Qualcomm MobileAP Software API for MDM9x15 Interface Specification

80-N5576-27 B

November 20, 2012

Submit technical questions at:
https://support.cdmatech.com/

Confidential and Proprietary – Qualcomm Technologies, Inc.

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Contents

Tables

Revision history

Revision	Date	Description
A	Apr 2012	Initial release
B	Nov 2012	Updated Sections 2.1, 2.2.8, 2.2.10, 3.1.1.3, 3.1.2.9.2 and 3.2; added Sections 2.2.26 to 2.2.27 and 3.1.2.9.12 to 3.1.2.9.16

1 Introduction

1.1 Purpose

This document is the reference specification for the Qualcomm Mobile Access Point Software (QCMobileAP) Application Programming Interface (API) on MDM9x15 platforms.

1.2 Scope

This document is intended for internal distribution to engineering and product marketing teams involved in the development, integration, and deployment of the arbitration manager feature, and also for external distribution to commercial test equipment vendors, infrastructure interoperability test partners, and handset software licensees and carriers.

1.3 Conventions

Function declarations, function names, type declarations, and code samples appear in a different font, e.g., `#include`.

Code variables appear in angle brackets, e.g., `<number>`.

Parameter types are indicated by arrows:

→	Designates an input parameter
←	Designates an output parameter
↔	Designates a parameter used for both input and output

Shading indicates content that has been added or changed in this revision of the document.

1.4 References

Reference documents are listed in Table 1-1. Reference documents that are no longer applicable are deleted from this table; therefore, reference numbers may not be sequential.

Table 1-1 Reference documents and standards

Ref.	Document	
Qualcomm Technologies		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*Data Services API Interface Specification and Operational Description*	80-V6415-1
Q3	*RM Network (RmNet) Feature Description Document*	80-VT270-1
Q4	*Qualcomm MSM™ Interface (QMI) Architecture*	80-VB816-1
Standards		
S1	*Dynamic Host Configuration Protocol*	RFC 2131
S2	*DHCP Options and BOOTP Vendor Extensions*	RFC 2132

1.5 Technical assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at https://support.cdmatech.com/.

If you do not have access to the CDMATech Support Service website, register for access or send email to support.cdmatech@qti.qualcomm.com.

1.6 Acronyms

For definitions of terms and abbreviations, see [Q1].

2 API Definition

2.1 Data definitions – QCMobileAP configuration data structure

The following data structures are used to configure the QCMobileAP using the Connection Manager library APIs that are listed in Section 2.2. Reference Connection Manager software provided by Qualcomm makes use of these data structures and APIs and as such can be used by customers as examples. The configuration can also be read and stored in .xml files described in Chapter 3.

```
/*---------------------------------------------------------------------------
--
           Port Forwarding Entry Request Type.
---------------------------------------------------------------------------
*/
typedef enum
{
  QCMAP_CM_ADD_SNAT                   = 0x01,
  QCMAP_CM_GET_SNAT                   = 0x02,
  QCMAP_CM_DELETE_SNAT                = 0x03
} qcmap_cm_snat_req_enum_type;

/*---------------------------------------------------------------------------
--
           Port Forwarding Entry Configuration.
---------------------------------------------------------------------------
*/
typedef struct
{
  uint32   port_fwding_private_ip;
  uint16   port_fwding_private_port;
  uint16   port_fwding_global_port;
  uint8    port_fwding_protocol;
} qcmap_cm_port_fwding_entry_conf_t;
```

```
/*-----------------------------------------------------------------------------
--
           FireWall Enable Request Type.
-----------------------------------------------------------------------------
*/
typedef enum
{
  QCMAP_CM_ENABLE_FIREWALL             = 0x01,
  QCMAP_CM_GET_FIREWALL                = 0x02,
  QCMAP_CM_DISABLE_FIREWALL            = 0x03
} qcmap_cm_firewall_req_enum_type;

/*-----------------------------------------------------------------------------
--
           FireWall Entry Request Type.
-----------------------------------------------------------------------------
*/
typedef enum
{
  QCMAP_CM_ADD_FIREWALL_ENTRY_RULE          = 0x01,
  QCMAP_CM_GET_FIREWALL_ENTRY_RULE          = 0x02,
  QCMAP_CM_DELETE_FIREWALL_ENTRY_RULE       = 0x03,
  QCMAP_CM_ADD_EXTD_FIREWALL_ENTRY_RULE     = 0x04,
  QCMAP_CM_GET_EXTD_FIREWALL_ENTRY_RULE     = 0x05,
  QCMAP_CM_GET_FIREWALL_HANDLE_LIST         = 0x06,
  QCMAP_CM_DELETE_EXTD_FIREWALL_ENTRY_RULE = 0x07
} qcmap_cm_firewall_entry_req_e;

/*-----------------------------------------------------------------------------
--
           FireWall Entry Configuration.
-----------------------------------------------------------------------------
*/
typedef struct
{
  uint16   firewall_start_dest_port;
  uint16   firewall_end_dest_port;
  uint8    firewall_protocol;
  uint32   firewall_handle;
} qcmap_cm_firewall_entry_conf_t;
```

```
/*-----------------------------------------------------------------------
--
           Extended FireWall Entry Configuration.
------------------------------------------------------------------------
*/
typedef struct
{
  ip_filter_type filter_spec;
  uint32         firewall_handle;
} qcmap_cm_extd_firewall_entry_conf_t;

/*-----------------------------------------------------------------------
--
           Extended FireWall handle list configuration.
------------------------------------------------------------------------
*/
typedef struct
{
  int handle_list[QCMAP_MAX_FIREWALL_ENTRIES_V01];
  ip_version_enum_type ip_family;
  int num_of_entries;
} qcmap_cm_get_extd_firewall_handle_list_conf_t;

/*-----------------------------------------------------------------------
--
           Extended FireWall handle configuration.
------------------------------------------------------------------------
*/
typedef struct
{
  int handle;
  ip_version_enum_type ip_family;
} qcmap_cm_extd_firewall_handle_conf_t;

/*-----------------------------------------------------------------------
--
           Extended FireWall configuration.
------------------------------------------------------------------------
*/
typedef union
{
  qcmap_cm_extd_firewall_entry_conf_t extd_firewall_entry;
  qcmap_cm_get_extd_firewall_handle_list_conf_t extd_firewall_handle_list;
  qcmap_cm_extd_firewall_handle_conf_t extd_firewall_handle;
} qcmap_cm_extd_firewall_conf_t;
```

```
/*---------------------------------------------------------------------------
--
           DMZ Request Type.
---------------------------------------------------------------------------
*/
typedef enum
{
  QCMAP_CM_ADD_DMZ                      = 0x01,
  QCMAP_CM_GET_DMZ                      = 0x02,
  QCMAP_CM_DELETE_DMZ                   = 0x03
} qcmap_cm_dmz_req_enum;

/*---------------------------------------------------------------------------
--
           VPN Request Type.
---------------------------------------------------------------------------
*/
typedef enum
{
  QCMAP_CM_SET_IPSEC                    = 0x01,
  QCMAP_CM_GET_IPSEC                    = 0x02,
  QCMAP_CM_SET_L2TP                     = 0x03,
  QCMAP_CM_GET_L2TP                     = 0x04,
  QCMAP_CM_SET_PPTP                     = 0x05,
  QCMAP_CM_GET_PPTP                     = 0x06
}qcmap_cm_vpn_req_enum;

/*---------------------------------------------------------------------------
--
           NAT Entry Timeout Request Type.
---------------------------------------------------------------------------
*/
typedef enum
{
  QCMAP_CM_SET_NAT_ENTRY_TIMEOUT        = 0x01,
  QCMAP_CM_GET_NAT_ENTRY_TIMEOUT        = 0x02
} qcmap_cm_timeout_req_enum;

/*---------------------------------------------------------------------------
--
           LAN Interface Access Profile Type.
---------------------------------------------------------------------------
*/
typedef enum
{
  QCMAP_CM_PROFILE_FULL_ACCESS          = 0x01,
  QCMAP_CM_PROFILE_INTERNET_ONLY        = 0x02
} qcmap_cm_access_profile_type;
```

```
/*---------------------------------------------------------------------------
--
           LAN Interface Device Mode Type.
---------------------------------------------------------------------------
*/
typedef enum
{
  QCMAP_CM_DEVMODE_AP                  = 0x01,
  QCMAP_CM_DEVMODE_STA                 = 0x02
} qcmap_cm_devmode_type;

/*---------------------------------------------------------------------------
--
           Network Config Mode Type.
---------------------------------------------------------------------------
*/
typedef enum
{
  QCMAP_CM_WIFI_MODE_NOT_SET,
  QCMAP_CM_WIFI_AP_MODE,
  QCMAP_CM_WIFI_AP_AP_MODE,
  QCMAP_CM_WIFI_AP_STA_MODE,
  QCMAP_CM_WIFI_AP_AP_STA_MODE
} qcmap_cm_wifi_mode_type;
```

```
/*-----------------------------------------------------------------------------
--
           NAT Configuration.
-----------------------------------------------------------------------------
*/
typedef struct
{
  uint16   nat_entry_timeout;
  uint32   dmz_ip; /* 0 means disable DMZ */
  uint8    enable_ipsec_vpn_pass_through;
  uint8    enable_pptp_vpn_pass_through;
  uint8    enable_l2tp_vpn_pass_through;

  uint8    num_port_fwding_entries;
  qcmap_cm_port_fwding_entry_conf_t
           port_fwding_entries[QCMAP_MAX_SNAT_ENTRIES_V01];

  char     firewall_config_file[QCMAP_CM_MAX_FILE_LEN];

  uint8    num_extd_firewall_entries;
  qcmap_cm_extd_firewall_conf_t
           extd_firewall_entries[QCMAP_MAX_FIREWALL_ENTRIES_V01];



  uint8    firewall_enabled;
  uint8    firewall_pkts_allowed;
} qcmap_cm_nat_conf_t;

/*-----------------------------------------------------------------------------
--
           LAN Configuration.
-----------------------------------------------------------------------------
*/
#define QCMAP_LAN_INVALID_QCMAP_HANDLE (-1)
#define QCMAP_LAN_INVALID_IFACE_INDEX  (-1)
#define QCMAP_LAN_MAX_IPV4_ADDR_SIZE   16    /* 3 dots + 4 * 3 #s + 1 null
*/
#define QCMAP_IP_V4V6_V01              10
#define QCMAP_CM_MAX_FILE_LEN          64
/* 3 Interfaces - Possible Modes: AP, AP+AP, STA+AP, STA+AP+AP */
#define QCMAP_MAX_NUM_INTF             3


typedef struct
{
  /* Enable and configure main interface. */
  boolean  enable;
  qcmap_cm_devmode_type devmode;

```

```
    /* Path to WLAN AP config which contain SSID/Mode/Encryption info */
    char     path_to_hostapd_conf[QCMAP_CM_MAX_FILE_LEN];

    /* Main interface configuration. All Addresses are in host order */
    uint32   a5_ip_addr;
    uint32   sub_net_mask;

    /* Type of access main interface has to networks. */
    qcmap_cm_access_profile_type access_profile;

    /* DHCP server config */
    boolean  enable_dhcpd;
    uint32   dhcp_start_address;
    uint32   dhcp_end_address;
    uint32   dhcp_lease_time;

     /* Interface parameters specific to STA Mode. */
    boolean  enable_supplicant;
    char     path_to_supplicant_conf[QCMAP_CM_MAX_FILE_LEN];
    char     external_ap_ssid[QCMAP_CM_MAX_FILE_LEN];

  } qcmap_cm_intf_conf_t;

  typedef struct
  {
    /* Interface information. */
    qcmap_cm_intf_conf_t interface[QCMAP_MAX_NUM_INTF];

    char     module[QCMAP_CM_MAX_FILE_LEN];

    uint32   a5_rmnet_ip_addr;
    uint32   q6_ip_addr_facing_a5;
    uint32   usb_rmnet_ip_addr;
    uint32   q6_ip_addr_facing_usb_rmnet;
    uint32   nat_ip_addr;

  } qcmap_cm_lan_conf_t;
```

```
/*------------------------------------------------------------------------------
--
           WAN Configuration.
------------------------------------------------------------------------------
*/
#define QCMAP_WAN_INVALID_QCMAP_HANDLE 0xFFFFFFFF
#define QCMAP_WAN_MAX_ERI_DATA_SIZE    256
#define QCMAP_WAN_TECH_ANY             0
#define QCMAP_WAN_TECH_3GPP            1
#define QCMAP_WAN_TECH_3GPP2           2

typedef struct
{
  boolean   auto_connect;
  boolean   roaming;
  char      eri_config_file[QCMAP_CM_MAX_FILE_LEN];

  int       tech;
  int       umts_profile_index;
  int       cdma_profile_index;
  int       ip_family;

} qcmap_cm_wan_conf_t;

/*------------------------------------------------------------------------------
--
           Master Mobile AP Config.
------------------------------------------------------------------------------
*/
typedef struct
{
  qcmap_cm_nat_conf_t nat_config;
  qcmap_cm_wan_conf_t wan_config;
  qcmap_cm_lan_conf_t lan_config;

} qcmap_cm_conf_t;
```

2.2 API for QCMobileAP Connection Manager

The reference QCMobileAP Connection Manager is provided as an example of how to call an API exposed by the QCMobileAP Connection Manager library to enable/bring up LAN/bring up WAN/tear down WAN/tear down LAN. The reference QCMobileAP Connection Manager is a C++ class with the following functions exported.

2.2.1 QCMAP_ConnectionManager:: QCMAP_ConnectionManager

This function initializes the QCMobileAP Connection Manager. This function uses the QCMobileAP Connection Manager library to read the passed QCMobileAP .xml config file, and store the WAN, LAN, and NAT configuration locally.

Parameters

```
QCMAP_ConnectionManager    (
                       char      *xml_path
                       )
```

→	QCMAP_ConnectionManager:: QCMAP_ConnectionManager		
	→	xml_path	Path and filename for .xml config file. A NULL value for this parameter will cause the class to use default configuration values.

Return value

None

2.2.2 QCMAP_ConnectionManager::Enable()

This function calls the QCMobileAP Connection Manager library to enable the NAT and LAN on Hexagon. When successfully completed, the QCMobileAP Connection Manager library calls the QCMAP_ConnectionManager_callback callback to indicate completion. The reference QCMobileAP Connection Manager calls EnableWLAN to bring up the Linux WLAN interfaces.

Parameters

None, the configuration parameters are read from the QCMobileAP .xml configuration file.

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.3 QCMAP_ConnectionManager::Disable()

This function calls the QCMobileAP Connection Manager library to disable the NAT and LAN on Hexagon. When successfully completed, the QCMobileAP Connection Manager library calls the QCMAP_ConnectionManager_callback callback to indicate completion. The reference QCMobileAP Connection Manager calls DisableWLAN to tear down the Linux WLAN interfaces.

Parameters

None, the configuration parameters are read from the QCMobileAP .xml configuration file.

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.4 QCMAP_ConnectionManager:: ConnectBackHaul ()

Purpose

This function calls the QCMobileAP Connection Manager library to place a data call on the Hexagon to obtain an Internet address. When successfully completed, the QCMobileAP Connection Manager library calls the QCMAP_ConnectionManager_callback callback to indicate completion.

Parameters

None – Configuration parameters are read from the QCMobileAP XML configuration file.

Return Value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.5 QCMAP_ConnectionManager:: DisconnectBackHaul ()

This function calls the QCMobileAP Connection Manager library to disconnect a data call on Hexagon. When successfully completed, the QCMobileAP Connection Manager library calls the QCMAP_ConnectionManager_callback callback to indicate completion.

Parameters

None, the configuration parameters are read from the QCMobileAP .xml configuration file.

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.6 QCMAP_ConnectionManager:: AddStaticNatEntry () QCMAP_ConnectionManager:: DeleteStaticNatEntry()

This function calls the QCMobileAP Connection Manager library to add/delete static nat configuration entries on Hexagon. If successful, the QCMobileAP Connection Manager .xml configuration file is updated.

Parameters

```
qcmap_cm_port_fwding_entry_conf_t    *nat_entry
```

→	QCMAP_ConnectionManager:: AddStaticNatEntry QCMAP_ConnectionManager:: DeleteStaticNatEntry		
	→	nat_entry	Static NAT configuration to add or delete

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.7 QCMAP_ConnectionManager:: GetStaticNatEntry ()

This function calls the QCMobileAP Connection Manager library to retrieve the static nat configuration entries on Hexagon.

Parameters

```
qcmap_cm_port_fwding_entry_conf_t         *nat_entry
int                                      max_entries
```

→	QCMAP_ConnectionManager::GetStaticNatEntry		
	←	nat_entry	Pointer to an array of Static nat entries to store retrieved configuration
	→	max_entries	The total number of entries in the passed array

Return value

`(int)number of records`

- ≥ 0 if successful
- -1 otherwise

2.2.8 QCMAP_ConnectionManager:: AddFireWallEntry () QCMAP_ConnectionManager:: DeleteFireWallEntry()

These functions call the QCMobileAP Connection Manager library to add/delete firewall configuration entries on Hexagon. If successful, the mobileap_firewall.xml is updated.

Parameters

`qcmap_cm_firewall_entry_conf_t* firewall_entry`

→	`QCMAP_ConnectionManager::AddFireWallEntry` `QCMAP_ConnectionManager::DeleteFireWallEntry`		
	→	`firewall_entry`	Firwall configuration to add or delete

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.9 QCMAP_ConnectionManager:: GetFireWallEntry ()

This function calls the QCMobileAP Connection Manager library to retrieve the firewall configuration entries on Hexagon.

Parameters

```
qcmap_cm_firewall_entry_conf_t      *firewall_entry
int                                 max_entries
```

→	`QCMAP_ConnectionManager::GetFireWallEntry`		
	←	`firewall_entry`	Pointer to an array of firewall entries to store retrieved configuration
	→	`max_entries`	The total number of entries in the passed array

Return value

`(int)number of records`

- ≥ 0 if successful
- -1 otherwise

2.2.10 QCMAP_ConnectionManager:: AddExtdFireWallEntry () QCMAP_ConnectionManager:: DeleteExtdFireWallEntry()

This function calls the QCMobileAP Connection Manager library to add/delete extended firewall configuration entries on Hexagon. If successful, the mobileap_firewall.xml configuration file is updated.

Parameters

`qcmap_cm_extd_firewall_conf_t* extd_firewall_conf`

→	`QCMAP_ConnectionManager::AddExtdFireWallEntry` `QCMAP_ConnectionManager::DeleteExtdFireWallEntry`		
	→	`extd_firewall_conf`	Firewall configuration to add or delete

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.11 QCMAP_ConnectionManager::GetExtdFireWallEntry ()

This function calls the QCMobileAP Connection Manager library to retrieve the extended firewall configuration entries on Hexagon.

Parameters

`qcmap_cm_extd_firewall_conf_t* extd_firewall_conf`

→	`QCMAP_ConnectionManager::GetFireWallEntry`		
	←	`extd_firewall_conf`	Pointer to an array of Static NAT entries to store retrieved configuration

Return value

`(int)number of records`

- ≥ 0 if successful
- -1 otherwise

2.2.12 QCMAP_ConnectionManager:: GetFireWallHandleList ()

This function calls the QCMobileAP Connection Manager library to retrieve all firewall handle entries on Hexagon.

Parameters

`qcmap_cm_extd_firewall_conf_t* extd_firewall_conf`

→	QCMAP_ConnectionManager::GetFireWallHandleList		
	←	extd_firewall_conf	Pointer to firewall handle list

Return value

`(int)number of records`

- ≥ 0 if successful
- -1 otherwise

2.2.13 QCMAP_ConnectionManager:: SetFirewall ()

This function calls the QCMobileAP Connection Manager library to set the firewall status on Hexagon.

Parameters

```
boolean          enable
boolean          pkts_allowed
```

→	QCMAP_ConnectionManager::SetFirewall		
	→	Enable	Enable or disable the firewall setting on the Hexagon
	→	pkts_allowed	Allow or disallow packets setting on the Hexagon

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.14 QCMAP_ConnectionManager:: AddDMZ () QCMAP_ConnectionManager:: DeleteDMZ()

These functions call the QCMobileAP Connection Manager library to add/delete the DMZ IP address configuration entry on Hexagon. If successful, the QCMobileAP Connection Manager .xml configuration file is updated.

Parameters

`uint32    dmz_ip`

→	`QCMAP_ConnectionManager::AddDMZ` `QCMAP_ConnectionManager::DeleteDMZ`		
	→	`dmz_ip`	IP address of the internal address to forward all incoming network packets

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.15 QCMAP_ConnectionManager:: GetDMZ ()

This function calls the QCMobileAP Connection Manager library to retrieve the DMZ IP adresss configured on Hexagon.

Parameters

None

Return value

`(int)IP address of computer registered for DMZ`

2.2.16 QCMAP_ConnectionManager:: SetNATEntryTimeout ()

This function calls the QCMobileAP Connection Manager library to set the dynamic NAT entry timeout value on Hexagon.

Parameters

`uint16                timeout`

→	`QCMAP_ConnectionManager::SetNATEntryTimeout`		
	→	`Timeout`	The timeout in seconds for dynamic NAT entry timeout

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.17 QCMAP_ConnectionManager:: GetNATEntryTimeout()

This function calls the QCMobileAP Connection Manager library to get the dynamic NAT entry timeout value configured on Hexagon.

Parameters

None

Return value

`(uint16)Dynamic NET entry timeout`

2.2.18 QCMAP_ConnectionManager:: SetIPSECVpnPassThrough () QCMAP_ConnectionManager:: SetPPTPVpnPassThrough () QCMAP_ConnectionManager:: SetL2TPVpnPassThrough ()

These functions call the QCMobileAP Connection Manager library to enable or disable the IPSEC, PPTP, L2TP configuration entry on Hexagon. If successful, the QCMobileAP Connection Manager .xml configuration file is updated.

Parameters

`boolean    enable`

→	QCMAP_ConnectionManager::SetIPSECVpnPassThrough QCMAP_ConnectionManager::SetPPTPVpnPassThrough QCMAP_ConnectionManager::SetL2TPVpnPassThrough		
	→	enable	Enables or disables the IPSEC, PPTP, L2TP VPN passthrough setting

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.19 QCMAP_ConnectionManager:: SetAutoconnect ()

This function calls the QCMobileAP Connection Manager library to enable or disable the autoconnect setting. If TRUE and no WWAN data call is established, the QCMobileAP Connection Manager library will attempt a data call when a network is detected. If successful, the QCMobileAP Connection Manager .xml configuration file is updated.

Parameters

```
boolean    enable
```

→	`QCMAP_ConnectionManager::SetAutoconnect`		
	→	`enable`	Enables or disables the autoconnect setting

Return value

```
(boolean)status
```

- TRUE if successful
- FALSE otherwise

2.2.20 QCMAP_ConnectionManager:: SetRoaming ()

This function calls the QCMobileAP Connection Manager library to enable or disable the roaming setting. If TRUE, the QCMobileAP Connection Manager library will attempt a data call when a roaming network is detected. If successful, the QCMobileAP Connection Manager .xml configuration file is updated.

Parameters

```
boolean    enable
```

→	`QCMAP_ConnectionManager::Set Roaming`		
	→	`enable`	Enables or disables the roaming setting

Return value

```
(boolean)status
```

- TRUE if successful
- FALSE otherwise

2.2.21 QCMAP_ConnectionManager:: SetHostAPDConfig ()

This function verifies that the passed file exists. If the file does, then the local configuration information for the passed interface is updated. If successful, the QCMobileAP Connection Manager .xml configuration file is updated.

Parameters

```
uint         intf
char * cfg_pathname
```

→	QCMAP_ConnectionManager::SetHostAPDConfig		
	→	intf	The interface whose HostAPD configuration file should be updated
	→	cfg_pathname	The full file and path name to the HostAPD configuration

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.22 QCMAP_ConnectionManager:: SetDHCPDConfig ()

This function is used to se DHCP configuration parameters on A5.

Parameters

```
uint         intf
uint         start
uint         end
char *  leasetime
```

→	QCMAP_ConnectionManager::SetDHCPDConfig		
	→	intf	Interface whose DHCPD configuration should be updated
	→	start	Starting IP address to provide DHCP clients
	→	end	Ending IP address to provide DHCP clients
	→	leasetime	Time in hours for which the DHCP lease for clients is valid

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.23 QCMAP_ConnectionManager::GetIPv4WWANNetwork Configuration()

This function returns the IP and DNS addresses acquired by the WWAN network interface.

Parameters

```
uint         *public_ip
uint         *primary_dns
uint         *secondary_dns
```

→	QCMAP_ConnectionManager::GetIPv4WWANNetworkConfiguration		
	←	public_ip	IP address acquired by the WWAN network
	←	primary_dns	Primary DNS address acquired by the WWAN network
	←	secondary_dns	Secondary DNS address acquired by the WWAN network

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.24 QCMAP_ConnectionManager::GetWWANStatistics()

This function returns the IP statistics for the passed IP version type.

Parameters

```
ip_version_enum_type ip_family
qcmap_cm_statistics_t *wwan_stats
int *p_error
```

→	QCMAP_ConnectionManager::GetWWANStatistics		
	→	ip_version_enum_type	▪ IP_V4 – Get IPv4 WWAN statistics ▪ IP_V6 – Get IPv6 WWAN statistics
	←	wwan_stats	Statistics for the passed IP version
	←	p_error	Extended error provided if the function fails

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.25 QCMAP_ConnectionManager::ResetWWANStatistics()

This function resets the IP statistics for the passed IP version type.

Parameters

```
ip_version_enum_type ip_family
int *p_error
```

→	QCMAP_ConnectionManager::ResetWWANStatistics		
	→	ip_version_enum_type	▪ IP_V4 – Get IPv4 WWAN statistics ▪ IP_V6 – Get IPv6 WWAN statistics
	←	p_error	Extended error provided if the function fails

Return value

```
(boolean)status
```

- TRUE if successful
- FALSE otherwise

2.2.26 QCMAP_ConnectionManager::StartEmbeddedCall()

NOTE: This section was added to this document revision.

This function calls the QCMobileAP Connection Manager library to place a data call on Hexagon to obtain an Internet address on the requested profile. If the profile is the same as the MobileAP call, the call will not be brought up.

Parameters

```
int IP_Family,
int UMTS_PROFILE_INDX,
int CDMA_PROFILE_INDX,
int Tech
```

→	QCMAP_ConnectionManager:: StartEmbeddedCall		
	→	IP_Family	▪ 4 - IP_V4 family ▪ 6 - IP_V6 family ▪ 10 – IP_V4V6 family
	→	UMTS_PROFILE_INDX	A valid UMTS profile index already set on Hexagon
	→	CDMA_PROFILE_INDX	A valid CMDA profile index already set on Hexagon
	→	Tech	▪ 0 – Connect the WAN on any available network ▪ 1 – Connect the WAN on a UMTS network ▪ 2 – Connect the WAN on a CDMA network

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

2.2.27 QCMAP_ConnectionManager:: EndEmbeddedCall()

NOTE: This section was added to this document revision.

This function calls the QCMobileAP Connection Manager library to disconnect the embedded data call on Hexagon.

Parameters

None

Return value

`(boolean)status`

- TRUE if successful
- FALSE otherwise

3 QCMobileAP Configuration Database Schema

The QCMobileAP configuration .xml will map to the QCMobileAP config data structure shown in this chapter.

3.1 XML sections

The tag for the QCMobileAP configuration database schema is <MobileAPCfg>. This tag contains subsections with configurations for QCMobileAP:

- NAT/Firewall
- LAN
- WWAN

3.1.1 QCMobileAP NAT/Firewall configuration

The tag for the QCMobileAP NAT/Firewall configuration database schema is <MobileAPNatCfg>. This tag contains subsections with configurations for port forwarding, simple and extended firewall, DMZ IP address, NAT timeout, and VPN passthrough.

3.1.1.1 FirewallEnabled

The FirewallEnabled tag instructs the NAT engine to filter packets based upon additional firewall configuration entries that follow. This tag should be set to:

- 1 – Enable the firewall
- 0 – Disable the firewall

Tag format	<FirewallEnabled>[boolean]</FirewallEnabled>
Valid values	▪ 1 – Enable Firewall ▪ 0 – Disable Firewall
Default value	0

3.1.1.2 FirewallPktsAllowed

The FirewallPktsAllowed tag instructs the NAT engine to drop or forward packets based upon additional firewall configuration entries that follow. This tag should be set to:

- 1 – Forward packets that match the firewall configuration
- 0 – Drop packets that match the firewall configuration

Tag format	<FirewallPktsAllowed>[boolean]</FirewallPktsAllowed>
Valid values	▪ 1 – Do not drop packets matching firewall configuration ▪ 0 – Drop packets matching firewall configuration
Default value	0

3.1.1.3 Firewall

NOTE: The following section has been updated.

The tag for the QCMobileAP Firewall XML file is <Firewall>. This will contain the full path and name of the firewall XML file and will be created once firewall entries are added. The firewall XML will contain individual firewall rules.

Tag format	<Firewall>[path and name for firewall xml]</Firewall>
Valid values	A valid path and filename for firewall xml
Default value	/etc/mobileap_firewall.xml

3.1.1.4 PortFwding

The tag for the QCMobileAP Port Forwarding Entry configuration database schema is <PortFwding>. This tag contains subsections to specify the private IP address, the private and global ports, and the protocol to forward.

3.1.1.4.1 PortFwdingPrivateIP

The PortFwdingPrivateIP tag is used to specify the destination IP address of incoming packets to forward.

Tag format	<PortFwdingPrivateIP>[IP Address]</PortFwdingPrivateIP>
Valid values	An IP address entered in dotted decimal format
Default value	NA – Must be set to be a valid entry

3.1.1.4.2 PortFwdingPrivatePort

The PortFwdingPrivatePort tag is used to specify the port to which to forward incoming packets.

Tag format	<PortFwdingPrivatePort>[unsigned int]</PortFwdingPrivatePort>
Valid values	An integer from 0 to 65535
Default value	NA – Must be set to be a valid entry

3.1.1.4.3 PortFwdingGlobalPort

The PortFwdingGlobalPort tag is used to specify the destination global port of incoming packets to forward.

Tag format	<PortFwdingGlobalPort>[unsigned int]</PortFwdingGlobalPort>
Valid values	An integer from 0 to 65535
Default value	NA – Must be set to be a valid entry

3.1.1.4.4 PortFwdingProtocol

The PortFwdingProtocol tag is used to specify the protocol of packets to forward.

Tag format	<PortFwdingProtocol>[unsigned int]</PortFwdingProtocol>
Valid values	▪ 1 – ICMP ▪ 6 – TCP ▪ 17 – UDP ▪ 58 – ICMPv6 ▪ 253 – TCP_UDP
Default value	NA – Must be set to be a valid entry

3.1.1.5 NatEntryTimeout

The NatEntryTimeout tag specifies the dynamic NAT entry timeout in seconds.

Tag format	<NatEntryTimeout>[unsigned int]</NatEntryTimeout>
Valid values	An integer from 0 to 65535
Default value	120

3.1.1.6 DmzIP

The DmzIP tag specifies an internal IP address to forward all incoming packets that do not match the existing firewall and port forwarding entries.

Tag format	<DmzIP>[IP address]</DmzIP>
Valid values	An IP address entered in dotted decimal format
Default value	0.0.0.0 – DMZ is disabled

3.1.1.7 EnableIPSECVpnPassthrough

The EnableIPSECVpnPassthrough tag contains a boolean value used to set the state of the IPSEC VPN passthrough.

Tag format	<EnableIPSECVpnPassthrough>[boolean]</EnableIPSECVpnPassthrough>
Valid values	▪ 1 – Enable IPSEC VPN passthrough functionality ▪ 0 – Disable IPSEC VPN passthrough functionality
Default value	0

3.1.1.8 EnablePPTPVpnPassthrough

The EnablePPTPVpnPassthrough tag contains a boolean value used to set the state of the PPTP VPN passthrough.

Tag format	<EnablePPTPVpnPassthrough>[boolean]</ EnablePPTPVpnPassthrough>
Valid values	▪ 1 – Enable PPTP VPN passthrough functionality ▪ 0 – Disable PPTP VPN passthrough functionality
Default value	0

3.1.1.9 EnableL2TPVpnPassthrough

The EnableL2TPVpnPassthrough tag contains a boolean value used to set the state of the L2TP VPN passthrough.

Tag format	<EnableL2TPVpnPassthrough>[boolean]</ EnableL2TPVpnPassthrough>
Valid values	▪ 1 – Enable L2TP VPN passthrough functionality ▪ 0 – Disable L2TP VPN passthrough functionality
Default value	0

3.1.2 QCMobileAP LAN configuration

The tag for the QCMobileAP LAN configuration database schema is <MobileAPLanCfg>. This tag contains subsections with configurations for A5 side.

3.1.2.1 Module

The Module tag specifies the name of the Linux kernel module to load to enable the WLAN adapter. This is specified as the name of the module, without the .ko extension or the path to the module.

Tag format	<Module>[string]</Module>
Valid values	The filename without path or extension of the Linux kernel module to load
Default value	ar6000

3.1.2.2 DevMode

The DevMode tag specifies the type of WLAN to configure, Access Point, or Station. In Access Point mode, the WLAN adapter acts as an AP and allows clients to connect to its network. In Station mode, the WLAN adapter searches for and joins the network of an external Access Point. The primary WLAN interface must be configured as an AP. Setting this field to STA will cause the software to fail WLAN bringup.

Tag format	<DevMode>[string]</DevMode>
Valid values	▪ AP – Use Access Point mode ▪ STA – Use Station mode
Default value	AP

3.1.2.3 HostAPDCfg

The HostAPDCfg tag specifies the full path and filename of the HostAPD configuration file to use to configure an AP. This file contains configuration information such as SSID and encryption modes.

Tag format	<HostAPDCfg>[string]</HostADPCfg>
Valid values	Full path and filename of a valid HostAPD config file
Default value	/etc/hostapd.conf

3.1.2.4 APIPAddr

The APIPAddr tag specifies the IP address to assign to the WLAN interface. This address is typically on a nonroutable subnet such as 192.168.1.0 or 10.0.0.0.

Tag format	<APIPAddr>[IP address]</APIPAddt>
Valid values	An IP address entered in dotted decimal format
Default value	192.168.1.1

3.1.2.5 SubNetMask

The SubNetMask tag specifies the subnet mask for the APIPAddr network.

Tag format	<SubNetMask>[IP address]</SubNetMask>
Valid values	An IP address entered in dotted decimal format
Default value	255.255.255.0

3.1.2.6 AccessProfile

The AccessProfile tag specifies the access to various service clients of a WLAN. Presently, FULL access allows access to services running on the QCMobileAP, Internet access, and access to clients on other WLAN interfaces. INTERNETONLY access allows Internet access. Access to other WLAN interfaces and services running on the QCMobileAP are denied.

Tag format	<AccessProfile>[string]</AccessProfile>
Valid values	▪ FULL – Access to local services, other WLANs and Internet ▪ INTERNETONLY – Internet access only
Default value	FULL

3.1.2.7 EnableDHCPServer

The EnableDHCPServer tag specifies that a DCHPD server should be started to provide WLAN clients IP addresses.

Tag format	<EnableDHCPServer>[string]</EnableDHCPServer>
Valid values	▪ 1 – Enable DHCP server on this WLAN ▪ 0 – Do not enable DHCP server on this WLAN
Default value	1

3.1.2.8 DHCPCfg

The tag for the QCMobileAP DHCPD Configuration Database schema is <DHCPCfg>. The DHCPCfg tag specifies configuration information needed to configure the DHCPD server.

3.1.2.8.1 StartIP

The StartIP tag specifies the starting IP address that may be assigned to clients of this WLAN.

Tag format	<StartIP>[IP address]</StartIP>
Valid values	An IP address entered in dotted decimal format
Default value	192.168.1.21

3.1.2.8.2 EndIP

The EndIP tag specifies the ending IP address that may be assigned to clients of this WLAN.

Tag format	<EndIP>[IP address]</EndIP>
Valid values	An IP address entered in dotted decimal format
Default value	192.168.1.41

3.1.2.8.3 LeaseTime

The LeaseTime tag specifies the amount of time DHCP client leases are valid before they must renew.

Tag format	<LeaseTime>[string]</LeaseTime>
Valid values	▪ An integer value in sec for the lease time, e.g., 43200 ▪ An integer followed by 'H' for hours and 'M' for minutes, e.g., 12 H or 720 M ▪ The word 'infinite' to specify an infinite DHCP client lease time
Default value	12 H

3.1.2.9 SSID2Cfg

The tag for the QCMobileAP Second SSID Database schema is <SSID2Cfg>. The SSID2Cfg tag specifies configuration for the second WLAN interface.

3.1.2.9.1 Enable

The Enable tag specifies whether the second WLAN interface should be enabled and configured. When this tag indicates that the second interface should not be brought up, all other tags in this section are not needed.

Tag format	<Enable>[boolean]</Enable>
Valid values	▪ 1 – Enable the second WLAN interface ▪ 0 – Do not enable the second WLAN interface
Default value	0

3.1.2.9.2 DevMode

The DevMode tag specifies the type of WLAN to configure, Access Point or Station.

- Access Point mode – WLAN adapter acts as an AP and allows clients to connect to its network
- Station mode – WLAN adapter searches for and joins the network of an external AP

When set to AP mode, the following tags are required:

- HostAPDCfg
- APIPAddr
- SubNetMask
- Access Profile
- EnableDHCPServer

When set to STA mode, the following tags are required:

- EnableSupplicant
- SupplicantCfg
- ExternalAPSSID
- STAModeConnType
- StaticIPAddr
- StaticConfigDNSAddr
- StaticConfigGWAddr
- StaticConfigNetMask

Tag format	<DevMode>[string]</DevMode>
Valid values	▪ AP – Use Access Point mode ▪ STA – Use Station mode
Default value	AP

3.1.2.9.3 HostAPDCfg

The HostAPDCfg tag specifies the full path and filename of the HostAPD configuration file to use to configure an AP. This file contains configuration information such as SSID and encryption modes.

Tag format	<HostAPDCfg>[string]</HostADPCfg>
Valid values	Full path and filename of a valid HostAPD config file
Default value	/etc/hostapd.conf

3.1.2.9.4 APIPAddr

The APIPAddr tag specifies the IP address to assign to the WLAN interface. This address is typically on a nonrouteable subnet such as 192.168.1.0 or 10.0.0.0.

Tag format	<APIPAddr>[IP address]</APIPAddt>
Valid values	An IP address entered in dotted decimal format
Default value	192.168.2.1

3.1.2.9.5 SubNetMask

The SubNetMask tag specifies the subnet mask for the APIPAddr network.

Tag format	<SubNetMask>[IP address]</SubNetMask>
Valid values	An IP address entered in dotted decimal format
Default value	255.255.255.0

3.1.2.9.6 AcessProfile

The AccessProfile tag specifies the access to various services clients of this WLAN. Presently, FULL access allows access to services running on the QCMobileAP, Internet access, and access to clients on other WLAN interfaces. INTERNETONLY access allows Internet access. Access to other WLAN interfaces and services running on the QCMobileAP are denied.

Tag format	<AccessProfile>[string]</AccessProfile>
Valid values	▪ FULL – Access to local services, other WLANs and Internet ▪ INTERNETONLY – Internet access only
Default value	FULL

3.1.2.9.7 EnableDHCPServer

The EnableDHCPServer tag specifies that a DCHPD server should be started to provide WLAN clients IP address.

Tag format	<EnableDHCPServer>[boolean]</EnableDHCPServer>
Valid values	▪ 1 – Enable DHCP server on this WLAN ▪ 0 – Do not enable DHCP server on this WLAN
Default value	1

3.1.2.9.8 DHCPCfg

See Section 3.1.2.8 for information on completing the DHCPCfg.

3.1.2.9.9 EnableSupplicant

The EnableSupplicant tag specifies the use of the Linux utility, wpa_supplicant, to bring up and configure the WLAN interface configured for Station mode.

Tag format	<EnableSupplicant>[boolean]</EnableSupplicant>
Valid values	▪ 1 – Use Linux utility, wpa_supplicant, to configure this WLAN interface ▪ 0 – Do not use Linux utility, wpa_supplicant
Default value	0

3.1.2.9.10 SupplicantCfg

The SupplicantCfg tag specifies the full path and filename of the configuration file to use with the Linux utility, wpa_supplicant. This tag is only used if the EnableSupplicant tag is set to TRUE. This configuration file contains information about external APs, such as their SSIDs, encryption modes, access keys, and passwords.

Tag format	<SupplicantCfg>[string]</SupplicantCfg>
Valid values	Full path and filename of a valid wpa_supplicant config file
Default value	NA

3.1.2.9.11 ExternalAPSSID

The ExternalAPSSID tag specifies the SSID of an open network to which to connect the QCMobileAP for Internet access. This tag is only used if EnableSupplicant is set to FALSE.

Tag format	<ExternalAPSSID>[string]</ExternalAPSSID>
Valid values	SSID to connect
Default value	NA

3.1.2.9.12 STAModeConnType

NOTE: This section was added to this document revision.

The STAModeConnType tag specifies the type of connection in Station mode. The type of connection can be Dynamic, i.e., the IP address configuration will be through DHCP or it can be Static, i.e., the IP address will be configured manually. This tag is only used for Station mode.

Tag format	< STAModeConnType>[boolean]</ STAModeConnType>
Valid values	▪ 1 – Static mode of IP configuration ▪ 0 – Dynamic mode of IP configuration
Default value	0

3.1.2.9.13 StaticIPAddr

NOTE: This section was added to this document revision.

The StaticIPAddr tag specifies the IP address to assign to the Station mode WLAN interface. This address is typically on a nonrouteable subnet such as 192.168.1.0 or 10.0.0.0. This tag is only used if STAModeConnType is 1.

Tag format	< StaticIPAddr>[IP address]</StaticIPAddr>
Valid values	An IP address entered in dotted decimal format
Default value	NA

3.1.2.9.14 StaticConfigDNSAddr

NOTE: This section was added to this document revision.

The StaticConfigDNSAddr tag specifies the DNS address of the hotspot network in Station mode.

Tag format	< StaticConfigDNSAddr>[IP address]</ StaticConfigDNSAddr>
Valid values	An IP address entered in dotted decimal format
Default value	NA

3.1.2.9.15 StaticConfigGWAddr

NOTE: This section was added to this document revision.

The StaticConfigGWAddr tag specifies the Gateway address of the hotspot network in Station mode.

Tag format	< StaticConfigGWAddr>[IP address]</ StaticConfigGWAddr>
Valid values	An IP address entered in dotted decimal format
Default value	NA

3.1.2.9.16 StaticConfigNetMask

NOTE: This section was added to this document revision.

The StaticConfigNetMask tag specifies the subnet mask of the hotspot network in Station mode.

Tag format	< StaticConfigNetMask>[IP address]</ StaticConfigNetMask>
Valid values	An IP address entered in dotted decimal format
Default value	NA

3.1.2.10 SSID3Cfg

The tag for the QCMobileAP Second SSID Database schema for concurrent AP+STA mode is <SSID3Cfg>. The SSID3Cfg tag specifies configuration for the second WLAN interface. This interface will not be available in the first release of the QCMobileAP hardware and software. The sub-elements of this tag are configured the same as Section 3.1.2.9 QCMobileAP Second SSID Database schema.

3.1.3 QCMobileAP WAN Configuration

The tag for the QCMobileAP WAN Configuration Database schema is <MobileAPWanCfg>. This tag contains subsections with configurations for network interfaces and Hexagon IP addresses.

3.1.3.1 AutoConnect

The AutoConnect tag specifies if the software should make attempts to reconnect a dropped WAN connection.

Tag format	<AutoConnect>[boolean]</AutoConnect>
Valid values	▪ 1 – Attempt to re-connect a dropped WAN connection ▪ 0 – Do not attempt
Default value	0

3.1.3.2 Roaming

The Roaming tag specifies if the WAN connection should be made on a roaming network. This tag is only used when the AutoConnect tag is set to TRUE.

Tag format	<Roaming>[boolean]</Roaming>
Valid values	▪ 1 – Connect the WAN on a roaming network ▪ 0 – Do not Connect
Default value	0

3.1.3.3 EriConfig

The EriConfig tag specifies a filename to read to provide Extended Roaming Indicators to use when determining if a call should be made on a roaming network. This tag is only used when the Roaming tag is set to FALSE.

Tag format	<EriConfig>[string]</EriConfig>
Valid values	Full path and filename of a binary file containing Extended Roaming Indicators
Default value	/etc/mobileap_eri_config.bin

3.1.3.4 TECH

The TECH tag specifies a technology type to use when opening a WAN connection over the mobile network. TECH, UMTS_PROFILE_INDEX, CDMA_PROFILE_INDEX, and IPFamily specify the mobile nework policy to use to connect the WAN.

Tag format	<TECH>[string]</TECH>
Valid values	▪ 3GPP – Connect the WAN on a UMTS network ▪ 3GPP2 – Connect the WAN on a CDMA network ▪ ANY – Connect the WAN on any available network
Default value	ANY

3.1.3.5 UMTS_PROFILE_INDEX

The UMTS_PROFILE_INDEX tag specifies the profile index to use when opening a WAN connection over a mobile UMTS network. TECH, UMTS_PROFILE_INDEX, CDMA_PROFILE_INDEX, and IPFamily specify the mobile nework policy to use to connect the WAN.

Tag format	<UMTS_PROFILE_INDEX>[value]</UMTS_PROFILE_INDEX>
Valid values	A valid UMTS profile index already set on the Hexagon
Default value	0

3.1.3.6 CDMA_PROFILE_INDEX

The CDMA_PROFILE_INDEX tag specifies the profile index to use when opening a WAN connection over a mobile CDMA network. TECH, UMTS_PROFILE_INDEX, CDMA_PROFILE_INDEX, and IPFamily specify the mobile nework policy to use to connect the WAN.

Tag format	<CDMA_PROFILE_INDEX>[value]</CDMA_PROFILE_INDEX>
Valid values	A valid CDMA profile index already set on the Hexagon
Default value	0

3.1.3.7 IPFamily

The IPFamily tag specifies an IP family to set up when opening a WAN connection over the mobile network. TECH, UMTS_PROFILE_INDEX, CDMA_PROFILE_INDEX, and IPFamily specify the mobile nework policy to use to connect the WAN.

Tag format	<IPFamily>[string]</IPFamily>
Valid values	▪ IPv4 – Connect to an IPv4 network ▪ IPv6 – Connect to an IPv6 network ▪ IPv4v6 – Connect to an IPv4 and IPv6 network
Default value	IPv4

3.2 XML example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<system xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mobileap_cfg.xsd">
  <MobileAPCfg>
    <MobileAPNatCfg>
      <FirewallEnabled>1</FirewallEnabled>
      <FirewallPktsAllowed>0</FirewallPktsAllowed>
             <Firewall>/etc/mobileap_firewall.xml</Firewall>
      <PortFwding>
        <PortFwdingPrivateIP>192.168.1.8</PortFwdingPrivateIP>
        <PortFwdingPrivatePort>5500</PortFwdingPrivatePort>
        <PortFwdingGlobalPort>5500</PortFwdingGlobalPort>
        <PortFwdingProtocol>17</PortFwdingProtocol>
      </PortFwding>
      <NatEntryTimeout>200</NatEntryTimeout>
      <DmzIP>0.0.0.0</DmzIP>
      <EnableIPSECVpnPassthrough>1</EnableIPSECVpnPassthrough>
      <EnablePPTPVpnPassthrough>1</EnablePPTPVpnPassthrough>
      <EnableL2TPVpnPassthrough>1</EnableL2TPVpnPassthrough>
    </MobileAPNatCfg>
    <MobileAPLanCfg>
      <Module>ar6000</Module>
      <DevMode>AP</DevMode>
      <HostAPDCfg>/etc/hostapd.conf</HostAPDCfg>
      <APIPAddr>192.168.1.1</APIPAddr>
      <SubNetMask>255.255.255.0</SubNetMask>
      <AccessProfile>FULL</AccessProfile>
      <EnableDHCPServer>1</EnableDHCPServer>
      <DHCPCfg>
        <StartIP>192.168.1.20</StartIP>
        <EndIP>192.168.1.40</EndIP>
        <LeaseTime>12h</LeaseTime>
      </DHCPCfg>
      <SSID2Cfg>
        <Enable>0</Enable>
        <DevMode>AP</DevMode>
        <HostAPDCfg>/etc/hostapd-eth1.conf</HostAPDCfg>
        <APIPAddr>192.168.2.1</APIPAddr>
        <SubNetMask>255.255.255.0</SubNetMask>
        <AccessProfile>INTERNETONLY</AccessProfile>
        <EnableSupplicant>0</EnableSupplicant>
```

```
          <SupplicantCfg>NA</SupplicantCfg>
          <ExternalAPSSID>NA</ExternalAPSSID>
          <STAModeConnType>NA</STAModeConnType>
          <StaticIPAddr>NA</StaticIPAddr>
          <StaticConfigDNSAddr>NA</StaticConfigDNSAddr>
          <StaticConfigGWAddr>NA</StaticConfigGWAddr>
          <StaticConfigNetMask>NA</StaticConfigNetMask>
          <EnableDHCPServer>1</EnableDHCPServer>
          <DHCPCfg>
            <StartIP>192.168.2.20</StartIP>
            <EndIP>192.168.2.40</EndIP>
            <LeaseTime>12h</LeaseTime>
          </DHCPCfg>
        </SSID2Cfg>
        <SSID3Cfg>
          <Enable>0</Enable>
          <DevMode>STA</DevMode>
          <HostAPDCfg>NA</HostAPDCfg>
          <APIPAddr>NA</APIPAddr>
          <SubNetMask>NA</SubNetMask>
          <AccessProfile>NA</AccessProfile>
          <EnableSupplicant>1</EnableSupplicant>
          <SupplicantCfg>/etc/wpa_supplicant.conf</SupplicantCfg>
          <ExternalAPSSID>dlink-ap</ExternalAPSSID>
          <STAModeConnType>0</STAModeConnType>
          <StaticIPAddr>0</StaticIPAddr>
          <StaticConfigDNSAddr>0</StaticConfigDNSAddr>
          <StaticConfigGWAddr>0</StaticConfigGWAddr>
          <StaticConfigNetMask>0</StaticConfigNetMask>
          <EnableDHCPServer>0</EnableDHCPServer>
          <DHCPCfg>
            <StartIP>NA</StartIP>
            <EndIP>NA</EndIP>
            <LeaseTime>0</LeaseTime>
          </DHCPCfg>
        </SSID3Cfg>
      </MobileAPLanCfg>
```

```
          <MobileAPWanCfg>
            <AutoConnect>0</AutoConnect>
            <Roaming>0</Roaming>
            <EriConfig>/etc/mobileap_eri_config.bin</EriConfig>
            <TECH>ANY</TECH>
            <UMTS_PROFILE_INDEX>0</UMTS_PROFILE_INDEX>
            <CDMA_PROFILE_INDEX>0</CDMA_PROFILE_INDEX>
            <IPFamily>IPv4</IPFamily>
          </MobileAPWanCfg>
        </MobileAPCfg>
      </system>
```