flag = 1 then rxfifo will be read out and data will be stored in din. If din is NULL data will be read out and discarded. If flag = 0 data will not be read. e.g. Some flash chip interfaced over ssp might need to discard read data when writing some command to flash chip.

8.35.4.6 int ssp_drv_read (mdev_t * *dev*, uint8_t * *data*, uint32_t *num*)

Read from SSP port

This function is used to read data from the SSP The Read call is NONBLOCKING for slave and BLOCKING for master.

Parameters

in	*dev*	Handle to the SSP device returned by ssp_drv_open().
out	*data*	Pointer to an allocated buffer of size equal to or more than the value of the third parameter num.
in	*num*	The maximum number of bytes to be read from the SSP.

Returns

Number of 8 bit elements actually read (may be less than num if timeout occurs) -WM_FAIL otherwise.

Note

: read/write timeout is supported for dma io only

8.35.4.7 void ssp_drv_cs_activate (mdev_t * *dev*)

Assert chip select gpio

This will assert chip select gpio (only applicable to master mode) as per level configured during driver init step. Use of this api is entirely optional.

Parameters

in	*dev*	Handle to the SSP device returned by ssp_drv_open().

8.35.4.8 void ssp_drv_cs_deactivate (mdev_t * *dev*)

De-assert chip select gpio

This will de-assert chip select gpio (only applicable to master mode) as per level configured during driver init step.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Use of this api is entirely optional.

Parameters

in	*dev*	Handle to the SSP device returned by ssp_drv_open().

8.35.4.9 uint32_t ssp_drv_set_clk (SSP_ID_Type *ssp_id*, uint32_t *freq*)

Set clock frequency

This is optional call to set ssp clock frequency. By default clock frequency is set to 10MHz. This call should be made after ssp_drv_init and before ssp_drv_open to over-write default clock configuration.

Parameters

in	*ssp_id*	SSP ID of the driver
in	*freq*	frequency to be set

Returns

Actual frequency set by driver
-WM_FAIL on error

8.35.4.10 int ssp_drv_rxbuf_size (SSP_ID_Type *ssp_id*, uint32_t *size*)

Set RX Ringbuffer size

This is optional call to change rx ringbuffer size. By default ringbuffer size is set to 2K. This call should be made after ssp_drv_init and before ssp_drv_open to over-write default configuration.

Parameters

in	*ssp_id*	SSP ID of the driver
in	*size*	size of ring buffer in bytes

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.35.4.11 int ssp_drv_timeout (SSP_ID_Type *ssp_id*, uint32_t *txtout*, uint32_t *rxtout*)

Set timeout for SSP read/write operations for dma io only.

This function sets the timeout for SSP dma read/write operations. It is disabled by default. This function needs to be called after init and can be optinally called before any read/write operation in dma io. Once set, it will use the same timeout values for all operations, till they are changed again by another call.

Parameters

in	*ssp_id*	SSP ID of the driver
in	*txtout*	Tx timeout value in milli seconds, or OS_WAIT_FOREVER to disable timeout.
in	*rxtout*	Rx timeout value in milli seconds, or OS_WAIT_FOREVER to disable timeout.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.35.5 Enumeration Type Documentation

8.35.5.1 enum SSP_DMA

enum to enable or disable DMA in SSP data transfer

Enumerator

DMA_DISABLE Disable ssp DMA controller
DMA_ENABLE Enable ssp DMA controller

8.36 mdev_uart.h File Reference

Universal Asynchronous Receiver Transmitter (UART) driver.

8.36.1 Detailed Description

UART is a piece of hardware that translates data between parallel and serial forms. It provides serial interface for communication to external world. UART driver provides interface to use all available UARTs.

8.36.2 Usage

A typical UART device usage scenario is as follows:

1. Verify that board file API board_uart_pin_config() located at wmsdk_bundle-x.y.z/wmsdk/src/boards/board_-file.c reflects the correct pinmux settings. If found missing, please add the appropriate pinmux configuration corresponding to the UART_ID.
   - -# Initialize the UART driver using uart_drv_init().
2. Open the UART device handle using uart_drv_open() call.
3. Use uart_drv_read() or uart_drv_write() calls to read UART data or to write data to UART respectively.
4. Close the UART device using uart_drv_close() after its use. uart_drv_close() call informs the UART driver to release the resources that it is holding. It is always good practice to free the resources after use.
5. Deinitialize the UART driver using uart_drv_deinit().

   Code snippet:

   Following code demonstrates basic read operation over UART0 UART0 is configured as 8 bits per character & baud-rate of 115200.

```
{
    int ret;
    uint8_t buf[512];
    mdev_t *uart_dev;
    ret = uart_drv_init(UART0_ID, UART_8BIT);
    if (ret != WM_SUCCESS)
       return;
    uart_dev = uart_drv_open(UART0_ID, 115200);
    if (uart_dev == NULL)
       return;
    len = uart_drv_read(uart_dev, buf, sizeof(buf));
    ret = uart_drv_close(uart_dev);
    if (ret != WM_SUCCESS)
       return;
    uart_drv_deinit(UART0_ID);
}
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Note

1. Specify standard baud rates such as 9600,115200..
2. When baudrate is 0 in uart_drv_open() call, UART driver will take default baudrate as 115200.
3. UART_ID_Type enumeration has values UART0_ID, UART1_ID, UART2_ID and UART3_ID, each of which represents individual UART device.
4. External buffer protection is needed in uart_drv_write().
5. The caller must provide the read buffer in uart_drv_read().

8.36.3 Function Documentation

8.36.3.1 int uart_drv_init (UART_ID_Type *id*, int *mode*)

Initialize UART Driver

This function initializes UART driver. And registers the device with the mdev interface.

Parameters

in	*id*	Port ID of UART.
in	*mode*	UART_8BIT or UART_9BIT.

Returns

WM_SUCCESS on success
-WM_FAIL on error.

8.36.3.2 void uart_drv_deinit (UART_ID_Type *id*)

De-initialize UART Driver

This function de-initializes UART driver. Note that if a UART device has been opened using uart_drv_open(), it should be first closed using uart_drv_close() before calling this function.

Parameters

in	*id*	Port ID of UART.

8.36.3.3 int uart_drv_xfer_mode (UART_ID_Type *port_id*, UART_DMA_MODE *dma_mode*)

Set UART Transfer Mode

UART supports DMA signalling with handshaking signals. This is an optional call to set uart transfer mode. By default UART transfer mode is set to UART_DMA_DISABLE mode. This call should be made after uart_drv_init() and before uart_drv_open() to over-write default transfer mode and to enable DMA transfer mode.

If DMA is enabled then set appropriate DMA block size using uart_drv_dma_rd_blk_size(). uart_drv_read() can read data from internal ringbuffer only if block size of data is received by uart driver.

Parameters

in	*port_id*	Port ID of UART.
in	*dma_mode*	UART_DMA_DISABLE or UART_DMA_ENABLE.

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS on success.
-WM_FAIL on error.

8.36.3.4 mdev_t* uart_drv_open (UART_ID_Type *port_id*, uint32_t *baud*)

Open handle to UART Device

This function opens the handle to UART device and enables application to use the device. This handle should be used in other calls related to UART device.

Parameters

in	*port_id*	Port ID of UART.
in	*baud*	Baud rate.

Returns

Handle to the UART device.
NULL on error.

8.36.3.5 int uart_drv_close (mdev_t * *dev*)

Close handle to UART Device

This function closes the handle to UART device and resets the UART.

Parameters

in	*dev*	Handle to the UART device returned by uart_drv_open().

Returns

WM_SUCCESS on success
-WM_FAIL on error.

8.36.3.6 uint32_t uart_drv_read (mdev_t * *dev*, uint8_t * *buf*, uint32_t *num*)

Read data from UART

This function is used to read data arriving serially over serial in line.

Parameters

in	*dev*	Handle to the UART device returned by uart_drv_open().
out	*buf*	Pointer to an allocated buffer of size equal to or more than the value of the third parameter num.
in	*num*	The maximum number of bytes to be read from the UART. Note that the actual read bytes can be less than this.

Returns

Number of bytes read.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.36.3.7 uint32_t uart_drv_write (mdev_t * *dev*, const uint8_t * *buf*, uint32_t *num*)

Write data to UART

This function is used to write data to the UART in order to transmit them serially over serial out line.

Parameters

in	*dev*	Handle to the UART device returned by uart_drv_open().
in	*buf*	Pointer to a buffer which has the data to be written out.
in	*num*	Number of bytes to be written.

Returns

Number of bytes written.

8.36.3.8 uint32_t uart_drv_get_portid (mdev_t * *dev*)

Get UART Port Id

This is used to get the port ID of the registered UART. Port ID of UART can have one of the values from UART_ID_Type enumeration mentioned in usage section.

Parameters

in	*dev*	Handle to the UART device returned by uart_drv_open().

Returns

UART Port Id.

8.36.3.9 void uart_drv_tx_flush (mdev_t * *dev*)

Flush UART transmit buffer

This is used to flush all the data from UART transmit FIFO buffer.

Parameters

in	*dev*	Handle to the UART device returned by uart_drv_open().

8.36.3.10 int uart_drv_rxbuf_size (UART_ID_Type *uart_id*, uint32_t *size*)

Set receive Ring-buffer size

This is optional call to change receive ringbuffer size. By default ringbuffer size is set to 1024/256 bytes; based on the value of the flag UARTDRV_SIZE_RX_BUF. This call should be made after uart_drv_init() and before uart_drv_open() to over-write default configuration.

Parameters

in	*uart_id*	Port ID of UART.
in	*size*	size of ring buffer in bytes.

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS on success.
-WM_FAIL on error.

8.36.3.11 int uart_drv_dma_rd_blk_size (UART_ID_Type *uart_id*, uint32_t *size*)

Set DMA read block length

This is optional call to change DMA block length size. By default DMA read block size is set to 128 bytes. This should be called only after enabling DMA mode. DMA mode can be enabled using uart_drv_xfer_mode(). Block size should be set in between 1 to 1023.

uart_drv_read() can read data from internal ringbuffer only if block size of data is received by uart driver. Though uart driver internally truncates software ringbuffer buffer length depending on DMA block size, it is recommended to set block size multiple of ringbuffer. Size of internal ringbuffer can be changed using uart_drv_rxbuf_size().

Parameters

in	*uart_id*	Port ID of UART.
in	*size*	DMA block size in between 1 to 1023.

Returns

WM_SUCCESS on success.
-WM_FAIL on error.

8.36.3.12 int uart_drv_set_opts (UART_ID_Type *uart_id*, uint32_t *parity*, uint32_t *stopbits*, **flow_control_t** *flow_control*)

Set UART line options

This is optional call to change UART line options. By default parity is set to NONE and stopbits to ONE. This call should be made after uart_drv_init() and before uart_drv_open() to over-write default configuration. Parameter flow_control is to enable and disable software flow control. If enabled, when uart_rx_buffer is almost full, it sends XOFF to the transimitter to pause the transmission and when enough free space is available in uart_rx_buffer, it sends XON to transmitter to resume the transmission. Default values of XON and XOFF are defined in mdev_uart.h, which are standard values 0x11 and 0x13. In case of binary data, if data contains binary strings same as 0x11 or 0x13, they are escaped using escape character, which is also defined in mdev_uart.h

Parameters

in	*uart_id*	Port ID of UART.
in	*parity*	Valid options are UART_PARITY_NONE, UART_PARITY_ODD, UART_PARITY_EVEN.
in	*stopbits*	Valid options are UART_STOPBITS_1, UART_STOPBITS_1P5_2.
in	*flow_control*	to set flow control mechanism from /ref flow_control_t options.

Returns

WM_SUCCESS on success
-WM_FAIL on error.

8.36.3.13 int uart_drv_blocking_read (UART_ID_Type *uart_id*, bool *is_blocking*)

Make uart_drv_read() call blocking

This is an optional call to make uart_drv_read() call blocking. By default uart_drv_read() call is non-blocking. This call should be made after uart_drv_init() & before uart_drv_open() to overwrite the default configuration. Usage of

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

this call is only for non-DMA mode UART. If enabled, uart_drv_read() will be blocking till it reads greater than 0 bytes from UART port.

Parameters

in	*uart_id*	Port ID of UART.
in	*is_blocking*	true to make uart_drv_read() blocking else false.

Returns

WM_SUCCESS on success.
-WM_FAIL on error.

8.36.3.14 int uart_drv_rx_buf_reset (mdev_t * *dev*)

API to reset the UART driver RX buffer This call is optional and may be used if the user wants to discard the old data stored in the RX buffer.

Parameters

in	*dev*	Handle to the UART device returned by uart_drv_open().

Returns

WM_SUCCESS on success.
-WM_FAIL on error.

8.36.4 Macro Documentation

8.36.4.1 #define UART_8BIT 0

Flag to enable UART in 8bit mode

8.36.4.2 #define UART_9BIT 1

Flag to enable UART in 9bit mode

8.36.4.3 #define UARTDRV_SIZE_RX_BUF 256 /* power of 2 */

Size of UART RX and TX ring buffers

8.36.4.4 #define UARTDRV_THRESHOLD_RX_BUF 64

Software flow control parameters

8.36.5 Enumeration Type Documentation

8.36.5.1 enum UART_DMA_MODE

Enum to enable and disable DMA mode for uart transfer

Enumerator

UART_DMA_DISABLE Disable UART DMA controller

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

UART_DMA_ENABLE Enable UART DMA controller

8.36.5.2 enum flow_control_t

Enum to select flow control method to be used for uart transfer

Enumerator

FLOW_CONTROL_NONE Flow control disabled

FLOW_CONTROL_SW Software flow control

FLOW_CONTROL_HW Hardware flow control

8.37 mdev_usb.h File Reference

Universal Serial Bus (USB) Driver.

8.37.1 Detailed Description

The USB Driver (for USB device only, no support for USB Host) is meant to use usb hardware to communicate with external devices connected on the USB bus.

8.37.2 Usage

A typical USB device usage scenario is as follows:

1. Initialize the USB driver using usb_drv_init(usb_class).
2. Open the USB device handle using usb_drv_open() call.
3. Use usb_drv_read() or usb_drv_write() calls to read USB data or to write data to USB respectively.
4. Close the USB device using usb_drv_close() after its use.

Code snippet:

Following code demonstrates how to use USB driver APIs for read/write operation, it reads 4 bytes and echoes them back.

```
{
 int ret, timeout;
 mdev_t *usb_dev;
 char buf[4];

 ret = usb_drv_init(USB_CDC_CLASS);
 if (ret != WM_SUCCESS)
        return;
 usb_dev = usb_drv_open("MDEV_USB");
 if (usb_dev == NULL)
        return;

 timeout = 60;
 ret = usb_drv_read(usb_dev, buf, sizeof(buf), timeout, 2);
 if (ret != 0 && ret != WM_E_USB_TIMEOUT_ERROR &&
        ret != WM_E_USB_CONNECTION_ERROR)
     usb_drv_write(usb_dev, buf, sizeof(buf), timeout, 2);

 usb_drv_close(usb_dev);
}
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.37.3 Function Documentation

8.37.3.1 int usb_drv_init (int *usb_class*)

Initialize the USB Driver

This function registers the USB driver with the mdev interface and initializes the USB device with the provided operational class.

Parameters

in	*usb_class*	for which device should be used. e.g USB_CDC_CLASS, USB_HID_CLASS or USB_MSC_CLASS

Returns

WM_Success on Success
-WM_FAIL on failure

8.37.3.2 mdev_t* usb_drv_open (const char * *name*)

Open USB driver

This returns a handle that should be used for other USB driver calls.

Parameters

name	Name of mdev usb driver. It should be "MDEV_USB" string.

Returns

handle to driver on Success
NULL otherwise

8.37.3.3 int usb_drv_close (mdev_t * *dev*)

Close USB Device

This function closes the device after use.

Parameters

in	*dev*	Handle to the USB driver returned by usb_drv_open()

Returns

WM_Success on Success

8.37.3.4 int usb_drv_read (mdev_t * *dev*, uint8_t * *buf*, uint32_t *nbytes*, uint32_t *timeout*, uint32_t *endpointnum*)

Read from USB

This function reads the specified number of bytes into the provided buffer from the USB device. The timeout parameter makes this call NON-BLOCKING.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*dev*	mdev_t handle to the driver
out	*buf*	Buffer for storing the values read
in	*nbytes*	Number of bytes to be read
in	*timeout*	for read operation in OS ticks
in	*endpointnum*	number from which read needs to be performed

Returns

number of bytes read on Success
WM_E_USB_TIMEOUT_ERROR or WM_E_USB_CONNECTION_ERROR otherwise

8.37.3.5 int usb_drv_write (mdev_t * *dev*, uint8_t * *buf*, uint32_t *nbytes*, uint32_t *timeout*, uint32_t *endpointnum*)

Write to USB

This function writes the specified number of bytes to the USB device. The timeout parameter makes this call NON-BLOCKING, write operation gets aborted on time out.

Parameters

in	*dev*	mdev_t handle to the driver
in	*buf*	Buffer containing data to be written
in	*nbytes*	Number of bytes to be written
in	*timeout*	for write operation in OS ticks
in	*endpointnum*	number on which write needs to be performed

Returns

number of bytes written on Success
WM_E_USB_CONNECTION_ERROR otherwise

8.37.4 Macro Documentation

8.37.4.1 #define WM_E_USB_TIMEOUT_ERROR (-1)

USB timeout error

8.37.4.2 #define WM_E_USB_CONNECTION_ERROR (-2)

USB connection error

8.37.5 Enumeration Type Documentation

8.37.5.1 enum usb_class

enum indicates usb device class

Enumerator

USB_CDC_CLASS USB Communication Device Class

USB_HID_CLASS USB Human Interface Device Class

USB_MSC_CLASS USB Mass Storage Class

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.38 mdev_usb_host.h File Reference

Universal Serial Bus (USB) Host driver.

8.38.1 Detailed Description

The USB host driver is used to initiate and arbitrate communication over USB-OTG port of MCU. It is meant to use USB hardware to communicate with external devices connected on the USB-OTG bus.

8.38.2 Usage

A typical USB host usage scenario is as follows:

1. Initialize USB host driver using class specific usb_host_xxx_drv_init().
2. Open the USB host handle using usb_host_drv_open() call.
3. Configure and start USB device using usb_host_drv_start() call
4. Use usb_host_drv_read() or usb_host_drv_write() calls to read USB data or to write USB data respectively.
5. Use usb_host_drv_block_read() or usb_host_drv_block_write() calls to read or write block/sector wise data from a USB device respectively.
6. Use usb_host_drv_info() to get USB device specific information
7. Stop USB device using usb_host_drv_stop() call
8. Close USB host using usb_host_drv_close() call after its use.

Code snippet:

Following code demonstrate how to use USB CDC host driver APIs for read/write operation. It sets line coding parameters as 9600,8,N,1 and reads 4 bytes of data and writes it back.

```
{
 int ret;
 usb_cdc_drv_data drv;
 mdev_t *usb_dev;
 char buf[4];

 drv.Parity = 0;
 drv.Baud = 9600;
 drv.StopBits = 1;
 drv.WordLength = 8;

 ret = usb_host_cdc_drv_init(&drv);
 if (ret != WM_SUCCESS)
    return;

 usb_dev = usb_host_drv_open("MDEV_USB_HOST");
 if (usb_dev == NULL)
    return;

 ret = usb_host_drv_start(usb_dev);
 if (ret != WM_SUCCESS)
    return;

 while((usb_host_drv_read(usb_dev, buf, sizeof(buf))) <= 0);
 ret = usb_host_drv_write(usb_dev, buf, sizeof(buf));
 if (ret < 0)
    return;
 usb_host_drv_stop(usb_dev);
 usb_host_drv_close(usb_dev);
}
```


Proprietary Information.
August 21, 2015 17:48

Copyright © 2015 Marvell

8.38.3 Function Documentation

8.38.3.1 int usb_host_uac_drv_init (void * *data*)

Initialize USB Host Driver for USB audio class (UAC) devices only

This function registers the USB Host driver with the mdev interface and initializes the USB device with the audio class.

Parameters

in	*data*	USB Host class specific data pointer

Returns

WM_SUCCESS on success
-WM_FAIL on failure

8.38.3.2 int usb_host_uvc_drv_init (void * *data*)

Initialize USB Host Driver for USB video class (UVC) devices only

This function registers the USB Host driver with the mdev interface and initializes the USB device with the video class.

Parameters

in	*data*	USB Host class specific data pointer

Returns

WM_SUCCESS on success
-WM_FAIL on failure

Note

Video class driver currently supports STREAM_MJPEG and STREAM_UNCOMPRESSED video formats only

8.38.3.3 int usb_host_cdc_drv_init (void * *data*)

Initialize USB Host Driver for USB communication class (CDC) devices only

This function registers the USB Host driver with the mdev interface and initializes the USB device with the communication class.

Parameters

in	*data*	USB Host class specific data pointer

Returns

WM_SUCCESS on success
-WM_FAIL on failure

8.38.3.4 int usb_host_msc_drv_init (void * *data*)

Initialize USB Host Driver for USB mass storage class (MSC) devices only

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

This function registers the USB Host driver with the mdev interface and initializes the USB device with the mass storage class.

Parameters

in	*data*	USB Host class specific data pointer

Returns

WM_SUCCESS on success
-WM_FAIL on failure

8.38.3.5 mdev_t* usb_host_drv_open (const char * *name*)

Open USB device

This function enables (power on) USB Host EHCI Controller.

This returns a handle that should be used for other USB Host driver calls.

Parameters

in	*name*	Name of mdev usb driver. It should be "MDEV_USB_HOST" string.

Returns

handle to driver on success
NULL otherwise

8.38.3.6 int usb_host_drv_start (mdev_t * *dev*)

Start USB device

In addition specific USB Class driver can do its own implementation, e.g. for video class this function will configure USB video device with configuration provided during init phase and starts video streaming.

Parameters

in	*dev*	mdev handle returned from call to usb_drv_open()

Returns

WM_SUCCESS on success
error code on failure

8.38.3.7 int usb_host_drv_stop (mdev_t * *dev*)

Stop USB device

In addition specific USB Class driver can do its own implementation, e.g. for video class this function will stop video streaming from end device, putting interface into zero bandwidth mode.

Parameters

in	*dev*	mdev handle returned from call to usb_drv_open()

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS on success
error code on failure

8.38.3.8 int usb_host_drv_info (mdev_t * *dev*, void * *buf*)

Get info from USB device

This function reads the class specific information into the provided buffer from the USB Host device.

Parameters

in	*dev*	mdev_t handle to the driver
out	*buf*	Buffer for storing retrived device information

Returns

WM_SUCCESS or negative error number

8.38.3.9 int usb_host_drv_read (mdev_t * *dev*, void * *buf*, uint32_t *nbytes*)

Read from USB device

This function reads the specified number of bytes into the provided buffer from the USB Host device. This call is NON-BLOCKING one.

Parameters

in	*dev*	mdev_t handle to the driver
out	*buf*	Buffer for storing the values read
in	*nbytes*	Number of bytes read out, 0 if data is not yet available, -1 on buffer overflow error

Returns

number of bytes read

Note

For USB Video Class, video data will start from frame boundary at first read or after a buffer overflow exception has occurred.

8.38.3.10 int usb_host_drv_write (mdev_t * *dev*, const void * *buf*, uint32_t *nbytes*)

Write to USB device

This function writes the specified number of bytes to the USB Host device

Parameters

in	*dev*	mdev_t handle to the driver
in	*buf*	Pointer to buffer to be written
in	*nbytes*	Number for bytes to be written

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

number of bytes written on success
-1 on failure

8.38.3.11 int usb_host_drv_block_read (mdev_t * *dev*, void * *buf*, uint32_t *block_no*, uint32_t *no_of_blocks*)

Block Read from USB device

This function reads the specified number of blocks/sectors into the provided buffer from the USB Host device. This call is BLOCKING one.

Parameters

in	*dev*	mdev_t handle to the driver
out	*buf*	Buffer for storing the values read
in	*block_no*	initial block/sector number from which to read
in	*no_of_blocks*	number of blocks/sectors to read

Returns

number of bytes read or negative value on error

8.38.3.12 int usb_host_drv_block_write (mdev_t * *dev*, const void * *buf*, uint32_t *block_no*, uint32_t *no_of_blocks*)

Block Write to USB device

This function writes the specified number of blocks/sectors from the provided buffer to the USB Host device. This call is BLOCKING one.

Parameters

in	*dev*	mdev_t handle to the driver
in	*buf*	Pointer to buffer to be written
in	*block_no*	initial block/sector number to which to write
in	*no_of_blocks*	number of blocks/sectors to write

Returns

number of bytes written or negative value on error

8.38.3.13 int usb_host_drv_close (mdev_t * *dev*)

Stop USB Host EHCI Controller

This function disables (power off) USB Host EHCI Controller.

Parameters

in	*dev*	mdev handle to the driver to be closed

Returns

WM_SUCCESS on success

8.38.4 Macro Documentation

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.38.4.1 #define USB_UVC_DEBUG 1

Enable/Disable debug logs for USB UVC Class Driver

8.38.4.2 #define USB_CDC_DEBUG 1

Enable/Disable debug logs for USB CDC Class Driver

8.39 mdev_wdt.h File Reference

WatchDog Timer (WDT) driver.

8.39.1 Detailed Description

A watchdog timer is a piece of hardware that can be used to automatically detect software anomalies and reset the processor if any occur. The watchdog timer regains control in case of system failure (due to a software error) to increase application reliability. The WDT can generate a reset or an interrupt when the counter reaches a given timeout value. WDT driver provides interface to the watchdog hardware.

8.39.2 Usage

A typical WDT device usage scenario is as follows:

1. Initialize the watchdog timer driver using wdt_drv_init().
2. Open the WDT device handle for use using wdt_drv_open() call.
3. Call wdt_drv_set_timeout() in order to set timeout index value. It should be a 4 bit value between 0 - 15.
4. Call wdt_drv_start() to start the WatchDog timer.
5. To strobe/restart WDT device call wdt_drv_strobe().

   Code snippet:

   Following code demonstrates how to configure WDT for timeout of 90 seconds.

```
{
    int ret;
    mdev_t *wdt_dev;
    ret = wdt_drv_init();
    if (ret != WM_SUCCESS)
       return;
    wdt_dev = wdt_drv_open("MDEV_WDT");
    if (wdt_dev == NULL)
       return;
    ret = wdt_drv_set_timeout(wdt_dev,0xF);
    if (ret != WM_SUCCESS)
       return;
    wdt_drv_start(wdt_dev);
}
```

8.39.3 Function Documentation

8.39.3.1 int wdt_drv_init (void)

Initialize WDT driver

This function initializes WDT driver. And registers the device with mdev interface.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS on success.

8.39.3.2 mdev_t* wdt_drv_open (const char * *name*)

Open handle to WDT device

This function opens the handle to WDT device and enables application to use the device. This handle should be used for other calls related to WDT device.

Parameters

in	*name*	Name of the driver to be opened. Name should be "MDEV_WDT" string.

Returns

NULL if error.
mdev_t handle otherwise.

8.39.3.3 int wdt_drv_set_timeout (mdev_t * *dev*, unsigned char *index*)

Set WDT Timeout

This function sets timeout value for watchdog timer.

Parameters

in	*dev*	Handle to the WDT device returned by wdt_drv_open().
in	*index*	for timeout value in between 0 - 15. If index is entered as 15 (0xF), the actual timeout is around 90 seconds. It reduces to half, after decrementing the index by 1 everytime.

Returns

WM_SUCCESS on success.
-WM_E_INVAL on error.

8.39.3.4 void wdt_drv_start (mdev_t * *dev*)

Start watchdog timer

This function starts WDT operation.

Parameters

in	*dev*	Handle to the WDT device returned by wdt_drv_open().

8.39.3.5 void wdt_drv_strobe (mdev_t * *dev*)

Strobe watchdog timer

This function strobes/restarts the watchdog timer.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*dev*	Handle to the WDT device returned by wdt_drv_open().

8.40 mdns.h File Reference

The mDNS/DNS-SD Responder-Querier.

8.40.1 Detailed Description

mDNS (Multicast DNS) and DNS-SD (DNS based Service Discovery) together enable Service Discovery in Zero configuration networks by using the standard DNS protocol packets but with minor changes.

8.40.2 Example Usage:

8.40.2.1 Working with mDNS Responder (Announcing the service)

The following code announces the service named wm_demo._smartenergy._tcp.local with the given TXT records' key-value pairs.

```
#include <mdns.h>
#include <wm_net.h>

struct mdns_service myservice;
char service_name[]="wm_demo";
char serv_type[]="smartenergy";
char keyvals[]="txtvers=2:path=/wm_demo:description=Wireless" \
                        "Microcontroller:Product=WM1";
char domain[]="local";
char dname[]="myhost";
int wm_mdns_start()
{
        int ret = 0;
        void *handle;
        memset(&myservice,0,sizeof(myservice));
        myservice.proto = MDNS_PROTO_TCP;
        myservice.servname = service_name;
        myservice.servtype = serv_type;
        myservice.keyvals = keyvals;
        myservice.port = 8888;
        myservice.domain = domain;
        handle = net_get_sta_handle();
        mdns_start(domain, dname);
        ret = mdns_announce_service(&myservice, handle);
        return ret;
}

int wm_mdns_stop()
{
        mdns_stop();
        return 0;
}
```

In order to announce a set of services at once, use mdns_announce_service_arr().

Service Discovery with Linux

The services published in the way described above can be discovered from a Linux machine connected in the same network using avahi-tools. For example, if all service instances belonging to a particular type of service in the network need to be discovered.

Following command looks for service instances of the type "_smartenergy._tcp" in the network. The second line shows the output for the command, which discovered the service instance published with the given service type.

```
avahi-browse _smartenergy._tcp
+ wlan0 IPv4 wm_demo                                _smartenergy._tcp    local
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.40.2.2 Working with mDNS Querier (Querying the service)

The following code monitors for the service of type "_smartenergy._tcp.local." Any state change to the monitored service is reported through a callback function wm_demo_mdns_query_cb.

```
#include <mdns.h>
#include <wm_net.h>

char fqst[]="_smartenergy._tcp.local";

static int wm_demo_mdns_query_cb(void *data, const struct mdns_service *s,
                        int status)
{
       wmprintf("Got callback for %s\r\n", s->servname);

       if (status == MDNS_DISCOVERED)
        wmprintf("DISCOVERED:\r\n");
       else if (status == MDNS_CACHE_FULL)
             wmprintf("NOT_CACHED:\r\n");
       else if (status == MDNS_DISAPPEARED)
                wmprintf("DISAPPEARED:\r\n");
       else if (status == MDNS_UPDATED)
                wmprintf("UPDATED:\r\n");
       else {
             wmprintf("Warning: unknown status %d\r\n", status);
             return WM_SUCCESS;
       }
       wmprintf("%s._%s._%s.%s.\r\n",s->servname, s->servtype,
        s->proto == MDNS_PROTO_UDP ? "udp" : "tcp",
        s->domain);
       wmprintf("at %d.%d.%d.%d:%d\r\n",
        (s->ipaddr >> 24) & 0xff, (s->ipaddr >> 16) & 0xff,
        (s->ipaddr >> 8) & 0xff, s->ipaddr & 0xff,
        htons(s->port));
       wmprintf("TXT : %s\r\n", s->keyvals ? s->keyvals : "no key vals");

       return WM_SUCCESS;
}

int wm_mdns_start()
{
     void *iface = net_get_sta_handle();
     mdns_start(NULL, NULL);
     iface = net_get_sta_handle();
     mdns_query_monitor(fqst, wm_demo_mdns_query_cb, NULL, iface);
}

int wm_mdns_stop()
{
       mdns_stop();
       return 0;
}
```

8.40.2.2.1 mDNS subtype base querier support

WMSDK mDNS querier also supports subtype base discovery. To browse for a subtype base query, "fqst" should be in following format: <subtype_name>._sub._<user_protocol>._<tcp or udp>.<domain>

In order to search for "_marvell" subtype based services chanage value of fqst in the querier code above to "_marvell._sub._http._tcp.local";

Publishing Service using Linux

Following command publishes service using avahi publish, which could be queried using above code snippet.

avahi-publish -s "MarvellService" _smartenergy._tcp 80

Following command publishes "subtype" based service using avahi publish, which could be queried by making changes in querier code snippet as described in mDNS subtype base querier support section.

avahi-publish -s –domain="local" –subtype="_marvell._sub._http._tcp" "MarvellService" "_http._tcp" 80

Proprietary Information.
August 21, 2015 17:48

Copyright © 2015 Marvell

8.40.2.3 Build Options

A number of build options are available in order to optimize footprint, specify the target system, etc. These are conventionally provided as config options as opposed to being defined directly in a header file. They include:

CONFIG_MDNS_QUERY: Developers who require the ability to query for services as opposed to just responding to such queries should define MDNS_QUERY. This enables the mdns_query_∗ functions described below. These functions will return -WM_E_MDNS_NOIMPL if the mdns library was built without MDNS_QUERY defined. By default this option is disabled.

CONFIG_MDNS_CHECK_ARGS: Define this option to enable various error checking on the input. Developers may wish to enable this during development to ensure that various inputs such as host names are legal. Then, if the inputs are not going to change, this option can be turned off to save code space. By default this option is disabled.

CONFIG_XMDNS: Define this option to select Extended-mDNS operations. Extended mDNS extends specification of mDNS to site-local scope in order to support multi-hop LANs that forward multicast packets but do not provide a unicast DNS service. Note that selecting Extended-mDNS requires top-level domain name ".site", instead of m-DNS' ".local" domain name. Both domains are functionally disjoint, both from a name space and address space perspective. Application could register to only one domain name i.e enabling ex-mDNS would mean .local will not work for querier as well as responder.

CONFIG_MDNS_DEBUG: Define this option to include debugging and logging. By default this option is disabled.

8.40.3 Function Documentation

8.40.3.1 int mdns_start (const char ∗ *domain*, char ∗ *hostname*)

mdns_start

Start the responder thread (and the querier thread if querying is enabled).

Note that the mdns_start() function must be called only after the network stack is up.

The responder thread wait for application's request to announce services. Using mdns_announce_service() and mdns_announce_service_arr() call, services are added to mDNS service list and are announced on a given interface.

The querier thread just opens a socket and then waits until a request for monitoring any given service type or subtype is received from the application. When received, the same is queried and the responses are processed, cached and monitored.

Parameters

domain	domain name string. If this value is NULL, the domain ".local" will be used. The domain must not exceed MDNS_MAX_LABEL_LEN bytes.
hostname	string that is the hostname to resolve. This would be the "foo" in "foo.local", for example. The hostname must not exceed MDNS_MAX_LABEL_LEN bytes. If hostname is NULL, the responder capability will not be launched, only query support will be enabled. This is useful if only the query functionality is desired.

Returns

WM_SUCCESS for success or mdns error code

NOTES:

The domain and hostname must persist and remain unchanged between calls to mdns_start and mdns_stop. Hence define these variables in global memory.

While mdns_start returns immediately, the hostname and any servnames may not be unique on the network. In the event of a conflict, the names will be appended with an integer. For example, if the hostname "foo.local" is taken, mdns will attempt to claim "foo-2.local", then foo-3.local, and so on until the conflicts cease. If mdns gets all the way to foo-9.local and still fail, it waits for 5 seconds (per the mDNS specification) and then starts back at foo.local. If a

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

developer anticipates a network to have many of her devices, she should devise a sensible scheme outside of mdns to ensure that the names are unique.

8.40.3.2 void mdns_stop (void)

mdns_stop

Halt the mDNS responder thread (and querier thread if querying is enabled), delete the threads and close the sockets

Any services being monitored will be unmonitored.

8.40.3.3 int mdns_query_monitor (char * *fqst*, mdns_query_cb *cb*, void * *data*, void * *iface*)

mdns_query_monitor

Query for and monitor instances of a service

When instances of the specified service are discovered, the specified query callback is called as described above.

Parameters

fqst	Pointer to a null-terminated string specifying the fully-qualified service type. For example, "_-http._tcp.local" would query for all http servers in the ".local" domain.
cb	an mdns_query_cb to be called when services matching the specified fqst are discovered, are updated, or disappear. cb will be passed the opaque data argument described below, a struct mdns_service that represents the discovered service, and a status code.
data	a void * that will passed to cb when services are discovered, are updated, or disappear. This can be anything that the user wants, such as pointer to a custom internal data structure.
iface	Interface handle on which services are to be queried. Interface handle can be obtained from net_get_sta_handle() or net_get_uap_handle() function calls

Returns

WM_SUCCESS: the query was successfully launched. The caller should expect the mdns_query_cb to be invoked as instances of the specified service are discovered.
WM_E_MDNS_INVAL: cb was NULL or fqst was not valid.
-WM_E_MDNS_NOMEM: CONFIG_MDNS_MAX_SERVICE_MONITORS is already being monitored. Either this value must be increased, or a service must be unmonitored by calling mdns_query_unmonitor.
-WM_E_MDNS_INUSE: The specified service type is already being monitored by another callback, and multiple callbacks per service are not supported.
-WM_E_MDNS_NORESP: No response from the querier. Perhaps it was not launched or it has crashed.

Note: multiple calls to mdns_query_service_start are allowed. This enables the caller to query for more than just one service type.

8.40.3.4 void mdns_query_unmonitor (char * *fqst*)

mdns_query_unmonitor

Stop monitoring a particular service

Parameters

fqst	The service type to stop monitoring, or NULL to unmonitor all services.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Note

Suppose a service has just been discovered and is being processed while the call to mdns_query_monitor is underway. A callback may be generated before the service is unmonitored.

8.40.3.5 int mdns_announce_service (struct mdns_service * *service*, void * *iface*)

mdns_announce_service

Announce single mDNS service on an interface

This function checks validity of service and if service is invalid then it will return with appropriate error code. Function sends command on mDNS control socket for announcing service on "iface" interface. mDNS then announces the service by following prescribed set of steps mentioned below:

1. Send probing packets for a given service or services 3 times, spaced 250 ms apart.
2. If a conflict is received for any service record, resolve them by appending a number to that particular record to make it unique. Go to step 1.
3. Announce the service or services. In order to announce multiple services simultaneously, use mdns_announce_service_arr(). This is recommended to reduce network congestion. Maximum no. of services that can be announced per interface is MAX_MDNS_LST

Parameters

service	Pointer to mdns_service structure corresponding to service to be announced. Structure must persist and remain unchanged between calls to mdns_start and mdns_stop. Hence define this variable in global memory.
iface	Pointer to interface handle on which service is to be announced. Interface handle can be obtained from net_get_sta_handle or net_get_uap_handle function calls.

Returns

WM_SUCCESS: success
-WM_E_MDNS_INVAL: input was invalid. Perhaps a label exceeded MDNS_MAX_LABEL_LEN, or a name composed of the supplied label exceeded MDNS_MAX_NAME_LEN. Perhaps hostname was NULL and the query capability is not compiled. Perhaps service pointer or iface pointer is NULL
-WM_E_MDNS_BADSRC: one of the service descriptors in the service was invalid. Perhaps key/val pair exceeded MDNS_MAX_KEYVAL_LEN
-WM_E_MDNS_TOOBIG: The combination of name information and service descriptors does not fit in a single packet, which is required by this implementation
-WM_FAIL: No space to add new interface or a new service in a given interface.
-WM_E_MDNS_FSOC: failed to create loopback control socket
-WM_E_MDNS_FBIND: failed to bind control socket

8.40.3.6 int mdns_deannounce_service (struct mdns_service * *service*, void * *iface*)

mdns_deannounce_service

Deannounce single service from an interface

This function sends command on mDNS control socket to deannounce a given service on "iface" interface. mDNS then de-announces the service by sending a good bye packet with ttl=0 so that mDNS clients can remove the services from their mDNS cache table. Function removes service from a list of services associated with that interface. In order to de-announce multiple services simultaneously, use mdns_deannounce_service_arr(). This is recommended to reduce network congestion.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

service	Pointer to mdns_service structure corresponding to service to be de-announced
iface	Pointer to interface handle on which service is to be de-announced. Interface handle can be obtained from net_get_sta_handle or net_get_uap_handle function calls

Returns

WM_SUCCESS: success
-WM_E_MDNS_INVAL: invalid parameters. Perhaps service pointer or iface pointer is NULL
-WM_E_MDNS_FSOC: failed to create loopback control socket
-WM_E_MDNS_FBIND: failed to bind control socket

8.40.3.7 int mdns_iface_state_change (void * *iface*, enum iface_state *state*)

mdns_iface_state_change

Send interface state change event to mdns

This will start or stop mdns state machine for given interface. Before calling this function for first time, interface and services should be added to mDNS using mdns_announce_service() or mdns_announce_service_arr().

For UP state, mdns responder will initially send init probes for all announced services to check for name conflict in services. If there are any conflicts it will resolve them by appending integer after service name. After checking name conflicts, it will announce services to multicast group.

DOWN state will send good bye packet corresponding to all announced services on a given interface. Note that taking an interface DOWN will not remove services from a list of services associated with that interface. In order to do so, use mdns_deannounce_service() or mdns_deannounce_service_arr() or mdns_deannounce_service_all().

REANNOUNCE will forcefully reannounce services. REANNOUNCE do not send init probes and directly re-announces the services. Generally REANNOUNCE should be used during when TXT records are updated.

In the events like link-lost, DHCP change event, or change in host name or the service name, application should repeat the initial probing sequence. This can be achieved by calling this function with DOWN state, followed by calling it again with UP state. This would help to resolve service name conflicts if introduced by other devices. Also the previous service entry in mDNS clients will be removed and new updated entry will be added.

Parameters

iface	interface handle
state	valid interface state from iface_state

Returns

WM_SUCCESS: success
-WM_E_MDNS_FSOC: failed to create loopback control socket
-WM_E_MDNS_FBIND: failed to bind control socket

8.40.3.8 int mdns_set_txt_rec (struct mdns_service * *s*, char * *keyvals*, char *separator*)

Set mDNS TXT Records

This function sets the TXT record field for a given mDNS service. mDNS TXT record string is in-place converted and stored in a format defined by TXT resource record format.

Note

This function MUST be called before adding or announcing the mDNS service. Otherwise the resultant behavior can be erroneous.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

s	Pointer to mDNS service
keyvals	TXT record string consisting of one or more key value pairs, which can be separated by a separator like ':', '|'. For example, `char keyvals[] = "key1=value1|key2=value2|key3=value3"` The separator value is also passed to this function as mentioned below. keyvals string must persist between the calls to mdns_start and mdns_stop. Hence define these variables in global memory
separator	The separator used to separate individual key value pairs in the above mentioned TXT record string. If TXT record contains single key value pair then set separator value appropriately. For example, it can be set to '|' if the character '|' is not used in key value pair.

Returns

WM_SUCCESS on success
-WM_E_MDNS_TOOBIG if length of TXT record exceeds the permissible limit as specified by MDNS_MAX_KEYVAL_LEN
-WM_E_MDNS_INVAL if value of TXT record string is invalid

8.40.4 Macro Documentation

8.40.4.1 #define MDNS_MAX_LABEL_LEN 63 /∗ defined by the standard ∗/

Maximum length of labels

A label is one segment of a DNS name. For example, "foo" is a label in the name "foo.local.". RFC 1035 requires that labels do not exceed 63 bytes.

8.40.4.2 #define MDNS_MAX_NAME_LEN 255 /∗ defined by the standard : 255∗/

Maximum length of names

A name is a list of labels such as "My Webserver.foo.local" or mydevice.local. RFC 1035 requires that names do not exceed 255 bytes.

8.40.4.3 #define MDNS_MAX_KEYVAL_LEN 255 /∗ defined by the standard : 255∗/

Maximum length of key/value pair

TXT records associated with a service are populated with key/value pairs. These key/value pairs must not exceed this length.

8.40.4.4 #define MAX_MDNS_LST 5

Maximum no. of services allowed to be announced on a single interface.

8.40.4.5 #define MAX_INTERFACE 2

Maximum no. of different interfaces supported by mDNS.

8.40.4.6 #define MDNS_PROTO_TCP 0

protocol values for the proto member of the mdns_service descriptor TCP Protocol

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.40.4.7 #define MDNS_PROTO_UDP 1

UDP Protocol

8.40.5 Typedef Documentation

8.40.5.1 typedef int(∗ mdns_query_cb)(void ∗data, const struct mdns_service ∗s, int status)

Note Configuring CONFIG_MDNS_SERVICE_CACHE_SIZE in mDNS Querier

This is a menuconfig option and can be configured from "make menuconfig" -> Modules -> MDNS -> mDNS Full-Fledged Query(monitoring) API -> Maximum number of service instances that can be monitored.

Suppose CONFIG_MDNS_SERVICE_CACHE_SIZE is 16 and that a user has invoked mdns_query_monitor to monitor services of type _http._tcp.local. Assume the query callback handler returns WM_SUCCESS for all the instances discovered.

Further,suppose that this particular domain has 17 instances of this type. The first 16 instances to be discovered will result in 16 callbacks with the status MDNS_DISCOVERED. These instances will be cached and monitored for updates, disappearance, etc. When the 17th instance is discovered, the callback will be called as usual, but the status will be MDNS_CACHE_FULL, and the service will not be monitored. While a best effort is made to deliver all of the service information, the mdns_service may be incomplete. Specifically, the ipaddr may be 0 and the service name may be "". Further, the callback may be called again if the 17th instance of the service announces itself on the network again. If one of the other services disappears, the next announcement from the 17th instance will result in a callback with status MDNS_DISCOVERED, and from that point forward it will be monitored.

So what's the "best" value for CONFIG_MDNS_SERVICE_CACHE_SIZE? This depends on the application and on the field in which the application is deployed. If a particular application knows that it will never see more than 6 instances of a service, then 6 is a fine value for CONFIG_MDNS_SERVICE_CACHE_SIZE. In this case, callbacks with a status of MDNS_CACHE_FULL would represent a warning or error condition. Similarly, if an application cannot handle any more than 10 instances of a service, then CONFIG_MDNS_SERVICE_CACHE_SIZE should be 10 and callbacks with a status of MDNS_CACHE_FULL can be ignored. If the maximum number of service instances is not known, and the application retains its own state for each instance of a service, it may be able to use that state to do the right thing when the callback status is MDNS_CACHE_FULL.

For applications with constrained memory ,a point to note is that each service instance requires little above 1K bytes. This should be considered while deciding the CONFIG_MDNS_SERVICE_CACHE_SIZE.

The default value of CONFIG_MDNS_SERVICE_CACHE_SIZE is set to 2. mdns query callback

A user initiates a query for services by calling the mdns_query_monitor function with a fully-qualified service type, an mdns_query_cb, and an opaque argument. When a service instance is discovered, the query callback will be invoked with following arguments:

Parameters

data	a void ∗ that was passed to mdns_query_monitor(). This can be anything that the user wants, such as pointer to a custom internal data structure.
s	A pointer to the struct mdns_service that was discovered. The struct mdns_service is only valid until the callback returns. So if attributes of the service (such as IP address and port) are required by the user for later use, they must be copied and preserved elsewhere.
status	A code that reports the status of the query. It takes one of the following values:

MDNS_DISCOVERED: The mdns_service s has just been discovered on the network and will be monitored by the mdns stack.

MDNS_UPDATED: The mdns_service s, which is being monitored, has been updated in some way (e.g., it's IP address has changed, it's key/value pairs have changed.)

MDNS_DISAPPEARED: The mdns_service has left the network. This usually happens when a service has shut down, or when it has stopped responding queries. Applications may also detect this condition by some means outside of mdns, such as a closed TCP connection.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

MDNS_CACHE_FULL: The mdns_service has been discovered. However, the number of monitored service instances has exceeded CONFIG_MDNS_SERVICE_CACHE_SIZE. So the returned mdns_service may not be complete. See NOTES below on other implications of an MDNS_CACHE_FULL status.

NOTES:

The query callback should return WM_SUCCESS in the case where it has discovered service of interest (MDNS_DISCOVERED, MDNS_UPDATED status). If the callback return non-zero value for MDNS_DISCOVERED and MDNS_UPDATED status codes, that particular service instance is not cached by the mDNS querier. This is required as each cached service instance takes little above 1KB memory and the device can't monitor large number of service instances.

Callback implementers must take care to not make any blocking calls, nor to call any mdns API functions from within callbacks.

8.40.6 Enumeration Type Documentation

8.40.6.1 enum iface_state

mDNS Interface State mDNS interface state can be changed by using mdns_iface_state_change() function. For details about when to use the enum please refer to documentation for mdns_iface_state_change().

Enumerator

UP UP the interface and announce services mDNS will probe and announce all services announced via mdns_announce_service() and/or mdns_announce_service_arr(). mDNS will go through entire probing sequence explained in above functions. Interface state can be changed to UP, if its DOWN.

DOWN DOWN the interface and de-announce services mDNS sends good bye packet with ttl=0 so that mDNS clients can remove the services from their mDNS cache table.

REANNOUNCE Forcefully re-announce services This state should be used after services are already announced and force announcement is needed due to some reason. mDNS will not perform probing sequence, as it does in case of UP, and will directly re-announce services.

8.40.6.2 enum wm_mdns_errno

MDNS Error Codes

Enumerator

WM_E_MDNS_INVAL invalid argument

WM_E_MDNS_BADSRC bad service descriptor

WM_E_MDNS_TOOBIG not enough room for everything

WM_E_MDNS_NOIMPL unimplemented feature

WM_E_MDNS_NOMEM insufficient memory

WM_E_MDNS_INUSE requested resource is in use

WM_E_MDNS_NORESP requested resource is in use

WM_E_MDNS_FSOC failed to create socket for mdns

WM_E_MDNS_FREUSE failed to reuse multicast socket

WM_E_MDNS_FBINDTODEVICE failed to bind mdns socket to device

WM_E_MDNS_FBIND failed to bind mdns socket

WM_E_MDNS_FMCAST_JOIN failed to join multicast socket

WM_E_MDNS_FMCAST_SET failed to set multicast socket

WM_E_MDNS_FQUERY_SOC failed to create query socket

WM_E_MDNS_FQUERY_THREAD failed to create mdns thread

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.41 nw_utils.h File Reference

Networking CLIs.

8.41.1 Detailed Description

Collection of networking CLIs like ping etc.

8.41.2 Function Documentation

8.41.2.1 int nw_utils_cli_init ( )

Register Network Utility CLI commands.

Register the Network Utility CLI commands. Currently, only ping command is supported.

Note

This function can only be called by the application after wlan_init() called.

Returns

WM_SUCCESS if the CLI commands are registered
-WM_FAIL otherwise (for example if this function was called while the CLI commands were already registered)

8.41.2.2 int nw_utils_cli_deinit ( )

Unregister Network Utility CLI commands.

Unregister the Network Utility CLI commands.

Returns

WM_SUCCESS if the CLI commands are unregistered
-WM_FAIL otherwise

8.42 overlays.h File Reference

Overlay Manager.

8.42.1 Detailed Description

This provides support for loading various overlays from flash to RAM. This is very useful in scenarios where two or more sections are mutually exclusive to each other and have to be used only one at a time. The overlay manager can be instructed to load the required section in ram as desired.

An overlay can contain multiple ranges. Each range within an overlay can be loaded at the same ram location. Each range may contain text, data and bss. The overlay_load() function can be used to load a given range into RAM.

Application developers may optionally use the ov_utl utility (wmsdk/tools/bin) to create and manage overlays.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.42.2 Function Documentation

8.42.2.1 int overlay_load (struct overlay * *ov*, unsigned char *index*)

Load Overlay

The overlay load function loads an overlay from the flash into ram. In the process it overwrites any contents already in the flash.

Note

This function assumes that the flash address are relative to the start of the firmware binary before any headers are appended to the firmware image. This eases the linker script since this care need not be taken in the script. This function also works in conjunction with the active passive upgrades. It checks the boot2 setup to determine which partition have we loaded from, and picks the overlay contents from the appropriate location.
Along with loading the text and data sections from flash it also zeroes out the bss section.

Parameters

ov	The overlay structure that describes the overlay
index	The range that should be loaded from this overlay

Returns

WM_SUCCESS on success, error otherwise

8.43 p2p.h File Reference

Wi-Fi Direct/P2P.

8.43.1 Detailed Description

Wi-Fi Direct (P2P) is a standard that defines a new way for Wi-Fi devices to work together. Wi-Fi devices will be able to make direct connections to one another quickly and conveniently to do things like print, sync, and share content even when an access point or router is unavailable. Wi-Fi Direct connections will work at typical Wi-Fi speeds and range, protected by WPA2 security protocols.

Two methods that are supported by SDK are:

- PIN Method: In this method the same PIN (Personal Identification Number) is entered on the both Wi-Fi Direct devices.
- PBC Method: In which the user simply has to push a button, either an actual or virtual one, on both the Wi-Fi Direct devices.

8.43.2 Usage

In a typical implementation P2P thread is started by calling p2p_start. A valid initialized p2p_config structure needs to be passed to this function. This structure contains a callback handler which is invoked by the P2P thread on occurrence of various events. Once the P2P thread is started, applications can scan for P2P devices or group in the range using p2p_scan. A connection attempt can be made to another p2p device using the pin or pbc method in p2p_connect. If auto_go option is enabled in p2p_config then P2P thread starts as Group Owner, else Group Owner negotiation protocol is performed to determine the Wi-Fi Direct device role (client/AP).

On successful P2P session, callback handler is invoked with P2P_SESSION_SUCCESSFUL event with wlan_network structure that contains all the security information for the network with which P2P session was attempted. After this p2p_stop should be called to terminate the P2P thread.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Note

P2P implementation lets application decide PIN policy. PIN could either be generated by the application, or by the access point or can be statically generated at device manufacturing time. p2p_generate_pin and p2p_-validate_pin utility functions can be used by the application to generate and validate PIN.
P2P thread internally uses multiple dynamic allocations. Please do not reduce the heap size below 64KB if the application wishes to use P2P.

8.43.3 Function Documentation

8.43.3.1 int p2p_start (struct p2p_config * *p2p_conf*)

Start P2P Service

This makes the device discoverable to other devices in the vicinity.

Parameters

p2p_conf	A pointer to P2P custom configuration structure

Returns

WM_SUCCESS if successful, -WM_FAIL otherwise

8.43.3.2 int p2p_stop ( )

Stop P2P Service

Returns

WM_SUCCESS if successful, -WM_FAIL otherwise

8.43.3.3 struct p2p_scan_result* p2p_scan (int * *no*)

Scan P2P Devices

Fetches the table that has entries for all the P2P devices or groups in the vicinity. It is the responsibility of the application to free this list. One of the entries in this list can be used as a parameter to p2p_connect.

Parameters

out	*no*	number of entries in the array pointed by the return value.

Returns

Pointer to the result structure.

8.43.3.4 int p2p_connect (enum p2p_session_command *pbc*, uint32_t *pin*, struct p2p_scan_result * *remote*)

Connect to P2P Device

Connect to a remote p2p device. This function is typically called whenever the users pushes the p2p button or enters p2p pin.

If the device is not a part of any P2P group to begin with the P2P GO negotiation is performed to determine the Group Owner. The caller is notified in the p2p_callback if the device is a group owner or a station.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

If the device is a group owner, the P2P thread starts the corresponding micro-AP network and the DHCP server on this interface. If the device is a station, the P2P thread establishes the connection with the group owner.

Parameters

pbc	Set to 1 if Push-button session is desired
pin	Ignore if pbc is 1. If pbc is 0, this indicates the pin that should be used.
remote	The remote device to connect to. If this is set to NULL, session is attempted with the first device.

Returns

WM_SUCCESS if successful, -WM_FAIL otherwise

8.43.3.5 int p2p_disconnect ( )

Disconnect from P2P Group

Disconect from the current P2P group. If the device is a GO, the entire group is taken down.

Returns

WM_SUCCESS if successful, -WM_FAIL otherwise

8.43.3.6 int p2p_generate_pin (uint32_t * *p2p_pin*)

Generate 8 digit P2P PIN value with random number generator

Parameters

p2p_pin	Generated 8 digit P2P PIN value

Returns

WM_SUCCESS if successful, -WM_FAIL otherwise

8.43.3.7 int p2p_validate_pin (uint32_t *p2p_pin*)

Validate checksum of PIN

Parameters

p2p_pin	P2P PIN value

Returns

WM_SUCCESS if successful, -WM_FAIL otherwise

8.43.4 Macro Documentation

8.43.4.1 #define P2P_LOG(...) wmlog("p2p", ##__VA_ARGS__)

P2P log

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.43.4.2 #define P2P_GROUP_OWNER_MASK 0x01

Check Bit GO set by peer Bit 0

8.43.4.3 #define MAX_DEVICE_NAME_LEN 32

Maximum P2P device name length

8.43.5 Enumeration Type Documentation

8.43.5.1 enum p2p_session_command

enum : P2P session commands

Enumerator

CMD_P2P_PIN Command to start P2P PIN session

CMD_P2P_PBC Command to start P2P PBC session

8.43.5.2 enum p2p_event

enum : P2P events

Enumerator

P2P_STARTED P2P thread started

P2P_AUTO_GO_STARTED P2P Auto GO started

P2P_DEVICE_STARTED P2P Device started

P2P_CLIENT_STARTED P2P Client started

P2P_GO_STARTED P2P GO started

P2P_SESSION_STARTED P2P PBC/PIN Session started

P2P_SESSION_PIN_CHKSUM_FAILED P2P PIN checksum failed

P2P_SESSION_ABORTED P2P Session aborted

P2P_SESSION_TIMEOUT P2P Session registration timeout

P2P_SESSION_SUCCESSFUL P2P Session attempt successful

P2P_AP_SESSION_SUCCESSFUL P2P AP Session attempt successful

P2P_SESSION_FAILED P2P Session failed

P2P_FAILED P2P thread start failed

P2P_FINISHED P2P thread stopped

8.44 partition.h File Reference

Partition Management.

8.44.1 Detailed Description

This defines functions for managing the flash partitions.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.44.2 Usage

Applications can define and use their own flash layout using the flashprog tool. Each partition in the layout has a name, type, start address and a size associated with it. Applications can use the part_get_layout_by_name() or part_get_layout_by_id() functions to retrieve pointers to the partition entry. This partition entry can be converted to a flash descriptor using the part_to_flash_desc() function. All components that use the flash accept the flash descriptor to recognize what portion of the flash they use.

Additionally, the partition entries store information about active-passive upgrades. Applications can define their own components in the flash in an active-passive manner. Applications can then retrieve the active component using the part_get_active_partition() API. If after an upgrade happens the partitions need to be toggled then the part_get_active_partition() API is used.

8.44.3 Function Documentation

8.44.3.1 int part_init (void)

Initialize the partitioning module

Returns

WM_SUCCESS on success or error code

8.44.3.2 void part_to_flash_desc (struct partition_entry * *p*, flash_desc_t * *f*)

Convert from partition to flash_descriptor

All the components that own a portion of flash in the SDK use the flash descriptor function to identify this portion. This function converts a partition_entry into a corresponding flash descriptor that can be passed to these modules.

Parameters

in	*p*	a pointer to a partition entry
out	*f*	a pointer to a flash descriptor

8.44.3.3 struct partition_entry* part_get_layout_by_id (enum flash_comp *comp*, short * *start_index*)

Get partition entry by ID

Retrieve the pointer to a partition entry using an ID. This function can be called to repetitively to retrieve more entries of the same type. The value of start_index is modified by the function so that search starts from this index in the next call.

Parameters

in	*comp*	the flash component to search for
in,out	*start_index*	the start index for the search. The first call to this function should always have a start index of 0.

Returns

a pointer to a partition entry on success, null otherwise.

8.44.3.4 struct partition_entry* part_get_layout_by_name (const char * *name*, short * *start_index*)

Get partition entry by name

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Retrieve the pointer to a partition entry using a name. This function can be called to repetitively to retrieve more entries of the same type. The value of start_index is modified by the function so that search starts from this index in the next call.

Parameters

in	*name*	the flash component's name as mentioned in the layout.
in,out	*start_index*	the start index for the search. The first call to this function should always have a start index of 0.

Returns

a pointer to a partition entry on success, null otherwise.

8.44.3.5 struct partition_entry* part_get_active_partition (struct partition_entry * *p1*, struct partition_entry * *p2*)

Find the active partition

Given two partition entries that are upgrade buddies (that is, have the same friendly name), this function returns the partition that is currently marked as active.

Parameters

in	*p1*	Pointer to a partition entry
in	*p2*	Pointer to the upgrade buddy for the first partition entry

Returns

a pointer to the active partition entry on success, null otherwise.

8.44.3.6 int part_set_active_partition (struct partition_entry * *p*)

Set active partition

Given a pointer to a currently passive partition entry, this function marks this partition entry as active.

Parameters

in	*p*	Pointer to partition entry object

Returns

-WM_E_INVAL if invalid parameter.
WM_SUCCESS if operation successful.

8.44.4 Enumeration Type Documentation

8.44.4.1 enum flash_comp

The various components in a flash layout

Enumerator

FC_COMP_BOOT2 The secondary stage boot loader to assist firmware bootup

FC_COMP_FW The firmware image. There can be a maximum of two firmware components available in a flash layout. These will be used in an active-passive mode if rfget module is enabled.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

FC_COMP_WLAN_FW The wlan firmware image. There can be one wlan firmware image in the system. The contents of this location would be downloaded to the WLAN chip.

FC_COMP_FTFS The FTFS image.

FC_COMP_PSM The PSM data

FC_COMP_USER_APP Application Specific component

8.45 provisioning.h File Reference

Provisioning Support.

8.45.1 Detailed Description

8.45.2 Introduction

Provisioning process lets user configure the WMSDK enabled device to connect to his home network first time. Provisioning library in WMSDK provides implementation of three provisioning methods to the application developer. These methods are:

- micro-AP network based provisioning
- WPS based provisioning
- EZConnect (Sniffer-based) provisioning

Of the above methods, the provisioning library allows co-existence of micro-AP and WPS provisioning methods.

8.45.2.1 WLAN network (micro-AP) based provisioning

In this method, a network is hosted by WMSDK enabled device. Clients can associate with this network and use the provided HTTP APIs to configure the home network properties on to the device. These HTTP APIs let users scan the environment to find available access points, choose the access point of the interest, provide security passphrase for the access point and finally push the settings to the device to complete provisioning. These HTTP APIs are listed in the Marvell System API guide.

For microAP based provisioning, a secure network can be hosted on the device thus ensuring that the data transferred over the air during the provisioning process is encrypted.

8.45.2.2 WPS based provisioning

WiFi Protected Setup is a standard that attempts to allow easy yet secure association of WiFi clients to the access point. Provisioning library allows developers to extend WPS association methods to the end-user. It supports both a) PIN and b) Push-button methods. These are described below.

- PIN Method: In this method a 8-digit numerical PIN should be used on both device and access point. This PIN can be static or dynamically generated on the device. Depending on the capabilities of the device the PIN can be printed or displayed on the device screen. When the same PIN is entered within interval of 2 minutes on the access point and WPS session is started, association succeeds.
- Push-button Method: In this method a physical or virtual pushbutton on the access point as well as on the device is pressed within interval of 2 minutes, the association succeeds.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.45.2.3 EZConnect provisioning

In this method, the device acts as a sniffer and captures all the packets on the air looking for a certain signature. A smart phone application (Android / iOS) is then used to transmit the target network credentials in a certain format. The target network credentials can be encrypted using a pre-shared key. The device sniffs these packets off the air, decrypts the data and establishes connection with the target network.

8.45.3 Function Documentation

8.45.3.1 int prov_start (struct provisioning_config * *pc*)

Starts provisioning with provided configuration

Parameters

pc	Pointer to provisioning_config structure that contains application specific provisioning configuration.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS if the call is successful -WM_FAIL in case of error

Note

The application should not create this structure instance through local variable. Provisioning thread needs to access this throughout its lifetime.

8.45.3.2 void prov_finish (void)

Stops provisioning thread

8.45.3.3 int prov_ezconnect_start (struct provisioning_config * *pc*)

Start EZConnect Provisioning

This function starts the EZConnect provisioning process.

Parameters

pc	Pointer to provisioning_config structure that contains application specific provisioning configuration.

Returns

WM_SUCCESS if the call is successful -WM_FAIL in case of error

Note

The application should not create this structure instance through local variable. Provisioning thread needs to access this throughout its lifetime.

8.45.3.4 void prov_ezconnect_finish (void)

Stops ezconnect provisioning thread

8.45.3.5 int prov_get_scan_config (struct prov_scan_config * *scan_cfg*)

Returns currently configured scan configuration

Parameters

scan_cfg	Current configuration is copied into the structure pointed by this pointer

Returns

WM_SUCCESS if the call is successful -WM_FAIL in case of error

8.45.3.6 int prov_set_scan_config (struct prov_scan_config * *scan_cfg*)

Sets scan configuration

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

scan_cfg	Pointer to structure that holds valid scan configuration

Returns

WM_SUCCESS if the call is successful non-zero error-code in case of error

Note

Call to this function generates application callback with event PROV_SCAN_CFG_SET_CB.

8.45.3.7 int prov_set_scan_config_no_cb (struct prov_scan_config * *scan_cfg*)

Sets scan configuration without application callback

Parameters

scan_cfg	Pointer to structure that holds valid scan configuration

Returns

WM_SUCCESS if the call is successful non-zero error-code in case of error

Note

Call to this function does not generate application callback with event PROV_SCAN_CFG_SET_CB. If application itself wants to apply scan configuration settings (e.g. at initialization), this call should be used.

8.45.3.8 int verify_scan_interval_value (int *scan_interval*)

Verify scan interval value

Parameters

scan_interval	Scan interval in seconds

Returns

WM_SUCCESS if interval is valid -WM_FAIL if interval is invalid

8.45.3.9 int register_provisioning_web_handlers (void)

Register provisioning HTTP API handlers

Returns

WM_SUCCESS if handlers registered successfully -WM_FAIL if handlers registration fails

8.45.3.10 void unregister_provisioning_web_handlers (void)

Unregister provisioning HTTP API handlers

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.45.3.11 int register_secure_provisioning_web_handlers (uint8_t * *key*, int *len*, nw_save_cb_t *nw_save_cb*)

Register secure provisioning HTTP API handlers

Parameters

key	Pointer to provisioning key string for encryption. Pass key as NULL to start ezconnect provisioning without encryption.
len	Length of the provisioning key.
cb	Pointer to the function that will be called when network is set.

Returns

WM_SUCCESS if handlers registered successfully -WM_FAIL if handlers registration fails

8.45.3.12 void unregister_secure_provisioning_web_handlers (void)

Unregister secure provisioning HTTP API handlers

8.45.3.13 int prov_register_secure_provisioning_pin (uint8_t * *key*, int *len*)

Set secure provisioning device pin

The device pin is used in multi model provisioning to decrypt network information received by sniffing multicast packets and in uAP based provisioning with secured web handlers.

Parameters

key	Pointer to provisioning key string for encryption. Pass key as NULL to start ezconnect provisioning without encryption.
len	length of provisioning key.

Returns

WM_SUCCESS if key registered successfully -WM_FAIL if key registration fails

8.45.3.14 void prov_unregister_secure_provisioning_pin ( )

Unset secure provisioning device pin

8.45.3.15 void register_ezconnect_provisioning_ssid (char * *ssid*)

Set ezconnect provisioning micro-AP ssid

In multi model provisioning, device enters in SMC mode in which it sniffs packets and also sends beacons and probe responses. SSID information is required in populating beacon information. Hence, this API should be called before configuring SMC mode.

Parameters

ssid	The micro-AP SSID

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.45.3.16 int sys_wq_enqueue_job_wlan_scan ( )

Enqueue wlan_scan job in system work queue

In multi model provisioning, when device receives micro AP auth request, it starts micro AP. To make scan results available for /prov/scan secure web handler, wlan_scan is enqueued in system work queue which updates scan list periodically.

Returns

WM_SUCCESS on success
-WM_FAIL otherwise

8.45.3.17 int sys_wq_dequeue_job_wlan_scan ( )

Dequeue wlan_scan job from system work queue

Returns

WM_SUCCESS on success
-WM_FAIL otherwise

8.45.3.18 int set_network_cb_register (int(∗)(struct wlan_network ∗) *cb_fn*)

This function lets user use /sys/network handler in "provisioned" mode by letting application register a callback handler that is called with received wlan_network structure that gives control directly to the application bypassing provisioning module.

Parameters

cb_fn	Function pointer that should be called in case of POST on /sys/network

Returns

WM_SUCCESS if handler registered successfully -WM_FAIL if handler registration fails

8.45.3.19 int prov_set_scan_policy (enum prov_scan_policy *scan_policy*)

Set Provisioning Scan Policy

The micro-AP provisioning process performs a periodic scan. The scan results are available through the /sys/scan HTTP API. This API can be used to modify the provisioning scan policy.

The default policy is to scan periodically. The periodicity can be configured using the prov_set_scan_config() function or through the /sys/scan-config HTTP API.

Parameters

scan_policy	can be either of PROV_PERIODIC_SCAN or PROV_ON_DEMAND_SCAN

Returns

-WM_FAIL in case of the scan policy cannot be set.
WM_SUCCESS if scan policy is set successfully.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.45.3.20 int prov_get_lock_scan_results ( )

Lock the scanned entries

Get the lock on scanned entries structure. This is required since the provisioning module periodically updates the scanned entries. This update should not take place in between access to those scanned network entries

Returns

-WM_FAIL if error in locking the mutex WM_SUCCESS if success in locking the mutex

8.45.3.21 void prov_put_lock_scan_results ( )

Revoke the lock on scanned entries

Revoke the lock from scanned entries structure

8.45.3.22 int prov_get_scan_count ( )

Get the number of the network scanned entries

This API returns the count of scanned network entries by the firmware

Returns

number of valid scanned entries

8.45.3.23 int prov_get_scan_result (struct wlan_scan_result ** *sptr*, unsigned int *index*)

Get the scanned entry present

This API is used to retrieve the network scanned entry from the specified index

Parameters

out	*sptr*	address of the pointer of type wlan_scan_result pointing to the scan result at given index
in	*index*	specifies the index of the result to be copied

Returns

WM_SUCCESS if valid index -WM_FAIL if index is invalid

8.45.3.24 int prov_set_network (int *prov_method*, struct wlan_network * *net*)

Sets the current station interface network

This API is used to command the station interface to connect to the specified network

Parameters

in	*prov_method*	An integer to specify the type of WLAN provisioning mode. This can be one of the following PROVISIONING_WLANNW, PROVISIONING_WPS or PROVISIONING_EZCONNECT
in	*net*	pointer to the wlan_network containing the network credentials to be associated with

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS on success
error otherwise

8.45.4 Macro Documentation

8.45.4.1 #define PROVISIONING_WLANNW (unsigned int)(1<<0)

WLAN network (uAP) based provisioning mode

8.45.4.2 #define PROVISIONING_WPS (unsigned int)(1<<1)

WPS provisioning mode

8.45.4.3 #define PROVISIONING_EZCONNECT (unsigned int)(1<<2)

Marvell EZConnect Provisioning mode

8.45.5 Enumeration Type Documentation

8.45.5.1 enum prov_scan_policy

Provisioning Scan Policies

The micro-AP provisioning process performs a periodic scan. The scan results are available through the /sys/scan HTTP API. The following scan policies can be used for this scanning.

Enumerator

PROV_PERIODIC_SCAN Periodic Scan. This is the default scan policy. The periodicity can be configured using the prov_set_scan_config() function or through the /sys/scan-config HTTP API.

PROV_ON_DEMAND_SCAN On-Demand Scan. In this policy a network scan is performed on provisioning start-up. Further calls are only triggered on subsequent requests to the /sys/scan HTTP API.

8.45.5.2 enum prov_event

This enum enlists the events that are delivered to the application by provisioning module through the provisioning_-event_handler callback registered in struct provisioning_config. Note that all the events may not be delivered in the context of provisioning thread.

Enumerator

PROV_START This event is delivered immediately after when provisioning thread begins its operation i.e. when prov_start is called

PROV_START_SMART_MODE This event is delivered immediately after smart mode is started i.e. when wlan_start_smart_mode is called

PROV_STOP_SMART_MODE This event is delivered immediately after smart mode is stopped i.e. when wlan_stop_smart_mode is called

PROV_RESET_TO_SMC This event is delivered when device switches from micro AP mode to smart mode

PROV_NETWORK_SET_CB This event is received along with struct wlan_network data. This event is delivered to application in case of successful POST to /sys/network or in case of successful WPS session attempt. Application can use received wlan_network structure to connect as well as to store in PSM.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

PROV_MICRO_AP_UP_REQUESTED This event is received when device receives micro-AP connection request in smc mode. Generated only when EZConnect provisioning is enabled

PROV_ABORTED This event is currently unused

PROV_SCAN_CFG_SET_CB This event is sent to the application on successful POST on /sys/scan-cfg. This is sent along with struct prov_scan_config data. Provisioning module already modifies the current scan configuration as per provided values. Application can choose to write this data into PSM for future use.

PROV_WPS_SSID_SELECT_REQ This event is generated by provisioning library when

- PIN is registered with provisioning thread and found scan results contain atleast one SSID with on-going WPS session with PIN mode active
- Pushbutton is registered with provisioning thread and found scan results contain atleast one SSID with on-going WPS session with Pushbutton mode active.

Along with this event an array of struct wlan_scan_results is sent. The number of array elements can be found by dividing "len" by sizeof struct wlan_scan_results. The application is expected to return index of chosen SSID with which WPS session should be attempted after going through the provided scan list. The application should return -1, if it doesn't intend to attempt session with any SSID found in the list.

PROV_WPS_SESSION_STARTED This event is generated when WPS session attempt is started by the provisioning thread. Application can use this to provide visual indicator to end-user.

PROV_WPS_PIN_TIMEOUT This event is generated when the PIN times out. The timeout interval is 2 minutes. Application may choose to re-send the PIN on timeout.

PROV_WPS_PBC_TIMEOUT This event is generated when the Pushbutton event times out. The timeout interval is 2 minutes.

PROV_WPS_SESSION_SUCCESSFUL This event is generated when WPS session succeeds. This event is preceded by PROV_SET_NETWORK_CB event that delivers the home network security parameters to the application.

PROV_WPS_SESSION_UNSUCCESSFUL This event is generated when the WPS session can not succeed due to some condition like PIN values at both ends did not match.

8.45.5.3 enum provisioning_state

This enum enlists the states in which provisioning thread can be. Application may want to do some validation depending on the state before sending any request to provisioning thread.

Enumerator

PROVISIONING_STATE_INITIALIZING This state indicates the provisioning thread is still initializing provisioning network and/or WPS thread. Application should not pass events like PIN or Pushbutton registration in this state as they will be ignored.

PROVISIONING_STATE_ACTIVE This state indicates that the device is in provisioning state ready to get network settings applied.

PROVISIONING_STATE_SET_CONFIG This state indicates either

- POST on /sys/network has arrived or
- WPS session attempt is in progress

This state protects from simultaneous attempts to configure the device

PROVISIONING_STATE_DONE This state indicates that provisioning is successful

PROVISIONING_STATE_STOP_REQUESTED This state indicates that provisioning state machine is about to be stopped

PROVISIONING_STATE_STOPPED This state indicates that provisioning state machine is stopped

8.46 psm-v2.h File Reference

Persistent storage module version 2.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.46.1 Detailed Description

PSM (Persistent Storage Module) is a module which provides a PSM object abstraction layer over low level flash sector read/write operations. This is similar to how a filesystem provides a file abstraction over a block device driver.

The PSM is specifically developed to work on flash memory devices which can be read/written on byte level but erases are possible only on block level. PSM helps reduce the flash usage thus helping in wear leveling. The working is described in brief as follows. A PSM object is an atomic structure on the flash and contains metadata and data (value). Metadata contains object name, length, etc. When an object value is changed a new object with same name is appended to the flash block instead of:

- Backing up other objects
- Erasing flash
- Restoring objects
- Writing new object.

The earlier object with obsolete value will be inactivated and it is practically deleted from the PSM. It will be physically deleted during the compaction process.

8.46.2 Compaction process

As new objects are appended to current objects a time will come when the flash space is close to getting full. At this point all the objects (excluding the inactivated ones) are copied to a special variable sized reserved area of the same flash partition. The data area i.e. the area of the partition which contained the active and inactive objects uptill now, is erased and the objects are copied back to data area. In effect, all obsolete objects are now physically deleted.

During the process of compaction, or during new object write a power failure could occur. PSM is designed to handle this situation and has error recovery algorithms. It will ensure that no existing objects are corrupted and corrupted objects are safely removed during psm module initialization.

The PSM module is designed to be able to read/write to arbitrary sized objects spanning multiple flash blocks. The only limitation is flash size and space reserved for compaction process.

8.46.3 Terminology

Block

This is the storage unit for the PSM. The size of a block has to be in multiples of sector size of the NOR flash chip used.

8.46.4 Memory usage of PSM module

Flash space usage

PSM has 12 bytes of overhead per PSM object. Thus, every PSM object size is: Data size (user defined) + name length (user defined) + 12 bytes (PSM defined).

Heap usage

PSM init (psm_module_init()) allocates about 34 bytes for (internal) PSM singleton object. This object will be present on heap till psm_module_deinit() is called.

psm_object_open() allocates about 46 bytes internally and remains allocated till psm_object_close() is called. If instead psm_set_variable() or psm_get_variable() is used no heap is used.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.46.5 Thread safety

PSM module is designed to be completely thread safe. Even across multiple instances of PSM module (i.e. multiple psm_module_init() calls) on separate sections/partitions of flash. Only caveat is that low level flash driver also needs take care of thread safety if multiple PSM partitions are on the *same* flash chip.

8.46.6 Read only usage

In certain use cases it may be desired to use PSM in a read only mode. One of the benefits of this is that reserved space is not required and PSM can fit in lesser number of sectors. When the read only flag is passed to the psm_module_init(), PSM will run in read only mode for that partition and not allow any writes.

8.46.7 Usage

PSM provides two types of API's for separate use cases. First type (Simple case) is when the caller has entire value in a buffer for write or has a buffer size equal to entire data to be read. Second type (Advanced case) is when caller needs to read/write huge files in multiple KB's or MB's. In this case the caller needs an API which helps him read/write across multiple calls. Both the cases: are illustrated below.

PSM initialization

This needs to be done for both the simple and advanced case. It is generally done only once after boot.

Statically/dynamically allocate and populate flash_desc_t object. Note that this object needs to persist until psm_-module_deinit() returns. PSM does *not* make an internal copy of this object to avoid wasting space.

```
psm_hnd_t phandle;
int rv = psm_module_init(&ppart, &phandle, NULL);
if (rv != WM_SUCCESS) {
  wmprintf("PSM init failed");
  return rv;
}
```

This handle 'phandle' can now be used for other API which require it. The third parameter is passed as NULL here. If any special flags need to be passed the psm_cfg_t param can be passed accordingly.

PSM simple usage case

Write and read back a small variable.

```
char var[] = "name";
char value[] = "Tim";
int rv = psm_set_variable(phandle, var, value, strlen(value));
if (rv != WM_SUCCESS) {
     wmprintf("Variable write failed");
     return rv;
}

char read_value[10];
rv = psm_get_variable(phandle, var, read_value, strlen(value));
if (rv != WM_SUCCESS) {
     wmprintf("Variable read failed");
     return rv;
}
```

PSM advanced use case

Write a huge valued object

```
char var[] = "hello.mp3";
char value[1024]; //Assume data is already in this buffer
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

```
psm_object_handle_t ohandle;
int rv = psm_object_open(phandle, var, PSM_MODE_WRITE, sizeof(value),
NULL, &ohandle);
if (rv != WM_SUCCESS) {
     //Failed
     return rv;
}
//Write the object in two chunks just to show valid usage pattern
rv = psm_object_write(ohandle, value, 512);
if (rv != WM_SUCCESS) {
     //Failed
     return rv;
}

rv = psm_object_write(ohandle, &value[512], 512);
if (rv != WM_SUCCESS) {
     //Failed *
     return rv;
}

rv = psm_object_close(&ohandle);
if (rv != WM_SUCCESS) {
     //Failed
     return rv;
}
```

De initialize PSM

This would generally be done once only during system shutdown.

```
int rv = psm_module_deinit(&phandle);
if (rv != WM_SUCCESS) {
     //Failed
     return rv;
}
```

8.46.8 Indexing and caching

Indexing and Caching are additional features to be used when fast access to a PSM object is required. Both are to be enabled separately per object. When indexing is enabled the address of the PSM object is saved in RAM. This avoids a search for the object in the flash. When caching is enabled the object name and value are cached in RAM. This gives instant access to the values and NO flash is accessed.

Predictably caching consumes more RAM and amount of RAM consumed depends on the size of the name and value of object. Indexing uses 14 bytes per object.

8.46.9 PSM Encryption (Secure PSM)

User may choose to store the data in PSM in encrypted format. If encryption is enabled, data is encrypted before storing it into PSM and decrypted when data is retrieved. PSMv2 currently uses AES-CTR for encryption. The keys are retrieved from the secure keystore. If encryption is enabled and secure keystore does not have the keys psm initialization will fail. Each PSM object is encrypted separately before storing to flash. The metadata is not encrypted. The name field and data field is encrypted.

Note that, even if Secure PSM is enabled, users can still use PSM in non-encrypted mode. This is useful when there are more than one PSM partitions and only some of them are encrypted.

8.46.10 Function Documentation

8.46.10.1 int psm_module_init (flash_desc_t * *fdesc*, psm_hnd_t * *phandle*, psm_cfg_t * *psm_cfg*)

Initialize the PSM module

This API initializes the PSM module. The given flash space will be accessed and initialized if necessary. Since this function is the first one called after a power cycle, if a flash operation was in progress recovery actions will be

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

performed.

Note

If secure PSM is enabled KEY_PSM_ENCRYPT_KEY and KEY_PSM_NONCE needs to be present in the keystore.

Parameters

in	*fdesc*	Populated flash descriptor flash_desc_t having partition details. Please refer to partition.h for relevant APIs.
out	*phandle*	Pointer to the handle psm_hnd_t. Populated by PSM.
in	*psm_cfg*	psm_cfg_t PSM configuration structure. If NULL is passed default configuration i.e. read/write mode is used.

Returns

WM_SUCCESS Init success.
-WM_E_INVAL Invalid arguments.
-WM_E_NOMEM Memory allocation failure.
-WM_FAIL If any other error occurs.

8.46.10.2 int psm_object_open (psm_hnd_t *phandle*, const char * *name*, psm_mode_t *mode*, uint32_t *max_data_len*, psm_objattrib_t * *attribs*, psm_object_handle_t * *ohandle*)

Open an existing or a new PSM object

This API allows the caller to open a new or existing PSM object. This call is essential before called psm_object_read() or psm_object_write().

Note

If write mode is specified and if an object with the same name is already present, the existing object will be deleted and replaced with the new one.

Parameters

in	*phandle*	Handle returned in earlier call to psm_module_init()
in	*name*	NULL terminated ASCII name of PSM object.
in	*mode*	Specify read mode (PSM_MODE_READ) or a write mode(PSM_MODE_WRITE).
in	*max_data_len*	Valid only for write mode. This specifies the maximum length of the data the caller expect to write across single or multiple calls of psm_object_write(). It is ok if the caller later actually writes less number of bytes to be object than max_data_len given here. For e.g. Giving 'max_data_len' 20 but writing only 4 bytes is perfectly alright. The remaining 16 bytes will be released to the free space pool once this object is closed.
in	*attribs*	Additional attributes as per psm_objattrib_t. Pass NULL if not required.
out	*ohandle*	Pointer to Object handle. Will be populated by psm_object_open() if open operation was successful. This needs to be passed to other psm read/write API's.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS Init success.
-WM_E_INVAL Invalid arguments.
-WM_E_NOMEM Memory allocation failure.
-WM_E_NOSPC If enough space is unavailable for the new object. This error can only be returned in write mode.
-WM_FAIL If any other error occurs.

8.46.10.3 int psm_object_read (psm_object_handle_t *ohandle*, void * *buf*, uint32_t *max_len*)

Read data of an existing and opened object

This API allows the caller to read data from an object. This call can be invoked as many times as the caller wants till the data read finishes. Offset from where data is read is automatically incremented depending on the size read. When data read is over, zero is returned.

Precondition

psm_object_open()

Note

To avoid double pass over flash data area, CRC calculations are done on-the-go as data is being read progressively by the caller. CRC verification is done only in the last call to psm_object_read(). The downside to this is that whole object needs to be read by caller before object integrity can be confirmed.
This API will block if other writers are currently using the PSM in write mode.

Parameters

in	*ohandle*	Handle returned from call to psm_object_open().
out	*buf*	Destination buffer of data read from flash. Ensure that it is atleast equal to max_len
in	*max_len*	Length of the buffer passed as second parameter. Data read can be less than this value is end of object is reached.

Returns

Non-zero positive value equal to number of bytes written to user given buffer
0 No more data available.
-WM_E_INVAL Invalid arguments.
-WM_E_CRC If CRC check failed.
-WM_FAIL If any other error occurs.

8.46.10.4 int psm_object_write (psm_object_handle_t *ohandle*, const void * *buf*, uint32_t *len*)

Write data to an opened PSM object.

This API allows the caller to write data to an object. This call can be invoked as many times as the caller wants as long as total size written is less that 'max_len' specified during psm_object_open(). Data passed in every invocation is appended to the existing data already written.

Precondition

psm_object_open()

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Note

This API will block if other readers/writers are currently using the PSM.

Parameters

in	*ohandle*	Handle returned from call to psm_object_open().
in	*buf*	Pointer to buffer containing data to be written to flash.
in	*len*	Length of the data in the buffer passed.

Returns

WM_SUCCESS Data was written successfully.
-WM_E_INVAL Invalid arguments.
-WM_E_PERM If object was opened in read mode.
-WM_E_NOSPC If caller tries to write more data than specified in call to psm_object_open().
-WM_FAIL If any other error occurs.

8.46.10.5 int psm_object_close (psm_object_handle_t * *ohandle*)

Close the PSM object

This API will close the earlier opened PSM object. The object is finalized in the flash only after this call.

Precondition

psm_object_open()

Parameters

in	*ohandle*	Handle returned from earlier call to psm_object_open()

Returns

WM_SUCCESS Object was closed successfully.
-WM_E_INVAL Invalid arguments.
-WM_FAIL If any other error occurs.

8.46.10.6 int psm_object_delete (psm_hnd_t *phandle*, const char * *name*)

Delete a PSM object

This API deletes an object from the PSM.

Parameters

in	*phandle*	Handle returned from earlier call to psm_module_init()
in	*name*	NULL terminated ASCII name of the PSM object to be deleted.

Note

This API will block if other readers/writers are currently using the PSM.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS Object was deleted successfully.
-WM_E_INVAL Invalid arguments.
-WM_E_NOSPC If caller tries to write more data than specified in call to psm_object_open().
-WM_FAIL If any other error occurs.

8.46.10.7 int psm_format (psm_hnd_t *phandle*)

Format the PSM partition

This API allows the caller to erase and initialize the PSM partition. All the existing objects will be erased.

Note

This API will block if other readers/writers are currently using the PSM.

Parameters

in	*phandle*	Handle to PSM returned earlier from the call to psm_module_init()

Returns

WM_SUCCESS Format operation was successful.
-WM_E_INVAL Invalid arguments.
-WM_FAIL If any other error occurs.

8.46.10.8 void psm_objects_dump (psm_hnd_t *phandle*)

Print all the objects and their data to console.

Parameters

in	*phandle*	Handle to PSM returned earlier from the call to psm_module_init()

8.46.10.9 int psm_set_variable (psm_hnd_t *phandle*, const char * *variable*, const void * *value*, uint32_t *len*)

Set a variable value pair

This API is built on top of existing psm_object_open(), psm_object_write() and psm_object_close() APIs. psm_set_variable() is preferable when small sized objects are being written.

Note

psm_object_open() and psm_object_close() are not required to be called if this API is used.

Parameters

in	*phandle*	Handle to PSM returned earlier from the call to psm_module_init()
in	*variable*	Name of the object.
in	*value*	Pointer to buffer having the data to be written
in	*len*	Length of the data to be written.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS Variable-value pair was set successfully.
-WM_E_INVAL Invalid arguments.
-WM_FAIL If any other error occurs.

8.46.10.10 static int psm_set_variable_str (psm_hnd_t *phandle*, const char * *variable*, const void * *value*) [inline], [static]

Helper function to simplify set of NULL terminated value.

psm_set_variable_str() will self calculate the length of the NULL terminated string sent by caller.

Parameters

in	*phandle*	See documentation of psm_set_variable()
in	*variable*	See documentation of psm_set_variable()
in	*value*	See documentation of psm_set_variable()

Returns

See documentation of psm_set_variable()

8.46.10.11 static int psm_set_variable_int (psm_hnd_t *phandle*, const char * *variable*, int *value*) [inline], [static]

Helper function to set integer value to PSM

This function will convert given int to string representation and store it to PSM. psm_get_variable_int() can be used to read the value back again in int format.

for e.g. 1234 given by the user will be stored as "1234"

Parameters

in	*phandle*	See documentation of psm_get_variable()
in	*variable*	See documentation of psm_get_variable()
out	*value*	The value to be stored to PSM

Returns

See documentation of psm_set_variable()

8.46.10.12 int psm_get_variable (psm_hnd_t *phandle*, const char * *variable*, void * *value*, uint32_t *max_len*)

Get a variable value pair

This API is built on top of existing psm_object_open(), psm_object_read() and psm_object_close() APIs. psm_get_variable() is preferable when small sized objects are being read.

Note

psm_object_open() and psm_object_close() are not required to be called if this API is used.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*phandle*	Handle to PSM returned earlier from the call to psm_module_init()
in	*variable*	Name of the object.
out	*value*	Pointer to buffer where value will be written by PSM.
in	*max_len*	Length of the buffer passed as third parameter. If the buffer is not enough to fit the whole object error will be returned. Partial read will not be done.

Note

Note that this function returns the length of the value. It will **not**< NULL terminate the returned data.

Returns

If operation is a success then length of the value field will be returned.
-WM_E_INVAL Invalid arguments.
-WM_E_NOSPC If variable value exceeds buffer length given.
-WM_FAIL If any other error occurs.

8.46.10.13 static int psm_get_variable_str (psm_hnd_t *phandle*, const char * *variable*, void * *value*, uint32_t *max_len*) [inline], [static]

Helper function to simplify get of NULL terminated value.

This function will automatically NULL terminate the byte string retrieved from PSM. One additional byte should be kept reserved by the caller for this NULL terminator.

Parameters

in	*phandle*	See documentation of psm_get_variable()
in	*variable*	See documentation of psm_get_variable()
out	*value*	See documentation of psm_set_gariable()
in	*max_len*	See documentation of psm_get_variable()

Returns

See documentation of psm_get_variable()

8.46.10.14 static int psm_get_variable_int (psm_hnd_t *phandle*, const char * *variable*, int * *value*) [inline], [static]

Helper function to get integer value from PSM

This function will automatically read string representation of a number from PSM and convert it to int format before returning.

for e.g. "1234" stored in PSM will be returned as 1234 in the user given int variable.

Parameters

in	*phandle*	See documentation of psm_get_variable()
in	*variable*	See documentation of psm_get_variable()
out	*value*	The value returned.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS if retrieval was successful
-WM_E_INVAL Invalid arguments.
-WM_FAIL If any other error occurs.

8.46.10.15 int psm_module_deinit (psm_hnd_t * *phandle*)

De-initialize the PSM module

This API unloads the PSM module from handling the flash partition.

Note

This API will block till other readers/writers are currently using the PSM.

Parameters

in	*phandle*	Handle to the PSM obtained earlier through psm_module_init()

Returns

WM_SUCCESS Module de-init success.
-WM_E_INVAL Invalid arguments.
-WM_FAIL If any other error occurs.

8.46.11 Typedef Documentation

8.46.11.1 typedef void* psm_hnd_t

PSM Handle

8.46.11.2 typedef void* psm_object_handle_t

PSM Object handle

8.46.12 Enumeration Type Documentation

8.46.12.1 enum psm_mode_t

PSM opening mode

Enumerator

PSM_MODE_READ Read mode
PSM_MODE_WRITE Write mode

8.46.12.2 enum psm_flags_t

PSM Flags

Required in psm_objattrib_t. Please refer Indexing and caching for details

Enumerator

PSM_INDEXING_ENABLED Indexing enabled
PSM_CACHING_ENABLED Caching enabled

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.47 psm.h File Reference

Persistent storage module version 1 (Deprecated)

8.47.1 Detailed Description

Please use version 2 (psm-v2.h)

8.47.2 Macro Documentation

8.47.2.1 #define PSM_CREAT 1

Flag to ask psm to create the partition if it doesn't exist. Used with psm_register_module()

8.48 push_button.h File Reference

Push button helper functions.

8.48.1 Function Documentation

8.48.1.1 int push_button_set_cb (input_gpio_cfg_t *input*, gpio_irq_cb *pb_cb*, int *time*, int *repeat_time*, void * *data*)

Set a callback function for push button operations

This API can be used to register different callbacks for the same push button based on the time for which the button is pressed. Consider following use cases to understand the API.

1. Single short press like a simple on/off toggle switch.

```
void button1_cb(int pin, void *data)
{
    <short press action>
}
{
    input_gpio_cfg_t button_1;
    button_1.gpio = GPIO_25;
    button_1.type = GPIO_ACTIVE_LOW;
    push_button_set_cb(button_1, button1_cb, 0, 0, NULL);
}
```

   Note that the time specified is 0. This is the most common use case.

2. Press and hold for a specified time and release

```
void button1_5sec_cb(int pin, void *data)
{
    <5 sec press and release action>
}
{
    push_button_set_cb(button_1, button1_5sec_cb, 5000, 0, NULL);
}
```

   The callback will be invoked if the button is pressed for more than 5 seconds and then released.

3. Press and hold for a specified time and then invoke a callback repeatedly.

```
void button1_10sec_repeat_cb(int pin, void *data)
{
    <repeat action to invoke>
}
{
    push_button_set_cb(button_1, button1_10sec_repeat_cb, 10000,
                200, NULL);
}
```


MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

With this, if a button is pressed and held for more than 10 seconds, after the 10 second timeout, the callback is invoked for every 200 msec until the button is released. Note that if the repeat time is going to be used, the "time" specified for that callback should be maximum for that particular button. So, if say another callback is registered with a time of 11 seconds, the repeat callback defined above will not take effect.

A single push button can have multiple such callbacks registered, with a maximum limit of MAX_PB_CB. Consider another example.

```
{
    push_button_set_cb(button_1, button1_cb, 0, 0, NULL);
    push_button_set_cb(button_1, button1_5sec_cb, 5000, 0, NULL);
    push_button_set_cb(button_1, button1_10sec_cb, 10000, 0, NULL);
}
```

- In this case, for a short press, button1_cb will be invoked. button1_cb will be invoked even if the button is pressed for a longer duration but released before 5 seconds.
- If the button is pressed for more than 5 seconds but released before 10 seconds, button1_5sec_cb will be invoked.
- Since 10 seconds is the maximum time configured for this button, if the button is pressed for 10+ seconds, the button1_10sec_cb will be invoked even if the button is not released.

If some pointer is specified as the last argument for push_button_set_cb, it will be received in the pb_cb.

This API takes care of key debouncing (oscillations because of the hardware button press) as well. Hence, the application will not get any false triggers.

If multiple callbacks are registered for the same timeout value, the call registered the latest will be invoked.

Parameters

in	*input*	The push button configuration as per input_gpio_cfg_t
in	*pb_cb*	The push button callback to be invoked for this event. Refer gpio_irq_cb
in	*time*	The time in milliseconds for which the button should be pressed to invoke the callback
in	*repeat_time*	The time in milliseconds for invoking the callbacks repeatedly as described above.
in	*data*	A pointer to the data that the applications wants to receive in the callback.

Returns

WM_SUCCESS on success
-WM_FAIL in case of an error

8.48.1.2 int push_button_reset_cb (input_gpio_cfg_t *input*, int *time*)

Reset callback function registered for a push button

This function resets the callbacks registered using push_button_set_cb(), one at a time.

Parameters

in	*input*	The push button configuration as per input_gpio_cfg_t. Should be the same as passed to push_button_set_cb()
in	*time*	The time as specified while registering the callback using push_button_set_cb()

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS on success
-WM_FAIL on failure. This will occur if the button is already pressed when this function is invoked. This is done to avoid unexpected behavior.

8.48.2 Macro Documentation

8.48.2.1 #define MAX_PB_CB 4

Maximum callbacks that can be registered for a given push button

8.48.2.2 #define PB_DEF_DEBOUNCE_TIME 100

Default key debounce time in milliseconds

8.49 pwrmgr.h File Reference

Power Manager Framework.

8.49.1 Detailed Description

The SDK provides a power manager framework to enable applications to enable various power saving mechanisms. This framework depends on the processor-specific APIs as defined in MCU Power Management .

The power manager provides a functionality to register callbacks on entry or exit from power modes power_state_t. Applications or drivers can register a power save callback using the function pm_register_cb() or deregister using pm_deregister_cb().

8.49.2 Usage

Power management can be used by applications as follows:

1. Call pm_init(). This initializes the power management module. The most basic power-management scheme is enabled at this time, which puts the MCU in PM1 mode on idle.
2. Register callback for low power entry and exit using pm_register_cb. This will ensure that the callback gets called everytime before entering the power save mode or exiting from the power save mode. Applications can perform any state save-restore operations in this callback.
3. Set power save configuration pm_mcu_cfg(). This call will ensure that the system continues to enter the configured power save mode as and when it detects inactivity. The power manager framework makes sure that the system comes up just in time to service any software timers, or threads waiting on timed sleep.
4. If an application/driver expects to receive events from an external source (apart from RTC or wakeup0/1 lines), then it may have to ensure that the system does not enter power save modes in this duration. This control can be exerted on the power manager framework by using wakelocks. Acquiring a wakelock using wakelock_get() prevents the power manager framework from putting the system into low power modes. The wakelock can be released using wakelock_put().

8.49.3 Function Documentation

8.49.3.1 int pm_register_cb (unsigned char *action*, power_save_state_change_cb_fn *cb_fn*, void * *data*)

Register Power Save Callback

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

This function registers a callback handler with the power manager. Based on the 'action' specified, this callback will be invoked on the corresponding power-save enter or exit events of the processor.

Note

The callbacks are called with interrupts disabled.

Parameters

in	*action*	Logical OR of set of power-save events power_save_event_t for which callback should be invoked.
in	*cb_fn*	pointer to the callback function
in	*data*	Pointer which can be a pointer to any object or can be NULL.

Returns

handle to be used in other APIs on success,
-WM_E_INVAL if incorrect parameter
-WM_E_NOMEM if no space left.

8.49.3.2 int pm_deregister_cb (int *handle*)

De-register Power Save Callback

This function de-registers the power save callback handlers from the power manager.

Parameters

in	*handle*	handle returned in pm_register_cb()

Returns

WM_SUCCESS on success
-WM_FAIL otherwise.

8.49.3.3 int pm_change_event (int *handle*, unsigned char *action*)

Modify Power Save Callback

Change the power-save events for which the callback will be invoked. This function allows the application to change the events without changing callback function.

Parameters

in	*handle*	handle returned in pm_register_cb()
in	*action*	logical OR of new set of power-save events power_save_event_t

Returns

WM_SUCCESS on success
-WM_FAIL otherwise.

8.49.3.4 int pm_init ( )

Initialize the Power Manager module.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

This functions initializes the Power Manager module and registers an idle hook with OS which will allow processor to toggle between Active (PM0) and Idle (PM1) mode. Please see power_state_t for details of power modes.

Returns

WM_SUCCESS on success
-WM_FAIL otherwise.

8.49.3.5 bool is_pm_init_done ( )

Check init status of the Power Manager module This functions check the init status of Power Manager module.

Returns

true if Power Manager module init is done.
false if not.

8.49.3.6 void pm_reboot_soc ( )

Do a reboot of the chip

This function forces a reboot of the chip, as if it were power-cycled.

Note

This function will not return as it will reboot the hardware.

8.49.3.7 int wakelock_get (const char * *id_str*)

Acquire WakeLock

When the power manager framework is enabled in an application using a call to pm_mc200_cfg(), the system opportunistically enters power save modes on idle-time detection. Wakelocks are used to prevent the system from entering such power save modes for certain sections of the code. The system will not automatically enter the MCU power save states if wakelock is being held.

Typically wakelocks have to be held only if any process/thread is waiting for an external event or completion of asynchronous data transmission. Systems continue to work properly for all other cases without acquiring the wakelocks.

Parameters

in	*id_str*	This is a logical wakelock ID string used for debugging. Logically correlated get/put should use the same string. This string is used internally to track the individual logical wakelock counts.

Note

This function should NOT be called in an interrupt context.

Returns

WM_SUCCESS on success
-WM_FAIL otherwise.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.49.3.8 int wakelock_put (const char * *id_str*)

Release WakeLock

Use this function to release a wakelock previously acquired using wakelock_get().The system enters MCU power save states, only when all the acquired wakelocks have been released.

Note

The number of wakelock_get() and wakelock_put() requests must exactly match to ensure that the system can enter power save states.

Parameters

in	*id_str*	Logical ID string used for wakelock debugging. Please read the documentation for wakelock_get.

Note

This function should NOT be called in an interrupt context.

Returns

WM_SUCCESS on success
-WM_FAIL otherwise.

8.49.3.9 int wakelock_isheld ( )

Check if WakeLock is held

Returns

0 if no one is holding a wake lock.
Non-zero otherwise.

8.49.3.10 int pm_cli_init (void)

Register Power-Mode Command Line Interface commands

Returns

0 on success
1 on failure

Note

This function is available to the application and can be directly called if the pm module is initialised.

8.49.3.11 int pm_mcu_cli_init ( )

Register MCU Power Manager CLI

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS on success
-WM_FAIL on failure

Note

This function can be called after the power manager is initialized

8.49.4 Typedef Documentation

8.49.4.1 typedef void(∗ power_save_state_change_cb_fn)(power_save_event_t event, void ∗data)

Function pointer for callback function called on low power entry and exit

8.49.5 Enumeration Type Documentation

8.49.5.1 enum power_save_event_t

Power save event enumerations

Enumerator

ACTION_ENTER_PM2 On entering PM2 mode

ACTION_EXIT_PM2 On exiting PM2 mode

ACTION_ENTER_PM3 On entering PM3 mode

ACTION_EXIT_PM3 On exiting PM3 mode

ACTION_ENTER_PM4 On entering PM4 mode

8.50 rfget.h File Reference

Remote Firmware Upgrade.

8.50.1 Detailed Description

This module provides APIs for getting an upgrade image over the network and updating on-flash firmware image. WMSDK supports redundant image based firmware upgrades whereby there are two partitions in the flash to hold the firmware: active and passive. Firmware upgrade process always writes new image on passive partition and finally modifies partition table to mark it as the new active partition. This then results into boot from new firmware image when device reboots.

This module provides APIs at different levels.

The simplest APIs to use are the High level RFGET APIs. Applications can directly download and upgrade the firmware over a secure TLS connection by using rfget_client_mode_update() and passing appropriate httpc_cfg_t structure. HTTP server mode upgrade can be performed using the rfget_server_mode_update() API.

If there is not enough memory for TLS, APIs that use other security mechanisms like ED-CHACHA (fw_upgrade_ed_chacha.h) or RSA-AESCTR (fw_upgrade_rsa_aes.h) are also available. These APIs ensure that the upgrade images are encrypted and signed, thus ensuring comparable security as TLS. If applications want to use some other crypto combination besides ED/CHACHA and AES/RSA, they can add it by using the generic secure upgrade mechanism specified in secure_upgrade.h

If an application does not want to use HTTP, but some other deta fetch mechanism, the Middle level RFGET APIs can be used. These are available for MCU Firmware (update_and_validate_firmware()), Wi-Fi firmware (update_and_validate_wifi_firmware())and File System (update_and_validate_fs())

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

For an even granular control, the Low level RFGET APIs should be used. These will be especially useful if some application specific partition is to be upgraded.

Other RFGET APIs can be found in Other RFGET APIs section.

8.50.2 Enumeration Type Documentation

8.50.2.1 enum wm_rfget_code

RFGET Error Code

Enumerator

WM_E_RFGET_INVPRO Protocol Not Supported

WM_E_RFGET_F2MSL Failed to set 2msl time

WM_E_RFGET_FHTTP_REQ Failed to send/recv http request

WM_E_RFGET_FHTTP_PARSE Failed to parse http header

WM_E_RFGET_INVHTTP_STATUS Invalid http status

WM_E_RFGET_FSOCK Error in rfget_sock_init

WM_E_RFGET_INVLEN rfget length unknown or invalid

WM_E_RFGET_FINIT Failed to init rfget

WM_E_RFGET_FUPDATE Firmware or ftfs Update failed

WM_E_RFGET_FRD Firmware or ftfs read fail

WM_E_RFGET_FWR Firmware or ftfs write fail

WM_E_RFGET_FFW_LOAD Failed to load firmware

WM_E_RFGET_INV_IMG Invalid file format

8.51 secure_upgrade.h File Reference

Secure firmware upgrades.

8.51.1 Detailed Description

This module is used in conjunction with rfget and httpc to provide a secure firmware upgrade mechanism. It is recommended to use the secure upgrade APIs provided in fw_upgrade_ed_chacha.h and fw_upgrade_rsa_aes.h rather than the lower level APIs provided by this module. The APIs given here should be used only if the application wants to use own crypto algorithms and not the ones (ED25519 + Chacha20 and RSA + AES-CTR) provided by the SDK.

Most important aspect of this module is populating the upgrade_struct_t structure with appropriate functions and data structures. Then, depending on the desired mode, the client_mode_secure_upgrade() or server_mode_secure_upgrade() API can be used.

Note

The usage of these APIs assumes that all accessories are programmed with the following information at the time of manufacturing:

1. Firmware Encryption Key unique for each device
2. Firmware Verification Key (Upgrade Server's Public Key)

The current secure firmware upgrade mechanism specification version is 0002, the details of which can be found in the following sections.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.51.2 Secure Firmware upgrade image format

The firmware image to be used for secure upgrades should be formatted as follows:

1. The checksum (using SHA, MD5, or any other suitable hashing algorithm as per the use case) of the firmware image, abc.bin, is calculated.

2. A signature of this checksum is created (with length signature_len) as the firmware signature. The private key that is used for generating the signature should correspond to the Firmware Verification Key programmed in all the accessories. This ensures that the accessory upgrades to firmware images that are delivered from a valid source.

3. The length of the firmware, fw_len is calculated.

4. The [signature+firmware image] is then encrypted with a suitable encryption algorithm using the Firmware Encryption Key. This ensures that only accessories that have the correct Firmware Encryption Key can access the firmware images.

5. A header is created by appending the 4 byte fw_upgrade_spec_version, 4 byte header_len, 4 byte fw_len and the Initialization Vector (IV) of length iv_len used for encryption. header_len is the number of bytes starting from fw_len till the end of the header. The purpose of this is to allow adding more elements in the header, without affecting older versions. Older versions will just skip the extra fields in the header.

6. The data encrypted above is then appended to this header and this entire image is used as the upgrade image. A different IV should be used during each communication so that the encryption will be better secured.

8.51.3 Firmware Upgrade Algorithm

The firmware upgrade algorithm will be as follows:

1. Fetch 4 bytes of fw_upgrade_spec_version and check if it is greater or equal to the upgrade specification version supported by the device. If not, abort the upgrade and return an error.

2. Fetch the 4 byte header_len and 4 byte fw_len.

3. Fetch iv_len bytes from the server and initialize the decryption context.

4. Read any more bytes left to be read from the header. There may be some additional fields added by newer versions.

5. Fetch signature_len bytes from the server, decrypt them and store as the signature for later use.

6. Fetch one block of the firmware image. If this is the first block, ensure that it contains the right headers.

7. Decrypt and write to the passive partition.

8. Maintain a running checksum of each decrypted block.

9. Repeat steps 4 to 6 until end of file is reached.

10. When end of file is reached, get the final value of the checksum and use the Firmware Verification Key to verify that the signature of the firmware that was calculated in step 1 is valid.

11. If the checksum matches, set the passive partition to be active partition, and perform a soft reset of the accessory.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.51.4 Function Documentation

8.51.4.1 int client_mode_secure_upgrade (enum flash_comp *flash_comp_type*, char * *fw_url*, upgrade_struct_t * *priv*, const tls_client_t * *cfg*)

Securely upgrade the firmware in client mode

This function can be used to securely upgrade the MCU Firmware or Wi-Fi firmware in HTTP client mode. This function will establish a session with the remote HTTP server, securely fetch the upgrade image and flash it. If there is no special handling required, the fw_data_read_fn field of upgrade_struct_t can be left NULL, so that the default data fetch function will be used.

Parameters

	flash_comp_type	Flash component to be updated. Valid values are FC_COMP_FW, and FC_COMP_WLAN_FW.
in	*fw_url*	Pointer to a string which holds the url from where the firmware image should be fetched.
in	*priv*	Pointer to a upgrade_struct_t structure to be used by the secure upgrade function. The location pointed by this must be available throughout the lifetime of the upgrade process.
in	*cfg*	Pass certificate data if TLS connection is used. Pass NULL if TLS connection is not required. Pass NULL if TLS is required but certificate verification if not required.

Returns

WM_SUCCESS if the secure upgrade succeeds.
-WM_FAIL if the secure upgrade fails.

8.51.4.2 int server_mode_secure_upgrade (enum flash_comp *flash_comp_type*, upgrade_struct_t * *priv*)

Securely upgrade the firmware in server mode

This function can be used to securely upgrade the MCU Firmware or Wi-Fi firmware in HTTP server mode. However, the data fetch function has to be provided externally by setting the fw_data_read_fn field of upgrade_struct_t to some valid value. session_priv_data can be used to maintain a context of the http session.

Parameters

	flash_comp_type	Flash component to be updated. Valid values are FC_COMP_FW, and FC_COMP_WLAN_FW.
in	*priv*	Pointer to a upgrade_struct_t structure to be used by the secure upgrade function. The location pointed by this must be available throughout the lifetime of the upgrade process.

Returns

WM_SUCCESS if the secure upgrade succeeds.
-WM_FAIL if the secure upgrade fails.

8.51.5 Typedef Documentation

8.51.5.1 typedef void(* decrypt_init_func_t)(void *context, void *key, void *iv)

Prototype for the decrypt initialization function. Please use a wrapper for your function if it does not match this signature.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in,out	*context*	A pointer to the decrypt function's context.
in	*Key*	to be used for Decryption.
in	*iv*	The initialization vector received from the server.

8.51.5.2 typedef int(∗ decrypt_func_t)(void ∗context, void ∗inbuf, int inlen, void ∗outbuf, int outlen)

Prototype for the decrypt function. Please use a wrapper for your function if it does not match this signature.

Parameters

in,out	*context*	A pointer to the decrypt function's context.
in	*Pointer*	to the input buffer which has the ciphertext.
in	*inlen*	Length of the input buffer (number of bytes).
out	*outbuf*	A pointer to the output buffer which will hold the decrypted data. It can be same as the inbuf if the algorithm uses in-place decryption.
in	*outlen*	Length of the output buffer.

Returns

len Length of the decrypted message.
-WM_FAIL on error.

8.51.5.3 typedef int(∗ signature_verify_func_t)(void ∗msg, int msglen, void ∗pk, void ∗signature)

Prototype for the signature verification function. Please use a wrapper for your function if it does not match this signature.

Parameters

in	*msg*	Pointer to a buffer which holds the message whose signature has to be verified.
in	*msglen*	Length of the message
in	*pk*	Pointer to the public key of the server.
in	*signature*	The signature against which the verification is to be done.

Returns

0 if signature verification successful.
any other value if signature verification failed.

8.51.5.4 typedef void(∗ hash_update_func_t)(void ∗context, void ∗buf, int buflen)

Prototype for the hash update function. Please use a wrapper for your function if it does not match this signature.

Parameters

in,out	*context*	A pointer to the hashing function's context.
in	*buf*	Pointer to a buffer whose hash (checksum) is to be calculated.
in	*buflen*	Length of the buffer

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.51.5.5 typedef void(∗ hash_finish_func_t)(void ∗context, void ∗result)

Prototype for the hash finish function. Please use a wrapper for your function if it does not match this signature.

Parameters

in	*context*	A pointer to the hashing function's context.
out	*result*	Pointer to a buffer where the result of the hash (checksum) will be stored.

8.51.5.6 typedef int(∗ fw_data_read_fn_t)(void ∗priv, void ∗buf, uint32_t max_len)

Prototype for the firmware data read function. Please use a wrapper for your function if it does not match this signature.

Parameters

in	*Private*	data associated with the data read session. This is defined by the session_-priv_data element in upgrade_struct_t
out	*buf*	Pointer to a buffer to be used to read the data.
in	*max_len*	The length of the buffer buf

Returns

Number of bytes read
0 if end of data is reached
-WM_FAIL on error

8.52 telnetd.h File Reference

The Telnet Server.

8.52.1 Function Documentation

8.52.1.1 int telnetd_start (int *port*)

Start the telnet daemon

This function starts the telnetd daemon. Remote clients can connect to the telnet server and execute commands on the CLI.

Parameters

port	The port number to listen for connections.

Returns

-WM_E_TELNETD_SERVER_RUNNING if server is already running.
-WM_E_TLENETD_THREAD_CREATE if telnet start failed.
WM_SUCCESS if telnetd start was successful.

8.52.1.2 void telnetd_stop (void)

Stop the telnet daemon

This function stops the telnet daemon.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Note

This function is usually called on a network disconnect/link lost event.

8.52.2 Enumeration Type Documentation

8.52.2.1 enum wm_telnetd_errno

TELNETD Error Codes

Enumerator

WM_E_TELNETD_SERVER_RUNNING Telnet server is already running

WM_E_TLENETD_THREAD_CREATE Failed to create telnet thread

WM_E_TELNETD_SOCKET Failed to create the socket

WM_E_TELNETD_BIND_SOCKET Failed to bind the socket

WM_E_TELNETD_LISTEN_SOCKET Failed to listen on socket

WM_E_TELNETD_ACCEPT_SOCKET Failed to accept new socket

WM_E_TELNETD_NONBLOCKING_SOCKET Failed to make socket non-blocking

8.53 websockets.h File Reference

Websocket Implementation in WMSDK using the WSLAY websocket library.

8.53.1 Detailed Description

This file exposes the API's to be used for websocket communication and which internally call the WSLAY library functions

8.53.2 Function Documentation

8.53.2.1 int ws_upgrade_socket (http_session_t * *handle*, http_req_t * *req*, const char * *protocol*, ws_context_t * *ws_ctx*)

Upgrade a socket to websocket

This function upgrades an already created socket to websocket.

Parameters

in	*handle*	Pointer to the handle retrieved from the call to http_open_session()
in	*req*	Pointer to the HTTP request retrieved from the call to http_prepare_req()
in	*protocol*	Websocket subprotocol to be used.
out	*ws_ctx*	The Web Socket context to be used from here onwards

Returns

WM_SUCCESS is returned on success. -WM_FAIL in case of an error.

8.53.2.2 int ws_frame_send (ws_context_t * *ws_ctx*, ws_frame_t * *f*)

Send a websocket frame

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*ws_ctx*	WS Context returned from a call to ws_upgrade_socket()
in	*f*	The frame (or fragment) to send out

Returns

Number of bytes written. -WM_FAIL in case of an error.

8.53.2.3 int ws_frame_recv (ws_context_t * *ws_ctx*, ws_frame_t * *f*)

Receive a websocket frame

Parameters

in	*ws_ctx*	WS Context returned from a call to ws_upgrade_socket()
out	*f*	The frame (or fragment) that is received. After this call, f->data contains the data that is received, while f->data_len is the length of this data.

Returns

Number of bytes read. -WM_FAIL in case of an error

8.53.2.4 void ws_close_ctx (ws_context_t * *ws_ctx*)

Close Web Socket context

Parameters

in	*ws_ctx*	WS Context returned from a call to ws_upgrade_socket()

8.53.2.5 void ws_set_genmask_callback (int(*)(uint8_t *buf, size_t len, void *user_data) *cb*)

Set websocket mask callback to generate mask key

The SDK has a default masking function that uses get_random_sequence() to generate mask key. This function can be used to override it. It is executed while sending masked data.

Parameters

in	*cb*	A user defined callback to generate mask key. The implementation of this function must write exactly len bytes of mask key to buf. This function should return WM_SUCCESS if successful, -WM_FAIL in case of an error. Note that user_data is not required in this case.

8.53.3 Typedef Documentation

8.53.3.1 typedef struct wslay_frame ws_frame_t

Web Sockets Transfer Frame

8.53.4 Enumeration Type Documentation

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.53.4.1 enum ws_event_t

Websocket events

8.54 wifi-decl.h File Reference

Wifi structure declarations.

8.54.1 Macro Documentation

8.54.1.1 #define MLAN_MAX_VER_STR_LEN 128

Version string buffer length

8.54.1.2 #define BSS_TYPE_STA 0

BSS type : STA

8.54.1.3 #define BSS_TYPE_UAP 1

BSS type : UAP

8.54.1.4 #define MLAN_MAX_SSID_LENGTH (32)

MLAN Maximum SSID Length

8.54.2 Typedef Documentation

8.54.2.1 typedef { ... } wifi_channel_desc_t

Data structure for Channel descriptor

Set CFG data for Tx power limitation

start_freq: Starting Frequency of the band for this channel 2407, 2414 or 2400 for 2.4 GHz 5000 4000 chan_width: Channel Width 20 chan_num : Channel Number

8.54.2.2 typedef { ... } wifi_chan_trpc_entry_t

Data structure for Modulation Group

mod_group : ModulationGroup 0: CCK (1,2,5.5,11 Mbps) 1: OFDM (6,9,12,18 Mbps) 2: OFDM (24,36 Mbps) 3: OFDM (48,54 Mbps) 4: HT20 (0,1,2) 5: HT20 (3,4) 6: HT20 (5,6,7) 7: HT40 (0,1,2) 8: HT40 (3,4) 9: HT40 (5,6,7) 10: HT2_20 (8,9,10) 11: HT2_20 (11,12) 12: HT2_20 (13,14,15) 13: HT2_40 (8,9,10) 14: HT2_40 (11,12) 15: HT2_40 (13,14,15) tx_power : Power Limit in dBm

8.54.2.3 typedef { ... } chan_trpc_config_t

Data structure for TRPC config

For TRPC support

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.54.2.4 typedef { ... } chan_trpc_t

Data structure for Channel TRPC config

For TRPC support

8.54.3 Variable Documentation

8.54.3.1 t_u16 start_freq

Starting frequency of the band for this channel

8.54.3.2 t_u8 chan_width

Channel width

8.54.3.3 t_u8 chan_num

Channel Number

8.54.3.4 t_u8 mod_group

Modulation group

8.54.3.5 t_u8 tx_power

Tx Power

8.54.3.6 t_u8 num_mod_grps

Number of modulation groups

8.54.3.7 wifi_channel_desc_t chan_desc

Chnannel descriptor

8.54.3.8 wifi_chan_trpc_entry_t chan_trpc_entry[10]

Channel Modulation groups

8.54.3.9 t_u8 num_chans

Number of Channels

8.54.3.10 chan_trpc_config_t chan_trpc_config[40]

TRPC config

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.55 wifi_events.h File Reference

Wi-Fi events.

8.55.1 Enumeration Type Documentation

8.55.1.1 enum wifi_event

Wifi events

8.55.1.2 enum wlan_bss_type

Network wireless BSS Type

8.55.1.3 enum wlan_bss_role

Network wireless BSS Role

Enumerator

WLAN_BSS_ROLE_STA Infrastructure network. The system will act as a station connected to an Access Point.

WLAN_BSS_ROLE_UAP uAP (micro-AP) network. The system will act as an uAP node to which other Wireless clients can connect.

WLAN_BSS_ROLE_ANY Either Infrastructure network or micro-AP network

8.55.1.4 enum wifi_wakeup_event_t

This enum defines various wakeup events for which wakeup will occur

Enumerator

WIFI_WAKE_ON_ALL_BROADCAST Wakeup on broadcast

WIFI_WAKE_ON_UNICAST Wakeup on uniccast

WIFI_WAKE_ON_MAC_EVENT Wakeup on MAC event

WIFI_WAKE_ON_MULTICAST Wakeup on multicast

WIFI_WAKE_ON_ARP_BROADCAST Wakeup on ARP broadcast

WIFI_WAKE_ON_MGMT_FRAME Wakeup on receiving a management frame

8.55.1.5 enum wifi_state

Enum for wifi card state

Enumerator

WIFI_UNINIT Un-initialized state

WIFI_INACTIVE Idle state

WIFI_ACTIVE Active state

WIFI_DEEP_SLEEP Deep sleep power save state

WIFI_PDN Power down power state

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.56 wlan.h File Reference

WLAN Connection Manager.

8.56.1 Detailed Description

The WLAN Connection Manager (WLCMGR) is one of the core components that provides WiFi-level functionality like scanning for networks, starting a network (Access Point) and associating / disassociating with other wireless networks. The WLCMGR manages two logical interfaces, the station interface and the micro-AP interface. Both these interfaces can be active at the same time.

8.56.2 Usage

The WLCMGR is initialized by calling wlan_init() and started by calling wlan_start(), one of the arguments of this function is a callback handler. Many of the WLCMGR tasks are asynchronous in nature, and the events are provided by invoking the callback handler. The various usage scenarios of the WLCMGR are outlined below:

- **Scanning:** A call to wlan_scan() initiates an asynchronous scan of the nearby wireless networks. The results are reported via the callback handler.
- **Network Profiles:** Starting / stopping wireless interfaces or associating / disassociating with other wireless networks is managed through network profiles. The network profiles record details about the wireless network like the SSID, type of security, security passphrase among other things. The network profiles can be managed by means of the wlan_add_network() and wlan_remove_network() calls.
- **Association:** The wlan_connect() and wlan_disconnect() calls can be used to manage connectivity with other wireless networks (Access Points). These calls manage the station interface of the system.
- **Starting a Wireless Network:** The wlan_start_network() and wlan_stop_network() calls can be used to start/stop our own (micro-AP) network. These calls manage the micro-AP interface of the system.

8.56.3 Station State Machine

The WLAN Connection Manager station state diagram is as shown below. The Yellow boxes indicate the various states. The top half is the name of the state, and the bottom half consists of any code that is executed on entering that state. Labels on the transitions indicate when that particular transition is taken.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.56.4 Micro-AP State Machine

The WLAN Connection Manager micro-AP state diagram is as shown below. The Yellow boxes indicate the various states. The top half is the name of the state, and the bottom half consists of any code that is executed on entering that state. Labels on the transitions indicate when that particular transition is taken.

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.56.5 Function Documentation

8.56.5.1 int wlan_init (flash_desc_t * *fl*)

Initialize the SDIO driver and create the wifi driver thread.

Parameters

in	*fl*	Flash descriptor of the WLAN firmware on flash. wm_wlan_init function describes how to derive flash descriptor.

Returns

WM_SUCCESS if the WLAN Connection Manager service has initialized successfully.
Negative value if Initialization failed.

8.56.5.2 int wlan_start (int(*)(enum wlan_event_reason reason, void *data) *cb*)

Start the WLAN Connection Manager service.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

This function starts the WLAN Connection Manager.

Note

The status of the WLAN Connection Manager is notified asynchronously through the callback, *cb*, with a WLAN_REASON_INITIALIZED event (if initialization succeeded) or WLAN_REASON_INITIALIZATION_FAILED (if initialization failed).
If the WLAN Connection Manager fails to initialize, the caller should stop WLAN Connection Manager via wlan_stop() and try wlan_start() again.

Parameters

in	*cb*	A pointer to a callback function that handles WLAN events. All further WLCMGR events will be notified in this callback. Refer to enum wlan_event_reason for the various events for which this callback is called.

Returns

WM_SUCCESS if the WLAN Connection Manager service has started successfully.
-WM_E_INVAL if the *cb* pointer is NULL.
-WM_FAIL if an internal error occurred.
WLAN_ERROR_STATE if the WLAN Connection Manager is already running.

8.56.5.3 int wlan_stop (void)

Stop the WLAN Connection Manager service.

This function stops the WLAN Connection Manager, causing station interface to disconnect from the currently connected network and stop the micro-AP interface.

Returns

WM_SUCCESS if the WLAN Connection Manager service has been stopped successfully.
WLAN_ERROR_STATE if the WLAN Connection Manager was not running.

8.56.5.4 void wlan_deinit (void)

Deinitialize SDIO driver, send shutdown command to WLAN firmware and delete the wifi driver thread.

8.56.5.5 void wlan_initialize_uap_network (struct wlan_network * *net*)

WLAN initialize micro-AP network information

This API intializes a default micro-AP network. The network ssid, passphrase is initialized to NULL. Channel is set to auto. The IP Address of the micro-AP interface is 192.168.10.1/255.255.255.0. Network name is set to 'uap-network'.

Parameters

out	*net*	Pointer to the initialized micro-AP network

8.56.5.6 int wlan_add_network (struct wlan_network * *network*)

Add a network profile to the list of known networks.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

This function copies the contents of *network* to the list of known networks in the WLAN Connection Manager. The network's 'name' field must be unique and between WLAN_NETWORK_NAME_MIN_LENGTH and WLAN_NETWORK_NAME_MAX_LENGTH characters. The network must specify at least an SSID or BSSID. The WLAN Connection Manager may store up to WLAN_MAX_KNOWN_NETWORKS networks.

Note

Profiles for the station interface may be added only when the station interface is in the WLAN_DISCONNECTED or WLAN_CONNECTED state.
This API can be used to add profiles for station or micro-AP interfaces.

Parameters

in	*network*	A pointer to the wlan_network that will be copied to the list of known networks in the WLAN Connection Manager successfully.

Returns

WM_SUCCESS if the contents pointed to by *network* have been added to the WLAN Connection Manager.
-WM_E_INVAL if *network* is NULL or the network name is not unique or the network name length is not valid.
-WM_E_NOMEM if there was no room to add the network.
WLAN_ERROR_STATE if the WLAN Connection Manager was running and not in the WLAN_DISCONNECTED, WLAN_ASSOCIATED or WLAN_CONNECTED state.

8.56.5.7 int wlan_remove_network (const char * *name*)

Remove a network profile from the list of known networks.

This function removes a network (identified by its name) from the WLAN Connection Manager, disconnecting from that network if connected.

Note

This function is asynchronous if it is called while the WLAN Connection Manager is running and connected to the network to be removed. In that case, the WLAN Connection Manager will disconnect from the network and generate an event with reason WLAN_REASON_USER_DISCONNECT. This function is synchronous otherwise.
This API can be used to remove profiles for station or micro-AP interfaces. Station network will not be removed if it is in WLAN_CONNECTED state and uAP network will not be removed if it is in WLAN_UAP_STARTED state.

Parameters

in	*name*	A pointer to the string representing the name of the network to remove.

Returns

WM_SUCCESS if the network named *name* was removed from the WLAN Connection Manager successfully. Otherwise, the network is not removed.
WLAN_ERROR_STATE if the WLAN Connection Manager was running and the station interface was not in the WLAN_DISCONNECTED state.
-WM_E_INVAL if *name* is NULL or the network was not found in the list of known networks.
-WM_FAIL if an internal error occurred while trying to disconnect from the network specified for removal.

8.56.5.8 int wlan_connect (char * *name*)

Connect to a wireless network (Access Point).

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

When this function is called, WLAN Connection Manager starts connection attempts to the network specified by *name*. The connection result will be notified asynchronously to the WLCMGR callback when the connection process has completed.

When connecting to a network, the event refers to the connection attempt to that network.

Calling this function when the station interface is in the WLAN_DISCONNECTED state will, if successful, cause the interface to transition into the WLAN_CONNECTING state. If the connection attempt succeeds, the station interface will transition to the WLAN_CONNECTED state, otherwise it will return to the WLAN_DISCONNECTED state. If this function is called while the station interface is in the WLAN_CONNECTING or WLAN_CONNECTED state, the WLAN Connection Manager will first cancel its connection attempt or disconnect from the network, respectively, and generate an event with reason WLAN_REASON_USER_DISCONNECT. This will be followed by a second event that reports the result of the new connection attempt.

If the connection attempt was successful the WLCMGR callback is notified with the event WLAN_REASON_SUCCESS, while if the connection attempt fails then either of the events, WLAN_REASON_NETWORK_NOT_FOUND, WLAN_REASON_NETWORK_AUTH_FAILED, WLAN_REASON_CONNECT_FAILED or WLAN_REASON_ADDRESS_FAILED are reported as appropriate.

Parameters

in	*name*	A pointer to a string representing the name of the network to connect to.

Returns

WM_SUCCESS if a connection attempt was started successfully
WLAN_ERROR_STATE if the WLAN Connection Manager was not running.
-WM_E_INVAL if there are no known networks to connect to or the network specified by *name* is not in the list of known networks or network *name* is NULL.
-WM_FAIL if an internal error has occurred.

8.56.5.9 int wlan_disconnect (void)

Disconnect from the current wireless network (Access Point).

When this function is called, the WLAN Connection Manager attempts to disconnect the station interface from its currently connected network (or cancel an in-progress connection attempt) and return to the WLAN_DISCONNECTED state. Calling this function has no effect if the station interface is already disconnected.

Note

This is an asynchronous function and successful disconnection will be notified using the WLAN_REASON_USER_DISCONNECT.

Returns

WM_SUCCESS if successful
WLAN_ERROR_STATE otherwise

8.56.5.10 int wlan_start_network (const char * *name*)

Start a wireless network (Access Point).

When this function is called, the WLAN Connection Manager starts the network specified by *name*. The network with the specified *name* must be first added using wlan_add_network and must be a micro-AP network with a valid SSID.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Note

The WLCMGR callback is asynchronously notified of the status. On success, the event WLAN_REASON_UAP_SUCCESS is reported, while on failure, the event WLAN_REASON_UAP_START_FAILED is reported.

Parameters

in	*name*	A pointer to string representing the name of the network to connect to.

Returns

WM_SUCCESS if successful.
WLAN_ERROR_STATE if in power save state or uAP already running.
-WM_E_INVAL if *name* was NULL or the network *name* was not found or it not have a specified SSID.

8.56.5.11 int wlan_stop_network (const char * *name*)

Stop a wireless network (Access Point).

When this function is called, the WLAN Connection Manager stops the network specified by *name*. The specified network must be a valid micro-AP network that has already been started.

Note

The WLCMGR callback is asynchronously notified of the status. On success, the event WLAN_REASON_UAP_STOPPED is reported, while on failure, the event WLAN_REASON_UAP_STOP_FAILED is reported.

Parameters

in	*name*	A pointer to a string representing the name of the network to stop.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *name* was NULL or the network *name* was not found or that the network *name* is not a micro-AP network or it is a micro-AP network but does not have a specified SSID.

8.56.5.12 int wlan_get_mac_address (uint8_t * *dest*)

Retrieve the wireless MAC address of station/micro-AP interface.

This function copies the MAC address of the wireless interface to the 6-byte array pointed to by *dest*. In the event of an error, nothing is copied to *dest*.

Parameters

out	*dest*	A pointer to a 6-byte array where the MAC address will be copied.

Returns

WM_SUCCESS if the MAC address was copied.
-WM_E_INVAL if *dest* is NULL.

8.56.5.13 int wlan_get_wfd_mac_address (uint8_t * *dest*)

Retrieve the wireless MAC address of Wi-Fi Direct interface.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

This function copies the MAC address of the wireless interface to the 6-byte array pointed to by *dest*. In the event of an error, nothing is copied to *dest*.

Parameters

out	*dest*	A pointer to a 6-byte array where the MAC address will be copied.

Returns

WM_SUCCESS if the MAC address was copied.
-WM_E_INVAL if *dest* is NULL.

8.56.5.14 int wlan_get_address (struct wlan_ip_config * *addr*)

Retrieve the IP address configuration of the station interface.

This function retrieves the IP address configuration of the station interface and copies it to the memory location pointed to by *addr*.

Note

This function may only be called when the station interface is in the WLAN_CONNECTED state.

Parameters

out	*addr*	A pointer to the wlan_ip_config.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *addr* is NULL.
WLAN_ERROR_STATE if the WLAN Connection Manager was not running or was not in the WLAN_CONNECTED state.
-WM_FAIL if an internal error occurred when retrieving IP address information from the TCP stack.

8.56.5.15 int wlan_get_uap_address (struct wlan_ip_config * *addr*)

Retrieve the IP address of micro-AP interface.

This function retrieves the current IP address configuration of micro-AP and copies it to the memory location pointed to by *addr*.

Note

This function may only be called when the micro-AP interface is in the WLAN_UAP_STARTED state.

Parameters

out	*addr*	A pointer to the wlan_ip_config.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *addr* is NULL.
WLAN_ERROR_STATE if the WLAN Connection Manager was not running or the micro-AP interface was not in the WLAN_UAP_STARTED state.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

-WM_FAIL if an internal error occurred when retrieving IP address information from the TCP stack.

8.56.5.16 int wlan_get_wfd_address (struct wlan_ip_config * *addr*)

Retrieve the IP address configuration of the Wi-Fi Direct interface.

This function retrieves the IP address configuration of the Wi-Fi Direct interface and copies it to the memory location pointed to by *addr*.

Note

This function may only be called when the Wi-Fi Direct interface is in the WLAN_CONNECTED or WLAN_P2P_STARTED state.

Parameters

out	*addr*	A pointer to the wlan_ip_config.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *addr* is NULL.
WLAN_ERROR_STATE if the WLAN Connection Manager was not running or was client and not in the WLAN_CONNECTED or was GO and not in WLAN_P2P_STARTED state.
-WM_FAIL if an internal error occurred when retrieving IP address information from the TCP stack.

8.56.5.17 int wlan_get_current_network (struct wlan_network * *network*)

Retrieve the current network configuration of station interface.

This function retrieves the current network configuration of station interface when the station interface is in the WLAN_CONNECTED state.

Parameters

out	*network*	A pointer to the wlan_network.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *network* is NULL.
WLAN_ERROR_STATE if the WLAN Connection Manager was not running or not in the WLAN_CONNECTED state.

8.56.5.18 int wlan_get_current_uap_network (struct wlan_network * *network*)

Retrieve the current network configuration of micro-AP interface.

This function retrieves the current network configuration of micro-AP interface when the micro-AP interface is in the WLAN_UAP_STARTED state.

Parameters

out	*network*	A pointer to the wlan_network.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *network* is NULL.
WLAN_ERROR_STATE if the WLAN Connection Manager was not running or not in the WLAN_UAP_STARTED state.

8.56.5.19 int wlan_get_current_wfd_network (struct wlan_network * *network*)

Retrieve the current network configuration of Wi-Fi Direct interface.

This function retrieves the current network configuration of Wi-Fi Direct interface when the the as client interface is in the WLAN_CONNECTED state or as a GO interface is in WLAN_P2P_STARTED state.

Parameters

out	*network*	A pointer to the wlan_network.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *network* is NULL.
WLAN_ERROR_STATE if the WLAN Connection Manager was not running or when as a client not in the WLAN_CONNECTED state or as a GO not in WLAN_P2P_STARTED state.

8.56.5.20 int is_uap_started (void)

Retrieve the status information of the micro-AP interface.

Returns

TRUE if micro-AP interface is in WLAN_UAP_STARTED state.
FALSE otherwise.

8.56.5.21 bool is_sta_connected (void)

Retrieve the status information of the station interface.

Returns

TRUE if station interface is in WLAN_CONNECTED state.
FALSE otherwise.

8.56.5.22 bool is_sta_ipv4_connected (void)

Retrieve the status information of the ipv4 network of station interface.

Returns

TRUE if ipv4 network of station interface is in WLAN_CONNECTED state.
FALSE otherwise.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.56.5.23 int is_wfd_started (void)

Retrieve the status information of the Wi-Fi Direct interface.

Returns

TRUE if Wi-Fi Direct interface is in WLAN_P2P_STARTED state.
FALSE otherwise.

8.56.5.24 int wlan_get_network (unsigned int *index*, struct wlan_network * *network*)

Retrieve the information about a known network using *index*.

This function retrieves the contents of a network at *index* in the list of known networks maintained by the WLAN Connection Manager and copies it to the location pointed to by *network*.

Note

wlan_get_network_count() may be used to retrieve the number of known networks. wlan_get_network() may be used to retrieve information about networks at *index* 0 to one minus the number of networks.
This function may be called regardless of whether the WLAN Connection Manager is running. Calls to this function are synchronous.

Parameters

in	*index*	The index of the network to retrieve.
out	*network*	A pointer to the wlan_network where the network configuration for the network at *index* will be copied.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *network* is NULL or *index* is out of range.

8.56.5.25 int wlan_get_network_byname (char * *name*, struct wlan_network * *network*)

Retrieve information about a known network using *name*.

This function retrieves the contents of a named network in the list of known networks maintained by the WLAN Connection Manager and copies it to the location pointed to by *network*.

Note

This function may be called regardless of whether the WLAN Connection Manager is running. Calls to this function are synchronous.

Parameters

in	*name*	The name of the network to retrieve.
out	*network*	A pointer to the wlan_network where the network configuration for the network having name as *name* will be copied.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *network* is NULL or *name* is NULL.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.56.5.26 int wlan_get_network_count (unsigned int * *count*)

Retrieve the number of networks known to the WLAN Connection Manager.

This function retrieves the number of known networks in the list maintained by the WLAN Connection Manager and copies it to *count*.

Note

This function may be called regardless of whether the WLAN Connection Manager is running. Calls to this function are synchronous.

Parameters

out	*count*	A pointer to the memory location where the number of networks will be copied.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *count* is NULL.

8.56.5.27 int wlan_get_connection_state (enum **wlan_connection_state** * *state*)

Retrieve the connection state of station interface.

This function retrieves the connection state of station interface, which is one of WLAN_DISCONNECTED, WLAN_CONNECTING, WLAN_ASSOCIATED or WLAN_CONNECTED.

Parameters

out	*state*	A pointer to the wlan_connection_state where the current connection state will be copied.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *state* is NULL
WLAN_ERROR_STATE if the WLAN Connection Manager was not running.

8.56.5.28 int wlan_get_uap_connection_state (enum **wlan_connection_state** * *state*)

Retrieve the connection state of micro-AP interface.

This function retrieves the connection state of micro-AP interface, which is one of WLAN_UAP_STARTED, or WLAN_UAP_STOPPED.

Parameters

out	*state*	A pointer to the wlan_connection_state where the current connection state will be copied.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *state* is NULL
WLAN_ERROR_STATE if the WLAN Connection Manager was not running.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.56.5.29 int wlan_get_wfd_connection_state (enum wlan_connection_state * *state*)

Retrieve the connection state of Wi-Fi Direct interface.

This function retrieves the connection state of Wi-Fi Direct interface, which is one of WLAN_P2P_STARTED, or WLAN_P2P_STOPPED.

Parameters

out	*state*	A pointer to the wlan_connection_state where the current connection state will be copied.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *state* is NULL
WLAN_ERROR_STATE if the WLAN Connection Manager was not running.

8.56.5.30 int wlan_scan (int(*)(unsigned int count) *cb*)

Scan for wireless networks.

When this function is called, the WLAN Connection Manager starts scan for wireless networks. On completion of the scan the WLAN Connection Manager will call the specified callback function *cb*. The callback function can then retrieve the scan results by using the wlan_get_scan_result() function.

Note

This function may only be called when the station interface is in the WLAN_DISCONNECTED or WLAN_CONNECTED state. Scanning is disabled in the WLAN_CONNECTING state.
This function will block until it can issue a scan request if called while another scan is in progress.

Parameters

in	*cb*	A pointer to the function that will be called to handle scan results when they are available.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *cb* is NULL,
WLAN_ERROR_STATE if the WLAN Connection Manager was not running or not in the WLAN_DISCONNECTED or WLAN_CONNECTED states.
-WM_FAIL if an internal error has occurred and the system is unable to scan.

8.56.5.31 int wlan_get_scan_result (unsigned int *index*, struct wlan_scan_result * *res*)

Retrieve a scan result.

This function may be called to retrieve scan results when the WLAN Connection Manager has finished scanning. It must be called from within the scan result callback (see wlan_scan()) as scan results are valid only in that context. The callback argument 'count' provides the number of scan results that may be retrieved and wlan_get_scan_result() may be used to retrieve scan results at *index* 0 through that number.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Note

This function may only be called in the context of the scan results callback.
Calls to this function are synchronous.

Parameters

in	*index*	The scan result to retrieve.
out	*res*	A pointer to the wlan_scan_result where the scan result information will be copied.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *res* is NULL
WLAN_ERROR_STATE if the WLAN Connection Manager was not running
-WM_FAIL if the scan result at *index* could not be retrieved (that is, *index* is out of range).

8.56.5.32 int wlan_enable_ed_mac_mode (short *offset*)

Enable ED MAC mode in Wireless Firmware.

Note

When ed mac mode is enabled, Wireless Firmware will behave following way:

when background noise had reached -70dB or above, WiFi chipset/module should hold data transmitting until condition is removed. It is applicable for both 5GHz and 2.4GHz bands.

This function may be called to enable ed mac mode in firmware. This should be call before wlan_init() function.

Parameters

in	*offset*	Offset value can be 0 no offset, 0xffff offset set to -1 or 1 offset set to 1.

Returns

WM_SUCCESS if the call was successful.
-WM_FAIL if failed.

8.56.5.33 void wlan_set_cal_data (uint8_t * *cal_data,* unsigned int *cal_data_size*)

Set wireless calibration data in WLAN firmware.

This function may be called to set wireless calibration data in firmware. This should be call before wlan_init() function.

Parameters

in	*cal_data*	The calibration data buffer
in	*cal_data_size*	Size of calibration data buffer.

8.56.5.34 void wlan_set_mac_addr (uint8_t * *mac*)

Set wireless MAC Address in WLAN firmware.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

This function may be called to set wireless MAC Address in firmware. This should be call before wlan_init() function.

Parameters

in	*mac*	The MAC Address in 6 byte array format like uint8_t mac[] = { 0x00, 0x50, 0x43, 0x21, 0x19, 0x6E};

8.56.5.35 void wlan_configure_listen_interval (int *listen_interval*)

Configure Listen interval of IEEE power save mode.

Note

Delivery Traffic Indication Message (DTIM): It is a concept in 802.11 It is a time duration after which AP will send out buffered BROADCAST / MULTICAST data and stations connected to the AP should wakeup to take this broadcast / multicast data.
Traffic Indication Map (TIM): It is a bitmap which the AP sends with each beacon. The bitmap has one bit each for a station connected to AP.
Each station is recognized by an Association Id (AID). If AID is say 1 bit number 1 is set in the bitmap if unicast data is present with AP in its buffer for station with AID = 1 Ideally AP does not buffer any unicast data it just sends unicast data to the station on every beacon when station is not sleeping. When broadcast data / multicast data is to be send AP sets bit 0 of TIM indicating broadcast / multicast. The occurrence of DTIM is defined by AP. Each beacon has a number indicating period at which DTIM occurs. The number is expressed in terms of number of beacons. This period is called DTIM Period / DTIM interval. For example: If AP has DTIM period = 3 the stations connected to AP have to wake up (if they are sleeping) to receive broadcast /multicast data on every third beacon. Generic: When DTIM period is X AP buffers broadcast data / multicast data for X beacons. Then it transmits the data no matter whether station is awake or not. Listen interval: This is time interval on station side which indicates when station will be awake to listen i.e. accept data. Long listen interval: It comes into picture when station sleeps (IEEEPS) and it does not want to wake up on every DTIM So station is not worried about broadcast data/multicast data in this case. This should be a design decision what should be chosen Firmware suggests values which are about 3 times DTIM at the max to gain optimal usage and reliability. In the IEEEPS power save mode, the WiFi firmware goes to sleep and periodically wakes up to check if the AP has any pending packets for it. A longer listen interval implies that the WiFi card stays in power save for a longer duration at the cost of additional delays while receiving data. Please note that choosing incorrect value for listen interval will causes poor response from device during data transfer. Actual listen interval selected by firmware is equal to closest DTIM. For e.g.:- AP beacon period : 100 ms AP DTIM period : 2 Application request value: 500ms Actual listen interval = 400ms (This is the closest DTIM). Actual listen interval set will be a multiple of DTIM closest to but lower than the value provided by the application.
This API can be called before/after association. The configured listen interval will be used in subsequent association attempt.

Parameters

in	*listen_interval*	Listen interval as below 0 : Unchanged, -1 : Disable, 1-49: Value in beacon intervals, >= 50: Value in TUs

8.56.5.36 void wlan_configure_null_pkt_interval (int *time_in_secs*)

Configure Null packet interval of IEEE power save mode.

Note

In IEEEPS station sends a NULL packet to AP to indicate that the station is alive and AP should not kick it off. If null packet is not send some APs may disconnect station which might lead to a loss of connectivity. The time is specified in seconds. Default value is 30 seconds.
This API should be called before configuring IEEEPS

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*time_in_secs*	: -1 Disables null packet transmission, 0 Null packet interval is unchanged, n Null packet interval in seconds.

8.56.5.37 int wlan_deepsleepps_on ( )

Turn on Deep Sleep Power Save mode.

Note

This call is asynchronous. The system will enter the power-save mode only when all requisite conditions are met. For example, wlan should be disconnected for this to work.

Returns

WM_SUCCESS if the call was successful.
WLAN_ERROR_STATE if the call was made in a state where such an operation is illegal.

8.56.5.38 int wlan_deepsleepps_off ( )

Turn off Deep Sleep Power Save mode.

Note

This call is asynchronous. The system will exit the power-save mode only when all requisite conditions are met.

Returns

WM_SUCCESS if the call was successful.
WLAN_ERROR_STATE if the call was made in a state where such an operation is illegal.

8.56.5.39 int wlan_pdnps_on ( )

Turn on Power Down mode.

Note

This call is asynchronous. The system will enter the power-save mode only when all requisite conditions are met. For example, wlan should be disconnected for this to work.

Returns

WM_SUCCESS if the call was successful.
WLAN_ERROR_STATE if the call was made in a state where such an operation is illegal.

8.56.5.40 int wlan_pdnps_off ( )

Turn off Power Down mode.

Note

This call is asynchronous. The system will exit the power-save mode only when all requisite conditions are met.

Returns

WM_SUCCESS if the call was successful.
WLAN_ERROR_STATE if the call was made in a state where such an operation is illegal.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.56.5.41 int wlan_get_current_ant (unsigned int * *ant*)

This function gets current antenna.

Parameters

out	*ant*	Pointer to antenna variable.

Returns

WM_SUCCESS if successful.
WLAN_ERROR_STATE if unsuccessful.

8.56.5.42 int wlan_set_current_ant (unsigned int *ant*)

This function sets current antenna.

Parameters

in	*ant*	Antenna Valid values are 1, 2 and 65535. 1: Set Antenna 1, 2: Set Antenna 2, 65535: Set Antenna diversity.

Returns

WM_SUCCESS if successful.
WLAN_ERROR_STATE if unsuccessful.

8.56.5.43 void wlan_get_firmware_version_ext (char * *version_str*)

Get the wifi firmware version extension string.

Parameters

out	*version_str*	The buffer in which the wifi firmware version extension string will be copied.

8.56.5.44 int wlan_uap_inact_sleep_ps_on (unsigned int *ctrl_bitmap,* unsigned int *inactivity_to,* unsigned int *min_sleep,* unsigned int *max_sleep,* unsigned int *min_awake,* unsigned int *max_awake*)

Turn on Power Save for micro-AP (Inactivity Timeout)

This puts the micro-AP in a power-save mode after the inactivity timeout. Please refer to the wireless chips documentation for more details about the parameters.

Note

This call is asynchronous. System will enter power-save only when all requisite conditions are met.

Parameters

in	*ctrl_bitmap*	disable (0) or enable (1) protection frame Tx before power save
in	*inactivity_to*	inactivity timeout in microseconds
in	*min_sleep*	minimum sleep duration in microseconds
in	*max_sleep*	maximum sleep duration in microseconds
in	*min_awake*	minimum awake duration in microseconds
in	*max_awake*	maximum awake duration in microseconds

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS if the call was successful.
-WM_E_INVAL if given parameters are not valid.
WLAN_ERROR_STATE if the call was made in a state where such an operation is illegal.

8.56.5.45 int wlan_uap_inact_sleep_ps_off ( )

Turn off Power Save for micro-AP (Inactivity Timeout)

This puts the micro-AP out of the inactivity timeout based power save state.

Note

This call is asynchronous. System will exit power-save only when all requisite conditions are met.

Returns

WM_SUCCESS if the call was successful.
WLAN_ERROR_STATE if the call was made in a state where such an operation is illegal.

8.56.5.46 int wlan_get_ps_mode (enum wlan_ps_mode * *ps_mode*)

Get station interface power save mode.

Parameters

out	*ps_mode*	A pointer to wlan_ps_mode where station interface power save mode will be stored.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *ps_mode* was NULL.

8.56.5.47 int wlan_get_uap_ps_mode (enum wlan_uap_ps_mode * *ps_mode*)

Get micro-AP interface power save mode.

Parameters

out	*ps_mode*	A pointer to wlan_uap_ps_mode where micro-AP power save mode will be stored.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *ps_mode* was NULL.

8.56.5.48 int wlan_get_wfd_ps_mode (enum wlan_wfd_ps_mode * *ps_mode*)

Get Wi-Fi Direct interface power save mode.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

out	*ps_mode*	A pointer to wlan_wfd_ps_mode where Wi-Fi Direct interface power save mode will be stored.

Returns

WM_SUCCESS if successful.
-WM_E_INVAL if *ps_mode* was NULL.

8.56.5.49 int wlan_wlcmgr_send_msg (enum wifi_event *event*, int *reason*, void * *data*)

Send message to WLAN Connection Manager thread.

Parameters

in	*event*	An event from wifi_event.
in	*reason*	A reason code.
in	*data*	A pointer to data buffer associated with event.

Returns

WM_SUCCESS if successful.
-WM_FAIL if failed.

8.56.5.50 int wlan_basic_cli_init (void)

Register basic WLAN CLI commands

This function registers basic WLAN CLI commands like showing version information, MAC address

Note

This function can only be called by the application after wlan_init() called.
This function gets called by wlan_cli_init(), hence only one function out of these two functions should be called in the application.

Returns

WLAN_ERROR_NONE if the CLI commands were registered or
WLAN_ERROR_ACTION if they were not registerd (for example if this function was called while the CLI commands were already registered).

8.56.5.51 int wlan_cli_init (void)

Register WLAN CLI commands.

Try to register the WLAN CLI commands with the CLI subsystem. This function is available for the application for use.

Note

This function can only be called by the application after wlan_init() called.
This function internally calls wlan_basic_cli_init(), hence only one function out of these two functions should be called in the application.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS if the CLI commands were registered or
-WM_FAIL if they were not (for example if this function was called while the CLI commands were already registered).

8.56.5.52 int wlan_iw_cli_init (void)

Register WLAN iw∗ CLI commands.

Register the WLAN iw∗ CLI commands like ifconfig, iwlist, iwconfig etc with the CLI subsystem.

Note

This function can only be called by the application after wlan_init() called.

Returns

WM_SUCCESS if the CLI commands were registered or
-WM_FAIL if they were not (for example if this function was called while the CLI commands were already registered).

8.56.5.53 int wlan_uaputl_cli_init (void)

Register WLAN uaputl CLI commands.

Register the WLAN uaputl CLI commands with the CLI subsystem.

Note

This function can only be called by the application after wlan_init() called.

Returns

WM_SUCCESS if the CLI commands were registered or
-WM_FAIL if they were not (for example if this function was called while the CLI commands were already registered).

8.56.5.54 int wlan_enhanced_cli_init (void)

Register WLAN enhanced CLI commands.

Register the WLAN enhanced CLI commands like set or get tx-power, tx-datarate, tx-modulation etc with the CLI subsystem.

Note

This function can only be called by the application after wlan_init() called.

Returns

WM_SUCCESS if the CLI commands were registered or
-WM_FAIL if they were not (for example if this function was called while the CLI commands were already registered).

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.56.5.55 int wlan_pm_cli_init (void)

Register WLAN Power Mgmt CLI commands.

Register the WLAN Power Mgmt CLI commands like pm-ieeeps-hs-cfg, pm-wifi-deepsleep-enter with the CLI subsystem.

Note

This function can only be called by the application after wlan_init() called.

Returns

WLAN_ERROR_NONE if the CLI commands were registered or
WLAN_ERROR_ACTION if they were not (for example if this function was called while the CLI commands were already registered).

8.56.5.56 void pm_ieeeps_hs_cfg (char *enabled*, unsigned int *wakeup_conditions*)

Configure Power Mgr Framework's IEEEPS/Host Sleep Configuration

Enables/Disables the power manager framework's power savings for the wireless interface. When enabled, the power manager framework opportunistically puts the wireless card into IEEEPS mode. Before putting the Wireless card in power save this also sets the hostsleep configuration on the card as specified. This makes the card generate a wakeup for the MC200 processor if any of the wakeup conditions are met.

Parameters

in	*enabled*	1 to enable, 0 to disable IEEE PS support
in	*wakeup_conditions*	conditions to wake the MC200 up on. This should be a logical OR of the conditions in wlan_wakeup_event_t. Typically devices would want to wake up on WAKE_ON_ALL_BROADCAST, WAKE_ON_UNICAST, WAKE_ON_MAC_EVENT. WAKE_ON_MULTICAST, WAKE_ON_ARP_BROADCAST, WAKE_ON_MGMT_FRAME

Precondition

pm_init() should be called before this function is used . More details are found at mdev_pm.h and pwrmgr.h

Note

This function should be used after Station gets connected to a network.

8.56.5.57 int load_wep_key (const uint8_t * *input*, uint8_t * *output*, uint8_t * *output_len*, const unsigned *max_output_len*)

Validate input string as a WEP key and convert it to binary if it is in hexadecimal form.

Parameters

in	*input*	A pointer to WEP key string.
out	*output*	A pointer to output WEP key in binary format.
in	*max_output_len*	Maximum output len
out	*output_len*	A pointer to output len.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

8.56.5.58 int wlan_sniffer_start (const uint16_t *filter_flags,* const uint8_t *radio_type,* const uint8_t *channel,* void(∗)(const wlan_frame_t ∗frame, const uint16_t len) *sniffer_callback*)

Starts sniffing with station interface on given channel and filters. The captured frames are delivered to application using registered sniffer callback.

Parameters

in	*filter_flags*	A filter for control, management and data frames.
in	*radio_type*	Radio type 2.4 or 5 GHz.
in	*channel*	Channel number to sniff as per radio type.
in	*sniffer_callback*	An application callback to process captured frames.

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

Note

Filter flags: Bit[0]: Enable/disable management frame receive, Bit[1]: Enable/disable control frame receive, Bit[2]: Enable/disable data frame receive.

Radio_type: Band configuration type Bits[7:6]: Reserved (set to 0) Bits[5:4]: Secondary channel offset 00 = no secondary channel 01 = secondary channel is above primary channel 10 = reserved 11 = secondary channel is below primary channel Bits[3:2]: Channel width 00 = 20 MHz channel width 01, 10 and 11 = reserved Bits[1:0]: Band information 00 = 2.4 GHz band 01 = 5 GHz band 10 and 11 = reserved

Note

To sniff packets in 40 MHz channel bandwidth, specify secondary channel offset above/below in radio type.

wlan_sniffer_start(4, 17, 40, sniffer_cb);

Note

The above call with start sniffer for data frames on channel number 40 and 44 in 5 GHz band, as secondary channel specified is above primary channel.

8.56.5.59 int wlan_sniffer_status ( )

Get sniffer status of station interface.

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.56.5.60 int wlan_sniffer_stop ( )

Stops sniffing on station interface.

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

8.56.5.61 unsigned int wlan_get_uap_supported_max_clients ( )

Get maximum number of WLAN firmware supported stations that will be allowed to connect to the uAP.

Returns

Maximum number of WLAN firmware supported stations.

Note

Get operation is allowed in any uAP state.

8.56.5.62 int wlan_get_uap_max_clients (unsigned int * *max_sta_num*)

Get current maximum number of stations that will be allowed to connect to the uAP.

Parameters

out	*max_sta_num*	A pointer to variable where current maximum number of stations of uAP interface will be stored.

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

Note

Get operation is allowed in any uAP state.

8.56.5.63 int wlan_set_uap_max_clients (unsigned int *max_sta_num*)

Set maximum number of stations that will be allowed to connect to the uAP.

Parameters

in	*max_sta_num*	Number of maximum stations for uAP.

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Note

Set operation in not allowed in WLAN_UAP_STARTED state.

8.56.5.64 int wlan_get_sta_tx_power ( )

Get Station interface transmit power

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

8.56.5.65 int wlan_set_sta_tx_power (int *power_level*)

Set Station interface transmit power

Parameters

in	*power_level*	Transmit power level.

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

8.56.5.66 int wlan_get_sta_tx_rate_index ( )

Get Station interface transmit rate index

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

8.56.5.67 int wlan_set_sta_tx_rate_index (int *rate_index*)

Set Station interface transmit rate index

Parameters

in	*rate_index*	Transmit rate index.

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

8.56.5.68 int wlan_get_mgmt_ie (enum wlan_bss_type *bss_type*, unsigned int *index*, void * *buf*, unsigned int * *buf_len*)

Get Management IE for given BSS type (interface) and index.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*bss_type*	BSS Type of interface.
in	*index*	IE index.
out	*buf*	Buffer to store requested IE data.
out	*buf_len*	To store length of IE data.

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

8.56.5.69 int wlan_set_mgmt_ie (enum wlan_bss_type *bss_type*, IEEEtypes_ElementId_t *id*, void * *buf*, unsigned int *buf_len*)

Set Management IE for given BSS type (interface) and index.

Parameters

in	*bss_type*	BSS Type of interface.
in	*id*	Type/ID of Management IE.
in	*buf*	Buffer containing IE data.
in	*buf_len*	Length of IE data.

Returns

IE index if successful.
-WM_FAIL if unsuccessful.

8.56.5.70 int wlan_clear_mgmt_ie (enum wlan_bss_type *bss_type*, unsigned int *index*)

Clear Management IE for given BSS type (interface) and index.

Parameters

in	*bss_type*	BSS Type of interface.
in	*index*	IE index.

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

8.56.5.71 bool wlan_get_11d_enable_status ( )

Get current status of 11D support.

Returns

true if 11d support is enabled by application.
false if not enabled.

8.56.5.72 int wlan_get_current_signal_strength (short * *rssi*, int * *snr*)

Get current RSSI and Signal to Noise ratio from WLAN firmware.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*rssi*	A pointer to variable to store current RSSI
in	*snr*	A pointer to variable to store current SNR.

Returns

WM_SUCCESS if successful.

8.56.5.73 int wlan_inject_frame (const uint8_t * *buff*, size_t *len*)

Frame Tx - Injecting Wireless frames from Host

This function is used to Inject Wireless frames from application directly.

Note

All injected frames will be sent on station interface. Application needs minimum of 2 KBytes stack for successful operation. Also application have to take care of allocating buffer for 802.11 Wireless frame (Header + Data) and freeing allocated buffer. Also this API may not work when Power Save is enabled on station interface.

Parameters

in	*buff*	Buffer holding 802.11 Wireless frame (Header + Data).
in	*len*	Length of the 802.11 Wireless frame.

Returns

WM_SUCCESS on success or error code.

8.56.5.74 int wlan_remain_on_channel (const bool *status*, const uint8_t *channel*, const uint32_t *duration*)

This API is is used to set/cancel the remain on channel configuration.

Note

When status is false, channel and duration parameters are ignored.

Parameters

in	*status*	false : Cancel the remain on channel configuration true : Set the remain on channel configuration
in	*channel*	The channel to configure
in	*duration*	The duration for which to configure

Returns

WM_SUCCESS on success or error code.

8.56.5.75 int wlan_get_otp_user_data (uint8_t * *buf*, uint16_t *len*)

Get User Data from OTP Memory

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*buf*	Pointer to buffer where data will be stored
in	*len*	Number of bytes to read

Returns

WM_SUCCESS if user data read operation is successful.
-WM_E_INVAL if buf is not valid or of insufficient size.
-WM_FAIL if user data field is not present or command fails.

8.56.5.76 int wlan_get_cal_data (wlan_cal_data_t * *cal_data*)

Get Calibration data from WLAN firmware

Parameters

out	*cal_data*	Pointer to calibration data structure where calibration data and it's length will be stored.

Returns

WM_SUCCESS if cal data read operation is successful.
-WM_E_INVAL if cal_data is not valid.
-WM_FAIL if command fails.

Note

The user of this API should free the allocated buffer for calibration data.

8.56.5.77 int wlan_set_trpc (chan_trpc_t * *chan_trpc*)

Set the TRPC channel configuration.

Parameters

in	*chan_trpc*	Channel TRPC configuration.

Returns

WM_SUCCESS on success, error otherwise.

8.56.5.78 int wlan_auto_reconnect_enable (wlan_auto_reconnect_config_t *auto_reconnect_config*)

Enable Auto Reconnect feature in WLAN firmware.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*auto_reconnect_config*	Auto Reconnect configuration structure holding following parameters: 1. reconnect counter(0x1-0xff) - The number of times the WLAN firmware retries connection attempt with AP. The value 0xff means retry forever. (default 0xff). 2. reconnect interval(0x0-0xff) - Time gap in seconds between each connection attempt (default 10). 3. flags - Bit 0: Set to 1: Firmware should report link-loss to host if AP rejects authentication/association while reconnecting. Set to 0: Default behaviour: Firmware does not report link-loss to host on AP rejection and continues internally. Bit 1-15: Reserved.

Returns

WM_SUCCESS if operation is successful.
-WM_FAIL if command fails.

8.56.5.79 int wlan_auto_reconnect_disable ( )

Disable Auto Reconnect feature in WLAN firmware.

Returns

WM_SUCCESS if operation is successful.
-WM_FAIL if command fails.

8.56.5.80 int wlan_get_auto_reconnect_config (wlan_auto_reconnect_config_t * *auto_reconnect_config*)

Get Auto Reconnect configuration from WLAN firmware.

Parameters

out	*auto_reconnect_config*	Auto Reconnect configuration structure where response from WLAN firmware will get stored.

Returns

WM_SUCCESS if operation is successful.
-WM_E_INVAL if auto_reconnect_config is not valid.
-WM_FAIL if command fails.

8.56.6 Macro Documentation

8.56.6.1 #define WLAN_RESCAN_LIMIT 10

The number of times that the WLAN Connection Manager will look for a network before giving up.

8.56.6.2 #define WLAN_NETWORK_NAME_MIN_LENGTH 1

The minimum length for network names, see wlan_network. This must be between 1 and WLAN_NETWORK_NAME_MAX_LENGTH

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.56.6.3 #define WLAN_NETWORK_NAME_MAX_LENGTH 32

The space reserved for storing network names, wlan_network

8.56.6.4 #define WLAN_PSK_MAX_LENGTH 65

The space reserved for storing PSK (password) phrases.

8.56.6.5 #define WLAN_PMK_LENGTH 32

Length of a pairwise master key (PMK). It's always 256 bits (32 Bytes)

8.56.6.6 #define MAX_IPV6_ADDRESSES 3

The maximum number of IPV6 addresses that will be stored

8.56.6.7 #define WLAN_ERROR_NONE 0

The operation was successful.

8.56.6.8 #define WLAN_ERROR_PARAM 1

The operation failed due to an error with one or more parameters.

8.56.6.9 #define WLAN_ERROR_NOMEM 2

The operation could not be performed because there is not enough memory.

8.56.6.10 #define WLAN_ERROR_STATE 3

The operation could not be performed in the current system state.

8.56.6.11 #define WLAN_ERROR_ACTION 4

The operation failed due to an internal error.

8.56.6.12 #define WLAN_ERROR_PS_ACTION 5

The operation to change power state could not be performed

8.56.6.13 #define WLAN_ERROR_NOT_SUPPORTED 6

The requested feature is not supported

8.56.7 Typedef Documentation

8.56.7.1 typedef { ... } wlan_beacon_info_t

WLAN Beacon info

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

This data structure represents a WLAN Beacon info.

8.56.7.2 typedef { ... } wlan_probe_req_info_t

WLAN Probe Request info

This data structure represents a WLAN Probe Request info.

8.56.7.3 typedef { ... } wlan_auth_info_t

WLAN Probe Request info

This data structure represents a WLAN Probe Request info.

8.56.7.4 typedef { ... } wlan_data_info_t

WLAN Data info

This data structure represents a WLAN Data info.

8.56.7.5 typedef { ... } wlan_frame_t

WLAN Frame info

This data structure represents a WLAN Frame info.

8.56.8 Enumeration Type Documentation

8.56.8.1 enum wm_wlan_errno

enum for wlan errors

Enumerator

WLAN_ERROR_FW_DNLD_FAILED The Firmware download operation failed.

WLAN_ERROR_FW_NOT_READY The Firmware ready register not set.

WLAN_ERROR_CARD_NOT_DETECTED The WiFi card not found.

WLAN_ERROR_FW_NOT_DETECTED The WiFi Firmware not found.

WLAN_BSSID_NOT_FOUND_IN_SCAN_LIST BSSID not found in scan list

8.56.8.2 enum wlan_event_reason

WLAN Connection Manager event reason

Enumerator

WLAN_REASON_SUCCESS The WLAN Connection Manager has successfully connected to a network and is now in the WLAN_CONNECTED state.

WLAN_REASON_CONNECT_FAILED The WLAN Connection Manager failed to connect before actual connection attempt with AP due to incorrect wlan network profile.

WLAN_REASON_NETWORK_NOT_FOUND The WLAN Connection Manager could not find the network that it was connecting to (or it has tried all known networks and failed to connect to any of them) and it is now in the WLAN_DISCONNECTED state.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

WLAN_REASON_NETWORK_AUTH_FAILED The WLAN Connection Manager failed to authenticate with the network and is now in the WLAN_DISCONNECTED state.

WLAN_REASON_ADDRESS_SUCCESS DHCP lease has been renewed.

WLAN_REASON_ADDRESS_FAILED The WLAN Connection Manager failed to obtain an IP address or TCP stack configuration has failed or the IP address configuration was lost due to a DHCP error. The system is now in the WLAN_DISCONNECTED state.

WLAN_REASON_LINK_LOST The WLAN Connection Manager has lost the link to the current network.

WLAN_REASON_CHAN_SWITCH The WLAN Connection Manager has received the channel switch announcement from the current network.

WLAN_REASON_WPS_DISCONNECT The WLAN Connection Manager has disconnected from the WPS network (or has canceled a connection attempt) by request and is now in the WLAN_DISCONNECTED state.

WLAN_REASON_USER_DISCONNECT The WLAN Connection Manager has disconnected from the current network (or has canceled a connection attempt) by request and is now in the WLAN_DISCONNECTED state.

WLAN_REASON_INITIALIZED The WLAN Connection Manager is initialized and is ready for use. That is, it's now possible to scan or to connect to a network.

WLAN_REASON_INITIALIZATION_FAILED The WLAN Connection Manager has failed to initialize and is therefore not running. It is not possible to scan or to connect to a network. The WLAN Connection Manager should be stopped and started again via wlan_stop() and wlan_start() respectively.

WLAN_REASON_PS_ENTER The WLAN Connection Manager has entered power save mode.

WLAN_REASON_PS_EXIT The WLAN Connection Manager has exited from power save mode.

WLAN_REASON_UAP_SUCCESS The WLAN Connection Manager has started uAP

WLAN_REASON_UAP_CLIENT_ASSOC A wireless client has joined uAP's BSS network

WLAN_REASON_UAP_CLIENT_DISSOC A wireless client has left uAP's BSS network

WLAN_REASON_UAP_START_FAILED The WLAN Connection Manager has failed to start uAP

WLAN_REASON_UAP_STOP_FAILED The WLAN Connection Manager has failed to stop uAP

WLAN_REASON_UAP_STOPPED The WLAN Connection Manager has stopped uAP

8.56.8.3 enum wlan_wakeup_event_t

Wakeup events for which wakeup will occur

Enumerator

WAKE_ON_ALL_BROADCAST Wakeup on broadcast

WAKE_ON_UNICAST Wakeup on unicast

WAKE_ON_MAC_EVENT Wakeup on MAC event

WAKE_ON_MULTICAST Wakeup on multicast

WAKE_ON_ARP_BROADCAST Wakeup on ARP broadcast

WAKE_ON_MGMT_FRAME Wakeup on receiving a management frame

8.56.8.4 enum wlan_connection_state

WLAN station/micro-AP/Wi-Fi Direct Connection/Status state

Enumerator

WLAN_DISCONNECTED The WLAN Connection Manager is not connected and no connection attempt is in progress. It is possible to connect to a network or scan.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

WLAN_CONNECTING The WLAN Connection Manager is not connected but it is currently attempting to connect to a network. It is not possible to scan at this time. It is possible to connect to a different network.

WLAN_ASSOCIATED The WLAN Connection Manager is not connected but associated.

WLAN_CONNECTED The WLAN Connection Manager is connected. It is possible to scan and connect to another network at this time. Information about the current network configuration is available.

WLAN_UAP_STARTED The WLAN Connection Manager has started uAP

WLAN_UAP_STOPPED The WLAN Connection Manager has stopped uAP

WLAN_P2P_STARTED The WLAN Connection Manager has started P2P uAP

WLAN_P2P_STOPPED The WLAN Connection Manager has stopped P2P uAP

WLAN_SCANNING The WLAN Connection Manager is not connected and network scan is in progress.

WLAN_ASSOCIATING The WLAN Connection Manager is not connected and network association is in progress.

8.56.8.5 enum wlan_ps_mode

Station Power save mode

Enumerator

WLAN_ACTIVE Active mode

WLAN_IEEE IEEE power save mode

WLAN_DEEP_SLEEP Deep sleep power save mode

WLAN_PDN Power down power save mode

8.56.8.6 enum wlan_uap_ps_mode

micro-AP power save mode

Enumerator

WLAN_UAP_ACTIVE Active mode

WLAN_UAP_DTIM_SLEEP DTIM Sleep mode

WLAN_UAP_INACTIVITY_SLEEP Inactivity Sleep mode

8.56.8.7 enum wlan_wfd_ps_mode

Wi-Fi Direct(GO) power save mode

Enumerator

WLAN_P2P_ACTIVE Active mode

WLAN_P2P_DTIM_SLEEP DTIM Sleep mode

WLAN_P2P_INACTIVITY_SLEEP Inactivity Sleep mode

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.56.8.8 enum wlan_security_type

Network security types

Enumerator

WLAN_SECURITY_NONE The network does not use security.

WLAN_SECURITY_WEP_OPEN The network uses WEP security with open key.

WLAN_SECURITY_WEP_SHARED The network uses WEP security with shared key.

WLAN_SECURITY_WPA The network uses WPA security with PSK.

WLAN_SECURITY_WPA2 The network uses WPA2 security with PSK.

WLAN_SECURITY_WPA_WPA2_MIXED The network uses WPA/WPA2 mixed security with PSK

WLAN_SECURITY_EAP_TLS The network uses WPA2 Enterprise EAP-TLS security

WLAN_SECURITY_WILDCARD The network can use any security method. This is often used when the user only knows the name and passphrase but not the security type.

8.56.8.9 anonymous enum

Address types to be used by the element wlan_ip_config.addr_type below

Enumerator

ADDR_TYPE_STATIC static IP address

ADDR_TYPE_DHCP Dynamic IP address

ADDR_TYPE_LLA Link level address

8.56.8.10 enum wlan_frame_type_t

WLAN Frame type

Enumerator

BEACON Beacon

PROBE_REQ Probe Request

AUTH Auth Request

DATA Data

QOS_DATA QOS Data

8.56.9 Variable Documentation

8.56.9.1 uint8_t frame_ctrl_flags

Frame Control flags

8.56.9.2 uint16_t duration

Duration

8.56.9.3 char dest[MLAN_MAC_ADDR_LENGTH]

Destination MAC Address

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.56.9.4 char src[MLAN_MAC_ADDR_LENGTH]

Source MAC Address

8.56.9.5 char bssid[MLAN_MAC_ADDR_LENGTH]

BSSID MAC Address

8.56.9.6 uint16_t seq_frag_num

Sequence and Fragmentation number

8.56.9.7 uint8_t timestamp[8]

Time stamp

8.56.9.8 uint16_t beacon_interval

Beacon Interval

8.56.9.9 uint16_t cap_info

Capability Information

8.56.9.10 uint8_t ssid_element_id

SSID Element ID = 00

8.56.9.11 uint8_t ssid_len

SSID Length

8.56.9.12 char ssid[MLAN_MAX_SSID_LENGTH]

SSID

8.56.9.13 uint16_t qos_ctrl

QoS Control bits

8.56.9.14 wlan_frame_type_t frame_type

WLAN Frame type

8.56.9.15 wlan_beacon_info_t beacon_info

WLAN Beacon info

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.56.9.16 wlan_probe_req_info_t probe_req_info

WLAN Probe Request info

8.56.9.17 wlan_auth_info_t auth_info

WLAN Auth info

8.56.9.18 wlan_data_info_t data_info

WLAN Data info

8.57 wlan_11d.h File Reference

WLAN module 11d API.

8.57.1 Function Documentation

8.57.1.1 static int wlan_set_domain_params (wifi_domain_param_t * *dp*) [inline],[static]

Sets the domain parameters for the Station interface.

Note

This API should be called after WLAN is initialized but before making any connection attempts on station interface.
Refer wlan_uap_set_domain_params API documentation for wifi_sub_band_set_t structures for different regions.

```
// We will be using the KR 2.4 and 5 GHz bands for this example

int nr_sb = (sizeof(subband_EU_AU_KR_CN_2_4GHz)
                + sizeof(subband_EU_AU_KR_5GHz))
                / sizeof(wifi_sub_band_set_t);

// We already have space for first sub band info entry in
// wifi_domain_param_t
wifi_domain_param_t *dp = os_mem_alloc(sizeof(
      wifi_domain_param_t) +
                (sizeof(wifi_sub_band_set_t) * (nr_sb - 1)));

// COUNTRY_CODE_LEN is 3. Add extra ' ' as country code is 2 characters
memcpy(dp->country_code, "KR ", COUNTRY_CODE_LEN);

dp->no_of_sub_band = nr_sb;
memcpy(&dp->sub_band[0], &subband_EU_AU_KR_CN_2_4GHz[0],
                1 * sizeof(wifi_sub_band_set_t));
memcpy(&dp->sub_band[1], &subband_EU_AU_KR_5GHz,
                (nr_sb - 1) * sizeof(wifi_sub_band_set_t));

wlan_set_domain_params(dp);
os_mem_free(dp);
```

Parameters

in	*dp*	The wifi domain parameters

Returns

-WM_E_INVAL if invalid parameters were passed.
WM_SUCCESS if operation was successful.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.57.1.2 static int wlan_set_region_code (uint32_t *region_code*) [inline],[static]

Set 11D region code.

Parameters

in	*region_code*	11D region code to set.

Returns

-WM_FAIL if operation was failed.
WM_SUCCESS if operation was successful.

8.57.1.3 uint8_t* wlan_11d_country_index_2_string (int *country*)

Get country string from country code

This function converts country index to country string

Parameters

in	*country*	Country index

Returns

Country string

8.57.2 Typedef Documentation

8.57.2.1 typedef { ... } country_code_t

802.11d country codes

8.57.3 Enumeration Type Documentation

8.57.3.1 anonymous enum

802.11d country codes

Enumerator

COUNTRY_US US FCC
COUNTRY_CA IC Canada
COUNTRY_SG Singapore
COUNTRY_EU ETSI
COUNTRY_AU Australia
COUNTRY_KR Republic Of Korea
COUNTRY_FR France
COUNTRY_JP Japan
COUNTRY_CN China

8.58 wm-tls.h File Reference

TLS API's.

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.58.1 Detailed Description

The TLS module is a software components that provides an interface to a selected TLS library. Currently this module interfaces with the CyaSSL library. This provides functionality to work with secure TLS connections for both client and server applications.

A TLS client or TLS server application can be built using the APIs present in this module. This module provides a thin layer on top of the third party TLS library. The main goal of this module is to provide a third party library independent API for the users. This protects the other dependant modules from change if in case a different third party library is used in the future.

Most functionality of this module is centered in TLS session init API. Other APIs are thin wrappers over the functions provided by the third party library. One of the major tasks during a session init is to set up all the certificates required for client and/or server certificate validation.

8.58.2 Function Documentation

8.58.2.1 int ctaocrypt_lib_init (void)

Initialize the CTAO Crypto library

Returns

-WM_FAIL if initialization failed.
WM_SUCCESS if operation successful.

8.58.2.2 int tls_lib_init (void)

Initialize the TLS library

Returns

Standard WMSDK return codes.

8.58.2.3 int tls_session_init (tls_handle_t * *h*, int *sockfd*, const tls_init_config_t * *cfg*)

Start an TLS session on the given socket.

Parameters

out	*h*	Pointer to the handle,
in	*sockfd*	A socket descriptor on which 'connect' is called before. This parameter is not to be given for TLS server init. The client socket is to be
in	*cfg*	TLS configuration request structure. To be allocated and populated by the caller.

Returns

Standard WMSDK return codes.

8.58.2.4 SSL_CTX* tls_create_server_context (const tls_init_config_t * *cfg*)

Create server context.

This function creates a TLS context for a server implementation. This context can then be reused for multiple client

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

connections. Note that after the context is created the user buffers containing the certificates and keys can be freed. CyaSSL maintains its own copies.

Postcondition

When this context is not required anymore please call tls_purge_server_context() to free the internal buffers allocated by tls_create_server_context().

Parameters

in	*cfg*	Certificate details.

Returns

Allocated SSL_CTX structure.
NULL if structure could not be allocated.

8.58.2.5 void tls_purge_server_context (SSL_CTX * *ctx*)

Purge the context created by tls_create_server_context()

This function is generally called during HTTP server shutdown.

Parameters

in	*ctx*	Context structure of type SSL_CTX created by tls_create_server_context()

8.58.2.6 SSL_CTX* tls_create_client_context (const tls_client_t * *cfg*)

Create client context.

This function creates a TLS context for a client implementation. This context can then be reused for multiple connections with a server. Note that, after the context is created the user buffers containing the certificates and keys can be freed. CyaSSL maintains its own copies.

Postcondition

When this context is not required anymore please call tls_purge_client_context() to free the internal buffers allocated by tls_create_client_context().

Parameters

in	*cfg*	Certificate details.

Returns

Allocated SSL_CTX structure.
NULL if structure could not be allocated.

8.58.2.7 void tls_purge_client_context (SSL_CTX * *ctx*)

Purge the context created by tls_create_client_context()

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*ctx*	Context structure of type SSL_CTX created by tls_create_client_context()

8.58.2.8 int tls_send (tls_handle_t *h*, const void * *buf*, int *len*)

Send data over an existing TLS connection.

Parameters

in	*h*	Handle returned from a previous call to TLS_session_init
in	*buf*	Buffer to send.
in	*len*	Length to write

Returns

Amount data written to the network

8.58.2.9 int tls_recv (tls_handle_t *h*, void * *buf*, int *max_len*)

Receive data from an existing TLS connection.

Parameters

in	*h*	Handle returned from a previous call to TLS_session_init
in	*buf*	Buffer to receive data.
in	*max_len*	Max length of the buffer.

Returns

Amount data read from the network

8.58.2.10 void tls_close (tls_handle_t * *h*)

Close the tls connection.

This function will not close the socket. It will only terminate TLS connection over it.

Parameters

in,out	*h*	Handle returned from a previous call to TLS_session_init. Will set it to NULL before returning.

8.58.2.11 int tls_pending (tls_handle_t *h*)

Get pending length

Parameters

in	*h*	Handle returned from a previous call to TLS_session_init

Returns

pending length

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.58.2.12 int tls_get_error (tls_handle_t *h*, int *rc*)

Turn error to SSL type

Parameters

in	*h*	Handle returned from a previous call to TLS_session_init
in	*rc*	error code.

Returns

SSL error code

8.58.3 Typedef Documentation

8.58.3.1 typedef int tls_handle_t

The tls_init_config_t parameter in the tls_session_init() takes a pointer to the certificate buffer. The certificate buffer can either have a single (or a chain of) X.509 PEM format certificate(s). If the server uses a self-signed certificate, the same certificate needs to be present in this buffer. If the server uses a CA signed certificate, you need to have either of the following:

- the root CA certificate only
- the chain of certificates (entire or partial starting with the Root CA certificate) Note: The system time needs to be set correctly for successful certificate validation.

8.58.4 Enumeration Type Documentation

8.58.4.1 enum tls_flags_t

TLS session flags. Set single or combination of these in tls_init_config_t flags member before passing it to tls_-session_init()

8.59 wm_net.h File Reference

Network Abstraction Layer.

8.59.1 Detailed Description

This provides the calls related to the network layer. The SDK uses lwIP as the network stack.

Here we document the network utility functions provided by the SDK. The detailed lwIP API documentation can be found at: `http://lwip.wikia.com/wiki/Application_API_layers`

8.59.2 Function Documentation

8.59.2.1 static int net_gethostbyname (const char * *cp*, struct hostent ** *hentry*) [inline],[static]

Get network host entry

Parameters

in	*cp*	Hostname or an IPv4 address in the standard dot notation.
in	*hentry*	Pointer to pointer of host entry structure.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Note

This function is not thread safe. If thread safety is required please use lwip_getaddrinfo() - lwip_freeaddrinfo() combination.

Returns

WM_SUCESS if operation successful.
-WM_FAIL if operation fails.

8.59.2.2 int netif_add_udp_broadcast_filter (uint16_t *port_number*)

Add application defined UDP broadcast filters

By default, UDP broadcast packets received on ports DHCPD(0x43), DHCPC(0x44) and DNS(0x35) are accepted and others are discarded. This is done in order to save on extra processing time and memory requirement in networking stack for un-necessary UDP broadcast packets. Application needs to call this API to allow and process UDP broadcast packet coming on specific port other than ports mentioned above before wlan_init().

It will add filter in process_data_packet() function which will check if the packet received is UDP broadcast packet with destination port matching with the port_number. If yes, it will accept the packet, otherwise discard it.

Currently, this API allows IPv4 as well as IPv6 UDP broadcast traffic coming on given port. In future, an additional parameter needs to be added which will specify IPv4, IPv6 or both.

Parameters

in	*port_number*	Port number on which UDP broadcast packets need to be accepted

Returns

WM_SUCCESS on success
-WM_FAIL otherwise

8.59.2.3 int netif_remove_udp_broadcast_filter (uint16_t *port_number*)

Remove UDP broadcast packet filter

This API will remove UDP broadcast packet filter for a port. And hence, UDP broadcast packets coming on this port will be dropped here onwards.

Parameters

in	*port_number*	Port number on which filter was applied and needs to be removed.

Returns

WM_SUCCESS on success
-WM_FAIL otherwise

8.59.2.4 void* net_sock_to_interface (int *sock*)

Get interface handle from socket descriptor

Given a socket descriptor this API returns which interface it is bound with.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*sock*	socket descriptor

Returns

[out] interface handle

8.59.2.5 void* net_get_sta_handle (void)

Get station interface handle

Some APIs require the interface handle to be passed to them. The handle can be retrieved using this API.

Returns

station interface handle

8.59.2.6 void* net_get_uap_handle (void)

Get micro-AP interface handle

Some APIs require the interface handle to be passed to them. The handle can be retrieved using this API.

Returns

micro-AP interface handle

8.59.2.7 void* net_get_wfd_handle (void)

Get WiFi Direct (P2P) interface handle

Some APIs require the interface handle to be passed to them. The handle can be retrieved using this API.

Returns

WiFi Direct (P2P) interface handle

8.59.2.8 void net_interface_up (void * *intrfc_handle*)

Take interface up

Change interface state to up. Use net_get_sta_handle(), net_get_uap_handle() or net_get_wfd_handle() to get interface handle.

Parameters

in	*intrfc_handle*	interface handle

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

void

8.59.2.9 void net_interface_down (void * *intrfc_handle*)

Take interface down

Change interface state to down. Use net_get_sta_handle(), net_get_uap_handle() or net_get_wfd_handle() to get interface handle.

Parameters

in	*intrfc_handle*	interface handle

Returns

void

8.59.2.10 void net_interface_dhcp_stop (void * *intrfc_handle*)

Stop DHCP client on given interface

Stop the DHCP client on given interface state. Use net_get_sta_handle(), net_get_uap_handle() or net_get_wfd_handle() to get interface handle.

Parameters

in	*intrfc_handle*	interface handle

Returns

void

8.59.2.11 int net_get_if_addr (struct wlan_ip_config * *addr,* void * *intrfc_handle*)

Get interface IP Address in wlan_ip_config

This function will get the IP address of a given interface. Use net_get_sta_handle(), net_get_uap_handle() or net_get_wfd_handle() to get interface handle.

Parameters

out	*addr*	wlan_ip_config
in	*intrfc_handle*	interface handle

Returns

WM_SUCCESS on success or error code.

8.59.2.12 int net_get_if_ip_addr (uint32_t * *ip,* void * *intrfc_handle*)

Get interface IP Address

This function will get the IP Address of a given interface. Use net_get_sta_handle(), net_get_uap_handle() or net_get_wfd_handle() to get interface handle.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

out	*ip*	ip address pointer
in	*intrfc_handle*	interface handle

Returns

WM_SUCCESS on success or error code.

8.59.2.13 int net_get_if_ip_mask (uint32_t * *mask*, void * *intrfc_handle*)

Get interface IP Subnet-Mask

This function will get the Subnet-Mask of a given interface. Use net_get_sta_handle(), net_get_uap_handle() or net_get_wfd_handle() to get interface handle.

Parameters

in	*mask*	Subnet Mask pointer
in	*intrfc_handle*	interface

Returns

WM_SUCCESS on success or error code.

8.59.2.14 void net_ipv4stack_init (void)

Initialize the network stack

This function initializes the network stack. This function is called by wlan_start().

Applications may optionally call this function directly:

if they wish to use the networking stack (loopback interface)

without the wlan functionality.

if they wish to initialize the networking stack even before wlan

comes up.

Note

This function may safely be called multiple times.

8.59.2.15 int raw_low_level_output (const u8_t *interface*, const u8_t * *buf*, t_u32 *len*)

Frame Tx - Injecting Wireless frames from Host

This function is used to Inject Wireless frames from application directly.

Parameters

in	*interface*	Interface on which frame to be injected.
in	*buf*	Buffer holding 802.11 Wireless frame (Header + Data).
in	*len*	Length of the 802.11 Wireless frame.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS on success or error code.

8.60 wm_os.h File Reference

WMSDK OS Abstraction Layer.

8.60.1 Detailed Description

The WMSDK OS abstraction layer provides wrapper APIs over some of the commonly used OS primitives. Since the behaviour and semantics of the various OSes differs widely, some abstraction APIs require a specific handling as listed below.

8.60.2 Usage

The WMSDK OS abstraction layer provides the following types of primitives:

- Thread: Create or delete a thread using os_thread_create() or os_thread_delete(). Block a thread using os_thread_sleep(). Complete a thread's execution using os_thread_self_complete().
- Message Queue: Create or delete a message queue using os_queue_create() or os_queue_delete(). Send a message using os_queue_send() and received a message using os_queue_recv().
- Mutex: Create or delete a mutex using os_mutex_create() or os_mutex_delete(). Acquire a mutex using os_mutex_get() and release it using os_mutex_put().
- Semaphores: Create or delete a semaphore using os_semaphore_create() / os_semaphore_create_counting() or os_semaphore_delete. Acquire a semaphore using os_semaphore_get() and release it using os_semaphore_put().
- Timers: Create or delete a timer using os_timer_create() or os_timer_delete(). Change the timer using os_timer_change(). Activate or de-activate the timer using os_timer_activate() or os_timer_deactivate(). Reset a timer using os_timer_reset().
- Dynamic Memory Allocation: Dynamically allocate memory using os_mem_alloc(), os_mem_calloc() or os_mem_realloc() and free it using os_mem_free().
- Block Allocator: Create or delete memory pools for the block allocator using os_block_pool_create() or os_block_pool_delete(). Allocate memory blocks using os_block_alloc() and free it using os_block_free().

8.60.3 Function Documentation

8.60.3.1 static uint32_t os_get_usec_counter ( ) [inline], [static]

Get the current value of free running microsecond counter

Note

This will wraparound after CNTMAX and the caller is expected to take care of this.

Returns

The current value of microsecond counter.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.60.3.2 uint32_t os_get_timestamp (void)

Returns time in micro-secs since bootup

Note

The value returned will wrap around after sometime and caller is expected to guard itself against this.

Returns

Time in micro-secs since bootup

8.60.3.3 int os_thread_create (os_thread_t * *thandle*, const char * *name*, void(*)(os_thread_arg_t arg) *main_func*, void * *arg*, os_thread_stack_t * *stack*, int *prio*)

Create new thread

This function starts a new thread. The new thread starts execution by invoking main_func(). The parameter arg is passed as the sole argument of main_func().

After finishing execution, the new thread should either call:

- os_thread_self_complete() to suspend itself OR
- os_thread_delete() to delete itself

Failing to do this and just returning from main_func() will result in undefined behavior.

Parameters

out	*thandle*	Pointer to a thread handle
in	*name*	Name of the new thread. A copy of this string will be made by the OS for itself. The maximum name length is defined by the macro configMAX_TASK_NAME-_LEN in FreeRTOS header file . Any name length above it will be truncated.
in	*main_func*	Function pointer to new thread function
in	*arg*	The sole argument passed to main_func()
in	*stack*	A pointer to initialized object of type os_thread_stack_t. The object should be created and initialized using os_thread_stack_define().
in	*prio*	The priority of the new thread. One value among OS_PRIO_0, OS_PRIO_1, OS_PRIO_2, OS_PRIO_3 and OS_PRIO_4 should be passed. OS_PRIO_0 represents the highest priority and OS_PRIO_4 represents the lowest priority.

Returns

WM_SUCCESS if thread was created successfully
-WM_FAIL if thread creation failed

8.60.3.4 static int os_thread_delete (os_thread_t * *thandle*) [inline], [static]

Terminate a thread

This function deletes a thread. The task being deleted will be removed from all ready, blocked, suspended and event lists.

Parameters

in	*thandle*	Pointer to the thread handle of the thread to be deleted. If self deletion is required NULL should be passed.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS if operation success
-WM_FAIL if operation fails

8.60.3.5 static int os_thread_sleep (int *ticks*) [inline],[static]

Sleep for specified number of OS ticks

This function causes the calling thread to sleep and block for the given number of OS ticks. The actual time that the task remains blocked depends on the tick rate. The function os_msec_to_ticks() is provided to convert from real-time to ticks.

Any other thread can wake up this task specifically using the API os_thread_wait_abort()

Parameters

in	*ticks*	Number of ticks to sleep

Returns

0 If slept for given ticks or more
Positive value if woken up before given ticks.

Note

The value returned is amount of ticks left before the task was to be originally scheduled to be woken up. So if sleep was for 10 ticks and the task is woken up after 8 ticks then 2 will be returned.

8.60.3.6 static void os_thread_wait_abort (os_thread_t * *thandle*) [inline],[static]

Wake up a task from sleep.

Parameters

in	*thandle*	Pointer to a handle for the task.

8.60.3.7 static unsigned long os_msec_to_ticks (unsigned long *msecs*) [inline],[static]

Convert milliseconds to OS ticks

This function converts the given millisecond value to the number of OS ticks.

This is useful as functions like os_thread_sleep() accept only ticks as input.

Parameters

in	*msecs*	Milliseconds

Returns

Number of OS ticks corresponding to msecs

8.60.3.8 static unsigned long os_ticks_to_msec (unsigned long *ticks*) [inline],[static]

Convert ticks to milliseconds

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

This function converts the given ticks value to milliseconds. This is useful as some functions, like os_ticks_get(), return values in units of OS ticks.

Parameters

in	*ticks*	OS ticks

Returns

Number of milliseconds corresponding to ticks

8.60.3.9 static void os_thread_self_complete (os_thread_t * *thandle*) [inline],[static]

Suspend the given thread

- The function os_thread_self_complete() will **permanently** suspend the given thread. Passing NULL will suspend the current thread. This function never returns.
- The thread continues to consume system resources. To delete the thread the function os_thread_delete() needs to be called separately.

Parameters

in	*thandle*	Pointer to thread handle

8.60.3.10 int os_queue_create (os_queue_t * *qhandle,* const char * *name,* int *msgsize,* os_queue_pool_t * *poolname*)

Create an OS queue

This function creates a new queue instance. This allocates the storage required by the new queue and returns a handle for the queue.

Parameters

out	*qhandle*	Pointer to the handle of the newly created queue
in	*name*	String specifying the name of the queue
in	*msgsize*	The number of bytes each item in the queue will require. Items are queued by copy, not by reference, so this is the number of bytes that will be copied for each posted item. Each item on the queue must be the same size.
in	*poolname*	The object of the type os_queue_pool_t. The helper macro os_queue_pool_define() helps to define this object.

Returns

WM_SUCCESS if queue creation was successful
-WM_FAIL if queue creation failed

8.60.3.11 static int os_queue_send (os_queue_t * *qhandle,* const void * *msg,* unsigned long *wait*) [inline], [static]

Post an item to the back of the queue.

This function posts an item to the back of a queue. The item is queued by copy, not by reference. This function can also be called from an interrupt service routine.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*qhandle*	Pointer to the handle of the queue
in	*msg*	A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from msg into the queue storage area.
in	*wait*	The maximum amount of time, in OS ticks, the task should block waiting for space to become available on the queue, should it already be full. The function os_msec_to_ticks() can be used to convert from real-time to OS ticks. The special values OS_WAIT_FOREVER and OS_NO_WAIT are provided to respectively wait infinitely or return immediately.

Returns

WM_SUCCESS if send operation was successful
-WM_E_INVAL if invalid parameters are passed
-WM_FAIL if send operation failed

8.60.3.12 static int os_queue_recv (os_queue_t * *qhandle,* void * *msg,* unsigned long *wait*) [inline],[static]

Receive an item from queue

This function receives an item from a queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

Note

This function must not be used in an interrupt service routine.

Parameters

in	*qhandle*	Pointer to handle of the queue
out	*msg*	Pointer to the buffer into which the received item will be copied. The size of the items in the queue was defined when the queue was created. This pointer should point to a buffer as many bytes in size.
in	*wait*	The maximum amount of time, in OS ticks, the task should block waiting for messages to arrive on the queue, should it already be empty. The function os_msec_to_ticks() can be used to convert from real-time to OS ticks. The special values OS_WAIT_FOREVER and OS_NO_WAIT are provided to respectively wait infinitely or return immediately.

Returns

WM_SUCCESS if receive operation was successful
-WM_E_INVAL if invalid parameters are passed
-WM_FAIL if receive operation failed

8.60.3.13 static int os_queue_delete (os_queue_t * *qhandle*) [inline],[static]

Delete queue

This function deletes a queue. It frees all the memory allocated for storing of items placed on the queue.

Parameters

in	*qhandle*	Pointer to handle of the queue to be deleted.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

Currently always returns WM_SUCCESS

8.60.3.14 static int os_queue_get_msgs_waiting (os_queue_t * *qhandle*) [inline], [static]

Return the number of messages stored in queue.

Parameters

in	*qhandle*	Pointer to handle of the queue to be queried.

Returns

Number of items in the queue
-WM_E_INVAL if invalid parameters are passed

8.60.3.15 static int os_setup_idle_function (void(*)(void) *func*) [inline], [static]

Setup idle function

This function sets up a callback function which will be called whenever the system enters the idle thread context.

Parameters

in	*func*	The callback function

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.60.3.16 static int os_setup_tick_function (void(*)(void) *func*) [inline], [static]

Setup tick function

This function sets up a callback function which will be called on every SysTick interrupt.

Parameters

in	*func*	The callback function

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.60.3.17 static int os_remove_idle_function (void(*)(void) *func*) [inline], [static]

Remove idle function

This function removes an idle callback function that was registered previously using os_setup_idle_function().

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*func*	The callback function

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.60.3.18 static int os_remove_tick_function (void(∗)(void) *func*) [inline], [static]

Remove tick function

This function removes a tick callback function that was registered previously using os_setup_tick_function().

Parameters

in	*func*	Callback function

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.60.3.19 static int os_mutex_create (os_mutex_t ∗ *mhandle*, const char ∗ *name*, int *flags*) [inline], [static]

Create mutex

This function creates a mutex.

Parameters

out	*mhandle*	Pointer to a mutex handle
in	*name*	Name of the mutex
in	*flags*	Priority inheritance selection. Valid options are OS_MUTEX_INHERIT or OS_MUTEX_NO_INHERIT.

Note

Currently non-inheritance in mutex is not supported.

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.60.3.20 static int os_mutex_get (os_mutex_t ∗ *mhandle*, unsigned long *wait*) [inline], [static]

Acquire mutex

This function acquires a mutex. Only one thread can acquire a mutex at any given time. If already acquired the callers will be blocked for the specified time duration.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*mhandle*	Pointer to mutex handle
in	*wait*	The maximum amount of time, in OS ticks, the task should block waiting for the mutex to be acquired. The function os_msec_to_ticks() can be used to convert from real-time to OS ticks. The special values OS_WAIT_FOREVER and OS_NO_WAIT are provided to respectively wait infinitely or return immediately.

Returns

WM_SUCCESS when mutex is acquired
-WM_E_INVAL if invalid parameters are passed
-WM_FAIL on failure

8.60.3.21 static int os_mutex_put (os_mutex_t * *mhandle*) [inline],[static]

Release mutex

This function releases a mutex previously acquired using os_mutex_get().

Note

The mutex should be released from the same thread context from which it was acquired. If you wish to acquire and release in different contexts, please use os_semaphore_get() and os_semaphore_put() variants.

Parameters

in	*mhandle*	Pointer to the mutex handle

Returns

WM_SUCCESS when mutex is released
-WM_E_INVAL if invalid parameters are passed
-WM_FAIL on failure

8.60.3.22 static int os_recursive_mutex_create (os_mutex_t * *mhandle,* const char * *name*) [inline],[static]

Create recursive mutex

This function creates a recursive mutex. A mutex used recursively can be 'get' repeatedly by the owner. The mutex doesn't become available again until the owner has called os_recursive_mutex_put() for each successful 'get' request.

Note

This type of mutex uses a priority inheritance mechanism so a task 'get'ing a mutex MUST ALWAYS 'put' the mutex back once no longer required.

Parameters

out	*mhandle*	Pointer to a mutex handle
in	*name*	Name of the mutex as NULL terminated string

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS on success
-WM_E_INVAL on invalid parameter.
-WM_FAIL on error

8.60.3.23 static int os_recursive_mutex_get (os_mutex_t * *mhandle,* unsigned long *wait*) [inline],[static]

Get recursive mutex

This function recursively obtains, or 'get's, a mutex. The mutex must have previously been created using a call to os_recursive_mutex_create().

Parameters

in	*mhandle*	Pointer to mutex handle obtained from os_recursive_mutex_create().
in	*wait*	The maximum amount of time, in OS ticks, the task should block waiting for the mutex to be acquired. The function os_msec_to_ticks() can be used to convert from real-time to OS ticks. The special values OS_WAIT_FOREVER and OS_NO_WAIT are provided to respectively wait for portMAX_DELAY (0xffffffff) or return immediately.

Returns

WM_SUCCESS when recursive mutex is acquired
-WM_FAIL on failure

8.60.3.24 static int os_recursive_mutex_put (os_mutex_t * *mhandle*) [inline],[static]

Put recursive mutex

This function recursively releases, or 'give's, a mutex. The mutex must have previously been created using a call to os_recursive_mutex_create()

Parameters

in	*mhandle*	Pointer to the mutex handle

Returns

WM_SUCCESS when mutex is released
-WM_FAIL on failure

8.60.3.25 static int os_mutex_delete (os_mutex_t * *mhandle*) [inline],[static]

Delete mutex

This function deletes a mutex.

Parameters

in	*mhandle*	Pointer to the mutex handle

Note

A mutex should not be deleted if other tasks are blocked on it.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS on success

8.60.3.26 static int os_semaphore_create (os_semaphore_t * *mhandle,* const char * *name*) [inline],[static]

Create binary semaphore

This function creates a binary semaphore. A binary semaphore can be acquired by only one entity at a given time.

Parameters

out	*mhandle*	Pointer to a semaphore handle
in	*name*	Name of the semaphore

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.60.3.27 static int os_semaphore_create_counting (os_semaphore_t * *mhandle,* const char * *name,* unsigned long *maxcount,* unsigned long *initcount*) [inline],[static]

Create counting semaphore

This function creates a counting semaphore. A counting semaphore can be acquired 'count' number of times at a given time.

Parameters

out	*mhandle*	Pointer to a semaphore handle
in	*name*	Name of the semaphore
in	*maxcount*	The maximum count value that can be reached. When the semaphore reaches this value it can no longer be 'put'
in	*initcount*	The count value assigned to the semaphore when it is created. For e.g. If '0' is passed, then os_semaphore_get() will block until some other thread does an os_semaphore_put().

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.60.3.28 static int os_semaphore_get (os_semaphore_t * *mhandle,* unsigned long *wait*) [inline],[static]

Acquire semaphore

This function acquires a semaphore. At a given time, a binary semaphore can be acquired only once, while a counting semaphore can be acquired as many as 'count' number of times. Once this condition is reached, the other callers of this function will be blocked for the specified time duration.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*mhandle*	Pointer to a semaphore handle
in	*wait*	The maximum amount of time, in OS ticks, the task should block waiting for the semaphore to be acquired. The function os_msec_to_ticks() can be used to convert from real-time to OS ticks. The special values OS_WAIT_FOREVER and OS_NO_WAIT are provided to respectively wait infinitely or return immediately.

Returns

WM_SUCCESS when semaphore is acquired
-WM_E_INVAL if invalid parameters are passed
-WM_FAIL on failure

8.60.3.29 static int os_semaphore_put (os_semaphore_t * *mhandle*) [inline],[static]

Release semaphore

This function releases a semaphore previously acquired using os_semaphore_get().

Note

This function can also be called from interrupt-context.

Parameters

in	*mhandle*	Pointer to a semaphore handle

Returns

WM_SUCCESS when semaphore is released
-WM_E_INVAL if invalid parameters are passed
-WM_FAIL on failure

8.60.3.30 static int os_semaphore_getcount (os_semaphore_t * *mhandle*) [inline],[static]

Get semaphore count

This function returns the current value of a semaphore.

Parameters

in	*mhandle*	Pointer to a semaphore handle

Returns

current value of the semaphore

8.60.3.31 static int os_semaphore_delete (os_semaphore_t * *mhandle*) [inline],[static]

Delete a semaphore

This function deletes the semaphore.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

in	*mhandle*	Pointer to a semaphore handle

Note

Do not delete a semaphore that has tasks blocked on it (tasks that are in the Blocked state waiting for the semaphore to become available)

Returns

WM_SUCCESS on success

8.60.3.32 int os_rwlock_create (os_rw_lock_t * *lock*, const char * *mutex_name*, const char * *lock_name*)

Create reader-writer lock

This function creates a reader-writer lock.

Parameters

in	*lock*	Pointer to a reader-writer lock handle
in	*mutex_name*	Name of the mutex
in	*lock_name*	Name of the lock

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.60.3.33 void os_rwlock_delete (os_rw_lock_t * *lock*)

Delete a reader-write lock

This function deletes a reader-writer lock.

Parameters

in	*lock*	Pointer to the reader-writer lock handle

8.60.3.34 int os_rwlock_write_lock (os_rw_lock_t * *lock*, unsigned int *wait_time*)

Acquire writer lock

This function acquires a writer lock. While readers can acquire the lock on a sharing basis, writers acquire the lock in an exclusive manner.

Parameters

in	*lock*	Pointer to the reader-writer lock handle
in	*wait_time*	The maximum amount of time, in OS ticks, the task should block waiting for the lock to be acquired. The function os_msec_to_ticks() can be used to convert from real-time to OS ticks. The special values OS_WAIT_FOREVER and OS_NO_WAIT are provided to respectively wait infinitely or return immediately.

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.60.3.35 void os_rwlock_write_unlock (os_rw_lock_t * *lock*)

Release writer lock

This function releases a writer lock previously acquired using os_rwlock_write_lock().

Parameters

in	*lock*	Pointer to the reader-writer lock handle

8.60.3.36 int os_rwlock_read_lock (os_rw_lock_t * *lock*, unsigned int *wait_time*)

Acquire reader lock

This function acquires a reader lock. While readers can acquire the lock on a sharing basis, writers acquire the lock in an exclusive manner.

Parameters

in	*lock*	pointer to the reader-writer lock handle
in	*wait_time*	The maximum amount of time, in OS ticks, the task should block waiting for the lock to be acquired. The function os_msec_to_ticks() can be used to convert from real-time to OS ticks. The special values OS_WAIT_FOREVER and OS_NO_WAIT are provided to respectively wait infinitely or return immediately.

Returns

WM_SUCCESS on success
-WM_FAIL on error

8.60.3.37 int os_rwlock_read_unlock (os_rw_lock_t * *lock*)

Release reader lock

This function releases a reader lock previously acquired using os_rwlock_read_lock().

Parameters

in	*lock*	pointer to the reader-writer lock handle

Returns

WM_SUCCESS if unlock operation successful.
-WM_FAIL if unlock operation failed.

8.60.3.38 int os_timer_create (os_timer_t * *timer_t*, const char * *name*, os_timer_tick *ticks*, void(*)(os_timer_arg_t) *call_back*, void * *cb_arg*, os_timer_reload_t *reload*, os_timer_activate_t *activate*)

Create timer

This function creates a timer.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

out	*timer_t*	Pointer to the timer handle
in	*name*	Name of the timer
in	*ticks*	Period in ticks
in	*call_back*	Timer expire callback function
in	*cb_arg*	Timer callback data
in	*reload*	Reload Options, valid values include OS_TIMER_ONE_SHOT or OS_TIMER_PERIODIC.
in	*activate*	Activate Options, valid values include OS_TIMER_AUTO_ACTIVATE or OS_TIMER_NO_ACTIVATE

Returns

WM_SUCCESS if timer created successfully
-WM_FAIL if timer creation fails

8.60.3.39 static int os_timer_activate (os_timer_t * *timer_t*) [inline], [static]

Activate timer

This function activates (or starts) a timer that was previously created using os_timer_create(). If the timer had already started and was already in the active state, then this call is equivalent to os_timer_reset().

Parameters

in	*timer_t*	Pointer to a timer handle

Returns

WM_SUCCESS if timer activated successfully
-WM_FAIL if timer fails to activate

8.60.3.40 static int os_timer_change (os_timer_t * *timer_t*, os_timer_tick *ntime*, os_timer_tick *block_time*) [inline], [static]

Change timer period

This function changes the period of a timer that was previously created using os_time_create(). This function changes the period of an active or dormant state timer.

Parameters

in	*timer_t*	Pointer to a timer handle
in	*ntime*	Time in ticks after which the timer will expire
in	*block_time*	This option is currently not supported

Returns

WM_SUCCESS on success
-WM_FAIL on failure

8.60.3.41 static bool os_timer_is_running (os_timer_t * *timer_t*) [inline], [static]

Check the timer active state

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

This function checks if the timer is in the active or dormant state. A timer is in the dormant state if (a) it has been created but not started, or (b) it has expired and a one-shot timer.

Parameters

in	*timer_t*	Pointer to a timer handle

Returns

true if timer is active
false if time is not active

8.60.3.42 static void* os_timer_get_context (os_timer_t * *timer_t*) [inline], [static]

Get the timer context

This function helps to retrieve the timer context i.e. 'cb_arg' passed to os_timer_create().

Parameters

in	*timer_t*	Pointer to timer handle. The timer handle is received in the timer callback.

Returns

The timer context i.e. the callback argument passed to os_timer_create().

8.60.3.43 static int os_timer_reset (os_timer_t * *timer_t*) [inline], [static]

Reset timer

This function resets a timer that was previously created using using os_timer_create(). If the timer had already been started and was already in the active state, then this call will cause the timer to re-evaluate its expiry time so that it is relative to when os_timer_reset() was called. If the timer was in the dormant state then this call behaves in the same way as os_timer_activate().

Parameters

in	*timer_t*	Pointer to a timer handle

Returns

WM_SUCCESS on success
-WM_FAIL on failure

8.60.3.44 static int os_timer_deactivate (os_timer_t * *timer_t*) [inline], [static]

Deactivate timer

This function deactivates (or stops) a timer that was previously started.

Parameters

in	*timer_t*	handle populated by os_timer_create()

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS on success
-WM_FAIL on failure

8.60.3.45 static int os_timer_delete (os_timer_t * *timer_t*) [inline],[static]

Delete timer

This function deletes a timer.

Parameters

in	*timer_t*	Pointer to a timer handle

Returns

WM_SUCCESS on success
-WM_FAIL on failure

8.60.3.46 static void* os_mem_calloc (size_t *size*) [inline],[static]

Allocate memory and zero it

This function allocates memory dynamically and sets the memory contents to zero.

Parameters

in	*size*	Size of the memory to be allocated

Returns

Pointer to the allocated memory
NULL if allocation fails

8.60.3.47 static int os_heap_add_bank (void * *start*, size_t *size*) [inline],[static]

Add more memory to heap

This function can add additional chunks of memory to the heap. This is useful in scenarios where free memory is available in separate memory banks.

Parameters

in	*start*	Start of the memory bank/chunk
in	*size*	Size of the memory to be added to the heap

Returns

NULL if allocation fails

8.60.3.48 static void os_dump_mem_stats (void) [inline],[static]

This function dumps complete statistics of the heap memory.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.60.4 Macro Documentation

8.60.4.1 #define os_thread_relinquish( ) taskYIELD();

Force a context switch

8.60.4.2 #define os_ticks_get( ) xTaskGetTickCount()

Get current OS tick counter value

8.60.4.3 #define os_total_ticks_get( ) xTaskGetTotalTickCount()

Get (wraparound safe) current OS tick counter. Returns a 64 bit unsigned integer. To give a rough idea, for an OS tick period of 1 mS it takes thousands of years before the counter value returned by this API to wrap around. Thus, users of this API can ignore the wrap around problem

8.60.4.4 #define os_ticks_to_unblock( ) xTaskGetUnblockTime()

Get ticks to next thread wakeup

8.60.4.5 #define os_thread_stack_define(*stackname, stacksize*)

Value:

```
os_thread_stack_t stackname =                          \
                {(stacksize) / (sizeof(portSTACK_TYPE))}
```

Helper macro to define the stack size (in bytes) before a new thread is created using the function os_thread_create().

8.60.4.6 #define os_queue_pool_define(*poolname, poolsize*) os_queue_pool_t poolname = { poolsize };

Define OS Queue pool

This macro helps define the name and size of the queue to be created using the function os_queue_create().

8.60.4.7 #define OS_WAIT_FOREVER portMAX_DELAY

Wait Forever

8.60.4.8 #define OS_NO_WAIT 0

Do Not Wait

8.60.4.9 #define OS_MUTEX_INHERIT 1

Priority Inheritance Enabled

8.60.4.10 #define OS_MUTEX_NO_INHERIT 0

Priority Inheritance Disabled

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.60.4.11 #define os_mem_alloc(*size*) pvPortMalloc(size)

Allocate memory

This function allocates memory dynamically.

Parameters

in	*size*	Size of the memory to be allocated

Returns

Pointer to the allocated memory
NULL if allocation fails

8.60.4.12 #define os_mem_realloc(*old_ptr, new_size*) pvPortReAlloc(old_ptr, new_size)

Reallocate memory

This function attempts to resize a previously allocated memory block.

Parameters

in	*old_ptr*	Pointer to earlier allocated memory
in	*new_size*	The new size

Returns

Pointer to the newly resized memory block
NULL if reallocation fails

8.60.4.13 #define os_mem_free(*ptr*) vPortFree(ptr)

Free Memory

This function frees dynamically allocated memory using any of the dynamic allocation primitives.

Parameters

in	*ptr*	Pointer to the memory to be freed

8.60.4.14 #define os_get_runtime_stats(*__buff__*) vTaskGetRunTimeStats(__buff__)

Get ASCII formatted run time statistics

Please ensure that your buffer is big enough for the formatted data to fit. Failing to do this may cause memory data corruption.

8.60.5 Typedef Documentation

8.60.5.1 typedef struct os_thread_stack os_thread_stack_t

Structure to be used during call to the function os_thread_create(). Please use the macro os_thread_stack_define instead of using this structure directly.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.60.5.2 typedef struct os_queue_pool os_queue_pool_t

Structure used for queue definition

8.60.5.3 typedef int(∗ cb_fn)(os_rw_lock_t ∗plock, unsigned int wait_time)

This is prototype of reader callback

8.60.5.4 typedef enum os_timer_reload os_timer_reload_t

OS Timer reload Options

8.60.5.5 typedef enum os_timer_activate os_timer_activate_t

OS Timer Activate Options

8.60.6 Enumeration Type Documentation

8.60.6.1 enum os_timer_reload

OS Timer reload Options

Enumerator

OS_TIMER_ONE_SHOT Create one shot timer. Timer will be in the dormant state after it expires.

OS_TIMER_PERIODIC Create a periodic timer. Timer will auto-reload after it expires.

8.60.6.2 enum os_timer_activate

OS Timer Activate Options

Enumerator

OS_TIMER_AUTO_ACTIVATE Start the timer on creation.

OS_TIMER_NO_ACTIVATE Do not start the timer on creation.

8.61 wm_supplicant.h File Reference

WMSDK Host Supplicant.

8.61.1 Function Documentation

8.61.1.1 int supplicant_start (int(∗)(enum supp_event event, void ∗data, uint16_t len) *supp_callback*)

Start supplicant thread

Parameters

in	*supp_callback*	supplicant callback funtion pointer.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

WM_SUCCESS on success or -WM_FAIL on error.

8.61.1.2 int supplicant_send_cmd (supplicant_cmd_id_t *cmd_id*, char * *ssid*, int *security*, struct wlan_cipher * *mcstCipher*, struct wlan_cipher * *ucstCipher*, char * *passphrase*, char * *bssid*, uint8_t * *sta_mac*, bool *is_pmf_required*)

Send command to supplicant thread.

Parameters

in	*cmd_id*	Command ID.
in	*ssid*	SSID of known wlan network.
in	*security*	Security of known wlan network.
in	*mcstCipher*	Multicast cipher of known wlan network.
in	*ucstCipher*	Unicast cipher of known wlan network.
in	*passphrase*	Passphrase of known wlan network.
in	*bssid*	BSSID of known wlan network.
in	*sta_mac*	MAC address of station associated to uAP network.
in	*is_pmf_required*	PMF support is required or not.

Returns

WM_SUCCESS on success or -WM_FAIL on error.

8.61.1.3 void supplicant_finish (void)

Stop supplicant thread

8.61.2 Enumeration Type Documentation

8.61.2.1 enum supp_event

enum : SUPP events

Enumerator

SUPP_STARTED SUPP thread started

SUPP_SESSION_STARTED SUPP PBC/PIN Session started

SUPP_SESSION_TIMEOUT SUPP Session registration timeout

SUPP_SESSION_SUCCESSFUL SUPP Session attempt successful

SUPP_SESSION_FAILED SUPP Session failed

SUPP_FAILED SUPP start failed

SUPP_FINISHED SUPP thread stopped

8.62 wm_utils.h File Reference

WMSDK Utility functions.

8.62.1 Detailed Description

Collection of some common helper functions

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.62.2 Function Documentation

8.62.2.1 unsigned int hex2bin (const uint8_t * *ibuf,* uint8_t * *obuf,* unsigned *max_olen*)

Convert a given hex string to a equivalent binary representation.

E.g. If your input string of 4 bytes is {'F', 'F', 'F', 'F'} the output string will be of 2 bytes {255, 255} or to put the same in other way {0xFF, 0xFF}

Note that hex2bin is not the same as strtoul as the latter will properly return the integer in the correct machine binary format viz. little endian. hex2bin however does only in-place like replacement of two ASCII characters to one binary number taking 1 byte in memory.

Parameters

in	*ibuf*	input buffer
out	*obuf*	output buffer
in	*max_olen*	Maximum output buffer length

Returns

length of the binary string

8.62.2.2 void bin2hex (uint8_t * *src,* char * *dest,* unsigned int *src_len,* unsigned int *dest_len*)

Convert given binary array to equivalent hex representation.

Parameters

in	*src*	Input buffer
out	*dest*	Output buffer
in	*src_len*	Length of the input buffer
in	*dest_len*	Length of the output buffer

Returns

void

8.62.2.3 int random_register_handler (random_hdlr_t *func*)

Register a random entropy generator handler

This API allows applications to register their own random entropy generator handlers that will be internally used by get_random_sequence() to add even more randomization to the byte stream generated by it.

Parameters

in	*func*	Function pointer of type random_hdlr_t

Returns

WM_SUCCESS if successful
-WM_E_NOSPC if there is no space available for additional handlers

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.62.2.4 int random_unregister_handler (random_hdlr_t *func*)

Un-register a random entropy generator handler

This API can be used to un-register a handler registered using random_register_handler()

Parameters

in	*func*	Function pointer of type random_hdlr_t used during registering

Returns

WM_SUCCESS if successful
-WM_E_INVAL if the passed pointer is invalid

8.62.2.5 int random_register_seed_handler (random_hdlr_t *func*)

Register a random seed generator handler

For getting better random numbers, the initial seed (ideally required only once on every boot) should also be random. This API allows applications to register their own seed generators. Applications can use any logic such that a different seed is generated every time. A sample seed generator which uses a combination of DAC (generating random noise) and ADC (that internally samples the random noise) along with the flash id has already been provided. Please have a look at sample_initialise_random_seed().

The seed generator handler is called only once by the get_random_sequence() function. Applications can also explicitly initialize the seed by calling random_initialize_seed() after registering a handler.

Parameters

in	*func*	Function pointer of type random_hdlr_t

Returns

WM_SUCCESS if successful
-WM_E_NOSPC if there is no space available for additional handlers

8.62.2.6 int random_unregister_seed_handler (random_hdlr_t *func*)

Un-register a random seed generator handler

This API can be used to un-register a handler registered using random_register_seed_handler()

Parameters

in	*func*	Function pointer of type random_hdlr_t used during registering

Returns

WM_SUCCESS if successful
-WM_E_INVAL if the passed pointer is invalid

8.62.2.7 void random_initialize_seed ( )

Initialize the random number generator's seed

The get_random_sequence() uses a random number generator that is initialized with a seed when get_random_sequence() is called for the first time. The handlers registered using random_register_seed_handler() are used to

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

generate the seed. If an applications wants to explicitly initialize The seed, this API can be used. the seed will then not be re-initialized in get_random_sequence().

8.62.2.8 uint32_t sample_initialise_random_seed ( )

Sample random seed generator

This is a sample random seed generator handler that can be registered using random_register_seed_handler() to generate a random seed. This uses a combination of DAC (generating random noise) and ADC (that internally samples the random noise) along with the flash id to generate a seed. It is recommended to register this handler and immediately call random_initialize_seed() before executing any other application code, especially if the application is going to use ADC/DAC for its own purpose.

Returns

Random seed

8.62.2.9 void get_random_sequence (unsigned char * *buf*, unsigned int *size*)

Generate random sequence of bytes

This function generates random sequence of bytes in the user provided buffer.

Parameters

out	*buf*	The buffer to be populated with random data
in	*size*	The number of bytes of the random sequence required

8.62.2.10 char* strdup (const char * *s*)

Returns a pointer to a new string which is a duplicate of the input string s. Memory for the new string is obtained allocated by the function.

It is caller's responsibility to free the memory after its use.

Parameters

in	*s*	Pointer to string to be duplicated

Returns

Pointer to newly allocated string which is duplicate of input string
NULL on error

8.62.2.11 uint32_t soft_crc32 (const void * *__data*, int *data_size*, uint32_t *crc*)

Calculate CRC32 using software algorithm

Precondition

soft_crc32_init()

soft_crc32() allows the user to calculate CRC32 values of arbitrary sized buffers across multiple calls.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Parameters

in	*__data*	Input buffer over which CRC32 is calculated.
in	*data_size*	Length of the input buffer.
in	*crc*	Previous CRC32 value used as starting point for given buffer calculation.

Returns

Calculated CRC32 value

8.62.3 Macro Documentation

8.62.3.1 #define container_of(*ptr, type, member*)

Value:

```
({                          \
        const typeof( ((type *)0)->member ) *__ptr = (ptr);     \
        (type *)( (char *)__ptr - offsetof(type, member) );})
```

Cast a member of a structure out to the containing structure

Parameters

in	*ptr*	The pointer to the member.
in	*type*	The type of the container struct this is embedded in.
in	*member*	The name of the member within the struct.

8.62.4 Typedef Documentation

8.62.4.1 typedef uint32_t(* random_hdlr_t)(void)

Function prototype for a random entropy/seed generator

Returns

a 32bit random number

8.63 wm_wlan.h File Reference

Architecture specific system include file.

8.63.1 Detailed Description

Copyright (C) 2008-2015, Marvell International Ltd. All Rights Reserved.

8.63.2 Function Documentation

8.63.2.1 int wm_wlan_init ( )

Initialize WLAN

This function is responsible for initializing only wireless.

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

Returns 0 on success, error otherwise.

8.63.2.2 void wm_wlan_deinit ( )

De Initialize WLAN

This function is responsible for deinitializing only wireless.

8.63.2.3 int wm_core_and_wlan_init (void)

Initialize SDK core components plus WLAN

This function is responsible for initializing core SDK components like uart, console, flash and wireless.

Returns

Returns 0 on success, error otherwise.

8.64 wmcrypto.h File Reference

Crypto Functions.

8.64.1 Detailed Description

This provides crypto wrapper functions that selectively call the Marvell or CyaSSL crypto functions based on the selected configuration.

8.64.2 Function Documentation

8.64.2.1 void* mrvl_dh_setup_key (u8 * *public_key*, u32 *public_len*, u8 * *private_key*, u32 *private_len*, DH_PG_PARAMS * *dh_params*)

Parameters

public_key	Pointers to public key generated
public_len	Length of public key
private_key	Pointers to private key generated randomly
private_len	Length of private key
dh_params	Parameters for DH algorithm

Returns

0 on success, -1 on failure

8.64.2.2 int mrvl_dh_compute_key (void * *dh*, u8 * *shared_key*, u32 *shared_len*, u8 * *public_key*, u32 *public_len*, u8 * *private_key*, u32 *private_len*, DH_PG_PARAMS * *dh_params*)

Parameters

dh	DH key
shared_key	Pointer to agreed shared key generated
shared_len	Length of agreed shared key

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

public_key	Pointer to public key generated
public_len	Length of public key
private_key	Pointer to private key generated randomly
private_len	Length of private key
dh_params	Parameters for DH algorithm

Returns

0 on success, -1 on failure

8.64.2.3 void mrvl_dh_free (void * *dh_context*)

Parameters

dh_context	DH key

Returns

None

8.64.2.4 u32 mrvl_sha1_vector (size_t *nmsg*, const u8 * *msg[]*, const size_t *msglen[]*, u8 * *mac*, size_t *maclen*)

Parameters

nmsg	Number of elements in the data vector
msg	Pointers to the data areas
msglen	Lengths of the data blocks
mac	Buffer for the hash
maclen	Length of hash buffer

Returns

0 on success, -1 of failure

8.64.2.5 u32 mrvl_sha256_vector (size_t *nmsg*, const u8 * *msg[]*, const size_t *msglen[]*, u8 * *mac*, size_t *maclen*)

Parameters

nmsg	Number of elements in the data vector
msg	Pointers to the data areas
msglen	Lengths of the data blocks
mac	Buffer for the hash
maclen	Length of hash buffer

Returns

0 on success, -1 on failure

8.64.2.6 void mrvl_sha256 (size_t *num_elem*, const u8 * *addr[]*, const size_t * *len*, u8 * *mac*)

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

num_elem	Number of elements in the data vector
addr	Pointers to the data areas
len	Lengths of the data blocks
mac	Buffer for the hash

Returns

0 on success, -1 on failure

8.64.2.7 u32 mrvl_hmac_sha256 (const u8 * *key*, u32 *keylen*, u8 * *msg*, u32 *msglen*, u8 * *mac*, u32 *maclen*)

Parameters

key	Key for HMAC operations
keylen	Length of the key in bytes
msg	Pointers to the data areas
msglen	Lengths of the data blocks
mac	Buffer for the hash (32 bytes)
maclen	Length of hash buffer

Returns

0 on success, -1 on failure

8.64.2.8 int mrvl_kdf (u8 * *key*, u32 *key_len*, u8 * *result*, u32 *result_len*)

Parameters

key	Input key to generate authentication key (KDK)
key_len	Length of input key
result	result buffer
result_len	Length of result buffer

Returns

0 on success, 1 otherwise

8.64.2.9 int mrvl_aes_wrap (u8 * *plain_txt*, u32 *txt_len*, u8 * *cip_txt*, u8 * *kek*, u32 *kek_len*, u8 * *iv*)

Parameters

plain_txt	Plaintext key to be wrapped
txt_len	Length of the plain key in bytes (16 bytes)
cip_txt	Wrapped key
kek	Key encryption key (KEK)
kek_len	Length of KEK in bytes (must be divisible by 16)
iv	Encryption IV for CBC mode (16 bytes)

Returns

0 on success, -1 on failure

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.64.2.10 int mrvl_aes_unwrap (u8 * *cip_txt*, u32 *txt_len*, u8 * *plain_txt*, u8 * *kek*, u32 *kek_len*, u8 * *iv*)

Parameters

cip_txt	Wrapped key to be unwrapped
txt_len	Length of the wrapped key in bytes (16 bytes)
plain_txt	Plaintext key
kek	Key encryption key (KEK)
kek_len	Length of KEK in bytes (must be divisible by 16)
iv	Encryption IV for CBC mode (16 bytes)

Returns

0 on success, -1 on failure

8.64.2.11 int mrvl_aes_wrap_ext (u8 * *plain_txt*, u32 *plain_len*, u8 * *cip_txt*, u8 * *kek*, u32 *kek_Len*, u8 * *iv*)

Parameters

plain_txt	Plaintext key to be wrapped
plain_len	Length of the plain key in bytes (16 bytes)
cip_txt	Wrapped key
kek	Key encryption key (KEK)
kek_Len	Length of KEK in bytes (must be divisible by 16)
iv	Encryption IV for CBC mode (16 bytes)

Returns

0 on success, -1 on failure

8.64.2.12 int mrvl_aes_unwrap_ext (u8 * *cip_txt*, u32 *cip_Len*, u8 * *plain_txt*, u8 * *kek*, u32 *key_len*, u8 * *iv*)

Parameters

cip_txt	Wrapped key to be unwrapped
cip_Len	Length of the wrapped key in bytes (16 bytes)
plain_txt	Plaintext key
kek	Key encryption key (KEK)
key_len	Length of KEK in bytes (must be divisible by 16)
iv	Encryption IV for CBC mode (16 bytes)

Returns

0 on success, -1 on failure

8.64.2.13 void mrvl_crypto_hmac_md5 (u8 * *input*, int *len*, u8 * *hash*, char * *hash_key*)

Parameters

input	Pointer to input data
len	Length of input data
hash	Pointer to hash
hash_key	Pointer to hash key

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Returns

None

8.64.2.14 void mrvl_crypto_md5 (u8 * *input*, int *len*, u8 * *hash*, int *hlen*)

Parameters

input	Pointer to input data
len	Length of input data
hash	Pointer to hash
hlen	Pointer to hash len

Returns

None

8.64.2.15 void mrvl_crypto_pass_to_key (char * *password*, unsigned char * *ssid*, int *ssidlength*, int *iterations*, int *count*, unsigned char * *output*)

Parameters

password	Pointer to password
ssid	Pointer to ssid
ssidlength	Length of ssid
iterations	No if iterations
count	Count
output	Pointer to output data

Returns

None

8.64.3 Macro Documentation

8.64.3.1 #define u8 uint8_t

Unsigned char

8.64.3.2 #define u32 uint32_t

Unsigned long integer

8.64.3.3 #define SHA256_DIGEST_SIZE (256 / 8)

Digest size

8.64.3.4 #define SHA256_BLOCK_SIZE (512 / 8)

Block size

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.65 wmerrno.h File Reference

Error Management.

8.65.1 Detailed Description

Copyright 2008-2015, Marvell International Ltd. All Rights Reserved.

8.65.2 Macro Documentation

8.65.2.1 #define MOD_UNUSED_3 2

Unused

8.65.2.2 #define MOD_HTTPD 3

HTTPD module index

8.65.2.3 #define MOD_AF 4

Application framework module index

8.65.2.4 #define MOD_FTFS 5

FTFS module index

8.65.2.5 #define MOD_RFGET 6

RFGET module index

8.65.2.6 #define MOD_JSON 7

JSON module index

8.65.2.7 #define MOD_TELNETD 8

TELNETD module index

8.65.2.8 #define MOD_SMDNS 9

SIMPLE MDNS module index

8.65.2.9 #define MOD_EXML 10

EXML module index

8.65.2.10 #define MOD_DHCPD 11

DHCPD module index

Confidential

MARVELL
Copyright © 2015 Marvell
Proprietary Information.
August 21, 2015 17:48

8.65.2.11 #define MOD_MDNS 12

MDNS module index

8.65.2.12 #define MOD_SYSINFO 13

SYSINFO module index

8.65.2.13 #define MOD_UNUSED_1 14

Unused module index

8.65.2.14 #define MOD_CRYPTO 15

CRYPTO module index

8.65.2.15 #define MOD_HTTPC 16

HTTP-CLIENT module index

8.65.2.16 #define MOD_PROV 17

PROVISIONING module index

8.65.2.17 #define MOD_SPI 18

SPI module index

8.65.2.18 #define MOD_PSM 19

PSM module index

8.65.2.19 #define MOD_TTCP 20

TTCP module index

8.65.2.20 #define MOD_DIAG 21

DIAGNOSTICS module index

8.65.2.21 #define MOD_UNUSED_2 22

Unused module index

8.65.2.22 #define MOD_WPS 23

WPS module index

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.65.2.23 #define MOD_WLAN 24

WLAN module index

8.65.2.24 #define MOD_USB 25

USB module index

8.65.2.25 #define MOD_WIFI 26

WIFI driver module index

8.65.2.26 #define MOD_CRIT_ERR 27

Critical error module index

8.65.2.27 #define MOD_ERR_LAST 50

Last module index .Applications can define their own modules beyond this

8.66 wmstats.h File Reference

Wireless Microcontroller statistics.

8.66.1 Enumeration Type Documentation

8.66.1.1 enum wm_reboot_reason_t

Define return values

8.67 wmstdio.h File Reference

Standard Input Output.

8.67.1 Function Documentation

8.67.1.1 int wmprintf (const char * *format*, ...)

The printf function

This function prints data to the output port. The function prototype is analogous to the printf() function of the C language.

Parameters

in	*format*	The format string

Returns

Number of bytes written to the console.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.67.1.2 int wmstdio_putchar (char * *ch*)

The printf function

This function prints character to the output port. This function should be used for printing data character by character.

Parameters

in	*ch*	Pointer to the character to written to the console

Returns

Number of bytes written to the console.

8.67.1.3 void ll_printf (const char * *format*, ...)

The low level printf function

This function prints data to the output port. This function should be used while printing from interrupts.

Parameters

in	*format*	Pointer to the character to written to the console

8.67.1.4 uint32_t wmstdio_flush (void)

Flush Standard I/O

This function flushes the send buffer to the output port.

Returns

WM_SUCCESS

8.67.1.5 uint32_t wmstdio_getchar (uint8_t * *pu8c*)

Read one byte

Parameters

pu8c	Pointer to the location to store the byte read.

Returns

Number of bytes read from console port.

8.67.1.6 int wmstdio_getconsole_port (int * *port_id*)

Returns port id used for console

Parameters

out	*port_id*	pointer to port_id

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Returns

WM_SUCCESS

8.67.1.7 int wmstdio_init (UART_ID_Type *uart_port,* int *baud*)

Initialize the standard input output facility

Note

This function is called by the wm_core_init function. Applications need not explicitly call this function if they have already called wm_core_init().

Parameters

in	*uart_port*	UART Port to be used as the console (e.g. UART0_ID, UART1_ID etc.
in	*baud*	Baud rate to be used on the port. 0 implies the default baud rate

Returns

WM_SUCCESS on success, error code otherwise.

8.67.2 Macro Documentation

8.67.2.1 #define MAX_MSG_LEN 127

Maximum number of characters that can be printed using wmprintf

8.68 wmsysinfo.h File Reference

System Info Utilities Here is the sysinfo usage: sysinfo [options] service name [command] where service name can be one of the following: [thread|socket|netstat|heap|stats|mdns|-h].

8.68.1 Detailed Description

sysinfo is a diagnostic utility

The following services are supported:

netstat: Dump info about network stack. It can be used as follows: sysinfo netstat link/ip/tcp/udp Note: All available info about network stack is displayed if no argument is passed to netstat

thread: Dump info about operating system threads. In addition to dumping info, the 'stackmark' command writes a pattern to the unused portion of stacks' threads. After marking the stacks, dumping the info will report the highwater mark of the stack. This is useful for detecting stack overruns.

socket: Dump statistics of socket descriptor usage including available sockets, used sockets, tcp sockets etc

stats: Dump statistics related to the WMSDK.

heap: Dump the heap metadata in this following format HST:next-free-block:prev-block:block-size:block-type HST : A tag to seperate other logs from this log next-free-block : The address of the block which is free. This list is sorted by block size - lowest size block first prev-block : Address of the previous block in the serial list block-size : Size of the current block block-type : The type of the block: X: First block. This is not a part of the heap and points to first real block in this heap F: Free block A: Allocated block

Options: -h Print this message -n name Only operate on the instance of the service with this name. E.g., 'sysinfo -n foo thread' will only dump info for the foo thread.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.68.2 Function Documentation

8.68.2.1 int sysinfo_init (void)

Initialize System Information Utility

This function registers system info CLI commands like 'sysinfo', 'memdump' and 'memwrite' that can be used to query information about the various susbsystems.

Returns

WM_SUCCESS on success, error otherwise

8.69 wmtime.h File Reference

Time Management Subsystem.

8.69.1 Detailed Description

This module provides the time managment APIs. Internally wireless microcontroller manages time in terms of seconds since epoch (1 Jan 1970 00:00). This representation is known as "posix time". These routines can be used to do conversions between human readable time and posix time as well as to set the system time to desired value.

8.69.2 Function Documentation

8.69.2.1 time_t http_date_to_time (const char * *date*)

Convert HTTP date format to POSIX time format

Parameters

in	*date*	HTTP date format

Returns

success or failure as: -WM_FAIL: Conversion failed. Invalid format/data else valid time_t value

8.69.2.2 int wmtime_time_set (const struct tm * *tm*)

Set the date and time

Parameters

in	*tm*	The rtc value is updated with the values in tm structure

Returns

success or failure as: 0: Success -1: Failed validation of tm structure

8.69.2.3 int wmtime_time_get (struct tm * *tm*)

Get date and time

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

out	*tm*	tm structure is updated to get the current value in rtc

Returns

success or failure as: 0: Success non-zero: Internal error

8.69.2.4 int wmtime_time_set_posix (time_t *time*)

Set the date and time using posix time

Parameters

in	*time*	The rtc value is updated with the value present in time

Returns

success or failure as: 0: Success non-zero: Internal error

8.69.2.5 time_t wmtime_time_get_posix (void)

Get date and time in posix format

Returns

time_t value from RTC

8.69.2.6 struct tm* gmtime_r (const time_t * *time*, struct tm * *result*)

Convert to tm structure from POSIX/Unix time (Seconds since epoch)

Parameters

in	*time*	This is POSIX time that is to be converted into tm
out	*result*	This should point to pre-allocated tm instance

Returns

pointer to struct tm; NULL in case of error

8.69.2.7 time_t mktime (struct tm * *tm*)

Converts to POSIX/Unix time from tm structure

Parameters

in	*tm*	This is tm instance that is to be converted into time_t format

Returns

time_t POSIX/Unix time equivalent

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

8.69.2.8 char∗ asctime (const struct tm ∗ *tm*)

Converts the broken-down time value tm into a null-terminated string.

Parameters

in	*tm*	This is tm instance that is to be converted string.

Returns

Pointer to a statically allocated string which contains the date and time format as follows "Tue Mar 24 09:20:14 2015". The statically allocated string might be overwritten by subsequent calls to any of the date and time functions.

8.69.2.9 int wmtime_init (void)

Initialize time subsystem including RTC. Sets system time to 1/1/1970 00:00 (i.e. epoch 0)

Returns

WM_SUCCESS on success, zero otherwise

Note

This function is called by the wm_core_init function. Applications need not explicitly call this function if they have already called wm_core_init().

8.69.2.10 int wmtime_cli_init (void)

Register wmtime cli commands for the application use

Returns

success or failure as: 0: Success non-zero: Failure

Note

This function can only be used by the application if wm_core_and_wlan_init() function is called

8.70 wps.h File Reference

WPS - WiFi Protected Setup.

8.70.1 Detailed Description

Wi-Fi Protected Setup (WPS) is a standard for easy and secure wireless network set up and connections. Using this standard, wireless clients can associate with the WPS enabled access point by entering the same PIN at client and AP end or by pushing a pushbutton on both. This eliminates the need for manually entering the security configuration on the client side.

Two methods that are supported by SDK are:

- PIN Method: In this method the same PIN (Personal Identification Number) is entered on the wireless client and access point. The PIN can be static or dynamically generated on any of the AP or wireless client.
- PBC Method: in which the user simply has to push a button, either an actual or virtual one, on both the AP and wireless client.

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

8.70.2 Usage

In a typical implementation WPS thread is started by calling wps_start. A valid initialized wps_config structure needs to be passed to this function. This structure contains a callback handler which is invoked by the WPS thread on occurrence of various events. Once the WPS thread is started, commands can be sent to it using wps_connect function. Typically these commands can be start PIN session, or start a pushbutton session. This requires passing wps_session_command structure that contains a valid wlan_scan_result that contains network information for a WPS enabled wireless network. On successful WPS session, callback handler is invoked with WPS_SESSION_SUCCESSFUL event with wlan_network structure that contains all the security information for the network with which WPS session was attempted. After this wps_stop should be called to terminate the WPS thread.

Note

WPS implementation lets application decide PIN policy. PIN could either be generated by the application, or by the access point or can be statically generated at device manufacturing time. wps_generate_pin and wps_validate_pin utility functions can be used by the application to generate and validate PIN.
WPS thread internally uses multiple dynamic allocations. Please do not reduce the heap size below 64KB if the application wishes to use WPS.

8.70.3 Function Documentation

8.70.3.1 int wps_start (struct wps_config * *wps_conf*)

Starts WPS thread and enables commands delivery

Parameters

wps_conf	A pointer to WPS custom configuration

Returns

WM_SUCCESS if successful, -WM_FAIL otherwise

8.70.3.2 int wps_connect (enum wps_session_command *pbc*, uint32_t *pin*, struct wlan_scan_result * *res*)

Connect to WPS enabled AP

Connect to a WPS enabled AP. This function is typically called whenever the users pushes the wps button or enters wps pin.

Parameters

pbc	Set to 1 if Push-button session is desired
pin	Ignore if pbc is 1. If pbc is 0, this indicates the pin that should be used. The pin can be 4 or 8 digits.
res	The WPS enabled AP to connect to.

Returns

WM_SUCCESS if successful, -WM_FAIL otherwise

8.70.3.3 int wps_generate_pin (uint32_t * *wps_pin*)

Generate 8 digit WPS PIN value with random number generator

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

Parameters

wps_pin	Generated 8 digit WPS PIN value

Returns

WM_SUCCESS if successful, -WM_FAIL otherwise

8.70.3.4 int wps_validate_pin (uint32_t *wps_pin*)

Validate checksum of PIN

Parameters

wps_pin	WPS PIN value

Returns

WM_SUCCESS if successful, -WM_FAIL otherwise

8.70.3.5 int wps_stop ( )

Deletes WPS thread and message queue

Returns

WM_SUCCESS if successful, -WM_FAIL otherwise

8.70.4 Macro Documentation

8.70.4.1 #define WPS_LOG(...) wmlog("wps", ##__VA_ARGS__)

wps log

8.70.5 Enumeration Type Documentation

8.70.5.1 enum wps_session_command

enum : WPS session commands

Enumerator

CMD_WPS_PIN Command to start WPS PIN session

CMD_WPS_PBC Command to start WPS PBC session

8.70.5.2 enum wps_event

enum : WPS events

Enumerator

WPS_STARTED WPS thread started

WPS_SESSION_STARTED WPS PBC/PIN Session started

WPS_SESSION_PIN_CHKSUM_FAILED WPS PIN checksum failed

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

WPS_SESSION_ABORTED WPS Session aborted

WPS_SESSION_TIMEOUT WPS Session registration timeout

WPS_SESSION_SUCCESSFUL WPS Session attempt successful

WPS_SESSION_FAILED WPS Session failed

WPS_FINISHED WPS thread stopped

8.70.5.3 enum wps_session_types

Enum that indicates type of WPS session either a push button or a PIN based session is determined by value fo this enum

Enumerator

WPS_SESSION_INACTIVE WPS session is not active

WPS_SESSION_PBC WPS Push Button session active

WPS_SESSION_PIN WPS PIN session active

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

Index

_wifi_cal_data_t, 31
    data, 31
    data_len, 31
_wifi_mgmt_frame_tx, 31
    addr1, 32
    addr2, 32
    addr3, 32
    addr4, 32
    duration_id, 32
    frm_ctl, 32
    frm_len, 32
    payload, 32
    seq_ctl, 32

ACTION_ENTER_PM2
    pwrmgr.h, 301
ACTION_ENTER_PM3
    pwrmgr.h, 301
ACTION_ENTER_PM4
    pwrmgr.h, 301
ACTION_EXIT_PM2
    pwrmgr.h, 301
ACTION_EXIT_PM3
    pwrmgr.h, 301
ADDR_TYPE_DHCP
    wlan.h, 344
ADDR_TYPE_LLA
    wlan.h, 344
ADDR_TYPE_STATIC
    wlan.h, 344
AF_EVT_INIT_DONE
    app_framework.h, 101
AF_EVT_NORMAL_CHAN_SWITCH
    app_framework.h, 102
AF_EVT_NORMAL_CONNECT_FAILED
    app_framework.h, 102
AF_EVT_NORMAL_CONNECTED
    app_framework.h, 102
AF_EVT_NORMAL_CONNECTING
    app_framework.h, 102
AF_EVT_NORMAL_DHCP_RENEW
    app_framework.h, 102
AF_EVT_NORMAL_INIT
    app_framework.h, 102
AF_EVT_NORMAL_LINK_LOST
    app_framework.h, 102
AF_EVT_NORMAL_PRE_RESET_PROV
    app_framework.h, 102
AF_EVT_NORMAL_RESET_PROV
    app_framework.h, 102
AF_EVT_NORMAL_USER_DISCONNECT
    app_framework.h, 102
AF_EVT_NORMAL_WPS_DISCONNECT
    app_framework.h, 102
AF_EVT_NW_SET
    app_framework.h, 102
AF_EVT_P2P_FAILED
    app_framework.h, 103
AF_EVT_P2P_FINISHED
    app_framework.h, 103
AF_EVT_P2P_ROLE_NEGOTIATED
    app_framework.h, 103
AF_EVT_P2P_SESSION_STARTED
    app_framework.h, 103
AF_EVT_P2P_STARTED
    app_framework.h, 103
AF_EVT_PROV_CLIENT_DONE
    app_framework.h, 103
AF_EVT_PROV_DONE
    app_framework.h, 102
AF_EVT_PROV_WPS_REQ_TIMEOUT
    app_framework.h, 102
AF_EVT_PROV_WPS_SSID_SELECT_REQ
    app_framework.h, 102
AF_EVT_PROV_WPS_START
    app_framework.h, 102
AF_EVT_PROV_WPS_SUCCESSFUL
    app_framework.h, 102
AF_EVT_PROV_WPS_UNSUCCESSFUL
    app_framework.h, 102
AF_EVT_PS_ENTER
    app_framework.h, 102
AF_EVT_PS_EXIT
    app_framework.h, 102
AF_EVT_UAP_CLIENT_ASSOC
    app_framework.h, 102
AF_EVT_UAP_CLIENT_DISSOC
    app_framework.h, 102
AF_EVT_UAP_STARTED
    app_framework.h, 102
AF_EVT_UAP_STOPPED
    app_framework.h, 102
AF_EVT_WLAN_INIT_DONE
    app_framework.h, 101
APP_CTRL_EVT_INTERNAL
    app_framework.h, 103
AUTH
    wlan.h, 344
AUTH_FAILED

Proprietary Information.
August 21, 2015 17:48

Copyright © 2015 Marvell

app_framework.h, 101
AES_KEY_LEN
fw_upgrade_rsa_aes.h, 131
about_to_die
healthmon_handler, 37
acomp_drv_close
mdev_acomp.h, 190
acomp_drv_deinit
mdev_acomp.h, 190
acomp_drv_init
mdev_acomp.h, 189
acomp_drv_open
mdev_acomp.h, 189
acomp_drv_result
mdev_acomp.h, 189
adc_drv_close
mdev_adc.h, 193
adc_drv_deinit
mdev_adc.h, 193
adc_drv_get_samples
mdev_adc.h, 193
adc_drv_init
mdev_adc.h, 191
adc_drv_open
mdev_adc.h, 192
adc_drv_result
mdev_adc.h, 192
adc_drv_timeout
mdev_adc.h, 192
adc_get_config
mdev_adc.h, 191
adc_modify_default_config
mdev_adc.h, 191
addr
update_desc, 66
addr1
_wifi_mgmt_frame_tx, 32
addr2
_wifi_mgmt_frame_tx, 32
addr3
_wifi_mgmt_frame_tx, 32
addr4
_wifi_mgmt_frame_tx, 32
addr_type
ipv4_config, 47
address
ipv4_config, 47
aes_ctr_final
mdev_aes.h, 197
aes_drv_ccm_decrypt
mdev_aes.h, 197
aes_drv_ccm_encrypt
mdev_aes.h, 196
aes_drv_ccm_setkey
mdev_aes.h, 195
aes_drv_close
mdev_aes.h, 198
aes_drv_decrypt
mdev_aes.h, 197
aes_drv_encrypt
mdev_aes.h, 196
aes_drv_init
mdev_aes.h, 195
aes_drv_open
mdev_aes.h, 195
aes_drv_setkey
mdev_aes.h, 195
agent
httpd_request_t, 42
app_framework.h
AF_EVT_INIT_DONE, 101
AF_EVT_NORMAL_CHAN_SWITCH, 102
AF_EVT_NORMAL_CONNECT_FAILED, 102
AF_EVT_NORMAL_CONNECTED, 102
AF_EVT_NORMAL_CONNECTING, 102
AF_EVT_NORMAL_DHCP_RENEW, 102
AF_EVT_NORMAL_INIT, 102
AF_EVT_NORMAL_LINK_LOST, 102
AF_EVT_NORMAL_PRE_RESET_PROV, 102
AF_EVT_NORMAL_RESET_PROV, 102
AF_EVT_NORMAL_USER_DISCONNECT, 102
AF_EVT_NORMAL_WPS_DISCONNECT, 102
AF_EVT_NW_SET, 102
AF_EVT_P2P_FAILED, 103
AF_EVT_P2P_FINISHED, 103
AF_EVT_P2P_ROLE_NEGOTIATED, 103
AF_EVT_P2P_SESSION_STARTED, 103
AF_EVT_P2P_STARTED, 103
AF_EVT_PROV_CLIENT_DONE, 103
AF_EVT_PROV_DONE, 102
AF_EVT_PROV_WPS_REQ_TIMEOUT, 102
AF_EVT_PROV_WPS_SSID_SELECT_REQ, 102
AF_EVT_PROV_WPS_START, 102
AF_EVT_PROV_WPS_SUCCESSFUL, 102
AF_EVT_PROV_WPS_UNSUCCESSFUL, 102
AF_EVT_PS_ENTER, 102
AF_EVT_PS_EXIT, 102
AF_EVT_UAP_CLIENT_ASSOC, 102
AF_EVT_UAP_CLIENT_DISSOC, 102
AF_EVT_UAP_STARTED, 102
AF_EVT_UAP_STOPPED, 102
AF_EVT_WLAN_INIT_DONE, 101
APP_CTRL_EVT_INTERNAL, 103
AUTH_FAILED, 101
CONN_STATE_CONNECTED, 101
CONN_STATE_CONNECTING, 101
CONN_STATE_NOTCONNECTED, 101
DHCP_FAILED, 101
NETWORK_NOT_FOUND, 101
P2P_CONNECT_FAILED, 101
ROLE_AUTO_GO, 103
ROLE_CLIENT, 103
ROLE_GO, 103
ROLE_UNDECIDED, 103
WM_E_AF_NW_ADD, 101
WM_E_AF_NW_CONN, 101

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

WM_E_AF_NW_DEL, 101
WM_E_AF_NW_RD, 101
WM_E_AF_NW_WR, 101
WM_E_AF_PSM_INIT, 101
WM_E_AF_REB_TMR_WR, 101
WM_E_AF_WD_INIT, 101
WM_E_AF_WLCMGR_FAIL, 101
app_configure_uap_ip
app_framework.h, 83
app_conn_failure_reason_t
app_framework.h, 101
app_conn_status_t
app_framework.h, 101
app_ctrl_event_name
app_framework.h, 99
app_ctrl_event_t
app_framework.h, 101
app_ctrl_getmode
app_framework.h, 100
app_ctrl_notify_application_event
app_framework.h, 97
app_event_handler_t
app_framework.h, 100
app_ezconnect_provisioning_start
app_framework.h, 89
app_ezconnect_provisioning_stop
app_framework.h, 90
app_framework.h, 81
app_configure_uap_ip, 83
app_conn_failure_reason_t, 101
app_conn_status_t, 101
app_ctrl_event_name, 99
app_ctrl_event_t, 101
app_ctrl_getmode, 100
app_ctrl_notify_application_event, 97
app_event_handler_t, 100
app_ezconnect_provisioning_start, 89
app_ezconnect_provisioning_stop, 90
app_framework_start, 83
app_ftfs_init, 90
app_get_connection_status, 99
app_get_device_name, 95
app_httpd_only_start, 91
app_httpd_start, 91
app_httpd_stop, 92
app_httpd_with_fs_start, 90
app_init_device_name, 95
app_mdns_add_service, 93
app_mdns_add_service_arr, 93
app_mdns_iface_state_change, 94
app_mdns_refresh_hostname, 93
app_mdns_remove_service, 93
app_mdns_remove_service_all, 94
app_mdns_remove_service_arr, 94
app_mdns_start, 92
app_mdns_stop, 92
app_network_ip_get, 96
app_p2p_disconnect, 98
app_p2p_ip_get, 99
app_p2p_role_data_t, 100
app_p2p_role_t, 103
app_p2p_scan, 98
app_p2p_session_start, 98
app_p2p_start, 97
app_p2p_stop, 98
app_prov_wps_session_start, 89
app_provisioning_only_start, 89
app_provisioning_only_stop, 90
app_provisioning_start, 88
app_provisioning_stop, 90
app_psm_hnd, 103
app_psm_init, 99
app_reboot, 96
app_reset_saved_network, 88
app_set_device_name, 96
app_sta_save_network, 87
app_sta_save_network_and_start, 87
app_sta_start, 86
app_sta_start_by_network, 87
app_sta_stop, 87
app_sys_get_uuid, 96
app_sys_http_update_all, 97
app_sys_register_diag_handler, 91
app_sys_register_perf_handler, 92
app_sys_register_upgrade_handler, 91
app_uap_start, 85
app_uap_start_by_network, 86
app_uap_start_on_channel, 85
app_uap_start_on_channel_with_dhcp, 84
app_uap_start_with_dhcp, 84
app_uap_stop, 86
wm_af_errno, 101
app_framework_start
app_framework.h, 83
app_ftfs_init
app_framework.h, 90
app_get_connection_status
app_framework.h, 99
app_get_device_name
app_framework.h, 95
app_httpd_only_start
app_framework.h, 91
app_httpd_start
app_framework.h, 91
app_httpd_stop
app_framework.h, 92
app_httpd_with_fs_start
app_framework.h, 90
app_init_device_name
app_framework.h, 95
app_init_state, 33
backup_fw, 33
factory_reset, 33
rst_cause, 33
app_mdns_add_service
app_framework.h, 93

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

app_mdns_add_service_arr
app_framework.h, 93
app_mdns_iface_state_change
app_framework.h, 94
app_mdns_refresh_hostname
app_framework.h, 93
app_mdns_remove_service
app_framework.h, 93
app_mdns_remove_service_all
app_framework.h, 94
app_mdns_remove_service_arr
app_framework.h, 94
app_mdns_start
app_framework.h, 92
app_mdns_stop
app_framework.h, 92
app_network_ip_get
app_framework.h, 96
app_p2p_disconnect
app_framework.h, 98
app_p2p_ip_get
app_framework.h, 99
app_p2p_role_data, 33
data, 34
role, 34
app_p2p_role_data_t
app_framework.h, 100
app_p2p_role_t
app_framework.h, 103
app_p2p_scan
app_framework.h, 98
app_p2p_session_start
app_framework.h, 98
app_p2p_start
app_framework.h, 97
app_p2p_stop
app_framework.h, 98
app_prov_wps_session_start
app_framework.h, 89
app_provisioning_only_start
app_framework.h, 89
app_provisioning_only_stop
app_framework.h, 90
app_provisioning_start
app_framework.h, 88
app_provisioning_stop
app_framework.h, 90
app_psm_hnd
app_framework.h, 103
app_psm_init
app_framework.h, 99
app_reboot
app_framework.h, 96
app_reset_saved_network
app_framework.h, 88
app_set_device_name
app_framework.h, 96
app_sta_save_network
app_framework.h, 87
app_sta_save_network_and_start
app_framework.h, 87
app_sta_start
app_framework.h, 86
app_sta_start_by_network
app_framework.h, 87
app_sta_stop
app_framework.h, 87
app_sys_get_uuid
app_framework.h, 96
app_sys_http_update_all
app_framework.h, 97
app_sys_register_diag_handler
app_framework.h, 91
app_sys_register_perf_handler
app_framework.h, 92
app_sys_register_upgrade_handler
app_framework.h, 91
app_uap_start
app_framework.h, 85
app_uap_start_by_network
app_framework.h, 86
app_uap_start_on_channel
app_framework.h, 85
app_uap_start_on_channel_with_dhcp
app_framework.h, 84
app_uap_start_with_dhcp
app_framework.h, 84
app_uap_stop
app_framework.h, 86
arch_pm_init
mdev_pm.h, 229
arch_reboot
mdev_pm.h, 229
asctime
wmtime.h, 391
auth_info
wlan.h, 346
auto_go
p2p_config, 55

BEACON
wlan.h, 344
BIT10
mdev_i2c.h, 223
BSS_TYPE_STA
wifi-decl.h, 309
BSS_TYPE_UAP
wifi-decl.h, 309
ba_block_allocate
block_alloc.h, 105
ba_block_release
block_alloc.h, 105
ba_cli_init
block_alloc.h, 105
ba_pool_create
block_alloc.h, 104
ba_pool_delete

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

block_alloc.h, 104
backup_fw
app_init_state, 33
beacon_info
wlan.h, 345
beacon_interval
wlan.h, 345
bin2hex
wm_utils.h, 376
block_alloc.h, 103
ba_block_allocate, 105
ba_block_release, 105
ba_cli_init, 105
ba_pool_create, 104
ba_pool_delete, 104
board.h, 106
board_32k_osc, 107
board_32k_xtal, 106
board_button_1, 111
board_button_2, 111
board_button_3, 108
board_button_pressed, 108
board_card_detect, 107
board_cpu_freq, 106
board_gpio_power_on, 108
board_i2c_pin_config, 109
board_lcd_backlight_off, 111
board_lcd_backlight_on, 111
board_lcd_reset, 111
board_led_1, 109
board_led_2, 109
board_led_3, 110
board_led_4, 110
board_led_off, 110
board_led_on, 110
board_main_osc, 106
board_main_xtal, 106
board_rc32k_calib, 107
board_sdio_pdn, 107
board_sdio_pdn_support, 108
board_sdio_pin_config, 109
board_sdio_pwr, 107
board_sdio_reset, 107
board_ssp_pin_config, 109
board_uart_pin_config, 108
board_usb_pin_config, 109
board_wakeup0_functional, 111
board_wakeup0_wifi, 111
board_wakeup1_functional, 111
board_wakeup1_wifi, 112
board_32k_osc
board.h, 107
board_32k_xtal
board.h, 106
board_button_1
board.h, 111
board_button_2
board.h, 111
board_button_3
board.h, 108
board_button_pressed
board.h, 108
board_card_detect
board.h, 107
board_cpu_freq
board.h, 106
board_gpio_power_on
board.h, 108
board_i2c_pin_config
board.h, 109
board_lcd_backlight_off
board.h, 111
board_lcd_backlight_on
board.h, 111
board_lcd_reset
board.h, 111
board_led_1
board.h, 109
board_led_2
board.h, 109
board_led_3
board.h, 110
board_led_4
board.h, 110
board_led_off
board.h, 110
board_led_on
board.h, 110
board_main_osc
board.h, 106
board_main_xtal
board.h, 106
board_rc32k_calib
board.h, 107
board_sdio_pdn
board.h, 107
board_sdio_pdn_support
board.h, 108
board_sdio_pin_config
board.h, 109
board_sdio_pwr
board.h, 107
board_sdio_reset
board.h, 107
board_ssp_pin_config
board.h, 109
board_uart_pin_config
board.h, 108
board_usb_pin_config
board.h, 109
board_wakeup0_functional
board.h, 111
board_wakeup0_wifi
board.h, 111
board_wakeup1_functional
board.h, 111

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

board_wakeup1_wifi
board.h, 112
body_nbytes
httpd_request_t, 42
bssid
wlan.h, 345
wlan_network, 72
wlan_scan_result, 75
bssid_specific
wlan_network, 73
bytes_to_read
upgrade_struct_t, 69

CMD_P2P_PBC
p2p.h, 273
CMD_P2P_PIN
p2p.h, 273
CMD_WPS_PBC
wps.h, 394
CMD_WPS_PIN
wps.h, 394
CONN_STATE_CONNECTED
app_framework.h, 101
CONN_STATE_CONNECTING
app_framework.h, 101
CONN_STATE_NOTCONNECTED
app_framework.h, 101
COUNTRY_AU
wlan_11d.h, 347
COUNTRY_CA
wlan_11d.h, 347
COUNTRY_CN
wlan_11d.h, 347
COUNTRY_EU
wlan_11d.h, 347
COUNTRY_FR
wlan_11d.h, 347
COUNTRY_JP
wlan_11d.h, 347
COUNTRY_KR
wlan_11d.h, 347
COUNTRY_SG
wlan_11d.h, 347
COUNTRY_US
wlan_11d.h, 347
ca_cert
tls_client_t, 62
tls_init_config_t, 64
ca_cert_size
tls_client_t, 62
tls_init_config_t, 64
cap_info
wlan.h, 345
cb_fn
wm_os.h, 374
chan_desc
wifi-decl.h, 310
chan_num
wifi-decl.h, 310
chan_trpc_config
wifi-decl.h, 310
chan_trpc_config_t
wifi-decl.h, 309
chan_trpc_entry
wifi-decl.h, 310
chan_trpc_t
wifi-decl.h, 309
chan_width
wifi-decl.h, 310
channel
wlan_network, 72
wlan_scan_result, 75
channel_specific
wlan_network, 73
check_interval
healthmon_handler, 37
chunked
http_resp_t, 40
httpd_request_t, 42
cli.h, 112
cli_init, 113
cli_register_command, 112
cli_register_commands, 113
cli_stop, 113
cli_unregister_command, 112
cli_unregister_commands, 113
cli_command, 34
function, 34
help, 34
name, 34
cli_init
cli.h, 113
cli_register_command
cli.h, 112
cli_register_commands
cli.h, 113
cli_stop
cli.h, 113
cli_unregister_command
cli.h, 112
cli_unregister_commands
cli.h, 113
client
tls_init_config_t, 64
client_cert
httpd_tls_certs_t, 44
tls_client_t, 62
tls_init_config_t, 64
client_cert_is_chained
tls_client_t, 62
client_cert_size
httpd_tls_certs_t, 44
tls_client_t, 63
tls_init_config_t, 64
client_key
tls_client_t, 63
tls_init_config_t, 64

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

client_key_size
tls_client_t, 63
tls_init_config_t, 64
client_mode_secure_upgrade
secure_upgrade.h, 304
config_methods
p2p_config, 56
wps_config, 77
consecutive_failures
healthmon_handler, 37
container_of
wm_utils.h, 379
content
http_req_t, 38
content_encoding
http_resp_t, 40
content_len
http_req_t, 38
content_length
http_resp_t, 40
content_length_field_present
http_resp_t, 40
content_type
http_resp_t, 40
httpd_request_t, 42
country_code
wifi_domain_param_t, 69
country_code_t
wlan_11d.h, 347
crc
partition_table, 59
crc_drv_block_calc
mdev_crc.h, 200
crc_drv_close
mdev_crc.h, 201
crc_drv_init
mdev_crc.h, 200
crc_drv_open
mdev_crc.h, 200
crc_drv_stream_feed
mdev_crc.h, 201
crc_drv_stream_get_result
mdev_crc.h, 201
crc_drv_stream_setlen
mdev_crc.h, 201
crc_mode_t
mdev_crc.h, 202
ctaocrypt_lib_init
wm-tls.h, 348
cur
jobj_t, 48
Custom Wi-Fi Region Configuration, 17

DATA
wlan.h, 344
DHCP_FAILED
app_framework.h, 101
DMA_DISABLE
mdev_ssp.h, 244
DMA_ENABLE
mdev_ssp.h, 244
DOWN
mdns.h, 268
DEFAULT_RETRY_COUNT
httpc.h, 144
DH_PG_PARAMS, 35
generator, 35
generatorLen, 35
prime, 35
primeLen, 35
dac_drv_close
mdev_dac.h, 204
dac_drv_deinit
mdev_dac.h, 204
dac_drv_init
mdev_dac.h, 203
dac_drv_open
mdev_dac.h, 203
dac_drv_output
mdev_dac.h, 204
dac_modify_default_config
mdev_dac.h, 203
data
_wifi_cal_data_t, 31
app_p2p_role_data, 34
wslay_frame, 79
data_fetch
Middle level RFGET APIs, 23
data_info
wlan.h, 346
data_len
_wifi_cal_data_t, 31
wslay_frame, 79
data_validate
Middle level RFGET APIs, 23
decrypt
upgrade_struct_t, 67
decrypt_ctx
upgrade_struct_t, 68
decrypt_func_t
secure_upgrade.h, 305
decrypt_init
upgrade_struct_t, 67
decrypt_init_func_t
secure_upgrade.h, 304
decrypt_key
upgrade_struct_t, 69
delete_handler
httpd_wsgi_call, 46
dest
wlan.h, 344
dev
update_desc, 66
device
partition_entry, 58
device_address
p2p_scan_result, 57

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

device_id
p2p_scan_result, 57
device_name
p2p_config, 55
p2p_scan_result, 57
wps_config, 77
dhcp-server.h
WM_E_DHCPD_ARP_SEND, 115
WM_E_DHCPD_BCAST_ADDR, 115
WM_E_DHCPD_BUFFER_FULL, 115
WM_E_DHCPD_DNS_IGNORE, 115
WM_E_DHCPD_INCORRECT_HEADER, 115
WM_E_DHCPD_INIT, 115
WM_E_DHCPD_INVALID_INPUT, 115
WM_E_DHCPD_INVALID_OPCODE, 115
WM_E_DHCPD_IOCTL_CALL, 115
WM_E_DHCPD_IP_ADDR, 115
WM_E_DHCPD_MUTEX_CREATE, 115
WM_E_DHCPD_NETMASK, 115
WM_E_DHCPD_REGISTER_CMDS, 115
WM_E_DHCPD_RESP_SEND, 115
WM_E_DHCPD_SERVER_RUNNING, 115
WM_E_DHCPD_SOCKET, 115
WM_E_DHCPD_SPOOF_NAME, 115
WM_E_DHCPD_THREAD_CREATE, 115
dhcp-server.h, 114
dhcp_get_ip_from_mac, 115
dhcp_server_lease_timeout, 114
dhcp_server_start, 114
dhcp_server_stop, 114
dhcpd_cli_init, 114
wm_dhcpd_errno, 115
dhcp_get_ip_from_mac
dhcp-server.h, 115
dhcp_server_lease_timeout
dhcp-server.h, 114
dhcp_server_start
dhcp-server.h, 114
dhcp_server_stop
dhcp-server.h, 114
dhcpd_cli_init
dhcp-server.h, 114
dma_cfg
dma_config_type, 36
dma_config_type, 35
dma_cfg, 36
perDmaInter, 36
dma_drv_close
mdev_dma.h, 207
dma_drv_deinit
mdev_dma.h, 208
dma_drv_init
mdev_dma.h, 205
dma_drv_open
mdev_dma.h, 205
dma_drv_set_cb
mdev_dma.h, 206
dma_drv_transfer
mdev_dma.h, 206
dma_drv_wait_for_transfer_complete
mdev_dma.h, 207
dns1
ipv4_config, 47
dns2
ipv4_config, 47
domain
mdns_service, 50
duration
wlan.h, 344
duration_id
_wifi_mgmt_frame_tx, 32

EGPIO0
mdev_pm.h, 231
EGPIO1
mdev_pm.h, 231
ERTC
mdev_pm.h, 231
EWLAN
mdev_pm.h, 231
ed_chacha_client_mode_upgrade_fw
fw_upgrade_ed_chacha.h, 128
ed_chacha_server_mode_upgrade_fw
fw_upgrade_ed_chacha.h, 128
end
jsontok_t, 49
etag_val
httpd_request_t, 42
ezxml.h
WM_E_EXML_INBUF, 122
WM_E_EXML_INLEN, 122
ezxml.h, 116
ezxml_add_child, 119
ezxml_add_child_d, 121
ezxml_attr, 117
ezxml_child, 117
ezxml_cut, 120
ezxml_error, 118
ezxml_free, 118
ezxml_get, 117
ezxml_idx, 117
ezxml_insert, 120
ezxml_move, 121
ezxml_name, 120
ezxml_new, 118
ezxml_new_d, 121
ezxml_next, 120
ezxml_parse_str, 116
ezxml_pi, 118
ezxml_remove, 121
ezxml_set_attr, 119
ezxml_set_attr_d, 121
ezxml_set_flag, 119
ezxml_set_txt, 119
ezxml_set_txt_d, 121
ezxml_toxml, 118
ezxml_txt, 120

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

wm_exml_errno, 121
ezxml_add_child
ezxml.h, 119
ezxml_add_child_d
ezxml.h, 121
ezxml_attr
ezxml.h, 117
ezxml_child
ezxml.h, 117
ezxml_cut
ezxml.h, 120
ezxml_error
ezxml.h, 118
ezxml_free
ezxml.h, 118
ezxml_get
ezxml.h, 117
ezxml_idx
ezxml.h, 117
ezxml_insert
ezxml.h, 120
ezxml_move
ezxml.h, 121
ezxml_name
ezxml.h, 120
ezxml_new
ezxml.h, 118
ezxml_new_d
ezxml.h, 121
ezxml_next
ezxml.h, 120
ezxml_parse_str
ezxml.h, 116
ezxml_pi
ezxml.h, 118
ezxml_remove
ezxml.h, 121
ezxml_set_attr
ezxml.h, 119
ezxml_set_attr_d
ezxml.h, 121
ezxml_set_flag
ezxml.h, 119
ezxml_set_txt
ezxml.h, 119
ezxml_set_txt_d
ezxml.h, 121
ezxml_toxml
ezxml.h, 118
ezxml_txt
ezxml.h, 120

FC_COMP_BOOT2
partition.h, 275
FC_COMP_FTFS
partition.h, 276
FC_COMP_FW
partition.h, 275
FC_COMP_PSM
partition.h, 276
FC_COMP_USER_APP
partition.h, 276
FC_COMP_WLAN_FW
partition.h, 275
FLOW_CONTROL_HW
mdev_uart.h, 250
FLOW_CONTROL_NONE
mdev_uart.h, 250
FLOW_CONTROL_SW
mdev_uart.h, 250
factory_reset
app_init_state, 33
filename
httpd_request_t, 42
fin
wslay_frame, 78
first_chan
wifi_sub_band_set_t, 70
fl_dev
flash_desc, 36
fl_size
flash_desc, 36
fl_start
flash_desc, 36
flags
httpc_cfg_t, 41
mdns_service, 51
psm_objattrib_t, 62
tls_init_config_t, 64
flash.h, 122
flash_desc_t, 125
flash_drv_close, 123
flash_drv_erase, 124
flash_drv_eraseall, 124
flash_drv_get_id, 124
flash_drv_init, 123
flash_drv_open, 123
flash_drv_read, 123
flash_drv_write, 123
flash_comp
partition.h, 275
flash_desc, 36
fl_dev, 36
fl_size, 36
fl_start, 36
flash_desc_t
flash.h, 125
flash_drv_close
flash.h, 123
flash_drv_erase
flash.h, 124
flash_drv_eraseall
flash.h, 124
flash_drv_get_id
flash.h, 124
flash_drv_init
flash.h, 123

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

flash_drv_open
flash.h, 123
flash_drv_read
flash.h, 123
flash_drv_write
flash.h, 123
flow_control_t
mdev_uart.h, 250
fpu_deinit
mdev_fpu.h, 208
fpu_init
mdev_fpu.h, 208
fqsn
mdns_service, 51
frame_ctrl_flags
wlan.h, 344
frame_type
wlan.h, 345
free_priv_data_fn
httpd_request_t, 43
frm_ctl
_wifi_mgmt_frame_tx, 32
frm_len
_wifi_mgmt_frame_tx, 32
ft_fclose
ftfs.h, 126
ft_fopen
ftfs.h, 125
ft_fread
ftfs.h, 126
ft_fseek
ftfs.h, 126
ft_ftell
ftfs.h, 126
ft_read_header
ftfs.h, 127
ftfs.h
WM_E_FTFS_INIT_FAIL, 127
WM_E_FTFS_INVALID_FILE, 127
WM_E_FTFS_SEEK_FAIL, 127
ftfs.h, 125
ft_fclose, 126
ft_fopen, 125
ft_fread, 126
ft_fseek, 126
ft_ftell, 126
ft_read_header, 127
ftfs_cli_init, 127
ftfs_init, 125
wm_ftfs_errno, 127
ftfs_cli_init
ftfs.h, 127
ftfs_init
ftfs.h, 125
function
cli_command, 34
httpd_ssi_call, 43
fw_data_read_fn
upgrade_struct_t, 68
fw_data_read_fn_t
secure_upgrade.h, 306
fw_upgrade_ed_chacha.h, 128
ed_chacha_client_mode_upgrade_fw, 128
ed_chacha_server_mode_upgrade_fw, 128
fw_upgrade_rsa_aes.h, 129
AES_KEY_LEN, 131
rsa2048_aes128_client_mode_upgrade_fw, 130
rsa2048_aes128_server_mode_upgrade_fw, 130

GPIO_ACTIVE_HIGH
generic_io.h, 131
GPIO_ACTIVE_LOW
generic_io.h, 131
gen_level
partition_entry, 58
partition_table, 59
generator
DH_PG_PARAMS, 35
generatorLen
DH_PG_PARAMS, 35
generic_io.h, 131
GPIO_ACTIVE_HIGH, 131
GPIO_ACTIVE_LOW, 131
get_handler
httpd_wsgi_call, 46
get_random_sequence
wm_utils.h, 378
gmtime_r
wmtime.h, 391
go_ssid
p2p_scan_result, 57
gpio
input_gpio_cfg_t, 46
output_gpio_cfg_t, 53
gpio_drv_close
mdev_gpio.h, 210
gpio_drv_get_io_domain
mdev_gpio.h, 212
gpio_drv_init
mdev_gpio.h, 210
gpio_drv_open
mdev_gpio.h, 210
gpio_drv_read
mdev_gpio.h, 211
gpio_drv_set_cb
mdev_gpio.h, 212
gpio_drv_setdir
mdev_gpio.h, 211
gpio_drv_write
mdev_gpio.h, 211
gpio_irq_cb
mdev_gpio.h, 212
gpt_drv_close
mdev_gpt.h, 214
gpt_drv_init
mdev_gpt.h, 213
gpt_drv_open

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

mdev_gpt.h, 214
gpt_drv_pwm
mdev_gpt.h, 215
gpt_drv_set
mdev_gpt.h, 214
gpt_drv_setcb
mdev_gpt.h, 214
gpt_drv_start
mdev_gpt.h, 215
gpt_drv_stop
mdev_gpt.h, 215
group_capability
p2p_scan_result, 57
gw
ipv4_config, 47

HTTP_CONTENT_HTML
httpd.h, 165
HTTP_CONTENT_JPEG
httpd.h, 165
HTTP_CONTENT_JSON
httpd.h, 165
HTTP_CONTENT_PLAIN_TEXT
httpd.h, 165
HTTP_CONTENT_XML
httpd.h, 165
HTTPD_HDR_ADD_CACHE_CTRL
httpd.h, 165
HTTPD_HDR_ADD_CACHE_CTRL_NO_CHK
httpd.h, 165
HTTPD_HDR_ADD_CONN_CLOSE
httpd.h, 165
HTTPD_HDR_ADD_CONN_KEEP_ALIVE
httpd.h, 165
HTTPD_HDR_ADD_PRAGMA_NO_CACHE
httpd.h, 165
HTTPD_HDR_ADD_SERVER
httpd.h, 165
HTTPD_HDR_ADD_TYPE_CHUNKED
httpd.h, 165
HW_AES_MODE_BYPASS
mdev_aes.h, 198
HW_AES_MODE_CBC
mdev_aes.h, 198
HW_AES_MODE_CCM
mdev_aes.h, 198
HW_AES_MODE_CTR
mdev_aes.h, 198
HW_AES_MODE_ECB
mdev_aes.h, 198
HW_AES_MODE_MMO
mdev_aes.h, 198
HM_NAME_MAX
healthmon.h, 134
HTTP_EVENT_200
httpd.h, 163
HTTP_PORT
httpd.h, 159
HTTP_RES_200
httpd.h, 161
HTTP_RES_204
httpd.h, 162
HTTP_RES_207
httpd.h, 162
HTTP_RES_301
httpd.h, 162
HTTP_RES_304
httpd.h, 162
HTTP_RES_400
httpd.h, 162
HTTP_RES_401
httpd.h, 162
HTTP_RES_404
httpd.h, 162
HTTP_RES_405
httpd.h, 162
HTTP_RES_429
httpd.h, 162
HTTP_RES_470
httpd.h, 162
HTTP_RES_500
httpd.h, 162
HTTP_RES_505
httpd.h, 162
HTTPD_DONE
httpd.h, 160
HTTPD_MAX_MESSAGE
httpd.h, 160
HTTPD_MAX_SSI_ARGS
httpd.h, 163
HTTPD_REQ_TYPE_GET
httpd.h, 160
HTTPD_REQ_TYPE_HEAD
httpd.h, 160
HTTPD_REQ_TYPE_POST
httpd.h, 160
HTTPD_REQ_TYPE_PUT
httpd.h, 160
HTTPD_SSI_CALL
httpd.h, 163
HTTPS_PORT
httpd.h, 159
hash
upgrade_struct_t, 68
hash_ctx
upgrade_struct_t, 68
hash_finish
upgrade_struct_t, 68
hash_finish_func_t
secure_upgrade.h, 305
hash_len
upgrade_struct_t, 68
hash_update
upgrade_struct_t, 68
hash_update_func_t
secure_upgrade.h, 305
hdr_fields

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

- httpd_wsgi_call, 45

hdr_parsed
- httpd_request_t, 42

healthmon.h, 131
- HM_NAME_MAX, 134
- healthmon_change_check_interval, 133
- healthmon_init, 132
- healthmon_register_cli, 133
- healthmon_register_handler, 132
- healthmon_set_final_about_to_die_handler, 133
- healthmon_start, 132
- healthmon_unregister_handler, 132
- MAX_HM_HANDLERS, 134
- WD_STROBE_INTERVAL, 134
- WKUP_INTERVAL, 134

healthmon_change_check_interval
- healthmon.h, 133

healthmon_handler, 37
- about_to_die, 37
- check_interval, 37
- consecutive_failures, 37
- is_sick, 37
- name, 37

healthmon_init
- healthmon.h, 132

healthmon_register_cli
- healthmon.h, 133

healthmon_register_handler
- healthmon.h, 132

healthmon_set_final_about_to_die_handler
- healthmon.h, 133

healthmon_start
- healthmon.h, 132

healthmon_unregister_handler
- healthmon.h, 132

help
- cli_command, 34

hex2bin
- wm_utils.h, 376

High level RFGET APIs, 27
- rfget_client_mode_update, 28
- rfget_server_mode_update, 27

http_add_header
- httpc.h, 136

http_close_session
- httpc.h, 142

http_content_type
- httpd.h, 165

http_date_to_time
- wmtime.h, 390

http_flags
- httpd_wsgi_call, 45

http_get_response_hdr
- httpc.h, 137

http_get_response_hdr_all
- httpc.h, 139

http_get_response_hdr_value
- httpc.h, 138

http_get_sockfd_from_handle
- httpc.h, 143

http_getsockopt
- httpc.h, 143

http_hdr_field_sel_t
- httpc.h, 144

http_lowlevel_read
- httpc.h, 141

http_lowlevel_write
- httpc.h, 141

http_open_flags_t
- httpc.h, 145

http_open_session
- httpc.h, 136

http_parse.h, 134
- httpd_get_tag_from_post_data, 134
- httpd_get_tag_from_url, 134

http_parse_URL
- httpc.h, 140

http_prepare_req
- httpc.h, 136

http_read_content
- httpc.h, 140

http_req_t, 38
- content, 38
- content_len, 38
- resource, 38
- type, 38
- version, 38

http_resp_t, 39
- chunked, 40
- content_encoding, 40
- content_length, 40
- content_length_field_present, 40
- content_type, 40
- keep_alive_ack, 40
- keep_alive_max, 40
- keep_alive_timeout, 40
- protocol, 39
- reason_phrase, 39
- server, 39
- status_code, 39
- version, 39

http_send_request
- httpc.h, 137

http_setsockopt
- httpc.h, 143

httpc.h
- TLS_ENABLE, 145
- WM_E_HTTPC_BAD_REQUEST, 145
- WM_E_HTTPC_FILE_NOT_FOUND, 145
- WM_E_HTTPC_SOCKET_ERROR, 145
- WM_E_HTTPC_SOCKET_SHUTDOWN, 145
- WM_E_HTTPC_TCP_CONNECT_FAIL, 145
- WM_E_HTTPC_TLS_NOT_ENABLED, 145

httpc.h, 135
- DEFAULT_RETRY_COUNT, 144
- http_add_header, 136

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

http_close_session, 142
http_get_response_hdr, 137
http_get_response_hdr_all, 139
http_get_response_hdr_value, 138
http_get_sockfd_from_handle, 143
http_getsockopt, 143
http_hdr_field_sel_t, 144
http_lowlevel_read, 141
http_lowlevel_write, 141
http_open_flags_t, 145
http_open_session, 136
http_parse_URL, 140
http_prepare_req, 136
http_read_content, 140
http_send_request, 137
http_setsockopt, 143
httpc_get, 144
httpc_validate_url, 144
httpc_write_chunked, 142
wm_httpc_errno, 145
httpc_cfg_t, 40
flags, 41
retry_cnt, 41
httpc_get
httpc.h, 144
httpc_validate_url
httpc.h, 144
httpc_write_chunked
httpc.h, 142
httpd.h
HTTP_CONTENT_HTML, 165
HTTP_CONTENT_JPEG, 165
HTTP_CONTENT_JSON, 165
HTTP_CONTENT_PLAIN_TEXT, 165
HTTP_CONTENT_XML, 165
HTTPD_HDR_ADD_CACHE_CTRL, 165
HTTPD_HDR_ADD_CACHE_CTRL_NO_CHK, 165
HTTPD_HDR_ADD_CONN_CLOSE, 165
HTTPD_HDR_ADD_CONN_KEEP_ALIVE, 165
HTTPD_HDR_ADD_PRAGMA_NO_CACHE, 165
HTTPD_HDR_ADD_SERVER, 165
HTTPD_HDR_ADD_TYPE_CHUNKED, 165
WM_E_HTTPD_DATA_RD, 165
WM_E_HTTPD_HANDLER_400, 165
WM_E_HTTPD_HANDLER_404, 165
WM_E_HTTPD_HANDLER_500, 165
WM_E_HTTPD_HDR_FNAME, 165
WM_E_HTTPD_NO_HANDLER, 165
WM_E_HTTPD_NOTSUPP, 165
WM_E_HTTPD_SSI_MAX, 165
WM_E_HTTPD_TOOLONG, 165
httpd.h, 145
HTTP_EVENT_200, 163
HTTP_PORT, 159
HTTP_RES_200, 161
HTTP_RES_204, 162
HTTP_RES_207, 162
HTTP_RES_301, 162
HTTP_RES_304, 162
HTTP_RES_400, 162
HTTP_RES_401, 162
HTTP_RES_404, 162
HTTP_RES_405, 162
HTTP_RES_429, 162
HTTP_RES_470, 162
HTTP_RES_500, 162
HTTP_RES_505, 162
HTTPD_DONE, 160
HTTPD_MAX_MESSAGE, 160
HTTPD_MAX_SSI_ARGS, 163
HTTPD_REQ_TYPE_GET, 160
HTTPD_REQ_TYPE_PUT, 160
HTTPD_SSI_CALL, 163
HTTPS_PORT, 159
http_content_type, 165
httpd_clear_priv_data, 151
httpd_cli_init, 154
httpd_free_priv_data_fn_t, 164
httpd_get_data, 158
httpd_get_priv_data, 151
httpd_handle_file, 153
httpd_hdr_field_sel_t, 165
httpd_init, 149
httpd_is_running, 150
httpd_parse_hdr_cb_t, 164
httpd_parse_hdr_tags, 158
httpd_parse_hdr_tags_with_cb, 157
httpd_purge_all_socket_data, 159
httpd_purge_headers, 157
httpd_recv, 154
httpd_recv_func_t, 164
httpd_recv_override, 159
httpd_register_ssi, 153
httpd_register_wsgi_handler, 151
httpd_register_wsgi_handlers, 152
httpd_send, 150
httpd_send_chunk, 150
httpd_send_chunk_begin, 151
httpd_send_crlf, 163
httpd_send_default_headers, 157
httpd_send_func_t, 164
httpd_send_hdr_from_code, 154
httpd_send_header, 156
httpd_send_override, 159
httpd_send_response, 155
httpd_send_response_301, 156
httpd_send_response_header, 155
httpd_shutdown, 149
httpd_ssifunction, 164
httpd_start, 149
httpd_stop, 149
httpd_unregister_ssi, 153
httpd_unregister_wsgi_handler, 152
httpd_unregister_wsgi_handlers, 152
httpd_use_tls_certificates, 158

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

- httpd_validate_uri, 154
- MAX_REGISTERED_SSIS, 163
- wm_httpd_errno, 165

httpd_clear_priv_data
- httpd.h, 151

httpd_cli_init
- httpd.h, 154

httpd_free_priv_data_fn_t
- httpd.h, 164

httpd_get_data
- httpd.h, 158

httpd_get_priv_data
- httpd.h, 151

httpd_get_tag_from_post_data
- http_parse.h, 134

httpd_get_tag_from_url
- http_parse.h, 134

httpd_handle_file
- httpd.h, 153

httpd_hdr_field_sel_t
- httpd.h, 165

httpd_init
- httpd.h, 149

httpd_is_running
- httpd.h, 150

httpd_parse_hdr_cb_t
- httpd.h, 164

httpd_parse_hdr_tags
- httpd.h, 158

httpd_parse_hdr_tags_with_cb
- httpd.h, 157

httpd_priv
- httpd_request_t, 42

httpd_purge_all_socket_data
- httpd.h, 159

httpd_purge_headers
- httpd.h, 157

httpd_recv
- httpd.h, 154

httpd_recv_func_t
- httpd.h, 164

httpd_recv_override
- httpd.h, 159

httpd_register_ssi
- httpd.h, 153

httpd_register_wsgi_handler
- httpd.h, 151

httpd_register_wsgi_handlers
- httpd.h, 152

httpd_request_t, 41
- agent, 42
- body_nbytes, 42
- chunked, 42
- content_type, 42
- etag_val, 42
- filename, 42
- free_priv_data_fn, 43
- hdr_parsed, 42
- httpd_priv, 42
- if_none_match, 42
- remaining_bytes, 42
- sock, 42
- type, 42
- wsgi, 42

httpd_send
- httpd.h, 150

httpd_send_chunk
- httpd.h, 150

httpd_send_chunk_begin
- httpd.h, 151

httpd_send_crlf
- httpd.h, 163

httpd_send_default_headers
- httpd.h, 157

httpd_send_func_t
- httpd.h, 164

httpd_send_hdr_from_code
- httpd.h, 154

httpd_send_header
- httpd.h, 156

httpd_send_override
- httpd.h, 159

httpd_send_response
- httpd.h, 155

httpd_send_response_301
- httpd.h, 156

httpd_send_response_header
- httpd.h, 155

httpd_shutdown
- httpd.h, 149

httpd_ssi_call, 43
- function, 43
- name, 43

httpd_ssifunction
- httpd.h, 164

httpd_start
- httpd.h, 149

httpd_stop
- httpd.h, 149

httpd_tls_certs_t, 43
- client_cert, 44
- client_cert_size, 44
- server_cert, 44
- server_cert_chained, 44
- server_cert_size, 44
- server_key, 44
- server_key_size, 44

httpd_unregister_ssi
- httpd.h, 153

httpd_unregister_wsgi_handler
- httpd.h, 152

httpd_unregister_wsgi_handlers
- httpd.h, 152

httpd_use_tls_certificates
- httpd.h, 158

httpd_useragent_t, 44

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

product, 45
version, 45
httpd_validate_uri
httpd.h, 154
httpd_wsgi_call, 45
delete_handler, 46
get_handler, 46
hdr_fields, 45
http_flags, 45
put_handler, 46
set_handler, 46
uri, 45
hw_aes_mode_t
mdev_aes.h, 198

I2C_ACTIVE
mdev_i2c.h, 224
I2C_DMA_DISABLE
mdev_i2c.h, 224
I2C_DMA_ENABLE
mdev_i2c.h, 224
I2C_ERROR
mdev_i2c.h, 224
I2C_INACTIVE
mdev_i2c.h, 224
I2C_10_BIT_ADDR
mdev_i2c.h, 224
I2C_CLK_100KHZ
mdev_i2c.h, 223
I2C_CLK_10KHZ
mdev_i2c.h, 223
I2C_CLK_400KHZ
mdev_i2c.h, 224
I2C_DEVICE_SLAVE
mdev_i2c.h, 224
I2C_DMA_MODE
mdev_i2c.h, 224
I2C_SLAVEADR
mdev_i2c.h, 224
i2c_drv_close
mdev_i2c.h, 220
i2c_drv_deinit
mdev_i2c.h, 223
i2c_drv_dma_read
mdev_i2c.h, 222
i2c_drv_dma_write
mdev_i2c.h, 222
i2c_drv_init
mdev_i2c.h, 217
i2c_drv_open
mdev_i2c.h, 218
i2c_drv_read
mdev_i2c.h, 221
i2c_drv_rxbuf_size
mdev_i2c.h, 218
i2c_drv_set_address
mdev_i2c.h, 220
i2c_drv_set_callback
mdev_i2c.h, 219
i2c_drv_set_clkcnt
mdev_i2c.h, 217
i2c_drv_slv_dma_read
mdev_i2c.h, 221
i2c_drv_timeout
mdev_i2c.h, 223
i2c_drv_wait_till_inactivity
mdev_i2c.h, 220
i2c_drv_write
mdev_i2c.h, 221
i2c_drv_xfer_mode
mdev_i2c.h, 218
i2c_status
mdev_i2c.h, 224
IO_DOMAIN_0
mdev_pm.h, 232
IO_DOMAIN_1
mdev_pm.h, 232
IO_DOMAIN_2
mdev_pm.h, 232
IO_DOMAIN_3
mdev_pm.h, 232
if_none_match
httpd_request_t, 42
iface_state
mdns.h, 268
input_gpio_cfg_t, 46
gpio, 46
type, 46
intended_address
p2p_scan_result, 57
io_domain_t
mdev_pm.h, 231
ip
wlan_network, 72
ipaddr
mdns_service, 51
ipv4
wlan_ip_config, 71
ipv4_config, 47
addr_type, 47
address, 47
dns1, 47
dns2, 47
gw, 47
netmask, 47
is_pm_init_done
pwrmgr.h, 299
is_pmf_required
wlan_network_security, 74
is_sick
healthmon_handler, 37
is_sta_connected
wlan.h, 321
is_sta_ipv4_connected
wlan.h, 321
is_uap_started
wlan.h, 321

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

is_wfd_started
wlan.h, 321
iv
upgrade_struct_t, 68
iv_len
upgrade_struct_t, 68

jobj_t, 48
cur, 48
js, 48
num_tokens, 48
parser, 48
tokens, 48
js
jobj_t, 48
json_array_get_array_object
jsonv2.h, 182
json_array_get_bool
jsonv2.h, 179
json_array_get_composite_object
jsonv2.h, 181
json_array_get_float
jsonv2.h, 180
json_array_get_int
jsonv2.h, 179
json_array_get_int64
jsonv2.h, 179
json_array_get_num_elements
jsonv2.h, 178
json_array_get_str
jsonv2.h, 180
json_array_get_str_len
jsonv2.h, 180
json_array_release_array_object
jsonv2.h, 182
json_array_release_composite_object
jsonv2.h, 181
json_close_array
json_generator.h, 168
json_close_object
json_generator.h, 168
json_generator.h, 165
json_close_array, 168
json_close_object, 168
json_pop_array_object, 167
json_pop_object, 168
json_push_array_object, 167
json_push_object, 167
json_set_array_bool, 171
json_set_array_float, 171
json_set_array_int, 171
json_set_array_str, 171
json_set_val_bool, 170
json_set_val_float, 170
json_set_val_float_precision, 170
json_set_val_int, 169
json_set_val_null, 171
json_set_val_str, 169
json_set_val_uint, 169
json_set_val_uint_64, 169
json_start_array, 168
json_start_object, 167
json_str_init, 166
json_get_array_object
jsonv2.h, 178
json_get_composite_object
jsonv2.h, 177
json_get_val_bool
jsonv2.h, 175
json_get_val_float
jsonv2.h, 176
json_get_val_int
jsonv2.h, 175
json_get_val_int64
jsonv2.h, 175
json_get_val_str
jsonv2.h, 176
json_get_val_str_len
jsonv2.h, 176
json_init
jsonv2.h, 174
json_is_array
jsonv2.h, 174
json_is_object
jsonv2.h, 174
json_pop_array_object
json_generator.h, 167
json_pop_object
json_generator.h, 168
json_push_array_object
json_generator.h, 167
json_push_object
json_generator.h, 167
json_release_array_object
jsonv2.h, 178
json_release_composite_object
jsonv2.h, 177
json_set_array_bool
json_generator.h, 171
json_set_array_float
json_generator.h, 171
json_set_array_int
json_generator.h, 171
json_set_array_str
json_generator.h, 171
json_set_val_bool
json_generator.h, 170
json_set_val_float
json_generator.h, 170
json_set_val_float_precision
json_generator.h, 170
json_set_val_int
json_generator.h, 169
json_set_val_null
json_generator.h, 171
json_set_val_str
json_generator.h, 169

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

json_set_val_uint
    json_generator.h, 169
json_set_val_uint_64
    json_generator.h, 169
json_start_array
    json_generator.h, 168
json_start_object
    json_generator.h, 167
json_str_init
    json_generator.h, 166
jsontok_t, 48
    end, 49
    parent, 49
    size, 49
    start, 49
    type, 49
jsonv2.h
    WM_E_JSON_FAIL, 182
    WM_E_JSON_INCOMPLETE, 182
    WM_E_JSON_INVAL, 182
    WM_E_JSON_INVALID_INDEX, 183
    WM_E_JSON_INVALID_JARRAY, 183
    WM_E_JSON_INVALID_JOBJ, 183
    WM_E_JSON_INVALID_TYPE, 183
    WM_E_JSON_INVSTART, 183
    WM_E_JSON_NOMEM, 182
    WM_E_JSON_NOT_FOUND, 183
    WM_E_JSON_OBUF, 183
jsonv2.h, 172
    json_array_get_array_object, 182
    json_array_get_bool, 179
    json_array_get_composite_object, 181
    json_array_get_float, 180
    json_array_get_int, 179
    json_array_get_int64, 179
    json_array_get_num_elements, 178
    json_array_get_str, 180
    json_array_get_str_len, 180
    json_array_release_array_object, 182
    json_array_release_composite_object, 181
    json_get_array_object, 178
    json_get_composite_object, 177
    json_get_val_bool, 175
    json_get_val_float, 176
    json_get_val_int, 175
    json_get_val_int64, 175
    json_get_val_str, 176
    json_get_val_str_len, 176
    json_init, 174
    json_is_array, 174
    json_is_object, 174
    json_release_array_object, 178
    json_release_composite_object, 177
    wm_json_errno, 182

KEY_APP_DATA_END
    keystore.h, 185
KEY_APP_DATA_START
    keystore.h, 185
keep_alive_ack
    http_resp_t, 40
keep_alive_max
    http_resp_t, 40
keep_alive_timeout
    http_resp_t, 40
keystore.h
    KEY_APP_DATA_END, 185
    KEY_APP_DATA_START, 185
keystore.h, 183
    keystore_get, 184
    keystore_get_ref, 184
    keystore_iterate, 184
    keystore_iterate_ref, 185
    keystore_iterate_start, 184
    ks_tlv_type, 185
keystore_get
    keystore.h, 184
keystore_get_ref
    keystore.h, 184
keystore_iterate
    keystore.h, 184
keystore_iterate_ref
    keystore.h, 185
keystore_iterate_start
    keystore.h, 184
keyvals
    mdns_service, 51
ks_tlv_type
    keystore.h, 185
kvlen
    mdns_service, 51

LED_OFF
    led_indicator.h, 187
LED_ON
    led_indicator.h, 187
led_indicator.h
    LED_OFF, 187
    LED_ON, 187
led_blink
    led_indicator.h, 186
led_fast_blink
    led_indicator.h, 186
led_get_state
    led_indicator.h, 186
led_indicator.h, 186
    led_blink, 186
    led_fast_blink, 186
    led_get_state, 186
    led_off, 186
    led_on, 186
    led_slow_blink, 187
    led_state, 187
led_off
    led_indicator.h, 186
led_on
    led_indicator.h, 186
led_slow_blink

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

led_indicator.h, 187
led_state
led_indicator.h, 187
listen_channel
p2p_scan_result, 57
ll_printf
wmstdio.h, 388
load_wep_key
wlan.h, 332
Low level RFGET APIs, 20
rfget_get_passive_firmware, 22
rfget_get_passive_wifi_firmware, 22
rfget_init, 20
rfget_is_update_in_progress, 22
rfget_update_abort, 21
rfget_update_begin, 20
rfget_update_complete, 21
rfget_update_data, 21
update_desc_t, 20
verify_load_firmware, 22

MAX_DEVICE_NAME_LEN
p2p.h, 273
MAX_HM_HANDLERS
healthmon.h, 134
MAX_INTERFACE
mdns.h, 266
MAX_IPV6_ADDRESSES
wlan.h, 340
MAX_MDNS_LST
mdns.h, 266
MAX_MSG_LEN
wmstdio.h, 389
MAX_PB_CB
push_button.h, 297
MAX_REGISTERED_SSIS
httpd.h, 163
MDNS_MAX_KEYVAL_LEN
mdns.h, 266
MDNS_MAX_LABEL_LEN
mdns.h, 266
MDNS_MAX_NAME_LEN
mdns.h, 266
MDNS_PROTO_TCP
mdns.h, 266
MDNS_PROTO_UDP
mdns.h, 266
MOD_AF
wmerrno.h, 385
MOD_CRIT_ERR
wmerrno.h, 387
MOD_CRYPTO
wmerrno.h, 386
MOD_DHCPD
wmerrno.h, 385
MOD_DIAG
wmerrno.h, 386
MOD_ERR_LAST
wmerrno.h, 387
MOD_EXML
wmerrno.h, 385
MOD_FTFS
wmerrno.h, 385
MOD_HTTPC
wmerrno.h, 386
MOD_HTTPD
wmerrno.h, 385
MOD_JSON
wmerrno.h, 385
MOD_MDNS
wmerrno.h, 385
MOD_PROV
wmerrno.h, 386
MOD_PSM
wmerrno.h, 386
MOD_RFGET
wmerrno.h, 385
MOD_SMDNS
wmerrno.h, 385
MOD_SPI
wmerrno.h, 386
MOD_SYSINFO
wmerrno.h, 386
MOD_TELNETD
wmerrno.h, 385
MOD_TTCP
wmerrno.h, 386
MOD_UNUSED_1
wmerrno.h, 386
MOD_UNUSED_2
wmerrno.h, 386
MOD_UNUSED_3
wmerrno.h, 385
MOD_USB
wmerrno.h, 387
MOD_WIFI
wmerrno.h, 387
MOD_WLAN
wmerrno.h, 386
MOD_WPS
wmerrno.h, 386
magic
partition_table, 59
manufacture
p2p_config, 55
wps_config, 77
max_tx_pwr
wifi_sub_band_set_t, 70
mcstCipher
wlan_network_security, 74
mdev.h, 187
mdev_deregister, 188
mdev_get_handle, 188
mdev_register, 187
mdev_aes.h
HW_AES_MODE_BYPASS, 198
HW_AES_MODE_CBC, 198

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

HW_AES_MODE_CCM, 198
HW_AES_MODE_CTR, 198
HW_AES_MODE_ECB, 198
HW_AES_MODE_MMO, 198
mdev_i2c.h
I2C_ACTIVE, 224
I2C_DMA_DISABLE, 224
I2C_DMA_ENABLE, 224
I2C_ERROR, 224
I2C_INACTIVE, 224
mdev_pm.h
EGPIO0, 231
EGPIO1, 231
ERTC, 231
EWLAN, 231
IO_DOMAIN_0, 232
IO_DOMAIN_1, 232
IO_DOMAIN_2, 232
IO_DOMAIN_3, 232
PM0, 232
PM1, 232
PM2, 232
PM3, 232
PM4, 232
mdev_ssp.h
DMA_DISABLE, 244
DMA_ENABLE, 244
mdev_uart.h
FLOW_CONTROL_HW, 250
FLOW_CONTROL_NONE, 250
FLOW_CONTROL_SW, 250
UART_DMA_DISABLE, 249
UART_DMA_ENABLE, 249
mdev_usb.h
USB_CDC_CLASS, 252
USB_HID_CLASS, 252
USB_MSC_CLASS, 252
mdev_acomp.h, 188
acomp_drv_close, 190
acomp_drv_deinit, 190
acomp_drv_init, 189
acomp_drv_open, 189
acomp_drv_result, 189
mdev_adc.h, 190
adc_drv_close, 193
adc_drv_deinit, 193
adc_drv_get_samples, 193
adc_drv_init, 191
adc_drv_open, 192
adc_drv_result, 192
adc_drv_timeout, 192
adc_get_config, 191
adc_modify_default_config, 191
mdev_aes.h, 193
aes_ctr_final, 197
aes_drv_ccm_decrypt, 197
aes_drv_ccm_encrypt, 196
aes_drv_ccm_setkey, 195
aes_drv_close, 198
aes_drv_decrypt, 197
aes_drv_encrypt, 196
aes_drv_init, 195
aes_drv_open, 195
aes_drv_setkey, 195
hw_aes_mode_t, 198
mdev_crc.h, 198
crc_drv_block_calc, 200
crc_drv_close, 201
crc_drv_init, 200
crc_drv_open, 200
crc_drv_stream_feed, 201
crc_drv_stream_get_result, 201
crc_drv_stream_setlen, 201
crc_mode_t, 202
mdev_dac.h, 202
dac_drv_close, 204
dac_drv_deinit, 204
dac_drv_init, 203
dac_drv_open, 203
dac_drv_output, 204
dac_modify_default_config, 203
mdev_deregister
mdev.h, 188
mdev_dma.h, 204
dma_drv_close, 207
dma_drv_deinit, 208
dma_drv_init, 205
dma_drv_open, 205
dma_drv_set_cb, 206
dma_drv_transfer, 206
dma_drv_wait_for_transfer_complete, 207
mdev_fpu.h, 208
fpu_deinit, 208
fpu_init, 208
mdev_get_handle
mdev.h, 188
mdev_gpio.h, 208
gpio_drv_close, 210
gpio_drv_get_io_domain, 212
gpio_drv_init, 210
gpio_drv_open, 210
gpio_drv_read, 211
gpio_drv_set_cb, 212
gpio_drv_setdir, 211
gpio_drv_write, 211
gpio_irq_cb, 212
mdev_gpt.h, 213
gpt_drv_close, 214
gpt_drv_init, 213
gpt_drv_open, 214
gpt_drv_pwm, 215
gpt_drv_set, 214
gpt_drv_setcb, 214
gpt_drv_start, 215
gpt_drv_stop, 215
mdev_i2c.h, 216

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

BIT10, 223
I2C_10_BIT_ADDR, 224
I2C_CLK_100KHZ, 223
I2C_CLK_10KHZ, 223
I2C_CLK_400KHZ, 224
I2C_DEVICE_SLAVE, 224
I2C_DMA_MODE, 224
I2C_SLAVEADR, 224
i2c_drv_close, 220
i2c_drv_deinit, 223
i2c_drv_dma_read, 222
i2c_drv_dma_write, 222
i2c_drv_init, 217
i2c_drv_open, 218
i2c_drv_read, 221
i2c_drv_rxbuf_size, 218
i2c_drv_set_address, 220
i2c_drv_set_callback, 219
i2c_drv_set_clkcnt, 217
i2c_drv_slv_dma_read, 221
i2c_drv_timeout, 223
i2c_drv_wait_till_inactivity, 220
i2c_drv_write, 221
i2c_drv_xfer_mode, 218
i2c_status, 224
NUM_I2C_PORTS, 223
mdev_pinmux.h, 224
pinmux_drv_close, 225
pinmux_drv_get_gpio_func, 226
pinmux_drv_init, 225
pinmux_drv_open, 225
pinmux_drv_setfunc, 226
mdev_pm.h, 226
arch_pm_init, 229
arch_reboot, 229
io_domain_t, 231
pm_mcu_cfg, 228
pm_mcu_get_mode, 229
pm_mcu_io_cfg, 229
pm_mcu_is_enabled, 229
pm_mcu_state, 228
pm_wakeup_source, 229
pmu_drv_powerdeepoff_vddio, 231
pmu_drv_powernormal_vddio, 231
pmu_drv_poweroff_vddio, 230
pmu_drv_poweron_vddio, 230
power_state_t, 232
wakeup_cause, 231
mdev_register
mdev.h, 187
mdev_rtc.h, 232
rtc_drv_close, 236
rtc_drv_get, 235
rtc_drv_get_uppval, 235
rtc_drv_init, 233
rtc_drv_open, 233
rtc_drv_reset, 234
rtc_drv_set, 234
rtc_drv_set_alarm, 235
rtc_drv_set_alarm_cb, 234
rtc_drv_set_cb, 234
rtc_drv_start, 235
rtc_drv_stop, 235
mdev_sdio.h, 236
sdio_drv_creg_read, 237
sdio_drv_creg_write, 237
sdio_drv_deinit, 238
sdio_drv_init, 238
sdio_drv_open, 238
sdio_drv_read, 237
sdio_drv_set_cb1, 239
sdio_drv_write, 238
mdev_ssp.h, 239
SSP_DMA, 244
ssp_drv_close, 241
ssp_drv_cs_activate, 242
ssp_drv_cs_deactivate, 242
ssp_drv_deinit, 240
ssp_drv_init, 240
ssp_drv_open, 241
ssp_drv_read, 242
ssp_drv_rxbuf_size, 243
ssp_drv_set_clk, 243
ssp_drv_timeout, 243
ssp_drv_write, 241
mdev_uart.h, 244
flow_control_t, 250
UART_8BIT, 249
UART_9BIT, 249
UART_DMA_MODE, 249
uart_drv_blocking_read, 248
uart_drv_close, 246
uart_drv_deinit, 245
uart_drv_dma_rd_blk_size, 248
uart_drv_get_portid, 247
uart_drv_init, 245
uart_drv_open, 246
uart_drv_read, 246
uart_drv_rx_buf_reset, 249
uart_drv_rxbuf_size, 247
uart_drv_set_opts, 248
uart_drv_tx_flush, 247
uart_drv_write, 246
uart_drv_xfer_mode, 245
mdev_usb.h, 250
usb_class, 252
usb_drv_close, 251
usb_drv_init, 251
usb_drv_open, 251
usb_drv_read, 251
usb_drv_write, 252
mdev_usb_host.h, 253
USB_CDC_DEBUG, 258
USB_UVC_DEBUG, 257
usb_host_cdc_drv_init, 254
usb_host_drv_block_read, 257

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

usb_host_drv_block_write, 257
usb_host_drv_close, 257
usb_host_drv_info, 256
usb_host_drv_open, 255
usb_host_drv_read, 256
usb_host_drv_start, 255
usb_host_drv_stop, 255
usb_host_drv_write, 256
usb_host_msc_drv_init, 254
usb_host_uac_drv_init, 254
usb_host_uvc_drv_init, 254
mdev_wdt.h, 258
wdt_drv_init, 258
wdt_drv_open, 259
wdt_drv_set_timeout, 259
wdt_drv_start, 259
wdt_drv_strobe, 259
mdns.h
DOWN, 268
REANNOUNCE, 268
UP, 268
WM_E_MDNS_BADSRC, 268
WM_E_MDNS_FBIND, 268
WM_E_MDNS_FBINDTODEVICE, 268
WM_E_MDNS_FMCAST_JOIN, 268
WM_E_MDNS_FMCAST_SET, 268
WM_E_MDNS_FQUERY_SOC, 268
WM_E_MDNS_FQUERY_THREAD, 268
WM_E_MDNS_FREUSE, 268
WM_E_MDNS_FSOC, 268
WM_E_MDNS_INUSE, 268
WM_E_MDNS_INVAL, 268
WM_E_MDNS_NOIMPL, 268
WM_E_MDNS_NOMEM, 268
WM_E_MDNS_NORESP, 268
WM_E_MDNS_TOOBIG, 268
mdns.h, 260
iface_state, 268
MAX_INTERFACE, 266
MAX_MDNS_LST, 266
MDNS_MAX_LABEL_LEN, 266
MDNS_MAX_NAME_LEN, 266
MDNS_PROTO_TCP, 266
MDNS_PROTO_UDP, 266
mdns_announce_service, 264
mdns_deannounce_service, 264
mdns_iface_state_change, 265
mdns_query_cb, 267
mdns_query_monitor, 263
mdns_query_unmonitor, 263
mdns_set_txt_rec, 265
mdns_start, 262
mdns_stop, 263
wm_mdns_errno, 268
mdns_announce_service
mdns.h, 264
mdns_deannounce_service
mdns.h, 264
mdns_iface_state_change
mdns.h, 265
mdns_query_cb
mdns.h, 267
mdns_query_monitor
mdns.h, 263
mdns_query_unmonitor
mdns.h, 263
mdns_service, 49
domain, 50
flags, 51
fqsn, 51
ipaddr, 51
keyvals, 51
kvlen, 51
port, 51
proto, 51
ptrname, 51
servname, 50
servtype, 50
mdns_set_txt_rec
mdns.h, 265
mdns_start
mdns.h, 262
mdns_stop
mdns.h, 263
Middle level RFGET APIs, 23
data_fetch, 23
data_validate, 23
update_and_validate_firmware, 24
update_and_validate_fs, 25
update_and_validate_wifi_firmware, 24
update_firmware, 25
update_wifi_firmware, 26
mktime
wmtime.h, 391
mod_group
wifi-decl.h, 310
model_name
p2p_config, 55
wps_config, 77
model_number
p2p_config, 55
wps_config, 77
mrvl_aes_unwrap
wmcrypto.h, 382
mrvl_aes_unwrap_ext
wmcrypto.h, 383
mrvl_aes_wrap
wmcrypto.h, 382
mrvl_aes_wrap_ext
wmcrypto.h, 383
mrvl_crypto_hmac_md5
wmcrypto.h, 383
mrvl_crypto_md5
wmcrypto.h, 384
mrvl_crypto_pass_to_key
wmcrypto.h, 384

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

mrvl_dh_compute_key
wmcrypto.h, 380
mrvl_dh_free
wmcrypto.h, 381
mrvl_dh_setup_key
wmcrypto.h, 380
mrvl_hmac_sha256
wmcrypto.h, 382
mrvl_kdf
wmcrypto.h, 382
mrvl_sha1_vector
wmcrypto.h, 381
mrvl_sha256
wmcrypto.h, 381
mrvl_sha256_vector
wmcrypto.h, 381

NETWORK_NOT_FOUND
app_framework.h, 101
NUM_I2C_PORTS
mdev_i2c.h, 223
name
cli_command, 34
healthmon_handler, 37
httpd_ssi_call, 43
partition_entry, 58
wlan_network, 72
net_get_if_addr
wm_net.h, 354
net_get_if_ip_addr
wm_net.h, 354
net_get_if_ip_mask
wm_net.h, 355
net_get_sta_handle
wm_net.h, 353
net_get_uap_handle
wm_net.h, 353
net_get_wfd_handle
wm_net.h, 353
net_gethostbyname
wm_net.h, 351
net_interface_dhcp_stop
wm_net.h, 354
net_interface_down
wm_net.h, 354
net_interface_up
wm_net.h, 353
net_ipv4stack_init
wm_net.h, 355
net_sock_to_interface
wm_net.h, 352
netif_add_udp_broadcast_filter
wm_net.h, 352
netif_remove_udp_broadcast_filter
wm_net.h, 352
netmask
ipv4_config, 47
no_of_chan
wifi_sub_band_set_t, 70
no_of_sub_band
wifi_domain_param_t, 69
num_chans
wifi-decl.h, 310
num_mod_grps
wifi-decl.h, 310
num_tokens
jobj_t, 48
nw_utils.h, 269
nw_utils_cli_deinit, 269
nw_utils_cli_init, 269
nw_utils_cli_deinit
nw_utils.h, 269
nw_utils_cli_init
nw_utils.h, 269

OS_TIMER_AUTO_ACTIVATE
wm_os.h, 374
OS_TIMER_NO_ACTIVATE
wm_os.h, 374
OS_TIMER_ONE_SHOT
wm_os.h, 374
OS_TIMER_PERIODIC
wm_os.h, 374
o_bss_end
overlay_ranges, 54
o_bss_start
overlay_ranges, 54
o_name
overlay, 53
o_no_parts
overlay, 53
o_ram_start
overlay, 53
o_range
overlay, 53
OS_MUTEX_INHERIT
wm_os.h, 372
OS_NO_WAIT
wm_os.h, 372
OS_WAIT_FOREVER
wm_os.h, 372
op_channel
p2p_scan_result, 57
opcode
wslay_frame, 78
or_flash_end
overlay_ranges, 54
or_flash_start
overlay_ranges, 54
os_dump_mem_stats
wm_os.h, 371
os_get_runtime_stats
wm_os.h, 373
os_get_timestamp
wm_os.h, 356
os_get_usec_counter
wm_os.h, 356
os_heap_add_bank

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

wm_os.h, 371
os_mem_alloc
wm_os.h, 372
os_mem_calloc
wm_os.h, 371
os_mem_free
wm_os.h, 373
os_mem_realloc
wm_os.h, 373
os_msec_to_ticks
wm_os.h, 358
os_mutex_create
wm_os.h, 362
os_mutex_delete
wm_os.h, 364
os_mutex_get
wm_os.h, 362
os_mutex_put
wm_os.h, 363
os_queue_create
wm_os.h, 359
os_queue_delete
wm_os.h, 360
os_queue_get_msgs_waiting
wm_os.h, 361
os_queue_pool, 51
size, 52
os_queue_pool_define
wm_os.h, 372
os_queue_pool_t
wm_os.h, 373
os_queue_recv
wm_os.h, 360
os_queue_send
wm_os.h, 359
os_recursive_mutex_create
wm_os.h, 363
os_recursive_mutex_get
wm_os.h, 364
os_recursive_mutex_put
wm_os.h, 364
os_remove_idle_function
wm_os.h, 361
os_remove_tick_function
wm_os.h, 362
os_rwlock_create
wm_os.h, 367
os_rwlock_delete
wm_os.h, 367
os_rwlock_read_lock
wm_os.h, 368
os_rwlock_read_unlock
wm_os.h, 368
os_rwlock_write_lock
wm_os.h, 367
os_rwlock_write_unlock
wm_os.h, 368
os_semaphore_create
wm_os.h, 365
os_semaphore_create_counting
wm_os.h, 365
os_semaphore_delete
wm_os.h, 366
os_semaphore_get
wm_os.h, 365
os_semaphore_getcount
wm_os.h, 366
os_semaphore_put
wm_os.h, 366
os_setup_idle_function
wm_os.h, 361
os_setup_tick_function
wm_os.h, 361
os_thread_create
wm_os.h, 357
os_thread_delete
wm_os.h, 357
os_thread_relinquish
wm_os.h, 372
os_thread_self_complete
wm_os.h, 359
os_thread_sleep
wm_os.h, 358
os_thread_stack, 52
size, 52
os_thread_stack_define
wm_os.h, 372
os_thread_stack_t
wm_os.h, 373
os_thread_wait_abort
wm_os.h, 358
os_ticks_get
wm_os.h, 372
os_ticks_to_msec
wm_os.h, 358
os_ticks_to_unblock
wm_os.h, 372
os_timer_activate
wm_os.h, 369, 374
os_timer_activate_t
wm_os.h, 374
os_timer_change
wm_os.h, 369
os_timer_create
wm_os.h, 368
os_timer_deactivate
wm_os.h, 370
os_timer_delete
wm_os.h, 371
os_timer_get_context
wm_os.h, 370
os_timer_is_running
wm_os.h, 369
os_timer_reload
wm_os.h, 374
os_timer_reload_t

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

wm_os.h, 374
os_timer_reset
wm_os.h, 370
os_total_ticks_get
wm_os.h, 372
os_version
p2p_config, 56
wps_config, 78
Other RFGET APIs, 29
rfget_cli_init, 29
rfget_ftfs_init, 29
output_gpio_cfg_t, 52
gpio, 53
type, 53
overlay, 53
o_name, 53
o_no_parts, 53
o_ram_start, 53
o_range, 53
overlay_load
overlays.h, 270
overlay_ranges, 54
o_bss_end, 54
o_bss_start, 54
or_flash_end, 54
or_flash_start, 54
overlays.h, 269
overlay_load, 270

P2P_AP_SESSION_SUCCESSFUL
p2p.h, 273
P2P_AUTO_GO_STARTED
p2p.h, 273
P2P_CLIENT_STARTED
p2p.h, 273
P2P_CONNECT_FAILED
app_framework.h, 101
P2P_DEVICE_STARTED
p2p.h, 273
P2P_FAILED
p2p.h, 273
P2P_FINISHED
p2p.h, 273
P2P_GO_STARTED
p2p.h, 273
P2P_SESSION_ABORTED
p2p.h, 273
P2P_SESSION_FAILED
p2p.h, 273
P2P_SESSION_PIN_CHKSUM_FAILED
p2p.h, 273
P2P_SESSION_STARTED
p2p.h, 273
P2P_SESSION_SUCCESSFUL
p2p.h, 273
P2P_SESSION_TIMEOUT
p2p.h, 273
P2P_STARTED
p2p.h, 273
P2P_GROUP_OWNER_MASK
p2p.h, 272
P2P_LOG
p2p.h, 272
p2p.h
CMD_P2P_PBC, 273
CMD_P2P_PIN, 273
P2P_AP_SESSION_SUCCESSFUL, 273
P2P_AUTO_GO_STARTED, 273
P2P_CLIENT_STARTED, 273
P2P_DEVICE_STARTED, 273
P2P_FAILED, 273
P2P_FINISHED, 273
P2P_GO_STARTED, 273
P2P_SESSION_ABORTED, 273
P2P_SESSION_FAILED, 273
P2P_SESSION_PIN_CHKSUM_FAILED, 273
P2P_SESSION_STARTED, 273
P2P_SESSION_SUCCESSFUL, 273
P2P_SESSION_TIMEOUT, 273
P2P_STARTED, 273
p2p.h, 270
P2P_LOG, 272
p2p_connect, 271
p2p_disconnect, 272
p2p_event, 273
p2p_generate_pin, 272
p2p_scan, 271
p2p_session_command, 273
p2p_start, 271
p2p_stop, 271
p2p_validate_pin, 272
p2p_callback
p2p_config, 56
p2p_config, 54
auto_go, 55
config_methods, 56
device_name, 55
manufacture, 55
model_name, 55
model_number, 55
os_version, 56
p2p_callback, 56
p2p_msg_max_retry, 56
p2p_msg_timeout, 56
pin_generator, 55
pin_len, 56
primary_dev_category, 56
primary_dev_subcategory, 56
rf_bands, 56
role, 55
serial_number, 56
version, 55
version2, 55
p2p_connect
p2p.h, 271
p2p_disconnect
p2p.h, 272

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

p2p_event
p2p.h, 273
p2p_generate_pin
p2p.h, 272
p2p_msg_max_retry
p2p_config, 56
p2p_msg_timeout
p2p_config, 56
p2p_scan
p2p.h, 271
p2p_scan_result, 57
device_address, 57
device_id, 57
device_name, 57
go_ssid, 57
group_capability, 57
intended_address, 57
listen_channel, 57
op_channel, 57
p2p_session_command
p2p.h, 273
p2p_start
p2p.h, 271
p2p_stop
p2p.h, 271
p2p_validate_pin
p2p.h, 272
PM0
mdev_pm.h, 232
PM1
mdev_pm.h, 232
PM2
mdev_pm.h, 232
PM3
mdev_pm.h, 232
PM4
mdev_pm.h, 232
PROBE_REQ
wlan.h, 344
PROV_ABORTED
provisioning.h, 284
PROV_MICRO_AP_UP_REQUESTED
provisioning.h, 283
PROV_NETWORK_SET_CB
provisioning.h, 283
PROV_ON_DEMAND_SCAN
provisioning.h, 283
PROV_PERIODIC_SCAN
provisioning.h, 283
PROV_RESET_TO_SMC
provisioning.h, 283
PROV_SCAN_CFG_SET_CB
provisioning.h, 284
PROV_START
provisioning.h, 283
PROV_START_SMART_MODE
provisioning.h, 283
PROV_STOP_SMART_MODE
provisioning.h, 283
PROV_WPS_PBC_TIMEOUT
provisioning.h, 284
PROV_WPS_PIN_TIMEOUT
provisioning.h, 284
PROV_WPS_SESSION_STARTED
provisioning.h, 284
PROV_WPS_SESSION_SUCCESSFUL
provisioning.h, 284
PROV_WPS_SESSION_UNSUCCESSFUL
provisioning.h, 284
PROV_WPS_SSID_SELECT_REQ
provisioning.h, 284
PROVISIONING_STATE_ACTIVE
provisioning.h, 284
PROVISIONING_STATE_DONE
provisioning.h, 284
PROVISIONING_STATE_INITIALIZING
provisioning.h, 284
PROVISIONING_STATE_SET_CONFIG
provisioning.h, 284
PROVISIONING_STATE_STOP_REQUESTED
provisioning.h, 284
PROVISIONING_STATE_STOPPED
provisioning.h, 284
PSM_CACHING_ENABLED
psm-v2.h, 294
PSM_INDEXING_ENABLED
psm-v2.h, 294
PSM_MODE_READ
psm-v2.h, 294
PSM_MODE_WRITE
psm-v2.h, 294
PROVISIONING_WLANNW
provisioning.h, 283
PROVISIONING_WPS
provisioning.h, 283
PSM_CREAT
psm.h, 295
parent
jsontok_t, 49
parser
jobj_t, 48
part_get_active_partition
partition.h, 275
part_get_layout_by_id
partition.h, 274
part_get_layout_by_name
partition.h, 274
part_init
partition.h, 274
part_set_active_partition
partition.h, 275
part_to_flash_desc
partition.h, 274
partition.h
FC_COMP_BOOT2, 275
FC_COMP_FTFS, 276

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

FC_COMP_FW, 275
FC_COMP_PSM, 276
FC_COMP_USER_APP, 276
FC_COMP_WLAN_FW, 275
partition.h, 273
flash_comp, 275
part_get_active_partition, 275
part_get_layout_by_id, 274
part_get_layout_by_name, 274
part_init, 274
part_set_active_partition, 275
part_to_flash_desc, 274
partition_entries_no
partition_table, 59
partition_entry, 58
device, 58
gen_level, 58
name, 58
size, 58
start, 58
type, 58
partition_table, 59
crc, 59
gen_level, 59
magic, 59
partition_entries_no, 59
version, 59
payload
_wifi_mgmt_frame_tx, 32
pe
update_desc, 66
perDmaInter
dma_config_type, 36
pin_generator
p2p_config, 55
wps_config, 77
pin_len
p2p_config, 56
wps_config, 78
pinmux_drv_close
mdev_pinmux.h, 225
pinmux_drv_get_gpio_func
mdev_pinmux.h, 226
pinmux_drv_init
mdev_pinmux.h, 225
pinmux_drv_open
mdev_pinmux.h, 225
pinmux_drv_setfunc
mdev_pinmux.h, 226
pk
upgrade_struct_t, 68
pm_change_event
pwrmgr.h, 298
pm_cli_init
pwrmgr.h, 300
pm_deregister_cb
pwrmgr.h, 298
pm_ieeeps_hs_cfg
wlan.h, 332
pm_init
pwrmgr.h, 298
pm_mcu_cfg
mdev_pm.h, 228
pm_mcu_cli_init
pwrmgr.h, 300
pm_mcu_get_mode
mdev_pm.h, 229
pm_mcu_io_cfg
mdev_pm.h, 229
pm_mcu_is_enabled
mdev_pm.h, 229
pm_mcu_state
mdev_pm.h, 228
pm_reboot_soc
pwrmgr.h, 299
pm_register_cb
pwrmgr.h, 297
pm_wakeup_source
mdev_pm.h, 229
pmk
wlan_network_security, 74
pmk_valid
wlan_network_security, 74
pmu_drv_powerdeepoff_vddio
mdev_pm.h, 231
pmu_drv_powernormal_vddio
mdev_pm.h, 231
pmu_drv_poweroff_vddio
mdev_pm.h, 230
pmu_drv_poweron_vddio
mdev_pm.h, 230
port
mdns_service, 51
power_save_event_t
pwrmgr.h, 301
power_save_state_change_cb_fn
pwrmgr.h, 301
power_state_t
mdev_pm.h, 232
primary_dev_category
p2p_config, 56
wps_config, 77
primary_dev_subcategory
p2p_config, 56
wps_config, 77
prime
DH_PG_PARAMS, 35
primeLen
DH_PG_PARAMS, 35
probe_req_info
wlan.h, 345
product
httpd_useragent_t, 45
proto
mdns_service, 51
protocol

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

http_resp_t, 39
prov_event
provisioning.h, 283
prov_ezconnect_finish
provisioning.h, 278
prov_ezconnect_start
provisioning.h, 278
prov_finish
provisioning.h, 278
prov_get_lock_scan_results
provisioning.h, 281
prov_get_scan_config
provisioning.h, 278
prov_get_scan_count
provisioning.h, 282
prov_get_scan_result
provisioning.h, 282
prov_mode
provisioning_config, 60
prov_put_lock_scan_results
provisioning.h, 282
prov_register_secure_provisioning_pin
provisioning.h, 280
prov_scan_config, 60
scan_interval, 60
wifi_scan_params, 60
prov_scan_policy
provisioning.h, 283
prov_set_network
provisioning.h, 282
prov_set_scan_config
provisioning.h, 278
prov_set_scan_config_no_cb
provisioning.h, 279
prov_set_scan_policy
provisioning.h, 281
prov_start
provisioning.h, 277
prov_unregister_secure_provisioning_pin
provisioning.h, 280
provisioning.h
PROV_ABORTED, 284
PROV_MICRO_AP_UP_REQUESTED, 283
PROV_NETWORK_SET_CB, 283
PROV_ON_DEMAND_SCAN, 283
PROV_PERIODIC_SCAN, 283
PROV_RESET_TO_SMC, 283
PROV_SCAN_CFG_SET_CB, 284
PROV_START, 283
PROV_START_SMART_MODE, 283
PROV_STOP_SMART_MODE, 283
PROV_WPS_PBC_TIMEOUT, 284
PROV_WPS_PIN_TIMEOUT, 284
PROV_WPS_SESSION_STARTED, 284
PROV_WPS_SESSION_SUCCESSFUL, 284
PROV_WPS_SESSION_UNSUCCESSFUL, 284
PROV_WPS_SSID_SELECT_REQ, 284
PROVISIONING_STATE_ACTIVE, 284
PROVISIONING_STATE_DONE, 284
PROVISIONING_STATE_INITIALIZING, 284
PROVISIONING_STATE_SET_CONFIG, 284
PROVISIONING_STATE_STOP_REQUESTED, 284
PROVISIONING_STATE_STOPPED, 284
provisioning.h, 276
PROVISIONING_WLANNW, 283
PROVISIONING_WPS, 283
prov_event, 283
prov_ezconnect_finish, 278
prov_ezconnect_start, 278
prov_finish, 278
prov_get_lock_scan_results, 281
prov_get_scan_config, 278
prov_get_scan_count, 282
prov_get_scan_result, 282
prov_put_lock_scan_results, 282
prov_register_secure_provisioning_pin, 280
prov_scan_policy, 283
prov_set_network, 282
prov_set_scan_config, 278
prov_set_scan_config_no_cb, 279
prov_set_scan_policy, 281
prov_start, 277
prov_unregister_secure_provisioning_pin, 280
provisioning_state, 284
register_ezconnect_provisioning_ssid, 280
register_provisioning_web_handlers, 279
register_secure_provisioning_web_handlers, 279
set_network_cb_register, 281
sys_wq_dequeue_job_wlan_scan, 281
sys_wq_enqueue_job_wlan_scan, 280
unregister_provisioning_web_handlers, 279
unregister_secure_provisioning_web_handlers, 280
verify_scan_interval_value, 279
provisioning_config, 60
prov_mode, 60
provisioning_event_handler, 60
provisioning_event_handler
provisioning_config, 60
provisioning_state
provisioning.h, 284
psk
wlan_network_security, 74
psk_len
wlan_network_security, 74
psm-v2.h
PSM_CACHING_ENABLED, 294
PSM_INDEXING_ENABLED, 294
PSM_MODE_READ, 294
PSM_MODE_WRITE, 294
psm-v2.h, 284
psm_flags_t, 294
psm_format, 291
psm_get_variable, 292
psm_get_variable_int, 293

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

psm_get_variable_str, 293
psm_hnd_t, 294
psm_mode_t, 294
psm_module_deinit, 294
psm_module_init, 287
psm_object_close, 290
psm_object_delete, 290
psm_object_handle_t, 294
psm_object_open, 288
psm_object_read, 289
psm_object_write, 289
psm_objects_dump, 291
psm_set_variable, 291
psm_set_variable_int, 292
psm_set_variable_str, 292
psm.h, 295
PSM_CREAT, 295
psm_cfg_t, 61
read_only, 61
secure, 61
psm_flags_t
psm-v2.h, 294
psm_format
psm-v2.h, 291
psm_get_variable
psm-v2.h, 292
psm_get_variable_int
psm-v2.h, 293
psm_get_variable_str
psm-v2.h, 293
psm_hnd_t
psm-v2.h, 294
psm_mode_t
psm-v2.h, 294
psm_module_deinit
psm-v2.h, 294
psm_module_init
psm-v2.h, 287
psm_objattrib_t, 61
flags, 62
psm_object_close
psm-v2.h, 290
psm_object_delete
psm-v2.h, 290
psm_object_handle_t
psm-v2.h, 294
psm_object_open
psm-v2.h, 288
psm_object_read
psm-v2.h, 289
psm_object_write
psm-v2.h, 289
psm_objects_dump
psm-v2.h, 291
psm_set_variable
psm-v2.h, 291
psm_set_variable_int
psm-v2.h, 292
psm_set_variable_str
psm-v2.h, 292
ptrname
mdns_service, 51
push_button.h, 295
MAX_PB_CB, 297
push_button_reset_cb, 296
push_button_set_cb, 295
push_button_reset_cb
push_button.h, 296
push_button_set_cb
push_button.h, 295
put_handler
httpd_wsgi_call, 46
pwrmgr.h
ACTION_ENTER_PM2, 301
ACTION_ENTER_PM3, 301
ACTION_ENTER_PM4, 301
ACTION_EXIT_PM2, 301
ACTION_EXIT_PM3, 301
pwrmgr.h, 297
is_pm_init_done, 299
pm_change_event, 298
pm_cli_init, 300
pm_deregister_cb, 298
pm_init, 298
pm_mcu_cli_init, 300
pm_reboot_soc, 299
pm_register_cb, 297
power_save_event_t, 301
power_save_state_change_cb_fn, 301
wakelock_get, 299
wakelock_isheld, 300
wakelock_put, 299

QOS_DATA
wlan.h, 344
qos_ctrl
wlan.h, 345

REANNOUNCE
mdns.h, 268
ROLE_AUTO_GO
app_framework.h, 103
ROLE_CLIENT
app_framework.h, 103
ROLE_GO
app_framework.h, 103
ROLE_UNDECIDED
app_framework.h, 103
random_hdlr_t
wm_utils.h, 379
random_initialize_seed
wm_utils.h, 377
random_register_handler
wm_utils.h, 376
random_register_seed_handler
wm_utils.h, 377
random_unregister_handler

Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

wm_utils.h, 376
random_unregister_seed_handler
wm_utils.h, 377
raw_low_level_output
wm_net.h, 355
read_only
psm_cfg_t, 61
reason_phrase
http_resp_t, 39
register_ezconnect_provisioning_ssid
provisioning.h, 280
register_provisioning_web_handlers
provisioning.h, 279
register_secure_provisioning_web_handlers
provisioning.h, 279
remaining
update_desc, 67
remaining_bytes
httpd_request_t, 42
resource
http_req_t, 38
retry_cnt
httpc_cfg_t, 41
rf_bands
p2p_config, 56
wps_config, 77
rfget.h
WM_E_RFGET_F2MSL, 302
WM_E_RFGET_FFW_LOAD, 302
WM_E_RFGET_FHTTP_PARSE, 302
WM_E_RFGET_FHTTP_REQ, 302
WM_E_RFGET_FINIT, 302
WM_E_RFGET_FRD, 302
WM_E_RFGET_FSOCK, 302
WM_E_RFGET_FUPDATE, 302
WM_E_RFGET_FWR, 302
WM_E_RFGET_INV_IMG, 302
WM_E_RFGET_INVHTTP_STATUS, 302
WM_E_RFGET_INVLEN, 302
WM_E_RFGET_INVPRO, 302
rfget.h, 301
wm_rfget_code, 302
rfget_cli_init
Other RFGET APIs, 29
rfget_client_mode_update
High level RFGET APIs, 28
rfget_ftfs_init
Other RFGET APIs, 29
rfget_get_passive_firmware
Low level RFGET APIs, 22
rfget_get_passive_wifi_firmware
Low level RFGET APIs, 22
rfget_init
Low level RFGET APIs, 20
rfget_is_update_in_progress
Low level RFGET APIs, 22
rfget_server_mode_update
High level RFGET APIs, 27
rfget_update_abort
Low level RFGET APIs, 21
rfget_update_begin
Low level RFGET APIs, 20
rfget_update_complete
Low level RFGET APIs, 21
rfget_update_data
Low level RFGET APIs, 21
role
app_p2p_role_data, 34
p2p_config, 55
wlan_network, 72
wlan_scan_result, 75
wps_config, 77
rsa2048_aes128_client_mode_upgrade_fw
fw_upgrade_rsa_aes.h, 130
rsa2048_aes128_server_mode_upgrade_fw
fw_upgrade_rsa_aes.h, 130
rssi
wlan_scan_result, 76
rst_cause
app_init_state, 33
rtc_drv_close
mdev_rtc.h, 236
rtc_drv_get
mdev_rtc.h, 235
rtc_drv_get_uppval
mdev_rtc.h, 235
rtc_drv_init
mdev_rtc.h, 233
rtc_drv_open
mdev_rtc.h, 233
rtc_drv_reset
mdev_rtc.h, 234
rtc_drv_set
mdev_rtc.h, 234
rtc_drv_set_alarm
mdev_rtc.h, 235
rtc_drv_set_alarm_cb
mdev_rtc.h, 234
rtc_drv_set_cb
mdev_rtc.h, 234
rtc_drv_start
mdev_rtc.h, 235
rtc_drv_stop
mdev_rtc.h, 235

SUPP_FAILED
wm_supplicant.h, 375
SUPP_FINISHED
wm_supplicant.h, 375
SUPP_SESSION_FAILED
wm_supplicant.h, 375
SUPP_SESSION_STARTED
wm_supplicant.h, 375
SUPP_SESSION_SUCCESSFUL
wm_supplicant.h, 375
SUPP_SESSION_TIMEOUT
wm_supplicant.h, 375

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

SUPP_STARTED
wm_supplicant.h, 375
SHA256_BLOCK_SIZE
wmcrypto.h, 384
SHA256_DIGEST_SIZE
wmcrypto.h, 384
SSP_DMA
mdev_ssp.h, 244
sample_initialise_random_seed
wm_utils.h, 378
scan_interval
prov_scan_config, 60
sdio_drv_creg_read
mdev_sdio.h, 237
sdio_drv_creg_write
mdev_sdio.h, 237
sdio_drv_deinit
mdev_sdio.h, 238
sdio_drv_init
mdev_sdio.h, 238
sdio_drv_open
mdev_sdio.h, 238
sdio_drv_read
mdev_sdio.h, 237
sdio_drv_set_cb1
mdev_sdio.h, 239
sdio_drv_write
mdev_sdio.h, 238
secure
psm_cfg_t, 61
secure_upgrade.h, 302
client_mode_secure_upgrade, 304
decrypt_func_t, 305
decrypt_init_func_t, 304
fw_data_read_fn_t, 306
hash_finish_func_t, 305
hash_update_func_t, 305
server_mode_secure_upgrade, 304
signature_verify_func_t, 305
security
wlan_network, 72
security_specific
wlan_network, 73
seq_ctl
_wifi_mgmt_frame_tx, 32
seq_frag_num
wlan.h, 345
serial_number
p2p_config, 56
wps_config, 77
server
http_resp_t, 39
tls_init_config_t, 65
server_cert
httpd_tls_certs_t, 44
tls_init_config_t, 64
server_cert_chained
httpd_tls_certs_t, 44
server_cert_size
httpd_tls_certs_t, 44
tls_init_config_t, 64
server_key
httpd_tls_certs_t, 44
tls_init_config_t, 64
server_key_size
httpd_tls_certs_t, 44
tls_init_config_t, 65
server_mode_secure_upgrade
secure_upgrade.h, 304
servname
mdns_service, 50
servtype
mdns_service, 50
session_priv_data
upgrade_struct_t, 69
set_handler
httpd_wsgi_call, 46
set_network_cb_register
provisioning.h, 281
signature
upgrade_struct_t, 68
signature_len
upgrade_struct_t, 68
signature_verify
upgrade_struct_t, 67
signature_verify_func_t
secure_upgrade.h, 305
size
jsontok_t, 49
os_queue_pool, 52
os_thread_stack, 52
partition_entry, 58
sock
httpd_request_t, 42
soft_crc32
wm_utils.h, 378
src
wlan.h, 344
ssid
wlan.h, 345
wlan_network, 72
wlan_scan_result, 75
ssid_element_id
wlan.h, 345
ssid_len
wlan.h, 345
wlan_scan_result, 75
ssid_specific
wlan_network, 72
ssp_drv_close
mdev_ssp.h, 241
ssp_drv_cs_activate
mdev_ssp.h, 242
ssp_drv_cs_deactivate
mdev_ssp.h, 242
ssp_drv_deinit

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

mdev_ssp.h, 240
ssp_drv_init
mdev_ssp.h, 240
ssp_drv_open
mdev_ssp.h, 241
ssp_drv_read
mdev_ssp.h, 242
ssp_drv_rxbuf_size
mdev_ssp.h, 243
ssp_drv_set_clk
mdev_ssp.h, 243
ssp_drv_timeout
mdev_ssp.h, 243
ssp_drv_write
mdev_ssp.h, 241
start
jsontok_t, 49
partition_entry, 58
start_freq
wifi-decl.h, 310
status_code
http_resp_t, 39
strdup
wm_utils.h, 378
sub_band
wifi_domain_param_t, 69
supp_event
wm_supplicant.h, 375
supplicant_finish
wm_supplicant.h, 375
supplicant_send_cmd
wm_supplicant.h, 375
supplicant_start
wm_supplicant.h, 374
sys_wq_dequeue_job_wlan_scan
provisioning.h, 281
sys_wq_enqueue_job_wlan_scan
provisioning.h, 280
sysinfo_init
wmsysinfo.h, 390

TLS_ENABLE
httpc.h, 145
telnetd.h
WM_E_TELNETD_ACCEPT_SOCKET, 307
WM_E_TELNETD_BIND_SOCKET, 307
WM_E_TELNETD_LISTEN_SOCKET, 307
WM_E_TELNETD_NONBLOCKING_SOCKET, 307
WM_E_TELNETD_SERVER_RUNNING, 307
WM_E_TELNETD_SOCKET, 307
WM_E_TLENETD_THREAD_CREATE, 307
telnetd.h, 306
telnetd_start, 306
telnetd_stop, 306
wm_telnetd_errno, 307
telnetd_start
telnetd.h, 306
telnetd_stop
telnetd.h, 306
timestamp
wlan.h, 345
tls
tls_init_config_t, 65
tls_client_t, 62
ca_cert, 62
ca_cert_size, 62
client_cert, 62
client_cert_is_chained, 62
client_cert_size, 63
client_key, 63
client_key_size, 63
tls_close
wm-tls.h, 350
tls_create_client_context
wm-tls.h, 349
tls_create_server_context
wm-tls.h, 348
tls_flags_t
wm-tls.h, 351
tls_get_error
wm-tls.h, 350
tls_handle_t
wm-tls.h, 351
tls_init_config_t, 63
ca_cert, 64
ca_cert_size, 64
client, 64
client_cert, 64
client_cert_size, 64
client_key, 64
client_key_size, 64
flags, 64
server, 65
server_cert, 64
server_cert_size, 64
server_key, 64
server_key_size, 65
tls, 65
tls_lib_init
wm-tls.h, 348
tls_pending
wm-tls.h, 350
tls_purge_client_context
wm-tls.h, 349
tls_purge_server_context
wm-tls.h, 349
tls_recv
wm-tls.h, 350
tls_send
wm-tls.h, 350
tls_session_init
wm-tls.h, 348
tm, 65
tm_hour, 65
tm_mday, 66
tm_min, 65

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

tm_mon, 66
tm_sec, 65
tm_wday, 66
tm_year, 66
tm_hour
tm, 65
tm_mday
tm, 66
tm_min
tm, 65
tm_mon
tm, 66
tm_sec
tm, 65
tm_wday
tm, 66
tm_year
tm, 66
tokens
jobj_t, 48
tx_power
wifi-decl.h, 310
type
http_req_t, 38
httpd_request_t, 42
input_gpio_cfg_t, 46
jsontok_t, 49
output_gpio_cfg_t, 53
partition_entry, 58
wlan_network, 72
wlan_network_security, 74
wlan_scan_result, 75

u32
wmcrypto.h, 384
u8
wmcrypto.h, 384
UART_DMA_DISABLE
mdev_uart.h, 249
UART_DMA_ENABLE
mdev_uart.h, 249
UP
mdns.h, 268
USB_CDC_CLASS
mdev_usb.h, 252
USB_HID_CLASS
mdev_usb.h, 252
USB_MSC_CLASS
mdev_usb.h, 252
UART_8BIT
mdev_uart.h, 249
UART_9BIT
mdev_uart.h, 249
UART_DMA_MODE
mdev_uart.h, 249
USB_CDC_DEBUG
mdev_usb_host.h, 258
USB_UVC_DEBUG
mdev_usb_host.h, 257
uart_drv_blocking_read
mdev_uart.h, 248
uart_drv_close
mdev_uart.h, 246
uart_drv_deinit
mdev_uart.h, 245
uart_drv_dma_rd_blk_size
mdev_uart.h, 248
uart_drv_get_portid
mdev_uart.h, 247
uart_drv_init
mdev_uart.h, 245
uart_drv_open
mdev_uart.h, 246
uart_drv_read
mdev_uart.h, 246
uart_drv_rx_buf_reset
mdev_uart.h, 249
uart_drv_rxbuf_size
mdev_uart.h, 247
uart_drv_set_opts
mdev_uart.h, 248
uart_drv_tx_flush
mdev_uart.h, 247
uart_drv_write
mdev_uart.h, 246
uart_drv_xfer_mode
mdev_uart.h, 245
ucstCipher
wlan_network_security, 74
unregister_provisioning_web_handlers
provisioning.h, 279
unregister_secure_provisioning_web_handlers
provisioning.h, 280
update_and_validate_firmware
Middle level RFGET APIs, 24
update_and_validate_fs
Middle level RFGET APIs, 25
update_and_validate_wifi_firmware
Middle level RFGET APIs, 24
update_desc, 66
addr, 66
dev, 66
pe, 66
remaining, 67
update_desc_t
Low level RFGET APIs, 20
update_firmware
Middle level RFGET APIs, 25
update_wifi_firmware
Middle level RFGET APIs, 26
upgrade_struct_t, 67
bytes_to_read, 69
decrypt, 67
decrypt_ctx, 68
decrypt_init, 67
decrypt_key, 69
fw_data_read_fn, 68

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

hash, 68
hash_ctx, 68
hash_finish, 68
hash_len, 68
hash_update, 68
iv, 68
iv_len, 68
pk, 68
session_priv_data, 69
signature, 68
signature_len, 68
signature_verify, 67
uri
httpd_wsgi_call, 45
usb_class
mdev_usb.h, 252
usb_drv_close
mdev_usb.h, 251
usb_drv_init
mdev_usb.h, 251
usb_drv_open
mdev_usb.h, 251
usb_drv_read
mdev_usb.h, 251
usb_drv_write
mdev_usb.h, 252
usb_host_cdc_drv_init
mdev_usb_host.h, 254
usb_host_drv_block_read
mdev_usb_host.h, 257
usb_host_drv_block_write
mdev_usb_host.h, 257
usb_host_drv_close
mdev_usb_host.h, 257
usb_host_drv_info
mdev_usb_host.h, 256
usb_host_drv_open
mdev_usb_host.h, 255
usb_host_drv_read
mdev_usb_host.h, 256
usb_host_drv_start
mdev_usb_host.h, 255
usb_host_drv_stop
mdev_usb_host.h, 255
usb_host_drv_write
mdev_usb_host.h, 256
usb_host_msc_drv_init
mdev_usb_host.h, 254
usb_host_uac_drv_init
mdev_usb_host.h, 254
usb_host_uvc_drv_init
mdev_usb_host.h, 254

verify_load_firmware
Low level RFGET APIs, 22
verify_scan_interval_value
provisioning.h, 279
version
http_req_t, 38
http_resp_t, 39
httpd_useragent_t, 45
p2p_config, 55
partition_table, 59
wps_config, 77
version2
p2p_config, 55
wps_config, 77

WAKE_ON_ALL_BROADCAST
wlan.h, 342
WAKE_ON_ARP_BROADCAST
wlan.h, 342
WAKE_ON_MAC_EVENT
wlan.h, 342
WAKE_ON_MGMT_FRAME
wlan.h, 342
WAKE_ON_MULTICAST
wlan.h, 342
WAKE_ON_UNICAST
wlan.h, 342
WIFI_ACTIVE
wifi_events.h, 311
WIFI_DEEP_SLEEP
wifi_events.h, 311
WIFI_INACTIVE
wifi_events.h, 311
WIFI_PDN
wifi_events.h, 311
WIFI_UNINIT
wifi_events.h, 311
WIFI_WAKE_ON_ALL_BROADCAST
wifi_events.h, 311
WIFI_WAKE_ON_ARP_BROADCAST
wifi_events.h, 311
WIFI_WAKE_ON_MAC_EVENT
wifi_events.h, 311
WIFI_WAKE_ON_MGMT_FRAME
wifi_events.h, 311
WIFI_WAKE_ON_MULTICAST
wifi_events.h, 311
WIFI_WAKE_ON_UNICAST
wifi_events.h, 311
WLAN_ACTIVE
wlan.h, 343
WLAN_ASSOCIATED
wlan.h, 343
WLAN_ASSOCIATING
wlan.h, 343
WLAN_BSS_ROLE_ANY
wifi_events.h, 311
WLAN_BSS_ROLE_STA
wifi_events.h, 311
WLAN_BSS_ROLE_UAP
wifi_events.h, 311
WLAN_BSSID_NOT_FOUND_IN_SCAN_LIST
wlan.h, 341
WLAN_CONNECTED
wlan.h, 343

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

WLAN_CONNECTING
wlan.h, 342
WLAN_DEEP_SLEEP
wlan.h, 343
WLAN_DISCONNECTED
wlan.h, 342
WLAN_ERROR_CARD_NOT_DETECTED
wlan.h, 341
WLAN_ERROR_FW_DNLD_FAILED
wlan.h, 341
WLAN_ERROR_FW_NOT_DETECTED
wlan.h, 341
WLAN_ERROR_FW_NOT_READY
wlan.h, 341
WLAN_IEEE
wlan.h, 343
WLAN_P2P_ACTIVE
wlan.h, 343
WLAN_P2P_DTIM_SLEEP
wlan.h, 343
WLAN_P2P_INACTIVITY_SLEEP
wlan.h, 343
WLAN_P2P_STARTED
wlan.h, 343
WLAN_P2P_STOPPED
wlan.h, 343
WLAN_PDN
wlan.h, 343
WLAN_REASON_ADDRESS_FAILED
wlan.h, 342
WLAN_REASON_ADDRESS_SUCCESS
wlan.h, 342
WLAN_REASON_CHAN_SWITCH
wlan.h, 342
WLAN_REASON_CONNECT_FAILED
wlan.h, 341
WLAN_REASON_INITIALIZATION_FAILED
wlan.h, 342
WLAN_REASON_INITIALIZED
wlan.h, 342
WLAN_REASON_LINK_LOST
wlan.h, 342
WLAN_REASON_NETWORK_AUTH_FAILED
wlan.h, 341
WLAN_REASON_NETWORK_NOT_FOUND
wlan.h, 341
WLAN_REASON_PS_ENTER
wlan.h, 342
WLAN_REASON_PS_EXIT
wlan.h, 342
WLAN_REASON_SUCCESS
wlan.h, 341
WLAN_REASON_UAP_CLIENT_ASSOC
wlan.h, 342
WLAN_REASON_UAP_CLIENT_DISSOC
wlan.h, 342
WLAN_REASON_UAP_START_FAILED
wlan.h, 342
WLAN_REASON_UAP_STOP_FAILED
wlan.h, 342
WLAN_REASON_UAP_STOPPED
wlan.h, 342
WLAN_REASON_UAP_SUCCESS
wlan.h, 342
WLAN_REASON_USER_DISCONNECT
wlan.h, 342
WLAN_REASON_WPS_DISCONNECT
wlan.h, 342
WLAN_SCANNING
wlan.h, 343
WLAN_SECURITY_EAP_TLS
wlan.h, 344
WLAN_SECURITY_NONE
wlan.h, 344
WLAN_SECURITY_WEP_OPEN
wlan.h, 344
WLAN_SECURITY_WEP_SHARED
wlan.h, 344
WLAN_SECURITY_WILDCARD
wlan.h, 344
WLAN_SECURITY_WPA
wlan.h, 344
WLAN_SECURITY_WPA2
wlan.h, 344
WLAN_SECURITY_WPA_WPA2_MIXED
wlan.h, 344
WLAN_UAP_ACTIVE
wlan.h, 343
WLAN_UAP_DTIM_SLEEP
wlan.h, 343
WLAN_UAP_INACTIVITY_SLEEP
wlan.h, 343
WLAN_UAP_STARTED
wlan.h, 343
WLAN_UAP_STOPPED
wlan.h, 343
WM_E_AF_NW_ADD
app_framework.h, 101
WM_E_AF_NW_CONN
app_framework.h, 101
WM_E_AF_NW_DEL
app_framework.h, 101
WM_E_AF_NW_RD
app_framework.h, 101
WM_E_AF_NW_WR
app_framework.h, 101
WM_E_AF_PSM_INIT
app_framework.h, 101
WM_E_AF_REB_TMR_WR
app_framework.h, 101
WM_E_AF_WD_INIT
app_framework.h, 101
WM_E_AF_WLCMGR_FAIL
app_framework.h, 101
WM_E_DHCPD_ARP_SEND
dhcp-server.h, 115

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

WM_E_DHCPD_BCAST_ADDR
dhcp-server.h, 115
WM_E_DHCPD_BUFFER_FULL
dhcp-server.h, 115
WM_E_DHCPD_DNS_IGNORE
dhcp-server.h, 115
WM_E_DHCPD_INCORRECT_HEADER
dhcp-server.h, 115
WM_E_DHCPD_INIT
dhcp-server.h, 115
WM_E_DHCPD_INVALID_INPUT
dhcp-server.h, 115
WM_E_DHCPD_INVALID_OPCODE
dhcp-server.h, 115
WM_E_DHCPD_IOCTL_CALL
dhcp-server.h, 115
WM_E_DHCPD_IP_ADDR
dhcp-server.h, 115
WM_E_DHCPD_MUTEX_CREATE
dhcp-server.h, 115
WM_E_DHCPD_NETMASK
dhcp-server.h, 115
WM_E_DHCPD_REGISTER_CMDS
dhcp-server.h, 115
WM_E_DHCPD_RESP_SEND
dhcp-server.h, 115
WM_E_DHCPD_SERVER_RUNNING
dhcp-server.h, 115
WM_E_DHCPD_SOCKET
dhcp-server.h, 115
WM_E_DHCPD_SPOOF_NAME
dhcp-server.h, 115
WM_E_DHCPD_THREAD_CREATE
dhcp-server.h, 115
WM_E_EXML_INBUF
ezxml.h, 122
WM_E_EXML_INLEN
ezxml.h, 122
WM_E_FTFS_INIT_FAIL
ftfs.h, 127
WM_E_FTFS_INVALID_FILE
ftfs.h, 127
WM_E_FTFS_SEEK_FAIL
ftfs.h, 127
WM_E_HTTPC_BAD_REQUEST
httpc.h, 145
WM_E_HTTPC_FILE_NOT_FOUND
httpc.h, 145
WM_E_HTTPC_SOCKET_ERROR
httpc.h, 145
WM_E_HTTPC_SOCKET_SHUTDOWN
httpc.h, 145
WM_E_HTTPC_TCP_CONNECT_FAIL
httpc.h, 145
WM_E_HTTPC_TLS_NOT_ENABLED
httpc.h, 145
WM_E_HTTPD_DATA_RD
httpd.h, 165
WM_E_HTTPD_HANDLER_400
httpd.h, 165
WM_E_HTTPD_HANDLER_404
httpd.h, 165
WM_E_HTTPD_HANDLER_500
httpd.h, 165
WM_E_HTTPD_HDR_FNAME
httpd.h, 165
WM_E_HTTPD_NO_HANDLER
httpd.h, 165
WM_E_HTTPD_NOTSUPP
httpd.h, 165
WM_E_HTTPD_SSI_MAX
httpd.h, 165
WM_E_HTTPD_TOOLONG
httpd.h, 165
WM_E_JSON_FAIL
jsonv2.h, 182
WM_E_JSON_INCOMPLETE
jsonv2.h, 182
WM_E_JSON_INVAL
jsonv2.h, 182
WM_E_JSON_INVALID_INDEX
jsonv2.h, 183
WM_E_JSON_INVALID_JARRAY
jsonv2.h, 183
WM_E_JSON_INVALID_JOBJ
jsonv2.h, 183
WM_E_JSON_INVALID_TYPE
jsonv2.h, 183
WM_E_JSON_INVSTART
jsonv2.h, 183
WM_E_JSON_NOMEM
jsonv2.h, 182
WM_E_JSON_NOT_FOUND
jsonv2.h, 183
WM_E_JSON_OBUF
jsonv2.h, 183
WM_E_MDNS_BADSRC
mdns.h, 268
WM_E_MDNS_FBIND
mdns.h, 268
WM_E_MDNS_FBINDTODEVICE
mdns.h, 268
WM_E_MDNS_FMCAST_JOIN
mdns.h, 268
WM_E_MDNS_FMCAST_SET
mdns.h, 268
WM_E_MDNS_FQUERY_SOC
mdns.h, 268
WM_E_MDNS_FQUERY_THREAD
mdns.h, 268
WM_E_MDNS_FREUSE
mdns.h, 268
WM_E_MDNS_FSOC
mdns.h, 268
WM_E_MDNS_INUSE
mdns.h, 268

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

WM_E_MDNS_INVAL
mdns.h, 268
WM_E_MDNS_NOIMPL
mdns.h, 268
WM_E_MDNS_NOMEM
mdns.h, 268
WM_E_MDNS_NORESP
mdns.h, 268
WM_E_MDNS_TOOBIG
mdns.h, 268
WM_E_RFGET_F2MSL
rfget.h, 302
WM_E_RFGET_FFW_LOAD
rfget.h, 302
WM_E_RFGET_FHTTP_PARSE
rfget.h, 302
WM_E_RFGET_FHTTP_REQ
rfget.h, 302
WM_E_RFGET_FINIT
rfget.h, 302
WM_E_RFGET_FRD
rfget.h, 302
WM_E_RFGET_FSOCK
rfget.h, 302
WM_E_RFGET_FUPDATE
rfget.h, 302
WM_E_RFGET_FWR
rfget.h, 302
WM_E_RFGET_INV_IMG
rfget.h, 302
WM_E_RFGET_INVHTTP_STATUS
rfget.h, 302
WM_E_RFGET_INVLEN
rfget.h, 302
WM_E_RFGET_INVPRO
rfget.h, 302
WM_E_TELNETD_ACCEPT_SOCKET
telnetd.h, 307
WM_E_TELNETD_BIND_SOCKET
telnetd.h, 307
WM_E_TELNETD_LISTEN_SOCKET
telnetd.h, 307
WM_E_TELNETD_NONBLOCKING_SOCKET
telnetd.h, 307
WM_E_TELNETD_SERVER_RUNNING
telnetd.h, 307
WM_E_TELNETD_SOCKET
telnetd.h, 307
WM_E_TLENETD_THREAD_CREATE
telnetd.h, 307
WPS_FINISHED
wps.h, 395
WPS_SESSION_ABORTED
wps.h, 394
WPS_SESSION_FAILED
wps.h, 395
WPS_SESSION_INACTIVE
wps.h, 395
WPS_SESSION_PBC
wps.h, 395
WPS_SESSION_PIN
wps.h, 395
WPS_SESSION_PIN_CHKSUM_FAILED
wps.h, 394
WPS_SESSION_STARTED
wps.h, 394
WPS_SESSION_SUCCESSFUL
wps.h, 395
WPS_SESSION_TIMEOUT
wps.h, 395
WPS_STARTED
wps.h, 394
WD_STROBE_INTERVAL
healthmon.h, 134
WKUP_INTERVAL
healthmon.h, 134
WLAN_ERROR_ACTION
wlan.h, 340
WLAN_ERROR_NOMEM
wlan.h, 340
WLAN_ERROR_NONE
wlan.h, 340
WLAN_ERROR_PARAM
wlan.h, 340
WLAN_ERROR_STATE
wlan.h, 340
WLAN_PMK_LENGTH
wlan.h, 340
WLAN_PSK_MAX_LENGTH
wlan.h, 340
WLAN_RESCAN_LIMIT
wlan.h, 339
WPS_LOG
wps.h, 394
wakelock_get
pwrmgr.h, 299
wakelock_isheld
pwrmgr.h, 300
wakelock_put
pwrmgr.h, 299
wakeup_cause
mdev_pm.h, 231
wdt_drv_init
mdev_wdt.h, 258
wdt_drv_open
mdev_wdt.h, 259
wdt_drv_set_timeout
mdev_wdt.h, 259
wdt_drv_start
mdev_wdt.h, 259
wdt_drv_strobe
mdev_wdt.h, 259
websockets.h, 307
ws_close_ctx, 308
ws_event_t, 308
ws_frame_recv, 308

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

ws_frame_send, 307
ws_frame_t, 308
ws_set_genmask_callback, 308
ws_upgrade_socket, 307
wep
wlan_scan_result, 75
Wi-Fi Region Configuration, 15
wlan_enable_11d, 15
wlan_get_country, 16
wlan_set_country, 16
wifi-decl.h, 309
BSS_TYPE_STA, 309
BSS_TYPE_UAP, 309
chan_desc, 310
chan_num, 310
chan_trpc_config, 310
chan_trpc_config_t, 309
chan_trpc_entry, 310
chan_trpc_t, 309
chan_width, 310
mod_group, 310
num_chans, 310
num_mod_grps, 310
start_freq, 310
tx_power, 310
wifi_chan_trpc_entry_t, 309
wifi_channel_desc_t, 309
wifi_events.h
WIFI_ACTIVE, 311
WIFI_DEEP_SLEEP, 311
WIFI_INACTIVE, 311
WIFI_PDN, 311
WIFI_UNINIT, 311
WIFI_WAKE_ON_ALL_BROADCAST, 311
WIFI_WAKE_ON_ARP_BROADCAST, 311
WIFI_WAKE_ON_MAC_EVENT, 311
WIFI_WAKE_ON_MGMT_FRAME, 311
WIFI_WAKE_ON_MULTICAST, 311
WIFI_WAKE_ON_UNICAST, 311
WLAN_BSS_ROLE_ANY, 311
WLAN_BSS_ROLE_STA, 311
WLAN_BSS_ROLE_UAP, 311
wifi_chan_trpc_entry_t
wifi-decl.h, 309
wifi_channel_desc_t
wifi-decl.h, 309
wifi_domain_param_t, 69
country_code, 69
no_of_sub_band, 69
sub_band, 69
wifi_event
wifi_events.h, 311
wifi_events.h, 311
wifi_event, 311
wifi_state, 311
wifi_wakeup_event_t, 311
wlan_bss_role, 311
wlan_bss_type, 311
wifi_scan_params
prov_scan_config, 60
wifi_state
wifi_events.h, 311
wifi_sub_band_set_t, 70
first_chan, 70
max_tx_pwr, 70
no_of_chan, 70
wifi_wakeup_event_t
wifi_events.h, 311
wlan.h
ADDR_TYPE_DHCP, 344
ADDR_TYPE_LLA, 344
ADDR_TYPE_STATIC, 344
AUTH, 344
BEACON, 344
DATA, 344
PROBE_REQ, 344
QOS_DATA, 344
WAKE_ON_ALL_BROADCAST, 342
WAKE_ON_ARP_BROADCAST, 342
WAKE_ON_MAC_EVENT, 342
WAKE_ON_MGMT_FRAME, 342
WAKE_ON_MULTICAST, 342
WAKE_ON_UNICAST, 342
WLAN_ACTIVE, 343
WLAN_ASSOCIATED, 343
WLAN_ASSOCIATING, 343
WLAN_BSSID_NOT_FOUND_IN_SCAN_LIST, 341
WLAN_CONNECTED, 343
WLAN_CONNECTING, 342
WLAN_DEEP_SLEEP, 343
WLAN_DISCONNECTED, 342
WLAN_ERROR_CARD_NOT_DETECTED, 341
WLAN_ERROR_FW_DNLD_FAILED, 341
WLAN_ERROR_FW_NOT_DETECTED, 341
WLAN_ERROR_FW_NOT_READY, 341
WLAN_IEEE, 343
WLAN_P2P_ACTIVE, 343
WLAN_P2P_DTIM_SLEEP, 343
WLAN_P2P_INACTIVITY_SLEEP, 343
WLAN_P2P_STARTED, 343
WLAN_P2P_STOPPED, 343
WLAN_PDN, 343
WLAN_REASON_ADDRESS_FAILED, 342
WLAN_REASON_ADDRESS_SUCCESS, 342
WLAN_REASON_CHAN_SWITCH, 342
WLAN_REASON_CONNECT_FAILED, 341
WLAN_REASON_INITIALIZATION_FAILED, 342
WLAN_REASON_INITIALIZED, 342
WLAN_REASON_LINK_LOST, 342
WLAN_REASON_NETWORK_AUTH_FAILED, 341
WLAN_REASON_NETWORK_NOT_FOUND, 341
WLAN_REASON_PS_ENTER, 342
WLAN_REASON_PS_EXIT, 342
WLAN_REASON_SUCCESS, 341

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

WLAN_REASON_UAP_CLIENT_ASSOC, 342
WLAN_REASON_UAP_CLIENT_DISSOC, 342
WLAN_REASON_UAP_START_FAILED, 342
WLAN_REASON_UAP_STOP_FAILED, 342
WLAN_REASON_UAP_STOPPED, 342
WLAN_REASON_UAP_SUCCESS, 342
WLAN_REASON_USER_DISCONNECT, 342
WLAN_REASON_WPS_DISCONNECT, 342
WLAN_SCANNING, 343
WLAN_SECURITY_EAP_TLS, 344
WLAN_SECURITY_NONE, 344
WLAN_SECURITY_WEP_OPEN, 344
WLAN_SECURITY_WEP_SHARED, 344
WLAN_SECURITY_WILDCARD, 344
WLAN_SECURITY_WPA, 344
WLAN_SECURITY_WPA2, 344
WLAN_SECURITY_WPA_WPA2_MIXED, 344
WLAN_UAP_ACTIVE, 343
WLAN_UAP_DTIM_SLEEP, 343
WLAN_UAP_INACTIVITY_SLEEP, 343
WLAN_UAP_STARTED, 343
WLAN_UAP_STOPPED, 343
wlan.h, 312
auth_info, 346
beacon_info, 345
beacon_interval, 345
bssid, 345
cap_info, 345
data_info, 346
dest, 344
duration, 344
frame_ctrl_flags, 344
frame_type, 345
is_sta_connected, 321
is_sta_ipv4_connected, 321
is_uap_started, 321
is_wfd_started, 321
load_wep_key, 332
MAX_IPV6_ADDRESSES, 340
pm_ieeeps_hs_cfg, 332
probe_req_info, 345
qos_ctrl, 345
seq_frag_num, 345
src, 344
ssid, 345
ssid_element_id, 345
ssid_len, 345
timestamp, 345
WLAN_ERROR_ACTION, 340
WLAN_ERROR_NOMEM, 340
WLAN_ERROR_NONE, 340
WLAN_ERROR_PARAM, 340
WLAN_ERROR_STATE, 340
WLAN_PMK_LENGTH, 340
WLAN_RESCAN_LIMIT, 339
wlan_add_network, 315
wlan_auth_info_t, 341
wlan_auto_reconnect_disable, 339
wlan_auto_reconnect_enable, 338
wlan_basic_cli_init, 330
wlan_beacon_info_t, 340
wlan_clear_mgmt_ie, 336
wlan_cli_init, 330
wlan_configure_listen_interval, 326
wlan_configure_null_pkt_interval, 326
wlan_connect, 316
wlan_connection_state, 342
wlan_data_info_t, 341
wlan_deepsleepps_off, 327
wlan_deepsleepps_on, 327
wlan_deinit, 315
wlan_disconnect, 317
wlan_enable_ed_mac_mode, 325
wlan_enhanced_cli_init, 331
wlan_event_reason, 341
wlan_frame_t, 341
wlan_frame_type_t, 344
wlan_get_11d_enable_status, 336
wlan_get_address, 319
wlan_get_auto_reconnect_config, 339
wlan_get_cal_data, 338
wlan_get_connection_state, 323
wlan_get_current_ant, 327
wlan_get_current_network, 320
wlan_get_current_signal_strength, 336
wlan_get_current_uap_network, 320
wlan_get_current_wfd_network, 321
wlan_get_firmware_version_ext, 328
wlan_get_mac_address, 318
wlan_get_mgmt_ie, 335
wlan_get_network, 322
wlan_get_network_byname, 322
wlan_get_network_count, 322
wlan_get_otp_user_data, 337
wlan_get_ps_mode, 329
wlan_get_scan_result, 324
wlan_get_sta_tx_power, 335
wlan_get_sta_tx_rate_index, 335
wlan_get_uap_address, 319
wlan_get_uap_connection_state, 323
wlan_get_uap_max_clients, 334
wlan_get_uap_ps_mode, 329
wlan_get_uap_supported_max_clients, 334
wlan_get_wfd_address, 320
wlan_get_wfd_connection_state, 323
wlan_get_wfd_mac_address, 318
wlan_get_wfd_ps_mode, 329
wlan_init, 314
wlan_initialize_uap_network, 315
wlan_inject_frame, 337
wlan_iw_cli_init, 331
wlan_pdnps_off, 327
wlan_pdnps_on, 327
wlan_pm_cli_init, 331
wlan_probe_req_info_t, 341
wlan_ps_mode, 343

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

wlan_remain_on_channel, 337
wlan_remove_network, 316
wlan_scan, 324
wlan_security_type, 343
wlan_set_cal_data, 325
wlan_set_current_ant, 328
wlan_set_mac_addr, 325
wlan_set_mgmt_ie, 336
wlan_set_sta_tx_power, 335
wlan_set_sta_tx_rate_index, 335
wlan_set_trpc, 338
wlan_set_uap_max_clients, 334
wlan_sniffer_start, 333
wlan_sniffer_status, 333
wlan_sniffer_stop, 333
wlan_start, 314
wlan_start_network, 317
wlan_stop, 315
wlan_stop_network, 318
wlan_uap_inact_sleep_ps_off, 329
wlan_uap_inact_sleep_ps_on, 328
wlan_uap_ps_mode, 343
wlan_uaputl_cli_init, 331
wlan_wakeup_event_t, 342
wlan_wfd_ps_mode, 343
wlan_wlcmgr_send_msg, 330
wm_wlan_errno, 341
wlan_11d.h
COUNTRY_AU, 347
COUNTRY_CA, 347
COUNTRY_CN, 347
COUNTRY_EU, 347
COUNTRY_FR, 347
COUNTRY_JP, 347
COUNTRY_KR, 347
COUNTRY_SG, 347
COUNTRY_US, 347
wlan_11d.h, 346
country_code_t, 347
wlan_11d_country_index_2_string, 347
wlan_set_domain_params, 346
wlan_set_region_code, 346
wlan_11d_country_index_2_string
wlan_11d.h, 347
wlan_add_network
wlan.h, 315
wlan_auth_info_t
wlan.h, 341
wlan_auto_reconnect_disable
wlan.h, 339
wlan_auto_reconnect_enable
wlan.h, 338
wlan_basic_cli_init
wlan.h, 330
wlan_beacon_info_t
wlan.h, 340
wlan_bss_role
wifi_events.h, 311
wlan_bss_type
wifi_events.h, 311
wlan_clear_mgmt_ie
wlan.h, 336
wlan_cli_init
wlan.h, 330
wlan_configure_listen_interval
wlan.h, 326
wlan_configure_null_pkt_interval
wlan.h, 326
wlan_connect
wlan.h, 316
wlan_connection_state
wlan.h, 342
wlan_data_info_t
wlan.h, 341
wlan_deepsleepps_off
wlan.h, 327
wlan_deepsleepps_on
wlan.h, 327
wlan_deinit
wlan.h, 315
wlan_disconnect
wlan.h, 317
wlan_enable_11d
Wi-Fi Region Configuration, 15
wlan_enable_ed_mac_mode
wlan.h, 325
wlan_enhanced_cli_init
wlan.h, 331
wlan_event_reason
wlan.h, 341
wlan_frame_t
wlan.h, 341
wlan_frame_type_t
wlan.h, 344
wlan_get_11d_enable_status
wlan.h, 336
wlan_get_address
wlan.h, 319
wlan_get_auto_reconnect_config
wlan.h, 339
wlan_get_cal_data
wlan.h, 338
wlan_get_connection_state
wlan.h, 323
wlan_get_country
Wi-Fi Region Configuration, 16
wlan_get_current_ant
wlan.h, 327
wlan_get_current_network
wlan.h, 320
wlan_get_current_signal_strength
wlan.h, 336
wlan_get_current_uap_network
wlan.h, 320
wlan_get_current_wfd_network
wlan.h, 321

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell
Confidential

wlan_get_firmware_version_ext
wlan.h, 328
wlan_get_mac_address
wlan.h, 318
wlan_get_mgmt_ie
wlan.h, 335
wlan_get_network
wlan.h, 322
wlan_get_network_byname
wlan.h, 322
wlan_get_network_count
wlan.h, 322
wlan_get_otp_user_data
wlan.h, 337
wlan_get_ps_mode
wlan.h, 329
wlan_get_scan_result
wlan.h, 324
wlan_get_sta_tx_power
wlan.h, 335
wlan_get_sta_tx_rate_index
wlan.h, 335
wlan_get_uap_address
wlan.h, 319
wlan_get_uap_connection_state
wlan.h, 323
wlan_get_uap_max_clients
wlan.h, 334
wlan_get_uap_ps_mode
wlan.h, 329
wlan_get_uap_supported_max_clients
wlan.h, 334
wlan_get_wfd_address
wlan.h, 320
wlan_get_wfd_connection_state
wlan.h, 323
wlan_get_wfd_mac_address
wlan.h, 318
wlan_get_wfd_ps_mode
wlan.h, 329
wlan_init
wlan.h, 314
wlan_initialize_uap_network
wlan.h, 315
wlan_inject_frame
wlan.h, 337
wlan_ip_config, 70
ipv4, 71
wlan_iw_cli_init
wlan.h, 331
wlan_network, 71
bssid, 72
bssid_specific, 73
channel, 72
channel_specific, 73
ip, 72
name, 72
role, 72
security, 72
security_specific, 73
ssid, 72
ssid_specific, 72
type, 72
wlan_network_security, 73
is_pmf_required, 74
mcstCipher, 74
pmk, 74
pmk_valid, 74
psk, 74
psk_len, 74
type, 74
ucstCipher, 74
wlan_pdnps_off
wlan.h, 327
wlan_pdnps_on
wlan.h, 327
wlan_pm_cli_init
wlan.h, 331
wlan_probe_req_info_t
wlan.h, 341
wlan_ps_mode
wlan.h, 343
wlan_remain_on_channel
wlan.h, 337
wlan_remove_network
wlan.h, 316
wlan_scan
wlan.h, 324
wlan_scan_result, 74
bssid, 75
channel, 75
role, 75
rssi, 76
ssid, 75
ssid_len, 75
type, 75
wep, 75
wmm, 75
wpa, 76
wpa2, 76
wlan_security_type
wlan.h, 343
wlan_set_cal_data
wlan.h, 325
wlan_set_country
Wi-Fi Region Configuration, 16
wlan_set_current_ant
wlan.h, 328
wlan_set_domain_params
wlan_11d.h, 346
wlan_set_mac_addr
wlan.h, 325
wlan_set_mgmt_ie
wlan.h, 336
wlan_set_region_code
wlan_11d.h, 346

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

wlan_set_sta_tx_power
wlan.h, 335
wlan_set_sta_tx_rate_index
wlan.h, 335
wlan_set_trpc
wlan.h, 338
wlan_set_uap_max_clients
wlan.h, 334
wlan_sniffer_start
wlan.h, 333
wlan_sniffer_status
wlan.h, 333
wlan_sniffer_stop
wlan.h, 333
wlan_start
wlan.h, 314
wlan_start_network
wlan.h, 317
wlan_stop
wlan.h, 315
wlan_stop_network
wlan.h, 318
wlan_uap_inact_sleep_ps_off
wlan.h, 329
wlan_uap_inact_sleep_ps_on
wlan.h, 328
wlan_uap_ps_mode
wlan.h, 343
wlan_uaputl_cli_init
wlan.h, 331
wlan_wakeup_event_t
wlan.h, 342
wlan_wfd_ps_mode
wlan.h, 343
wlan_wlcmgr_send_msg
wlan.h, 330
wm-tls.h, 347
ctaocrypt_lib_init, 348
tls_close, 350
tls_create_client_context, 349
tls_create_server_context, 348
tls_flags_t, 351
tls_get_error, 350
tls_handle_t, 351
tls_lib_init, 348
tls_pending, 350
tls_purge_client_context, 349
tls_purge_server_context, 349
tls_recv, 350
tls_send, 350
tls_session_init, 348
wm_os.h
OS_TIMER_AUTO_ACTIVATE, 374
OS_TIMER_NO_ACTIVATE, 374
OS_TIMER_ONE_SHOT, 374
OS_TIMER_PERIODIC, 374
wm_supplicant.h
SUPP_FAILED, 375
SUPP_FINISHED, 375
SUPP_SESSION_FAILED, 375
SUPP_SESSION_STARTED, 375
SUPP_SESSION_SUCCESSFUL, 375
SUPP_SESSION_TIMEOUT, 375
SUPP_STARTED, 375
wm_af_errno
app_framework.h, 101
wm_core_and_wlan_init
wm_wlan.h, 380
wm_dhcpd_errno
dhcp-server.h, 115
wm_exml_errno
ezxml.h, 121
wm_ftfs_errno
ftfs.h, 127
wm_httpc_errno
httpc.h, 145
wm_httpd_errno
httpd.h, 165
wm_json_errno
jsonv2.h, 182
wm_mdns_errno
mdns.h, 268
wm_net.h, 351
net_get_if_addr, 354
net_get_if_ip_addr, 354
net_get_if_ip_mask, 355
net_get_sta_handle, 353
net_get_uap_handle, 353
net_get_wfd_handle, 353
net_gethostbyname, 351
net_interface_dhcp_stop, 354
net_interface_down, 354
net_interface_up, 353
net_ipv4stack_init, 355
net_sock_to_interface, 352
netif_add_udp_broadcast_filter, 352
netif_remove_udp_broadcast_filter, 352
raw_low_level_output, 355
wm_os.h, 356
cb_fn, 374
OS_MUTEX_INHERIT, 372
OS_NO_WAIT, 372
OS_WAIT_FOREVER, 372
os_dump_mem_stats, 371
os_get_runtime_stats, 373
os_get_timestamp, 356
os_get_usec_counter, 356
os_heap_add_bank, 371
os_mem_alloc, 372
os_mem_calloc, 371
os_mem_free, 373
os_mem_realloc, 373
os_msec_to_ticks, 358
os_mutex_create, 362
os_mutex_delete, 364
os_mutex_get, 362

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

os_mutex_put, 363
os_queue_create, 359
os_queue_delete, 360
os_queue_get_msgs_waiting, 361
os_queue_pool_define, 372
os_queue_pool_t, 373
os_queue_recv, 360
os_queue_send, 359
os_recursive_mutex_create, 363
os_recursive_mutex_get, 364
os_recursive_mutex_put, 364
os_remove_idle_function, 361
os_remove_tick_function, 362
os_rwlock_create, 367
os_rwlock_delete, 367
os_rwlock_read_lock, 368
os_rwlock_read_unlock, 368
os_rwlock_write_lock, 367
os_rwlock_write_unlock, 368
os_semaphore_create, 365
os_semaphore_create_counting, 365
os_semaphore_delete, 366
os_semaphore_get, 365
os_semaphore_getcount, 366
os_semaphore_put, 366
os_setup_idle_function, 361
os_setup_tick_function, 361
os_thread_create, 357
os_thread_delete, 357
os_thread_relinquish, 372
os_thread_self_complete, 359
os_thread_sleep, 358
os_thread_stack_define, 372
os_thread_stack_t, 373
os_thread_wait_abort, 358
os_ticks_get, 372
os_ticks_to_msec, 358
os_ticks_to_unblock, 372
os_timer_activate, 369, 374
os_timer_activate_t, 374
os_timer_change, 369
os_timer_create, 368
os_timer_deactivate, 370
os_timer_delete, 371
os_timer_get_context, 370
os_timer_is_running, 369
os_timer_reload, 374
os_timer_reload_t, 374
os_timer_reset, 370
os_total_ticks_get, 372
wm_reboot_reason_t
wmstats.h, 387
wm_rfget_code
rfget.h, 302
wm_supplicant.h, 374
supp_event, 375
supplicant_finish, 375
supplicant_send_cmd, 375
supplicant_start, 374
wm_telnetd_errno
telnetd.h, 307
wm_utils.h, 375
bin2hex, 376
container_of, 379
get_random_sequence, 378
hex2bin, 376
random_hdlr_t, 379
random_initialize_seed, 377
random_register_handler, 376
random_register_seed_handler, 377
random_unregister_handler, 376
random_unregister_seed_handler, 377
sample_initialise_random_seed, 378
soft_crc32, 378
strdup, 378
wm_wlan.h, 379
wm_core_and_wlan_init, 380
wm_wlan_deinit, 380
wm_wlan_init, 379
wm_wlan_deinit
wm_wlan.h, 380
wm_wlan_errno
wlan.h, 341
wm_wlan_init
wm_wlan.h, 379
wmcrypto.h, 380
mrvl_aes_unwrap, 382
mrvl_aes_unwrap_ext, 383
mrvl_aes_wrap, 382
mrvl_aes_wrap_ext, 383
mrvl_crypto_hmac_md5, 383
mrvl_crypto_md5, 384
mrvl_crypto_pass_to_key, 384
mrvl_dh_compute_key, 380
mrvl_dh_free, 381
mrvl_dh_setup_key, 380
mrvl_hmac_sha256, 382
mrvl_kdf, 382
mrvl_sha1_vector, 381
mrvl_sha256, 381
mrvl_sha256_vector, 381
SHA256_BLOCK_SIZE, 384
SHA256_DIGEST_SIZE, 384
u32, 384
u8, 384
wmerrno.h, 385
MOD_AF, 385
MOD_CRIT_ERR, 387
MOD_CRYPTO, 386
MOD_DHCPD, 385
MOD_DIAG, 386
MOD_ERR_LAST, 387
MOD_EXML, 385
MOD_FTFS, 385
MOD_HTTPC, 386
MOD_HTTPD, 385

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

MOD_JSON, 385
MOD_MDNS, 385
MOD_PROV, 386
MOD_PSM, 386
MOD_RFGET, 385
MOD_SMDNS, 385
MOD_SPI, 386
MOD_SYSINFO, 386
MOD_TELNETD, 385
MOD_TTCP, 386
MOD_UNUSED_1, 386
MOD_UNUSED_2, 386
MOD_UNUSED_3, 385
MOD_USB, 387
MOD_WIFI, 387
MOD_WLAN, 386
MOD_WPS, 386
wmm
wlan_scan_result, 75
wmprintf
wmstdio.h, 387
wmstats.h, 387
wm_reboot_reason_t, 387
wmstdio.h, 387
ll_printf, 388
MAX_MSG_LEN, 389
wmprintf, 387
wmstdio_flush, 388
wmstdio_getchar, 388
wmstdio_getconsole_port, 388
wmstdio_init, 389
wmstdio_putchar, 387
wmstdio_flush
wmstdio.h, 388
wmstdio_getchar
wmstdio.h, 388
wmstdio_getconsole_port
wmstdio.h, 388
wmstdio_init
wmstdio.h, 389
wmstdio_putchar
wmstdio.h, 387
wmsysinfo.h, 389
sysinfo_init, 390
wmtime.h, 390
asctime, 391
gmtime_r, 391
http_date_to_time, 390
mktime, 391
wmtime_cli_init, 392
wmtime_init, 392
wmtime_time_get, 390
wmtime_time_get_posix, 391
wmtime_time_set, 390
wmtime_time_set_posix, 391
wmtime_cli_init
wmtime.h, 392
wmtime_init
wmtime.h, 392
wmtime_time_get
wmtime.h, 390
wmtime_time_get_posix
wmtime.h, 391
wmtime_time_set
wmtime.h, 390
wmtime_time_set_posix
wmtime.h, 391
wpa
wlan_scan_result, 76
wpa2
wlan_scan_result, 76
wps.h
CMD_WPS_PBC, 394
CMD_WPS_PIN, 394
WPS_FINISHED, 395
WPS_SESSION_ABORTED, 394
WPS_SESSION_FAILED, 395
WPS_SESSION_INACTIVE, 395
WPS_SESSION_PBC, 395
WPS_SESSION_PIN, 395
WPS_SESSION_PIN_CHKSUM_FAILED, 394
WPS_SESSION_STARTED, 394
WPS_SESSION_SUCCESSFUL, 395
WPS_SESSION_TIMEOUT, 395
WPS_STARTED, 394
wps.h, 392
WPS_LOG, 394
wps_connect, 393
wps_event, 394
wps_generate_pin, 393
wps_session_command, 394
wps_session_types, 395
wps_start, 393
wps_stop, 394
wps_validate_pin, 394
wps_callback
wps_config, 78
wps_config, 76
config_methods, 77
device_name, 77
manufacture, 77
model_name, 77
model_number, 77
os_version, 78
pin_generator, 77
pin_len, 78
primary_dev_category, 77
primary_dev_subcategory, 77
rf_bands, 77
role, 77
serial_number, 77
version, 77
version2, 77
wps_callback, 78
wps_msg_max_retry, 78
wps_msg_timeout, 78

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell

wps_connect
wps.h, 393
wps_event
wps.h, 394
wps_generate_pin
wps.h, 393
wps_msg_max_retry
wps_config, 78
wps_msg_timeout
wps_config, 78
wps_session_command
wps.h, 394
wps_session_types
wps.h, 395
wps_start
wps.h, 393
wps_stop
wps.h, 394
wps_validate_pin
wps.h, 394
ws_close_ctx
websockets.h, 308
ws_event_t
websockets.h, 308
ws_frame_recv
websockets.h, 308
ws_frame_send
websockets.h, 307
ws_frame_t
websockets.h, 308
ws_set_genmask_callback
websockets.h, 308
ws_upgrade_socket
websockets.h, 307
wsgi
httpd_request_t, 42
wslay_frame, 78
data, 79
data_len, 79
fin, 78
opcode, 78

Confidential

MARVELL
Proprietary Information.
August 21, 2015 17:48
Copyright © 2015 Marvell