AR6000 Chipset Family Wireless Module Programmer's Guide

Revision 3.0

January 2010

ATHEROS®

Notice

Document Number: MKG-1299 Rev. 1

Document Conventions

Text Conventions

bold	Bold type within paragraph text indicates file names, directory names, paths, output, or returned values. Example: The DK_Client package will not function without the **wdreg_install** batchfile.
italic	Within commands, italics indicate a variable that the user must specify. Example: **MEM_ALLOC** *size_in_bytes* Titles of manuals or other published documents are also set in italics.
`Courier`	The Courier font indicates output or display. Example: `Error:Unable to allocate memory for transfer!`
[ ]	Within commands, items enclosed in square brackets are optional parameters or values that the user can choose to specify or omit.
{ }	Within commands, items enclosed in braces are options from which the user must choose.
|	Within commands, the vertical bar separates options.
...	An ellipsis indicates a repetition of the preceding parameter.
>	The right angle bracket separates successive menu selections. Example: Start > Programs > DK > wdreg_install.

Notices

NOTE: This message denotes neutral or positive information that calls out important points to the main text. A note provides information that may apply only in special cases.

Revision History

Revision	Description of Changes
January 2010	Release 3.0; supports AR6003
July 2009	Release 2.2
January 2009	Release 2.1.2
February 2008	Release 2.1; supports only AR6002
July 2007	New release for code release 2.0; supports AR6001 and AR6002 chipsets

Contents

Preface xiii

1 Overview 1-1

Hardware 1-2

Software (Host and Target) 1-3

Distribution 1-3

Host Software 1-4

Target Support for Host Bringup 1-4

Reference Host Software Stack 1-5

Software Messaging 1-6

Startup Sequence 1-7

Applications 1-8

Clocks and Crystals 1-9

Configuration/Customization Methods 1-10

2 Wireless Operation 2-1

Wireless Control Commands 2-1

Mechanics of Wireless Messaging 2-2

Power Management 2-2

Wireless Statistics 2-2

Heartbeat 2-3

Security 2-3

Wi-Fi Protected Access 2-3

Performance 2-4

3 Scan and Roam 3-1

Connection Services 3-2

Scanning 3-3

Active Scanning 3-4

World Mode Support and Passive Scanning 3-4

Foreground Scan 3-5
Background Scan 3-5
Roaming 3-6
Trigger 3-6
Roam Scan 3-6
BSS Selection 3-7
BSS Roaming Candidate List 3-8
Disconnect Event to Host 3-8
Channel Management 3-8

4 **Wake-on-Wireless** 4-1

Host Wakeup Overview 4-1
Host-Target Interface 4-2
Support Requirements 4-2
wmiconfig Commands 4-3
Sample addwowpattern Commands 4-5
Target-Side Statistics 4-5

A **Wireless Module Interface (WMI)** A-1

Data Frames A-1
WMI Message Structure A-2
Data Endpoints A-2
Connection States A-3
Message Types A-3
WMI Message Format A-4
Command Restrictions A-4
Command and Data Synchronization A-4
WMI Commands A-5
WMI Events A-53
WMI Extension Commands A-66
WMI Extension Events A-69
SoftAP WMI Commands A-71

B **Messaging and Communications** B-1

Bootloader Message Interface (BMI) B-1

BMI Requests B-1
BMI_DONE B-2
Host Interest Area B-2

Host/Target Message Interface B-2

Host Target Communications B-3
HTC 2.1 Protocol B-3
HTC 2.1 Overview B-4
HTC Single Mailbox Design B-4
HTC Endpoints and Services B-4
HTC Flow Control and Credit Rebalancing B-5
HTC Packet Bundling B-5
Bundling in the Receive Path B-6
Bundling in the Transmit Path B-6
HTC Protocol B-7
HTC Frame Header Format B-7
HTC Control Messages B-8
HTC Ready (Message ID = 1) B-8
Connect Service (Message ID = 2) B-9
Service Connect Response (Message ID = 3) B-10
HTC Setup Complete (Message ID = 4) B-10
HTC Ready (Message ID = 5) B-10
HTC Trailer Data B-11
HTC Service Connection Flow B-13
HTC Endpoint Data Flow B-13
HTC Credits and Buffers B-13
HTC QoS B-14
HTC Credit Synchronization B-14
Demand-Based Endpoint Credit Updates B-15
Using HTC with AR6000 Mailbox Interface B-16

C DataSets C-1

Board Data C-1

DataSets C-1

Regulatory Database C-2

Analog Configuration C-2

Startup Patch DataSet C-3

GPIO Configuration Patch DataSet C-3

D Host Interface Support D-1

Interface Protocols D-1

Software Interface Components D-1

Common Information Area D-2

Mailboxes. D-2

Messages D-2

Flow Control D-3

E SPI Host Interface E-1

SPI Bus Interface E-1

SPI Bus Protocol E-3

Power State Synchronization. E-3

SPI Bus Access Types E-4

SPI Internal Register Read/Write E-4

SPI External Register Read/Write. E-4

SPI External Read/Write to Mailbox E-4

SPI Command E-5

Command Types. E-5

Transfer Format E-6

SPI Internal Register Access E-6

SPI External Mailbox Access E-7

Transfer Mode E-7

SPI External Mailbox Write. E-8

SPI External Mailbox Read E-9

SPI External Core Logic Register Access E-10

Host Control Register Write E-10

Host Control Register Read E-10

Byte Swapping E-11

F Registers F-1

Host Interface. F-1

Pending Interrupt Status (HOST_INT_STATUS). F-2

CPU-Sourced Interrupt Status (CPU_INT_STATUS) F-2

Error or Wakeup Interrupt Status (ERROR_INT_STATUS). F-3

Host IF Credit Counter Interrupt (COUNTER_INT_STATUS) F-3

Mailbox FIFO Status (MBOX_FRAME) F-4

Valid Bits for Lookahead (RX_LOOKAHEAD_VALID). F-4

Lookahead to Next 4 MBOX Rx0 FIFO Bytes (RX_LOOKAHEAD0) . . F-4
Lookahead to Next 4 MBOX Rx1 FIFO Bytes (RX_LOOKAHEAD1) . . F-5
Lookahead to Next 4 MBOX Rx2 FIFO Bytes (RX_LOOKAHEAD2) . . F-5
Lookahead to Next 4 MBOX Rx3 FIFO Bytes (RX_LOOKAHEAD3) . . F-5
HOST_INT_STATUS Enable Bits (INT_STATUS_ENABLE) F-6
CPU Sourced Interrupt Status (CPU_INT_STATUS_ENABLE) F-6
Error Interrupt Status (ERROR_STATUS_ENABLE) F-7
Credit Counter Interrupt Status (COUNTER_INT_STATUS_ENABLE) F-7
Credit Counters Direct Access (COUNT) F-8
Credit Counter Atomic Decrement (COUNT_DEC) F-8
Interface Scratch (SCRATCH) F-9
FIFO Timeout Period (FIFO_TIMEOUT) F-9
FIFO Timeout Enable (FIFO_TIMEOUT_ENABLE). F-9
Disable Sleep Mode (DISABLE_SLEEP) F-10
Local Bust Host Interface (LOCAL_BUS) F-10
AR6002 CPU Interrupt (INT_WLAN) F-10
SPI Slave Interface Configuration (SPI_CONFIG). F-11
SPI Status (SPI_STATUS). F-12
SDIO CIS Tuples Copy (CIS_WINDOW). F-12

SPI Internal F-13
DMA Size (DMA_SIZE) F-13
Write Buffer Space Available (WRBUF_SPC_AVA) F-13
Read Buffer Byte Available (RDBUF_BYTE_AVA) F-14
SPI Configuration (SPI_CONFIG). F-14
SPI Status (SPI_STATUS). F-15
Host Control Register Access Byte Size (HOST_CTRL_BYTE_SIZE) . F-15
Host Control Register Configure (HOST_CTRL_CONFIG) F-15
Host Control Register Read Port (HOST_CTRL_RD_PORT). F-15
Host Control Register Write Port (HOST_CTRL_WR_PORT) F-16
Interrupt Cause (INTR_CAUSE) F-16
Interrupt Enable (INTR_ENABLE) F-17
Read Buffer Water Mark (RDBUF_WATERMARK) F-17
Write Buffer Water Mark (WRBUF_WATERMARK) F-17
Read Buffer Lookahead 1 (RDBUF_LOOKAHEAD1) F-18
Read Buffer Lookahead 2 (RDBUF_LOOKAHEAD2) F-18

G AP Bringup and Configuration in Linux G-1

Bring Up the Chipset in AP Mode With MAC Bridging G-1

Configure the AR6000 Chipset AP in Open Mode. G-2

Configure Security on the AR6000 Chipset AP. G-2

WEP Security. G-2
Shared Key Authentication. G-2
WPA (TKIP) Security G-2
WPA2 (AES) Security G-2
wmiconfig Application Commands for AP mode G-3
Switch Modes Between AP and STA on the AR6000 Chipset G-3
Switch from STA to AP mode G-3
Switch from AP to STA Mode G-3

H **Linux APIs** H-1

IOCTLs Support and Organization H-2
Private IOCTLs (SIOCWFIRSTPRIV) H-9
AR6000_IOCTL_EXTENDED H-33
Event Notifications. H-56

I **WinMobile** I-1

Architecture. I-1
DEVICE IOCTLs I-2
Generic IOCTLs I-2
BMI-Specific IOCTLs I-2
HTC-Specific IOCTLs I-2
OIDs I-4
NDIS OIDs I-4
Query OIDs I-4
Set OIDs I-5
Atheros OIDs. I-6
Query OIDs I-6
Set OIDs I-6
SoftAP OIDs I-7
Query OIDs I-7
Set OIDs I-7
WinMobile Registry Entries. I-8
AR6000 Registry Entries I-8
Modifying Registry Entries. I-10
AR6000 Host Driver Sources I-10
Private OID/IOCTL Structures I-11
AR6000 Generic IOCTLs I-11
NDIS Query OIDs. I-18

NDIS Set OIDs I-19
WMI/TCMD OID Structures I-22
WinCE NDIS Startup Architecture I-22

J Bluetooth Coexistence J-1
Overview J-1
Coexistence Protocol J-1
RF Front End J-2
Single Antenna with RF Switch J-2
Software Configuration J-3
SCO Coexistence J-3
SCO Coexistence Power-Save Mode J-3
SCO Coexistence Active Mode J-3
A2DP Coexistence J-3
A2DP Coexistence Power-Save J-3
A2DP Coexistence Active Mode J-3
ACL Coexistence J-4
WMI Commands J-4

K WLAN Power Save Options K-1
Overview K-1
Power Consumption of Atheros AR6000 . . . . K-2
Power Saving Techniques K-2
Legacy Power Save (PS)-Poll Scheme K-2
Legacy Power Save with Power Management (PM) Bit K-3
Unscheduled Automatic Power Save Delivery (U-APSD) K-4
Power Saving Techniques and VoIP Traffic K-4
VoIP Using U-APSD K-4
VoIP Using PS-Polling K-5
VoIP Using Legacy Power Save (PS) with PM Bit K-6
Performance Comparison of PS Schemes for VoIP K-6
Atheros Simulated U-APSD K-6

L Windows Mobile SoftAP L-1
Architecture L-1
Features L-2

M Debug Logging M-1

Introduction M-1

Theory of Operation M-1

Debug Primitives. M-2

N Address Resolution Protocol Offloading N-1

O Licensing Issues. O-1

Preface

This document provides an introduction to the AR6000 Wireless Module Software Development Kit (SDK). This document provides detailed information intended for a software development audience, and its information can be used to enhance, extend, or adapt the reference source code to meet customer requirements. This level of detail can also aid a user or developer to understand advanced functionality and internal operation beyond what is provided in the user's guide. This document is intended for software release 3.0.

This document does not attempt to cover every subject in exhaustive detail; rather, it allows the reader an opportunity to generally understand what the various components are and how they interact. Perusal of the code may provide more depth of understanding. Header files, especially ones that describe formal APIs, include valuable comments that exhaustively describe each interface and parameter. Therefore this document covers some topics briefly and refers the reader to the appropriate header files and documentation.

This document is organized into chapters intended to be read in order. Collectively, they provide a brief overview of all components followed by Appendixes that each focus on a topic likely to be of interest to a developer with a specific project in mind. These Appendixes can be read out of order or skipped entirely when they are not relevant to a particular project.

About this Document

The document consists of these chapters and appendixes:

Chapter 1	**Overview** — Overview of the AR6000 wireless modules. Also provides an overview of the major applications used in conjunction with the AR6000. Clocks and crystals are also discussed briefly.
Chapter 2	**Wireless Operation** — Describes the device CPU and memory. Also describes the WiFi protected access.
Chapter 3	**Scan and Roam** — Provides an overview of the AR6000 scanning and roaming procedures.

Chapter 4	**Wake-on-Wireless** — Describes the host wakeup feature.
Appendix A	**Wireless Module Interface (WMI)** — Describes the format and the usage model for WMI control and data messages between the host and the AR6000-based targets.
Appendix B	**Messaging and Communications** — Begins with an overview of the BMI services that allow the host to get target revision information, write code and data into specific locations in target memory, read code and data from target memory, read/write target registers, and execute at a specific target address. Also discusses HTC, the message transport used by WMI to send and receive control and data messages across the interconnect that connects the AR6000 targets and the host systems.
Appendix C	**DataSets** — Describes board data and DataSets.
Appendix D	**Host Interface Support**— Describes the interface host controller.
Appendix F	**Registers**— Describes the registers.
Appendix G	**AP Bringup and Configuration in Linux** — Describes bringup and configuration of the access point (AP).
Appendix H	**Linux APIs**— Describes Linux APIs.
Appendix I	**WinMobile**— Describes the WinMobile IOCTLs, OIDs, and registry entries.
Appendix J	**Bluetooth Coexistence** — Describes AR6000 coexistence with a Bluetooth implementation, with which it shares radio frequencies.
Appendix K	**WLAN Power Save Options** — Describes the WLAN power save options.
Appendix L	**Windows Mobile SoftAP** — Describes the SoftAP feature.
Appendix M	**Debug Logging** — Describes the debug logging feature.
Appendix N	**Address Resolution Protocol Offloading** — Describes the address resolution protocol offloading feature.
Appendix O	**Licensing Issues** — An overview of licensing agreements for software distributed with the AR6000 SDKs.

Audience

This guide is intended for engineers evaluating the Atheros AR6000 chipset family of WLAN chips.

Additional Resources

These resources should be referenced regarding topics that are not addressed in this document:

- *AR6003 ROCm*^{TM} *Single-Chip MAC/BB/Radio for 2.4/5 GHz Embedded WLAN Applications data sheet*
- *AR6003 ROCm*^{TM} *STA Reference Guide*
- *SDIO Setup and Configuration Guide*

1 Overview

Each AR6000 chipset contains a wireless LAN (WLAN) module that provides IEEE 802.11 WLAN network capability including:

- Wireless media access control (MAC)
- Radio
- Baseband
- IEEE 802.11 protocol processing handled by an on-chip network processor

Each AR6000 device provides wireless network capability to a host system (e.g., a hand-held device) with CPU, memory, and a standard interconnect (e.g., SDIO, or SPI). These devices allow a host system to connect to an IEEE 802.11 wireless network and achieve excellent performance with low power without requiring the host system to understand wireless protocols or radio hardware. Therefore the chipset appears like an ordinary NIC card (e.g., Ethernet card) but provides high-level wireless management features for a host that needs to control some aspects of wireless operation. The chipset network processor executes software (firmware) from ROM and RAM.

Host software is distributed for Fedora Core 9 Linux, WinMobile, and WinCE 6.0 OSs, and host software may be compiled within a Windows Mobile build environment. For a generic Windows CE 5/6 project, host software can be compiled either on the command line or as a project in Platform Builder. Windows CE/Mobile development requires Microsoft's Windows Mobile or Windows CE Platform Builder development toolkits hosted on a Windows PC. For linux, a Fedora Core 9 system with kernel source, GNU compiler and tool chains is required. Additional toolchains and kernel sources may be required to cross compile to other linux distributions or non-PC platforms. The reference target platform is an Atheros board with chipset, JTAG connector, and serial port connector. The reference host and target platforms illustrate the chipset wireless module capabilities and how a host system can use them. See Figure 1-1.

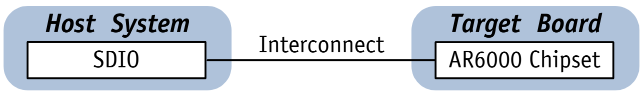

Figure 1-1. **Host System and Target Board**

Hardware

See the *AR6002 ROCm*^{TM} *Single-Chip MAC/BB/Radio for 2.4/5 GHz Embedded WLAN Applications data sheet* complete details of AR6002 hardware. See the *AR6003 ROCm*^{TM} *Single-Chip MAC/BB/Radio for 2.4/5 GHz Embedded WLAN Applications data sheet* complete details of AR6003 hardware. These data sheets describe the host view of the chipset, including its address space layout and bit-level details about each register.

AR6000 systems on a chip (SoC) include a CPU, DMA engines, external memory controller, control logic, a wireless MAC and baseband, a radio, general purpose I/O (GPIO) support, serial port support, and JTAG support.

The host system can be based on any processor (e.g., ARM, MIPS, x86, etc.) running any OS. As long as the host can communicate with the AR6000 chipset target using mechanisms and protocols expected by hardware and firmware, the chipset can provide wireless networking to the host. This communication between the host and the AR6000 chipset target occurs primarily through messages sent through a mailbox (bidirectional FIFO queues) and special-purpose AR6000 registers. The mailboxes and AR6000 registers are accessed through special addresses mapped into the device address space and accessed through services provided by a standard interconnect.

An interconnect is anything that allows the host and target to communicate. To support an interconnect, the AR6000 chipset must be embedded into a device (e.g. a board) compatible in terms of form factor, electrical characteristics, pinout, signaling, commands and functions, addressing, protocols, etc. The host software must also use an appropriate host controller to manage the attached chipset.

AR6000 chipsets support a variety of interconnects:

- SDIO (recommended)
  Version 2.0 in 1-bit SD and 4-bit SD modes (including SDIO HS mode); AR6002 also supports version 1.0
- Generic SPI interconnect

The host generally manages the AR6000 chipset in the same manner regardless of the type of interconnect. However, the mechanism used to manage this, may vary across interconnect types.

Software (Host and Target)

AR6000 software is partitioned into host-side and target-side software. Firmware runs on the chipset network processor and is stored in target memory, but host software runs on the hosts's CPU and uses host memory.

Firmware is written, owned, controlled, and maintained by Atheros. Atheros also provides reference host software for selected platforms. Customers may heavily leverage some host software components or even use some of them "as is"; other components are provided mostly for reference. While customers may leverage Atheros' efforts on host software, ultimately customers are responsible for the host software on their platforms, OSs, and interconnects. For Windows CE/Mobile, AR6000 host software conforms to the standard Windows CE SDIO stack driver model and can be used on any Windows CE platform that supports SDIO.

To facilitate development of host software, Atheros provides its customers source code for many host software reference implementations. See "Host Software" on page 1-4.

The AR6000 chipset executes from ROM and RAM embedded within the chipset, executes the downloaded application, and may execute code loaded into on-chip RAM. In all cases Atheros provides target-side firmware in binary form.

Distribution

ELF images contain symbol table information and facilitate debug and patches. However, they are not needed for normal operation. Some binary images (generated from the ELF images) are needed for normal operation. These binaries are pre-loaded into ROM.

Some AR6000 chipsets also contain on-chip ROM which is preloaded with an OS and WLAN firmware. For host software, reference host source code is included in the AR6000 SDK distributions.

The AR6000 wireless modules rely on several collections of data called DataSets. DataSets are stored in ROM or RAM and may be customized for a particular application. For example, the regulatory database containing regulatory information for domains around the world is stored in a DataSet. The analog configuration data that influences wireless radio characteristics is also in a DataSet. Patches may be required for correct operation. These patches are distributed along with instructions to use them.

Host Software

To use AR6000 modules with a host system, host software must be developed or ported from the Atheros reference host source. The end product does not need to look like the reference source. The API presented to the host OS may be completely different. The API presented to users may be different. Even internal APIs and the architecture and design within the host wireless support may be completely different as long as the host can communicate using the defined message interfaces; AR6000 provides wireless network capability.

The extent to which an implementation leverages the reference code determines how easy it is to integrate future changes (enhancements and fixes) into that implementation. To make it possible to leverage more code, OS-specific code is separated from generic code in the reference host source code. The header file **host/include/osapi.h** defines generic wrapper APIs to a set of OS-dependent services used by the OS-independent code. Most OS-dependent code is found in the directories named for the OS (e.g., the **linux** directory). Other directories are generic, both in terms of header files included and in terms of APIs used. The basic design, which separates host software into well-defined, layered modules with APIs (see Figure 1-2), also facilitates code maintenance as it allows distinct code components to be replaced without a rippling effect.

Developers should also note some code is flagged as optional, e.g., if no GPIO support is needed in an implementation, CONFIG_HOST_GPIO_SUPPORT can be undefined in **host/include/a_config.h**, and code under this configuration switch need not be ported.

Some aspects of the host platform beyond the choice of OS may complicate the implementation. Specifically, all data is passed across the interconnect to and from the AR6000 in 32-bit Little Endian format.

Target Support for Host Bringup

The AR6000 target provides some support for the host bringup effort. If possible, the reference target board from Atheros should be used during initial host driver development. This board is a known-working board, so it reduces the possible failure scenarios. Furthermore, it includes JTAG and serial port connectors, which can be used to aid in debugging. The AR6000 serial port is used for minimal debug output. Some firmware modules that are distributed with the SDK include explicit debug flags that enable verbose debug output. Other target modules with more debugging output are available from Atheros.

Reference Host Software Stack

Host software is organized into layers (see Figure 1-2), which collectively define the host software stack. In general, functions in the highest layer may call other functions at the same layer or one layer down. Functions do not make direct calls to higher layers, though upper layers may register callbacks with lower layers. Table 1-1 describes these host software layers. While some of the names are similar to names used in the ISO/OSI network model, they have no relationship. All layers described here are part of the design and implementation of the host wireless stack, part of the ISO/OSI network model's data link layer, or part of the TCP/IP network model's link layer. These descriptions applies to the Atheros reference host stack.

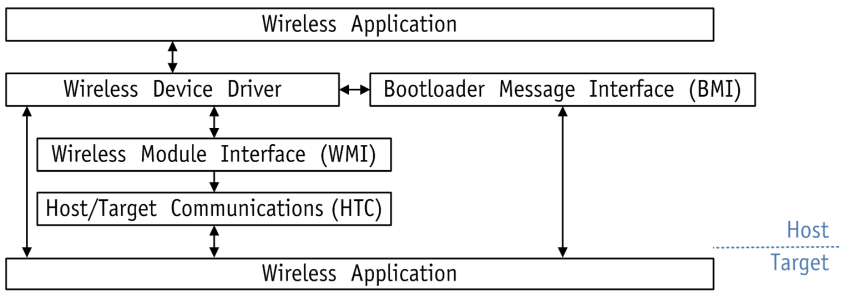
Wireless Application
Wireless Device Driver
Bootloader Message Interface (BMI)
Wireless Module Interface (WMI)
Host/Target Communications (HTC)
Host
Target
Wireless Application

Figure 1-2. **Host Software Layers**

Table 1-1. **Host Software Layers**

Layer	Description
Wireless Application	Produces data to send over wireless and consumes data coming in from wireless. This layer consists of the top of the host system's traditional network stack. It may also produce wireless control requests (e.g. OIDs). The wireless application calls to the wireless device driver for service. This component is platform-specific and provides the glue between platform-independent software and a particular platform. It handles application requests so data is sent and received and translates OS-specific control commands to platform- and interconnect-independent requests the target understands.
Wireless Device Driver/ Wireless Module Interface (WMI)	If the wireless application must send control messages to the chipset, it calls into the WMI layer to create the messages. This layer understands the host/target messaging protocol (WMI Protocol). Its source is at **host/wmi/**. The header files **include/wmi.h** and **include/wmix.h** list all messages from host to target (commands) and from target to host (requests and events).
Host/Target Communications (HTC)	The wireless device driver calls into HTC to handle message transport. HTC does not understand the contents of messages it transports (only WMI understands the contents of control messages), but it does understand the mechanics of messaging with the AR6000 chipset. It handles flow control and knows which chipset addresses must be read and written to relay messages. This layer source is at **host/htc2/**.
Hardware Interface (HIF)	HTC calls into the HIF layer when it needs to access the chipset address space. An HIF implementation exists for each combination of platform and interconnect API. This layer abstracts register and memory access details and provides an interconnect-independent and platform-independent API for use (mainly) by HTC. This layer source is at **host/hif/**.
Interconnect	The HIF layer relies on underlying interconnect-specific and platform-specific software to drive a hardware controller of some sort. The interconnect layer plays a role in device discovery, setting up an appropriate address space mapping and performing reads/writes through that address space, and deals with error management over the physical connection. For most serial buses, the HIF layers interfaces with a bus driver that provides an abstracted view of the underlying host bus adapter. These bus drivers may be provided by partners, OS vendors, or Atheros.

Data transmitted from the host traverses these software layers and transfers to the physical interconnect/bus. The AR6000 wireless module receives the data from the interconnect and the target firmware performs additional processing and delivers the data to the air. Control messages also make their way down this stack and over the interconnect, where the firmware interprets them and takes appropriate control actions.

When incoming data arrives over the air (or when the chipset wishes to report an event of interest), the flow is reversed: a message is created by the AR6000 chipset target and sent over the physical interconnect to the host. The host detects a pending message (via an interrupt) and retrieves the message through the interconnect. Received messages are passed up the stack for processing.

Software Messaging

All communication between host and target is comprised of messages that contain a message header, followed by payload data specific to the command. The target firmware and host software share several messaging interface specifications defined by a set of subcomponents (i.e., WMI, HTC, and BMI). The message interface details are defined in a header file for that interface. Messages may be transmitted between the host and target through a message transport (e.g. HTC). The message transport handles flow control, mechanisms needed to ensure reliable delivery, and additional required framing/padding. Table 1-2 describes the main message interfaces. All of these message interfaces use 32-bit little-Endian byte ordering.

Table 1-2. **Message Interface**

Interface	Description
Wireless Module Interface (WMI)	The wireless module interface (WMI) is used during normal operation to: ■ Tx data over wireless ■ Rx data from wireless ■ Control wireless operation. To control wireless operation, the host sends high-level WMI commands to the target. The target replies to some commands and it asynchronously notifies the host of WMI events. Flow control and framing for WMI is handled by the HTC message transport using a credit system, as described in "Messaging and Communications" on page B-1, which also details the HTC header prepended to each message and interconnect-dependent padding. Further WMI details (e.g., a complete list of commands, replies, and events) are included in the header files **include/wmi.h** (for wireless commands and events) and **include/wmix.h** (for non-wireless extensions).
	Example WMI command messages (host-to-target) ■ **"CONNECT"** (e.g. to an AP) ■ **START_SCAN** (e.g. for suitable APs) ■ **SET_RADIO_TRANSMIT_POWER** ■ **GET_STATISTICS**
	Example WMI event messages (target-to-host) ■ **WIRELESS_MODULE_READY** ■ **"DISCONNECT"** (e.g. from an AP) ■ **REPORT_STATISTICS** (e.g., for an earlier request from the host)

Table 1-2. **Message Interface**

Interface	Description
Bootloader Message Interface (BMI)	The BMI is used only during startup, before wireless activities have started. It affords the host some control over the boot and initialization procedure. Notably, BMI is used as a mechanism to install patches and platform-specific target customizations. A BMI command may request data from the target, and therefore may receive a BMI response. E.g., BMI responses include: ■ **TARGET ID** (in response to a **GET_TARGET_ID** command) ■ **REGISTER VALUE** (in response to a **READ_REGISTER** command) For message transport, the BMI does not rely on HTC. It uses its own simple message transport that allows only a single BMI command at a time and involves no further encapsulation. See also "Bootloader Message Interface" on page G-1.
	Example BMI commands ■ **GET_TARGET_ID** (e.g. ID and revision info) ■ **READ_REGISTER** ■ **WRITE_MEMORY**

Startup Sequence

This section describes the steps that occur from power up until wireless networking is enabled. Startup proceeds through a few phases on both host and target. Table 1-3 presents these phases in chronological order with descriptions of host and target activities.

Table 1-3. **Startup Sequence Phases**

Phase	Description	
Initialization	IO Enable	Once the host driver is loaded, it enables the AR6000 wireless modules in an interconnect-dependent way. For instance, on SDIO, the host uses a CIS/CCCR (common) register to power up and enable the device. Once the target is enabled, it executes firmware that initializes SoC and software states. AR6000 wireless components remain disabled and irrelevant during this period. Once initializations have completed, the target indicates to the host in an interconnect-dependent way that it is ready. For instance, SDIO contains a standard READY signal.
	CIS Tuples	Depending on the interconnect, the host driver may (e.g., in the case of SDIO) choose at this point to read CIS tuples that provide more information about the AR6000 device. CIS tuples are defined by PC card/PCMCIA standards and are used by SDIO and other interconnects. The host then enters the BMI phase.
BMI	The target issues a BMI command credit to the host and waits for communications from the host. While waiting for the host, the target places itself in a power-saving sleep state. Once the host receives the BMI command, it sends BMI messages to initialize and patch target state. Once the host has finished sending BMI commands, it sends a **BMI_DONE** command, causing the target to leave BMI mode. Once the target has left BMI mode, no further BMI messages are allowed. The host then waits for the target to indicate that it is ready for HTC communications. See the **"Bootloader Message Interface"** on page G-1 for details about BMI.	
Startup Dataset	The target then looks for a startup DataSet, which may indicate additional (non-standard) initializations.	

Table 1-3. **Startup Sequence Phases (continued)**

Phase	Description	
Application	After leaving BMI phase, the target handles a few more essential initializations. Notably, it initializes clocking so that the AR6000 runs at the desired speed. The target then jumps to a designated target application address. The default application is the athwlan WLAN application, which is preloaded into ROM or flash. In the case of a flash upgrade, the host would use BMI to load the target flash application into target RAM and to tell the target to execute that rather than the athwlan application.	
	Board Data	The athwlan WLAN application begins by reading the Board data (which includes radio calibration data). Board data may be stored in a target module EEPROM, in a host file, or elsewhere.
	Regulatory Information	Next, the WLAN application locates regulatory information for the regulatory domain specified in the board data. Such information is loaded from a DataSet that contains the entire regulatory database.
	Analog Configuration Information	Next, the WLAN application reads the Analog Configuration DataSet that corresponds to the crystal in use. A separate DataSet exists with a distinct ID for each supported crystal speed. Only the DataSets used need to be present.
	Host Communications	Finally, the WLAN application initializes target-side wireless DMA buffers, all wireless hardware, WMI communications, and HTC communications. As HTC is initialized, it sends an interrupt to the host. The host then initializes host-side HTC and WMI. At this point, a user can use the various host applications to start wireless communications.

Applications

Several applications are used in conjunction with the AR6000. This chapter briefly introduces the major applications. See the source and the run-time help built into each command for more detailed usage information.

Table 1-4 describes the major application types.

Table 1-4. **Major Application Types**

Application Type	Description
Host Tools	Host tools are programs that execute on the host. The reference host implementation uses host tools to: ■ Augment the user interface that allows an end-user to configure wireless operation ■ Apply patches and initialize the AR6000 chipset The host tools are command line driven and typically used from a Linux command shell or script. Windows CE/Mobile versions of these tools are also provided.
Target Applications	Target applications are special-purpose applications that run instead of the normal athwlan WLAN application. These special applications are loaded from the host into chipset RAM. For example, a target application is used for the test command tool.
Host and Target Test Applications	Host and target test applications can be useful during development, debug and bringup.

Table 1-5 describes the applications (note that the actual applications available are OS and package dependent).

Table 1-5. **Applications**

Application	Description
athwlan	This target-side application embodies the WLAN driver. Without this application, the AR6000 has no understanding of IEEE 802.11. The ELF image of the athwlan application are provided for use during bringup and debug. They are not intended for redistribution to end users. This target firmware is distributed both as an ELF image, **athwlan.out**, and as a binary, **athwlan.bin**. The binary image is downloaded at startup time from the host to target RAM.
bmiloader	The bmiloader application is a host-side application used before the AR6000 device is enabled. It offers a Command Line Interface (CLI) to BMI messages, which are used to influence AR6000 system behavior. For instance, **bmiloader** allows the host system to read/write individual registers and memory. It is used to install patches, to set some operational parameters, to download radio calibration data, and to load alternate target applications such as the target flash application. The bmiloader application interacts with firmware during the target's BMI phase of initialization. Once that BMI phase has finished, **bmiloader** cannot be used again until the target is reset. See "Startup Sequence" on page 1-7 for more information. The application source is located at **host/tools/bmiloader/**. NOTE: bmiloader is typically used in the Linux host software to facilitate script-driven initialization of AR6000 devices.
eeprom	The EEPROM host-side application is used to manage an EEPROM attached to the target. When an EEPROM is available (e.g. on systems without flash), it is used to store instance-specific board data. The eeprom application can be used to: ■ Read board data from the EEPROM into a host file ■ Write board data from a host file to the EEPROM ■ Transfer board data from the EEPROM directly to the proper location in target RAM where it may be used by firmware
fwpatch	The fwpatch reference host-side utility is used to apply an a ROM patch distribution format (RPDF) file to firmware. It reads and parses the RPDF file and uses BMI messages to apply ROM patches accordingly.
wmiconfig	This host-side application provides a command line interface (CLI) to configure wireless behavior by driving WMI commands to the target. For example, users can configure scan operations, examine wireless statistics, and generally influence wireless behavior. The application source is located at **host/tools/wmiconfig**.

Clocks and Crystals

AR6000 modules support various crystal speeds, including 19.2 MHz, 24 MHz, 26 MHz, 38.4 MHz, 40 MHz, and 52 MHz. During startup, firmware handles programming of the various system clocks.

Configuration/Customization Methods

Table 1-6 describes configuration/customization mechanisms.

Table 1-6. **Configuration/Customization Mechanisms**

Mechanism	Description
BMI	The bootloader messaging interface (BMI) is used during the AR6000 device startup, before wireless activities have started to apply patches, to load RAM-based software, override default initialization for AR6000 registers, set option flags, and to override firmware settings (variables). See "Bootloader Message Interface" on page G-1.
Board Data	Board data contains items that influence wireless operation such as MAC address, regulatory domain, band limitations, and radio calibration data. Board data is specific to a particular board instance. See "DataSets" on page C-1.
Interest Area	An area in target RAM that is used for simple configuration items, status, and software switches. The area is cleared by firmware at startup. The host may choose to write special values to override settings, then firmware uses those values during operation. See "Messaging and Communications" on page B-1 for more information about the Host Interest Area.
Patch DataSets	Patch DataSets are collections of patches usually stored in flash that can perform arbitrary initializations on the AR6000, much like BMI. The startup patch DataSet is applied during startup just before BMI phase. The GPIO patch DataSet is applied during GPIO initialization. See "DataSets" on page C-1.
Target Application	After the BMI phase has completed, the AR6000 executes the wireless LAN (athwlan) application, which can be altered in several ways. One way is to use the BMI APP START command to change the application starting address to some other location in RAM. It is also possible to deploy a startup patch that changes the application start address.
WMI	The wireless module interface (WMI) is used during wireless LAN operation to configure the wireless interface, including foreground and background scan parameters, filters, wireless band (802.11a, 802.11b/g/n), wireless channels/frequencies, SSID probe parameters, wireless authentication, encryption, and TKIP countermeasures, bad access points (APs), association information, ad hoc beacon interval, listen interval, disconnect interval, power management parameters, Tx power level, and prioritized data. See "Wireless Module Interface (WMI)" on page A-1.

2 Wireless Operation

The AR6000 includes CPU and memory that can be loaded with an appropriate application and can communicate over the interconnect with the host. However, much of the AR6000 hardware is specialized wireless LAN (WLAN) hardware with the primary purpose of running the Atheros WLAN application (athwlan) firmware.

Wireless Control Commands

The AR6000 accepts wireless control commands through the HTC transport and HIF interconnect mechanisms. These commands are HTC-encapsulated wireless module interface (WMI) messages. Through WMI control messages, the host can start and stop the wireless module, configure it for a particular behavior, control scanning behavior, influence power utilization, handle authentication and encryption, track wireless status and statistics, etc. WMI commands, the WMI message format, and the WMI message header are described in "Wireless Module Interface (WMI)" on page A-1. Reference host source code for WMI is in **host/wmi/**. "Scan and Roam" on page 3-1 provides useful background information on scanning and roaming features controlled using WMI commands.

Data is sent to and received from the AR6000 through the HTC transport layer and HIF interconnect. Data to send to the wireless module for transmission over the air is encapsulated using standard IEEE 802 Ethernet encapsulation with a WMI-specific header. Thus a data Tx message starts with a WMI header followed by 802.3 MAC, 802.2 LLC, and SNAP headers, followed by an IP packet. When the AR6000 receives such a message, the target uses header information to construct appropriate 802.11 encapsulation and then transmit it over the air. Conversely, when the AR6000 receives IEEE 802.11 packets over the air, it consumes the 802.11 encapsulation and synthesizes standard IEEE 802 Ethernet encapsulation with an additional header before forwarding the packet to the host.

Mechanics of Wireless Messaging

The mechanics of wireless messaging are embodied in the HTC layer of the reference host software. These mechanisms include AR6000-specific message flow control, message signaling, mailbox addressing, message encapsulation, and message initialization. See Appendix B, "Messaging and Communications".

Power Management

The AR6000 uses power management techniques that allows it to shut down any hardware components when not in use. The AR6000 also employs standard IEEE 802.11 techniques and some proprietary techniques that allow for power-efficient wireless communications. Through WMI messages, the host can influence AR6000 operation in a number of ways that permits it to trade-off power utilization against performance. The host can reduce Tx power, increase the listen interval, reduce scanning and roaming activities, change the beacon interval and announcement traffic indication message (ATIM) window, and adjust other operational parameters. Some WMI commands with the most direct influence on power consumption include:

- "SET_POWER_MODE"
- "SET_TX_PWR"
- "GET_TX_PWR"
- "SET_POWER_PARAMS"
- "SET_BEACON_INT"
- "SET_IBSS_PM_CAPS"
- "SET_RETRY_LIMITS"

On some interconnects, the host system may also control the speed at which the interconnect is clocked. Significant power savings (at the expense of performance) can be achieved by lowering clock speeds. Through BMI (or the startup patch DataSet), the host can reduce AR6000 system and CPU clocks for additional power savings. Additional information is also contained in the Appendix K, "WLAN Power Save Options".

Wireless Statistics

The AR6000 maintains counters of significant events to report statistics to the host upon request. The host requests this data using the WMI command **GET_STATISTICS**, which causes the AR6000 to send a **REPORT_STATISTICS** event with a **TARGET_STATS** payload. This payload includes statistics for Rx, Tx, connections, power management, encryption, errors, etc. When wireless statistics are reported, the AR6000 resets all statistics counters to 0. Wireless statistic counters silently wrap. It is the responsibility of the host to read the counters periodically before the wrap loses information or to use statistical methods over a short interval to sample wireless statistics.

Heartbeat

AR6000 chipset status is not monitored by a periodic heartbeat. A host implementation may construct such a monitor using a host-side timer that periodically causes any WMI GET commands requiring a response from the AR6000 to be issued.

Security

The AR6000 and reference host software support WEP, TKIP, and AES encryption and both enterprise and home/SOHO versions of WPA and WPA2 and LEAP authentication. All security is managed using these WMI commands (see "Wireless Module Interface (WMI)" on page A-1):

- "CONNECT"
- "ADD_CIPHER_KEY"
- "DELETE_CIPHER_KEY"
- "SET_AKMP_PARAMS"
- "SET_PMKID"
- "SET_PMKID_LIST_CMD"
- "SET_TKIP_COUNTERMEASURES"

Support for various EAP (authentication) methods varies between versions of the OS. See Figure 2-2-1.

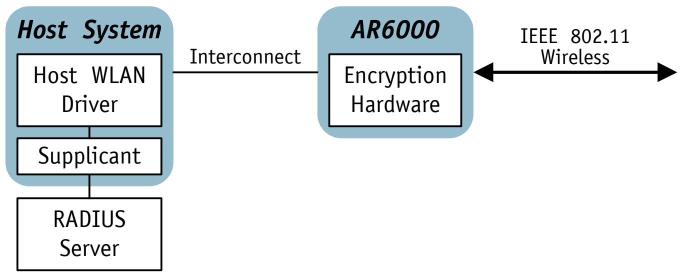

Figure 2-1. **Security**

Wi-Fi Protected Access

The goal of WiFi protected setup (WPS) is to simplify the security setup and management of WiFi networks.

Performance

Wireless performance is a function of:

- AR6000 hardware and firmware
- Host/target interconnect (especially its width and speed)
- Host platform and software
- Configuration (including AP configuration and choice of encryption)
- Trade-offs between power savings and speed, external factors such as interference over the air, etc.

Wireless performance is typically expressed in terms of:

- Uplink throughput (AR6000 client to AP)
- Downlink throughput (AP to AR6000 client)
- Uplink and downlink latency

To support voice over wireless (VoW), it is also useful to consider real-time characteristics of a wireless solution, especially in the context of a desire to conserve power. The AR6000 is designed to provide excellent uplink and downlink throughput with low latency and to be useful in VoW implementations.

3 Scan and Roam

The AR6000 device scans the list of available channels periodically when attempting to establish a connection or when explicitly requested by the host.

A foreground scan:

- Occurs when the AR6000 device is not connected
- Discovers all currently available wireless networks
- Finds the best BSS to join

A background scan:

- Occurs when the AR6000 device is connected to a network
- Discovers potential roaming candidates
- Maintains them in order of preference
- Finds new networks for the host to connect to

The frequency of the device scans is adjusted by the host using the WMI **"SET_SCAN_PARAMS"** command, allowing the host to make trade-offs between power consumption and the ability to react to changes in the strength and availability of wireless networks. This WMI command also allows the host to specify that ratio of short scans to normal scans.

A short scan is restricted to the current profile's SSID plus any SSIDs explicitly listed for probing by the **"SET_PROBED_SSID"** command. The profile in this context is information sent to the device in a **"CONNECT_CMD"**. It includes information such as SSID, cipher, and type of authentication.

The host uses the **"SET_CHANNEL_PARAMETERS"** command to limit which channels are scanned. This command also impacts scan time and thus power consumption. The host can cause the device to begin scanning using the **"START_SCAN"** command, and this command resets the relative time used by the device to trigger a scan. During scan operations, the device discovers basic service sets (BSSs) and forwards this information to the host using a **"BSSINFO"** event. Note that some BSSs represent different wireless networks while others represent different APs for the same network.

The host restricts which BSS discoveries generate **"BSSINFO"** events using the **"SET_BSS_FILTER"** command. With this command, the host disables all "BSSINFO" events, limits the **"BSSINFO"** events to a particular SSID, or requests the AR6000 to actively probe for specific networks. The host uses the information in the **"BSSINFO"** event to learn about the available wireless networks, their signal strength, and their properties (e.g., encryption).

Connection Services

Scanning and roaming are managed in the AR6000 device through the connection services (CS) module, which maintains a notion of the current connection state (see Table 3-1). The CS module is responsible for establishing a wireless connection, scanning, and roaming. Its features include:

- Small memory footprint
- 802.11a/g/n scanning and roaming
- Station (STA) only (no access point (AP) support)
- Supports both active and passive scanning
- Supports hidden SSID
- Supports CCX 3.0
- World mode support
- 802.11d support
- Low power consumption
- Host/application control of tunables
  - Channels scanned
  - Foreground scanning interval
  - Background scanning interval
  - Channel dwell time during active scanning
  - Channel dwell time during passive scanning
  - Disconnect timeout
- PMKID caching support

The AR6000 device manages scanning and roaming with the CS state machine. Table 3-1 shows the CS state machine states.

Table 3-1. **Connection Services State Machine States**

State	Description
DISCONNECTED	The AR6000 device is not connected to a wireless network. The CS module's main function is performing foreground scanning.
CONNECTED AT HOME	The device is connected to the wireless network on the channel that the BSS is connected to. Host data is transmitted and received.
CONNECTED OFF HOME	The device is connected to the wireless network but has temporarily changed channels to perform a requested operation. Background scanning is the most common off-channel operation requested. Host data is not transmitted or received in this state.
ROAMING	The device attempts to establish a wireless connection by connecting to its current BSS or finding another suitable candidate. Host data is not transmitted or received in this state. The device normally enters this state upon receiving a connect command or notification that the current connection is weak or lost.

Scanning

The main goal of scanning is to discover and obtain updates of available wireless networks and their properties. A foreground scan occurs when:

- The STA is not connected to a BSS, and
- No traffic flow other than scan traffic is occurring.

A background scan occurs when the STA is connected to a BSS or is in IBSS mode.

The host can change the frequency and time spent per channel for these scan types independently, allowing for trade-off such as time-to-connection versus power consumed.

The host also has the ability to limit the number of channels that the AR6000 scans using a **"SET_CHANNEL_PARAMETERS"** command. The host keeps track of various channels of interest used by the wireless network(s), limiting the set of channels which results in power savings. The host is only allowed to change the number of channels while disconnected.

The target performs all necessary scanning and roaming functions to establish and roam to a wireless connection. The host can choose to receive beacons and probe responses to also obtain a scan list.

During foreground scanning:

- The AR6000 device processes incoming beacon and probe responses and maintains the state regarding
  - Which channels had wireless networks and
  - The identity of those wireless networks
- This subset of channels is scanned once the host issues a **"CONNECT"** command

A secondary goal of scanning is determining the regulatory domain. The AR6000 firmware processes incoming beacons and probe responses for identification elements (IEs) indicating the country and possible Tx power constrains.

During background scanning, the AR6000 firmware also processes incoming beacons and probe responses to create a BSS list of roaming candidates. The AR6000 device maintain a list of up to eight potential roaming candidates. All roaming decisions are made using this BSS list. Only BSSs that are valid roaming candidates are entered into the roaming list.

Active Scanning

The wireless module performs active scanning by transmitting a probe request and waiting for a probe response. After changing the channel for scanning, the module sends these probe requests. See Figure 3-3-1.

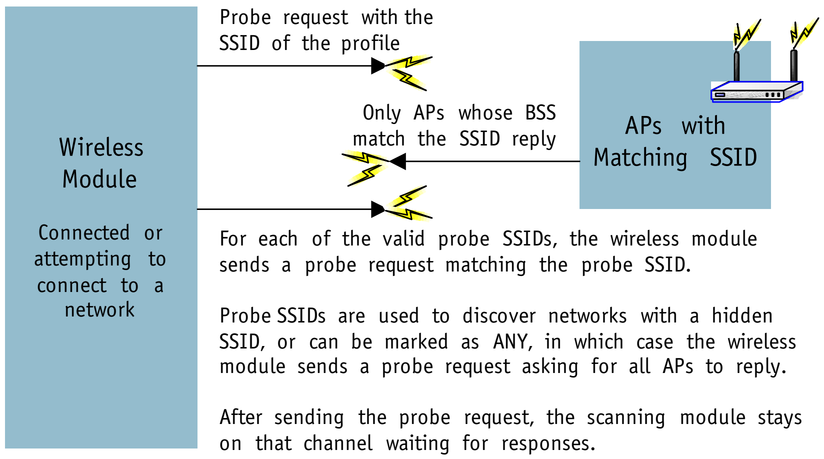
Wireless Module
Connected or attempting to connect to a network
Probe request with the SSID of the profile
Only APs whose BSS match the SSID reply
APs with Matching SSID
For each of the valid probe SSIDs, the wireless module sends a probe request matching the probe SSID.
Probe SSIDs are used to discover networks with a hidden SSID, or can be marked as ANY, in which case the wireless module sends a probe request asking for all APs to reply.
After sending the probe request, the scanning module stays on that channel waiting for responses.

Figure 3-1. **Active Scanning**

After changing the channel for the purposes of scanning the wireless module sends these probe requests:

- If the wireless module is connected or attempting to connect to a network, it sends a probe request with the SSID of the active profile. Only APs whose BSS match the SSID reply.
- For each valid probe SSID, the wireless module sends a probe request matching the probe SSID to discover networks whose SSID might be hidden. The **"SET_PROBED_SSID"** command configures the SSIDs to probe, up to a maximum of six. Set the ANY_SSID_FLAG in the **"SET_PROBED_SSID"** command if the SSID should not be transmitted in the probe request.

After sending the probe request, the scanning module remains on that channel awaiting responses. The amount of time spent waiting, is referred to as the active channel dwell time and is customizable by the host.

World Mode Support and Passive Scanning

In world mode, many channels are marked as passive scan only. In a passive scan, the wireless device is not allowed to transmit and thus cannot discover networks by sending probe requests. It discovers networks by listening for beacons. The amount of time the scanning module stays on a particular channel listening is referred to as the passive channel dwell time and it is customizable by the host. The default value is 105 ms.

If the wireless module receives a beacon while listening, it marks that channel as temporarily active and performs the active scan procedure. The channel is marked temporarily active for three scans, then returns to its passive status.

Foreground Scan

Foreground scanning is performed when the CS module is disconnected. During a foreground scan the CS module changes to a channel, performs a scan, changes to the next channel, performs a scan, and so on until all channels have been scanned. A foreground scan is started on one of these conditions:

- Receipt of a **"START_SCAN"** command while in the disconnected state
- Receipt of a **"CONNECT"** command
- The ROAMING algorithm is unable to find a BSS to join
- Expiration of the scan interval timer

The internal scan timer is controlled using the **"SET_SCAN_PARAMS"** command. It uses a binary backoff algorithm that doubles the interval between scans until a ceiling is reached. The host can configure the scans to be short or long. A long scan goes through the list of all the channels, but a short scan goes through the list of channels were a beacon or probe response was seen. The host can configure the number of short scans between two long scans using the short scan ratio (shortScanRatio) in **"SET_SCAN_PARAMS"**. The default value is three short scans between two long scans.

By default, the foreground start interval is one second and the foreground end interval is one minute. Using these defaults, the first interval between scans is one second, the second two seconds, the third four seconds, and so on until an interval of one minute is reached. All subsequent intervals are then one minute. Upon initial reset, the foreground scan interval is set to the default end interval (one minute) to minimize power consumption. The foreground scan interval adopts the start interval on receipt of a **"CONNECT"** command.

Background Scan

A background scan is performed when the AR6000 is connected to a network. A background scan is performed one channel at a time. The CS module changes the channel, performs a scan, then returns to the home channel. The data path is able to send/receive any application data before the CS module scans the next channel. A background scan starts on one of these conditions:

- Receipt of **"START_SCAN"** command while in the connected state
- The interval scan timer fires

The internal background scan timer is periodic. By default it is disabled. When the interval timer triggers a background scan, the data path has higher priority. The CS module only performs a scan opportunistically when the AR6000 reach an idle condition.

When a **"START_SCAN"** command triggers a background scan, the CS module still waits for an idle state before performing a scan.

Roaming

A single infrastructure wireless network is frequently serviced by more than one AP, but the AR6000 connects to only one BSS at a time. If the AR6000 device is connected, the CS module delays breaking the connection until it is ready to connect to the new BSS. If the same BSS is selected then the CS module simply returns to the CONNECTED state and regular traffic resumes.

Roaming support in the AR6000 device optimizes wireless networking by transparently switching between BSSs in an ESSID. To do this, whenever the device is connected or trying to connect, it maintains a list of potential BSSs to connect to. If a roam operation is triggered, the device updates link quality data for a potential BSS, then selects the best BSS from this roaming candidate list and switch the wireless connection appropriately. The roaming state is described as consisting of the phases trigger, roam scan, and BSS selection.

Trigger

Roaming triggers are events that cause the CS module to enter the roaming state. In the roaming state, the AR6000 attempt to establish or re-establish a wireless connection. If the AR6000 is connected, the CS module delays breaking the connection until it is ready to connect to the new BSS. If the same BSS is selected then the CS module returns to the CONNECTED state and regular traffic resumes. Table 3-2 shows the possible triggers.

Table 3-2. **Roaming Triggers**

Trigger	Description
"CONNECT" command	The host issues the **CONNECT** command and provides a profile by scanning to look for matching BSSs. The list of roaming candidates is empty and the CS module starts a scan to find all available BSSs. The scan starts as a foreground scan. If a valid candidate is found, a connection is established and the scan finishes as a background scan. If no candidate is found, a foreground scan follows a binary backoff algorithm.
MLME disconnect event	MLME can issue a DISCONNECT event notifying CS it lost connection or failed to connect to the BSS, possibly due to a failed 802.11 authentication or 802.11 association, unavailability of management frames, or receipt of a de-authentication or de-association frame from the connected AP.
Beacon miss (BMISS) interrupt	In the device, beacon processing is normally handled by hardware. Only certain events cause the hardware to interrupt the firmware, such as a BMISS interrupt. The hardware is programmed to interrupt the CS module if more than 15 consecutive beacons are missed.
Low average beacon RSSI threshold interrupt	The CS module normally measures link quality with its current BSS and detects changes to evaluate its current BSS using the RSSI beacon. Actual RSSI can vary and requires an averaging function, as provided by AR6000 hardware, with a programmable threshold allowing a software interrupt when the average RSSI drops below the programmed threshold.

Roam Scan

When one of the triggers described Table 3-2 fires and the CS module enters the roaming state, the CS module updates the RSSI of all members in the roaming candidate list using a roam scan consisting of sending a directed probe request to each of the candidates. However, the CS module does not wait for a response. It obtains the RSSI from the ACK to the probe request then moves on to the next candidate, allowing the CS module to refresh the relative strength of the available BSSs in a minimum amount of time.

BSS Selection

After performing a roam scan and updating the strength of the roaming candidates the CS module selects which BSS to connect to. The host can alter BSS selection, enabling the host to set the priority of selection of BSSs in the roaming list. Table 3-3 shows the possible selection modes.

Table 3-3. **BSS Selection Modes**

Mode	Description		
BSS selection based on the RSSI	The algorithm is based on three metrics: ■ The RSSI of the BSS ■ The first derivative of the RSSI. BSSs whose RSSIs are growing stronger are preferred over BSSs whose RSSIs are growing weaker. ■ The BSS_UTILITY. The BSS_UTILITY is a multiplier to the RSSI used by software to influence the selection. By default, the BSS_UTILITY given to a BSS is 10. These metrics derive what is called the ROAM_UTILITY of a BSS. The BSS with the highest ROAM_UTILITY is selected.		
	Factors that increase BSS_UTILITY	Current BSS	Provides for a some stickiness and minimizes a constant bouncing when the STA has two or more candidates with relatively the same strength. The BSS_UTILITY is only increased after the connection is deemed stable. For open connections, the CS module waits for 2 seconds before deeming the connection stable. For WPA connections, the CS module waits until keys are plumbed before deeming the connection stable.
		PMKID	A BSS with a cached PMKID is preferred over a BSS without one
	Factors that decrease BSS_UTILITY	No 802.11 ACK received for a roam scan probe	
		Receipt of a MLME disconnect event	
		Receipt of a BMISS	
	Once a selection has been made that differs from the current AP, the switch is:	Wake up radio if necessary	
		Deauth from current AP (a deauth frame may not be sent)	
		Start disconnect timer (if timer expires, the host is notified of the disassociation)	
		Disassociate from current AP	
		Start join, auth, and assoc operation on new AP. Join operation should not require sending a probe request and waiting for a response. Any required information should be obtained during a background scan.	
BSS Selection Based on the Host Bias	The host can choose to set its priority to various BSSs and influence BSS selection on the AR6000. The value assigned to each BSS is called the BSS host BIAS. The host can set each BIAS using **SET_ROAM_CTRL** with roamCtrlType set to **SET_HOST_BIAS**. Each BSS is assigned a signed 8-bit BIAS value. Once each host BIAS is set, the mode must be changed. The **SET_ROAM_CTRL** command with roamCtrlType set to **SET_ROAM_MODE** changes the mode. The host bias mode is selected by setting roamMode to **HOST_BIAS_ROAM_MODE**.		
Lock to the Current BSS	The AR6000 can be directed to not roam and stick with a BSS. The command **SET_ROAM_CTRL** with roamCtrlType set to **SET_ROAM_MODE** and roamMode set to **WMI_LOCK_BSS_MODE** prevents the AR6000 from roaming to another BSS.		

Default BSS RSSI selection:

- Host bias based selection
- No roaming lock to the current BSS

The **SET_ROAM_CTRL_CMD** configures these modes. The **GET_ROAM_TBL_CMD** can retrieve the AR6000 roaming table and metrics used to select the BSS. The AR6000 boot-up setting is the WMI_DEFAULT_ROAM_MODE.

All the algorithms are performed only on BSSs present in the roaming BSS list. Policy decisions regarding the validity of a particular BSS roaming candidate (SSID, regulatory, network type, security properties, etc.) is performed prior to adding a BSS to the list, reducing the working set and the number of checks needed for the algorithm to performed. It also enables a less reactive process to perform policy checks and add/remove BSSs from the roaming list.

BSS Roaming Candidate List

The roaming candidate list is a list of up to eight BSSs that match the current profile and are all valid candidates for roaming purposes. New candidates join the candidate list during a background scan. The list is erased when the host issues a **"DISCONNECT"** command.

Once the list is full, the BSS with the smallest BSS_UTILITY is removed from the list to make room for a new BSS.

During a background scan, the CS module keeps track of whether a new BSS was added to the candidate list. After the background scan completes, if the list has changed a BSS NEIGHBOR report event is sent to the host with the contents of the list. This event can be used by the host to trigger pre-authentication of APs in WPA2 networks.

Disconnect Event to Host

When the AR6000 device loses connection with the BSS, they do not send a **"DISCONNECT"** event to the host immediately. Instead the device waits for a disconnect timeout period before sending the **"DISCONNECT"** event to the host. The host can configure this period with the **SET_DISC_TIMEOUT** command. The default value is 10 seconds.

Channel Management

In the WLAN driver, multiple modules can perform a given operation on a particular channel independent of each other and independent of the current connection. These modules include the:

- Scanning module
- Roaming module
- Radio measurements module (CCX only)

Rather than each module independently attempting to manage channels, a centralized channel manager provides a mechanism to request an operation on a given channel for a certain duration. The channel manager ensures that any traffic for a current connection is stopped before it switches channels, and that traffic is resumed after returning to the home channel. If a higher priority operation is added while a low operation is in progress, the channel manager cancels the lower priority operation then starts the higher priority operation.

4 Wake-on-Wireless

This section describes the customer requirements for Wake on Wireless (WoW) implementation with the AR6000 device.

Host Wakeup Overview

The Host Wakeup feature allows a Host processor to go to suspend mode while the AR6000 device is operational in a low power state. This feature could be used in a VoIP scenario where the device wakes up the host on an incoming VoIP call. The device can be configured to wake up the host on a configurable wake-up pattern. This allows the device to conserve power and increase the standby time.

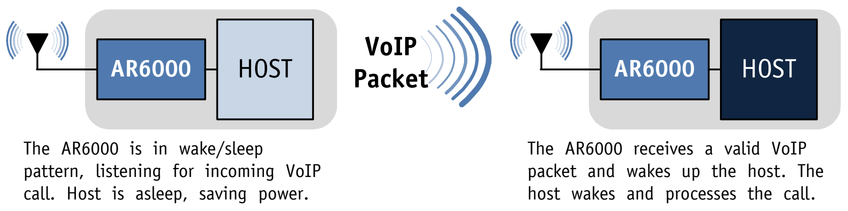
AR6000
HOST
VoIP Packet
AR6000
HOST
The AR6000 is in wake/sleep pattern, listening for incoming VoIP call. Host is asleep, saving power.
The AR6000 receives a valid VoIP packet and wakes up the host. The host wakes and processes the call.

Figure 4-1. **Host Wakeup in VoIP Scenario**

The wakeup setup depends upon the capabilities of the host platform. If the host is capable of waking up using SDIO interrupt (DAT1), the AR6000 sends an interrupt when incoming pattern matches the configured pattern. Alternately an AR6000 GPIO line (default GPIO14_ can be configured to assert (active high 1.8 V) on matched pattern and the AR6000 GPIO line tied to a host (wake-capable) GPIO line to wake the host.

The host application sets a packet filter list before it enables host wakeup. A filter list may contain multiple filters, the structure is as follows. Each filter list or filter can have variable size. Currently the maximum size of a pattern/ mask is limited to 64 bytes.

```
struct {
    A_UINT8 wow_valid_list;
    A_UINT8 wow_list_id;
    A_UINT8 wow_num_filters;
    A_UINT8 wow_total_list_size;
    WOW_FILTER list[WOW_MAX_FILTERS_PER_LIST];
} WOW_FILTER_LIST;
struct {
    A_UINT8 wow_valid_filter;
    A_UINT8 wow_filter_id;
    A_UINT8 wow_filter_size;
    A_UINT8 wow_filter_offset;
    A_UINT8 wow_filter_mask[WOW_MASK_SIZE];
    A_UINT8 wow_filter_pattern[WOW_PATTERN_SIZE];
} WOW_FILTER;
```

Only the bytes with mask not equal zero will be compared. Bytes with mask=0 will be ignored. Offset should be 4-byte aligned and filter_size should be multiple of 32 bits.

Each application can only send one filter_list to the device, but multiple applications may send one filter_list each with distinct filter_list ID.

There is only one filter list maintained at the target implementation with a filter_list_id of 0. All filters added by multiple applications will all go into this single filter list.

Five WMI commands enable an application to configure host wakeup. The command details are provided in WMI command reference.

- WMI_SET_HOST_SLEEP_MODE_CMD
- WMI_SET_WOW_MODE_CMD
- WMI_GET_WOW_LIST_REPLY
- WMI_ADD_WOW__PATTERN_CMD
- WMI_DEL_WOW_PATTERN_CMD

When Host Wakeup is enabled, all the filter list will be active. Any packet filter list that is not used needs to be removed immediately. Application should ADD/REMOVE PF_LIST when host wakeup is disabled.

Host-Target Interface

Support Requirements

Host

- Host supplies power to the target even when in suspend mode
- Host must support an external GPIO line which is wakeup capable (i.e., the host must be able to wake the CPU when the AR6000 raises an interrupt using the AR6000 GPIO line tied to it)

Target

- The target must get power even when host is in suspend mode
- The target (AR6000) should have a programmable way to pull the AR6000-GPIO line tied to the host wakeup

wmiconfig Commands

SET HOST SLEEP MODE	
wmiconfig --sethostmode <awake/asleep>	Used by the host to tell the target what its current state is. When the host sleep mode is set to be awake, all packets are passed up to the host, without any filtering based on the WOW patterns defined (even if WOW is enabled). When the host sleep mode is set to asleep, and WOW is enabled, only packets that pass any of the WOW filters are sent up to the host.
SET WOW MODE	
wmiconfig -setwowmode <enable/disable>	Used by the host to enable or disable all WOW filtering. If disabled, none of the WOW patterns added will be used as a filter. If WOW is enabled and the host mode is asleep, the target will only queue up packets to the host that pass any of the WOW filters added by the host.
GET WOW LIST	
wmiconfig -getwowlist <list-id>	Used by the host to get the list of all WOW patterns contained in the list with the specified list ID. In the current implementation, we only have one list. The default list ID is 0. See also OID_PNP_WAKE_UP_PATTERN_LIST.
DELETE WOW PATTERN	
wmiconfig -delpattern <wowlistid> <wowpattern-id>	Used by the host to delete a WOW pattern (with the specified pattern ID) from the list with the specified list ID. In the current implementation, we only have one list. The default list ID is 0. See also OID_PNP_REMOVE_WAKE_UP_PATTERN.
ADD WOW PATTERN	
wmiconfig -addpattern <wowlistid> <pattern-size> <pattern-offet> <pattern> <pattern-mask>	Used by the host to add a WOW pattern to the list with the specified list ID. In the current implementation, we only have one list. The default list ID is 0. Note that the pattern size is no longer flexible and cannot exceed 64 bytes. See also OID_PNP_ADD_WAKE_UP_PATTERN.

To enable filtering on the basis of the source and destination MAC, the implementation assumes that the frame contains a 14-byte MAC header. The pattern offset assumes that offset 0 is the beginning of this MAC header. Assuming the frame contains a ping packet, it looks like:

Name	Size (bytes)
RESERVED	2
dest MAC	6
source MAC	6
sequence + fragment	2
SNAP dr	6
Protocol type	2

Name	Size (bytes)
IP header	20
ICMP packet data	

Sample addwowpattern Commands

Source MAC=00:10:c6:df:f9:bd

Dest MAC (STA)=00:03:7f:01:40:48

Protocol type = IP = 0800

Protocol type in IP header = ICMP = 01

- **To filter for the destination MAC** (starts at offset 2)
  wmiconfig --addwowpattern 0 6 2 00037f014048 ffffffffffff
- **To filter for the source MAC** (starts at offset 8)
  wmiconfig --addwowpattern 0 6 8 0010c6dff9bd ffffffffffff
- **To filter for the SNAP header** (starts at offset 16)
  wmiconfig --addwowpattern 0 6 16 aaaa03000000 ffffffffffff
- **To filter for IP packets** (offset 22)
  wmiconfig --addwowpattern 0 2 22 0800 ffff
- **To filter for ICMP packets (both replies and request)** (offset 33)
  wmiconfig --addwowpattern 0 1 33 01 ff

Target-Side Statistics

As part of the wmiconfig -getTargetStats command, Wake on Wireless statistics are reported. WoW statistics maintained are:

tx_packets	Number of packets transmitted
tx_bytes	Number of bytes transmitted
tx_unicast_pkts	Number of Unicast packets transmitted
tx_unicast_bytes	Number of Unicast bytes transmitted
tx_multicast_pkts	Number of multicast packets transmitted
tx_multicast_bytes	Number of multicast bytes transmitted
tx_broadcast_pkts	Number of broadcast packets transmitted
tx_broadcast_bytes	Number of broadcast bytes transmitted
tx_rts_success_cnt	Transmit success count
tx_packet_per_ac[4]	Transmitted packets per AC
tx_errors	Number of transmit errors

A
Wireless Module Interface (WMI)

This section describes the format and the usage model for WMI control and data messages between the host and the AR6000-based targets. The header file **include/wmi.h** contains all command and event codes, constants, as well as structure typedefs for each set of command and reply parameters.

Data Frames

The data payload transmitted and received by the target follows RFC-1042 encapsulation and thus starts with an 802.2-style LLC-SNAP header. The WLAN module completes 802.11 encapsulation of the payload, including the MAC header, FCS, and WLAN security related fields. At the interface to the message transport (HTC), a data frame is encapsulated in a WMI message.

WMI Message Structure

The WMI protocol leverages an 802.3-style Ethernet header in communicating the source and destination information between the host and the AR6000 modules using a 14-byte 802.3 header ahead of the 802.2-style payload. In addition, the WMI protocol adds a header to all data messages:

`{`

`A_INT8`	`rssi`	The RSSI of the received packet and its units are shown in db above the noise floor, and the noise floor is shown in dbm.
`A_UINT8`	`info`	Contains information on message type and user priority. Message type differentiates between a data packet and a synchronization message.
`A_UINT16`	`info`	Contains information on sequence number used and other optional parameters.
`A_UINT16`	`info2`	Reserved

`} WMI_DATA_HDR`

User priority contains the 802.1d user priority info from host to target. Host software translates the host Ethernet format to 802.3 format prior to Tx and 802.3 format to host format in the Rx direction. The host does not transmit the FCS that follows the data. **MsgType** differentiates between a regular data packet (**msgType**=0) and a synchronization message (**msgType**=1).

Data Endpoints

The AR6000 chipset provides several data endpoints to support quality of service (QoS) and maintains separate queues and separate DMA engines for each data endpoint. Data endpoints are abstract message pipes provided by the HTC layer. The HTC layer is responsible for transferring messages to the chipset. A data endpoint can be bi-directional.

If QoS is desired over the interconnect, host software must classify each data packet and place it on the appropriate data endpoint. The information required to classify data is generally available in-band as an 802.1p/q style tag or as the ToS field in the IP header. The information may also be available out-of-band depending on the host DDI.

Connection States

Table A-1 describes the AR6000 WLAN connection states:

Table A-1. **AR6000 Connection States**

Connection State	Description
DISCONNECTED	In this state, the AR6000 device is not connected to a wireless network. The device is in this state after reset when it sends the **WIRELESS MODULE "READY" EVENT**, after it processes a **DISCONNECT** command, and when it loses its link with the access point (AP) that it was connected to. The device signals a transition to the DISCONNECTED state with a **"DISCONNECT"** event.
CONNECTED	In this state, the AR6000 device is connected to wireless networks. The device enters this state after successfully processing a **CONNECT**, which establishes a connection with a wireless network. The device signals a transition to the CONNECTED state with a **"CONNECT"** event.

Message Types

WMI uses commands, replies, and events for the control and configuration of the AR6000 device. The control protocol is asynchronous. Table A-2 describes AR6000 message types:

Table A-2. **AR6000 Message Types**

Message Type	Description
Commands	Control messages that flow from the host to the device
Replies/Events	Control messages that flow from the device to the host. The device issues a reply to some WMI commands, but not to others. The payload in a reply is command-specific, and some commands do not trigger a reply message at all. Events are control messages issued by the device to signal the occurrence of an asynchronous event.

WMI Message Format

All WMI control commands, replies and events use the header format:

WMI_CMD_HDR Header Format

{	
`A_UINT16  id`	This 16-bit constant identifies which WMI command the host is issuing, which command the target is replying to, or which event has occurred.
`WMI_CMD_HDR`	
}	

A variable-size command-, reply-, or event-specific payload follows the header. Over the interconnect, all fields in control messages (including WMI_CMD_HDR and the command specific payload) use 32-bit little Endian byte ordering and fields are packed. The AR6000 device always executes commands in order, and the host may send multiple commands without waiting for previous commands to complete. A majority of commands are processed to completion once received. Other commands trigger a longer duration activity whose completion is signaled to the host through an event.

Command Restrictions

Some commands may only be issued when the AR6000 device is in a certain state. The host is required to wait for an event signaling a state transition before such a command can be issued. For example, if a command requires the device to be in the CONNECTED state, then the host is required to wait for a **"CONNECT"** event before it issues that command.

The device ignores any commands inappropriate for its current state. If the command triggers a reply, the device generates an error reply. Otherwise, the device silently ignores the inappropriate command.

Command and Data Synchronization

WMI provides a mechanism for a host to advise the device of necessary synchronization between commands and data. The device implements synchronization; no implicit synchronization exists between endpoints.

The host controls synchronization using the **"SYNCHRONIZE"** command over the control channel and synchronization messages over data channels. The device stops each data channel upon receiving a synchronization message on that channel, processing all data packets received prior to that message. After the device receives synchronization messages for each data endpoint and the **"SYNCHRONIZE"** command, it resumes all channels.

When the host must guarantee a command executes before processing new data packets, it first issues the command, then issues the **"SYNCHRONIZE"** command and sends synchronization messages on data channels. When the host must guarantee the device has processed all old data packets before a processing a new command, it issues a **"SYNCHRONIZE"** command and synchronization messages on all data channels, then issues the desired command.

WMI Commands

Table A-3 lists the WMI commands.

Table A-3. **WMI Commands**

Command Name	Description	Page
ABORT_SCAN	Abort the current ongoing host-initiated scan	page A-8
ADD_BAD_AP	Cause the AR6000 device to avoid a particular AP	page A-8
ADD_CIPHER_KEY	Add or replace any of the four AR6000 encryption keys	page A-9
ADD_KRK	Provided to customers who have a CCX License	—
ADD_WOW_PATTERN	Used to add a pattern to the WoW pattern list	page A-10
CLR_RSSI_SNR	Clear the current calculated RSSI and SNR value	page A-10
CONNECT_CMD	Request that the AR6000 device establish a wireless connection with the specified SSID	page A-11
CREATE_PSTREAM	Create prioritized data endpoint between the host and device	page A-13
DELETE_BAD_AP	Clear an entry in the bad AP table	page A-15
DELETE_CIPHER_KEY	Delete a previously added cipher key	page A-15
DELETE_KRK	Provided to customers who have a CCX License	—
DELETE_PSTREAM	Delete a prioritized data endpoint	page A-15
DELETE_WOW_PATTERN	Remove a pre-specified pattern from the WoW pattern list	page A-16
DISCONNECT	Disconnect from the associated AP or abort connection process	page A-16
ENABLE_RM	Provided to customers who have a CCX License	—
EXTENSION	WMI message interface command	page A-16
GET_APPIE	No longer supported	—
GET_BIT_RATE	Retrieve rate most recently used by the AR6000	page A-17
GET_CHANNEL_LIST	Retrieve list of channels used by the AR6000	page A-17
GET_FIXRATES	Retrieves the rate-mask set via the SET_FIXRATES command.	page A-17
GET_KEEPALIVE	Used to get the configured keepalive interval	page A-18
GET_PMKID_LIST_CMD	Retrieve the firmware list of PMKIDs	page A-18
GET_ROAM_DATA	Internal use for data collection; available in special build only	—
GET_ROAM_TBL	Retrieve the roaming table maintained on the target	page A-18
GET_TARGET_STATS	Request that the target send the statistics it maintains	page A-19
GET_TX_PWR	Retrieve the current AR6000 device Tx power levels	page A-19
GET_WOW_LIST	Retrieve the current list of WoW patterns	page A-19
HOST_EXIT_NOTIFY	Informs the AR6000 that the host is unloading the driver	page A-20
LQ_THRESHOLD_PARAMS	Set the link quality thresholds	page A-20
OPT_TX_FRAME	Send a special frame (special feature)	page A-20
RECONNECT	Request a reconnection to a BSS	page A-21
RSSI_THRESHOLD_PARAMS	Configure how the AR6000 device monitors and reports signal strength (RSSI) of the connected BSS	page A-21

Table A-3. **WMI Commands (continued)**

Command Name	Description	Page
RX_FRAME_FORMAT	Communicates the desired format and behavior for received frames	page A-22
SCAN_PARAMS	Determine dwell time and changes scanned channels	page A-22
SET_ACCESS_PARAMS	Set access parameters for the wireless network	page A-23
SET_ADHOC_BSSID	Set the BSSID for an ad hoc network	page A-23
SET_AKMP_PARAMS	Set multiPMKID mode	page A-23
SET_APPIE	Add application-specified IE to a management frame	page A-24
SET_ASSOC_INFO	Specify the IEs the device should add to association or reassociation requests	page A-24
SET_AUTH_MODE	Set 802.11 authentication mode of reconnection	page A-25
SET_BEACON_INT	Set the beacon interval for an ad hoc network	page A-25
SET_BIT_RATE	Set the AR6000 to a specific fixed bit rate	page A-25
SET_BMISS_TIME	Set the beacon miss time	page A-25
SET_BSS_FILTER	Inform the AR6000 of network types about which it wants to receive information using a "BSSINFO" event	page A-26
SET_BT_PARAMS	Set the status of a Bluetooth stream (SCO or A2DP) or set Bluetooth coexistence register parameters	page A-27
SET_BT_STATUS	Set the status of a Bluetooth stream (SCO or A2DP)	page A-29
SET_CHANNEL_PARAMETERS	Configure WLAN channel parameters	page A-30
SET_DISC_TIMEOUT	Set the amount of time the AR6000 spends attempting to reestablish a connection	page A-31
SET_FIXRATES	Set the device to a specific fixed PHY rate (supported subset)	page A-31
SET_FRAMERATES	Disable or enable the Tx rate of a specific type frame	page A-32
SET_HB_TIMEOUT	Stops the AR6000 from sending trigger frames after exceeding the time limit set for receiving heartbeat frames from host	page A-32
SET_HOST_SLEEP_MODE	Set the host mode to asleep or awake	page A-32
SET_IBSS_PM_CAPS	Support a non-standard power management scheme for an ad hoc network	page A-33
SET_IP_CMD	Registers the IP address of the interface with AR6000 software	page A-34
SET_KEEPALIVE	Set a keepalive interval	page A-34
SET_LISTEN_INT	Request a listen interval	page A-34
SET_LPREAMBLE	Override the short preamble capability of the AR6000 device	page A-35
SET_MAX_OFFHOME_DURATION	Provided to customers who have a CCX License	—
SET_MAX_SP_LEN	Set the maximum service period	page A-35
SET_OPT_MODE	Set the special mode on/off (special feature)	page A-35
SET_PMKID	Set the pairwise master key ID (PMKID)	page A-36
SET_PMKID_LIST_CMD	Configure the firmware list of PMKIDs	page A-36
SET_POWER_MODE	Set guidelines on trade-off between power utilization	page A-37
SET_POWER_PARAMS	Configure power parameters	page A-38

Table A-3. **WMI Commands (continued)**

Command Name	Description	Page
SET_POWERSAVE_PARAMS	Set the two AR6000 power save timers	page A-39
SET_PROBED_SSID	Provide list of SSIDs the device should seek	page A-40
SET_RATE_POLICY	Set a rate policy with an ID used in Tx frame commands	page A-41
SET_REASSOC_MODE	Specify whether the disassociated frame should be sent upon reassociation	page A-41
SET_RETRY_LIMITS	Limit how many times the device tries to send a frame	page A-42
SET_ROAM_CTRL	Control roaming behavior	page A-43
SET_RTS	Determine when RTS should be sent	page A-43
SET_SCAN_PARAMS	Set the AR6000 scan parameters	page A-44
SET_TARGET_EVENT_REPORT	Filter out events from the target	page A-45
SET_THIN_MODE	Enable thin mode on the target	page A-46
SET_TKIP_COUNTERMEASURES	Enable/disable reports of TKIP MIC errors	page A-46
SET_TX_PWR	Specify the AR6000 device Tx power levels	page A-46
SET_TX_SELECTRATES	Configure the allowable rates used to transmit packets for each operational mode	page A-47
SET_VOICE_PKT_SIZE	Set voice packet size	page A-47
SET_WHAL_PARAM	Internal AR6000 command to set certain hardware parameters	page A-48
SET_WMM	Override the AR6000 WMM capability	page A-48
SET_WMM_TXOP	Configure TxOP bursting when sending traffic to a WMM-capable AP	page A-48
SET_WOW_MODE	Enable/disable WoW mode	page A-49
SET_WSC_STATUS	Enable/disable profile check in cserv when the WPS protocol is in progress	page A-49
SNR_THRESHOLD_PARAMS	Configure how the device monitors and reports SNR of BSS	page A-49
START_SCAN	Start a long or short channel scan	page A-50
SYNCHRONIZE	Force a synchronization point between command and data paths	page A-50
TARGET_REPORT_ERROR_BITMASK	Control “ERROR_REPORT” events from the AR6000	page A-51
TEST_CMD	Multiplexes four prior TCMDs (TCMD_ENABLE, TCMD_CONT_TX, TCMD_CONT_RX, TCMD_PM) together	page A-52
TX_SGI_PARAM	Configure the use of the short guard interval (SGI)	page A-52

Name	**ABORT_SCAN**
Synopsis	Abort the current ongoing host initiated scan.
Command Parameters	None
Reply Parameters	If the command aborts the host initiated scan, scan complete event with status A_ECANCELED will be generated
Reset Value	None
Restrictions	None
See Also	"SCAN_COMPLETE_EVENT" on page A-64

Name	**ADD_BAD_AP**		
Synopsis	The host uses this command to cause the AR6000 to avoid a particular AP. The AR6000 maintain a table with up to two APs to avoid. An ADD_BAD_AP command adds or replaces the specified entry in this bad AP table. If the AR6000 are currently connected to the AP specified in this command, they disassociate.		
Command	**`wmiconfig eth1 --badap <bssid> <badApIndex>`**		
Command Parameters	`A_UINT8`	`badApIndex`	Index [0...1] that identifies which entry in the bad AP table to use
	`A_UINT8`	`bssid[6]`	MAC address of the AP to avoid
Command Values	`badApIndex = 0, 1`		Entry in the bad AP table to use
Reset Value	The bad AP table is cleared		
Restrictions	None		
See Also	"DELETE_BAD_AP" on page A-15		

Name	**ADD_CIPHER_KEY**
Synopsis	The host uses this command to add/replace any of four encryption keys on the AR6000. The **ADD_CIPHER_KEY** command is issued after the **CONNECT** event has been received by the host for all dot11Auth modes except for **SHARED_AUTH**. When the dot11AuthMode is **SHARED_AUTH**, then the **ADD_CIPHER_KEY** command should be issued before the **"CONNECT"** command.
Command	`wmiconfig eth1 --cipherkey <keyIndex> <keyType> <keyUsage> <keyLength> <keyopctrl> <keyRSC> <key>`

Command Parameters

A_UINT8	keyIndex	Index (0...3) of the key to add/replace; uniquely identifies the key
A_UINT8	keyType	CRYPTO_TYPE
A_UINT8	keyUsage	Specifies usage parameters of the key when keyType = WEP_CRYPT
A_UINT8	keyLength	Length of the key in bytes
A_UINT8	keyOpCtrl	bit[0] = Initialize TSC (default), bit[1] = Initialize RSC
A_UINT8	keyRSC[8]	Key replay sequence counter (RSC) initial value the device should use
A_UINT8	key[32]	Key material used for this connection

Command Values

{

NONE_CRYPT	= 1	
WEP_CRYPT	= 2	
TKIP_CRYPT	= 3	
AES_CRYPT	= 4	
KEY_OP_INIT_TSC	0x01	
KEY_OP_INIT_RSC	0x02	
KEY_OP_INIT_VAL	0x03	Default is to Initialize the TSC
KEY_OP_VALID_MASK	0x04	Two operations defined

} CRYPTO_TYPE

{

PAIRWISE_USAGE	= 0	Set if the key is used for unicast traffic only
GROUP_USAGE	= 1	Set if the key is used to receive multicast traffic (also set for static WEP keys)
TX_USAGE	= 2	Set for the GROUP key used to transmit frames

All others are reserved

} KEY_USAGE

Reset Value	The four available keys are disabled.
Restrictions	The cipher should correspond to the encryption mode specified in the **"CONNECT"** command.
See Also	"DELETE_CIPHER_KEY" on page A-15

Name	**ADD_WOW_PATTERN**
Synopsis	The host uses this command to add a pattern to the WoW pattern list; used for pattern-matching for host wakeups by the WoW module. If the host mode is asleep and WoW is enabled, all packets are matched against the existing WoW patterns. If a packet matches any of the patterns specified, the target will wake up the host. All non-matching packets are discarded by the target without being sent up to the host.
Command	`wmiconfig -addwowpattern <list-id> <filter-size> <filter-offset> <pattern> <mask>`
Command Parameters	`A_UINT8` `filter_list_id` — ID of the list that is to include the new pattern
	`A_UINT8` `filter_size` — Size of the new pattern
	`A_UINT8` `filter_offset` — Offset at which the pattern matching for this new pattern should begin at
	`A_UINT8` `filter[1]` — Byte stream that contains both the pattern and the mask of the new WoW wake-up pattern
Reply Parameters	None
Reset Value	None defined (default host mode is awake)
Restrictions	None
See Also	"DELETE_WOW_PATTERN" on page A-16

Name	**CLR_RSSI_SNR**
Synopsis	Clears the current calculated RSSI and SNR value. RSSI and SNR are reported by running-average value. This command will clear the history and have a fresh start for the running-average mechanism.
Command	`wmiconfig eth1 --cleanRssiSnr`
Command Parameters	None
Reply Parameters	None
Reset Value	None defined
Restrictions	None

Name CONNECT_CMD

Synopsis New connect control information (connectCtrl) is added, with 32 possible modifiers.

CONNECT_ASSOC _POLICY_USER	Assoc frames are sent using the policy specified by the CONNECT_SEND_REASSOC flag
CONNECT_SEND _REASSOC	Send Reassoc frame while connecting otherwise send assoc frames
CONNECT_IGNORE _WPAx_GROUP_ CIPHER	Ignore WPAx group cipher for WPA/WPA2
CONNECT_PROFILE _MATCH_DONE	Ignore any profile check
CONNECT_IGNORE _AAC_BEACON	Ignore the admission control beacon
CONNECT_CSA_FOLLOW _BSS	If the STA receives a channel switch announcement from the BSS (AP in infrastructure mode) and this flags is set, the STA disconnects in the current channel and change to new channel and rejoins the AP in the new channel, and if the flag is reset the station disconnects from current BSS. It is the responsibility of the host to initiate the reconnection to a new or old BSS.

Command `wmiconfig --setconnectctrl <ctrl flags bitmask>`

Command Parameters

```
typedef struct{
A_UINT8  networktype;
A_UINT8  dot11authmode;
A_UINT8  authmode;
A_UINT8  pairwiseCryptoType; /*CRYPTO_TYPE*/
A_UINT8  pairwiseCryptoLen;
A_UINT8  groupCryptoType; /*CRYPTO_TYPE*/
A_UINT8  groupCryptoLen;
A_UINT8  ssidLength;
A_UCHAR  ssid[WMI_MAX_SSID_LEN];
A_UINT16 channel;
A_UINT8  bssid[AUTH_MAC_LEN];
A_UINT8  ctrl_flags; /*WMI_CONNECT_CTRL_FLAGS_BITS*/
} WMI_CONNECT_CMD;
ctrl flags bitmask
```

= 0x0001	CONNECT_ASSOC_ POLICY_USER	Assoc frames are sent using the policy specified by the flag
= 0x0002	CONNECT_SEND_ REASSOC	Send Reassoc frame while connecting, otherwise send assoc frames
= 0x0004	CONNECT_IGNORE _WPAx_GROUP _CIPHER	Ignore WPAx group cipher for WPA/WPA2
= 0x0008	CONNECT_ PROFILE_MATCH_ DONE	Ignore any profile check
= 0x0010	CONNECT_IGNORE _AAC_BEACON	Ignore the admission control information in the beacon
= 0x0020	CONNECT_CSA _FOLLOW_BSS	Follow BSS or not following CSA
= 0xFFFF		Reset all control flags

... **CONNECT_CMD**, continued

Command Values

```
typedef enum {
  INFRA_NETWORK    = 0x01,
  ADHOC_NETWORK    = 0x02,
  ADHOC_CREATOR    = 0x04,
} NETWORK_TYPE;
typedef enum {
  OPEN_AUTH      = 0x01,
  SHARED_AUTH     = 0x02,
  LEAP_AUTH      = 0x04,
} DOT11_AUTH_MODE;
typedef enum {
  NONE_AUTH      = 0x01,
  WPA_AUTH      = 0x02,
  WPA_PSK_AUTH    = 0x03,
  WPA2_AUTH      = 0x04,
  WPA2_PSK_AUTH    = 0x05,
  WPA_AUTH_CCKM    = 0x06,
  WPA2_AUTH_CCKM   = 0x07,
} AUTH_MODE;
typedef enum {
  NONE_CRYPT     = 0x01,
  WEP_CRYPT      = 0x02,
  TKIP_CRYPT     = 0x03,
  AES_CRYPT      = 0x04,
} CRYPTO_TYPE;
typedef enum {
  CONNECT_ASSOC_POLICY_USER = 0x0001,
  CONNECT_SEND_REASSOC = 0x0002,
  CONNECT_IGNORE_WPAx_GROUP_CIPHER = 0x0004,
  CONNECT_PROFILE_MATCH_DONE = 0x0008,
  CONNECT_IGNORE_AAC_BEACON = 0x0010,
    CONNECT_CSA_FOLLOW_BSS = 0x0020,
} WMI_CONNECT_CTRL_FLAGS_BITS;
```

pairwiseCryptoLen and groupCryptoLen are valid when the respective CryptoTypesis WEP_CRYPT, otherwise this value should be 0. This is the length in bytes.

To reset all the connect control flags, use 0xFFFF.

Reset Value None defined

Restrictions None

Name CREATE_PSTREAM

Synopsis The host uses this command to create a new prioritized data endpoint between the host and the AR6000 device that carries a prioritized stream of data. If the AP that the device connects to requires TSPEC stream establishment, the device requests the corresponding TSPEC with the AP. The maximum and minimum service interval ranges from 0 – 0x7FFFFFFF (ms), where 0 = disabled. The device does not send a reply event for this command, as it is always assumed the command has succeeded. An AP admission control response comes to the host via a WMI_CAC_INDICATION event, once the response for the ADDTS frame comes.

Examples of cases where reassociation is generated (when WMM) and cases where ADDTS is generated (when WMM and enabling ACM) are when:

- Changing UAPSD flags in WMM mode, reassociation is generated
- Changing the interval of sending auto QoS Null frame in WMM mode; reassociation is not generated
- Issuing a command with same previous parameters in WMM mode and enabling ACM, an ADDTS request is generated
- Changing the interval of a QoS null frame sending in WMM mode and enabling ACM, an ADDTS request is generated
- Issuing the command in disconnected state, reassociation or ADDTS is not generated but the parameters are available after (re)association

Command **`--createqos <user priority> <direction> <traffic class> <trafficType> <voice PS capability> <min service interval> <max service interval> <inactivity interval> <suspension interval> <service start time> <tsid> <nominal MSDU> <max MSDU> <min data rate> <mean data rate> <peak data rate> <max burst size> <delay bound> <min phy rate> <sba> <medium time>`** where:

`<user priority>`	802.1D user priority range (0–7)	
`<direction>`	`= 0`	Tx (uplink) traffic
	`= 1`	Rx (downlink) traffic
	`= 2`	Bi-directional traffic
`<traffic class>`	`= 1`	BK
	`= 2`	VI
	`= 3`	VO
`<trafficType>`	`= 0`	Aperiodic
	`= 1`	Periodic
`<voice PS capability>`	Specifies whether the voice power save mechanism (APSD if AP supports it or legacy/simulated APSD [using PS-Poll]) should be used	
	`= 0`	Disable voice power save for traffic class
	`= 1`	Enable APSD voice power save for traffic class
	`= 2`	Enable voice power save for all traffic classes
`<min service interval>`	(In ms)	
`<max service interval>`	Inactivity interval (in ms) (0 = Infinite)	
`<suspension interval>`	(In ms)	
`<service start time>`	Service start time	
`<tsid>`	TSID range (0–15)	
`<nominal MSDU>`	Nominal MAC SDU size	
`<max MSDU>`	Maximum MAC SDU size	
`<min data rate>`	Minimum data rate (in bps)	
`<mean data rate>`	Mean data rate (in bps)	
`<peak data rate>`	Peak data rate (in bps)	
`<max burst size>`	Maximum burst size (in bps)	
`<delay bound>`	Delay bound	
`<min phy rate>`	Minimum PHY rate (in bps)	
`<sba>`	Surplus bandwidth allowance	
`<medium time>`	Medium time in TU of 32-μs periods per sec	

... CREATE_PSTREAM (continued)

Command Parameters

`A_UINT8`	`trafficClass`	TRAFFIC_CLASS value
`A_UINT8`	`traffic Direction`	DIR_TYPE value
`A_UINT8`	`trafficType`	TRAFFIC_TYPE value
`A_UINT8`	`voicePS Capability`	VOICEPS_CAP_TYPE value
`A_UINT8`	`tsid`	Traffic stream ID
`A_UINT8`	`userPriority`	802.1D user priority
`A_UINT16`	`nominalMSDU`	Nominal MSDU in octets
`A_UINT16`	`maxMSDU`	Maximum MSDU in octets
`A_UINT32`	`minServiceInt`	Minimum service interval: the min. period of traffic specified (in ms)
`A_UINT32`	`maxServiceInt`	Maximum service interval: the max. period of traffic specified (in ms)
`A_UINT32`	`inactivityInt`	Indicates how many ms an established stream is inactive before the prioritized data endpoint is taken down and the corresponding T-SPEC deleted
`A_UINT32`	`suspensionInt`	Suspension interval (in ms)
`A_UINT32`	`service StartTime`	Service start time
`A_UINT32`	`minDataRate`	Minimum data rate (in bps)
`A_UINT32`	`meanDataRate`	Mean data rate (in bps)
`A_UINT32`	`peakDataRate`	Peak data rate (in bps)
`A_UINT32`	`maxBurstSize`	
`A_UINT32`	`delayBound`	
`A_UINT32`	`minPhyRate`	Minimum PHY rate for TSPEC (in bps)
`A_UINT32`	`sba`	Surplus bandwidth allowance
`A_UINT32`	`mediumTime`	Medium TSPEC time (in units of 32 µs)

Command Values

`{`		
`WMM_AC_BE`	`= 0`	Best Effort
`WMM_AC_BK`	`= 1`	Background
`WMM_AC_VI`	`= 2`	Video
`WMM_AC_VO`	`= 3`	Voice
`All other values reserved`		
`} TRAFFIC_CLASS`		
`{`		
`UPLINK_TRAFFIC`	`= 0`	From the AR6000 device to the AP
`DOWNLINK_TRAFFIC`	`= 1`	From the AP to the AR6000 device
`BIDIR_TRAFFIC`	`= 2`	Bi-directional traffic
`All other values reserved`		
`} DIR_TYPE`		
`{`		
`DISABLE_FOR_THIS_AC`	`= 0`	
`ENABLE_FOR_THIS_AC`	`= 1`	
`ENABLE_FOR_ALL_AC`	`= 2`	
`All other values reserved`		
`} VOICEPS_CAP_TYPE`		

Reset Value No pstream is present after reset; each of the BE, BK, VI,VO pstreams must be created (either implicitly by data flow or explicitly by user)

Restrictions At most four prioritized data endpoints can be created, one for each AC.

See Also "DELETE_PSTREAM" on page A-15

Name	**DELETE_BAD_AP**		
Synopsis	The host uses this command to clear a particular entry in the bad AP table		
Command	`wmiconfig eth1 --clrAP [--num=<index>] // used to clear a badAP entry. num is index from 0-1`		
Command Parameters	`A_UINT8`	`badApIndex`	Index [0...*n*] that identifies the entry in the bad AP table to delete
Command Values	`badApIndex = 0, 1`		Entry in the bad AP table
Reset Value	None defined		
Restrictions	None		
See Also	"ADD_BAD_AP" on page A-8		

Name	**DELETE_CIPHER_KEY**		
Synopsis	The host uses this command to delete a key that was previously added with the **"ADD_CIPHER_KEY"** command.		
Command	TBD		
Command Parameters	`A_UINT8`	`keyIndex`	Index (0...3) of the key to be deleted
Command Values	`keyIndex`	`= 0, 1,2, 3`	Key to delete
Reset Value	None		
Restrictions	The host should not delete a key that is currently in use by the AR6000.		
See Also	"ADD_CIPHER_KEY" on page A-9		

Name	**DELETE_PSTREAM**		
Synopsis	The host uses this command to delete a prioritized data endpoint created by a previous **"CREATE_PSTREAM"** command		
Command	**`--deleteqos <trafficClass> <tsid>`**, where:		
	`<traffic class>`	`= 0`	BE
		`= 1`	BK
		`= 2`	VI
		`= 3`	VO
	`<tsid>`		The TSpec ID; use the -qosqueue option to get the active TSpec IDs for each traffic class
Command Parameters	`A_UINT8`	`trafficClass`	Indicate the traffic class of the stream being deleted
	`A_UINT8`	`tsid`	Indicates the TSPEC ID within the traffic class to delete
Command Values	`{`		
	`WMM_AC_BE`	`= 0`	Best effort
	`WMM_AC_BK`	`= 1`	Background
	`WMM_AC_VI`	`= 2`	Video
	`WMM_AC_VO`	`= 3`	Voice
	`} TRAFFIC CLASS` `0-15 for TSID`		
Reply Values	N/A		
Restrictions	This command should only be issued after a **"CREATE_PSTREAM"** command has successfully created a prioritized stream		
See Also	"CREATE_PSTREAM" on page A-13		

Name	**DELETE_WOW_PATTERN**
Synopsis	The host uses this command to remove a pre-specified pattern from the WoW pattern list.
Command	**`wmiconfig -delwowpattern <list-id> <pattern-id>`**
Command Parameters	`A_UINT8` `filter_list_id` ID of the list that contains the WoW filter pattern to delete `A_UINT8` `filter_id` ID of the WoW filter pattern to delete
Reply Parameters	None
Reset Value	None defined
Restrictions	None
See Also	"ADD_WOW_PATTERN" on page A-10

Name	**DISCONNECT**
Synopsis	The host uses this command to disconnect from the currently associated AP or to abort the current connection process. If the host has issued a **"CONNECT_CMD"** command, it must issue the disconnect before it issues the next connect command.
Command Parameters	None
Reply Parameters	Disconnect event with reason set to DISCONNECT_CMD
Reset Value	None
Restrictions	None
See Also	"DISCONNECT" on page A-57

Name	**EXTENSION**
Synopsis	The WMI message interface is used mostly for wireless control messages to a wireless module applicable to wireless module management regardless of the target platform implementation. However, some commands only peripherally related to wireless management are desired during operation. These wireless extension commands may be platform-specific or implementation-dependent.
Command	N/A
Command Parameters	Command-specific
Command Values	Command-specific
Reply Parameters	Command-specific
Reset Values	None defined
Restrictions	None defined

Name	**GET_BIT_RATE**		
Synopsis	Used by the host to obtain the rate most recently used by the AR6000 device		
Command	**`wmiconfig eth1 --getfixrates`**		
Command Parameters	None		
Reply Parameters	`A_INT8`	`rateIndex`	See the **"SET_BIT_RATE"** command
Reset Values	None		
Restrictions	This command should only be used during development/debug; it is not intended for use in production. It is only valid when the device is in the CONNECTED state		
See Also	"SET_BIT_RATE" on page A-25		

Name	**GET_CHANNEL_LIST**		
Synopsis	Used by the host uses to retrieve the list of channels that can be used by the device while in the current wireless mode and in the current regulatory domain.		
Command	TBD		
Command Parameters	None		
Reply Parameters	`A_UINT8`	`reserved`	Reserved
	`A_UINT8`	`numberOfChannels`	Number of channels the reply contains
	`A_UINT16`	`channelList [numberOfChannels]`	Array of channel frequencies (in MHz)
Reset Values	`None defined`		
Restrictions	The maximum number of channels that can be reported are 32		

Name	**GET_FIXRATES**		
Synopsis	Clears the current calculated RSSI and SNR value. RSSI and SNR are reported by running-average value. This command will clear the history and have a fresh start for the running-average mechanism.		
Synopsis	This returns rate-mask set via WMI_SET_FIXRATES to retrieve the current fixed rate that the AR6000 is using. See **"SET_FIXRATES"**.		
Command	**`wmiconfig eth1 --getfixrates`**		
Command Parameters	`A_UINT16`	`fixRateMask;`	Note: if this command is used prior to using WMI_SET_FIXRATES, AR6000 returns 0xffff as fixRateMask, indicating all the rates are enabled
Reply Parameters	None		
Reset Value	None defined		
Restrictions	None		
See Also	"SET_FIXRATES" on page A-31		

Name	**GET_KEEPALIVE**
Synopsis	The host uses this command to get the configured keepalive interval. If no Tx or Rx activity occurs for the duration of the keepalive interval, the STA must send a NULL data packet to the AP it is connected to.
Command	**`wmiconfig -getkeepalive <keepalive-interval>`**
Command Parameters	`A_BOOL` `configured` Shows whether a a keepAliveInterval is configured
	`A_UINT8` `keepAliveInterval` keepAliveInterval (in ms)
Reply Parameters	None
Reset Values	None defined
Restrictions	CCX must be enabled
See Also	"SET_KEEPALIVE" on page A-34

Name	**GET_PMKID_LIST_CMD**
Synopsis	Retrieves the list of PMKIDs on the firmware. The **WMI_GET_PMKID_LIST_EVENT** is generated by the firmware.
Command	TBD
Command Parameters	
Reset Values	None
Restrictions	None
See Also	■ "SET_PMKID_LIST_CMD" on page A-36
	■ "GET_PMKID_LIST_EVENT" on page A-58

Name	**GET_ROAM_TBL**
Synopsis	Retrieve the roaming table maintained on the target. The response is reported asynchronously through the **"ROAM_TBL_EVENT"**.
Command	**`wmiconfig --getroamtable <roamctrl> <info>`**
Command Parameters	`A_UINT8` `roamCtrlType;`
	`A_UINT16` `roamMode`
	`A_UINT16` `numEntries`
	`WMI_BSS_ROAM_INFO` `bssRoamInfo[1]`
Reply Value	Reported asynchronously through the **"ROAM_TBL_EVENT"**
Reset Value	None defined
Restrictions	None
See Also	"SET_KEEPALIVE" on page A-34

Name	**GET_TARGET_STATS**
Synopsis	The host uses this command to request that the target send the statistics that it maintains. The statistics obtained from the target are accrued in the host every time the GET_TARGET_STATS command is issued. The --clearStats option is added to clear the target statistics maintained in the host.
Command	**`wmiconfig --getTargetStats --clearStats`**
Command Parameters	`TARGET_STATS` `targetStats` WMI_TARGET_STATS `A_UINT8` `clearStats`
Reply Value	RSSI return value (0–100)
Reset Values	All statistics are cleared (zeroed)
Restrictions	The --getTargetStats option must be used; the --clearStats option is also available also

Name	**GET_TX_PWR**
Synopsis	The host uses this command to retrieve the current Tx power level
Command	**`wmiconfig -i eth1 --getpower`**
Command Parameters	`None`
Reply Parameters	`A_UINT16` `dbM` The current Tx power level specified in dbM
Reset Values	The maximum permitted by the regulatory domain
Restrictions	None
See Also	"SET_TX_PWR" on page A-46

Name	**GET_WOW_LIST**		
Synopsis	The host uses this command to retrieve the current list of WoW patterns.		
Command	**`wmiconfig –getwowlist <list-id>`**		
Command Parameters	`A_UINT8`	`filter_list_id`	ID of the list of WoW patterns to retrieve
Reply Value(s)	`A_UINT16`	`num_filters`	Number of WoW patterns contained in the list
	`A_UINT8`	`wow_mode`	Current mode of WoW (enabled or disabled)
	`A_UINT8`	`host_mode`	Current host mode (asleep or awake)
	`WOW_FILTER`	`wow_filters[1]`	Contents of the WoW filter pattern list (contains mask, pattern, offset and size information for each of the patterns)
Reset Value	None defined		
Restrictions	None		
See Also	"SET_WSC_STATUS" on page A-49		

Name	**HOST_EXIT_NOTIFY**
Synopsis	The host uses this command to inform the AR6000 that host is unloading the driver and is therefore unavailable.
Command	TBD
Command Parameters	None
Command Values	None
Reset Value	None
Restrictions	None

Name	**LQ_THRESHOLD_PARAMS**		
Synopsis	Sets Link Quality thresholds, the sampling will happen at every unicast data frame Tx if a certain threshold is met, and the corresponding event will be sent to the host.		
Command	`--lqThreshold <enable> <upper_threshold_1> ... <upper_threshold_4> <lower_threshold_1> ... <lower_threshold_4>`		
Command Parameters	`<enable>`	= 0	Disable link quality sampling
		= 1	Enable link quality sampling
	`<upper_threshold_x>`		Above thresholds (value in [0,100]), in ascending order
	`<lower_threshold_x>`		Below thresholds (value in [0,100]), in ascending order
Command Values	See command parameters		
Reset Value	None defined		
Restrictions	None		

Name **OPT_TX_FRAME**

Synopsis Special feature, sends a special frame.

Command **`wmiconfig --sendframe <frmType> <dstaddr> <bssid> <optIEDatalen> <optIEData>`**

Command Parameters

```
{
A_UINT16        optIEDataLen;
A_UINT8         frmType;
A_UINT8         dstAddr[ATH_MAC_LEN];
A_UINT8         bssid[ATH_MAC_LEN];
A_UINT8         optIEData[1];
} WMI_OPT_TX_FRAME_CMD;
```

Command Values	`<frmtype>`	= 1	Probe request frame
		= 2	Probe response frame
		= 3	CPPP start
		= 4	CPPP stop
Reset Value	None defined		
Restrictions	Send a special frame only when special mode is on.		

Name	**RECONNECT**
Synopsis	This command requests a reconnection to a BSS to which the AR6000 device was formerly connected
Command	TBD
Command Parameters	`A_UINT16` `channel` Provides a hint as to which channel was used for a previous connection
	`A_UINT8` `bssid[6]` If set, indicates which BSSID to connect to
Command Values	None
Reset Values	None
Restrictions	None
See Also	"CONNECT_CMD" on page A-11

Name	**RSSI_THRESHOLD_PARAMS**
Synopsis	Configures how the AR6000 device monitors and reports signal strength (RSSI) of the connected BSS, which is used as a link quality metric. The four RSSI threshold sets (in dbM) of the host specification divide the signal strength range into six segments. When signal strength increases or decreases across one of the boundaries, an **RSSI_THRESHOLD** event is signaled to the host. The host may then choose to take action (such as influencing roaming).

Command

```
wmiconfig eth1 --rssiThreshold <weight> <pollTime>
<above_threshold_val_1> ... <above_threshold_tag_6>
<above_threshold_val_6>
<below_threshold_tag_1> <below_threshold_val_1> ...
<below_threshold_tag_6> <below_threshold_val_6>
```

Command Parameters		
`A_UINT8`	`weight`	Range in [1, 16] used to calculate average RSSI
`A_UINT32`	`pollTime`	RSSI (signal strength) sampling frequency in seconds (if pollTime = 0, single strength sampling is disabled)
`USER_RSS__THOLD`	`tholds[12]`	Thresholds (6 x 2)

Command Values	None defined
Reset Values	pollTime is 0, and sampling is disabled
Restrictions	Can only be issued if the AR6000 device is connected

Name	**RX_FRAME_FORMAT**		
Synopsis	Used to communicate to the target device the desired format and behavior for received frames prior to delivering the frame to the host. Specifically this command is used to dictate the following; the meta data version used by the target, the frame header format, whether 802.11 defragmentation should be performed by the device or should be left to the host system.		
Command	**`TBD`**		
Command Parameters	`A_UINT8`	`metaVersion`	(0–7) specifies the version of the meta data header that should appear in received frames. 0 indicates that no meta data should be present.
	`A_UINT8`	`dot11Hdr`	(0–1) a 0 indicates to the device that it should convert the 802.11 header to a 802.3 header whereas 1 indicates that the 802.11 header should be sent to the host.
	`A_UINT8`	`defragOnHost`	(0–1) a 0 indicates that the device will perform 802.11 defragmentation. A 1 indicates that defragmentation will not be performed and any sub-frames will be sent to the host as they are received.
Command Values	metaVersion : 0,1 dot11Hdr : 0,1 defragOnHost : 0,1		

Reset Values	metaVersion = 0
	dot11Hdr = 0
	defragOnHost = 0		

Restrictions	None		

Name	**SCAN_PARAMS**		
Synopsis	The minact parameter determines the minimum active channel dwell time, within which if the STA receives any beacon, it remains on that channel until the maxact channel dwell time. If the STA does not receive a beacon within the minact dwell time, it switches to scan the next channel.		
Command	**`wmiconfig -scan -minact=<ms> --maxact=<ms>`**		
Command Parameters	`A_UINT16`	`maxact`	Channel dwell time (in ms), default = 0
	`A_UINT16`	`minact`	Channel dwell time (in ms), default = 105
Command Values	See channel parameters		
Reset Values	None defined		
Restrictions	The minact value should be greater than 0; maxact should be between 5–65535 ms and greater than minact		

Name	**SET_ACCESS_PARAMS**
Synopsis	Allows the host to set access parameters for the wireless network. A thorough understanding of IEEE 802.11 is required to properly manipulate these parameters.
Command	`wmiconfig eth1 --acparams --txop <limit> --cwmin <0-15> --cwmax <0-15> --aifsn<0-15>`
Command Parameters	
Command Values	None
Reset Values	Reasonable defaults that vary, between endpoints (prioritized streams)
Restrictions	None

Command Parameters:

`A_UINT16`	`txop`	The maximum time (expressed in units of 32 µs) the device can spend transmitting after acquiring the right to transmit
`A_UINT8`	`eCWmin`	Minimum contention window
`A_UINT8`	`eCWmax`	Maximum contention window
`A_UINT8`	`aifsn`	The arbitration inter-frame space number

Name	**SET_ADHOC_BSSID**
Synopsis	Allows the host to set the BSSID for an ad hoc network. If a network with this BSSID is not found, the target creates an ad hoc network with this BSSID after the connect WMI command is triggered (e.g., by the SIOCSIWESSID IOCTL).
Command	`wmiconfig eth1 --adhocbssid <bssid>`
Command Parameters	`A_UINT8` `bssid[ATH_MAC_LEN]` BSSID is specified in xx:xx:xx:xx:xx:xx format
Command Values	None
Reset Values	None
Restrictions	None

Name	**SET_AKMP_PARAMS**
Synopsis	Enables or disables multi PMKID mode.
Command	`wmiconfig eth1 --setakmp --multipmkid=<on/off>`
Command Parameters	see below
Command Values	see below
Reset Values	MultiPMKID mode is disabled
Restrictions	None

Command Parameters:

```
typedef struct {
  A_UINT32  akmpInfo;
} WMI_SET_AKMP_PARAMS_CMD;
```

Command Values:

`akmpInfo;`	bit[0] = 0	MultiPMKID mode is disabled and PMKIDs that were set using the **WMI_SET_PMKID_CMD** are used in the [Re]AssocRequest frame.
	bit[0] = 1	MultiPMKID mode is enabled and PMKIDs issued by the **WMI_SET_PMKID_LIST_CMD** are used in the next [Re]AssocRequest sent to the AP.

Name	**SET_APPIE**	
Synopsis	Add an application-specified IE to a management frame. The maximum length is 76 bytes. Including the length and the element ID, this translates to 78 bytes.	
Command	`wmiconfig --setappie <frame> <IE>`, where:	
	`frame`	One of beacon, probe, respon, assoc
	`IE`	A hex string beginning with DD (if = 0, no IE is sent in the management frame)
Command Parameters	`mgmtFrmType;`	A WMI_MGMT_FRAME_TYPE
	`ieLen;`	Length of the IE to add to the GMT frame
Command Values	None	
Reset Value	None defined	
Restrictions	Supported only for the probe request and association request management frame types. Also, only one IE can be added per management frame type.	

Name	**SET_ASSOC_INFO**		
Synopsis	The host uses this command to specify any information elements (IEs) it wishes the AR6000 device to add to all future association and reassociation requests. IEs must be correct and are used as is by the device. IEs specified through this command are cleared with a DISCONNECT.		
Command	`wmiconfig eth1 --setAssocIe <IE>`		
Command Parameters	`A_UINT8`	`ieType`	Used directly in 802.11 frames
	`A_UINT8`	`bufferSize`	Size of assocInfo (in bytes) ranging from 0–240. If = 0, previously set IEs are cleared.
	`A_UINT8`	`assocInfo[bufferSize]`	Used directly in 802.11 frames
Command Values	None		
Reset Values	IEs are cleared		
Restrictions	This command can only be issued in the DISCONNECTED state		

Name	**SET_AUTHMODE**		
Synopsis	Sets the 802.11 authentication mode of reconnection		
Command	`wmiconfig eth1 --setauthmode <mode>`		
Command Parameters	`A_UINT8`	`mode`	
Command Values	`mode`	`= 0x00`	Proceed with authentication during reconnect
		`= 0x01`	Do not proceed with authentication during reconnect
Reset Values	Authentication		
Restrictions	None		

Name	**SET_BEACON_INT**
Synopsis	Sets the beacon interval for an ad hoc network. Beacon interval selection may have an impact on power savings. To some degree, a longer interval reduces power consumption but also decreases throughput. A thorough understanding of IEEE 802.11 ad hoc networks is required to use this command effectively.
Command	`wmiconfig eth1 --ibssconintv`
Command Parameters	`A_UINT16` `beaconInterval` Specifies the beacon interval in TU units (1024 μs)
Command Values	None
Reset Values	The default beacon interval is 100 TUs (102.4 ms)
Restrictions	This command can only be issued before the AR6000 device starts an ad hoc network
See Also	"SET_IBSS_PM_CAPS" on page A-33

Name	**SET_BIT_RATE**		
Synopsis	The host uses this command to set the AR6000 device to a specific fixed rate.		
Command	`iwconfig eth* rate R`		
Command Parameters	`A_UINT32`	`dataRate`	Data frame Tx rate in Kbps
Command Parameters	`A_UINT32`	`mgmtRate`	Management frame Tx rate in Kbps
	`A_UINT32`	`ctrlRate`	Control frame rate in Kbps
Command Values	See command parameters		
Reset Values	The dynamic rate is determined by the AR6000 device		
Restrictions	This command is intended for use only during development/debug; it is not intended for use in production		
See Also	"GET_BIT_RATE" on page A-17		

Name	**SET_BMISS_TIME**		
Synopsis	This command sets the beacon miss (BMISS) time, which the AR6000 hardware use to recognize missed beacons. When an excessive number (15) of consecutive beacons are missed, the AR6000 consider switching to a different BSS. The time can be specified in number of beacons or in TUs.		

Command(s)	`wmiconfig eth1 --setbmissbeacons=<val>`
	`wmiconfig eth1 --setbmisstime=<val>`		

Command Parameters	`A_UINT16`	`bmissTime`	Specifies the beacon miss time [1000...5000] in TUs (1024 μs)
	`A_UINT16`	`bmissbeacons`	Specifies the number of beacons [5...50]
Command Values	None		
Reset Values	bmissTime is 1500 TUs (1536 ms)		
Restrictions	None		

Name	**SET_BSS_FILTER**
Synopsis	The host uses this to inform the AR6000 device of the types of networks about which it wants to receive information from the "BSSINFO" event. As the device performs either foreground or background scans, it applies the filter and sends "BSSINFO" events only for the networks that pass the filter. If any of the bssFilter or the ieMask filter matches, a BSS Info is sent to the host. The ieMask currently is used as a match for the IEs in the beacons, probe responses and channel switch action management frame. See also "Scan and Roam" on page 3-1. The BSS filter command has been enhanced to support IE based filtering. The IEs can be specified as a bitmask through this command using this enum.
Command	`wmiconfig eth1 -filter = <filter> --ieMask 0x<mask>`
Command Parameters	`A_UINT8` `BssFilter`

Command Values

```
typedef struct {
 A_UINT8              bssFilter;        See WMI_BSS_FILTER
 A_UINT32             ieMask;
} __ATTRIB_PACK WMI_BSS_FILTER_CMD;
The ieMask can take this combination of values:
enum {
BSS_ELEMID_CHANSWITC  = 0x01
H
BSS_ELEMID_ATHEROS    = 0x02,
};
```

Reply Value	None
Reset Value	BssFilter = NONE_BSS_FILTER (0)
Restrictions	None
See Also	"CONNECT_CMD" on page A-11

Name	**SET_BT_PARAMS**
Synopsis	Passes configuration details for different BT_STREAMs as well as to indicate front end RF configuration.
Command	**`wmiconfig -setBTparams <paramType> <params>`**
Command Parameters	`struct {`

`union {`	
`BT_PARAMS_A2DP a2dpParams;`	
`BT_PARAMS_ACLCOEX aclCoexParams;`	
`BT_PARAMS_SCO scoCoexParams;`	
`A_UINT8 antType;`	■ 0 = Disabled (default) ■ 1 = BT_ANT_TYPE_DUAL ■ 2 = BT_ANT_TYPE_SPLITTER ■ 3 = BT_ANT_TYPE_SWITCH
`A_UINT8 coLocatedBtDev`	■ 0 = BT_COLOCATED_DEV_BTS4020 (default) ■ 1 = BT_COLCATED_DEV_CSR ■ 2 = BT_COLOCATED_DEV_VALKYRIE
`}`	
`A_UINT8 paramType;`	■ 1 = BT_PARAM_SCO ■ 1 = BT_PARAM_A2DP ■ 3 = BT_PARAM_ANTENNA_CONFIG ■ 4 = BT_PARAM_COLOCATED_BT_DEVICE ■ 5 = BT_PARAM_ACL_COEX
`}`	
`struct {`	
`A_UINT32 numScoCyclesForceTrigger;`	During coexistence, firmware queues PS-POLL to retrieve downlink data. If AP responds with Null frame, firmware will skip numScoCyclesForceTrigger number of SCO frames. (firmware Default = 10)
`A_UINT32 dataResponseTimeout;`	Firmware dynamically determines the amount of time to wait for downlink/stomp on Bluetooth, based on previously received data rates. The Firmware uses this value as the maximum duration (Default = 10 ms).
`A_UINT32 stompScoRules;`	This software release does not provide configuration for stomping BT. Firmware always stomps all Bluetooth traffic while waiting for downlink packets.
`A_UINT32 scoOptFlags;`	SCO optimization flags

Bits	Purpose
0	Allow close range optimization; set this to one if dynamic optimization is needed. If this bit is clear, no other scoOptFlags bit will have any effect.
1	Force the chip to always awake during optimization mode; can be used to tradeoff between power save and latency.
2	If set, firmware will use the host supplied thresholds for packet to low rate packets ratio (p2lrp). If set, ensure that p2lrpOptModeBound and p2lrpNonOptModeBound are supplied with valid values.
3:31	Unused

`A_UINT32 p2lrpOptModeBound`	In OPT mode, minimum ratio of packets to low rate packets required to continue in OPT mode

SET_BT_PARAMS, continued

`A_UINT32 p2lrpNonOptMode Bound`	In non OPT mode, minimum ratio of packets to low rate packets required to switch to OPT mode
`A_UINT8 stompDutyCyleVal;`	If SCO traffic is stomped while the AR6003 is waiting for downlink traffic, firmware will not queue trigger frames for the configured number of SCO cycles. stompDutyCycleVal represents the baseline number of configured frames. Recommended Default Value = 2.
`A_UINT8 stompDutyCyleMax Val;`	See stompDutyCyleVal. Under bad air conditions, the firmware will gradually increase the configured number of SCO cycles for which trigger frames are not queued to stompDutyCyleMaxVal. In addition, as air conditions improves, the configured number is gradually returned to the baseline. Recommended Value = 8.
`A_UINT8 psPollLatency Fraction;`	Under good WLAN signal strength it is possible to queue multiple PS-polls to retrieve downlink data. The fraction dictates which piece of the idle duration firmware can queue additional PS-polls. ■ 1 = BT_PARAM_SCO_PSPOLL_LATENCY_ONE_FOURTH ■ 2 = BT_PARAM_SCO_PSPOLL_LATENCY_HALF ■ 3 = BT_PARAM_SCO_PSPOLL_LATENCY_THREE_FOURTH
`A_UINT8 noSCOSlots;`	
`A_UINT8 noIdleSlots;`	
`A_UINT8 reserved`	Needed to maintain buffer alignment

```
} POSTPACK BT_PARAMS_SCO;
typedef PREPACK
struct {
```

`A_UINT32 a2dpWlanUsageLimit ;`	Max WLAN time used by the firmware to identify the WLAN medium
`A_UINT32 a2dpBurstCntMin`	Minimum number of BT data frames to replenish WLAN usage time
`A_UINT32 dataResponse Timeout;`	Firmware dynamically determines the amount of time to wait for downlink/stomp on Bluetooth, based on previously received data rates. The Firmware uses this value as the maximum duration (Default = 10 ms).
`A_UINT32 a2dpOptFlags;`	2DP Optimization Flags

Bits	Purpose
0	Allow close-range optimization; set this to one if dynamic optimization is needed. If this bit is clear, no other a2dpOptFlags bit will have any effect.
1	Force chip awake during optimization mode; can be used to tradeoff between power save and latency.
2	If set firmware uses host-supplied thresholds for packet to low rate packets ratio (p2lrp) for dynamic optimization. If set, ensure that p2lrpOptModeBound and p2lrpNonOptModeBound have valid values.
3:31	Unused

`A_UINT32 p2lrpOptModeBound`	In OPT mode, minimum ratio of packets to low rate packets required to continue in OPT mode
`A_UINT32 p2lrpNonOptModeBound`	In non OPT mode, minimum ratio of packets to low rate packets required to switch to OPT mode

SET_BT_PARAMS, continued

`A_UINT16 reserved16`	
`A_UINT8 isCoLocatedBtRoleMaster;`	
`A_UINT8 reserved8;`	
`}POSTPACK BT_PARAMS_A2DP`	
`struct {`	
`A_UINT32 aclWlanMediumUsageTime;`	WLAN usage time during ACL coexistence (non-A2DP)
`A_UINT32 aclBtMediumUsageTime;`	BT usage time during ACL coexistence
`A_UINT32 aclDataRespTimeout;`	Firmware dynamically determines the amount of time to wait for downlink/stomp on Bluetooth, based on previously received data rates. The Firmware uses this value as the maximum duration (Default = 10 ms).
`}BT_PARAMS_ACL_COEX`	

Command Values None defined

Reset Value None

Restrictions None

Name **SET_BT_STATUS**

Synopsis Indicates the status of collocated Bluetooth device.

Command **`wmiconfig -setBTstatus <streamType> <status>`**

Command Parameters `struct {`

`A_UINT8 streamType;`	
`BT_STREAM_SCO`	Whenever user starts/stops SCO traffic on collocated Bluetooth device, firmware is indicated of the Bluetooth state via this command.
`BT_PARAMS_A2DP`	
`BT_STREAM_A2DP`	Applicable only for front end RF switch. When a collocated Bluetooth device initiates a SCAN/INQUIRY, firmware must be indicated for the scan to go through at the cost WLAN traffic.
`BT_STREAM_ESCO`	Firmware currently treats SCO and ESCO similarly.
`A_UINT8 streamType;`	
`BT_STATUS_START`	
`BT_STATUS_STOP`	
`BT_STATUS_RESUME`	
`BT_STATUS_SUSPEND`	
`} info;`	

```
A_UINT8 paramType;
} WMI_SET_BT_PARAMS_CMD;
```

Reset Value None defined

Restrictions None

Name SET_CHANNEL_PARAMETERS

Synopsis Configures various WLAN parameters related to channels, sets the wireless mode, and can restrict the AR6000 device to a subset of available channels. The list of available channels varies depending on the wireless mode and the regulatory domain. The device never operates on a channel outside of its regulatory domain. The device starts to scan the list of channels right after this command.

Command `wmiconfig eth1 --wmode <mode> <list>`

Command Parameters

`A_UINT8`	`scanParam`	Set whether to enable auto scan
`A_UINT8`	`phyMode`	See the command values
`A_UINT8`	`numberOfChannels`	Number of channels in the channel array that follows. If = 0, then the device uses all of the channels permitted by the regulatory domain and by the specified phyMode.
`A_UINT16`	`channel [numberOfChannels ]`	Array listing the subset of channels (expressed as frequencies in MHz) the host wants the device to use. Any channel not permitted by the specified phyMode or by the specified regulatory domain is ignored by the device.

Command Values

`scanparam = 0`		Do not scan after executing this command
`scanparam = 1`		Autoscan after executing this command
`phyMode = {`		Wireless mode
`11a`	`= 0x01`	
`11g`	`= 0x02`	
`11ag`	`= 0x03`	
`11b`	`= 0x04`	
`11g only`	`= 0x05`	
`}`		

Reset Values

`phyMode`	`11ag`	802.11a/g modules
	`11g`	802.11g module
`channels`		Defaults to all channels permitted by the current regulatory domain.

Restrictions This command, if issued, should be issued soon after reset and prior to the first connection. This command should only be issued in the DISCONNECTED state.

Name	**SET_DISC_TIMEOUT**
Synopsis	The host uses this command to configure the amount of time that the AR6000 should spend when it attempts to reestablish a connection after losing link with its current BSS. If this time limit is exceeded, the AR6000 send a **“DISCONNECT”** event. After sending the **“DISCONNECT”** event the AR6000 continues to attempt to reestablish a connection, but they do so at the interval corresponding to a foreground scan as established by the **“SET_SCAN_PARAMS”** command. A timeout value of 0 indicates that the AR6000 will disable all autonomous roaming, so that the AR6000 will not perform any scans after sending a **“DISCONNECT”** event to the host. The state is maintained until a shutdown or host sets different timeout value from 0. When host issues a CONNECT command to connect a specific AP, if AR6000 fails to connect with the AP and this time limit is exceeded, AR6000 also send a DISCONNECT event with reason NO_NETWORK_AVAIL.
Command	`wmiconfig eth1 --disc=<timeout in seconds>`
Command Parameters	`A_UINT8` `disconnectTimeout` Specifies the time limit (in seconds) after which a failure to reestablish a connection results in a **“DISCONNECT”** event
Command Values	None
Reset Values	disconnectTimeout is 10 seconds
Restrictions	This command can only be issued while in a DISCONNECTED state

Name	**SET_FIXRATES**
Synopsis	By default, the AR6000 device uses all PHY rates based on mode of operation. If the host application requires the device to use subset of supported rates, it can set those rates with this command. In 802.11g mode, the AR6000 device takes the entire 802.11g basic rate set and the rates specified with this command and uses it as the supported rate set. This rate set is advertised in the probe request and the assoc/re-assoc request as supported rates. Upon successful association, the device modifies the rate set pool using the: intersection of AP-supported rates with the union of the 802.11g basic rate set and rates set using this command. The device picks transmission rates from this pool based on a rate control algorithm. The rate setting through this command is independent of current WLAN mode (a or b/g) of operation.
Command	`wmiconfig eth1 --setfixrates <rate_0>... <rate_n>`

Command Parameters

```
{
RATE_1M       = 0
RATE_2M       = 1
RATE_5.5M     = 2
RATE_11M      = 3
RATE_6M       = 4
RATE_9M       = 5
RATE_12M      = 6
RATE_18M      = 7
RATE_24M      = 8
RATE_36M      = 9
RATE_48M      = 10
RATE_54Mb     = 11
} WMI_BIT_RATE
```

Command Values	None
Reset Value	None defined
Restrictions	None
See Also	“GET_FIXRATES” on page A-17

Name	**SET_FRAMERATES**
Synopsis	The host uses this command to disable or enable the Tx rate of a specific type frame. It forms a corresponding relation between the rateMask bit and Tx rate table index. For example, setting bit 0 of rateMask to 1 means enable first entry of rate table.
Command	**`wmiconfig --setFrameRate bEnable frameType framSubtype rateMask`**

Command Parameters	`A_UINT8`	`bEnable`	Set to 1 to enable the rate mask
	`A_UINT8`	`type`	Specific frame type
	`A_UINT8`	`subType`	Specific frame subtype
	`A_UINT16`	`rateMask`	Rate mask for the specific rate Set the corresponding bit to 0 to disable the rate, bit [0] to set 1 M rate, bit [1] to set 2 M rate, etc. The rate table sequence is: 1 M, 2 M, 5.5 M, 11 M, 6 M, 9 M, 12 M, 18 M, 24 M, 36 M, 48 M, 54 M.

Command Values	None
Reset Value	None
Restrictions	None

Name	**SET_HB_TIMEOUT**
Synopsis	The host periodically sends a heartbeat to the AR6000; if the AR6000 does not receive this heartbeat from host for a set interval, it thinks host is temporarily unavailable. If the AR6000 is in APSD mode, it stops sending trigger frames after exceeding this interval. When the AR6000 receives the heartbeat again, it resumes sending trigger frames.
Command	TBD

Command Parameters	`A_UINT32`	`timeout`	Maximum duration for the target to not receive the heartbeat from the host

Command Values	None
Reset Value	None
Restrictions	None

Name	**SET_HOST_SLEEP_MODE**
Synopsis	The host uses this command to set the host mode to asleep or awake. All packets are delivered to the host when the host mode is awake. When host mode is asleep, only if WoW is enabled and the incoming packet matches one of the specified WoW patterns, will the packet be delivered to the host. The host will also be woken up by the target for pattern-matching packets and important events.
Command	**`wmiconfig -sethostmode=<asleep/awake>`**

Command Parameters	`A_BOOL`	`awake`	Set the host mode to awake
	`A_BOOL`	`asleep`	Set the host mode to asleep

Command Values	`1 = awake, 0 = asleep`
Reset Value	None defined (default host mode is awake)
Restrictions	None

Name SET_IBSS_PM_CAPS

Synopsis Used to support a non-standard power management scheme for an ad hoc wireless network consisting of up to eight stations (STAs) that support this form of power saving (e.g., Atheros-based STAs). A thorough understanding of IEEE 802.11 ad hoc networks is required to use this command effectively.

Command

```
wmiconfig eth1 --ibsspmcaps --ps=<enable/disable>
--aw=<ATIM Windows in ms>
--ttl=<Time to live in number of beacon periods>
--to=<timeout in ms>
```

Command Parameters

A_UINT8	power_saving	= 0	The non-standard power saving scheme is disabled and maximum throughput (with no power saving) is obtained.
		= 1	Ad hoc power saving scheme is enabled (but throughput may be decreased)
A_UINT16	atim_windows		Specifies the length (in ms) of the ad hoc traffic indication message (ATIM) windows used in an ad hoc network. All Atheros-based STAs that join the network use this duration ATIM window. The duration is communicated between wireless STAs through an IE in beacons and probe responses. The host sets atim_windows to control trade-offs between power use and throughput. The value chosen should be based on the beacon interval (see the **"SET_BEACON_INT"** command) on the expected number of STAs in the IBSS, and on the amount of traffic and traffic patterns between STAs.
A_UINT16	timeout_valu e		Specifies the timeout (in ms). The value is the same for all ad hoc connections, but tracks separately for each. Applicable only for a beacon period and used to derive actual timeout values on the Tx and Rx sides. On the Tx side, the value defines a window during which the STA accepts the frame(s) from the host for a particular connection. Until closed, the window restarts with every frame received from the host. On the Rx side, indicates the time until which the STA continues accepting frames from a particular connection. The value resets with every frame received. The value can be used to determine the trade off between throughput and power. Default = 10 ms
A_UINT8	ttl		Specifies the value in number of beacon periods. The value is used to set a limit on the time until which a frame is kept alive in the AR6000 before being discarded. Default = 5

Command Values None

Reset Values By default, power_saving is enabled with atim_window = 20 ms

Restrictions Can only be issued before the AR6000 starts an ad hoc network

See Also "SET_BEACON_INT" on page A-25

Name	**SET_IP_CMD**
Synopsis	Registers the IP address of the interface with AR6000 software.
Command	**`wmiconfig -i <interface> -ip x x x x`**

Command Parameters	`#define MAX_ARPOFFLOAD_IP_ADDRS 2`
	`typedef PREPACK struct {`
	`A_UINT32 ips[MAX_ARPOFFLOAD_IP_ADDRS];` The IP parameter value should be in network byte order
	`} POSTPACK WMI_SET_IP_CMD;`

Command Values	None
Reset Value	None defined
Restrictions	None

Name	**SET_KEEPALIVE**
Synopsis	The host uses this command to set a keepalive interval. If there is no transmission or reception activity for the duration of the keepalive interval, the STA must send a NULL data packet to the AP it is connected to.
Command	**`wmiconfig -setkeepalive <keepalive-interval>`**
Command Parameters	`A_UINT8` `keepAliveInterval` keepAliveInterval (in ms)
Command Values	None defined (just above user-defined keepAliveInterval)
Reset Value	None defined
Restrictions	CCX must be enabled
See Also	"GET_KEEPALIVE" on page A-18

Name	**SET_LISTEN_INT**		
Synopsis	The host uses this command to request a listen interval, which determines how often the AR6000 device should wake up and listen for traffic. The listen interval can be set by the TUs or by the number of beacons. The device may not be able to comply with the request (e.g., if the beacon interval is greater than the requested listen interval, the device sets the listen interval to the beacon interval). The actual listen interval used by the device is available in the "CONNECT" event.		
Command	**`wmiconfig eth1 --listen=<#of TUs, can range from 15 to 3000> --listenbeacons=<#of beacons, can range from 1 to 50>`**		
Command Parameters	`A_UINT16`	`listenInterval`	Specifies the listen interval in Kμs (1024 μs), ranging from 100 to 1000
	`A_UINT16`	`listenbeacons`	Specifies the listen interval in beacons, ranging from 1 to 50
Command Values	None		
Reset Values	The device sets the listen interval equal to the beacon interval of the AP it associates to.		
Restrictions	None		

Name	**SET_LPREAMBLE**		
Synopsis	Overrides the short preamble capability of the AR6000 device		
Command	TBD		
Command Parameters	`WMI_LPREAMBLE_DISABLED`		The device is short-preamble capable
	`WMI_LPREAMBLE_ENABLED`		The device supports only the long-preamble mode
Command Values	None		
Reset Value	None defined		
Restrictions	None		

Name	**SET_MAX_SP_LEN**		
Synopsis	Set the maximum service period; indicates the number of packets the AR6000 can receive from the AP when triggered		
Command	**`wmiconfig eth1 --setMaxSPLength <maxSPLen>`**		
Command Parameters	`A_UINT8`	`maxSPLen`	An APSD_SP_LEN_TYPE value
Command Values	see below		

```
{
DELIVER_ALL_PKT     = 0x0
DELIVER_2_PKT       = 0x1
DELIVER_4_PKT       = 0x2
DELIVER_6_PKT       = 0x3
}
APSD_SP_LEN_TYPE
```

Reset Values maxSPLen is DELIVER_ALL_PKT

Restrictions None

Name	**SET_OPT_MODE**		
Synopsis	Special feature, sets the special mode on/off		
Command	**`wmiconfig eth1 --mode <mode>`** Set the optional mode, where mode is special or off		
Command Parameters	see below		

```
enum {
      SPECIAL_OFF
      SPECIAL_ON
} OPT_MODE_TYPE;
```

Command Values	`OPT_MODE_TYPE`	`= 0`	SPECIAL_OFF
		`= 1`	SPECIAL_ON
Reset Value	Mode = Off		
Restrictions	None		

Name	**SET_PMKID**		
Synopsis	The host uses this command to enable or disable a pairwise master key ID (PMKID) in the AR6000 PMKID cache. The AR6000 clears its PMKID cache on receipt of a **DISCONNECT** command from the host. Individual entries in the cache might be deleted as the AR6000 detect new APs and decides to remove old ones.		
Command	**`wmiconfig eth1 --setbsspmkid --bssid=<aabbccddeeff> --bsspmkid=<pmkid>`**		
Command Parameters	`A_UINT8`	`bssid[6]`	The MAC address of the AP that the PMKID corresponds to (6 bytes in hex format)
	`A_UINT8`	`enable`	Either PMKID_DISABLE (0) to disable the PMKID or PMKID_ENABLE (1) to enable it (16 bytes in hex format)
	`A_UINT8`	`pmkid[16]`	Meaningful only if enable is PMKID_ENABLE, when it is the PMKID that the AR6000 should use on the next reassociation with the specified AP
Command Values	enable	= 0 (disable), 1 (enable)	`PKMID enabled/disabled`
Reset Values	None defined		
Restrictions	Only supported in infrastructure networks		

Name **SET_PMKID_LIST_CMD**

Synopsis Configures the list of PMKIDs on the firmware.

Command

```
wmiconfig --setpmkidlist --numpmkid=<n> --pmkid=<pmkid_1>
... --pmkid=<pmkid_n>
```

Where *n* is the number of pmkids (maximum = 8) and *pmkid_i* is the *i*th pmkid (16 bytes in hex format)

Command Parameters

```
{
A_UINT8       pmkid[WMI_PMKID_LEN];
} __ATTRIB_PACK WMI_PMKID;
{
A_UINT32      numPMKID;
WMI_PMKID     pmkidList
              [WMI_MAX_PMKID_CACHE];
} __ATTRIB_PACK WMI_SET_PMKID_LIST_CMD;
```

Command Values None

Reset Values None

Restrictions Supported only in infrastructure modes

Name	**SET_POWER_MODE**
Synopsis	The host uses this command to provide the AR6000 device with guidelines on the desired trade-off between power utilization and performance. ■ In normal power mode, the device enters a sleep state if they have nothing to do, which conserves power but may cost performance as it can take up to 2 ms to resume operation after leaving sleep state. ■ In maximum performance mode, the device never enters sleep state, thus no time is spent waking up, resulting in higher power consumption and better performance.
Command	TBD
Command Parameters	see below

`A_UINT8`	`powerMode`	WMI_POWER_MODE value
`{`		
`REC_POWER`	`= 1`	(Recommended setting) Tries to conserve power without sacrificing performance
`MAX_PERF_POWER`	`= 2`	Setting that maximizes performance at the expense of power

```
All other values are reserved
} WMI_POWER_MODE
```

Command Values See command parameters

Reset Values powerMode is REC_POWER

Restrictions

- For a PM-disabled ad hoc network, the power mode should remain in MAX_PERF_POWER.
- For a PM-enabled ad hoc network, the device can have REC_POWER or MAX_PERF_POWER set, but either way it must follow the power save ad hoc protocol. The host can change power modes in the CONNECTED state.

Host changes to the PS setting when the STA is off the home channel take no effect and cause a TARGET_PM_FAIL event.

Name	**SET_POWER_PARAMS**		
Synopsis	The host uses this command to configure power parameters		
Command	**`wmiconfig eth1 --pmparams --it=<ms> --np=<number of PS POLL> --dp=<DTIM policy: ignore/normal/stick>`**		
Command Parameters	`A_UINT16`	`idle_period`	Length of time (in ms) the AR6000 device remains awake after frame Rx/Tx before going to SLEEP state
	`A_UINT16`	`pspoll_number`	The number of PowerSavePoll (PS-poll) messages the device should send before notifying the AP it is awake
	`A_UINT16`	`dtim_policy`	A WMI_POWER_PARAMS_CMD value
	`{`		
	`IGNORE_DTIM`	`=1`	The device does not listen to any content after beacon (CAB) traffic
	`NORMAL_DTIM`	`= 2`	DTIM period follows the listen interval (e.g., if the listen interval is 4 and the DTIM period is 2, the device wakes up every fourth beacon)
	`STICK_DTIM`	`= 3`	Device attempt to receive all CAB traffic (e.g., if the DTIM period is 2 and the listen interval is 4, the device wakes up every second beacon)
	`} WMI_POWER_PARAMS_CMD`		
Command Parameters	See command parameters		
Reset Values	`idle_period`		200 ms
	`pspoll_number`	`= 1`	
	`dtim_policy`	`= NORMAL_DTIM`	
Restrictions	None		

Name	**SET_POWERSAVE_PARAMS**
Synopsis	Set the two AR6000 power save timers (PS-POLL timer and APSD trigger timer) and the two ASPD TIM policies
Command	**`wmiconfig eth1--psparams --psPollTimer=<psPollTimeout in ms> --triggerTimer=<triggerTimeout in ms> --apsdTimPolicy=<ignore/adhere> --simulatedAPSDTimPolicy=<ignore/adhere>`**

Command Parameters

```
typedef struct {
```

`A_UINT16`	`psPollTimeout;`	Timeout (in ms) after sending PS-POLL; the AR6000 device sleeps if it does not receive a data packet from the AP
`A_UINT16`	`triggerTimeout;`	Timeout (in ms) after sending a trigger; the device sleeps if it does not receive any data or null frame from the AP
`APSD_TIM_POLICY`	`apsdTimPolicy;`	TIM behavior with queue APSD enabled
`APSD_TIM_POLICY`	`simulatedAPSD TimPolicy;`	TIM behavior with simulated APSD enabled

```
typedef enum {
IGNORE_TIM_ALL_QUEUES_APSD = 0,
PROCESS_TIM_ALL_QUEUES_APSD = 1,
IGNORE_TIM_SIMULATED_APSD = 2,
POWERSAVE_TIMERS_POLICY = 3,
} APSD_TIM_POLICY;
```

Command Values	None
Reset Values	psPollTimeout is 50 ms; triggerTimeout is 10 ms; apsdTimPolicy = IGNORE_TIM_ALL_QUEUES_APSD; simulatedAPSDTimPolicy = POWERSAVE_TIMERS_POLICY
Restrictions	When this command is used, all parameters must be set; this command does not allow setting only one parameter.

Name	**SET_PROBED_SSID**
Synopsis	The AR6000 keeps a list of up to MAX_PROBED_SSID_INDEX + 1 SSIDs that the device should actively look for. The first entry is reserved for SSID provided by CONNECT CMD. Others entries are set by this command. By default, the device actively looks for only the SSID specified in the "CONNECT_CMD" command, and only when the regulatory domain allows active probing. With this command, specified SSIDs are probed for, even if they are hidden.
Command	`wmiconfig eth1 --ssid=<ssid> [--num=<index>]`

Command Parameters

`A_UINT8`	`entryIndex`	A number from 0 to MAX_PROBED_SSID_INDEX indicating the active SSID table entry index for this command (if the specified entry index already has an SSID, the SSID specified in this command replaces it). For example, etnryIndex = 0 modifies the second entry in the list.
`A_UINT8`	`flag`	WMI_SSID_FLAG indicates the current entry in the active SSID table
`A_UINT8`	`ssidLength`	Length of the specified SSID in bytes. If = 0, the entry corresponding to the index is erased
`A_UINT8`	`ssid[32]`	SSID string actively probed for when permitted by the regulatory domain

Command Values

`WMI_SSID_FLAG`		
`{`		
`DISABLE_SSID_FLAG`	`= 0`	Disables entry
`SPECIFIC_SSID_FLAG`	`= 1`	Probes specified SSID
`ANY_SSID_FLAG`	`= 2`	Probes for any SSID
`} WMI_SSID_FLAG`		

Reset Value The entries are unused.

Restrictions None

Name	**SET_RATE_POLICY**		
Synopsis	Used to set a rate policy with an ID that can be used subsequently in Tx frame commands to control the rates and Tx retries used for certain frames. Designed specifically for use with symbian OS.		
Command	**`TBD`**		
Command Parameters	`A_UINT32`	`ratefield`	A bitfield which represents the allowed rates for this policy. Each bit represents a rate, setting a bit allows that rate to be used. The index is determined by the WMI_BIT_RATE enum.
	`A_UINT8`	`id`	(1-5) The identifier for this policy. This ID is embedded in Tx frame commands causing the device rate control algorithm to use the configured policy.
	`A_UINT8`	`shorttrys`	(1-14) The number of Tx retries used for frames whose length is less than the RTS threshold value.
	`A_UINT8`	`longtrys`	(1-14) The number of Tx retries used for frames whose length is more than the RTS threshold value.
Reset Values	No rate policies are configured after reset.		
Restrictions	Valid identifiers are 1–5 only.		

Name	**SET_REASSOC_MODE**		
Synopsis	Specify whether the disassociated frame should be sent or not upon reassociation.		
Command	**`wmiconfig eth1 --setreassocmode <mode>`**		
Command Parameters	`A_UINT8`	`mode`	
Command Values	`mode`	`= 0x00`	Send disassoc to a previously connected AP upon reassociation
		`= 0x01`	Do not send disassoc to previously connected AP upon reassociation
Reset Values	None defined		
Restrictions	None		

Name SET_RETRY_LIMITS

Synopsis Allows the host to influence the number of times that the AR6000 device should attempt to send a frame before they give up.

Command

```
wmiconfig --setretrylimits <frameType> <trafficClass> <maxRetries>
<enableNotify>
```

Command Parameters {

A_UINT8	frameType	A WMI_FRAMETYPE specifying which type of frame is of interest.
A_UINT8	trafficClass	Specifies a traffic class (see **"CREATE_PSTREAM"**). This parameter is only significant when frameType = DATA_FRAMETYPE.
A_UINT8	maxRetries	Maximum number of times the device attempts to retry a frame Tx, ranging from WMI_MIN_RETRIES (2) to WMI_MAX_RETRIES (15). If the special value 0 is used, maxRetries is set to 15.
A_UINT8	enableNotify	Notify when enabled
} WMI_RETRY_LIMIT_INFO		
}		
A_UINT8	numEntries	
WMI_RETRY_LIMIT_INFO	retryLimitInfo[1]	
} WMI_SET_RETRY_LIMITS_CMD		

Command Values {

MGMT_FRAMETYPE	= 0	Management frame
CONTROL_FRAMETYPE	= 1	Control frame
DATA_FRAMETYPE	= 2	Data frame
} WMI_FRAMETYPE		

Reset Values Retries are set to 15

Restrictions None

Name	**SET_ROAM_CTRL**
Synopsis	Affects how the AR6000 device selects a BSS. The host uses this command to set and enable low RSSI scan parameters. The time period of low RSSI background scan is mentioned in scan period. Low RSSI scan is triggered when the current RSSI threshold (75% of current RSSI) is equal to or less than scan threshold. Low RSSI roam is triggered when the current RSSI threshold falls below the roam threshold and roams to a better AP by the end of the scan cycle. During Low RSSI roam, if the STA finds a new AP with an RSSI greater than roam RSSI to floor, during scan, it roams immediately to it instead of waiting for the end of the scan cycle. See also "Scan and Roam" on page 3-1.
Command	**`wmiconfig --roam <roamctrl> <info>`**, where info is <scan period> <scan threshold> <roam threshold> <roam rssi floor>

Command Parameters

`A_UINT8`	`roamCtrlType;`	

Command Values

`WMI_FORCE_ROAM`	`= 1`	Roam to the specified BSSID
`WMI_SET_ROAM_MODE`	`= 2`	Default, progd bias, no roam
`WMI_SET_HOST_BIAS`	`= 3`	Set the host bias
`WMI_SET_LOWRSSI_SCAN_PARAMS`	`= 4`	Info parameters
`A_UINT8`	`bssid[ATH_MAC_LEN];`	WMI_FORCE_ROAM
`A_UINT8`	`roamMode;`	WMI_SET_ROAM_MODE
`A_UINT8`	`bssBiasInfo;`	WMI_SET_HOST_BIAS
`A_UINT16`	`lowrssi_scan_period;`	WMI_SET_LOWRSSI_SCAN_PARAMS
`A_INT16`	`lowrssi_scan_threshold;`	WMI_SET_LOWRSSI_SCAN_PARAMS
`A_INT16`	`lowrssi_roam_threshold;`	WMI_SET_LOWRSSI_SCAN_PARAMS
`A_UINT8`	`roam_rssi_floor;`	WMI_SET_LOWRSSI_SCAN_PARAMS

Reset Value	None defined (default lowrssi scan is disabled. Enabled only when scan period is set.)
Restrictions	None

Name	**SET_RTS**
Synopsis	Decides when RTS should be sent.
Command	**`wmiconfig eth1 --setRTS <pkt length threshold>`**

Command Parameters

`A_UINT16`	`threshold;`	Command parameter threshold in bytes. An RTS is sent if the data length is more than this threshold. The default is to NOT send RTS.

Command Values	None
Reset Value	Not to send RTS.
Restrictions	None

Name **SET_SCAN_PARAMS**

Synopsis The host uses this command to set the AR6000 scan parameters, including the duty cycle for both foreground and background scanning. Foreground scanning takes place when the AR6000 device is not connected, and discovers all available wireless networks to find the best BSS to join. Background scanning takes place when the device is already connected to a network and scans for potential roaming candidates and maintains them in order of best to worst. A second priority of background scanning is to find new wireless networks.

The device initiates a scan when necessary. For example, a foreground scan is always started on receipt of a "CONNECT_CMD" command or when the device cannot find a BSS to connect to. Foreground scanning is disabled by default until receipt of a **CONNECT** command. Background scanning is enabled by default and occurs every 60 seconds after the device is connected.

The device implements a binary backoff interval for foreground scanning when it enters the DISCONNECTED state after losing connectivity with an AP or when a **CONNECT** command is received. The first interval is ForegroundScanStartPeriod, which doubles after each scan until the interval reaches ForegroundScanEndPeriod. If the host terminates a connection with **DISCONNECT**, the foreground scan period is ForegroundScanEndPeriod. All scan intervals are measured from the time a full scan ends to the time the next full scan starts. The host starts a scan by issuing a **"START_SCAN"** command. See also "Scan and Roam" on page 3-1.

Command

```
wmiconfig eth1 --scan --fgstart=<sec> --fgend=<sec> --bg=<sec> --
act=<msec> --pas=<msec> --sr=<short scan ratio> --scanctrlflags
<connScan> <scanConnected> <activeScan> <reportBSSINFO>
```

Command Parameters

A_UINT16	fg_start_period	First interval used by the device when it disconnects from an AP or receives a **CONNECT** command, specified in seconds (0–65535). If = 0, the device uses the reset value. If = 65535, the device disables foreground scanning.
A_UINT16	fg_end_period	The maximum interval the device waits between foreground scans specified in seconds (from ForegroundScanStartPeriod to 65535). If = 0, the device uses the reset value.
A_UINT16	bg_period	The period of background scan specified in seconds (0–65535). By default, it is set to the reset value of 60 seconds. If 0 or 65535 is specified, the device disables background scanning.
A_UINT16	max_chdwell_time	The period of time the device stays on a particular channel while active scanning. It is specified in ms (10–65535). If the special value of 0 is specified, the device uses the reset value.
A_UINT16	pas_chdwell_time	The period of time the device remains on a particular channel while passive scanning. It is specified in ms (10–65535). If the special value of 0 is specified, the device uses the reset value.
A_UINT8	shortScanRatio	Number of short scans to perform for each long scan.
A_UINT8	scanCtrlFlags	WMI_SCAN_CTRL_FLAGS_BITS
A_UINT16	min_chdwell_time	Specified in ms
A_UINT32	max_dfsch_act_time	Maximum time a DFS channel can stay active before being marked passive, specified in ms.

SET_SCAN_PARAMS, continued

Command Values

`CONNECT_SCAN_CTRL_FLAGS = 0x01`	Set whether the AR6000 can scan using the connect command, otherwise no probe request is sent out during connect.
`SCAN_CONNECTED_CTRL_FLAGS = 0x02`	Set whether the AR6000 can scan for the SSID it will connect to or is already connected to.
`ACTIVE_SCAN_CTRL_FLAGS = 0x04`	Set whether active scan is enabled for the SSID
`ROAM_SCAN_CTRL_FLAGS = 0x08`	Set whether roam scan is enabled when BMISS and a low RSSI event happens
`REPORT_BSSINFO_CTRL_FLAGS = 0x10`	Set whether follows customer BSSINFO reporting rule
`ENABLE_AUTO_CTRL_FLAGS = 0x20`	If not set, the target does not scan automatically to find an AP after a disconnect event.
`ENABLE_SCAN_ABORT_EVENT = 0x40`	A scan complete event with canceled status is generated when a scan is preempted before it completes.

```
}
```

Reset Values

`ForegroundScanStartPeriod`	1 sec
`ForegroundScanEndPeriod`	60 sec
`BackgroundScanPeriod`	60 sec
`ActiveChannelDwellTime`	105 ms

Name **SET_TARGET_EVENT_REPORT**

Synopsis The host uses this command to filter out the events from the target, which host is not interested in. The current software release supports filtering out the WMI_DISCONNECTED_EVENT with the reason = BSS_DISCONNECTED when the host or target reconnects with the AP. The reconnection sequence with the AP (e.g., RECONNECT WMI command) causes the target to send a WMI_DISCONNECTED_EVENT with reason = CSERV_DISCONNECT first followed by WMI_DISCONNECTED_EVENT with reason = BSS_DISCONNECTED. If the host is not interested in the second WMI_DISCONNECTED_EVENT with reason = BSS_DISCONNECTED, the host can use this command to do so.

Command **`wmiconfig –eth1 -settgtevt <event value>`**

Command Parameters

`struct {`		
`A_UINT32`	`evtConfig;`	The enum parameter of type TARGET_EVENT_REPORT_CONFIG
`}WMI_SET_TARGET_EVENT_REPORT_CMD;`		
`enum {`		
`DISCONN_EVT_IN_RECONN = 0,`		Default
`NO_DISCONN_EVT_IN_RECONN`		
`}TARGET_EVENT_REPORT_CONFIG;`		

Command Values

`evtConfig = 0`	Send a WMI_DISCONNECT_EVENT with disconnectReason = BSS_DISCONNECTED after re-connection with AP
`evtConfig = 1`	Do not send WMI_DISCONNECT_EVENT with disconnectReason = BSS_DISCONNECTED after re-connection with AP. Reconnection with the AP already sends DISCONNECT_EVENT with disconnectReason = CSERV_DISCONNECT first.

Reset Values Default evtConfig = 0

Restrictions None

Name	**SET_THIN_MODE**
Synopsis	The host uses this command to enable thin mode on the target. Thin mode is intended to be used by Operating Systems / host frameworks that own the 802.11 mlme operations. Examples include Symbian OS and linux MAC80211.
Command	`TBD`
Command Parameters	`struct {`
	`AUINT8` `enable` — Enables or disable thin mode functionality on the target
	`}`
Command Values	`0` Disable
	`1` Enable
Reset Values	0
Restrictions	Should be set one time prior to invoking any WLAN operations.

Name	**SET_TKIP_COUNTERMEASURES**
Synopsis	The host issues this command to tell the target whether to enable or disable TKIP countermeasures.
Command	TBD
Command Parameters	`A_UINT8` `WMI_TKIP_CM_ENABLE` — Enables the countermeasures
	`A_UINT8` `TKIP_CM_DISABLE` — Disables the countermeasures
Command Values	None
Reset Values	By default, TKIP MIC reporting is disabled
Restrictions	None

Name	**SET_TX_PWR**
Synopsis	The host uses this command to specify the Tx power level of the AR6000. Cannot be used to exceed the power limit permitted by the regulatory domain. The maximum output power is limited in the chip to 31.5 dBm; the range is 0 – 31.5 dbm.
Command	**`wmiconfig --power <dbM>`**
Command Parameters	`A_UINT8` `dbM` — The desired Tx power specified in dbM
Command Values	None
Reset Values	The maximum permitted by the regulatory domain
Restrictions	None
See Also	"GET_TX_PWR" on page A-19

Name	**SET_TX_SELECTRATES**
Synopsis	This command allows the host to configure the allowable bit rates used to transmit packets for each operational mode. The operational modes are defined by the WLAN_PHY_MODE enum. The allowed rates are provided in the form of a bitmask where each bit corresponds to a rate the index of which is determined by the WMI_BIT_RATE enum. The device rate control logic will use these masks to determine if a rate should not be used. Other factors are also taken into consideration to determine whether a rate can be used.
Command	**`wmiconfig eth1 --settxselrates [MODE_11A] [MODE_11G] [MODE_11B] [MODE_11GONLY] [MODE_11NA_HT20] [MODE_11NG_HT20] [MODE_11NA_HT40] [MODE_11NG_HT40]`**

Command Parameters	`A_UINT32` `mode11AMask`
	`A_UINT32` `mode11GMask`
	`A_UINT32` `mode11BMask`
	`A_UINT32` `mode11GOnlyMask`
	`A_UINT32` `mode11NAHT20Mask`
	`A_UINT32` `mode11NGHT20Mask`
	`A_UINT32` `mode11NAHT40Mask`
	`A_UINT32` `mode11NGHT40Mask`

Command Values	None
Reset Values	mode11AMask = 0x00000FF0 A_UINT32 mode11GMask = 0x00000FFF A_UINT32 mode11BMask = 0x0000000F A_UINT32 mode11GOnlyMask = 0x00000FF0 A_UINT32 mode11NAHT20Mask = 0x000FF000 A_UINT32 mode11NGHT20Mask = 0x000FF007 A_UINT32 mode11NAHT40Mask = 0x0Fc0F000 A_UINT32 mode11NGHT40Mask = 0x0Fc0F007
Restrictions	The command must be issued prior to association with a BSS for the new values to take affect.

Name	**SET_VOICE_PKT_SIZE**
Synopsis	If an AP does not support WMM, it has no way to differentiate voice from data. Because the voice packet is typically small, packet in size less than voicePktSize are assumed to be voice, otherwise it is treated as data.
Command	**`wmiconfig eth1 --setVoicePktSize <size-in-bytes>`**
Command Parameters	`A_UINT16` `voicePktSize` Packet size in octets
Command Values	None
Reset Values	voicePktSize default is 400 bytes
Restrictions	No effect if WMM is unavailable

Name	**SET_WHAL_PARAM**
Synopsis	An internal AR6000 command that is used to set certain hardware parameters. The description of this command is in $WORKAREA/include/halapi.h.
Command	TBD
Command Parameters	`ATH_HAL_SETCABTO_CMDID` Sets the timeout waiting for the multicast traffic after a DTIM beacon (in TUs).
Command Values	None
Reset Value	Default = 10 TUs
Restrictions	This command should be executed before issuing a connect command.

Name	**SET_WMM**	
Synopsis	Overrides the AR6000 device WMM capability	
Command	**`wmiconfig eth1 --setwmm <enable>`**	
Command Parameters	`WMI_WMM_ENABLED`	Enables WMM
	`WMI_WMM_DISABLED`	Disables WMM support

Command Values	`0 = disabled`
	`1 = enabled`	

Reset Value	WMM Disabled	
Restrictions	None	

Name	**SET_WMM_TXOP**		
Synopsis	Configures TxOP Bursting when sending traffic to a WMM capable AP		
Command	**`wmiconfig eth1 --txopbursting <burstEnable>`**		
	`<burstEnable>`	`= 0`	Disallow TxOp bursting
		`= 1`	Allow TxOp bursting
Command Parameters	`txopEnable`	`= WMI_TXOP_DISABLED`	Disabled
		`= WMI_TXOP_ENABLED`	Enabled
Command Values	`txopEnable`	`= 0`	Disabled
		`= 1`	Enabled
Reset Value	Bursting is off by default		
Restrictions	None		

Name	**SET_WOW_MODE**		
Synopsis	The host uses this command to enable or disable the WoW mode. When WoW mode is enabled and the host is asleep, pattern matching takes place at the target level. Only packets that match any of the pre-specified WoW filter patterns, will be passed up to the host. The host will also be woken up by the target. Packets which do not match any of the WoW patterns are discarded.		
Command	**`wmiconfig -setwowmode <enable/disable>`**		
Command Parameters	`A_BOOL`	`enable_wow`	Enable or disable WoW:
Command Values		`= 0`	Disable WoW
		`= 1`	Enable WoW
Reset Value	None defined (default WoW mode is disabled).		
Restrictions	None		
See Also	"GET_WOW_LIST" on page A-19		

Name	**SET_WSC_STATUS**		
Synopsis	The supplicant uses this command to inform the target about the status of the WSC registration protocol. During the WSC registration protocol, a flag is set so the target bypasses some of the checks in the CSERV module. At the end of the registration, this flag is reset.		
Command	N/A		
Command Parameters	`A_BOOL status`	`= 1`	WSC registration in progress
		`= 0`	WSC protocol not running
Reply Parameters	None		
Reset Value	None defined (default = 0)		
Restrictions	None		

Name	**SNR_THRESHOLD_PARAMS**	
Synopsis	Configures how the AR6000 device monitors and reports SNR of the connected BSS, used as a link quality metric.	
Command	**`--snrThreshold <weight> <upper_threshold_1> ... <upper_threshold_4> <lower_threshold_1> ... <lower_threshold_4> <pollTimer>`**	
Command Parameters	`<weight>`	Share with rssiThreshold. Range in [1, 16], used in the formula to calculate average RSSI
	`<upper_threshold_x>`	Above thresholds expressed in db, in ascending order
	`<lower_threshold_x>`	Below thresholds expressed in db, in ascending order
	`<pollTimer>`	The signal strength sampling frequency in seconds. If polltime = 0, signal strength sampling is disabled
Command Values	None	
Reset Value	None defined	
Restrictions	None	

Name	**START_SCAN**
Synopsis	The host uses this command to start a long or short channel scan. All future scans are relative to the time the AR6000 device processes this command. The device performs a channel scan on receipt of this command, even if a scan was already in progress. The host uses this command when it wishes to refresh its cached database of wireless networks. The isLegacy field will be removed (0 for now) because it is achieved by setting CONNECT_PROFILE_MATCH_DONE in the CONNECT command. See also "Scan and Roam" on page 3-1.
Command	`wmiconfig eth1 --startscan <scan type> <forcefgscan> 0 <homeDwellTime> <forceScanInterval> <list>`

Command Parameters	`A_UINT8`	`scanType`	WMI_SCAN_TYPE
Command Values	`{`		
	`WMI_LONG_SCAN`	`=0x0`	Requests a full scan
	`WMI_SHORT_SCAN`	`=0x1`	Requests a short scan
	`} WMI_SCAN_TYPE`		
	`A_BOOL`	`forceFgScan`	
	`forceFgScan`	`= 0`	Disable the foreground scan
	`forceFgScan`	`= 1`	Forces a foreground scan
	`A_UINT32`	`homeDwellTime`	Maximum duration in the home channel (in ms)
	`A_UINT32`	`forceScanInterval`	Time interval between scans (in ms)
Reset Value	Disable forcing foreground scan		
Restrictions	isLegacy field will no longer be supported (pass as 0 for now)		

Name	**SYNCHRONIZE**
Synopsis	The host uses this command to force a synchronization point between the command and data paths
Command	TBD
Command Parameters	None
Command Values	None
Reset Values	None
Restrictions	None

Name	TARGET_ERROR_REPORT_BITMASK
Synopsis	Allows the host to control **"ERROR_REPORT"** events from the AR6000 device. ■ If error reporting is disabled for an error type, a count of errors of that type is maintained by the device. ■ If error reporting is enabled for an error type, an **"ERROR_REPORT"** event is sent when an error occurs and the error report bit is cleared. Error counts for each error type are available through the **"GET_TARGET_STATS"** command.
Command	`wmiconfig eth1 --setErrorReportingBitmask`

Command Parameters

`A_UINT32`	`bitmask`	Represents the set of WMI_TARGET_ERROR_VAL error types enabled for reporting

Command Values

`{`

`WMI_TARGET_PM_ERR_FAIL`	=	`0x00000001`	Power save fails (only two cases): ■ Retry out of null function/QoS null function to associated AP for PS indication' ■ Host changes the PS setting when STA is off home channel
`WMI_TARGET_KEY_NOT_FOUND`	=	`0x00000002`	No cipher key
`WMI_TARGET_DECRYPTION_ERR`	=	`0x00000004`	Decryption error
`WMI_TARGET_BMISS`	=	`0x00000008`	Beacon miss
`WMI_PSDISABLE_NODE_JOIN`	=	`0x00000010`	A non-PS-enabled STA joined the PS-enabled network
`WMI_TARGET_COM_ERR`	=	`0x00000020`	Host/target communication error
`WMI_TARGET_FATAL_ERR`	=	`0x00000040`	Fatal error

`} WMI_TARGET_ERROR_VAL`

Reset Values	Bitmask is 0, and all error reporting is disabled
Restrictions	None

Name	**TEST_CMD**
Synopsis	Multiplexes the four separate deleted WMI commands; these IOCTLs continue to be supported: WMI_TCMD_ENABLE, WMI_TCMD_CONT_TX, WMI_TCMD_CONT_RX, WMI_TCMD_PM See **$WORKAREA/host/tools/tcmd/athtestcmd –usage** for more information.
Command	**`testcmd --tx <sine/frame/tx99/tx100/off>`**

Command Parameters

`--txfreq`	Tx channel or frequency (default = 2412)	
`--txrate`	Rate index	
`--txpwr`	Frame: 0–14dBm: 0–11dBm (default = 0 for CUS88, = 10 for others)	
`--txantenna`	1/2/0 (auto)	
`--txpktsz`	Packet size, [32–1500] (default = 1500)	
`--txpattern`	Tx data pattern	
	`= 0`	All zeros
	`= 1`	All ones
	`= 2`	Repeating 10
	`= 3`	PN7
	`= 4`	PN9
	`= 5`	PN15
`--ani`	Enable ANI. The ANI is disabled if this option is not specified.	
`--scrambleroff`	Disable scrambler. The scrambler is enabled by default.	
`--aifsn`	AIFS slots number, [0–255], used only under --tx frame mode	
`--rx`	Filter/report	
`--rxfreq`	Rx channel or frequency (default = 2412)	
`--rxantenna`	1/2/0 (auto)	
`--pm`	wakeup/sleep	
`--setmac`	MAC address, e.g, 00:03:7f:be:ef:11	
`--setAntSwitchTable <table1_decimal_value> <table2_decimal_value>`	Set table1=0 and table2=0 will restore the default AntSwitchTable	

Command Values

`TCMD_CONT_TX`	`= 801`
`TCMD_CONT_RX`	`= 802`
`TCMD_CONT_PM`	`= 803`

Reset Values	None defined
Restrictions	None

Name	**TX_SGI_PARAM**
Synopsis	Configure the use of SGI for MCS rates. Bss restrictions will also be respected when deciding whether to use SGI for a given rate.
Command	**`TBD`**

Command Parameters

`AUINT_32`	`sgiMask`	A bitfield which represents the rates where SGI can be used. The index is determined by the WMI_BIT_RATE enum.
`A_UINT8`	`sgiPERThreshold`	The Packet Error Rate (PER) threshold above which SGI should not be used.

Reset Values

sgiMask	0x00080000 @ 20 MHZ and 0x08000000 @ 40 MHZ effectively allowing only MCS 7 to use SGI.
sgiPERThreshold	= 10. If the PER exceeds 10% then SGI cannot be used.

Restrictions The command must be issued prior to association with a BSS for the new values to take effect.

WMI Events

Table A-4 lists the WMI events.

Table A-4. **WMI Events**

Event	Description	Page
APLIST_EVENT	Event for AP list; available for customers who have a CCX license	—
BSSINFO	Contains information describing BSSs collected during a scan	page A-54
CAC_EVENTID	Indicates signalling events in admission control	page A-55
CMDERROR	The AR6000 device encounters an error while attempting to process a command	page A-55
CONNECT	The device has connected to a wireless network	page A-56
DISCONNECT	The device lost connectivity with a wireless network	page A-57
ERROR_REPORT	An error has occurred for which the host previously requested notification with the command "TARGET_ERROR_REPORT_BITMASK"	page A-58
EXTENSION	WMI extension event	page A-58
GET_PMKID_LIST_EVENT	Created in response to a "GET_PMKID_LIST_CMD" command	page A-58
GET_WOW_LIST_EVENT	Response to the wmiconfig "GET_WOW_LIST" command to retrieve the configured WoW patterns	page A-59
NEIGHBOR_REPORT	Neighbor APs that match the current profile were detected	page A-60
OPT_RX_FRAME_EVENT	(Special feature) informs the host of the reception of a special frame	page A-60
PSTREAM_TIMEOUT	A prioritized stream has been idle for a specified interval	page A-60
READY	The AR6000 device is ready to accept commands	page A-61
REGDOMAIN	The regulatory domain has changed	page A-61
REPORT_ROAM_DATA _EVENT	Reports the roam time calculations made by the device (generated with a special build)	—
REPORT_STATISTICS	Reply to a "GET_TARGET_STATS" command	page A-62
ROAM_TBL_EVENT	Reports the roam table	page A-64
RSSI_THRESHOLD	Signal strength from the connected AP has crossed the threshold defined in the "RSSI_THRESHOLD_PARAMS" command	page A-64
SCAN_COMPLETE_EVENT	A scan has completed (added status SCAN_ABORTED in release 2.0)	page A-64
TEST_EVENT	Event generated by the TCMD	page A-65
TKIP_MICERROR	TKIP MIC errors were detected	page A-65

Name	**BSSINFO**	
Synopsis	Contains information describing one or more BSSs as collected during a scan. Information includes the BSSID, SSID, RSSI, network type, channel, supported rates, and IEs. **BSSINFO** events are sent only after the device receives a beacon or probe-response frame that pass the filter specified in the **"SET_BSS_FILTER"** command. **BSSINFO** events consist of a small header followed by a copy of the beacon or probe response frame. The 802.11 header is not present. For formats of beacon and probe-response frames please consult the IEEE 802.11 specification. The beacons or probe responses containing the IE specified by the WMI_BSS_FILTER_CMD are passed to the host through the WMI_BSSINFO_EVENT. The event carries a 32-bit bitmask that indicates the IEs that were detected in the management frame. The frame type field has been extended to indicate action management frames. This would be helpful to route these frames through the same event mechanism as used by the beacon processing function. If the bssFilter in the SET_BSS_FILTER matches, then the ieMask is not relevant because the BSSINFO event is sent to the host. If the bssFilter does not match in the beacons/probe responses, then the ieMask match dictates whether the BSSINFO event is sent to the host. In the case of action management frames, the ieMask is the filter that is applied.	
Event ID	0x1004	
Event Parameters	`typedef struct {`	
	`A_UINT16    channel;`	Specifies the frequency (in MHz) where the frame was received
	`A_UINT8     frameType;`	A WMI_BI_FTYPE value
	`A_UINT8     snr;`	
	`A_INT16     rssi;`	Indicates signal strength
	`A_UINT8     bssid[ATH_MAC_LEN];`	
	`A_UINT32    ieMask;`	
	`} _ATTRIB_PACK_WMI_BSS_INFO_HDR;`	
	Beacon or Probe Response Frame	
Event Values	`{`	
	`BEACON_FTYPE = 0x1`	Indicates a beacon frame
	`PROBERESP_FTYPE`	Indicates a probe response frame
	`ACTION_MGMT_FTYPE`	
	`} WMI_BI_FTYPE`	

Name	**CAC_EVENTID**		
Synopsis	Indicates signalling events in admission control. Events are generated when admission is accepted, rejected, or deleted by either the host or the AP. If the AP does not respond to an admission request within a timeout of 500 ms, an event is generated to the host.		
Event ID	0x1011		
Event Parameters	`A_UINT8`	`ac`	Access class pertaining to the signalling
	`A_UINT8`	`cac_indication`	Type of indication; indications are listed in WMI_CAC_INDICATION
	`A_UINT8`	`statusCode`	AP response status code for request
	`A_UINT8`	`tspecSuggestion[63]`	Suggested TSPEC from AP
Event Values	{		
	`CAC_INDICATION_ADMISSION`	`= 0x00`	
	`CAC_INDICATION_ADMISSION_RESP`	`= 0x01`	
	`CAC_INDICATION_DELETE`	`= 0x02`	
	`CAC_INDICATION_NO_RESP`	`= 0x03`	
	`} WMI_CAC_INDICATION`		

Name	**CMDERROR**		
Synopsis	Indicates that the AR6000 device encountered an error while attempting to process a command. This error is fatal and indicates that the device requires a reset.		
Event ID	0x1005		
Event Parameters	`A_UINT16`	`commandId`	Corresponds to the command that generated the error
	`A_UINT8`	`errorCode`	A WMI_ERROR_CODE value
Event Values	{		
	`INVALID_PARAM`	`= 1`	Invalid parameter
	`ILLEGAL_STATE`	`= 2`	Illegal state
	`INTERNAL_ERROR`	`= 3`	Internal Error
	`All other values reserved`		
	`} WMI_ERROR_CODE`		

Name	CONNECT		
Synopsis	Signals that the AR6000 connected to a wireless network. Connection occurs due to a **"CONNECT"** command or roaming to a new AP. For infrastructure networks, shows that the AR6000 successfully performed 802.11 authentication and AP association.		
Event ID	0x1002		
Event Parameters	`A_UINT16`	`channel`	Channel frequency (in MHz) of the network the AR6000 is connected to
	`A_UINT8`	`bssid[6]`	MAC address of the AP the AR6000 is connected to or the BSSID of the ad hoc network
	`A_UINT16`	`listenInterval`	Listen interval (in Kμs) that the AR6000 is using
	`A_UINT16`	`beaconInterval`	Beacon interval the connected AP is using
	`A_UINT8`	`networkType`	Network type in NETWORK_TYPE
	`UINT 8`	`beaconIeLen`	Length (in bytes) of the beacon IEs
	`A_UINT8`	`assocInfo`	Pointer to an array containing beacon IEs, followed first by association request IEs then by association response IEs
	`A_UINT8`	`assocReqLen`	Length (in bytes) of the assocReqIEs array
	`A_UINT8`	`assocRespLen`	Length (in bytes) of the assocRespIEs array
Event Values	`INFRA_NETWORK = 0x01`		Infrastructure network
	`ADHOC_NETWORK = 0x02`		Ad hoc network
	`ADHOC_CREATOR = 0x04`		Ad hoc network created by the AR6000

Name DISCONNECT

Synopsis Signals that the AR6000 device lost connectivity with the wireless network. DISCONENCT is generated when the device fails to complete a **"CONNECT"** command or as a result of a transition from a connected state to disconnected state.

After sending the **"DISCONNECT"** event the device continually tries to re-establish a connection, if roaming is enabled. A LOST_LINK occurs when STA cannot receive beacons within the specified time for the SET_BMISS_TIME command.

Event ID 0x1003

Event Parameters

`A_UINT16`	`protocolReason Status`	Reason code (see the 802.11 specification)
`A_UINT8`	`bssid[6]`	Indicates which BSS the device was connected to
`A_UINT8`	`disconnect Reason`	A WMI_DISCONNECT_REASON value
`A_UINT8`	`assocRespLen`	Length of the 802.11 association response frame that triggered this event, or 0 if not applicable
`A_UINT8`	`assocInfo [assocRespLen]`	Copy of the 802.11 association response frame

Event Values {

`NO_NETWORK_ AVAIL`	`=0x01`	Indicates that the device was unable to establish or find the desired network
`LOST_LINK`	`=0x02`	Indicates the devices is no longer receiving beacons from the BSS it was previously connected to
`DISCONNECT_ CMD`	`=0x03`	Indicates a **"DISCONNECT"** command was processed
`BSS_DISCONNE CTED`	`=0x04`	Indicates the BSS explicitly disconnected the device. Possible mechanisms include the AP sending 802.11 management frames (e.g., disassociate or deauthentication messages).
`AUTH_FAILED`	`=0x05`	Indicates that the device failed 802.11 authentication with the BSS
`ASSOC_FAILED`	`=0x06`	Indicates that the device failed 802.11 association with the BSS
`NO_RESOURCES _AVAIL`	`=0x07`	Indicates that a connection failed because the AP had insufficient resources to complete the connection
`CSERV_ DISCONNECT`	`=0x08`	Indicates that the device's connection services module decided to disconnect from a BSS, which can happen for a variety of reasons (e.g., the host marks the current connected AP as a bad AP).
`INVALID_ PROFILE`	`=0x0A`	Indicates that an attempt was made to reconnect to a BSS that no longer matches the current profile
`DOT11H _CHANNEL _SWITCH`	`=0x0B`	Indicates that the AP the device is connected to has sent the DOT11H CSA IE.

`PROFILE _MISMATCH`	`=0x0C`	Indicates that the AP the device is associated to has changed its profile and the station has detected the profile change.

```
All other values are reserved
} WMI_DISCONNECT_REASON
```

Name ERROR_REPORT

Synopsis Signals that a type of error has occurred for which the host previously requested notification through the **"TARGET_ERROR_REPORT_BITMASK"** command.

Event ID 0x100D

Event Parameters

`A_UINT32`	`errorVal`	WMI_TARGET_ERROR_VAL value. See **"TARGET_ERROR_REPORT_BITMASK"**.

Event Values

`errorVal`	`= 0x00000001`	Power save fails
	`= 0x00000002`	No cipher key
	`= 0x00000004`	Decryption error
	`= 0x00000008`	Beacon miss
	`= 0x00000010`	A non-power save disabled node has joined the PS-enabled network

Name EXTENSION

Synopsis The WMI is used mostly for wireless control messages to a wireless module that apply to wireless module management regardless of the target platform implementation. However, some events peripherally related to wireless management are desired during operation. These wireless extension events may be platform-specific or implementation-dependent. See "WMI Extension Commands" on page A-66.

Event ID 0x1010

Name GET_PMKID_LIST_EVENT

Synopsis Generated by firmware in response to a **"GET_PMKID_LIST_CMD"** command.

Event Parameters

```
typedef struct {
```

`A_UINT32`	`numPMKID;`	Contains the number of PMKIDs in the reply
`WMI_PMKID`	`pmkidList[1];`	

```
} __ATTRIB_PACK WMI_PMKID_LIST_REPLY;
```

Event Values None

Name	**GET_WOW_LIST_EVENT**
Synopsis	Response to the wmiconfig –getwowlist command to retrieve the configured Wake on Wireless patterns
Event ID	0x10018
Event Parameters	{

`A_UINT8`	`num_filters`	Total number of patterns in the list
`A_UINT8`	`this_filter_num`	The filter number
`A_UINT8`	`wow_mode`	Shows whether WoW is enabled or disabled
`A_UINT8`	`host_mode`	Shows whether the host is asleep or awake
`WOW_FILTER`	`wow_filters[1]`	List of WoW filters (pattern and mask data bytes)
} `WMI_GET_WOW_LIST_REPLY;`		
{		Each WOW_FILTER_LIST element shows:
`A_UINT8`	`wow_valid_filter`	Whether the filter is valid
`A_UINT8`	`wow_filter_list_id`	Filter List ID (23 = default)
`A_UINT8`	`wow_filter_size`	Size in bytes of the filter
`A_UINT8`	`wow_filter _offset`	Offset of the pattern to search in the data packet
`A_UINT8`	`wow_filter_mask[MASK_SIZE]`	The mask to be applied to the pattern
`A_UINT8`	`wow_filter_pattern[WOW_PATTERN_SIZE]`	The pattern that to match to wake up the host
} `WOW_FILTER`		

Event Values None

Name	**NEIGHBOR_REPORT**
Synopsis	Indicates the existence of neighbor APs that match the current profile. The host uses this event to populate the PMKID cache on the AR6000 and/or to perform preauthentication. This event is only generated in infrastructure mode. A total of numberOfAps pairs of bssid/bssFlags exist, one pair for each AP.
Event ID	0x1008

Event Parameters

`A_UINT8`	`numberOfAps`	The number of APs reported about in this event
{		
`A_UINT8`	`bssid[6]`	MAC address of a neighbor AP
`A_UINT8`	`bssFlags`	A WMI_BSS_FLAGS value
}`[numberOfAps]`		

Event Values

{		
`WMI_DEFAULT_BSS_FLAGS`	`= 0`	Logical OR of 1 or more WMI_BSS_FLAGS
`WMI_PREAUTH_CAPABLE_BSS`	`= 1`	Indicates that this AP is capable of preauthentication
`WMI_PMKID_VALID_BSS`	`= 2`	Indicates that the AR6000 have a valid pairwise master key for this AP
} `WMI_BSS_FLAGS`		

Name	**OPT_RX_FRAME_EVENT**
Synopsis	Special feature, informs host of the reception of a special frame.
Event ID	0x100E

Event Parameters

```
{
A_UINT16   channel;
A_UINT8    frameType;
A_INT8     snr;
A_UINT8    srcAddr[ATH_MAC_LEN];
A_UINT8    bssid[ATH_MAC_LEN];
}WMI_OPT_RX_INFO_HDR
```

Event Values None

Name	**PSTREAM_TIMEOUT**
Synopsis	Indicates that a priority stream that got created as a result of priority-marked data flow (priority marked in IP TOS) being idle for the default inactivity interval period (specified in the **"CREATE_PSTREAM"** command) used for priority streams created implicitly by the driver. This event is not indicated for user-created priority streams. User-created priority streams exist until the users delete them explicitly. They do not timeout due to data inactivity.
Event ID	0x1007

Event Parameters

`A_UINT8`	`txQueueNumber`	Obsolete
`A_UINT8`	`rxQueueNumber`	Obsolete

	`A_UINT8`	`trafficDirection`	Obsolete
	`A_UINT8`	`trafficClass`	Indicated the traffic class of priority stream that timed out
Event Values	`{`		
	`WMM_AC_BE`	`= 0`	Best effort
	`WMM_AC_BK`	`= 1`	Background
	`WMM_AC_VI`	`= 2`	Video
	`WMM_AC_VO`	`= 3`	Voice
	`} TRAFFIC CLASS`		

Name READY

Synopsis Indicates that the AR6000 device is prepared to accept commands. It is sent once after power on or reset. It also indicates the MAC address of the device.

Event ID 0x1001

Event Parameters	`A_UINT8`	`macAddr[6]`	Device MAC address
	`A_UINT8`	`phyCapability`	A WMI_PHY_CAPABILITY value. Indicates the capabilities of the device wireless module's radio
	`A_UINT32`	`ramVersion`	Target RAM build version

Event Values `{`

`WMI_11A_CAPABILITY`	`= 1`
`WMI_11G_CAPABILITY`	`= 2`
`WMI_11AG_CAPABILITY`	`= 3`

`} WMI_PHY_CAPABILITY`

Name REGDOMAIN

Synopsis Indicates that the regulatory domain has changed. It initially occurs when the AR6000 device reads the board data information. The regulatory domain can also change when the device is a world-mode SKU. In this case, the regulatory domain is based on the country advertised by APs per the IEEE 802.11d specification. A potential side effect of a regulatory domain change is a change in the list of available channels. Any channel restrictions that exist as a result of a previous **"SET_CHANNEL_PARAMETERS"** command are lifted.

Event ID 0x1006

Event Parameters	`A_UINT32`	`regDomain`	The range of 0x0000 – 0x00FF corresponds to an ISO country code. Other regCodes are reserved for world mode settings and specific regulatory domains.

Event Values `None`

Name	REPORT_STATISTICS
Synopsis	A reply to a **"GET_TARGET_STATS"** command.
Event ID	0x100B
Event Parameters	When the statistics are sent to the host, the AR6000 clear them so that a new set of statistics are collected for the next report.

`A_UINT32`	`tx_packets`	
`A_UINT32`	`tx_bytes`	
`A_UINT32`	`tx_unicast_pkts`	
`A_UINT32`	`tx_unicast_bytes`	
`A_UINT32`	`tx_multicast_pkts`	
`A_UINT32`	`tx_multicast_bytes`	
`A_UINT32`	`tx_broadcast_pkts`	
`A_UINT32`	`tx_broadcast_bytes`	
`A_UINT32`	`tx_rts_success_cnt`	
`A_UINT32`	`tx_packet_per_ac[4]`	Tx packets per AC: [0] = BE, [1] = BK, [2] = VI, [3] = VO
`A_UINT32`	`tx_errors`	Number of packets which failed Tx, due to all failures
`A_UINT32`	`tx_failed_cnt`	Number of data packets that failed Tx
`A_UINT32`	`tx_retry_cnt`	Number of Tx retries for all packets
`A_UINT32`	`tx_rts_fail_cnt`	Number of RTS Tx failed count
`A_UINT32`	`rx_packets`	
`A_UINT32`	`rx_bytes`	
`A_UINT32`	`rx_unicast_pkts`	
`A_UINT32`	`rx_unicast_bytes`	
`A_UINT32`	`rx_multicast_pkts`	
`A_UINT32`	`rx_multicast_bytes`	
`A_UINT32`	`rx_broadcast_pkts`	
`A_UINT32`	`rx_broadcast_bytes`	

... REPORT_STATISTICS, continued

`A_UINT32`	`rx_fragment_pkt`	Number of fragmented packets received
`A_UINT32`	`rx_errors`	Number of Rx errors due to all failures
`A_UINT32`	`rx_crcerr`	Number of Rx errors due to CRC errors
`A_UINT32`	`rx_key_cache_miss`	Number of Rx errors due to a key not being plumbed
`A_UINT32`	`rx_decrypt_err`	Number of Rx errors due to decryption failure
`A_UINT32`	`rx_duplicate_frames`	Number of duplicate frames received
`A_UINT32`	`tkip_local_mic_failure`	Number of TKIP MIC errors detected
`A_UINT32`	`tkip_counter_measures_invoked`	Number of times TKIP countermeasures were invoked
`A_UINT32`	`tkip_replays`	Number of frames that replayed a TKIP encrypted frame received earlier
`A_UINT32`	`tkip_format_errors`	Number of frames that did not conform to the TKIP frame format
`A_UINT32`	`ccmp_format_errors`	Number of frames that did not conform to the CCMP frame format
`A_UINT32`	`ccmp_replays`	Number of frames that replayed a CCMP encrypted frame received earlier
`A_UINT32`	`power_save_failure_cnt`	Number of failures that occurred when the AR6000 could not go to sleep
`A_UINT32`	`cs_bmiss_cnt`	Number of BMISS interrupts since connection
`A_UINT32`	`cs_lowRssi_cnt`	Number of the times the RSSI went below the low RSSI threshold
`A_UINT16`	`cs_connect_cnt`	Number of connection times
`A_UINT16`	`cs_disconnect_cnt`	Number of disconnection times
`A_UINT8`	`cs_aveBeacon_rssi`	The current averaged value of the RSSI from the beacons of the connected BSS
`A_UINT8`	`cs_lastRoam_msec`	Time that the last roaming took, in ms. This time is the difference between roaming start and actual connection.

Event Values `None defined`

Name	**ROAM_TBL_EVENT**		
Synopsis	Reports the roam table, which contains the current roam mode and this information for every BSS:		
Event ID	0x100F		
Event Parameters	`A_UINT8`	`bssid[ATH_MAC_LEN];`	BSSID
	`A_UINT8`	`rssi`	Averaged RSSI
	`A_UINT8`	`rssidt`	Change in RSSI
	`A_UINT8`	`last_rssi`	Last recorded RSSI
	`A_UINT8`	`roam_util`	Utility value used in roaming decision
	`A_UINT8`	`util`	Base utility with the BSS
	`A_UINT8`	`bias`	Host configured for this BSS
Event Values	`roamMode`		Current roam mode
		`= 1`	RSSI based roam
		`= 2`	Host bias-based roam
		`= 3`	Lock to the current BSS
		`= 4`	Autonomous roaming disabled

Name	**RSSI_THRESHOLD**		
Synopsis	Alerts the host that the signal strength from the connected AP has crossed a interesting threshold as defined in a previous **"RSSI_THRESHOLD_PARAMS"** command.		
Event ID	0x100C		
Event Parameters	`A_UINT8`	`range`	A WMI_RSSI_THRESHOLD_VAL value, which indicates the range of the average signal strength

Event Values

```
{
WMI_RSSI_LOWTHRESHOLD_BELOW_LOWERVAL   = 1
WMI_RSSI_LOWTHRESHOLD_LOWERVAL         = 2
WMI_RSSI_LOWTHRESHOLD_UPPERVAL         = 3
WMI_RSSI_HIGHTHRESHOLD_LOWERVAL        = 4
WMI_RSSI_HIGHTHRESHOLD_HIGHERVAL       = 5
} WMI_RSSI_THRESHOLD_VAL
```

Name	**SCAN_COMPLETE_EVENT**		
Synopsis	Indicates the scan status. if the Scan was not completed, this event is generated with the status A_ECANCELED. If the scan cannot start, this event is generated with status A_EBUSY. If the scan completes successfully, this event is generated with status A_OK.		
Event ID	0x100A		
Event Parameters	`A_UINT8`	`scanStatus`	
Event Values	`{`		
	`A_UINT8`	`scanStatus`	A_OK, A_ECANCELED, or A_EBUSY
	`} WMI_SCAN_COMPLETE_EVENT;`		

Name	**TEST_EVENT**		
Synopsis	The TCMD application uses a single WMI event (WMI_TEST_EVENTID) to communicate events from target to host. The events are parsed by the TCMD application and WMI layer is oblivious of it.		
Event ID	0x1016		
Event Parameters	`WMI_TEST_EVENTID`		
Event Values	`None`		

Name	**TKIP_MICERR**		
Synopsis	Indicates that TKIP MIC errors were detected.		
Event ID	0x1009		
Event Parameters	`A_UINT8`	`keyid`	Indicates the TKIP key ID
	`A_UINT8`	`ismcast`	■ 0 = Unicast ■ 1 = Multicast
Event Values	See event parameters		

WMI Extension Commands

The WMI EXTENSION command is used to multiplex a collection of commands that:

- Are not generic wireless commands
- May be implementation-specific
- May be target platform-specific
- May be optional for a host implementation

An extension command is sent to the AR6000 targets like any other WMI command message and uses the WMI_EXTENSION. The first field of the payload for this EXTENSION command is another commandId, sometimes called the subcommandId, which indicates which extension command is being used. A subcommandId-specific payload follows the subcommandId.

All extensions (subcommandIds) are listed in the header file **include/wmix.h**. See also "WMI Extension Events" on page A-69.

Table A-5 lists the WMI extension commands.

Table A-5. **WMI Extension Commands**

Event	Description	Page
GPIO_INPUT_GET	Read GPIO pins configured for input	page A-66
GPIO_INTR_ACK	Acknowledge and re-arm GPIO interrupts reported earlier	page A-67
GPIO_OUTPUT_SET	Manage output on GPIO pins configured for output	page A-67
GPIO_REGISTER_GET	Read an arbitrary GPIO register	page A-67
GPIO_REGISTER_SET	Dynamically change GPIO configuration	page A-68
SET_LQTHRESHOLD	Set link quality thresholds; the sampling happens at every unicast data frame Tx, if certain thresholds are met, and corresponding events are sent to the host	page A-68

Name	**GPIO_INPUT_GET**
Synopsis	Allows the host to read GPIO pins that are configured for input. The values read are returned through a **"GPIO_DATA"** extension event. **NOTE:** Support for GPIO is optional.
Command	N/A
Command Parameters	None
Reply Parameters	None
Reset Value	None
Restrictions	None

Name	**GPIO_INTR_ACK**
Synopsis	The host uses this command to acknowledge and to re-arm GPIO interrupts reported through an earlier **"GPIO_INTR"** extension event. A single **"GPIO_INTR_ACK"** command should be used to acknowledge all GPIO interrupts that the host knows to be outstanding (if pending interrupts are not acknowledged through **"GPIO_INTR_ACK"**, another **"GPIO_INTR"** extension event is raised). **NOTE:** Support for GPIO is optional.
Command	N/A
Command Parameters	`A_UINT32` `ack_mask` A mask of interrupting GPIO pins (e.g., ACK_MASK bit [3] acknowledges an interrupt from the pin GPIO3).
Command Values	None
Reset Value	None
Restrictions	The host should acknowledge only interrupts about which it was notified.

Name	**GPIO_OUTPUT_SET**		
Synopsis	Manages output on GPIO pins configured for output. Conflicts between set_mask and clear_mask or enable_mask and disable_mask result in undefined behavior. **NOTE:** Support for GPIO is optional.		
Command	N/A		
Command Parameters	`A_UINT32`	`set_mask`	Specifies which pins should drive a 1 out
	`A_UINT32`	`clear_mask`	Specifies which pins should drive a 0 out
	`A_UINT32`	`enable_mask`	Specifies which pins should be enabled for output
	`A_UINT32`	`disable_mask`	Specifies which pins should be disabled for output
Command Values	None		
Reset Value	None		
Restrictions	None		

Name	**GPIO_REGISTER_GET**		
Synopsis	Allows the host to read an arbitrary GPIO register. It is intended for use during bringup/debug. The target responds to this command with a **"GPIO_DATA"** event. **NOTE:** Support for GPIO is optional.		
Command	N/A		
Command Parameters	`A_UINT32`	`gpioreg_id`	Specifies a GPIO register identifier, as defined in **include/AR6000/AR6000_gpio.h**
Reply Parameters	None		
Reset Value	N/A		
Restrictions	None		

Name	**GPIO_REGISTER_SET**
Synopsis	Allows the host to dynamically change GPIO configuration (usually handled statically through the GPIO configuration DataSet). **NOTE:** Support for GPIO is optional.
Command	N/A

Command Parameters

`A_UINT32`	`gpioreg_id`	Specifies a GPIO register identifier, as defined in **include/AR6000/AR6000_gpio.h**
`A_UINT32`	`value`	Specifies a value to write to the specified GPIO register

Command Values	`None`
Reset Value	Initial hardware configuration is as defined in the *AR6003 ROCm^{TM} Single-Chip MAC/BB/Radio for 2.4/5 GHz Embedded WLAN Applications* data sheet. This configuration is modified by the GPIO Configuration DataSet, if one exists.
Restrictions	None

Name	**SET_LQTHRESHOLD**
Synopsis	Set link quality thresholds, the sampling happens at every unicast data frame Tx, if certain threshold is met, corresponding event will be sent to host.

Command

```
wmiconfig eth1 --lqThreshold <enable> <upper_threshold_1>...
<upper_threshold_4> <lower_threshold_1>... <lower_threshold_4>
```

Command Parameters

```
A_UINT8  enable;
A_UINT8  thresholdAbove1_Val;
A_UINT8  thresholdAbove2_Val;
A_UINT8  thresholdAbove3_Val;
A_UINT8  thresholdAbove4_Val;
A_UINT8  thresholdBelow1_Val;
A_UINT8  thresholdBelow2_Val;
A_UINT8  thresholdBelow3_Val;
A_UINT8  thresholdBelow4_Val;
```

Command Values

`enable`	`= 0`	Disable link quality sampling
	`= 1`	Enable link quality sampling
`thresholdAbove_Val`	`[1...4]`	Above thresholds (value in [0,100]), in ascending order threshold BELOW_VAL [1...4] = below thresholds (value in [0,100]), in ascending order

Reset Values	None
Restrictions	None

WMI Extension Events

The WMI EXTENSION event is used for a collection of events that:

- Are not generic wireless events
- May be implementation-specific
- May be target platform-specific
- May be optional for a host implementation

An extension event is sent from the AR6000 device targets to the host just like any other WMI event message, using the WMI_EXTENSION_EVENTID. The first field of the payload for this "EXTENSION" event is another commandId (sometimes called the subcommandId) that indicates which "EXTENSION" event is being used. A subcommandId-specific payload follows the subcommandId.

All extensions (subcommandIds) are listed in the header file **include/wmix.h**. See also "WMI Extension Commands" on page A-66.

Table A-6 lists the WMI extension events.

Table A-6. **WMI Extension Events**

Event	Description	Page
GPIO_ACK	Acknowledges a host set command has been processed by the device	page A-69
GPIO_DATA	Response to a host's request for data	page A-70
GPIO_INTR	Signals that GPIO interrupts are pending	page A-70

Name	GPIO_ACK
Synopsis	Acknowledges that a host set command (either **"GPIO_OUTPUT_SET"** or **"GPIO_REGISTER_SET"**) has been processed by the AR6000 device. **NOTE:** Support for GPIO is optional.
Event ID	N/A
Event Parameters	None
Event Values	None

Name	**GPIO_DATA**
Synopsis	The AR6000 device uses this event to respond to the host's earlier request for data (through either a **"GPIO_REGISTER_GET"** or a **"GPIO_INPUT_GET"** command). **NOTE:** Support for GPIO is optional.
Event ID	N/A

Event Parameters

`A_UINT32`	`value`	Holds the data of interest, which is either a register value (in the case of **"GPIO_REGISTER_GET"**) or a mask of pin inputs (in the case of **"GPIO_INPUT_GET"**).
`A_UINT32`	`reg_id`	Indicates which register was read (in the case of **"GPIO_REGISTER_GET"**) or is GPIO_ID_NONE (in the case of **"GPIO_INPUT_GET"**)

Event Values `None`

Name	**GPIO_INTR**
Synopsis	The AR6000 device raises this event to signal that GPIO interrupts are pending. These GPIOs may be interrupts that occurred after the last **"GPIO_INTR_ACK"** command was issued, or may be GPIO interrupts that the host failed to acknowledge in the last **"GPIO_INTR_ACK"**. The AR6000 will not raise another **GPIO_INTR** event until this event is acknowledged through a **"GPIO_INTR_ACK"** command. **NOTE:** Support for GPIO is optional.
Event ID	N/A

Event Parameters

`A_UINT32`	`intr_mask`	Indicates which GPIO interrupts are currently pending
`A_UINT32`	`input_values`	A recent copy of the GPIO input values, taken at the time the most recent GPIO interrupt was processed

Event Values `None`

SoftAP WMI Commands

Table A-6 lists the WMI extension events.

Table A-7. **WMI Extension Events**

Description	Page
Hide the SSID in the Beacon	page A-72
Get a List of Connected STAs	page A-73
Set the Maximum Number of Allowed STAs	page A-73
Set the ACL Policy	page A-74
Add a MAC Address to the ACL List	page A-75
Remove a MAC Address From the ACL List	page A-76
View a List of MAC Addresses on the ACL List	page A-76
Reconfigure the AP Profile Parameters	page A-77
Manage Connected STAs	page A-78
Set the STA Communication Timeout	page A-79
Retrieve the Firmware Version	page A-79
List all Valid Regulatory Country Codes	page A-80
Set the Regulatory Country Code	page A-80
Disable the Country IE in Beacons and Probe Responses	page A-81
Set the DTIM Period of the AP	page A-81
Set the Scan Time for 802.11b/g Protection	page A-82
Control Communication Between STAs	page A-82

Name	**Hide the SSID in the Beacon (Linux-specific)**
Synopsis	The host uses this command to set/reset the hidden SSID configuration of the AP. When this flag is set, the AR6000 AP will not advertise SSID in the beacon. The user must use the `iwconfig -i eth1 commit` command for the hidden SSID to take effect in the beacons.

Command	`wmiconfig -i eth1 --hiddenssid <value>`
	`iwconfig eth1 commit`

Command Parameters	`value - [0 reset / 1 set]`
Reply Parameters	None
IOCTL	`AR6000_XIOCTL_AP_HIDDEN_SSID`

IOCTL Parameters	`typedef struct {`
	`  A_UINT8hidden_ssid;`
	`} WMI_AP_HIDDEN_SSID_CMD;`

Reply Parameters	None
Reset Value	None
WMI	`WMI_AP_HIDDEN_SSID_CMDID`

WMI Parameters	`typedef struct {`
	`  A_UINT8hidden_ssid;`
	`} WMI_AP_HIDDEN_SSID_CMD;`

Reply Parameters	None
Reset Value	None
Restrictions	The AR6000 should be in AP mode; users must use the `commit` command for this command to take effect.

Name	**Get a List of Connected STAs**
Synopsis	The host uses this command to get a list of connected STAs.
Command	`wmiconfig -i eth1 --getsta`
Command Parameters	None
Reply Parameters	List of MAC addresses printed on the console
IOCTL	`AR6000_XIOCTL_AP_GET_STA_LIST`
IOCTL Parameters	see below
Reply Parameters	ap_get_sta_t
Reset Value	None
WMI	None
WMI Parameters	None
Reply Parameters	None
Reset Value	None
Restrictions	The AR6000 should be in AP mode

IOCTL Parameters:

```
ap_get_sta_t *
    typedef struct {
  A_UINT8 state;
  A_UINT8 mac[6];
  A_UINT8 aid;
} station_t;

typedef struct {
  station_t sta[4];
} ap_get_sta_t;
```

Name	**Set the Maximum Number of Allowed STAs**
Synopsis	The host uses this command to set the maximum number of STAs that can connect to the AR6000-AP.
Command	`wmiconfig -i eth1 --numsta <value>`
Command Parameters	value [1 to AP_MAX_NUM_STA(currently 4)]
Reply Parameters	None
IOCTL	`AR6000_XIOCTL_AP_SET_NUM_STA`
IOCTL Parameters	see below
Reply Parameters	None
Reset Value	None
WMI	`WMI_AP_SET_NUM_STA_CMDID`
WMI Parameters	see below
Reply Parameters	None
Reset Value	AP_MAX_NUM_STA
Restrictions	The AR6000 should be in AP mode

IOCTL Parameters:

```
typedef struct {
  A_UINT8num_sta;
} WMI_AP_SET_NUM_STA_CMD;
```

WMI Parameters:

```
typedef struct {
  A_UINT8num_sta;
} WMI_AP_SET_NUM_STA_CMD;
```

Name **Set the ACL Policy**

Synopsis The host uses this command to set policy for ACL. Depending on the policy STAs in the ACL list will be either allowed or denied to connect. If ACL policy is disabled, any STA can connect to AP and ACL list will not be used in this case.

Command **`wmiconfig -i eth1 --aclpolicy <policy> <retain>`**

Command Parameters

```
<Policy> 0 - Disable
     1 - Allow MAC in the ACL list
     2 - Deny MAC in the ACL list
<Retain> 0 - Clear the existing ACL list
     1 - Retain the existing ACL list
```

Reply Parameters None

IOCTL **`AR6000_XIOCTL_AP_SET_ACL_POLICY`**

IOCTL Parameters

```
#define AP_ACL_DISABLE     0x00
#define AP_ACL_ALLOW_MAC    0x01
#define AP_ACL_DENY_MAC     0x02
#define AP_ACL_RETAIN_LIST_MASK 0x80

typedef PREPACK struct {
  A_UINT8   policy;
} POSTPACK WMI_AP_ACL_POLICY_CMD;
```

Reply Parameters None

Reset Value

```
Policy = 0
Retain = 0
```

WMI **`WMI_AP_ACL_POLICY_CMDID`**

WMI Parameters

```
#define AP_ACL_DISABLE     0x00
#define AP_ACL_ALLOW_MAC    0x01
#define AP_ACL_DENY_MAC     0x02
#define AP_ACL_RETAIN_LIST_MASK 0x80

typedef PREPACK struct {
  A_UINT8   policy;
} POSTPACK WMI_AP_ACL_POLICY_CMD;
```

Reply Parameters None

Reset Value

```
Policy = 0
Retain = 0
```

Restrictions The AR6000 should be in AP mode

Name	**Add a MAC Address to the ACL List**
Synopsis	The host uses this command to add a MAC address (with wildcard support) to the ACL list. When a MAC address is added to this list, depending on the ACL policy, AP will either allow or deny the STA to connect.
Command	**`wmiconfig -i eth1 --addacl <mac>`**

Command Parameters	`A_UINT8 mac[] [Format: aa:bb:cc:dd:ee:ff]`
	`E.g. 00:11:22:33:44:55`
	`00:11:22:*:*:*`
	`00:11:22:33:44:*`
	`00:11:*:33:*:55`

Reply Parameters	None
IOCTL	**`AR6000_XIOCTL_AP_SET_ACL_MAC`**

IOCTL Parameters	`typedef struct {`
	`A_UINT8action; (=ADD_MAC_ADDR)`
	`A_UINT8index;`
	`A_UINT8 mac[ATH_MAC_LEN];`
	`} WMI_AP_ACL_MAC_CMD;`

Reply Parameters	None
Reset Value	None
WMI	**`WMI_AP_ACL_MAC_LIST_CMDID`**

WMI Parameters	`typedef struct {`
	`A_UINT8action; (=ADD_MAC_ADDR)`
	`A_UINT8index; (=location at which mac will be stored)`
	`A_UINT8 mac[ATH_MAC_LEN];`
	`} WMI_AP_ACL_MAC_CMD;`

Reply Parameters	None
Reset Value	None
Restrictions	The AR6000 should be in AP mode

Name	**Remove a MAC Address From the ACL List**
Synopsis	The host uses this command to remove already added MAC address from ACL list. When a last MAC address is removed from list, the AR6000 AP will allow any STA to connect (because list is empty) irrespective of the policy.
Command	**`wmiconfig -i eth1 --delacl <index>`**
Command Parameters	`<index> get index using --getacl cmd`
Reply Parameters	None
IOCTL	**`AR6000_XIOCTL_AP_SET_ACL_MAC`**

IOCTL Parameters	`typedef struct {`
	`  A_UINT8action; (=DEL_MAC_ADDR)`
	`  A_UINT8index;`
	`  A_UINT8    mac[ATH_MAC_LEN];`
	`} WMI_AP_ACL_MAC_CMD;`

Reply Parameters	None
Reset Value	None
WMI	**`WMI_AP_ACL_MAC_LIST_CMDID`**

WMI Parameters	`typedef struct {`
	`  A_UINT8action; (=DEL_MAC_ADDR)`
	`  A_UINT8index; (=location from which mac removed)`
	`  A_UINT8    mac[ATH_MAC_LEN];`
	`} WMI_AP_ACL_MAC_CMD;`

Reply Parameters	None
Reset Value	None
Restrictions	The AR6000 should be in AP mode

Name	**View a List of MAC Addresses on the ACL List**
Synopsis	The host uses this command to view ACL policy and ACL list.
Command	**`wmiconfig --getacl`**
Command Parameters	None
Reply Parameters	None
IOCTL	**`AR6000_XIOCTL_AP_GET_ACL_LIST`**
IOCTL Parameters	None

Reply Parameters	`typedef struct {`
	`  A_UINT8 index;`
	`  A_UINT8 acl_mac[AP_MAX_NUM_STA][ATH_MAC_LEN];`
	`  A_UINT8 policy;`
	`} WMI_AP_ACL;`

Reset Value	None
WMI	**None**
WMI Parameters	None
Reply Parameters	None
Reset Value	None
Restrictions	The AR6000 should be in AP mode

Name **Reconfigure the AP Profile Parameters**

Synopsis The host uses this command to indicate the firmware to update its profile configuration (SSID, hiddenSSID, channel, security settings and so on). These parameters are modified through individual commands, and these values will take effort only after commit command.

Command

```
wmiconfig --commit (or)
iwconfig eth1 commit
```

Command Parameters None

Reply Parameters None

IOCTL `AR6000_XIOCTL_AP_COMMIT_CONFIG`

IOCTL Parameters None

Reply Parameters None

Reset Value None

WMI `WMI_AP_CONFIG_COMMIT_CMDID`

WMI Parameters

```
typedef struct {
  A_UINT8   networkType;
  A_UINT8   dot11AuthMode;
  A_UINT8   authMode;
  A_UINT8   pairwiseCryptoType;
  A_UINT8   pairwiseCryptoLen;
  A_UINT8   groupCryptoType;
  A_UINT8   groupCryptoLen;
  A_UINT8   ssidLength;
  A_UCHAR   ssid[WMI_MAX_SSID_LEN];
  A_UINT16  channel;
  A_UINT8   bssid[ATH_MAC_LEN];
  A_UINT32  ctrl_flags;
} WMI_CONNECT_CMD;
```

Reply Parameters None

Reset Value None

Restrictions The AR6000 should be in AP mode

Name **Manage Connected STAs**

Synopsis The hostapd uses this command to manage connected STAs. It shall DEAUTH / DISASSOC / AUTHORIZE / UNAUTHORIZE a connected STA.

Command `wmiconfig --removesta <action> <reason> <mac>`

Command Parameters

```
<action> 2 - Disassoc (or) 3 - Deauth
<reason> protocol reason code, use 1 when in doubt
<mac> mac addr of a connected STA
```

Reply Parameters None

IOCTL `IEEE80211_IOCTL_SETMLME`

IOCTL Parameters

```
#define IEEE80211_MLME_DISASSOC  2
#define IEEE80211_MLME_DEAUTH   3
#define IEEE80211_MLME_AUTHORIZE  4
#define IEEE80211_MLME_UNAUTHORIZE 5

struct ieee80211req_mlme {
u_int8_tim_op;
u_int16_tim_reason;/* 802.11 reason code */
 u_int8_tim_macaddr[IEEE80211_ADDR_LEN];
}
```

Reply Parameters None

Reset Value None

WMI `WMI_AP_SET_MLME_CMDID`

WMI Parameters

```
#define WMI_AP_DISASSOC     2
#define WMI_AP_DEAUTH      3
#define WMI_AP_MLME_AUTHORIZE  4
#define WMI_AP_MLME_UNAUTHORIZE 5

typedef struct {
  A_UINT8  mac[ATH_MAC_LEN];
  A_UINT16  reason; /* 802.11 reason code */
  A_UINT8  cmd;  /* operation to perform */
} WMI_AP_SET_MLME_CMD;
```

Reply Parameters None

Reset Value None

Restrictions The AR6000 should be in AP mode

Name **Set the STA Communication Timeout**

Synopsis The host uses this command to set STA communication timeout value in minutes. If there is no data activity (tx and rx) for timeout period, that STA will be removed by sending deauth and removing entry from the connection table.

Command `wmiconfig --conninact <value>`

Command Parameters `value [time in minutes, default is 5 min]`

Reply Parameters None

IOCTL `AR6000_XIOCTL_AP_CONN_INACT_TIME`

IOCTL Parameters

```
typedef struct {
  A_UINT32 period;
} WMI_AP_CONN_INACT_CMD;
```

Reply Parameters `None`

Reset Value `None`

WMI `WMI_AP_CONN_INACT_CMDID`

WMI Parameters

```
typedef struct {
  A_UINT32 period;
} WMI_AP_CONN_INACT_CMD;
```

Reply Parameters None

Reset Value `5 minutes`

Restrictions The AR6000 should be in AP mode

Name **Retrieve the Firmware Version**

Synopsis The host uses this command to retrieve firmware version. The new event is posted to the host after WMI_READY_EVENT.

Command `wmiconfig --version`

Command Parameters `None`

Reply Parameters None

IOCTL `AR6000_IOCTL_WMI_GETREV`

IOCTL Parameters `None`

Reply Parameters

```
struct ar6000_version {
   A_UINT32  host_ver;
   A_UINT32  target_ver;
   A_UINT32  wlan_ver;
};
```

Reset Value `None`

WMI `WMI_WLAN_VERSION_EVENTID`

WMI Parameters

```
typedef struct {
  A_UINT32 version;
} WMI_WLAN_VERSION_EVENT;
```

Reply Parameters None

Reset Value `None`

Restrictions None

Name	**List all Valid Regulatory Country Codes**
Synopsis	This command is used to get the list of all valid country codes. These codes can be used to set the regulatory domain for the AP.
Command	`wmiconfig --countrycodes`
Command Parameters	`None`
Reply Parameters	List of countryCodes printed on the console.
IOCTL	`None`
IOCTL Parameters	`None`
Reply Parameters	`None`
Reset Value	`None`
WMI	`None`
WMI Parameters	`None`
Reply Parameters	None
Reset Value	`None`
Restrictions	None

Name	**Set the Regulatory Country Code**
Synopsis	This command is used to set the Country code for setting the AP in a specific regulatory domain.
Command	`wmiconfig --setcountry <countrycode>`
Command Parameters	`A valid countrycode from the list got using the` **`wmiconfig -countrycodes`** `command`
Reply Parameters	None
IOCTL	`AR6000_XIOCTL_AP_SET_COUNTRY`
IOCTL Parameters	`typedef struct {` `A_UCHAR countryCode[3];` `} WMI_AP_SET_COUNTRY_CMD;`
Reply Parameters	`None`
Reset Value	`None`
WMI	`WMI_AP_SET_COUNTRY_CMDID`
WMI Parameters	`typedef struct {` `A_UINT32 countryCode[3];` `} WMI_AP_SET_COUNTRY_CMD;`
Reply Parameters	None
Reset Value	Country: US; the AP always comes up in US Regulatory domain at startup
Restrictions	None

Name	**Disable the Country IE in Beacons and Probe Responses**
Synopsis	This command is used to disable sending the Country IE in beacons and probe responses.
Command	`wmiconfig --disableregulatory`
Command Parameters	None
Reply Parameters	None
IOCTL	`AR6000_XIOCTL_AP_SET_COUNTRY`

IOCTL Parameters	`typedef struct {`
	`  A_UCHAR countryCode[3];`
	`} WMI_AP_SET_COUNTRY_CMD;`
	The special country code FF is used to disable sending the Country IE in the beacons/Probe Rsps.

Reply Parameters	None
Reset Value	None
WMI	`WMI_AP_SET_COUNTRY_CMDID`

WMI Parameters	`typedef struct {`
	`  A_UINT32 countryCode[3];`
	`} WMI_AP_SET_COUNTRY_CMD;`
	`Special country code of "FF" is used to disable sending the Country IE in the beacons/Probe Rsps.`

Reply Parameters	None
Reset Value	Country: US; the AP always comes up in US Regulatory domain at startup
Restrictions	None

Name	**Set the DTIM Period of the AP**
Synopsis	This command is used to configure the DTIM period for the AP.
Command	`wmiconfig --dtim <period>`
Command Parameters	`<period> is a value between 1-10 (both inclusive) that can be configured as DTIM interval for the AP.`
Reply Parameters	None
IOCTL	`AR6000_XIOCTL_AP_SET_DTIM`

IOCTL Parameters	`typedef struct {`
	`  A_UINT8 dtim;`
	`} WMI_AP_SET_DTIM_CMD;`

Reply Parameters	None
Reset Value	None
WMI	`WMI_AP_SET_DTIM_CMDID`

WMI Parameters	`typedef struct {`
	`  A_UINT8 dtim;`
	`} WMI_AP_SET_DTIM_CMD;`

Reply Parameters	None
Reset Value	5; on reset, the AP always comes up with default DTIM period of 5 beacons intervals
Restrictions	None

Name	**Set the Scan Time for 802.11b/g Protection**
Synopsis	The host uses this command to set the 802.11b/g mode protection timers. When this timer is enabled, AP receives beacon from other APs in the same channel and enable/disable protection in the beacon.
Command	`wmiconfig --protectionscan <period> <dwell>`

Command Parameters	`<period> Time in min, at which AP does scan. Default is 5 min.`
	`<dwell> Time in ms, duration of scan. Default is 200 ms.`

Reply Parameters	None
IOCTL	`AR6000_XIOCTL_AP_PROT_SCAN_TIME`

IOCTL Parameters	`typedef struct {`
	`  A_UINT32 period_min;`
	`  A_UINT32 dwell_ms;`
	`} WMI_AP_PROT_SCAN_TIME_CMD;`

Reply Parameters	`None`
Reset Value	`None`
WMI	`WMI_AP_PROT_SCAN_TIME_CMDID`

WMI Parameters	`typedef struct {`
	`  A_UINT32 period_min;`
	`  A_UINT32 dwell_ms;`
	`} WMI_AP_PROT_SCAN_TIME_CMD;`

Reply Parameters	None
Reset Value	5 minutes and 200 ms
Restrictions	The AR6000 should be in AP mode

Name	**Control Communication Between STAs**
Synopsis	The host uses this command to enable/disable communication between connected STAs. When it is enabled, any connected STA will be able to communicate with any other connected STA (connected to same AP). If it is disabled, STA will be able to communicate with only AP and can not communicate with any other STA.
Command	`wmiconfig -i eth1 --intrabss <control>`

Command Parameters	`<control> 0 - Disable`
	`     1 - Enable`

Reply Parameters	None
IOCTL	`AR6000_XIOCTL_AP_INTRA_BSS_COMM`
IOCTL Parameters	`A_UINT8 control`
Reply Parameters	`None`
Reset Value	`Control = 1`
WMI	`None`
WMI Parameters	`None`
Reply Parameters	None
Reset Value	None
Restrictions	The AR6000 should be in AP mode

B Messaging and Communications

Bootloader Message Interface (BMI)

During initialization, the host has an opportunity to use the bootloader message interface (BMI) services that allow the host to get target revision information, download (write) code and data into specific locations in target memory, upload (read) code and data from target memory, read/write target registers, and execute at a specific target address. The bmiloader tool serves as an application that offers a command line interface to BMI messages.

BMI Requests

Every BMI request starts with a 32-bit command followed by command-specific arguments. The final BMI command (which the host must send) is a BMI_DONE message. If the host does not wish to use BMI, it sends the BMI_DONE command on mailbox 0 and proceeds with HTC initialization.

BMI does not use HTC for message transport, but uses a simpler subset of HTC transport where all BMI requests and responses occur over mailbox 0. Mailbox addressing and EOM signaling are identical to HTC. For flow control, BMI uses a single command credit implemented with credit counter #4. Initially, the target provides a single BMI command credit to the host. When the host is ready to send a BMI command, it decrements the command credit counter using the device's COUNT_DEC register, then sends a single BMI command. Once the target processes the command, the host can send the next BMI command to the target. BMI messages are not encapsulated or padded[1] and are sent directly over mailbox 0. BMI requires no configuration. Generally the first host-issued BMI command to the target is BMI_GET_TARGET_ID, which returns a structure including a four-byte identifier that may be used to select further actions. The reference host device driver always issues this command as it enters the BMI phase of startup.

[1] The bmiloader command pads all BMI messages to a multiple of 4 bytes, facilitating use of BMI with host controllers that have more stringent requirements.

BMI_DONE

In all cases, the final BMI command the host issues must be BMI_DONE. This command ends the BMI phase and causes the AR6000 target to begin executing the target application (usually the wireless LAN application athwlan). A user or script can explicitly send a BMI_DONE using bmiloader --done in the reference host software.

Before the BMI_DONE command has been issued, the target processes whatever BMI commands the host issues. These BMI commands can read/ write memory or registers in the AR6000. They can also load a target application into memory and immediately execute it. The BMI_SET_APP_START command changes which application is executed when BMI_DONE is eventually issued. For example, BMI is used to load the target flash program into target RAM and then to cause it to execute rather than the default athwlan application. Details, including message command constants, are in the BMI header file, **include/bmi_msg.h**.

Host Interest Area

See **include/targaddrs.h** for information on the host interest area. This macro references the relevant structure (e.g., AR6003_HOST_INTEREST_ITEM_OFFSET(item) for the AR6003).

The item is a pointer into a list of items which must remain constant across firmware revisions. The value returned from the macro is a target-specific memory address to set/get the value using a BMI memory read/write operation. Each item is an A_UINT32 value.

Host/Target Message Interface

The *AR6003 ROCm*™ *Single-Chip MAC/BB/Radio for 2.4/5 GHz Embedded WLAN Applications* data sheets provide details vital to the understanding of the host/target communications (HTC) message transport. Source code for HTC is in **host/htc2/**. HTC is the message transport used by WMI to send and receive control and data messages across the interconnect that connect the AR6000 device target and the host system.

Host Target Communications

HTC 2.1 Protocol

The HTC 2.1 protocol provides an abstract bi-directional message transfer protocol tailored for AR6000 devices operating over serial interconnects. Host-target communication (HTC) is packet-based and designed to minimize register access overhead in high-latency interconnects.

The packet-I/O nature of these serial links creates a number of challenges in software development. Some of them are enumerated here:

- Serial controllers are rarely ever standardized. To effectively manage software re-use, most architectures employ a layered I/O model. Layers require more software interaction and more complexity.
- OSs that support standard interconnects utilize abstract bus models. This requires host software to follow mandated I/O processing rules. For example, I/O requests may require interrupts (where applicable) as to not stall the CPU in a polling loop, which is typically not controllable. These interrupts require context switching that incurs overhead.
- Abstract bus models require more setup operations when preparing each I/O request. I/O request management (serialization, request routing, and parameter checking/setting) adds to the overhead. I/O request completion involves context switches and callback functions.

NOTE: In this document:
Send is defined as host-to-target transfers
Receive is defined as target-to-host transfers
Tx is defined as target-to-WLAN Media transfers
Rx is defined as WLAN media-to-target transfers

HTC addresses these challenges by implementing a protocol that takes into account high bus transaction latencies. "HTC 2.1 Overview" on page B-4 provides an overview of key points in the HTC 2.1 protocol.

HTC 2.1 Overview

HTC Single Mailbox Design

HTC operates over a single hardware mailbox. The serial I/O interconnects do not offer I/O concurrency or bus mastering capabilities that can take advantage of a multi-mailbox design. A single mailbox design is adequate to manage the flow of messages.

HTC Endpoints and Services

All messages are tagged with an endpoint ID. This provides a simple message MUX/de-MUX scheme through the physical transport (a single mailbox). An endpoint ID has no implied meaning. It is merely a pipe to send/receive reliable, flow-controlled and packetized (non-fragmented) messages. Endpoint 0 however, is reserved for HTC control and configuration. The system ultimately associates an endpoint to a service and application-specific data flows through the endpoints between clients (host) and servers (target).

An HTC service is an abstract concept to logically partition client-server protocols. The host views the target as a collection of services (or servers). To communicate with a service, the host needs an endpoint to facilitate message transfer.

To illustrate this partitioning, the WLAN application can be divided into a set of services:

- WMI control (WMI command/response/event message protocol)
- Network Data Best-Effort (pseudo-Ethernet frames)
- Network Data Background (pseudo-Ethernet frames)
- Network Data Voice (pseudo-Ethernet frames)
- Network Data Video (pseudo-Ethernet frames)

Services are not entirely disjoint, relationships can exist between services. For example, the WMI control service provides connection services to the WLAN media. The other four data services rely on the WMI service to prepare the WLAN media for data transmission (i.e AP connection, authentication etc.).

HTC Flow Control and Credit Rebalancing

HTC flow control is intended for transmit only. This design leverages the fact that the serial interconnects are non-busmastering in nature, as the target cannot push data directly into awaiting host buffers. A host can decide when to fetch pending Rx data from the target. An out-of-band technique such as masking interrupts until host buffers are available should be sufficient.

HTC incorporates in-band flow control to prevent the necessity of hardware credit counters, as realized through several mechanisms such as HTC credit messages, HTC message flags (signaling credit updates), and HTC message trailers (present in Rx messages) that can contain credit updates. Use of block-based I/O (in most serial interconnects) requires some amount of message padding. Wasted padding in the target-to-host path can convey more state information (such as credits) when possible.

A single mailbox design allows credits distribute virtually and dynamically. The ability to dynamically add more credits to active endpoints improves Tx performance by allowing the host to queue more data traffic. This pipelining effect helps when bursting data in protocols such as TCP/IP. Credit balancing decisions can be made on the host under full control of the developer.

The credit balancing concept can be as simple as the WLAN application requires. The host distributes initial credits to each endpoint at startup. When traffic patterns change, the host can redistribute credits between endpoints without target interaction or state changes. The target returns credits completed on a particular endpoint. The host collects these credits and bases distribution decisions on current traffic flows or customer-defined priorities. The credit return policy (target-to-host) creates a closed loop resource-reservation scheme for each traffic class (which now maps to an endpoint), allowing HTC to better model the WLAN MAC hardware queue backflow. A default credit distribution algorithm (for most WLAN applications) is provided in host reference software at **host\miscdrv\credit_dist.c**.

HTC Packet Bundling

The HTC protocol supports bundling multiple application level messages (wrapped in HTC frames) into a single host interconnect (HIF) transfer request to improve bandwidth by amortizing the OS to stack HIF overhead over multiple HTC messages. HTC bundling is an optimization to improve throughput when WLAN employs over-the-air aggregation. It has protocol features and procedures that facilitate bundling HTC frames over the host interconnect, extending to the HIF layer and requiring HIF cooperation to optimize bundled transfers. Bundling may include scatter-gather DMA support to map the bundle transfer over a set of discrete message buffers.

A bundle is a collection of individual HTC message buffers (send or receive) that can be submitted to the host interconnect layer as a single bus operation. A bundle consists of HTC messages directed at the same endpoint, however, each message may differ in size. The application determines the maximum number of HTC messages that can be handled as bundles with restrictions (i.e., maximum mailbox size) imposed by the HIF. HTC bundling mechanisms are defined differently for the send and receive paths.

Bundling in the Receive Path

An HTC bundle in the receive path is indicated by a Bundle lookahead record in the trailer space of the previously received HTC packet. Using this enhanced lookahead information, the host can fetch the next *N* packets with one HIF layer bus request. The bundle lookahead contains the lookahead information for each HTC packet in FIFO order. The host knows the full length and the destination endpoint of each packet and can optimally fetch the packet in a single bus transaction. The sequence for packet reception is:

1. The host is interrupted and fetches the current pending HTC message.
2. The host parses the HTC trailer and discovers a RX Bundle Look Ahead record. The record contains the HTC lookaheads of the next *N* packets behind the current packet.
3. The host can use the size and endpoint information to assemble *N* packet buffers to use in a single bus transaction to fetch all messages. If the host does not have enough packet buffers to extract this bundle, the host can fetch a smaller number of HTC messages in a bundle or individually.

The receive path bundling requires a starter HTC message in order to embed the special RX bundle lookahead record.

In high performance scenarios the application level messages in a bundle may all be the same size (that is, the maximum WLAN MTU size). Because the padded lengths of all these messages are identical, the bundle lookahead record can be omitted and replaced by a more optimal indication in the HTC header. Starter message HTC header flags indicate the number of messages behind the starter message identical in length to the starter message. The host can then retrieve the entire bundle in a single bus operation, knowing that the entire transfer is some multiple of the starter message length.

Bundling in the Transmit Path

HTC transmit bundling issues HTC messages back-to-back in a single host-interconnect bus transaction. Some interconnects may limit the amount of bundling due to the limitation on the total addressable register space (such as SPI and different chip versions).

HTC send bundling is entirely driven by the host in an opportunistic manner. Packets from the network stack can be classified and aggregated together (OS-specific) providing opportunities for packet bundling. The host collects a series of messages and assembles the messages back-to-back for use in a single HIF bus request. The host indicates to the target that an HTC message is part of a bundle by setting the HTC_MSG_BUNDLE flag in the flags field of the HTC header. The host must guarantee that each HTC message consumes a whole-number of credits. A message may not share a partial credit (straddling) with the next message in the bundle. The send bundling protocol allows the host to insert padding between messages to meet this requirement.

When HTC_MSG_BUNDLE is set, control byte 0 of the header indicates whether 0–255 bytes of padding were added to round up the message to the nearest credit. The PayloadLen field still contains the length of the application message, but when traversing a list of packets, the credit size padding is used to validate the packet.

HTC Protocol

All headers and data formats assume little-Endian ordering with first field to the left representing the first byte received or sent.

HTC Frame Header Format

Table B-1. **6-byte header with 4-byte lookahead limit**

Endpoint ID	Flags	Length	Control Byte 0	Control Byte 1	Payload and Trailer
1 byte	1 byte	2 bytes	1 byte	1 byte	N bytes

Table B-2. **Byte Fields Descriptions**

Field	Description
Endpoint ID	The endpoint (0 – 255) that the HTC message is sent/received on.
Flags	Transfer and target-state change flags, some flags are dependent on the direction of transfer (Send, Receive or Both)
Length	Length of payload in bytes (does not include HTC header length). In the Rx direction, the length may include the length of any additional trailer data (see flags below)
Control Bytes	Additional information based on the flag that is set. Control bytes can be used to extend the HTC header.

Table B-3. **Flags and Option Bytes**

Bit	Direction	Flag	Description
0	Send	NEED_CREDIT_UPDATE	Host is out of credits on this endpoint, target needs to send a credit report if the target has more than 1 credit available for the endpoint.
1	Receive	RECV_TRAILER_PRESENT	The HTC message contains a trailer with additional information. The number of bytes in the trailer is stored in Control Byte 0. Thus the application message is = (Length – Control Byte 0).
	Send	HTC_MSG_BUNDLE	Part of an HTC message bundle issued in the send direction. Control Byte 0 indicates the amount of padding (if any) applied to message in order to round the message up to the nearest credit.
7:4	Receive	RECV_BUNDLE_CNT	A non-zero entry in this field indicates the number of messages behind the current one that are the same block-padded size. The host can fetch 1 + (RECV_BUNDLE_CNT) number of frames (up to 16 total).

HTC Control Messages

HTC control messages are exchanged on endpoint 0. In the send direction, HTC control messages utilize the same flow control mechanism as any other endpoint. Because control messages in the send direction are used at initialization, the host is free to re-distribute endpoint 0 credits to application endpoints once the target and host are fully initialized.

All control messages are treated like ordinary HTC messages (using the same HTC header). The payload of these HTC messages use the control message formats in this document.

Each control message is preceded by a 2-byte MESSAGE ID as shown in Table B-4.

Table B-4. **2-byte Message ID**

Message ID	Message Payload
2 bytes	n-bytes

HTC Ready (Message ID = 1)

This message is issued to the host when the target system comes online. The host must wait for the arrival of this message before issuing any commands to the target. For version 2.0, the message size is 5 bytes. For version 2.1, the message size is 7 bytes. For both, the last byte is a version number. The host can distinguish between message types based on received message size. See Table B-5.

Table B-5. **Credit Size**

Credit Count	Credit Size	Max Endpoints	HTC Version	Max Messages Per Bundle
2 bytes	2 bytes	1 byte	1 byte	1 byte

Table B-6. **Ready Message Field Descriptions**

Field	Description
Credit Count	Total number of credits the target can offer (i.e., the number of buffers)
Credit Size	The number of bytes that each credit represents.
Max Endpoints	The maximum number of endpoint connections that the target can allow. This count includes endpoint 0.
HTC Version	Version 2.1
Max Messages Per Bundle	The largest number of messages supported in a bundle in both send and receive directions (typically 16).

Connect Service (Message ID = 2)

This message is issued to the target when the host connects to a service. The host must wait for a service connection response. The host should not issue more than one outstanding connection request.

Table B-7. **Connect Message Format**

Service ID	Flags	Service Metadata Length	Metadata
2 bytes	2 bytes	1 byte	0–128 bytes

Table B-8. **Connect Message Field Descriptions**

Field	Description	
Service ID	The unique ID assigned to the target service.	
Flags	Connection Flags (16-bit WORD): The target can reduce credit dribbling on a per-connection basis. The threshold to return credits when the host is empty is programmable to one of four levels.	
	bit [2]	REDUCE_CREDIT_DRIBBLE
	bits [1:0]	CREDIT_THRESHOLD_VALUE
	0x00	¼ outstanding credits
	0x01	½ outstanding credits
	0x02	¾ outstanding credits
	0x03	All outstanding credits
Metadata Length	The length of the optional service-specific metadata. Service metadata can contain private service initialization parameters. A service can choose not to use metadata exchange at all. The intent of providing metadata space is to create a simple mechanism to exchange parameters. More complex configuration should use service-specific control messages. The maximum length of metadata is 128 bytes.	

Service Connect Response (Message ID = 3)

This message is issued to the host in response to a connect request.

Table B-9. **Connect Response Message Format**

Service ID	Status	Endpoint ID	Service Metadata Length	Metadata
2 bytes	1 byte	1 byte	1 byte	0–128 bytes

Table B-10. **Connect Response Field Descriptions**

Field	Description	
Status	The connect status. These values are used:	
	0	Success
	1	Service not found
	2	Service connect failed (service connection was failed by the service itself)
	3	No resources (i.e. no more endpoints)
Service ID	The service ID of the connect request that caused this response.	
Endpoint ID	Newly assigned endpoint. The host must use this ID for all message exchanges for this service.	
Metadata Length	The length of the optional service-specific metadata. The response can contain private service configuration information that the host-side client may be interested in. A service can choose not to use the metadata exchange mechanism. The maximum length is 128 bytes	

HTC Setup Complete (Message ID = 4)

This message is issued to the target to indicate that the host has completed all service connection requests and is now ready for application-specific data flow. This message has no additional message bytes.

HTC Ready (Message ID = 5)

This message is issued to the target to indicate that the host has completed all service connection requests and is now ready for application-specific data flow. This message can only be issued to a target that supports v2.1 or higher. See Table B-11.

Table B-11. **HTC Setup Flags**

HTC Setup Flags	MaxMsgPerBundledRecv	Reserved Bytes
4 bytes	1 byte	4 bytes

Table B-12. **Ready Message Field Descriptions**

Field	Description
HTC Setup Flags	32-bit operating flags Bit [0] set to 1: Enables HTC receive packet bundling
MaxMsgPerBundledRecv	Maximum HTC messages per receive bundle
Reserved Bytes	Reserved bytes; must be set to zero

HTC Trailer Data

HTC packets can be received with additional protocol information such as credit and lookahead records. This data arrives at the end of the HTC message frame when space is available. This section describes credit, Rx lookahead, and Rx bundle record formats.

A credit report message is issued by the target when the target has at least one credit available and the host has previously signaled that it needs a credit update. Credit update policies and signaling mechanisms are detailed in "HTC Endpoint Data Flow" on page B-13. Credit reports can arrive on endpoint 0 as a zero-length HTC message with a trailer or received via any endpoint's Rx message (contained in the trailer). The later is an opportunistic mechanism to piggy-back state information through the Rx path. Trailer data on the Rx path takes advantage of block-level padding. A system designer could force padding at the message level so trailer space is always present.

Table B-13. **HTC Message Format with Trailer Data**

HTC Header	HTC Message	Receive Trailer Data
6 bytes	0–n bytes	0–255 bytes

The HTC header must have the RECV_TRAILER_PRESENT flag set and the control byte 0 parameter must reflect the total length of the trailer. The Rx trailer data can contain a variable number of variable-length records. Each record has the format:

Table B-14. **Rx Trailer Data Format**

Record Type	Length	Data
1 byte	1 byte	0–255 bytes

Table B-15. **Rx Trailer Data**

Field	Description	
Record Type	These record types are valid:	
	0	Reserved
	1	Credit report
	2	Rx packet lookahead
	3	Rx bundle lookahead
Length	Length of record data (does not include type or length fields)	

Credit Report Record

The credit report contains a variable number of endpoint credit pair structures with the format:

Table B-16. **Credit Report Record**

Endpoint ID	Credits
1 byte	1 byte

Table B-17. **Credit Report Record Descriptions**

Field	Description
Endpoint ID	The endpoint with available credits
Credits	New credits since the last credit report

Receive Packet lookahead Record

Table B-18. **Rx Packet lookahead Record**

Valid Pre Tag	Look Ahead	Valid Post Tag
1 byte	4 Bytes	1 bytes

Table B-19. **Rx Packet lookahead Description**

Field	Description
Valid Pre/Post Tags	This field combined with the valid post tag indicates the Lookahead is valid. The pre tag is the logical inverse of the post tag. The host can simply check that pre = ~post. If true, the lookahead is valid.
Look Ahead	The first four bytes of the next HTC frame header.

Receive Bundle lookahead Record

Table B-20. **Rx Packet lookahead Record**

Look Ahead 0...N[1]
4 + ... *N* x 4 Bytes

[1] The bundle record contains *N* number of 4-byte lookahead entries. The total number of entries is equal to the record data length divided by 4. Each HTC frame in the bundle has one lookahead entry.

Table B-21. **Rx Packet lookahead Record**

Field	Description
lookahead	The first four bytes of each HTC header; the record may contain one or more entries (maximum of 16).

HTC Service Connection Flow

At initialization, the host is required to wait for the system ready message on endpoint 0. The system ready message is an indicator that the target services are ready and the HTC communications link is fully initialized. The host can then proceed to issue service connection requests for each of the services of interest. The types of services are application-specific.

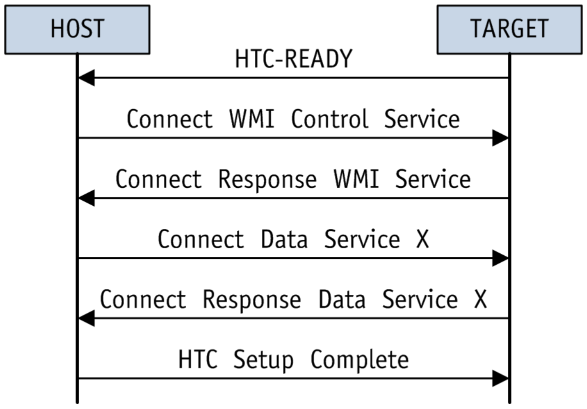
HOST
TARGET
HTC-READY
Connect WMI Control Service
Connect Response WMI Service
Connect Data Service X
Connect Response Data Service X
HTC Setup Complete

Figure B-1. **HTC Service Connection Flow**

The connect service message includes a service ID, a unique identifier initially defined by the service author. The connection request provides space for service specific metadata. This data is of limited size (up to 128 bytes) and can be used to convey configuration parameters, version information etc. Services are not required to use the metadata exchange mechanism. It is offered as a convenience for very simple configurations.

Limitations
The target firmware does not provide a disconnect mechanism. Once service connection is established it cannot be disconnected without a target reboot.

Once the service connection phase is complete, the host issues a HTC setup complete message and no longer issues connection requests. The target can act on this message to reconfigure resources if some services are not in use. At this point, the service connection phase is complete and application data can flow. The host can redistribute endpoint 0 credits to the application endpoints.

HTC Endpoint Data Flow

HTC Credits and Buffers

The HTC protocol manages credits in the send direction with each credit representing a fixed amount of buffer space. The meaning of each credit and the total available credits are conveyed to the host through the HTC ready message. The host may not issue a message on an endpoint whose credit balance is 0. The host must wait for the target to process buffers and return credits for the endpoint. The host however, ultimately decides how credits are distributed. In general, credits should be distributed back to the endpoint from which the buffer was processed, closing the resource-loop. However, based on traffic patterns, over-subscribed endpoints may have their credits re-distributed more evenly to other endpoints.

HTC QoS

The close-loop credit reporting architecture indirectly provides some level of QoS (quality of service). QoS is provided through resource reservation rather than bandwidth reservation. Credit availability paces the rate at which the host can send over a particular endpoint. Because the target processes high priority messages more frequently, endpoint credits are generally made more available to the endpoint with the highest priority. It is imperative that the host credit re-distribution policy take into account the desired close-loop behavior and redistribute credits back to the endpoints they were originally assigned to. Credit re-balancing, however can override this imperative when endpoints become inactive or active. QoS and bandwidth reservation is host-interconnect specific, but is beyond the scope of this document.

HTC Credit Synchronization

The host is offered a total credit count at startup representing the host-side cached credit pool. The host should distribute these credits to endpoints and keep a cached view of each endpoint. To minimize bus traffic attributed to credit synchronization, credit reporting uses both demand-based and opportunistic mechanisms (described below). Credit reports indicate the number of free credits (processed buffers) available on each endpoint since the last report. The diagram below shows a simplified flow of messages and credit reports. Credits can be reported on two paths, demand-based via zero-length HTC messages on endpoint 0 or opportunistic on an endpoint's Rx message path. The example shows an end point that has three allocated credits. The host sends three messages. The third message consumes the last credit and thus has the NEED_CREDIT_UPDATE flag set.

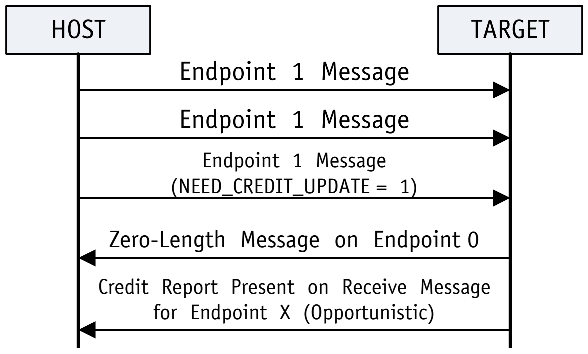
HOST
TARGET
Endpoint 1 Message
Endpoint 1 Message
Endpoint 1 Message (NEED_CREDIT_UPDATE = 1)
Zero-Length Message on Endpoint 0
Credit Report Present on Receive Message for Endpoint X (Opportunistic)

Figure B-2. **HTC Credit Synchronization**

Endpoint Credit Reporting State Machine

A target endpoint credit state can either be idle or in a state where the host needs credits. Figure B-B-3 shows the conditions to enter/exit the state (marked in green) and any required actions marked in red.

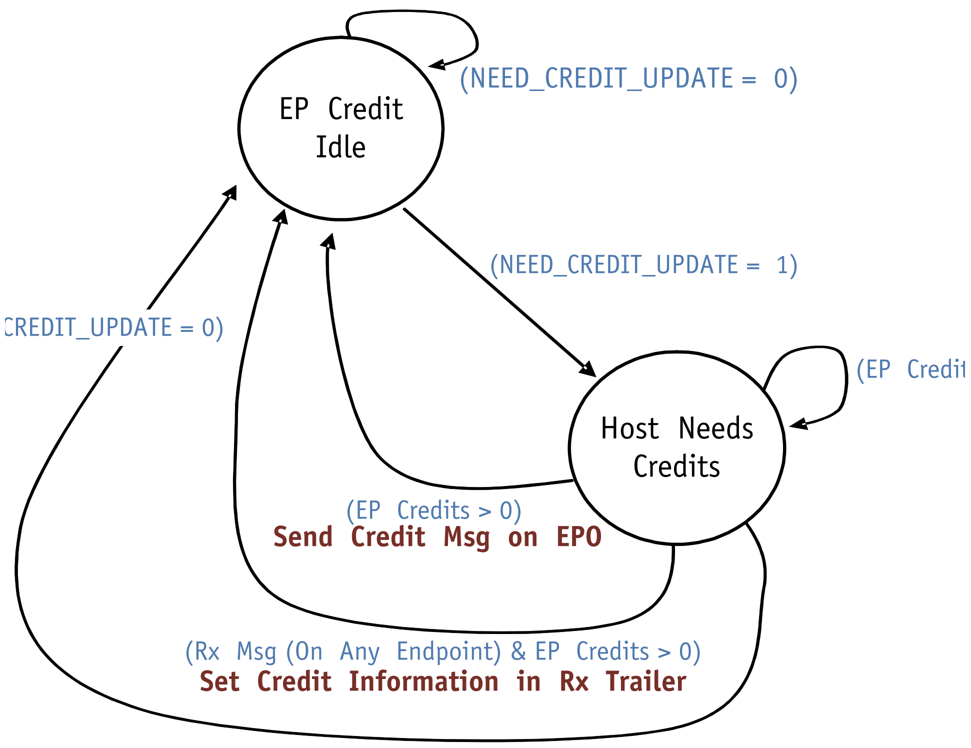

Figure B-3. **Endpoint Credit Reporting State Machine**

Demand-Based Endpoint Credit Updates

Demand-based triggering occurs when the host's view of credits drops to some level (typically 0). The host always checks an endpoint's credit balance prior to transmission and sets/clears the NEED_CREDIT_UPDATE flag of the outgoing message. One simple algorithm would be:

> If the current send message consumes the last credit then set the NEED_CREDIT_UPDATE flag

This flag is an indication to the target that the host is low or out of credits for a particular endpoint. Because the flag arrives in the header of the send message, there is zero overhead in issuing the request. Whether the flag triggers a credit report message depends on the current pending credits available (since the last update) for that endpoint. The host should not expect an immediate (or any) response to the NEED_CREDIT_UPDATE flag (a suggestion to the target that the host needs credits for an endpoint). When credits are not immediately available (buffers are still being processed), the target will maintain a host-needs-credits state and send a credit message as soon as it is possible.

If the target had taken an opportunity to send credit updates in un-related Rx traffic, the overlapping arrival of a NEED_CREDIT_UPDATE flag could place the endpoint in the host-needs-credits state, even though a report was already given, which may generate a new credit message as soon as one credit is made available again. This extra credit report is an acceptable side-effect in lieu of adding more complexity to the update mechanism (where the target may know what it had previously sent). Additionally, a host with enough credits would clear the NEED_CREDIT_UPDATE flag in subsequent frames and this would be sufficient for the target endpoint to move out of the host-needs-credits state.

The credit message always reports all endpoints that have at least one credit to offer even though the NEED_CREDIT_UPDATE flag was targeted for one specific endpoint, which is a second opportunistic mechanism for updating credits. Other credit reports can piggy-back onto the outgoing credit message.

Demand-based triggering has the consequence that the frequency of credit messages (adding to bus overhead) is governed by activity on the endpoint with the smallest number of allocated credits. The effects are somewhat diminished if there is a reasonable amount of Rx traffic (on any endpoint) to return credit reports. Because the host ultimately controls the setting of NEED_CREDIT_UPDATE flags, it is plausible that further modification of the host credit distribution algorithm can reduce the effects of endpoints with low credits that can cause a higher credit update frequency.

Opportunistic Endpoint Credit Updates

Credit reports arrive in the trailers of Rx messages. Whether it is a zero-length message (demand-based) or added to the end of an endpoint’s Rx message, the target sends new credit reports for all endpoints that have at least one credit to offer. The Rx message trailers however are of limited size and may not accommodate all credit reports. The zero-length HTC message, which should have enough trailer space at all times, covers any potential shortfalls.

Using HTC with AR6000 Mailbox Interface

The AR6000 device defines four bidirectional FIFO queues for communication between host and target. Each mailbox is represented by a 2 KB, 6 KB, or 12 KB (depending on the chip) section of AR6000 address space. Data written anywhere into that address space is pushed onto the end of the Tx FIFO for that mailbox. Data read from anywhere in that address space is taken from the beginning of the Rx FIFO for that mailbox HTC over SPI and SDIO interfaces utilize only mailbox 0 for all communications.

Additional AR6000 host-accessible registers are used to support send and receive operations via the mailboxes. These registers provide access/control to device interrupts, status and Rx lookaheads. Further information can be found in “Host Interface Support” on page D-1.

C DataSets

Board Data

Part of the manufacturing process for the AR6000-based board uses the ART application. This process produces calibration data that may be specific to a particular instance of a board. The cal data is stored in the EEPROM, host storage, and OTP along with other board data that includes information indicating which bands (802.11a, 802.11g, etc.) can be used by a board, the unique MAC address of the board, and the wireless regulatory domain information.

The WLAN will not function without proper checksummed board data and board-specific calibration data is required for optimal wireless networking performance.

NOTE: The Board Data however is NOT stored in a DataSet.

DataSets

A DataSet is a collection of related information that target firmware accesses through a key, the DataSet ID. DataSets store the wireless regulatory database, analog configuration, GPIO configuration, and startup patches. The format of a DataSet is completely DataSet dependent. The only standard attributes associated with a DataSet are dsetid, version, and length.

DataSets are stored in ROM or RAM.

DataSets may be of interest to the host-side developer who wishes to customize one or more DataSets for an implementation (e.g., to use a modified regulatory database).

Regulatory Database

The regulatory database contains information needed by the AR6000 wireless application to guarantee that the radio emissions comply with established regulations for a given domain. For instance, regulations that manage use of radio frequencies in Japan versus Europe versus the United States are all different.

Atheros distributes one version of the regulatory database, but vendors may change this database to keep up with recent regulations. Included in the SDK distribution is the regDomainGenDb tool. See the README file and the header file **regTableData.h** for more details.

Once an appropriate regulatory database is generated, it should be stored in a DataSet (e.g. in RAM or in flash) so the target firmware can retrieve it as needed. The regulatory database is retrieved using DataSet IDs starting with DSETID_REGDB (currently 2). See **include/regulatory/**, **include/dsetid.h**, and **host/support/dsets.txt**.

Analog Configuration

Analog configuration contains information needed to configure the AR6000 wireless radio. Atheros distributes one version of analog configuration data for each supported crystal speed (see **host/support/analogconfig**). A few parameters in this data can be modified through a special script available to board manufacturers (**modify_bnk6bin.pl**).

Analog configuration for a given crystal speed is stored in a DataSet. Target firmware retrieves this data using the DataSet IDs shown in Table C-1 (starting with DSETID_ANALOG_CONTROL_DATA_START, or dsetid=5):

Table C-1. **DataSet IDs**

Speed	dsetid
19.2 Mhz	5
26 MHz	6
40 MHz	7
52 MHz	8
38.4 MHz	9
24 MHz	10

Startup Patch DataSet

Just before entering the BMI phase of bootup, flash-based (not ROM-based) firmware looks for a startup patch DataSet to perform arbitrary initializations without any host interaction. It is more accurately described as a list of address/value pairs terminated with address = PATCH_END. When a Startup Patch is applied, software walks the list and plugs each value into each address. This allows for arbitrary initialization of registers. A patch DataSet consists of one or more patches, each containing a 32-bit command word followed by a command-dependent payload as shown in Table C-2.

Table C-2. **Patch DataSet Commands**

Command	Definition
PATCH_SKIP	Payload is a single word which indicates to the target a number of words to skip before looking for the start of the next patch
PATCH_CODE_ABS	Payload is an absolute target address, which contains code to be executed immediately
PATCH_CODE_REL	Payload is added to the address of the patch itself to produce a target address, which contains code to be executed immediately
PATCH_END	Marks the end of a patch DataSet

The startup patch DataSet is roughly equivalent in power/capability to a BMI and can augment BMI or substitute for BMI when host communications are inconvenient. In particular, the startup patch DataSet can perform initializations then cause firmware to bypass the BMI phase and proceed directly to the WLAN application (athwlan) using device option flags.

NOTE: New: A second set of startup patches (dsetid=0x29) is applied AFTER the BMI phase. The first set of patches (dsetid=0x26) is most useful in flash-based systems: the DataSet is stored in flash and used before BMI phase. The second set of patches can be downloaded during the BMI phase into Target RAM, and they are used after BMI phase. In general, it is easier to handle simple initializations directly through BMI messages rather than by loading a Startup Patch DataSet.

GPIO Configuration Patch DataSet

GPIO can be configured through the GPIO configuration patch DataSet (dsetid=DSETID_GPIO_CONFIG_PATCH=0x27). The format of this DataSet is identical to the format of the startup patch DataSet, but the GPIO patch is applied when GPIO is initialized rather than during startup. In the usual case, GPIO initializes when the athwlan wireless application begins.

A GPIO configuration patch is used only to initialize GPIO registers using patches that consist of address/data pairs.

D
Host Interface Support

The AR6000 module is physically connected to an interface host controller which is considered to be a part of the host system and controlled by the host processor. For details of the interface protocols, see the corresponding standards documentation.

Interface Protocols

The AR6000 module supports a variety of interfaces for connectivity to the host system.

- SDIO (both 1-bit and 4-bit modes)
- SPI

NOTE: Only one interface can be used in any solution configuration.

Software Interface Components

The device driver supporting the host controller handles all software interactions from the host to the AR6000 module. The software interface has two primary manifestations:

- Common information area (CIA)
- Mailbox operation

The AR6000 module presents a very similar software interface to each of the physical interfaces on the host system.

Both types of interactions rely on a host accessible address space provided by the AR6000 and an interrupt mechanism from the AR6000 to the host system.

Each interface protocol implies its own mechanism for accessing the AR6000's host visible address space and delivering interrupts from the AR6000. The SDIO interface uses CMD52 and CMD53 access mechanisms.

Common Information Area

The common information area (CIA) provides access to registers to identify and initialize the AR6000 device. It can also be used to get information about the chip's SDIO capabilities: support for byte/block mode, maximum block size, support and configuration for different bus modes (e.g., 1-bit or 4-bit).

Over the SDIO interface this area includes the card common control register (CCCR) and function basic register (FBR). The device also supports the CIS tuple space for CIS0 and CIS1. The CIA register layout and contents space are defined as part of the interface-specific standard. See the *AR6003 ROCm Single-Chip MAC/BB/Radio for 2.4/5 GHz Embedded WLAN Applications* data sheets and interface standards for CIA information.

Mailboxes

The device provides four bi-directional mailboxes for communication between the host and the target. The mailbox is a FIFO to send data to and from the host. Mailbox operation handles all AR6000 device communications, regardless of interface, e.g, having four mailboxes allows for the exchange of four parallel streams of data between the host and the device module. Although the AR6000 device provides four hardware mailboxes, the wireless module firmware may utilize fewer mailboxes. The number of mailboxes used in host-to-target communications can change between different software releases to meet new and evolving requirements.

For SDIO, the SDIO function-1 address space handles mailbox operation. For information on address maps and mailbox operation for each supported interface, see the *AR6003 ROCm Single-Chip MAC/BB/Radio for 2.4/5 GHz Embedded WLAN Applications* data sheet. Each of the mailboxes is implemented as an address space available to the host to read and write data.

- Writing a byte to anywhere in the address range of a mailbox pushes that byte into the FIFO of data being sent by the host to the device module.
- Reading a byte from anywhere in the address range pulls a byte from the FIFO of data being sent by the device to the host.

Messages

Data sent from the host to any of the mailboxes (referred to as the transmit data) can be packetized into messages. To send a packet over a mailbox, write packet data to anywhere in the address range corresponding to the mailbox. The last byte of the packet should be written to the end-of-message (EOM) address. The last address of a mailbox's address range is designated as the EOM address. Therefore, a message of *N* bytes can be sent using a single write of n bytes starting at the address *N* bytes ahead of the EOM address.

The size of the address space available to each mailbox depends on the interface. For messages longer than the available address space, the write operation must be split. The size of each mailbox address range depends on the interface mode (SDIO or SPI) and the chipset version.

Flow Control

For more information on mailbox operation, see the *AR6003 ROCm Single-Chip MAC/BB/Radio for 2.4/5 GHz Embedded WLAN Applications* data sheet. Note that software components may use hardware features to facilitate flow control while others may use a pure software approach. Refer to the "Bootloader Message Interface (BMI)" on page B-1 and "Host Target Communications" on page B-3.

E

SPI Host Interface

This document provides a register-level programming guide for the SPI I/O interface used on AR6000-based devices. This document provides bus-level protocol details and register programming models for SPI-connected AR6000 chipset devices.

SPI Bus Interface

AR6000 devices support multiple serial I/O bus protocols that can be selectively enabled over a set of host I/O pins. The most common serial protocols are secure digital I/O (SDIO) and serial peripheral interface (SPI). SDIO is the preferred interconnect for high performance applications. SPI provides legacy connectivity (at reduced performance) to many existing application processors that may only have SPI or a limited set of available SDIO host ports.

The programming interface to AR6000 devices remains virtually the same whether operating over SDIO, SPI, or other supported serial protocol. The AR6000 device provides a consistent register-level programming interface to the host. The host interconnect (SPI or SDIO), provides the low-level bus protocol and access methods to read and write these registers. Atheros reference software provides an abstract register-access model that converts high-level operations into bus-specific transactions (such as HIF). The majority of the host driver operates in an non-interface-specific manner.

In the system interconnect, only one serial bus protocol may be enabled at chip power-on. Each bus interface consists of internal registers (SDIO CCCR or SPI control) to configure and control the serial bus; these internal registers operate within the clock domain of the bus interface and are independent of the state of the chipset core device logic. Additionally, all interconnects have a bridging mechanism to bridge host accesses to registers residing in the chipset core logic domain. AR6000 devices use aggressive clock gating schemes to minimize power consumption, requiring that the bus interconnect manage bus transactions between the bus interface clock domain and the core clock domain. The bus protocols ensure the core logic is fully clocked and able to properly handle the transaction.

Figure E-1 provides a high level view of the system interconnect.

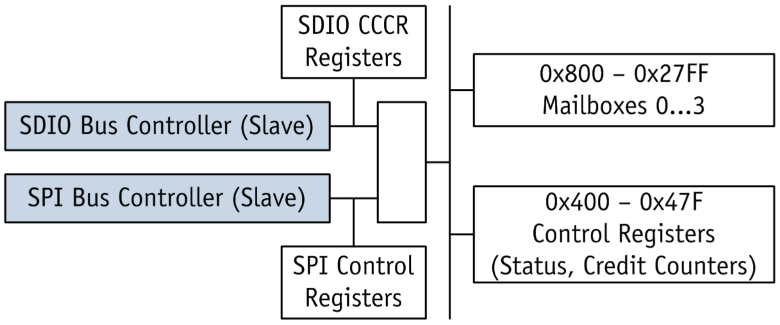
SDIO CCCR Registers
SDIO Bus Controller (Slave)
SPI Bus Controller (Slave)
SPI Control Registers
0x800 – 0x27FF Mailboxes 0...3
0x400 – 0x47F Control Registers (Status, Credit Counters)

Figure E-1. **SPI/SDIO Interface**

The AR6000 device operates as a SPI slave device through a 5-pin interface. The SPI slave uses the standard 4-wire Motorola-SPI protocol up to 48-MHz clock operation (see Figure E-2). The additional SPI_INTR is a signal from the AR6000 device to indicate a pending interrupt condition. The interrupt is level-triggered and typically tied to a host processor GINTERNAL signal.

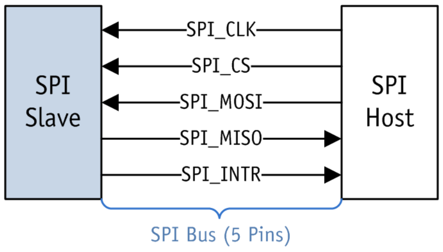

Figure E-2. **SPI Bus Diagram**

Signal	Description
SPI_CLK	SPI clock signal driven by the host during a bus transaction.
SPI_CS	Bus qualifier to select the AR6000 device if the device is in a shared bus-arrangement with other peripherals. The SPI slave tri-states SPI_MISO when CS is deasserted as well as ignore any data issued on SPI_MOSI (does not interpret SPI frames).
SPI_MOSI	The master-out-slave-in signal driven by the master to indicate a bus command or write data.
SPI_MISO	The master-in-slave-out signal; the return path of read data requested by the master.
SPI_INTR	AR6000 pending interrupt signal.

The SPI host bus controller should support these features:

- Minimum of 8-bit data framing (16-bit is recommended)
- Clock phase and polarity control (48 MHz requires SPI-Mode-3 clock phase and polarity)
- Level triggered interrupt for SPI_INTR line. This signal must be maskable by software.

For optimal performance, these features should be considered:

- 16 and 32-bit data framing (reduces inter-frame gaps for CS assertion)
- SPI-controller DMA support for large SPI data frame sequences

SPI Bus Protocol

The SPI protocol uses an Atheros proprietary framing protocol and bus command set. The minimum bus data unit is 1 byte and framing is processed on byte boundaries. A bus transaction consists of a command phase followed by a data phase. The command phase operates in the master-to-slave direction (bytes appear on SPI_MOSI line). The data phase occurs on either the SPI_MOSI line (for writes) or SPI_MISO line (for reads). See Figure E-3.

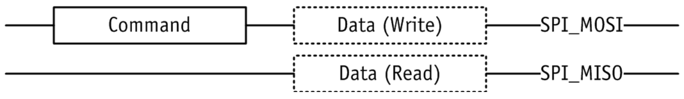

Figure E-3. **SPI Bus Protocol**

The bus command consists of two bytes followed by one or more bytes for the data phase of the transfer. Sequences of bytes on the SPI bus can be optimized to the host controller SPI data width capability (8, 16, or 32 bits). Minimally, all SPI controllers support 8-bits, but controllers can get better bus utilization if they support larger data frames. For example, the 2-byte command phase can be issued as either two 8-bit SPI data frames or a single 16-bit data frame. A 4-byte data sequence can be issued as four individual 8-bit SPI data frames or a single 32-bit data frame. The latter is more optimal on the bus as inter-frame gaps can be eliminated. Atheros host reference software attempts to use SPI data frames optimal to the host controller data framing capabilities. When 16 or 32-bit data frames are used, byte swapping may be necessary when the SPI host controller uses an external DMA controller. The AR6000 SPI controller has internal byte swapping accommodating such SPI host controller designs and can eliminate the overhead of software byte swapping.

Power State Synchronization

The bus protocol allows the host to address and access internal SPI control registers and core logic registers. Power state synchronization is handled by the bus protocol using internal state machines and SPI buffers (Tx/Rx FIFOs). The bus protocol alleviates the host from having to perform costly spin-wait operations until the core-logic clock is available. Other SPI-bus protocols can cause excessive host CPU use and affect other system tasks. The Atheros SPI bus protocol uses interrupts and internal SPI buffers with programmable full/empty thresholds to allow the host to interact with the AR6000 device in an asynchronous manner with much lower impact to CPU resources.

The AR6000 device uses an internal SPI buffer to provide bus-level flow control when the core logic is not clocked. A SPI-domain Tx buffer holds data flow from the host to the mailbox in the event that the AR6000 core logic is asleep. Internal SPI logic automatically transfers data from the SPI Tx buffer into core logic. The host manages the flow of data into this buffer using SPI control registers to determine the space left in the SPI buffer. Interrupts can be determine when space is available. An SPI-domain Rx buffer holds prefetched data from the core logic mailbox, allowing the host to quickly remove the data from the prefetch buffer even if the core logic has a clock. In VOIP applications, the entire VOIP packet can reside in the SPI buffer allowing the AR6000 device to aggressively sleep but still allowing the host to pull the VOIP frame out of the buffer because it already resides in the SPI domain.

SPI Bus Access Types

The SPI protocol provides three bus access methods :

- SPI internal register read/writes
- SPI external register read/writes (core logic register access)
- SPI external read/writes to the mailbox (over the SPI buffers)

SPI Internal Register Read/Write

This method provides access to internal SPI configuration and control registers. All operations operate in the SPI clock domain and are independent of the sleep state of the core-logic. Internal registers are used for interface setup and used to setup/program SPI external accesses.

Internal registers are 16-bits wide. The data phase of the bus transaction is always 16-bits (2 bytes).

SPI External Register Read/Write

This method provides access to core-logic registers in a power-safe manner. Core logic registers include interrupt control/status and counter registers. These register typically are accessed 4–32 bytes at a time. This bus access method requires the programming and activation of a SPI state machine to handle the bus transaction asynchronously. The SPI state machine is programmed using SPI-internal read/write operations. An interrupt (SPI_INTR) can be signal-upon-completion of the access or the host can poll for completion. The host can access up to 32-bytes of core-logic register space in a single operation. The host software is responsible for insuring that no more than 1 external register access may be in progress.

SPI External Read/Write to Mailbox

The host communicates to the target using a mailbox where the host can push or pull bytes at specific external mailbox address locations. The mailbox accesses are fully buffered through a set of FIFOs to synchronize the host with the core-logic sleep state.

SPI Command

The SPI interface is a slave implementation responding only to host-initiated bus commands. All SPI commands are 16-bits wide and include a read/write bit and external/internal address bit followed by a 14-bit address. The most significant bit of a command is transmitted first on the SPI_MOSI pin.

Command Types

Table E-1 shows the SPI operation types; all of them are distinguished by the 16-bit SPI command. See register information in "SPI Internal" on page F-13.

Table E-1. **SPI Command Types**

Command Type	Access	Internal/ External	Address
SPI (internal) register read	1	1	Any range within the internal address space
SPI (internal) register write	0	1	Any range within the internal address space
SPI DMA (mailbox) read	1	0	Any range; the address is ignored, because the read pops data from the mailbox
SPI DMA (mailbox) write	0	0	External address space: 14'h0800 to 14'h27FF (MBOX alias)

SPI external accesses to core-logic registers are handled through a series of internal register operations. The SPI host accesses external registers using a proxy built into the SPI controller. The proxy is programmed using a series of internal register writes that includes address/length and a read/write flag. The proxy performs the access and indicates to the SPI host when it is complete. When referencing a core-logic register, the core-logic register address map (also known as host-register address map) is used. Figure E-4 shows the core-logic register map.

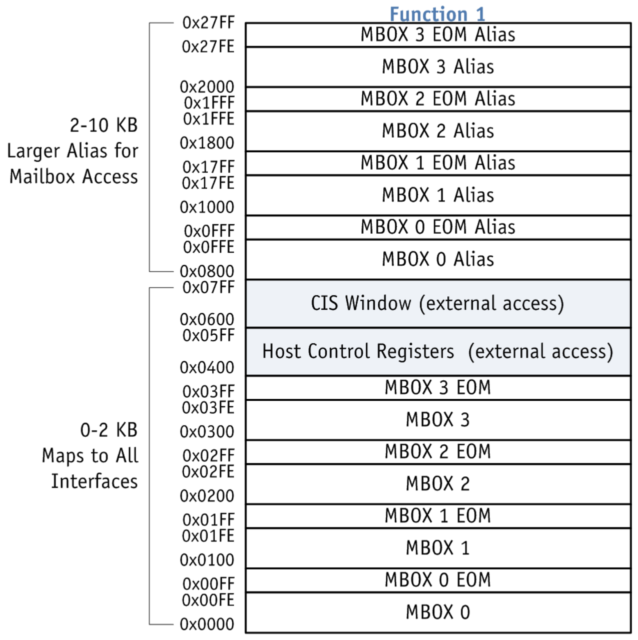

Figure E-4. **Core-Logic Address Map use for External Access**

Transfer Format

It is important for the SPI host to choose the SPI clock polarity and phase format that matches the SPI slave. The SPI slave samples the incoming data (SPI_MOSI) on the rising edge and shifts out the outgoing data (SPI_MISO) on the falling edge of SPI_CLK pin. See Figure E-5.

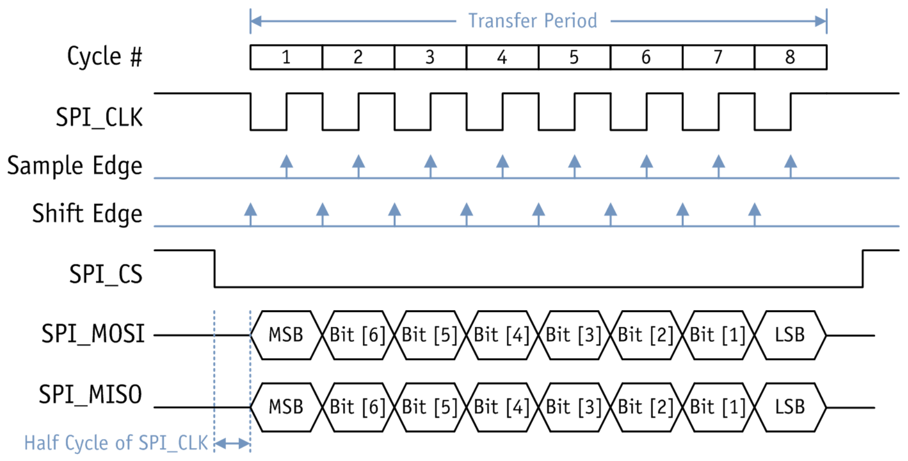

Figure E-5. **Transfer Format Diagram in 8-Bit Mode**

SPI Internal Register Access

SPI internal registers are fully accessible even when core logic is asleep (fully gated). All SPI internal registers are 16 bits wide and can only be accessed in 16-bit mode. The read or write data after the SPI command needs to be clock within a single SPI_CS assertion window. An SPI internal access is comprised of a16-bit command (command phase) followed by 16 clocks of data. For an internal write, the host shifts out 16 bits of data on the SPI_MOSI signal following the command phase. For an internal read, the SPI slave shifts out 16 bits of data on the SPI_MISO signal following the command phase. During the internal read data phase, SPI_MOSI is ignored by the slave controller.

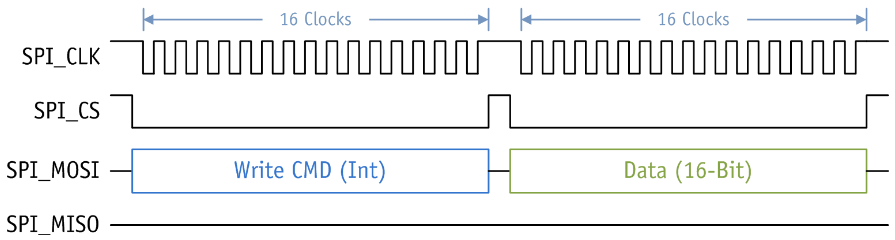

Figure E-6. **SPI Internal Write**

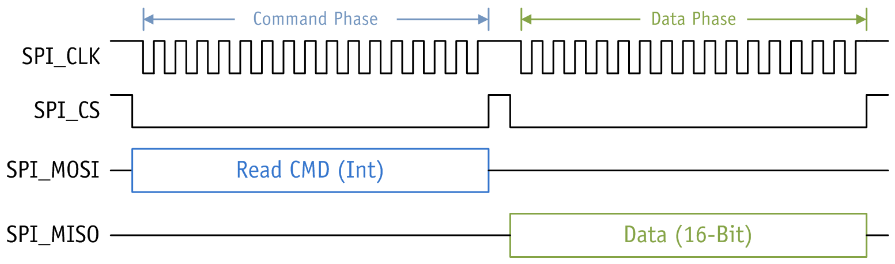

Figure E-7. **SPI Internal Read**

SPI External Mailbox Access

SPI mailbox accesses are the primary mechanism to transfer packet data between the host and target. AR6000 devices can accept mailbox data at any time, even with core logic asleep (clock gated). Internal SPI logic fully buffers packets until core logic is fully awake and able to transfer the data. In the transmit direction, a host may post as much data (packet writes) as there is SPI buffering available. In the receive direction, a host pulls as many data packets as indicated by the SPI receive buffer.

Transfer Mode

SPI supports 8-bit, 16-bit, and 32-bit data frames for the data phase portion of the transfer. A special Turbo mode ignores chip select toggles between frames. The entire data phase can be a continuous stream of data bits without chip-select gaps (if the host controller supports this mode). Turbo mode is automatic for mailbox accesses.

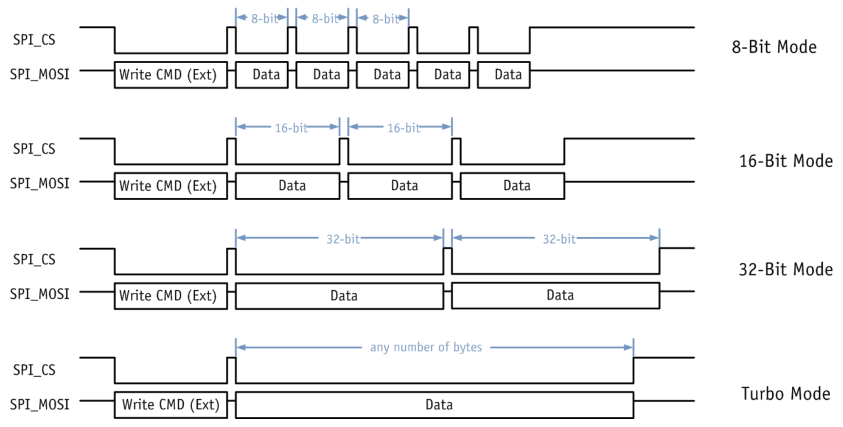

Figure E-8. **Transfer Mode**

SPI External Mailbox Write

Packet data from the host is written to the core-logic mailbox address range and stored in a write buffer (FIFO). The write buffer typically has 3200 bytes of storage capacity allowing for rapid back-to-back writes of two full network frames. The host can read the WRBUF_SPC_AVA register on power up to determine the buffer size. The buffer size may be reduced to optimize area.

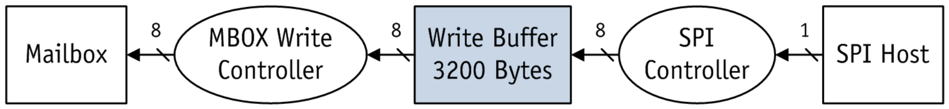

Figure E-9. **Write Buffer**

If the core-logic is awake and able to accept data, the packet data within the write buffer immediately transfers to the target mailbox logic. If core-logic is asleep the SPI interface wakes core-logic. Core-logic may require a few ms to return to a fully clocked state (thus the need for synchronizing FIFOs). While core-logic is returning to normal clocking, the write data is temporarily stored within the SPI write buffer. The host is free to fill this write buffer until it fills. The internal WRBUF_SPC_AVA register contains the total number of empty space (in bytes) available in the buffer and can be accessed at any time.

It is recommended for the host to issue an internal access for the WRBUF_SPC_AVA register before issuing the mailbox write command. If there is enough room for the next packet, the host should program the DMA_SIZE register with the upcoming packet size. The SPI host can then issue the DMA write command follow by the packet data.

Mailbox write steps:

1. Internal read from WRBUF_SPC_AVA register.
2. Internal write to DMA_SIZE register.
3. Start DMA write command follow by the data phases.
4. If the host has another packet to send to AR6000 devices, go back to step 1.

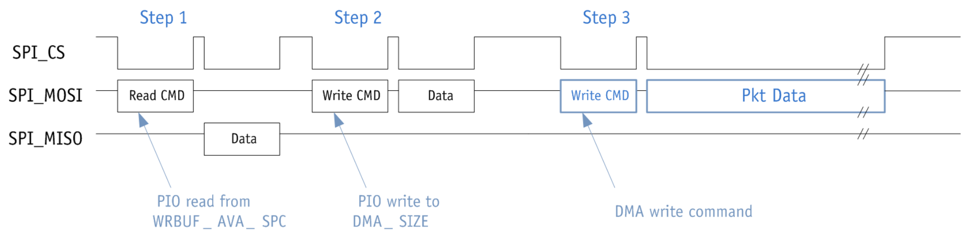

Figure E-10. **DMA Write Steps**

If the incoming write data overflows the buffer (DMA_SIZE > WRBUF_SPC_AVA), the SPI slave will drop the incoming data entirely. A write buffer error interrupt is issued to the host. The host must insure that write data never overflows the buffer.

SPI External Mailbox Read

The SPI controller pre-fetches data from the core-logic holding receive packets. Pre-fetched data is stored in a single 3200 byte read FIFO.

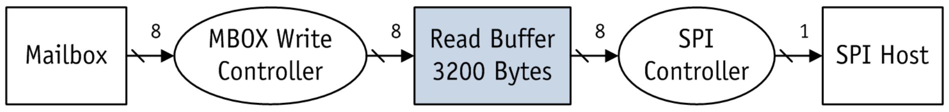

Figure E-11. **Read Buffer**

Once a full packet has been pre-fetched from the mailbox, or the total number of bytes within the read buffer has exceeded RDBUF_WATERMARK register, a packet available interrupt will be issued to the SPI host.

Once the host has received the interrupt, it can issue an internal read from the INTR_CAUSE register. If the interrupt is caused by the packet available interrupt, another INTERNAL read can be issued to RDBUF_BYTE_AVA register to determine the total number of data bytes held in the read buffer. The final step to fetch the packet is to set the DMA_SIZE register with the intended read length. The host can then issue the DMA read command. In order to correctly fetch read data on packet-boundaries, the host must read the RDBUF_LOOKAHEAD registers and determine the amount of data to fetch based on some application-specific packet header.

Mailbox Read Steps:

1. Interrupt going from AR6K devices to SPI host.
2. INTERNAL read from INTR_CAUSE register.
3. INTERNAL read from RDBUF_BYTE_AVA register.
4. INTERNAL write to DMA_SIZE register.
5. Start DMA read command and start reading the data by de-asserting chip select pin.
6. The packet available will be cleared by HW at the end of the DMA read.

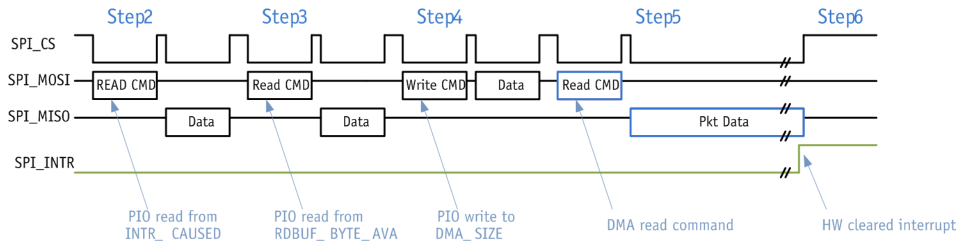

Figure E-12. **DMA Read Steps**

If the read operation is larger than the total available bytes within the read buffer (DMA_SIZE > RDBUF_BYTE_AVA), the read command will be ignored and the SPI slave will send out garbage onto SPI_MISO pin. At the same time, a read buffer error interrupt will be issued to the host.

SPI External Core Logic Register Access

The SPI host can access core-logic control registers using a fully power-synchronized transfer mechanism. This mechanism is intended for simple control register access that is infrequent (non-data). 32 bytes of the write and read buffer are reserved for core-logic register access. A SPI register transfer proxy handles the actual transfer. The host is required to setup the transfer and wait for completion.

The host may not allow more than one external access at a time. Host software must interlock the register read/write sequence and subsequent completion whether polled or interrupt driven.

Host Control Register Write

Host control register write steps:

1. INTERNAL write, number of bytes (*N*) to HOST_CTRL_BYTE_SIZE register (max of 32 bytes).
2. INTERNAL write to HOST_CTRL_WR_PORT with *N* data frames.
   Note: The data frames can be written using 8-bit, 16-bit, 32-bit, or Turbo mode. But the total number of bytes needs to match HOST_CTRL_BYTE_SIZE.byte_size.
3. INTERNAL write to HOST_CTRL_CONFIG.address, HOST_CTRL_CONFIG.direction and HOST_CTRL_CONFIG.enable.
   Note: As soon as the host asserts the enable bit, HW will automatically clear the INTR_CAUSE.host_ctrl_wr_done bit. This will prevent any stale copy of the done bit (from previous write transition) for triggering the main interrupt line.
4. INTERNAL read from SPI_STATUS.host_access_done.
   If not done after (*X*) retries, enable INTR_ENABLE.host_ctrl_wr_done.
5. Upon interrupt, clear interrupt by writing to INTR_CAUSE.host_ctrl_wr_done.

Host Control Register Read

Host control register read steps:

1. INTERNAL write number of bytes (*N*) to HOST_CTRL_BYTE_SIZE register (max of 32 bytes).
2. INTERNAL write to HOST_CTRL_CONFIG.address, HOST_CTRL_CONFIG.direction and HOST_CTRL_CONFIG.enable.
   Note: as soon as the host asserts the enable bit, hardware automatically clears the INTR_CAUSE.host_ctrl_rd_done bit, preventing any stale copy of the done bit (from previous read transition) for triggering the main interrupt line.
3. INTERNAL read from SPI_STATUS.host_access_done.
   If not done after (*X*) retries, enable INTR_ENABLE.host_ctrl_rd_done.
4. Upon interrupt, INTERNAL read to HOST_CTRL_RD_PORT using *N* data frames to retrieve read data.
   Note: The data frames can be read using 8-bit, 16-bit, 32-bit, or Turbo mode. But the total number of bytes needs to match HOST_CTRL_BYTE_SIZE.byte_size.
5. Clear interrupt by writing to INTR_CAUSE.host_ctrl_rd_done.

Byte Swapping

Some SPI bus controllers may swap bytes when 16-bit or 32-bit data frames are DMAd from host memory, which only affects data frames DMAd from host memory to/from the mailbox. SPI external core-logic register accesses do not require swapping as they typically do not use host DMA for such small transfers. Mailbox data however can be full network frames of 1500+ bytes and may use the host SPI DMA controller to optimize the transfer.

To avoid host software from having to correct the byte swapping, the AR6000 SPI controller has a byte swapping mode feature programmed using SPI_CONFIG.swap bit. The SPI controller must also be programmed for the size of the byte swap either 2 bytes (16-bit) or 4 bytes (32-bit), as controlled by a single SPI_CONFIG.16bit_mode. When 0, the swap is a 4-byte swap. When 1, the swap occurs on 2 byte pairs. The host must only issue mailbox data frames whose transfer length align to the swap size of 2 or 4 bytes.

Figure E-13 through Figure E-15 illustrate which bytes get received or sent first in the case of byte swapping. The diagrams only show the DMA write transfer, in which the incoming data byes are swapped before going into the write buffer. For DMA read transfer (not being shown in diagram), outgoing data bytes are swapped similarly coming out from the read buffer.

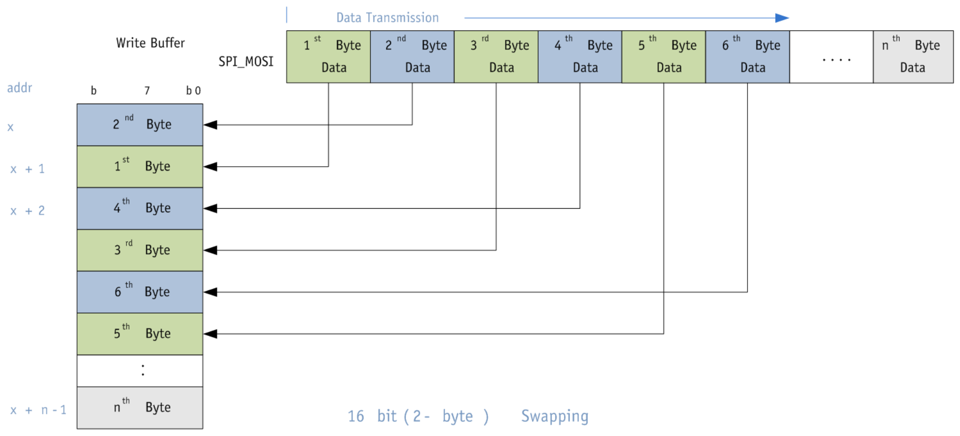

Figure E-13. **SPI 16-Bit Byte Swapping**

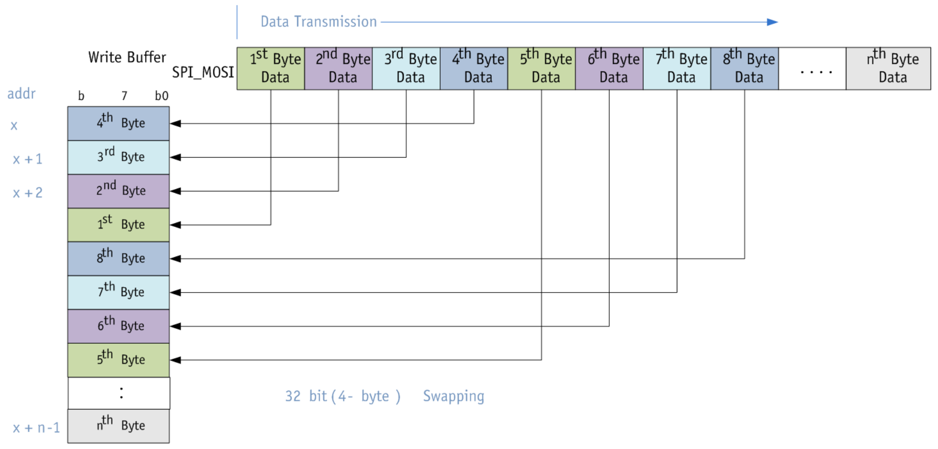

Figure E-14. **SPI 32-Bit Byte Swapping**

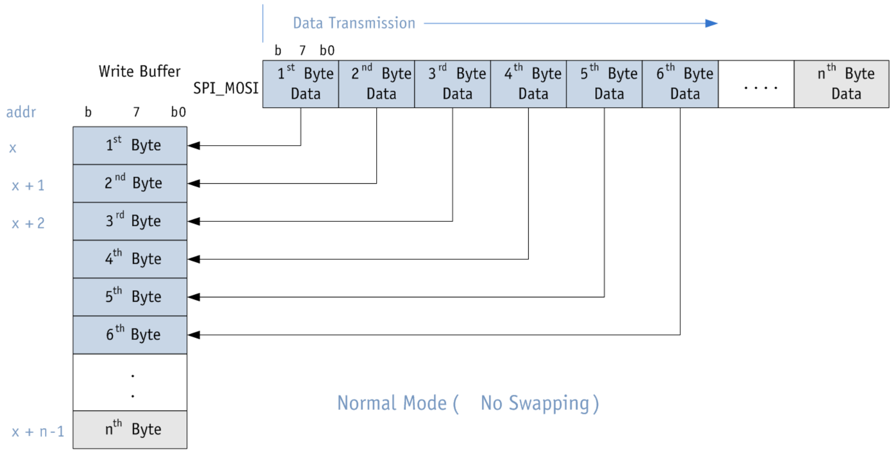

Figure E-15. **SPI Normal Mode (No Swapping)**

F Registers

Host Interface

Table F-1 summarizes the host interface registers, seen from the external host. Unlike the AHB registers, the address space is in bytes, not words. The CPU only views these registers using the HOST_IF_WINDOW register.

Table F-1. **Host Interface Registers**

Offset	Name	Description	Page
0x400	HOST_INT_STATUS	Address for AHB Read Access	page F-2
0x401	CPU_INT_STATUS	CPU-Sourced Interrupt Status	page F-2
0x402	ERROR_INT_STATUS	Error or Wakeup Interrupt Status	page F-3
0x403	COUNTER_INT_STATUS	Host Interface Credit Counter Interrupt	page F-3
0x404	MBOX_FRAME	Mailbox FIFO Status	page F-4
0x405	RX_LOOKAHEAD_VALID	Valid Bits for Lookahead	page F-4
0x408	RX_LOOKAHEAD0	Lookahead MBOX Rx0 FIFO Bytes	page F-4
0x40C	RX_LOOKAHEAD1	Lookahead MBOX Rx1 FIFO Bytes	page F-5
0x410	RX_LOOKAHEAD2	Lookahead MBOX Rx2 FIFO Bytes	page F-5
0x414	RX_LOOKAHEAD3	Lookahead MBOX Rx3 FIFO Bytes	page F-5
0x418	INT_STATUS_ENABLE	HOST_INT_STATUS Enable Bits	page F-6
0x419	CPU_INT_STATUS_ENABLE	CPU-Sourced Interrupt Status	page F-6
0x41A	ERROR_STATUS_ENABLE	Error Interrupt Status	page F-7
0x41B	COUNTER_INT_STATUS_ENABLE	Credit Counter Interrupt Status	page F-7
0x420	COUNT	Credit Counter Direct Access	page F-8
0x440	COUNT_DEC	Credit Counter Atomic Increment	page F-8
0x460	SCRATCH	Interface Scratch	page F-9
0x468	FIFO_TIMEOUT	FIFO Timeout Period	page F-9
0x469	FIFO_TIMEOUT_ENABLE	FIFO Timeout Enable	page F-9
0x46A	DISABLE_SLEEP	Disable Sleep Mode	page F-10
0x470	LOCAL_BUS	Local Bus and SPI Host Interface State	page F-10
0x472	INT_WLAN	Interrupt the CPU	page F-10
0x480	SPI_CONFIG	SPI Slave Interface	page F-11
0x481	SPI_STATUS	SPI Status	page F-12
0x600	CIS_WINDOW	SDIO CIS Tuples	page F-12

Pending Interrupt Status (HOST_INT_STATUS)

Offset: 0x400 Reset Value: 0x0 Access: Read-Only	Reads to this register return pending host interrupt bits. Writes to this register clear interrupt bits. Note: Write a 1 to each bit to be cleared. All bits written as 0 do not update the interrupt status for that bit.

Bits	Name	Description
7	ERROR	Error or Wakeup Interrupt (read only, clear using ERROR_INT_STATUS register)
6	CPU	Interrupt from the AR6002 CPU (read only, clear using CPU_INT_STATUS register)
5	AR6002_INT	Copy of the interrupt line to the AR6002 CPU. This interrupt is normally serviced by the AR6002 CPU, and should typically be disabled at the host interface. If the host does service this interrupt, more details can be read from the IN_STATUS register in the AR6002 address space, accessible via a window read.
4	COUNTER	Interrupt from software controlled credit counters. Read only, see the COUNTER_INT_STATUS register for details.
3:0	MBOX_DATA	Rx Data Pending in the corresponding MBOX (FIFO is not empty). This will be cleared when the FIFO is no longer empty.

CPU-Sourced Interrupt Status (CPU_INT_STATUS)

Offset: 0x401 Reset Value: 0x0 Access: Read/Write	Indicates the CPU interrupt condition being communicated by the AR6003 CPU. Software defines the meaning of each interrupt in this register. Writes to this register clear interrupt bits. Note: Write a 1 to each bit to be cleared. Writing a 0 does not change the bit value.

Bits	Name	Description	
7:0	BIT	Bit [7]	Interrupt #7
		Bit [6]	Interrupt #6
		Bit [5]	Interrupt #5
		Bit [4]	Interrupt #4
		Bit [3]	Interrupt #3
		Bit [2]	Interrupt #2
		Bit [1]	Interrupt #1
		Bit [0]	Interrupt #0

Error or Wakeup Interrupt Status (ERROR_INT_STATUS)

Offset: 0x402
Reset Value: 0x0
Access: Read/Write

Indicates a wakeup or error condition which caused the Error Interrupt in the register "Pending Interrupt Status (HOST_INT_STATUS)" on page F-2.

Bits	Name	Description
7:4	RES	Reserved
3	SPI	SPI Error Interrupt. This error can only be masked or cleared be accessing the SPI-specific registers from the SPI host.
2	WAKEUP	The client transitioning to the ON state. Set as the client enters ON, and can immediately accept host transactions. Writing a 1 clears the register field. Writing a 0 does not change the bit value.
1	RXUNDERFLOW	The host attempted to read a mailbox which did not contain data, and no data was available for a timeout period. This implies a software flow control error. Writing a 1 clears the register field. Writing a 0 does not change the bit value.
0	TXOVERFLOW	The host attempted to write a mailbox which was full and had no available buffer space for a timeout period. This implies a software flow control error. Writing a 1 to clears the register field. Writing a 0 does not change the bit value.

Host IF Credit Counter Interrupt (COUNTER_INT_STATUS)

Offset: 0x403
Reset Value: 0x0
Access: Read-Only

Read-only register to return counter interrupt status.

Bits	Name	Description	
7:0	COUNTER	Each counter sets and clears its interrupt bit as follows: ■ Set: Counter transitions from 0 → 1 ■ Clear: Counter transitions from 1 → 0 Bit mapping is as follows:	
		Bit [7]	Counter 7 interrupt
		Bit [6]	Counter 6 interrupt
		Bit [5]	Counter 5 interrupt
		Bit [4]	Counter 4 interrupt
		Bit [3]	Counter 3 interrupt
		Bit [2]	Counter 2 interrupt
		Bit [1]	Counter 1 interrupt
		Bit [0]	Counter 0 interrupt

Mailbox FIFO Status (MBOX_FRAME)

Offset: 0x404 Reset Value: 0x0 Access: Read-Only	Returns current MBOX framing status.

Bits	Name	Description
7:4	RXEOM	Rx FIFO contains a data byte with EOM marker set in the corresponding mailbox
3:0	RXSOM	Rx FIFO contains a data byte with Start of Message (SOM) marker set in the corresponding mailbox. A SOM byte always follows an EOM byte from the previous message.

Valid Bits for Lookahead (RX_LOOKAHEAD_VALID)

Offset: 0x405 Reset Value: 0x0 Access: Read-Only	Read only register to return current lookahead valid bits. If the bit is set, all four lookahead bytes are valid. If the bit is cleared not all bytes are valid. If cleared but the MBOX is not empty, only the first byte is valid but the others are not.

Bits	Name	Description	
7:4	RES	Reserved	
3:0	MBOX	Bit [3]	MBOX3 lookahead, all bytes valid
		Bit [2]	MBOX2 lookahead, all bytes valid
		Bit [1]	MBOX1 lookahead, all bytes valid
		Bit [0]	MBOX0 lookahead, all bytes valid

Lookahead to Next 4 MBOX Rx0 FIFO Bytes (RX_LOOKAHEAD0)

Offset: 0x408 Reset Value: 0x0 Access: Read-Only	This read only register array returns the first 4 bytes of the MBOX Rx0 FIFO. Reading this register will not pop or otherwise change the contents of the FIFO, it is non-destructive.

Address	Description
[1:0] = 0x0	Returns MBOX0 Rx FIFO Head-3 byte
[1:0] = 0x1	Returns MBOX0 Rx FIFO Head-2 byte
[1:0] = 0x2	Returns MBOX0 Rx FIFO Head-1 byte
[1:0] = 0x3	Returns MBOX0 Rx FIFO Head byte

Lookahead to Next 4 MBOX Rx1 FIFO Bytes (RX_LOOKAHEAD1)

Offset: 0x40C
Reset Value: 0x0
Access: Read-Only

This read only register array returns the first four bytes of the MBOX Rx1 FIFO. Reading this register not pop or otherwise change the contents of the FIFO, it is non-destructive.

Address	Description
[1:0] = 0x0	Returns MBOX1 Rx FIFO Head-3 byte
[1:0] = 0x1	Returns MBOX1 Rx FIFO Head-2 byte
[1:0] = 0x2	Returns MBOX1 Rx FIFO Head-1 byte
[1:0] = 0x3	Returns MBOX1 Rx FIFO Head byte

Lookahead to Next 4 MBOX Rx2 FIFO Bytes (RX_LOOKAHEAD2)

Offset: 0x410
Reset Value: 0x0
Access: Read-Only

This read only register array returns the first four bytes of the MBOX Rx2 FIFO. Reading this register not pop or otherwise change the contents of the FIFO, it is non-destructive.

Address	Description
[1:0] = 0x0	Returns MBOX2 Rx FIFO Head-3 byte
[1:0] = 0x1	Returns MBOX2 Rx FIFO Head-2 byte
[1:0] = 0x2	Returns MBOX2 Rx FIFO Head-1 byte
[1:0] = 0x3	Returns MBOX2 Rx FIFO Head byte

Lookahead to Next 4 MBOX Rx3 FIFO Bytes (RX_LOOKAHEAD3)

Offset: 0x414
Reset Value: 0x0
Access: Read-Only

This read only register array returns the first four bytes of the MBOX Rx3 FIFO. Reading this register not pop or otherwise change the contents of the FIFO, it is non-destructive.

Address	Description
[1:0] = 0x0	Returns MBOX3 Rx FIFO Head-3 byte
[1:0] = 0x1	Returns MBOX3 Rx FIFO Head-2 byte
[1:0] = 0x2	Returns MBOX3 Rx FIFO Head-1 byte
[1:0] = 0x3	Returns MBOX3 Rx FIFO Head byte

HOST_INT_STATUS Enable Bits (INT_STATUS_ENABLE)

Offset: 0x418
Reset Value: 0x1
Access: Read/Write

Enable bits for the HOST_INT_STATUS register. Each bit enables the corresponding bit in the HOST_INT_STATUS register. Bit values:

- 0 = interrupt is disabled
- 1 = interrupt is enabled

Bits	Name	Description
7	ERROR	Enable Error Interrupt
6	CPU	Enable AR6003 CPU interrupt
5	AR6000_INT	Enable a copy of the AR6003 CPU interrupt to be sent to the host. This interrupt is normally serviced by the 3 CPU, and should typically be disabled at the host interface.
4	COUNTER	Enable counter interrupt
3:0	MBOX_DATA	Enable Rx Data Pending Interrupt in the corresponding MBOX

CPU Sourced Interrupt Status (CPU_INT_STATUS_ENABLE)

Offset: 0x419
Reset Value: 0xFF
Access: Read/Write

Enable bits for the CPU_INT_STATUS register. Each bit enables the corresponding bit in the CPU_INT_STATUS register. Bit values:

- 0 = interrupt is disabled
- 1 = interrupt is enabled

Bits	Name	Description	
7:0	BIT	Bit [7]	Enable interrupt #7
		Bit [6]	Enable interrupt #6
		Bit [5]	Enable interrupt #5
		Bit [4]	Enable interrupt #4
		Bit [3]	Enable interrupt #3
		Bit [2]	Enable interrupt #2
		Bit [1]	Enable interrupt #1
		Bit [0]	Enable interrupt #0

Error Interrupt Status (ERROR_STATUS_ENABLE)

Offset: 0x41A
Reset Value: 0x0
Access: Read/Write

Enable bits for the ERROR_STATUS register. Each bit enables the corresponding bit in the ERROR_STATUS register. Bit values:

- 0 = interrupt is disabled
- 1 = interrupt is enabled

Bits	Name	Description
7:3	RES	Reserved
2	WAKEUP	Wakeup interrupt enable
1	RXUNDERFLOW	RXUNDERFLOW interrupt enable
0	TXOVERFLOW	TXOVERFLOW interrupt enable

Credit Counter Interrupt Status (COUNTER_INT_STATUS_ENABLE)

Offset: 0x41B
Reset Value: 0x0
Access: Read/Write

Enable bits for the COUNTER_INT_STATUS register. Each bit enables the corresponding bit in the COUNTER_INT_STATUS register. Bit values:

- 0 = interrupt is disabled
- 1 = interrupt is enabled

Bits	Name	Description	
7:0	BIT	Bit [7]	Enable interrupt #7
		Bit [6]	Enable interrupt #6
		Bit [5]	Enable interrupt #5
		Bit [4]	Enable interrupt #4
		Bit [3]	Enable interrupt #3
		Bit [2]	Enable interrupt #2
		Bit [1]	Enable interrupt #1
		Bit [0]	Enable interrupt #0

Credit Counters Direct Access (COUNT)

Offset: 0x420
Reset Value: 0x0
Access: Read/Write

Ordinary read/write access to credit counter registers. Read/modify/write operations are not atomic. Reset value for all counters is 0. Address decode is:

Address	Description
[2:0] = 0x7	Accesses COUNT7
[2:0] = 0x6	Accesses COUNT6
[2:0] = 0x5	Accesses COUNT5
[2:0] = 0x4	Accesses COUNT4
[2:0] = 0x3	Accesses COUNT3
[2:0] = 0x2	Accesses COUNT2
[2:0] = 0x1	Accesses COUNT1
[2:0] = 0x0	Accesses COUNT0

Credit Counter Atomic Decrement (COUNT_DEC)

Offset: 0x440
Reset Value: 0x0
Access: Read/Write

Reading or writing to this register causes a unit decrement. Reads return the old value, then decrement. If the value of COUNT is 0, the decrement does not occur. If read returns a 0, software knows the decrement does not occur. Write data is ignored.

Registers are word aligned to allow 16-bit or 32-bit accesses from the host to read or write a single atomic register. Reads or writes to non-word aligned addresses have no effect.

Reset value for all counters is 0.

Address	Description
[4:0] = 0x1C	Decrements COUNT7
[4:0] = 0x18	Decrements COUNT6
[4:0] = 0x14	Decrements COUNT5
[4:0] = 0x10	Decrements COUNT4
[4:0] = 0xC	Decrements COUNT3
[4:0] = 0x8	Decrements COUNT2
[4:0] = 0x4	Decrements COUNT1
[4:0] = 0x0	Decrements COUNT0

Interface Scratch (SCRATCH)

Offset: 0x460
Reset Value: 0x0
Access: Read/Write

Eight scratch registers are available for host and local CPU read/write. These registers are not atomic, data is always from the last writer.

Reset value for all scratch registers is 0.

Address	Description
[4:0] = 0x1C	Scratch register 7
[4:0] = 0x18	Scratch register 6
[4:0] = 0x14	Scratch register 5
[4:0] = 0x10	Scratch register 4
[4:0] = 0xC	Scratch register 3
[4:0] = 0x8	Scratch register 2
[4:0] = 0x4	Scratch register 1
[4:0] = 0x0	Scratch register 0

FIFO Timeout Period (FIFO_TIMEOUT)

Offset: 0x468
Reset Value: 0xFF
Access: Read/Write

If a MBOX Rx FIFO is empty and a host read arrives, or a MBOX Tx FIFO is full and a host write arrives, the SDIO interface wait for this timeout period before declaring an error state. After an error is declared, all writes to full FIFOs are dropped, and all reads to empty FIFOs return garbage data (instead of waiting for data). After the error conditions have been cleared, a write to the ERROR_INT_STATUS clears the error condition and the FIFOs return to normal operation.

Bits	Name	Description
7:0	VALUE	Timeout value, in ms, when CORE_CLK=40 MHz, or in 0.5 ms when CORE_CLK=80 MHz. This bit should never be set to 0.

FIFO Timeout Enable (FIFO_TIMEOUT_ENABLE)

Offset: 0x469
Reset Value: 0x0
Access: Read/Write

Can be used to disable all timeout conditions.

Bits	Name	Description	
7:1	RES	Reserved	
0	SET	0	FIFO timeouts are disabled
		1	FIFO timeouts are enabled

Disable Sleep Mode (DISABLE_SLEEP)

Offset: 0x46A
Reset Value: 0x0
Access: Read/Write

Disable the AR6003 from entering SLEEP state.

Bits	Name	Description	
7:2	RES	Reserved	
1	FOR_INT	0	AR6003 may enter SLEEP when a host interrupt is pending
		1	AR6003 will never enter SLEEP when a host interrupt is pending
0	ON	0	AR6003 may enter SLEEP state
		1	AR6002 will never enter SLEEP state

Local Bust Host Interface (LOCAL_BUS)

Offset: 0x470
Reset Value: 0x0
Access: Read/Write

In local bus and SPI modes, this register is used to probe the chip state. Unlike other registers, this one can be read and written when the chip is in SLEEP state. This register cannot be written by the CPU, only by the host.

Bits	Name	Description	
7:2	RES	Reserved	
1:0	STATE	Returns the current chip state	
		0	SHUTDOWN
		1	ON
		2	SLEEP
		3	WAKEUP

AR6002 CPU Interrupt (INT_WLAN)

Offset: 0x472
Reset Value: 0x0
Access: Read/Write

The host may write to this register to interrupt to the AR6002 CPU. Software defines the meaning of each interrupt. Writes to this register will set interrupt bits, the AR6002 CPU must clear the bits. Writing a 1 sets the register field. Writing a 0 does not change the bit value. These bits are cleared by hardware.

Bits	Name	Description	
31:8	RES	Reserved	
7:0	VECTOR	Bit [7]	Interrupt 7 (write 1 to set)
		Bit [6]	Interrupt 6 (write 1 to set)
		Bit [5]	Interrupt 5 (write 1 to set)
		Bit [4]	Interrupt 4 (write 1 to set)
		Bit [3]	Interrupt 3 (write 1 to set)
		Bit [2]	Interrupt 2 (write 1 to set)
		Bit [1]	Interrupt 1 (write 1 to set)
		Bit [0]	Interrupt 0 (write 1 to set)

SPI Slave Interface Configuration (SPI_CONFIG)

Offset: 0x480
Reset Value:
See field descriptions
Access: Read/Write

Bits	Name	Access	Reset	Description	
7:5	RES	—	0x0	Reserved	
4	SPI_RESET	R/W	0x0	Controls the reset state of SPI interface	
				0	Normal operational mode
				1	Reset SPI core
3	INTERRUPT _ENABLE	R/W	0x0	Enables the SPI interface interrupt to propagate to AR6002 interrupt logic	
				0	SPI interrupt disabled
				1	SPI interrupt enabled
2	TEST_MODE	R/W	0x0	For test mode (Loopback) operation. When set, data received is transmitted back (echo) after 1 transaction delay.	
				0	SPI normal mode
				1	SPI test mode
1:0	DATA_SIZE	R/W	0x2	Selects the data size for SPI. Note: The address phase is always 16-bit. (Default = 32-bit)	
				0	8-bit data
				1	16-bit data
				2	32-bit data
				3	Reserved

Detailed Description of the SPI_CONFIG Fields

Name	Description
Test Mode	Enter Test (Loop back) mode This mode is for debug purposes only. This bit should be reset for normal operation. If set, all received bits on the SPI interface (on SPI_MOSI) are sent back to the Host on the SPI Interface (SPI_MISO).
Reset Mode	Assert active low reset This mode results in the reset of all state machines of the SPI slave. All operation registers (config, status, address, and count registers) retain the last value. This bit is auto-clearing.
Interrupt Enable	Interrupts are disabled
	Interrupts are enabled, resulting in any error conditions (e.g., IF error, ADDR Error, RD Error, and WR error) to assert the INTR output of SPI_SLV
Data Size	DATA8 All data phases are 8 bits in size
	DATA16 All data phases are maximum 16 bits in size. Allowable data sizes are 16 bits for internal registers access, 8 and 16bits for mailbox single reads and writes, and 16 bits for DMA transfers. 8 bits are also allowed for the last data phase of a DMA transaction
	DATA32 All data phases are maximum 32 bits in size. Allowable data sizes are 16-bit for internal registers access, 8-,16-, 24-, and 32-bit for mailbox single reads and writes, and 32-bit for DMA transfers. 8-,16-, and 24-bit is allowed for the last data phase of a DMA transaction

SPI Status (SPI_STATUS)

Offset: 0x481
Reset Value:
See field descriptions
Access:
See field descriptions

Bit	Name	Access	Reset	Description	
7:4	RES	—	0x0	Reserved	
3	ADDR_ERR	R/WC	0x0	0	No error
				1	Non-existent internal register address received OR: An 8- or 32-bit address/command phase received
2	RD_ERR	R/WC	0x0	0	No error
				1	Indicates read error occurred Can be cleared by writing to this register. A read error is indicated by a 16- or 32- SPI_CLK data phase occurring in a DATA8 mode read, or by a 32- SPI_CLK data phase occurring in DATA16 mode.
1	WR_ERR	R/WC	0x0	0	Indicates no write error
				1	Indicates write error occurred Can be cleared by writing a 1 to this register
0	READY	RO	0x1	0	Indicates current command pending
				1	Indicates current request completed Ready to accept SPI transaction

SDIO CIS Tuples Copy (CIS_WINDOW)

Offset: 0x600
Reset Value: 0x0
Access: Read/Write

This address space is a copy of the SDIO CIS tuples, the first tuple for CIS 0 begins at offset 0x0. The first tuple for CIS 1 (function 1 space) begins at offset 0x100. This space is read-only for general CIS access. CIS0 contains 32 bytes of programmable tuples, which begin after the last fixed tuple in CIS0. These tuples can be written by the MIPS CPU to pass configuration information to host drivers.

SPI Internal

Table F-2 summarizes the SPI internal registers, accessed rom the SPI host.

Table F-2. **SPI Internal Registers**

Offset	Name	Description	Page
0x0100	DMA_SIZE	DMA Size	page F-13
0x0200	WRBUF_SPC_AVA	Write Buffer Space Available	page F-13
0x0300	RDBUF_BYTE_AVA	Read Buffer Byte Available	page F-14
0x0400	SPI_CONFIG	SPI Configuration	page F-14
0x0500	SPI_STATUS	SPI Status	page F-15
0x0600	HOST_CTRL_BYTE_SIZE	Host Control Register Access Byte Size	page F-15
0x0700	HOST_CTRL_CONFIG	Host Control Register Configure	page F-15
0x0800	HOST_CTRL_RD_PORT	Host Control Register Read Port	page F-15
0x0A00	HOST_CTRL_WR_PORT	Host Control Register Write Port	page F-16
0x0C00	INTR_CAUSE	Interrupt Cause	page F-16
0x0D00	INTR_ENABLE	Interrupt Enable	page F-17
0x1200	RDBUF_WATERMARK	Read Buffer Water Mark	page F-17
0x1300	WRBUF_WATERMARK	Write Buffer Water Mark	page F-17
0x1400	RDBUF_LOOKAHEAD1	Read Buffer Lookahead 1	page F-18
0x1500	RDBUF_LOOKAHEAD2	Read Buffer Lookahead 2	page F-18

DMA Size (DMA_SIZE)

Offset: 0x0100
Reset Value: 0x1
Access: Read/Write

This register holds the DMA transfer size in terms of bytes.

Bits	Name	Description
15:12	RES	Reserved
11:0	DMA_SIZE	Indicates DMA transfer size (in bytes). Programming of zero is not allowed.

Write Buffer Space Available (WRBUF_SPC_AVA)

Offset: 0x0200
Reset Value: 0x0
Access: Read-Only

This register holds the total number of empty space within the write buffer.

Bits	Name	Description
15:11	RES	Reserved
10:0	SPACE_AVA	Indicates write buffer empty space (in bytes)

Read Buffer Byte Available (RDBUF_BYTE_AVA)

Offset: 0x0300
Reset Value: 0x0
Access: Read-Only

This register holds the total number of data bytes within the read buffer.

Bits	Name	Description
15:12	RES	Reserved
11:0	BYTE_AVA	Indicates the total number of data bytes within read buffer.

SPI Configuration (SPI_CONFIG)

Offset: 0x0400
Reset Value: 0x0
Access: Read/Write

This register is used to configure the SPI core.

Bits	Name	Description	
15	SPI_RESET	Asserting this bit will reset the SPI core. All data within the read or write buffer will be lost. This bit is auto-clear.	
14:11	RES	Reserved	
10:9	MISO_MUXEL	Select between the negedge (default) MISO flop or delay version of posedge MISO flop.	
8	SPI2MBOX _INTR_EN	Enable interrupt going from SPI core to Mailbox	
		0	No SPI interrupt will be generated to Mailbox
		1	Interrupt will be generated for error conditions and packet available within read buffer
7	SPI_IO _ENABLE	This is the SPI version of SDIO-CCCR.IOE1 bit. This bit should be reset to 0 after power on reset. The SPI host can program this bit to enable the mailbox and the rest of the chip.	
6:5	RES	Reserved	
4	KEEP_AWAKE_ FOR_INTR	0	The chip is allowed to go to sleep
		1	The chip may never enter SLEEP when a host interrupt is pending
3	KEEP_AWAKE_ EN	0	Allows the chip to go to sleep
		1	Wakes the chip and prevents it from going to sleep
2	SWAP	0	No swapping
		1	Byte swap
1	16BIT_MODE	If swap bit is zero, this bit has no effect. If swap bit is asserted, and: ■ 16 bit_mode = 1: swap every 2 bytes ■ 16 bit_mode = 0: swap every 4 bytes	
0	PREFETCH _MODE	Mailbox Rx FIFO prefetch scheme	
		0	Indicates strict priority (MBOX FIFO0 has highest priority, MBOX FIFO3 has lowest priority)
		1	Round robin

SPI Status (SPI_STATUS)

Offset: 0x0500
Reset Value: 0x0
Access: Read-Only

This register indicates the current status of the SPI core.

Bits	Name	Description	
15:3	RES	Reserved	
2:1	RFC_STATE	Current state of the chip	
		2'b00	Shutdown
		2'b01	On
		2'b10	Sleep
		2'b11	Wakeup
0	HOST_ACCESS _DONE	Indicates the host control register access (read or write) has finished	

Host Control Register Access Byte Size (HOST_CTRL_BYTE_SIZE)

Offset: 0x0600
Reset Value: See field
Access: Read/Write

This register contains the size of the host control register access in terms of bytes.

Bits	Name	Reset	Description	
15:7	RES	0x0	Reserved	
6	NO_ADDR _INCREMENT	0x0	0	Increment the host control address by 1 for every byte
			1	Address to the host control register stays the same for the entire transfer
5:0	BYTE_SIZE	0x1	Indicates the size of the host control register access. The maximum value is 32. Programming of zero is not allowed.	

Host Control Register Configure (HOST_CTRL_CONFIG)

Offset: 0x0700
Reset Value: See field
Access: Read/Write

This register holds the starting read address, direction, and enable bit for host control register access.

Bits	Name	Reset	Description	
15	ENABLE	0x0	Once this bit is asserted, the SPI core will read from or write to the host control register inside mailbox. This bit is auto-clear.	
14	DIRECTION	0x0	0	Host control register read
			1	Host control register write
13:0	ADDRESS	0x0400	The starting address of host control register access. This address should range from 0x0400 to 0x07FF.	

Host Control Register Read Port (HOST_CTRL_RD_PORT)

Offset: 0x0800

This register provides PIO access to the host control register read port. SPI host can issue a read command, with internal bit asserted and the address of this register, for retrieving host control read data.

Host Control Register Write Port (HOST_CTRL_WR_PORT)

Offset: 0x0A00

This register provides PIO access to the host control register write port. This register (port) is write only. SPI host can issue a write command, with internal bit asserted and the address of this register, for sending host control write data.

Interrupt Cause (INTR_CAUSE)

Offset: 0x0C00
Reset Value: 0x0
Access: See field

This register holds all the pending SPI interrupts. All interrupt bits can be cleared by programming a value of one, except mbox_interrupt, counter_interrupt, and local_cpu_interrupt.

- 0: No interrupt
- 1: Interrupt is pending

Bits	Name	Access	Description
15:11	RES	RW	Reserved
10	WBUF_BELOW _WATERMARK	RW/W1C	Indicates the number of data bytes within the write buffer is below the water mark level.
9	HOST_CTRL _RD_DONE	RW/W1C	Indicates the host control register read has completed. The read data can now be accessed through the HOST_CTRL_RD_PORT. Write one will clear the interrupt. Write zero has no effect.
8	HOST_CTRL_ WR_DONE	RW/W1C	Indicates the host control register write has completed. Write one will clear the interrupt. Write zero has no effect.
7	ALL_CPU _INTERRUPT	RO	Shadow copy of the mailbox to host interrupt. The sum of all the mailbox internal interrupts. Please refer to the mailbox specification. Will be cleared when the mailbox interrupt line is de-asserted.
6	CPU_ON	RW/W1C	CPU has just been awaken. Write one will clear the interrupt. Write zero has no effect.
5	COUNTER _INTERRUPT	RO	This is a shadow copy of the mailbox credit counter interrupt. One or more of the 8 mailbox credit counters transition from 0 to 1. Will be cleared when mailbox credit counters transition from 1 to 0.
4	LOCAL_CPU _INTERRUPT	RO	This is a shadow copy of the mailbox CPU interrupt line. One or more of the local CPU interrupt lines has been asserted. Clears when the host control register (CPU_INT_STATUS) clears.
3	ADDRESS _ERROR	RW/W1C	A DMA access with the out of bound address has occurred. Writing one clears the interrupt. Writing zero has no effect.
2	WBUF_ERROR	RW/W1C	Indicates an error has occurred while accessing the write buffer. Writing one clears the interrupt. Writing zero has no effect.
1	RDBUF _ERROR	RW/W1C	Indicates an error has occurred while accessing the read buffer. It means that the host is sending a DMA read in which the size is larger the total available bytes within the read buffer. Writing one will clear the interrupt. Writing zero has no effect.
0	PACKET _AVAILABLE	RW/W1C	Read buffer has at least one packet or the total number of available bytes has reached or exceeded the read buffer watermark. After a DMA read, HW clears this interrupt. Writing one will clear the interrupt. Writing zero has no effect.

Interrupt Enable (INTR_ENABLE)

Offset: 0x0D00
Reset Value: 0x0
Access: Read/Write

This register is used to mask/enable interrupts going to the SPI host.

- 0: Interrupt is disabled
- 1: Interrupt is enabled

Bits	Name	Description
15:11	RES	Reserved
10	WBUF_BELOW _WATERMARK	Enable the write buffer below watermark interrupt.
9	HOST_CTRL _RD_DONE	Enable the host control read done interrupt.
8	HOST_CTRL_ WR_DONE	Enable the host control write done interrupt.
7	ALL_CPU _INTERRUPT	Enable the all the CPU interrupts coming from mailbox.
6	CPU_ON	Enable the CPU on interrupt.
5	COUNTER _INTERRUPT	Enable the mailbox credit counter interrupt.
4	LOCAL_CPU _INTERRUPT	Enable the local CPU sourced interrupt.
3	ADDRESS _ERROR	Enable DMA address error interrupt.
2	WBUF_ERROR	Enable the write buffer error interrupt.
1	RDBUF _ERROR	Enable th3e read buffer error interrupt.
0	PACKET _AVAILABLE	Enable the packet available interrupt.

Read Buffer Water Mark (RDBUF_WATERMARK)

Offset: 0x1200
Reset Value: 0x1
Access: Read/Write

This register is holds the water mark value for the read buffer.

Bits	Name	Description
15:12	RES	Reserved
110	WATERMARK	Read buffer watermark. Zero is not allowed.

Write Buffer Water Mark (WRBUF_WATERMARK)

Offset: 0x1300
Reset Value: 0x0
Access: Read/Write

This register is holds the water mark value for the write buffer.

Bits	Name	Description
15:12	RES	Reserved
11:0	WATERMARK	Write buffer watermark. Zero is not allowed.

Read Buffer Lookahead 1 (RDBUF_LOOKAHEAD1)

Offset: 0x1400
Reset Value: N/A
Access: Read-Only

This register returns the first 2 bytes of the read buffer. Reading this register will not pop or change the contents of the read buffer, it is non-destructive. If the read buffer is empty, reading this register will return stale data. Writing to this register has no effect.

Bits	Name	Description
15:8	FIRST_BYTE	Contains the first byte from the head of the read buffer.
7:0	SECOND_BYTE	Contains the second byte from the head of the read buffer.

Read Buffer Lookahead 2 (RDBUF_LOOKAHEAD2)

Offset: 0x1500
Reset Value:
Access: Read/Write

This register returns the second 2 bytes of the read buffer. Reading this register will not pop or change the contents of the read buffer, it is non-destructive. If the read buffer is empty, reading this register will return stale data. Writing to this register has no effect.

Bits	Name	Description
15:8	THIRD_BYTE	Contains the third byte from the head of the read buffer.
7:0	FOURTH_BYTE	Contains the fourth byte from the head of the read buffer.

G

AP Bringup and Configuration in Linux

Bring Up the Chipset in AP Mode With MAC Bridging

This procedure describes bringing up the AR6000 chipset in AP mode on a Linux system. The Linux system should have bridge control module compiled in to the kernel or as a loadable module.

NOTE: In these commands, *eth0* is the Ethernet interface on the Linux system and *eth1* is the chipset wireless interface. Do not assign an IP address to *eth0* or *eth1*.

1. By default the AR6003 comes up in STA mode; use this command to export the WiFi interface as *eth1*.
   ```
   /host/support/loadAR6000.sh --mode ap
   ```
2. Configure the *eth0* and *eth1* interface.
   ```
   ifconfig eth0 0.0.0.0
   ifconfig eth1 0.0.0.0
   ```
3. Add the Ethernet interface to the bridge.
   ```
   brctl addbr br0brctl addif br0 eth0
   ```
4. Add the WiFi interface to the bridge.
   ```
   brctl addif br0 eth1
   ```
5. Configure the *br0* interface.
   ```
   ifconfig br0 up
   iwconfig eth1 mode Master
   iwconfig eth1 channel 1
   iwconfig eth1 essid AR6003AP
   ```
6. Commit the new settings.
   ```
   iwconfig eth1 commit
   ```

Once this process is complete, beacons are seen in the air. At this point, one can connect WiFi STAs to the AR6003AP and ping the AP and other STAs on the Ethernet backbone or on the BSS.

Configure the AR6000 Chipset AP in Open Mode

Configure the chipset AP in open mode using one of two sets of commands:

-
  ```
  /host/support/loadAR6000.sh --mode ap
  iwconfig eth1 channel 1
  iwconfig eth1 essid AR6003AP
  iwconfig eth1 commit
  ```

Or:

-
  ```
  /host/support/loadAR6000.sh --mode ap
  cd host/.output/ LOCAL_i686-SDIO/image/
  ./hostapd ar6k.conf
  ```

NOTE: Edit **ar6k.conf** to change the interface name, SSID, and channel.

Configure Security on the AR6000 Chipset AP

WEP Security

Use these commands to configure WEP security on the chipset AP:

```
1.  ./host/support/loadAR6000.sh --mode ap
2.  iwconfig eth1 enc 0123456789
3.  iwconfig eth1 channel 1
4.  iwconfig eth1 essid AR6003AP
5.  iwconfig eth1 commit
```

Shared Key Authentication

Use these commands to configure shared authentication on the chipset AP:

```
1.  ./host/support/loadAR6000.sh --mode ap
2.  iwconfig eth1 enc 0123456789 restricted
3.  iwconfig eth1 channel 1
4.  iwconfig eth1 essid AR6003AP
5.  iwconfig eth1 commit
```

WPA (TKIP) Security

Use these commands to configure WPA (TKIP) security on the chipset AP:

1. `./host/support/loadAR6000.sh --mode ap`
2. `cd host/.outpu/ LOCAL_i686-SDIO/image/`
3. `./hostapd ar6k-wpa.conf`

NOTE: Edit **ar6k-wpa.conf** to change the interface name, SSID, channel, and passphrase.

WPA2 (AES) Security

Use these commands to configure WPA2 (AES) security on the chipset AP:

1. `./host/support/loadAR6000.sh --mode ap`
2. `cd host/.output/LOCAL_i686-SDIO/image/`
3. `./hostapd ar6k-wpa2.conf`

NOTE: Edit **ar6k-wpa2.conf** to change the interface name, SSID, channel, and passphrase.

wmiconfig Application Commands for AP mode

Refer to "Wireless Module Interface (WMI)" on page A-1 for descriptions and usage of all AP mode WMI commands.

Switch Modes Between AP and STA on the AR6000 Chipset

By default, when the script **loadAR6000.sh** loaded, the chipset comes up in STA mode. To change the default startup mode, use the `--mode` parameter of **loadAR6000.sh**:

Command	Description
Use the command: `loadAR6000.sh --mode ap`	Start the chipset in AP mode
Use the one of two commands: `loadAR6000.sh --mode sta` Or: `loadAR6000.sh`	Start the chipset in STA mode

Switch from STA to AP mode

To switch from STA mode to AP mode, set the mode and list the channel and SSID:

```
iwconfig eth1 mode Master
iwconfig eth1 channel 1
iwconfig eth1 essid AR6K-AP
iwconfig eth1 commit
```

Once the `commit` command is given, the chipset switches to AP mode and beacons can be seen in the air.

NOTE: If configuring the AP using hostAPd, then run the hostAPd application to switch the chipset to AP mode; hostApd automatically provides all needed commands to the chipset.

Switch from AP to STA Mode

To switch from AP mode to STA mode, set the mode and list the SSID:

```
iwconfig eth1 mode Managed
iwconfig eth1 essid WiFi
```
(to connect to a BSS named WiFi)

H
Linux APIs

This section describes the APIs the Linux reference host software exposes to user-space applications. Most map directly to WMI commands and events. User-space applications can configure the AR6001 WLAN software by calling into the specified supported IOCTLs. These applications can also receive event notifications from the AR6000 software over supported Netlink APIs.

$WORKAREA/host/os/linux/ar6000_drv.c is organized into separate files:

File	Description
ar6000_drv.c	Kernel driver entry points; drives initialization and cleanup routines
ioctl.c	Handles raw interface IOCTLs and dispatches wireless extension IOCTLs
wireless_ext.c	Handles Linux wireless extension IOCTLs
netbuf.c	Network buffer management
ar6000_raw_if.c	Raw HTC interface (reflashing and testing)

IOCTLs Support and Organization

The AR6001 driver support standard Linux wireless extensions and private IOCTLs. Private IOCTLs have been further extended to support growing application needs. Table H-1 shows the Linux wireless extension IOCTLs.

Table H-1. **Linux Wireless Extension IOCTLs**

IOCTL	Description	Page
SIOCGIWENCODE	Retrieves the WEP key for the AR6001	page H-8
SIOCSIWENCODE	Sets the WEP key for the AR6001	page H-7
SIOCGIWESSID	Retrieves the SSID of the network	page H-5
SIOCSIWESSID	Sets the SSID of the network	page H-5
SIOCGIWFREQ	Retrieves the frequency of the wireless device	page H-3
SIOCSIWFREQ	Sets the frequency for the wireless device operate on	page H-3
SIOCGIWMODE	Retrieves the mode of operation	page H-3
SIOCSIWMODE	Sets the mode of operation	page H-3
SIOCGIWNAME	Retrieves the name of the network device	page H-3
SIOCGIWRANGE	Retrieves network device wireless information	page H-4
SIOCGIWRETRY	Retrieves max retry value from failure frames (in software version 1.2 or higher.)	page H-8
SIOCSIWRETRY	Sets max retry value from failure frames (in software version 1.2 or higher).	page H-8
SIOCGIWAP	Retrieves the desired BSSID	page H-4
SIOCSIWAP	Sets the desired BSSID for the CONNECT command	page H-4
SIOCGIWRATE	Retrieves the transmission rate of the AR6001	page H-6
SIOCSIWRATE	Fixes the transmission rate of the AR6001	page H-6
SIOCGIWSCAN	Retrieves the list of APs from the SCAN operation	page H-5
SIOCSIWSCAN	Executes the SCAN command in the AR6001	page H-4
SIOCGIWTXPOW	Retrieves the transmission rate of the AR6001	page H-7
SIOCSIWTXPOW	Sets the transmission rate of the AR6001	page H-6

Name	**SIOCGIWNAME**
Description	Retrieves the name of the network device

Commands	`iwconfig eth1`
	`iwgetid`

Argument Type	`char *`
Arguments	Name of the wireless device
IOCTL Type	`get`
Class	Wireless extension

Name	**SIOCSIWFREQ**	
Description	Sets the frequency on which the wireless device should operate	
Commands	**`iwlist eth1`**	**`freq <freq|channel>`**
	`iwconfig eth1`	**`channel <freq|channel>`**
Argument Type	`struct iw_freq *`	
Arguments	Defined in **linux/wireless.h**	
IOCTL Type	`set`	
Class	Wireless extension	

Name	**SIOCGIWFREQ**	
Description	Retrieves the frequency on which the wireless device is operating if the device is connected. If the device is not connected, this command retrieves the range of frequencies in which the device can operate.	
Commands	**`iwlist eth1`**	`freq`
	`iwconfig eth1`	`channel`
Argument Type	`struct iw_freq *`	
Arguments	Defined in **linux/wireless.h**	
IOCTL Type	`get`	
Class	Wireless extension	

Name	**SIOCSIWMODE**	
Description	Sets the mode of operation	
Commands	**`iwconfig eth1`**	**`mode {ad-hoc | managed | 1 | 2}`**
Argument Type	`unsigned int *`	
Arguments	■ 1 = Ad hoc mode	
	■ 2 = Infrastructure mode	
IOCTL Type	`set`	
Class	Wireless extension	

Name	**SIOCGIWMODE**
Description	Retrieves the mode of operation
Commands	**`iwconfig eth1`**
Argument Type	`unsigned int *`
Arguments	■ 1 = Ad hoc mode
	■ 2 = Infrastructure mode
IOCTL Type	`get`
Class	Wireless extension

Name	**SIOCGIWRANGE**
Description	Retrieves the various wireless information related to the network device

Commands	**`iwconfig eth1`**
	`iwlist eth1        freq`
	`iwlist eth1        channel`

Argument Type	`struct iw_point *, struct iw_range *`
Arguments	`1. struct iw_point` Defined in **linux/wireless.h** Length = size of argument 2 `2. struct iw_range` Defined in **linux/wireless.h** Various device information
IOCTL Type	`get`
Class	Wireless extension

Name	**SIOCSIWAP**
Description	Sets the desired BSSID for the CONNECT command
Commands	**`iwconfig eth1      ap {<BSSID>|any|off}`**
Argument Type	`struct sockaddr *`
Arguments	MAC address of the BSSID
IOCTL Type	`get`
Class	Wireless extension

Name	**SIOCGIWAP**
Description	Retrieves the desired BSSID for the CONNECT command
Commands	**`iwconfig eth1`**
Argument Type	`struct sockaddr *`
Arguments	MAC address of the BSSID
IOCTL Type	`get`
Class	Wireless extension

Name	**SIOCSIWSCAN**
Description	Executes the SCAN command in the AR6001
Commands	**`iwlist eth1        scan`**
Argument Type	None
Arguments	None
IOCTL Type	`set`
Class	Wireless extension

Name	**SIOCGIWSCAN**
Description	Retrieves the list of APs from the SCAN operation
Commands	`iwlist eth1 scan`

Argument Type	`struct ar_giwscan_param {`
	`char *current_ev;`
	`char *end_buf;`
	`A_BOOL firstPass;`
	`} *`

Arguments	`current_ev` — Beginning of the buffer for the scan data `end_buf` — End of the buffer for the scan data
IOCTL Type	`get`
Class	Wireless extension

Name	**SIOCSIWESSID**
Description	Sets the SSID of the network the AR6001 should connect to. Usually, the WMI_CONNECT_CMD is called. However, when the STA is already associated to an AP and the SSID to be used is the same as the AP's (i.e., the same as the previous setting), then the WMI_RECONNECT_CMD is called.
Commands	`iwconfig eth1 essid {<ESSID>|any|off}` Where ESSID is the ESSID of the AP the STA wants to connect to
Argument Type	`struct iw_point *, char *`
Arguments	1. `struct iw_point` Defines the SSID length Length = size of argument 2 2. String representation of the SSID
IOCTL Type	`set`
Class	Wireless extension

Name	**SIOCGIWESSID**
Description	Retrieves the ESSID of the network the AR6001 is connected to or wants to connect to
Commands	`iwconfig eth1`
Argument Type	`struct iw_point *, char *`
Arguments	1. `struct iw_point` Defines the ESSID length Length = size of argument 2 2. String representation of the ESSID
IOCTL Type	`get`
Class	Wireless extension

Name	**SIOCSIWRATE**	
Description	Fixes the transmission rate of the AR6001	
Commands	`iwconfig eth1`	`rate <rate>`
Argument Type	`struct iw_param *`	
Arguments	Defined in **linux/wireless.h**	
	`value`	Transmit (Tx) rate in bps
	`fixed`	■ TRUE = Software fixes this rate ■ FALSE = Allow hardware to auto-select the rate
IOCTL Type	`set`	
Class	Wireless extension	

Name	**SIOCGIWRATE**	
Description	Retrieves the current transmission rate of the AR6001	
Commands	`iwconfig eth1`	`rate`
	`iwlist eth1`	`rate`
Argument Type	`struct iw_param *`	
Arguments	Defined in **linux/wireless.h**	
	`value`	Transmit (Tx) rate in bps
	`fixed`	■ TRUE = Software fixes this rate ■ FALSE = Allow hardware to auto-select the rate
IOCTL Type	`get`	
Class	Wireless extension	

Name	**SIOCSIWTXPOW**	
Description	Sets the power of transmitted packets in AR6001	
Commands	`iwconfig eth1`	`txpower {txpower in dbM | txpower in mW | auto} {[fixed | auto]}`
Argument Type	`iw_param *`	
Arguments	Defined in **linux/wireless.h**	
	`value`	Transmit (Tx) rate in dBm
	`disabled`	■ TRUE = Featured not enabled ■ FALSE = The AR6001 wishes to fix the Tx power
	`fixed`	■ TRUE = Software fixes this rate ■ FALSE = Allow hardware to auto-select the rate
	`flags`	Units of Tx power value ■ IW_TXPOW_DBM = In dBm ■ IW_TXPOW_MWATT = Not supported
IOCTL Type	`set`	
Class	Wireless extension	

Name	**SIOCGIWTXPOW**
Description	Retrieves the power of transmitted packet in the AR6001
Commands	**`iwlist eth1          txpower`**
Argument Type	`struct iw_param *`
Arguments	Defined in **linux/wireless.h**

`value`	Transmit (Tx) rate in dBm
`disabled`	■ TRUE = Featured not enabled ■ FALSE = The AR6001 wishes to fix the Tx power
`fixed`	■ TRUE = Software fixes this rate ■ FALSE = Allow hardware to auto-select the rate
`flags`	Units of Tx power value ■ IW_TXPOW_DBM = In dBm

IOCTL Type	`get`
Class	Wireless extension

Name	**SIOCSIWENCODE**
Description	Sets the WEP key for the AR6001
Commands	**`iwconfig eth1     key [key index] <k>`**
Argument Type	`struct iw_point *, char *`
Arguments	`struct iw_point defines the WEP encryption method and key length`

`1. flags`	`IW_ENCODE_DISABLED`	No encryption
	`IW_ENCODE_ENABLED`	Encryption enabled
	`IW_ENCODE_OPEN`	Accept non-encoded packets
	`IW_ENCODE_RESTRICTED`	Refuse non-encode packets
	`IW_ENCODE_INDEX`	Key index
	`IW_ENCODE_NOKEY/` `IW_ENCODE_TEMP`	Not supported
`length`	5, 13, or 16	Length of valid ASCII WEP key

2. String representation of the WEP key (ASCII)

IOCTL Type	`set`
Class	Wireless extension

Name	**SIOCGIWENCODE**
Description	Retrieves the WEP key for the AR6001
Commands	`iwlist eth1 key`
Argument Type	`struct iw_point *, char *`
Arguments	`struct iw_point defines the WEP encryption method and key length`

`1. flags`	`IW_ENCODE_DISABLED`	No encryption
	`IW_ENCODE_ENABLED`	Encryption enabled
	`IW_ENCODE_OPEN`	Accept non-encoded packets
	`IW_ENCODE_RESTRICTED`	Refuse non-encode packets
	`IW_ENCODE_INDEX`	Key index
`length`	5, 13, or 16	Length of valid ASCII WEP key

2. String representation of the WEP key (ASCII)

IOCTL Type	`get`
Class	Wireless extension

Name	**SIOCSIWRETRY**
Description	Sets the maximum retry value from the failure frames. Supported in software version 1.2 or higher.
Commands	`iwconfig eth1 retry <R>`
Argument Type	`struct iw_param *, char *`
Arguments	`struct iw_param` Defines the value of retry number

`value`	Retry limit	
`disabled`	■ TRUE = Featured not enabled ■ FALSE = The AR6001 wishes to fix the retry limit	
`flags`	`IW_RETRY_LIMIT`	Maximum number of retries
	`IW_RETRY_LIFETIME`	Maximum duration of retries in µs
	`IW_RETRY_MIN`	The value is a minimum
	`IW_RETRY_MAX`	The value is a maximum

IOCTL Type	`set`
Class	Wireless extension

Name	**SIOCGIWRETRY**
Description	Retrieves the maximum retry value from the failure frames. Supported in software version 1.2 or higher.
Commands	`iwlist eth1 retry`
Argument Type	`struct iw_param *, char *`
Arguments	`struct iw_param` Defines the value of retry number

`value`	Retry limit	
`disabled`	■ TRUE = Featured not enabled ■ FALSE = The AR6001 wishes to fix the retry limit	
`flags`	`IW_RETRY_LIMIT`	Maximum number of retries
	`IW_RETRY_LIFETIME`	Maximum duration of retries in µs
	`IW_RETRY_MIN`	The value is a minimum
	`IW_RETRY_MAX`	The value is a maximum

IOCTL Type	`get`
Class	Wireless extension

Private IOCTLs (SIOCWFIRSTPRIV)

The AR6001 driver supports standard Linux wireless extensions and private IOCTLs. Private IOCTLs have been further extended to support growing application needs. Table H-2 shows the Linux private IOCTLs.

Table H-2. **Linux Private IOCTLs**

IOCTL	Description	Page
IEEE80211_IOCTL_ADDPMKID	Adds a PMKID to the pmkid cache	page H-11
IEEE80211_IOCTL_DELKEY	This command is not supported. (The keys are deleted when the STA disconnects from an AP. When an ADDKEY IOCTL is issued, the new keys are used.)	—
IEEE80211_IOCTL_GETOPTIE	This command is not supported.	—
IEEE80211_IOCTL_GETPARAM	This command is not supported.	—
IEEE80211_IOCTL_GETWMMPARAM	This command is not supported.	—
IEEE80211_IOCTL_SETAUTHALG	This command is not supported.	—
IEEE80211_IOCTL_SETKEY	Configures the key	page H-10
IEEE80211_IOCTL_SETMLME	This command is not supported.	—
IEEE80211_IOCTL_SETOPTIE	This command is not supported.	—
IEEE80211_IOCTL_SETPARAM	Sets WPA/WPA2 configuration parameters	page H-10
IEEE80211_IOCTL_SETWMMPARAMS	This command is not supported.	—
AR6000_IOCTL_WMI_CREATE_QOS	Creates a new prioritized data endpoint	page H-16
AR6000_IOCTL_WMI_DELETE_QOS	Deletes a prioritized data endpoint	page H-18
AR6000_IOCTL_WMI_GETREV	Retrieves the revision/version numbers	page H-11
AR6000_IOCTL_WMI_GET_QOS_QUEUE	Returns the queue number to the traffic class	page H-16
AR6000_IOCTL_WMI_GET_TARGET_STATS	Request statistics from the AR6001	page H-19
AR6000_IOCTL_WMI_SETBSSFILTER	Requests receive notification	page H-13
AR6000_IOCTL_WMI_SETLISTENINT	Requests to listen for travel at this interval	page H-13
AR6000_IOCTL_WMI_SETPWR	Sets to maximum power-saving or performance	page H-11
AR6000_IOCTL_WMI_SETSCAN	Sets the scan parameters	page H-12
AR6000_IOCTL_WMI_SET_ACCESS_PARAMS	Sets access parameters for the wireless network	page H-20
AR6000_IOCTL_WMI_SET_ASSOC_INFO	Specifies information elements to add to all future association and reassociation requests	page H-19
AR6000_IOCTL_WMI_SET_BADAP	Configures the AR6001 to avoid certain APs	page H-16
AR6000_IOCTL_WMI_SET_BMISS_TIME	Sets the BMISS time in the AR6001	page H-20
AR6000_IOCTL_WMI_SET_CHANNELPARAMS	Sets to wireless mode and limits channels	page H-14
AR6000_IOCTL_WMI_SET_DISC_TIMEOUT	Configures the amount of time to spend reestablishing a connection	page H-20
AR6000_IOCTL_WMI_SET_ERROR_REPORT _BITMASK	Control ERROR_REPORT	page H-18
AR6000_IOCTL_WMI_SET_IBSS_PM_CAPS	Supports a non-standard ad hoc power saving management scheme	page H-21
AR6000_IOCTL_WMI_SET_PMPARAMS	Configures power parameters	page H-15
AR6000_IOCTL_WMI_SET_PROBEDSSID	Lists SSIDs to look for. Changed in 2.0 to support multiple SSIDs.	page H-14
AR6000_IOCTL_WMI_SET_SNRTHRESHOLD	Configures monitoring and reporting SNR of the connected BSS, used as a link quality metric	page H-21

Name	**IEEE80211_IOCTL_SETPARAM**
Description	Sets the various WPA/WPA2 configuration parameters
Commands	**`iwpriv eth1 setparam <Param> <Value>`**
Argument Type	32-bit int

Arguments

1. Parameter types

`IEEE80211_PARAM_AUTHMODE (3)`	Authentication Mode
`IEEE80211_PARAM_MCASTCIPHER (5)`	Multicast Cipher
`IEEE80211_PARAM_UCASTCIPHER (8)`	Unicast Cipher
`IEEE80211_PARAM_WPA (10)`	WPA mode (0,1,2)
`IEEE80211_PARAM_COUNTERMEASURES (14)`	WPA/TKIP countermeasures

2. Parameter values:

`AUTHMODE`	`= 6`	WPA PSK
`MCASTCIPHER`	`= 1`	TKIP
	`= 3`	AES-CCM
`UCASTCIPHER`	`= 1`	TKIP
	`= 3`	AES-CCM
`WPA`	`= 1`	WPA1 only
	`= 2`	WPA2 only
`COUNTERMEASURES`	`= 0`	Disable countermeasures
	`= 1`	Enable countermeasures

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**IEEE80211_IOCTL_SETKEY**
Description	Configures the key
Commands	**`iwpriv eth1 setkey <type> <keyindex> <key length> <flags> <Mac address> <key rsc> <key tsc> <key data>`**

Argument Type

```
struct ieee80211req_key {
unsigned char       ik_type;            Key/cipher type
unsigned char       ik_pad;
unsigned short      ik_keyix;           Key index
unsigned char       ik_keylen;          Key length in bytes
unsigned char       ik_flags;
unsigned char       ik_macaddr[6];
unsigned long       ik_keyrsc;          Key Rx sequence counter
unsigned char       ik_keytsc;          Key Tx sequence counter
unsigned char       ik_keydata[32];
}
```

Arguments

`ik_type`	`= 0`	WEP
	`= 1`	TKIP
	`= 3`	AES_CCM
`ik_flags`	`= 0x01`	Tx key
	`= 0x02`	Rx key
	`= 0x80`	Default key

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**IEEE80211_IOCTL_ADDPMKID**
Description	Adds a PMKID to the wireless module's pmkid cache
Commands	**`iwpriv eth1 addpmkid <BSSID> <Pmkid capability> <PMKID>`**

Argument Type	`struct ieee80211req_addpmkid {`
	`unsigned char     pi_bssid[6];`
	`unsigned char     pi_enable;`
	`unsigned char     pi_pmkid[16];`
	`};`

Arguments	`pi_bssid` — MAC address of the AP the pmkid corresponds to `pi_enable = 1` — Sets the pmkid `pi_pmkid` — pmkid to add
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_IOCTL_WMI_GETREV**
Description	Retrieves the revision/version numbers of the Host driver and the Target firmware
Commands	**`wmiconfig eth1 --version`**

Argument Type	`struct ar6000_version {`
	`unsigned int      host_ver;`
	`unsigned int      target_ver;`
	`};`

Arguments	`host_ver` — Host driver version number `target_ver` — Target firmware's version number
IOCTL Type	`get`
Class	iwpriv (private IOCTLs)

Name	**AR6000_IOCTL_WMI_SETPWR**
Description	Configures the AR6001 to maximum power-saving or max performance mode
Commands	**`wmiconfig --power <mode>`**

Argument Type	`struct {`
	`A_UINT8           powerMode;` WMI_POWER_MODE
	`} WMI_POWER_MODE_CMD;`

Arguments	`powerMode` `REC_POWER = 0x01` — Recommended setting `MAX_PERF = 0x02` — Maximum performance at the potential
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_IOCTL_WMI_SETSCAN**
Description	The host uses this command to set the scan parameters in the AR6001, including the duty cycle for both foreground and background scanning
Commands	`wmiconfig eth1 --scan --fgstart=<sec> --fgend=<sec> --bg=<sec> --act=<msec> --pas=<msec> --sr=<short scan ratio> --scanctrlflags <connScan> <scanConnected> <activeScan> <reportBSSINFO>`
Argument Type	`struct {`

`A_UINT16`	`fg_start_period;`	Seconds
`A_UINT16`	`fg_end_period;`	Seconds
`A_UINT16`	`bg_period;`	Seconds
`A_UINT16`	`act_chdwell_time;`	ms
`A_UINT16`	`pas_chdwell_time;`	ms
`A_UINT8`	`shortScanRatio;`	Number of short scans per one long
`A_UINT8`	`scanCtrlFlags;`	Flags about how to scan

`} WMI_SCAN_PARAMS_CMD;`

Arguments

`fg_start_period`	First interval used by the AR6001 when it disconnects from an AP
`fg_end_period`	Maximum interval the AR6001 waits between foreground scans
`bg_period`	Background scan period
`act_chdwell_time`	Period of time the AR6001 stays on a particular channel while active scanning
`pas_chdwell_time`	Period of time the AR6001 stays on a particular channel while passive scanning
`shortScanRatio`	Number of short scans performed for each long scan

`scanCtrlFlags =`	`0x01`	Enable scan in connect command
	`0x02`	Enable scan the SSID AR6001 connecting with
	`0x04`	Enable active scan
	`0x08`	Enable scan when signal is low or BMISS
	`0x10`	Enable the report of BSSINFO

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_IOCTL_WMI_SETLISTENINT**
Description	Requests that the AR6001 wake up and listen for travel at this interval

Commands	**`wmiconfig eth1 --listen=<#of TUs, ranging from 15 to 3000>`**
	`wmiconfig eth1 --listenbeacons=<#of beacons, ranging from 1 to 50>`

Argument Type	`struct {`
	`A_UINT16          listenInterval;`
	`A_UINT8           numBeacons;`
	`} WMI_LISTEN_INT_CMD;`

Arguments	Listen interval value in 1024 µs or number of beacons
	`struct {`
	`A_UINT16 ;`
	`A_UINT16 ;`
	`}`

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_IOCTL_WMI_SETBSSFILTER**
Description	Requests that the AR6001 receive notification when it finds certain wireless networks
Commands	**`wmiconfig eth1 --filter=<filter>`**
Argument Type	Unsigned char

Arguments	`BSS filter type:`
	`{`

`NONE_BSS_FILTER     = 0x0,`	No beacons forwarded
`ALL_BSS_FILTER,`	All beacons forwarded
`PROFILE_FILTER,`	Only beacons matching the profile
`ALL_BUT_PROFILE_ FILTER,`	All beacons except the ones matching the profile
`CURRENT_BSS_FILTER,`	Only beacons matching the current BSS
`ALL_BUT_BSS_FILTER,`	All beacons except the ones matching the BSS
`PROBED_SSID_FILTER,`	Beacons matching the probed SSID
`LAST_BSS_FILTER,`	Marker only
`} WMI_BSS_FILTER;`	

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_IOCTL_WMI_SET_CHANNELPARAMS**
Description	Configures the AR6001 to a particular wireless mode and limits the channels it can operate on
Commands	**`wmiconfig eth1 --wmode=<mode> <list>`**

Argument Type

```
struct {
A_UINT8          phyMode;           Wireless mode
A_UINT8          numChannels;       Number of channels that follow
A_UINT16         channelList[1];    Channels in MHz
} WMI_CHANNEL_PARAMS_CMD;
```

Arguments

phyMode	= 0x01	802.11a
	= 0x02	802.11g
	= 0x03	802.11ag
	= 0x04	802.11b
	= 0x05	802.11g only
`numchannels`		Total number of channels to consider
`channelList`		Array of channels to consider

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_IOCTL_WMI_SET_PROBEDSSID**
Description	The host uses this command to provide a list of up to MAX_PROBED_SSID_INDEX (six) SSIDs that the AR6000 should actively look for. It lists the active SSID table. By default, the AR6000 actively look for only the SSID specified in the **CONNECT** command, and only when the regulatory domain allows active probing. With this command, specified SSIDs are also probed for, even if the SSIDs are hidden.
Commands	**`wmiconfig eth1 --ssid=<ssid> [--num=<index>]`**

Argument Type

{

`A_UINT8`	`numSsids`	A number from 0 to MAX_PROBED_SSID_INDEX indicating the active SSID table entry index for this command. If the specified entry index already has an SSID, the SSID specified in this command replaces it.
`WMI_PROBED_SSID_ INFO`	`probedSSID[1]`	

`} WMI_PROBED_SSID_CMD;`

{

`A_UINT8`	`flag`	A WMI_SSID_FLAG indicating the current disposition of an entry in the active SSID table
`A_UINT8`	`ssidLength`	The length of the specified SSID in bytes. If the length is 0 then the entry corresponding to the index is erased
`A_UINT8`	`ssid[32]`	An SSID string that is actively probed for when permitted by the regulatory domain

`} WMI_PROBED_SSID_INFO`

Command Values	{	
	`DISABLE_SSID_FLAG = 0,`	Disables entry
	`SPECIFIC_SSID_FLAG = 0x01,`	Probes specified SSID
	`ANY_SSID_FLAG = 0x02,`	Probes for any SSID
	} WMI_SSID_FLAG;	
Arguments	`entryIndex`	Active SSID table entry index
	`flag`	Specific action for the SSID
	`ssidLength`	Length of SSID in bytes
	`ssid`	SSID to probe
IOCTL Type	`set`	
Class	iwpriv (private IOCTLs)	

Name	**AR6000_IOCTL_WMI_SET_PMPARAMS**		
Description	Configures the power parameters in the AR6001		
Commands	**`wmiconfig eth1 --pmparams --it=<msec> --np=<number of PS POLL> --dp=<DTIM policy: ignore/normal/stick>`**		
Argument Type	`struct {`		
	`A_UINT16`	`idle_period;`	In ms
	`A_UINT16`	`pspoll_number;`	
	`A_UINT16`	`dtim_policy;`	
	`} WMI_POWER_PARAMS_CMD;`		
Arguments	`idle_period`	Length of time the AR6001 remains awake after network traffic (in ms)	
	`pspoll_number`	Number of PS-POLLs the AR6001 sends before it notifies the AP that it is awake	
	`dtim_policy`	`0x01`	IGNORE_DTIM
		`0x02`	NORMAL_DTIM, follow listen interval
		`0x03`	STICK_DTIM, receive all CAB traffic
IOCTL Type	`set`		
Class	iwpriv (private IOCTLs)		

Name	**AR6000_IOCTL_WMI_SET_BADAP**
Description	Configures the AR6001 to avoid certain APs
Commands	**`wmiconfig eth1 --badAP=<macaddr> [--num=<index>]`**
Argument Type	`struct {`
	`A_UINT8` `badApIndex;` `0 to WMI_MAX_BAD_AP_INDEX`
	`A_UINT8` `bssid [ATH_MAC_LEN];`
	`} WMI_ADD_BAD_AP_CMD`
Arguments	`badApIndex` Entry index
	`bssid` MAC address of the AP that should be avoided
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_IOCTL_WMI_GET_QOS_QUEUE**		
Description	Returns the active traffic streams (TSIDs) within a given traffic class		
Commands	**`wmiconfig -i eth1 --qosqueue <trafficclass>`**		
Argument Type	`struct ar6000_queuereq {`		
	`A_UINT8`	`trafficClass;`	
	`A_UINT16`	`activeTsids;`	
	`}`		
Arguments	`trafficClass`	`0`	Best effort
		`1`	Background
		`2`	Video
		`3`	Voice
	`activeTsids`		Active Tspecs within this trafficClass (return parameter)
IOCTL Type	`get`		
Class	iwpriv (private IOCTLs)		

Name	**AR6000_IOCTL_WMI_CREATE_QOS**		
Description	Creates a new prioritized data endpoint. Performs TSPECs, if the AP requires it by setting the ACM bit in WMM.		
Commands	**`wmiconfig eth1 --createqos <user priority> <direction> <traffic class> <trafficType> <voice PS capability> <min service interval> <max service interval> <inactivity interval> <suspension interval> <service start time> <tsid> <nominal MSDU> <max MSDU> <min data rate> <mean data rate> <peak data rate> <max burst size> <delay bound> <min phy rate> <sba> <medium time> <nominalPHY>`**		
Argument Type	`struct {`		
	`A_UINT32`	`minServiceInt;`	In ms
	`A_UINT32`	`maxServiceInt;`	In ms
	`A_UINT32`	`inactivityInt;`	In ms
	`A_UINT32`	`suspensionInt;`	In ms
	`A_UINT32`	`serviceStartTime;`	

`A_UINT32`	`minDataRate;`	In bps
`A_UINT32`	`meanDataRate;`	In bps
`A_UINT32`	`peakDataRate;`	In bps
`A_UINT32`	`maxBurstSize;`	
`A_UINT32`	`delayBound;`	
`A_UINT32`	`minPhyRate;`	In bps
`A_UINT32`	`sba;`	
`A_UINT32`	`mediumTime;`	
`A_UINT16`	`nominalMSDU;`	In octets
`A_UINT16`	`maxMSDU;`	In octets
`A_UINT8`	`trafficClass;`	
`A_UINT8`	`trafficType;`	TRAFFIC_TYPE
`A_UINT8`	`trafficDirection;`	TRAFFIC_DIR
`A_UINT8`	`voicePSCapability;`	VOICEPS_CAP_TYPE

... AR6000_IOCTL_WMI_CREATE_QOS, continued

`A_UINT8`	`tsid;`	
`A_UINT8`	`userPriority;`	802.1D user priority
`A_UINT8`	`nominalPHY;`	Nominal PHY rate for CCX TSRS IE support

```
} POSTPACK WMI_CREATE_PSTREAM_CMD;
```

Arguments

`user priority`	802.1D user priority range: 0-7
`direction`	0 = Tx (uplink) traffic, 1 = Rx (downlink) traffic, 2 = Bi-directional traffic
`traffic class`	1 = BK, 2 = VI, 3 = VO
`trafficType`	0 = Aperiodic, 1 = Periodic
`voice PS capability`	Specifies whether the voice power save mechanism (APSD if AP supports it or legacy/simulated APSD [using PS-POLL]) should be used 0 = Disable voice power save for this traffic class 1 = Enable APSD voice power save for this traffic class 2 = Enable voice power save for ALL traffic classes
`min service interval`	In ms
`max service interval`	In ms
`inactivity interval`	In ms (0 = Infinite inactivity interval)
`suspension interval`	In ms
`service start time`	Service start time
`tsid`	TSID range (0–15)
`nominal MSDU`	Nominal MAC SDU size
`max MSDU`	Maximum MAC SDU size
`min data rate`	Minimum data rate in bps
`mean data rate`	Mean data rate in bps
`peak data rate`	Peak data rate in bps
`max burst size`	Maximum burst size in bps
`delay bound`	Delay bound
`min phy rate`	Minimum PHY rate in bps

	`sba`	Surplus bandwidth allowance
	`medium time`	Medium time in TU of 32-µs periods per second
	`nominalPHY`	Nominal PHY rate in bps; if nominalPHY ≥ Minimum PHY rate, then AR6000 enables TSRS IE support (see CCX v4 s54.2.6 Traffic Stream Rate Set IE)
IOCTL Type	`get`	
Class	iwpriv (private IOCTLs)	

Name **AR6000_IOCTL_WMI_DELETE_QOS**

Description Deletes a prioritized data endpoint

Commands **`wmiconfig eth1 --deleteqos <trafficClass> <tsid>`**

Argument Type

```
struct {
A_UINT8            trafficClass;
A_UINT8            tsid;
} WMI_DELETE_PSTREAM_CMD
```

Arguments

`trafficClass`	Indicates the traffic Class of the PSTREAM to delete
`tsid`	Indicated the TSPEC ID within the traffic class to delete

IOCTL Type `get`

Class iwpriv (private IOCTLs)

Name **AR6000_IOCTL_WMI_SET_ERROR_REPORT_BITMASK**

Description Allows the host to control ERROR_REPORT events from the AR6001

Commands **`wmiconfig eth1 --setErrorReportingBitmask`**

Argument Type

```
struct {
A_UINT32        bitmask;
} WMI_TARGET_ERROR_REPORT_BITMASK;
```

Arguments

`bitmask`	`0x00000001`	Power save fails
	`0x00000002`	No cipher key
	`0x00000004`	Decryption error
	`0x00000008`	Beacon Miss
	`0x00000010`	Non-PS-enable STA joined PS-Enabled network

IOCTL Type `get`

Class iwpriv (private IOCTLs)

Name	**AR6000_IOCTL_WMI_GET_TARGET_STATS**
Description	The host uses this command to request that the AR6001 send statistics that it maintains. The 'clearStats' option is added to clear the target statistics maintained in the host. The structure of the WMI_TARGET_STATS is defined in **/include/wmi.h**.
Commands	`wmiconfig eth1 --getTargetStats -clearStats`

Argument Type	`struct {`
	`TARGET_STATS      targetStats;`
	`UINT8             clearStats`
	`} TARGET_STATS_CMD`

Arguments	`clearStats` Used to optionally clear the Stats
IOCTL Type	`get`
Class	iwpriv (private IOCTLs)

Name	**AR6000_IOCTL_WMI_SET_ASSOC_INFO**
Description	Specifies information elements that wish the AR6001 to add to all future association and reassociation requests. IEs are required to be correct and are used "as is" by the AR6000. IEs specified through this command are cleared when a **DISCONNECT** command is processed.
Commands	`wmiconfig eth1 --setAssocIe <IE>`

Argument Type	`struct {`
	`A_UINT8          ieType;`
	`A_UINT8          bufferSize;`
	`A_UINT8          assocInfo[1];`
	`} WMI_SET_ASSOC_INFO_CMD`

Arguments	`ieType`	Used directly in 802.11 frames
	`bufferSize`	Size of assocInfo in bytes, ranging from 0 to 240. If = 0,then previously set IEs are cleared.
	`assocInfo`	Pointer to the assocInfo which used directly in 802.11 frames
IOCTL Type	`set`	
Class	iwpriv (private IOCTLs)	

Name	**AR6000_IOCTL_WMI_SET_ACCESS_PARAMS**		
Description	Allows the host to set access parameters for the wireless network		
Commands	**`wmiconfig eth1 --acparams --txop <limit> --cwmin <0-15> --cwmax <0-15> --aifsn<0-15>`**		
Argument Type	`struct {`		
	`A_UINT16 txop;`		In units of 32 µs
	`A_UINT8`		`eCWmin;`
	`A_UINT8`	`eCWmax;`	
	`A_UINT8`	`aifsn;`	
	`} WMI_SET_ACCESS_PARAMS_CMD;`		
Arguments	`txop`	Maximum time the AR6001 can spend transmitting after it acquires the right to transmit	
	`eCWmin`	Minimum contention window	
	`eCWmax`	Maximum contention window	
	`aifsn`	Arbitration inter-frame space number	
IOCTL Type	`set`		
Class	iwpriv (private IOCTLs)		

Name	**AR6000_IOCTL_WMI_SET_BMISS_TIME**		
Description	Sets the beacon miss (BMISS) time in AR6001		

Commands	**`wmiconfig eth1 --setbmissbeacons=<val>`**
	`wmiconfig eth1 --setbmisstime=<val>`		

Argument Type	`struct {`		
	`A_UINT16`	`bmissTime;`	
	`A_UINT16`	`numBeacons;`	
	`} WMI_BMISS_TIME_CMD;`		
Arguments	`bmissTime`	Number of TUs, can range from 1000 to 5000	
	`numBeacons`	Number of beacons, can range from 5 to 50	
IOCTL Type	`set`		
Class	iwpriv (private IOCTLs)		

Name	**AR6000_IOCTL_WMI_SET_DISC_TIMEOUT**		
Description	The host uses this command to configure the length of time the AR6001 should spend when it attempts to reestablish a connection after losing link with its current BSS		
Commands	**`wmiconfig eth1 --disc=<timeout in seconds>`**		
Argument Type	`struct {`		
	`A_UINT8`	`disconnectTimeout;`	In seconds
	`} WMI_DISC_TIMEOUT_CMD;`		
Arguments	`disconnect Timeout`	Specifies the time limit after which a failure to reestablish a connection results in a DISCONNECT event	
IOCTL Type	`set`		
Class	iwpriv (private IOCTLs)		

Name	**AR6000_IOCTL_WMI_SET_IBSS_PM_CAPS**
Description	The host uses this command to support a non-standard power saving management scheme for ad hoc wireless networks consisting of up to eight stations that supports this form of power savings
Commands	`wmiconfig eth1 --ibsspmcaps --ps=<enable/disable> --aw=<ATIM Windows in millisecond> --ttl=<Time to live in number of beacon periods> --to=<timeout in milliseconds>`
Argument Type	`struct {`
IOCTL Type	set
Class	iwpriv (private IOCTLs)

```
struct {
A_UINT8         power_saving;
A_UINT8         ttl;              Number of beacon periods
A_UINT16        atim_windows;     ms
A_UINT16        timeout_value;    ms
} WMI_IBSS_PM_CAPS_CMD;
```

Arguments

power_saving	0	The non-standard power saving scheme is disabled; if = 1, the ad hoc power saving scheme is enabled
atim_windows		Specifies the length of the ad hoc traffic indication message (ATIM)
ttl		Specified the number of beacon periods for which the frame will be kept active
timeout_value		Specifies the timeout on Tx and Rx side

IOCTL Type set

Class iwpriv (private IOCTLs)

Name **AR6000_IOCTL_WMI_SET_SNRTHRESHOLD**

Description Configures how the AR6000 monitors and reports SNR of the connected BSS, which is used as a link quality metric.

Commands

```
wmiconfig eth1 --snrThreshold <weight> <upper_threshold_1>...
<upper_threshold_4> <lower_threshold_1>... <lower_threshold_4>
<pollTimer>
```

Argument Type

```
 struct WMI_SNR_THRESHOLD_PARAMS{
A_UINT8         weight;                Alpha
A_UINT8         thresholdAbove1_Val;   Lowest of upper
A_UINT8         thresholdAbove2_Val;
A_UINT8         thresholdAbove3_Val;
A_UINT8         thresholdAbove4_Val;   Highest of upper
A_UINT8         thresholdBelow1_Val;   Lowest of bellow
A_UINT8         thresholdBelow2_Val;
A_UINT8         thresholdBelow3_Val;
A_UINT8         thresholdBelow4_Val;   Highest of bellow
A_UINT32        pollTime;              Polling time in seconds
} WMI_SNR_THRESHOLD_PARAMS_CMD;
```

Arguments

weight = range in [1, 16]	Used in the formula to calculate average RSSI
thresholdAbove_Val [1...4] =	Above thresholds expressed in db, in ascending order
thresholdBelow_Val [1...4] =	Below thresholds expressed in db, in ascending order

	`pollTime =`	The signal strength sampling frequency in seconds
	`If polltime = 0,`	Signal strength sampling is disabled
IOCTL Type	`set`	
Class	iwpriv (private IOCTLs)	

Name **AR6000_XIOCTL_TCMD_CONT_TX**

Description Enable/disable continuous Tx test command; only works when the AR6000 is awake

Commands

```
athtestcmd --tx <sine/data/off> --txfreq <Tx channel or freq>
--txrate <rate index> --txpwr <data: 5-14dBm sine: 5-11dBm>
--txantenna <1/2/0 (auto)>
```

Argument Type `First word of buffer: AR6000_XIOCTL_TCMD_CONT_TX; starting from second word of buffer:`

```
struct {
A_UINT32        mode;
A_UINT32        freq;
A_UINT32        dataRate;     Only applies to modulated Tx mode
 A_UINT32       txPwr;
A_UINT32        antenna;
} WMI_TCMD_CONT_TX_CMD
```

Arguments

`mode =`	`0`	Disabling continuous Tx
	`1`	Enable continuous unmodulated Tx
	`2`	Enable continuous modulated Tx
`freq =`	Channel frequency in MHz (i.e., 2412 for Channel 1 in 802.11g)	
`dataRate =`	`0`	1 Mbps
	`1`	2 Mbps
	`2`	5.5 Mbps
	`3`	11 Mbps
	`4`	6 Mbps
	`5`	9 Mbps
	`6`	12 Mbps
	`7`	18 Mbps
	`8`	24 Mbps
	`9`	36 Mbps
	`10`	48 Mbps
	`11`	54 Mbps
`txPwr =`	Tx power in dBm [5-11] for unmod Tx, [5-14] for mod Tx	
`antenna =`	`1`	Antenna 1
	`2`	Antenna 2

IOCTL Type `set`

Name	**AR6000_XIOCTL_TCMD_CONT_RX**
Description	Enable/disable continuous Rx test command; only works when the AR6000 is awake.
Commands	`athtestcmd --rx <promis/filter/report> --rxfreq <Rx channel or freq> --rxantenna <1/2/0 (auto)>`

Argument Type

```
First word of buffer: AR6000_XIOCTL_TCMD_CONT_RX
Starting from 2nd word of buffer:
struct {
```

`A_UINT32`	`act;`	
`A_UINT32`	`freq;`	TotalPkt if act = 2, first 4 bytes of MAC address if act = 3
`A_UINT32`	`antenna;`	avgRssi if act = 2, remainder 2 bytes of MAC address if act = 3

```
} WMI_TCMD_CONT_RX_CMD
```

Arguments

`act =`	`0`	Promiscuous mode: enabling continuous Rx mode (accept all incoming frames)
	`1`	Filter mode: enabling continuous Rx mode (accept only frames with dest. address being equal to the specified MAC address (see act=3))
	`2`	Off mode: disabling continuous Rx mode and get the report from the last continuous Rx test
	`3`	setMacAddr mode: setting the MAC address of AR6000 - this overrides the AR6000's default MAC address to facilitate testing; this is the MAC address that will be compared against the destination address of incoming frames if act = 2
`freq`	Channel frequency in MHz (i.e., 2412 for Channel 1 in 802.11g) (when act = 2) = total number of successfully received packets (when act = 3) = first 4 bytes of MAC address to set to this STA	
`antenna`	`0`	Auto antenna (using ant diversity)
	`1`	Antenna 1
	`2`	Antenna 2 (when act = 2) = average RSSI of received packets (when act = 3) = upper two bytes equal to the final two bytes of the MAC address

IOCTL Type	`set`
Class	iwpriv (private IOCTLs): blocking

Name	**AR6000_XIOCTL_TCMD_PM**
Description	Sets the AR6000 in SLEEP or AWAKE mode. TCMD_CONT_TX/TCMD_CONT_RX commands are disabled when the AR6000 is set to SLEEP mode.
Commands	`athtestcmd --pm <wakeup/sleep>`

Argument Type

```
First word of buffer: AR6000_XIOCTL_TCMD_PM; Starting from 2nd
word of buffer:
struct {
     A_UINT8 mode;
}  WMI_TCMD_PM_CMD
```

Arguments

`mode =`	`1`	Wakeup the AR6000
	`2`	Force the AR6000 to sleep

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_SETFIXRATES**
Description	Set the AR6000 Tx rate to fixed values.
Commands	`wmiconfig eth1 --setfixrates <rate index>`

Argument Type

```
First word of buffer: AR6000_XIOCTL_WMI_SETFIXRATES; Starting from
2nd word of buffer:
struct {
A_UINT16      fixRateMask;
}  WMI_FIX_RATES_CMD
```

Arguments

`fixRateMask`	`0x0001`	1 Mbps
	`0x0002`	2 Mbps
	`0x0004`	5.5 Mbps
	`0x0008`	11 Mbps
	`0x0010`	6 Mbps
	`0x0020`	9 Mbps
	`0x0040`	12 Mbps
	`0x0080`	18 Mbps
	`0x0100`	24 Mbps
	`0x0200`	36 Mbps
	`0x0400`	48 Mbps
	`0x0800`	54 Mbps

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_GETFIXRATES**
Description	Retrieve the current fixed rate that the AR6000 is using.
Commands	`wmiconfig eth1 --getfixrates`

Argument Type

```
First word of buffer: AR6000_XIOCTL_WMI_SETFIXRATES; Starting from
2nd word of buffer:
struct {
A_UINT16      fixRateMask;
}  WMI_FIX_RATES_REPLY
```

Arguments

`fixRateMask`	`0x0001`	1 Mbps
	`0x0002`	2 Mbps
	`0x0004`	5.5 Mbps
	`0x0008`	11 Mbps
	`0x0010`	6 Mbps
	`0x0020`	9 Mbps
	`0x0040`	12 Mbps
	`0x0080`	18 Mbps
	`0x0100`	24 Mbps
	`0x0200`	36 Mbps
	`0x0400`	48 Mbps
	`0x0800`	54 Mbps

IOCTL Type	`get`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_SET_RSSITHRESHOLD**
Description	Set RSSI above/below thresholds; when a certain threshold is met, corresponding user defined tags are sent to the upper layer application.

Commands

```
wmiconfig eth1 --rssiThreshold <weight> <pollTime>
<above_threshold_tag_1> <above_threshold_val_1>...
<above_threshold_tag_6> <above_threshold_val_6>
<below_threshold_tag_1> <below_threshold_val_1>...
<below_threshold_tag_6> <below_threshold_val_6>
```

Argument Type

```
First word of buffer: AR6000_XIOCTL_WMI_SET_RSSITHRESHOLD;
Starting from 2nd word of buffer:
 struct user_rssi_params_t {
A_UINT8            weight;
A_UINT32           pollTime;
USER_RSSI_THOLD    tholds[12];
} USER_RSSI_PARAMS;
```

Arguments

`weight`	Range in [1, 16], used in the formula to calculate average RSSI
`pollTime`	RSSI sampling frequency in seconds. If polltime=0, RSSI sampling is disabled.
`tholds [12]`	User-defined RSSI thresholds (in dbM). The first six elements are ABOVE thresholds and can be set in any order, the last six elements are BELOW thresholds, and can also be set in any order. Users can specify an int16 type tag to every RSSI threshold.

```
struct user_rssi_thold_t {
A_INT16     tag;
A_INT16     rssi;
} USER_RSSI_THOLD;
```

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_CLR_RSSISNR**
Description	Clear the current calculated RSSI and SNR value.
Commands	`wmiconfig eth1 --cleanRssiSnr`
Arguments	N/A (send WMI_CLR_RSSI_SNR_CMDID to the AR6000 directly
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_SET_LQTHRESHOLD**
Description	Set link quality thresholds, the sampling happens at every unicast data frame Tx, if certain threshold is met, corresponding event will be sent to host.
Commands	**`wmiconfig eth1 --lqThreshold <enable> <upper_threshold_1>... <upper_threshold_4> <lower_threshold_1>... <lower_threshold_4>`**

Argument Type

```
First word of buffer: AR6000_XIOCTL_WMI_SET_LQTHRESHOLD; Starting
from 2nd word of buffer:
 struct WMI_LQ_THRESHOLD_PARAMS {
A_UINT8     enable;
A_UINT8     thresholdAbove1_Val;
A_UINT8     thresholdAbove2_Val;
A_UINT8     thresholdAbove3_Val;
A_UINT8     thresholdAbove4_Val;
A_UINT8     thresholdBelow1_Val;
A_UINT8     thresholdBelow2_Val;
A_UINT8     thresholdBelow3_Val;
A_UINT8     thresholdBelow4_Val;
}   WMI_LQ_THRESHOLD_PARAMS_CMD
```

Arguments

`enable`	`1`	Enable Link Quality sampling
	`0`	Disable Link Quality sampling
`thresholdAbove_Val [1...4]`		Above thresholds (value in [0,100]), in ascending order thresholdBelow_Val [1...4] = below thresholds (value in [0,100]), in ascending order

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_SET_RTS**
Description	Set RTS threshold.
Commands	**`wmiconfig eth1 --setRTS <pkt length threshold>`**

Argument Type

```
First word of buffer: AR6000_XIOCTL_WMI_SET_RTS; Starting from
2nd word of buffer:
struct {
A_UINT16    threshold;
} WMI_SET_RTS_CMD;
```

Arguments	`threshold` — `RTS Threshold`
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_SET_LPREAMBLE**
Description	Set force long preamble mode of the AR6000
Commands	**`wmiconfig eth1 --setlongpreamble <enable>`**

Argument Type

```
First word of buffer: AR6000_XIOCTL_WMI_SET_LPREAMBLE; Starting
from 2nd word of buffer:
struct {
A_UINT8     status;
} WMI_SET_LPREAMBLE_CMD;
```

Arguments

`status`	`0`	Disable force long preamble
	`1`	Enable force long preamble

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_SET_AUTHMODE**
Description	Set authentication mode of reconnection.
Commands	**`wmiconfig eth1 --setauthmode <mode>`**

Argument Type	`First word of buffer: AR6000_XIOCTL_WMI_SET_AUTHMODE; Starting from 2nd word of buffer:`
	`struct {`
	`A_UINT8 mode;`
	`}  WMI_SET_AUTH_MODE_CMD;`

Arguments	`mode` `0x00` Proceed authentication during reconnecting `0x01` Do NOT proceed authentication during reconnection
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_SET_REASSOCMODE**
Description	Specify whether disassociated frame should be sent or not upon reassociation.
Commands	**`wmiconfig eth1 --setreassocmode <mode>`**

Argument Type	`First word of buffer: AR6000_XIOCTL_WMI_SET_REASSOCMODE; Starting from 2nd word of buffer:`
	`struct {`
	`A_UINT8 mode;`
	`}  WMI_SET_REASSOC_MODE_CMD;`

Arguments	`mode` `0x00` Send disassoc frame upon reassociation `0x01` Do not send disassoc frame upon reassociation
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_GET_KEEPALIVE**
Description	Get the configured keepalive interval.
Commands	**`wmiconfig --getkeepalive`**

Argument Type	`UINT8 cmd AR6000_XIOCTL_WMI_GET_KEEPALIVE`
	`UINT8 keepAliveInterval  // The interval in milliseconds`
	`A_BOOL configured // Is the keepalive interval configured?`

IOCTL Type	`get`
Class	iwpriv

Name	**AR6000_XIOCTL_WMI_SET_KEEPALIVE**
Description	Set the keepalive interval.
Commands	**`wmiconfig -setkeepalive <keepalive-interval>`**

Argument Type	`UINT8 cmd AR6000_XIOCTL_WMI_SET_KEEPALIVE`
	`A_UINT8 keepAliveInterval`

IOCTL Type	`get`
Class	iwpriv

Name	**AR6000_XIOCTL_BMI_ROMPATCH_INSTALL**
Description	Install a ROM patch.
Commands	**`fwpatch -ifname=<interface> --file=<patch-filename>`**

Argument Type	`union {`
	`struct {`
	`UINT32 cmd (AR6000_XIOCTL_BMI_ROMPATCH_INSTALL)`
	`UINT32 ROM Address // address in ROM`
	`UINT32 RAM Address         // address in RAM`
	`UINT32 number of bytes // size of the patch`
	`UINT32 activate // 0 = do not activate, 1 = activate patch`
	`} A_UINT32 rompatch ID`

IOCTL Type	`set`
Class	iwpriv

Name	**AR6000_XIOCTL_BMI_ROMPATCH_UNINSTALL**
Description	Uninstall a ROM patch.
Commands	Used internally by the fwpatch tool.

Argument Type	`struct {`
	`UINT32 cmd (AR6000_XIOCTL_BMI_ROMPATCH_UNINSTALL)`
	`A_UINT32 rompatch ID // the ID of the rompatch to be uninstalled`
	`}`

IOCTL Type	`set`
Class	iwpriv

Name	**AR6000_XIOCTL_BMI_ROMPATCH_ACTIVATE**
Description	Activate ROM patch(es).
Commands	Used internally by the fwpatch tool.

Argument Type	`struct {`
	`UINT32 cmd (AR6000_XIOCTL_BMI_ROMPATCH_ACTIVATE)`
	`UINT32 rompatch_count // How many ROM patches to be actived?`
	`UINT32 rompatch ID[rompatch_count] // List of rompatch IDs to`
	`activate`
	`}`

IOCTL Type	`set`
Class	iwpriv

Name	**AR6000_XIOCTL_BMI_ROMPATCH_DEACTIVATE**
Description	Deactivate ROM patch(es).
Commands	Used internally by the fwpatch tool.

Argument Type	`struct {`
	`UINT32 cmd (AR6000_XIOCTL_BMI_ROMPATCH_DEACTIVATE)`
	`UINT32 rompatch_count // How many ROM patches to be deactivated`
	`UINT32 rompatch ID[rompatch_count] // List of rompatch IDs to`
	`deactivate`
	`}`

IOCTL Type	`set`
Class	iwpriv

Name **AR6000_XIOCTL_DBGLOG_CFG_MODULE**

Description Configure the debug logging on a per module basis.

Commands

```
wmiconfig -setdbglogconfig -mmask=<mask> --rep=<0/1>
--tsr=<tsr codes> --size=<size>
```

Argument Type

```
struct dbglog_config_s {
UINT32 cfgvalid; // Mask with valid config bits
union {
       struct {
       UINT32 mmask:16; //Mask of modules with logging enabled
       UINT32 rep:1; // Is reporting enabled?
       UINT32 tsr:3; // Time stamp resolution
       UINT32 size:10; // Number of messages in report
       UINT32 reserved:2; // Reserved
}
UINT32 value;
```

IOCTL Type set

Class iwpriv

Name **AR6000_XIOCTL_DBGLOG_GET_DEBUG_LOGS**

Description Get debug logs from target

Commands

```
wmiconfig --getdbglogs
```

Argument Type

IOCTL Type get

Class iwpriv

Name **AR6000_XIOCTL_WMI_SET_BT_STATUS**

Description Set the status for a Bluetooth stream.

Commands

```
wmiconfig -setBTstatus <streamType> <status>
```

Argument Type

```
struct {
A_UINT8 streamType; // stream type
A_UINT8 status; // stream status
}WMI_SET_BT_STATUS_CMD;

enum {
    BT_STREAM_UNDEF = 0,
    BT_STREAM_SCO,             // SCO stream
    BT_STREAM_A2DP,            // A2DP stream
    BT_STREAM_MAX
} BT_STREAM_TYPE;

enum {
    BT_STATUS_UNDEF = 0,
    BT_STATUS_START,
    BT_STATUS_STOP,
    BT_STATUS_RESUME,
    BT_STATUS_SUSPEND,
    BT_STATUS_MAX
} BT_STREAM_STATUS;
```

IOCTL Type set

Class iwpriv

Name **AR6000_XIOCTL_WMI_SET_BT_PARAMS**

Description Set the Bluetooth parameters for SCO or A2DP streams.

Commands **`wmiconfig -setBTparams <paramType> <params>`**

Argument Type

```
struct {
    union {
        BT_PARAMS_SCO scoParams;
        BT_PARAMS_A2DP a2dpParams;
        BT_PARAMS_MISC miscParams;
        BT_COEX_REGS regs;
    } info;
    A_UINT8 paramType;
} WMI_SET_BT_PARAMS_CMD;

struct {
    A_UINT8 noSCOPkts; // Number of SCO packets between
consecutive PS-POLLs
    A_UINT8 pspollTimeout;
    A_UINT8 stompbt;
} BT_PARAMS_SCO;

struct { // Parameters for A2DP Bluetooth streams
    A_UINT32 period;
    A_UINT32 dutycycle;
    A_UINT8  stompbt;
} BT_PARAMS_A2DP;

struct {
    union {
        WLAN_PROTECT_POLICY_TYPE protectParams;
        A_UINT16 wlanCtrlFlags;
    }info;
    A_UINT8 paramType;
} BT_PARAMS_MISC;

struct { // Values for Bluetoth coexistence registers
    A_UINT32 mode; // Coexistence mode
    A_UINT32 scoWghts; // WLAN and BT weights
    A_UINT32 a2dpWghts;
    A_UINT32 genWghts;
    A_UINT32 mode2; // Coexistence mode2
    A_UINT8  setVal;
} BT_COEX_REGS;
```

IOCTL Type set

Class iwpriv

Name	**AR6000_XIOCTL_WMI_SET_HOST_SLEEP_MODE**
Description	Set the host sleep mode (awake or asleep).
Commands	**`wmiconfig -sethostmode <awake/asleep>`**
Argument Type	see below
IOCTL Type	set
Class	iwpriv

```
struct {
A_BOOL awake;                // Is the host awake?
A_BOOL asleep;               // Is the host asleep?
}
```

Name	**AR6000_XIOCTL_WMI_SET_WOW_MODE**
Description	Enable or disable Wake on Wireless
Commands	**`wmiconfig -setwowmode <enable/disable>`**
Argument Type	see below
IOCTL Type	set
Class	iwpriv

```
struct {
A_BOOL enable_wow; // Enable or disable the Wake on Wireless
feature
}
```

Name	**AR6000_XIOCTL_WMI_GET_WOW_LIST**
Description	Get the list of Wake on Wireless patterns
Commands	**`wmiconfig -getwowlist <list-id>`**
Argument Type	see below
IOCTL Type	get
Class	iwpriv

```
struct {
A_UINT8 filter_list_id; // The ID of the list of patterns to
return
}
```

Name	**AR6000_XIOCTL_WMI_ADD_WOW_PATTERN**
Description	Add a Wake on Wireless pattern
Commands	**`wmiconfig -addwowpattern <list-id> <filter-size> <filter-offset> <pattern> <mask>`**
Argument Type	see below
IOCTL Type	set
Class	iwpriv

```
struct {
A_UINT8filter_list_id; // The ID of the list to which the pattern
is to be added
A_UINT8 filter_size; // The size of the pattern in bytes
A_UINT8filter_offset; // The offset within the frame that the
pattern starts
A_UINT8 filter[1]; // Size bytes of the pattern data followed by
size bytes of the mask to be used.
}
```

Name	**AR6000_XIOCTL_WMI_DEL_WOW_PATTERN**
Description	Delete one of the configured Wake on Wireless patterns
Commands	**`wmiconfig -delwowpattern <list-id> <pattern-id>`**

Argument Type	`struct {`
	`A_UINT16filter_list_id; // The ID of the list which contains the pattern`
	`A_UINT16filter_ id; // The ID of the (WoW) filter pattern to be deleted`
	`}`

IOCTL Type	`set`
Class	iwpriv

AR6000_IOCTL_EXTENDED

NOTE: All XIOCTL commands are based on IOCTL commands.

Table H-3 shows the AR6000_IOCTL_EXTENDED Linux private IOCTLs.

Table H-3. **AR6000_IOCTL_EXTENDED**

IOCTL	Description	Page
AR6000_XIOCTL_BMI_DONE	Indicates that the host is finished using BMI	page H-42
AR6000_XIOCTL_BMI_DSETPATCH_INSTALL	Unsupported	—
AR6000_XIOCTL_BMI_EXECUTE	Host causes the AR6001 to execute code	page H-42
AR6000_XIOCTL_BMI_READ_MEMORY	Host reads the AR6001 memory	page H-39
AR6000_XIOCTL_BMI_READ_SOC_REGISTER	Host reads a 32-bit target SOC register	page H-43
AR6000_XIOCTL_BMI_ROMPATCH_ACTIVATE	Activate ROM patch(es)	page H-28
AR6000_XIOCTL_BMI_ROMPATCH_DEACTIVATE	Deactivate ROM patch(es)	page H-28
AR6000_XIOCTL_BMI_ROMPATCH_INSTALL	Install a ROM patch	page H-28
AR6000_XIOCTL_BMI_ROMPATCH_UNINSTALL	Uninstall a ROM patch	page H-28
AR6000_XIOCTL_BMI_SET_APP_START	Sets the starting address of the target application	page H-42
AR6000_XIOCTL_BMI_TEST	Deprecated	page H-43
AR6000_XIOCTL_BMI_WRITE_MEMORY	Host writes to the AR6001 memory	page H-41
AR6000_XIOCTL_BMI_WRITE_SOC_REGISTER	Host writes a 32-bit target SOC register	page H-43
AR6000_XIOCTL_CHECK_TARGET_READY	Returns 0 if the target exists and is ready to accept IOCTLs	page H-47
AR6000_XIOCTL_DBGLOG_CFG_MODULE	Configure the debug logging on a per module basis	page H-29
AR6000_XIOCTL_DBGLOG_GET_DEBUG_LOGS	Get debug logs from target	page H-29
AR6000_XIOCTL_DIAG_READ	Used to read the contents from the target's memory through the diagnostic window	page H-40
AR6000_XIOCTL_DIAG_WRITE	Used to write the contents to the target's memory through the diagnostic window	page H-40
AR6000_XIOCTL_HTC_RAW_CLOSE	Closes the HTC RAW interface	page H-46
AR6000_XIOCTL_HTC_RAW_OPEN	Provides a way for the application to directly interact with the target, bypassing WMI	page H-46
AR6000_XIOCTL_HTC_RAW_READ	Reads a set of bytes from the specified mailbox	page H-46
AR6000_XIOCTL_HTC_RAW_WRITE	Writes a set of bytes to the specified mailbox	page H-47
AR6000_XIOCTL_FORCE_TARGET_RESET	Unsupported	—
AR6000_XIOCTL_GPIO_INPUT_GET	Allows the host to read the GPIO pins that are configured for input	page H-48
AR6000_XIOCTL_GPIO_INTR_ACK	Acknowledges and re-arms APIO interrupts	page H-49
AR6000_XIOCTL_GPIO_INTR_WAIT	Waits for a GPIO interrupt until one occurs, then returns the information about the interrupt(s)	page H-49
AR6000_XIOCTL_GPIO_OUTPUT_SET	Manages output on GPIO pins configured for output	page H-48
AR6000_XIOCTL_GPIO_REGISTER_GET	Allows the host to read an arbitrary GPIO register	page H-49
AR6000_XIOCTL_GPIO_REGISTER_SET	Allows the host to dynamically change GPIO configuration	page H-48

Table H-3. **AR6000_IOCTL_EXTENDED (continued)**

IOCTL	Description	Page
AR6000_XIOCTL_SET_ADHOC_BEACON _INTVAL	Sets the beacon interval for an ad hoc network	page H-50
AR6000_XIOCTL_SET_ADHOC_BSSID	Allows the host to specify the BSSID for an ad hoc network	page H-50
AR6000_XIOCTL_SET_MAX_SP	Sets the maximal service period	page H-51
AR6000_XIOCTL_SET_VOICE_PKT_SIZE	Allows the host to control which packets the target treats as voice packets	page H-50
AR6000_XIOCTL_OPT_SEND_FRAME	Special feature, sends a special frame	page H-37
AR6000_XIOCTL_TCMD_CONT_TX	Enables/disables continuous Tx test command	page H-22
AR6000_XIOCTL_TCMD_CONT_RX	Enables/disables continuous Rx test command	page H-23
AR6000_XIOCTL_TCMD_PM	Sets SLEEP or AWAKE mode	page H-23
AR6000_XIOCTL_USER_SETKEYS	If invoked, keys that have been installed via the previous add key IOCTL are reused; extended for WMI_ADD_CIPHER_KEY_CMD info	page H-38
AR6000_XIOCTL_WMI_ADD_WOW_PATTERN	Add a Wake on Wireless pattern	page H-31
AR6000_XIOCTL_WMI_CLR_RSSISNR	Clears the current RSSI and SNR value	page H-25
AR6000_XIOCTL_WMI_CLEAR_TARGET_STATS	Clears the target statistics	page H-37
AR6000_XIOCTL_WMI_CONNECT	Allows the host to export one IOCTL to set all of the profile information and any other information to modify the connect handling	page H-38
AR6000_XIOCTL_WMI_DEL_WOW_PATTERN	Delete one of the configured Wake on Wireless patterns	page H-32
AR6000_XIOCTL_WMI_GETFIXRATES	Retrieves the current fixed rate	page H-24
AR6000_WMI_GET_HB_CHALLENGE_RESP	Causes the driver to send a challenge to the target	page H-40
AR6000_XIOCTL_WMI_GET_KEEPALIVE	Get the configured keepalive interval	page H-27
AR6000_XIOCTRL_WMI_GET_POWER_MODE	Retrieves the target power mode	page H-52
AR6000_WMI_GET_RD	Returns the regulatory domain of the WLAN device	page H-41
AR6000_XIOCTL_WMI_GET_ROAM_DATA	Retrieves the target roam data	page H-53
AR6000_XIOCTL_WMI_GET_ROAM_TBL	Retrieves the roam table maintained by the target and contains all the APs that to roam to	page H-51
AR6000_XIOCTL_WMI_GET_WOW_LIST	Get the list of Wake on Wireless patterns	page H-31
AR6000_XIOCTL_WMI_SET_APPIE	Add Application-Specified IE to a management frame	page H-36
AR6000_XIOCTL_WMI_SET_AUTHMODE	Sets authentication mode of reconnection	page H-27
AR6000_XIOCTL_WMI_SET_BT_PARAMS	Set the Bluetooth parameters for SCO or A2DP streams	page H-30
AR6000_XIOCTL_WMI_SET_BT_STATUS	Set the status for a Bluetooth stream	page H-29
AR6000_XIOCTL_WMI_SET_CONNECT_CTRL	Alters a connect operation by specifying action modifications that would have been performed by the AR6000 when connecting to an AP	page H-38
AR6000_XIOCTL_WMI_SETFIXRATES	Sets the Tx rate to fixed values	page H-45
AR6000_WMI_SET_HB_CHALLENGE_RESP _PARAMS	Sets the parameters used by the module and schedules a periodic timer that sends the challenge messages to the target	page H-42

Table H-3. **AR6000_IOCTL_EXTENDED (continued)**

IOCTL	Description	Page
AR6000_XIOCTL_WMI_SET_HOST_SLEEP_MODE	Set the host sleep mode	page H-31
AR6000_XIOCTL_WMI_SET_KEEPALIVE	Set the keepalive interval	page H-27
AR6000_XIOCTL_WMI_SET_LPREAMBLE	Sets force long preamble mode of the AR6000	page H-26
AR6000_XIOCTL_WMI_SET_LQTHRESHOLD	Sets link quality thresholds	page H-25
AR6000_WMI_SET_MGMT_FRM_RX_FILTER	Sets the type of frames that should be received	page H-41
AR6000_XIOCTRL_WMI_SET_POWERSAVE _TIMERS	Set the power save timers for PS-POLL and triggers	page H-52
AR6000_XIOCTL_WMI_SET_REASSOCMODE	Specify whether disassociated frame should be sent or not upon reassociation	page H-27
AR6000_XIOCTL_WMI_SET_RETRYLIMITS	Allows the host to influence the number of times that the AR6000 should attempt to send a frame before it gives up	page H-36
AR6000_XIOCTL_WMI_SET_ROAM_CTRL	Sets roam mode, forces a roam to a specific BSSID, or sets the host bias for a BSSID	page H-51
AR6000_XIOCTL_WMI_SET_RSSITHRESHOLD	Sets RSSI above/below thresholds	page H-24
AR6000_XIOCTL_WMI_SET_RTS	Sets RTS threshold	page H-26
AR6000_XIOCTL_WMI_SET_TXOP	Enables or disables WMI TXOP bursting	page H-37
AR6000_XIOCTRL_WMI_SET_WLAN_STATE	Enables or disables the target WLAN capability	page H-52
AR6000_XIOCTL_WMI_SET_WSC_STATUS	Inform the target of WSC registration status	page H-53
AR6000_XIOCTL_WMI_SET_WOW_MODE	Enables or disables Wake on Wireless	page H-31
AR6000_XIOCTL_WMI_STARTSCAN	Starts a scan	page H-45
AR6000_XIOCTL_SET_IP	Conveys the IP binding to target	page H-54
AR6000_XIOCTL_AP_SET_ACL_POLICY	Sets policy for ACL	page H-54
AR6000_XIOCTL_AP_INTRA_BSS_COMM	Enables/disables communication between connected STAs	page H-55

Name	**AR6000_XIOCTL_BMI_DONE**
Description	Indicates that the host is finished using BMI
Commands	`bmiloader -i eth<x> --done`
Argument Type	First word of buffer: `AR6000_XIOCTL_BMI_DONE (=1)`
	Starting from second word of buffer:
	`A_UINT32` `cmd;` BMI command
Arguments	`BMI_DONE`
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_SET_APPIE**
Description	Add Application-specified IE to a management frame. The maximum length is 76 bytes. Including the length and the element id, this translates to 78 bytes.
Commands	`--setappie <frame> <IE>` where frame is one of beacon, probe, respon, assoc IE is a hex string starting with dd, if IE is 0 then no IE is sent in the management frame
Argument Type	`mgmtFrmType;     /* one of WMI_MGMT_FRAME_TYPE */`
Arguments	`ieLen;           /* Length  of the IE that should be added to the GMT frame */`
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_SET_RETRYLIMITS**
Description	Allows the host to influence the number of times that the AR6000 should attempt to send a frame before it gives up.
Commands	`--setretrylimits <frameType> <trafficClass> <maxRetries> <enableNotify>` Where: <frameType> = 0 for management frame (MGMT_FRAMETYPE) = 1 for control frame (CONTROL_FRAMETYPE) = 2 for data frame (DATA_FRAMETYPE) <trafficClass> = 1 for BK, 2 for VI, 3 for VO, only applies to data frame type <maxRetries> = # in [2 - 13]; <enableNotify> = ON to enable notification of max retries exceeded = OFF to disable notification of max retries exceeded

Argument Type	`UINT8`	`frameType`	A WMI_FRAMETYPE specifying which type of frame is of interest
Arguments	`UINT8`	`trafficClass`	Specifies a traffic class (see **"CREATE_PSTREAM"**). This parameter is only significant when frameType = DATA_FRAMETYPE.
	`UINT8`	`maxRetries`	The maximum number of times that the AR6000 attempts to retry a frame transmit. It must be in the range WMI_MIN_RETRIES (2) to WMI_MAX_RETRIES (15). If the special value 0 is used, maxRetries is set to 15.
IOCTL Type	`set`		
Class	iwpriv (private IOCTLs)		

Name	**AR6000_XIOCTL_OPT_SEND_FRAME**
Description	Special feature, sends a special frame
Commands	**`--sendframe <frmType> <dstaddr> <bssid> <optIEDatalen> <optIEData>`** Where: <frmType> = 1 for probe request frame = 2 for probe response frame = 3 for CPPP start = 4 for CPPP stop <dstaddr> = the destination mac address, in xx:xx:xx:xx:xx:xx format <bssid> = the bssid (in xx:xx:xx:xx:xx:xx format) <optIEDatalen> = optional IE data length <optIEData> = the pointer to optional IE data array
Argument Type	`struct` `FRAME CMD`

Arguments	`A_UINT16` `optIEDataLen;`
	`A_UINT8` `frmType;`
	`A_UINT8` `dstAddr[ATH_MAC_LEN];`
	`A_UINT8` `bssid[ATH_MAC_LEN];`
	`A_UINT8` `optIEData[1];`

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_CLEAR_TARGET_STATS**
Description	The host uses this command to request that the AR6001 send statistics that it maintains. The ‘clearStats’ is added to clear the target statistics maintained in the host. The structure of the WMI_TARGET_STATS is defined in **/include/wmi.h**.
Commands	**`wmiconfig eth1 --getTargetStats -clearStats`**
Argument Type	struct { TARGET_STATS `targetStats;` UINT8 `clearStats` } TARGET_STATS_CMD
Arguments	clearStats `Used to clear the Stats`

Name	**AR6000_XIOCTL_WMI_SET_TXOP**
Description	Enables (or disables) WMI TXOP bursting.
Commands	**`--txopbursting <burstEnable>`** Where: <burstEnable> = 0 to disallow TxOp bursting = 1 to allow TxOp bursting
Argument Type	`struct` `WMI SET TXOP Command`
Arguments	`A_UINT8` `txopEnable` WMI_TXOP_DISABLED (0) or WMI_TXOP_ENABLED (1)
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_USER_SETKEYS**
Description	This IOCTL has no argument. Once this IOCTL is invoked, the keys that have been installed via the previous add key IOCTL are reused. This reuse happens only once. This reuse happens after getting a connect event from the AR6000. If instead of the connect event, a disconnect event is received then the keys are cleared except for one case. The case where the keys are not cleared is if the disconnect event was received with the reason as CSERV_DISCONNECT. This command is now extended to modify the information sent in the WMI_ADD_CIPHER_KEY_CMD.
Commands	**`--usersetkeys <keyCtrl> <keyRSC[8]>`**
Argument Type	`A_UINT32 keyCtrl`

Arguments	`A_UINT32 keyRSC[8] =// If AR6000_XIOCTL_USER_SETKEY_USERSC is set`
	`AR6000_XIOCTL_USER_SETKEY_SAVE = 0x0001`
	`AR6000_XIOCTL_USET_SETKEY_USERSC = 0x0002`

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_SET_CONNECT_CTRL**
Description	This IOCTL alters the connect operation by specifying modifications to actions that would be performed by the AR6000 when connecting to an AP. Some of the modifications include: ■ The type of management frame to be sent to an AP when associating to it [Assoc or ReAssoc] ■ Ignore the group key setting when doing a profile match ■ Do not perform a profile match because an application has already performed it ■ Ignore the Admission Capacity information in the beacon of the AP.
Commands	**`--setconnectCtrl --<res[3]> // see bits below for res[]`**
Argument Type	`A_UINT32        connectCtrl`
Arguments	`res[3] =        // See bits below` `CONNECT_SEND_REASSOC` = 0x01 `CONNECT_IGNORE_GROUP_KEY` = 0x02 `CONNECT_PROFILE_MATCH_DONE` = 0x04 `CONNECT_IGONRE_AAC_BEACON` = 0x08
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_CONNECT**
Description	Instead of issuing multiple IOCTLs to set the frequency, BSSID, or ciphers, the host driver could export one IOCTL to set all of the profile information and any other information to modify the connect handling via the AR6000_XIOCTL_WMI_CONNECT interface.
Commands	**`--connectCtrl --<res[3]> // See bits below for res[]`**

Argument Type	`{`
	`A_UINT8         networkType`
	`A_UINT8         dot11AuthMode`
	`A_UINT8         authMode`
	`A_UINT8         pairwiseCryptoType`

```
A_UINT8         pairwiseCryptoLen
A_UINT8         groupCryptoType
A_UINT8         groupCryptoLen
A_UINT8         ssidLength
A_UCHAR         ssid[WMI_MAX_SSID_LEN];
A_UINT16        channel
A_UINT8         bssid[ATH_MAC_LEN]
A_UINT8         res[3]
A_UINT32        connectCtrl
} WMI_CONNECT_CMD
```

Arguments

```
res[3]          = 0x01 // CONNECT_SEND_REASSOC
                = 0x02 // CONNECT_IGNORE_GROUP_KEY
                = 0x04 // CONNECT_PROFILE_MATCH_DONE
                = 0x08 // CONNECT_IGONRE_AAC_BEACON
```

IOCTL Type `set`

Class iwpriv (private IOCTLs)

Name **AR6000_XIOCTL_BMI_READ_MEMORY**

Description The host reads the AR6001 memory

Commands **`bmiloader -i eth<x> --read --address=<address> --length=<bytes> --file=<filename>`**

Argument Type First word of buffer: `AR6000_XIOCTL_BMI_READ_MEMORY (=2)`

Starting from second word of buffer:

```
union {
  struct {
  A_UINT32        cmd;                BMI command
  A_UINT32        address;            Address in the AR6001 address
                                      space
  A_UINT32        length;             Length of data to be read
  }
A_UCHAR           results[length];    Data read from the AR6001
}
```

Arguments

`cmd`	BMI_READ_MEMORY
`address`	Should be word aligned
`length`	In bytes

IOCTL Type `set/get`

Class iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_DIAG_READ**
Description	Used to read the contents from the target's memory through the diagnostic window. No cooperation is required for this from the target.
Commands	`--diagread`
Argument Type	`First word of the buffer AR6000_XIOCTL_DIAG_READ. Starting from the second word in the buffer:` `A_UINT32 addr;` `A_UINT32 data;`

Arguments	`addr` — `A 32 bit value in the target's address space. Identifies the location from where the content has to be read.`
	`data` — `A 32 bit value identifying the content that was read from the location specified by the address above.`

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_DIAG_WRITE**
Description	Used to write the contents to the target's memory through the diagnostic window. No cooperation is required for this from the target.
Commands	`--diagwrite`
Argument Type	`First word of the buffer AR6000_XIOCTL_DIAG_WRITE. Starting from the second word in the buffer:` `A_UINT32 addr;` `A_UINT32 data;`

Arguments	`addr` — `A 32 bit value in the target's address space. Identifies the location where the content specified by 'data' has to be written.`
	`data` — `A 32 bit value identifying the content that needs to be written at the location specified by the address above.`

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_WMI_GET_HB_CHALLENGE_RESP**
Description	Causes the driver to send a challenge to the target. The IOCTL has a provision to accept a 32 bit cookie from the application which the latter can use to tag the challenge sent to the target. The same cookie shall be passed on to the application in the event notification corresponding to the reply sent by the target for this challenge.
Commands	`--getheartbeat --cookie=<cookie>` Where <cookie> is used to identify the response corresponding to a challenge sent

Argument Type	`First word of the buffer: AR6000_XIOCTL_WMI_GET_HB_CHHALLENGE_RESP (=53). Starting from 2nd word of buffer:`
	`    A_UINT32 cookie;`

Arguments	`Cookie: A 32 bit id associated with the challenge being sent.`
IOCTL Type	`get`
Class	iwpriv (private IOCTLs)

Name	**AR6000_WMI_GET_RD**
Description	This command returns the regulatory domain of the WLAN device. This is the value that was received via the WMI_REGDOMAIN_EVENTID from the AR6000.
Commands	`--getRD`
Argument Type	`A_UINT32` `regDomain`
IOCTL Type	`get`
Class	iwpriv (private IOCTLs)

Name	**AR6000_WMI_SET_MGMT_FRM_RX_FILTER**
Description	Sets the type of frames that should be received. This is used by the WPS application to receive beacons and probe responses.
Commands	`--setmgmtfilter <op> <frametype>` Where op is either set or clear and frametype is a beacon proberesp
Argument Type	`A_UINT32` `filterType. This can have the following values:`

Arguments	`(a) IEEE80211_FILTER_TYPE_BEACON - Receive the beacons`
	`(b) IEEE80211_FILTER_TYPE_PROBE_RESP - Receive the Probe Responses`
	`(c) IEEE80211_FILTER_TYPE_BEACON | IEEE80211_FILTER_TYPE_PROBE_RESP - Receive both beacons and probe responses.`
	`(d) 0 - Neither beacon nor probe responses are received.`

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_BMI_WRITE_MEMORY**
Description	The host writes to the AR6001 memory
Commands	**`bmiloader -i eth<x> --write --address=<address> --file=<filename>`**
Argument Type	First word of buffer: `AR6000_XIOCTL_BMI_WRITE_MEMORY (=3)` Starting from second word of buffer:

`A_UINT32`	`cmd;`	BMI command BMI_READ_MEMORY
`A_UINT32`	`address;`	Address in the AR6001 address space (should be word aligned)
`A_UINT32`	`length;`	Length of data to be written
`A_UCHAR`	`data[length];`	Data

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name **AR6000_WMI_SET_HB_CHALLENGE_RESP_PARAMS**

Description Sets the parameters used by the module and schedules a periodic timer that sends the challenge messages to the target. The 'timeout' parameter discussed in the design above is implicitly defined and is equal to the interval between two consecutive challenge messages.

Commands

```
--detecterror --frequency=<sec> --threshold=<count> where:
```

Where:

<frequency> is the periodicity of the challenge messages in seconds

<threshold> is the number of challenge misses after which the error detection module in the driver will report an error

Argument Type

```
First word of the buffer:
AR6000_XIOCTL_WMI_SET_HB_CHALLENGE_RESP_PARAMS (=52). Starting
from 2nd word of buffer:
typedef struct {
    A_UINT32 frequency;
    A_UINT8   threshold;
} WMI_SET_HB_CHALLENGE_RESP_PARAMS_CMD
```

Arguments

`Frequency`	`Periodicity of the challenge messages in seconds`
`Threshold`	`Number of challenge misses after which the error detection module in the driver reports an error.`

IOCTL Type `set`

Class iwpriv (private IOCTLs)

Name **AR6000_XIOCTL_BMI_EXECUTE**

Description The host causes the AR6001 to execute code

Commands

```
bmiloader -i eth<x> --execute --address=<function start address>
--param=<value>
```

Argument Type First word of buffer: `AR6000_XIOCTL_BMI_EXECUTE (=4)`

Starting from second word of buffer:

`A_UINT32`	`cmd;`	BMI command (BMI_EXECUTE)
`A_UINT32`	`address;`	Address in the AR6001 address space (Should be word aligned)
`A_UINT32`	`parameter;`	Parameter to pass as an argument to the code

IOCTL Type `set`

Class iwpriv (private IOCTLs)

Name **AR6000_XIOCTL_BMI_SET_APP_START**

Description The host sets the starting address of the target application

Commands

```
bmiloader -i eth<x> --begin --address=<function start address>
```

Argument Type First word of buffer: `AR6000_XIOCTL_BMI_SET_APP_START (=5)`

Starting from second word of buffer:

`A_UINT32`	`cmd;`	BMI command (BMI_SET_APP_START)
`A_UINT32`	`address;`	Address in the AR6001 address space (Should be word aligned)

IOCTL Type `set`

Class iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_BMI_READ_SOC_REGISTER**
Description	Host reads a 32-bit target SOC register
Commands	**`bmiloader -i eth<x> --get --address=<register address>`**
Argument Type	First word of buffer: `AR6000_XIOCTL_BMI_READ_SOC_REGISTER (=6)` Starting from second word of buffer:

```
union {
  struct {
  A_UINT32        cmd;               BMI command
                                     (BMI_READ_SOC_REGISTER)
  A_UINT32        address;           Address in the AR6001 address space
                                     (Should be word aligned)
  } A_UINT32      [results];         Data read from the AR6001
}
```

IOCTL Type	`set/get`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_BMI_WRITE_SOC_REGISTER**
Description	Host writes a 32-bit target SOC register
Commands	**`bmiloader -i eth<x> --set --address=<register address> --param=<value>`**
Argument Type	First word of buffer: `AR6000_XIOCTL_BMI_WRITE_SOC_REGISTER (=7)` Starting from second word of buffer:

	`A_UINT32`	`cmd;`	BMI command (BMI_READ_SOC_REGISTER)
	`A_UINT32`	`address;`	Address in the AR6001 address space (Should be word aligned)
	`A_UINT32`	`value`	Value to be written
Arguments	`scantype`	`0`	Do a long scan
		`1`	Do a short scan
		`forceFgScan`	Do foreground scan
IOCTL Type	`set`		
Class	iwpriv (private IOCTLs)		

Name	**AR6000_XIOCTL_BMI_TEST**
Description	Deprecated
Commands	N/A
Argument Type	N/A
Arguments	N/A
IOCTL Type	`N/A`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_STARTSCAN**
Description	Tell the AR6000 to start a long or short channel scan. All future scans are relative to the time that the AR6000 process this command. The AR6000 perform a channel scan on receipt of this command, even if a scan was already in progress. The host uses this command when it wishes to refresh its cached database of wireless networks. The isLegacy field is now removed.
Commands	**`wmiconfig eth1 --startscan <scan type> <forcefgscan>`**

Argument Type	`First word of the buffer: AR6000_XIOCTL_WMI_STARTSCAN; Starting from the second word of the buffer:`
	`struct {`
	`A_UINT8 scanType;`
	`A_BOOL  forceFgScan;`
	`} WMI_START_SCAN_CMD;`

Arguments	`scanType =` `0` Do a long scan (`WMI_LONG_SCAN`) `1` Do a short scan (`WMI_SHORT_SCAN`) `forceFgScan =` `0` Disable the foreground scan `1` Forces a foreground scan
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_HTC_RAW_OPEN**
Description	The HTC_RAW mechanism provides a way for the application to directly interact with the target, bypassing the WMI layer. This IOCTL must be issued before trying to use the AR6000_XIOCTL_HTC_RAW_READ or RAW_WRITE AR6000_XIOCTL_HTC_RAW_WRITE IOCTLs.
Commands	N/A
Argument Type	First word of buffer: `AR6000_XIOCTL_HTC_RAW_OPEN(=13)`
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_HTC_RAW_CLOSE**
Description	Closes the HTC RAW interface. No AR6000_XIOCTL_HTC_RAW_READ or RAW_WRITE AR6000_XIOCTL_HTC_RAW_WRITE IOCTLs should be issued after this IOCTL.
Commands	N/A
Argument Type	First word of buffer: `AR6000_XIOCTL_HTC_RAW_CLOSE (=14)`
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_HTC_RAW_READ**
Description	Reads a set of bytes from the specified mailbox. Prior to using this IOCTL, the AR6000_XIOCTL_HTC_RAW_OPEN IOCTL must be issued.
Commands	N/A
Argument Type	see below
IOCTL Type	`get`
Class	iwpriv (private IOCTLs)

`union {`		
`  struct {`		
`  UINT32`	`cmd`	AR6000_XIOCTL_HTC_RAW_READ (15)
`  UINT32`	`mailboxID`	Mailbox to read from. Legal values are 0 to 3.
`  UINT32`	`length`	Contains the read data
`  }`		
`results[length]`		Contains the read data
`}`		lengthread = Actual number of bytes successfully read

Name	**AR6000_XIOCTL_HTC_RAW_WRITE**
Description	Writes a set of bytes to the specified mailbox. Prior to using this IOCTL, the AR6000_XIOCTL_HTC_RAW_OPEN IOCTL must be issued.
Commands	N/A
Argument Type	see below
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

`struct {`		
`UINT32`	`cmd`	AR6000_XIOCTL_HTC_RAW_WRITE (16)
`UINT32`	`mailboxID`	Mailbox to write to; legal values are 0 to 3
`UINT32`	`length`	Length of data to write
`char`	`buffer[length]`	Contains the data to write
		lengthwritten = Actual number of bytes successfully written
`}`		

Name	**AR6000_XIOCTL_CHECK_TARGET_READY**
Description	Returns 0 if the target exists and is ready to accept IOCTLs (i.e., it has not been disabled using the AR6000_XIOCTRL_WMI_SET_WLAN_STATE IOCTL)
Commands	N/A
Argument Type	First word of buffer: `AR6000_XIOCTL_CHECK_TARGET_READY (=17)`

`UINT32`	`cmd`	AR6000_XIOCTL_HTC_RAW_WRITE (16)
`UINT32`	`mailboxID`	Mailbox to read from. Legal values are 0 to 3.
`UINT32`	`length`	Length of data to read
`char`	`buffer[length]`	
}		

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_GPIO_OUTPUT_SET**
Description	Manages output on GPIO pins configured for output. Conflicts between set_mask and clear_mask or enable_mask and disable_mask result in undefined behavior.
Commands	N/A
Argument Type	First word of buffer: `AR6000_XIOCTL_GPIO_OUTPUT_SET (=18)`

Starting from second word of buffer:

`struct ar6000_gpio_output_set_cmd_s {`

`A_UINT32`	`set_mask;`	Specifies which pins should drive a 1 out
`A_UINT32`	`clear_mask;`	Specifies which pins should drive a 0 out
`A_UINT32`	`enable_mask;`	Specifies which pins should be enabled
`A_UINT32`	`disable_mask;`	Specifies which pins should be disabled
`};`		

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_GPIO_INPUT_GET**
Description	Allows the host to read the GPIO pins that are configured for input
Commands	N/A
Argument Type	First word of buffer: `AR6000_XIOCTL_GPIO_INPUT_GET (=19)`

Starting from second word of buffer:

`A_UINT32`	`value;`	The GPIO pins that are configured for input (written by driver)

IOCTL Type	`get`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_GPIO_REGISTER_SET**
Description	Allows the host to dynamically change GPIO configuration (usually handled statically through the GPIO configuration DataSet).
Commands	N/A
Argument Type	First word of buffer: `AR6000_XIOCTL_GPIO_REGISTER_SET (=20)`

Starting from second word of buffer:

```
 struct ar6000_gpio_register_cmd_s {
A_UINT32        gpioreg_id;        GPIO register ID
A_UINT32        value;             Value to write
}
```

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_GPIO_REGISTER_GET**
Description	Allows the host to read an arbitrary GPIO register. Intended for use during bringup / debug.
Commands	N/A
Argument Type	First word of buffer: `AR6000_XIOCTL_GPIO_REGISTER_GET (21)`

Starting from second word of buffer:

```
struct ar6000_gpio_register_cmd_s {
A_UINT32        gpioreg_id;        GPIO register ID
A_UINT32        value;             Value read from the register (filled by
};                                 driver)
```

IOCTL Type	`get`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_GPIO_INTR_ACK**
Description	The host uses this command to acknowledge and to re-arm APIO interrupts reported through and earlier GPIO_INTR extension event
Commands	N/A
Argument Type	First word of buffer: `AR6000_XIOCTL_GPIO_INTR_WAIT (=22)`

Starting from second word of buffer:

```
struct ar6000_gpio_intr_ack_cmd_s {
A_UINT32        ack_mask;          A mask of interrupting GPIO pings (e.g.,
}                                  bit 3 of ack_mask acknowledges an
                                   interrupt from the pin GPIO3)
```

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_GPIO_INTR_WAIT**
Description	Waits for a GPIO interrupt until one occurs, then returns the information about the interrupt(s)
Commands	N/A

Argument Type	First word of buffer: `AR6000_XIOCTL_GPIO_INTR_WAIT (=23)`
	Starting from second word of buffer:
	`struct ar6000_gpio_intr_wait_cmd_s {`
	`A_UINT32` `intr_mask` GPIO interrupt mask (filled by driver)
	`A_UINT32` `input_values;` Current data values at the GPIO pin interrupts (filled by driver)
	`}`

IOCTL Type	`get`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_SET_ADHOC_BSSID**
Description	Allows the host to set the BSSID for an ad hoc network. If a network with this BSSID is not found, the target creates an ad hoc network with this BSSID after the connect WMI command is triggered (e.g., by the SIOCSIWESSID IOCTL).
Commands	**`wmiconfig eth1 --adhocbssid <bssid>`** Where bssid is in the xx:xx:xx:xx:xx:xx format

Argument Type	First word of buffer: `AR6000_XIOCTL_SET_ADHOC_BSSID (=24)`
	Starting from second word of buffer:
	`struct {`
	`A_UINT8` `bssid[6]` BSSID of the ad hoc network to connect to or create. A broadcast address (FF:FF:FF:FF:FF:FF) is not allowed.
	`} WMI_SET_ADHOC_BSSID_CMD;`

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_SET_ADHOC_BEACON_INTVAL**
Description	Sets the beacon interval for an ad hoc network
Commands	**`wmiconfig eth1 --ibssbconintvl <interval>`**

Argument Type	First word of buffer: `AR6000_XIOCTL_SET_ADHOC_BEACON_INTVAL (=27)`
	Starting from second word of buffer:
	`struct {`
	`A_UINT16` `beaconInterval;` Beacon interval in TUs (1 TU = 1024 μs)
	`} WMI_BEACON_INT_CMD;`

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_SET_VOICE_PKT_SIZE**
Description	Allows the host to control which packets the target treats as voice packets when WMM is not available. If WMM is available this IOCTL has no effect.
Commands	`wmiconfig eth1 --setVoicePktSize <size-in-bytes>`

Argument Type	First word of buffer: `AR6000_XIOCTL_SET_VOICE_PKT_SIZE (=29)`
	Starting from second word of buffer:
	`struct {`
	`A_UINT16 voicePktSize;` — Packets shorter than this size are treated as voice packets on non-WMM connections
	`} WMI_SET_VOICE_PKT_SIZE_CMD;`

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_SET_MAX_SP**
Description	Sets the maximal service period, which is the maximum number of total buffered packets the AP may deliver to the station during any service period triggered by the station. Applicable only to APSD-enabled connections.
Commands	`wmiconfig eth1 --setMaxSPLength` *`<maxSPLen>`*

Argument Type	First word of buffer: `AR6000_XIOCTL_SET_MAX_SP (=30)`
	Starting from second word of buffer:
	`struct {`
	`A_UINT8 maxSPLen;` — ■ 0 = deliver all packets ■ 1 = deliver up to 2 packets ■ 2 = deliver up to 4 packets ■ 3 = deliver up to 6 packets
	`} WMI_SET_MAX_SP_LEN_CMD;`

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_GET_ROAM_TBL**
Description	Retrieves the roam table maintained by the target and contains all the APs that to roam to
Commands	`wmiconfig -i eth1 --getroamtable`
Argument Type	None
IOCTL Type	`get`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_SET_ROAM_CTRL**
Description	Sets roam mode, forces a roam to a specific BSSID or sets the host bias for a BSSID
Commands	`wmiconfig -i eth1 --roam <roamctrl> <info>`
Argument Type	First word of buffer: `AR6000_XIOCTL_SET_VOICE_PKT_SIZE (=29)` Starting from second word of buffer: `struct {` `A_UINT8 roamCtrlType;` — Packets shorter than this size are treated as voice packets on non-WMM connections ■ 1 = Force a roam to specified bssid ■ 2 = Sets the roam mode ■ 3 = Sets the host bias of the BSSID

	`union {`
	`A_UINT8    bssid[ATH_MAC_LEN];` WMI_FORCE_ROAM
	`A_UINT8    roamMode;` WMI_SET_ROAM_MODE
	`WMI_BSS_ BIAS_INFO    bssBiasInfo;` WMI_SET_HOST_BIAS ■ roamctrl = 1: Info is BSSID<*aa:bb:cc:dd:ee:ff*> for roamctrl = 1 ■ roamctrl = 2: Info = DEFAULT, BSSBIAS, or LOCK ■ roamctrl = 3: BSSID<*aa:bb:cc:dd:ee:ff*> <*bias*>; bias = Value between –256 and 2
	`} info;`
	`} WMI_SET_ROAM_CTRL_CMD;`
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTRL_WMI_SET_POWERSAVE_TIMERS**
Description	Set the power save timers for PS-POLL and triggers
Commands	**`wmiconfig -i eth1 --psPollTimer <value> --eospTimer <value>`**

Argument Type	First word of buffer: `AR6000_XIOCTL_SET_VOICE_PKT_SIZE (=29)`
	Starting from second word of buffer:
	`struct {`
	`A_UINT16    psPollTimeout;` The timeout after which a PS-POLL is sent to fetch data from the AP (in ms)
	`A_UINT16    eospTimeout;` The timeout after which a trigger is sent to fetch data from the AP (in ms)
	`} WMI_POWERSAVE_TIMERS_CMD;`

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTRL_WMI_GET_POWER_MODE**
Description	Retrieves the target power mode; power mode may be performance or power save
Commands	**`wmiconfig -i eth1 --getpower`**
Argument Type	None
IOCTL Type	`get`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTRL_WMI_SET_WLAN_STATE**
Description	Enables or disables the WLAN capability of the target. If connected, disabling the WLAN disconnects the target and stop all Rx/Tx. Enabling WLAN reconnects the target if previously connected and enable Rx/Tx functions.
Commands	`wmiconfig -i eth1 --wlan <enable/disable>`
Argument Type	First word of buffer: `AR6000_XIOCTL_SET_VOICE_PKT_SIZE (=29)`

Starting from second word of buffer:

```
 enum {
```

`WLAN_DISABLED`	Disables WLAN
`WLAN_ENABLED`	Enables WLAN

```
} AR6000_WLAN_STATE;
```

IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_SET_WSC_STATUS**
Description	The supplicant uses this command to inform the target about the status of the WSC registration protocol. During the WSC registration protocol, a flag will be set so the target bypasses some of the checks in the CSERV module. At the end of the registration, this flag is reset.
Commands	N/A (Not available in wmiconfig application)
Argument Type	int
Arguments	int status WSC_REG_ACTIVE (1) or WSC_REG_INACTIVE (0)
IOCTL Type	`set`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_GET_ROAM_DATA**
Description	Retrieves the roam data maintained by the target, currently restricted to roam time
Commands	`wmiconfig -i eth1 --getroamdata`
Argument Type	First word of buffer: AR6000_XIOCTL_WMI_GET_ROAM_DATA starting from second word of buffer

```
struct {
```

`A_UINT8`	`roamDataType;`	Type of roaming data we want to get, e.g., ROAM_DATA_TIME

```
union {
WMI_TARGET_ROAM_TIME roamTime;} u; Holds the roam time data
} WMI_TARGET_ROAM_DATA;
```

IOCTL Type	`get`
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_WMI_TARGET_EVENT_REPORT**
Description	The host uses this command to filter out the events from the target that the host is not interested in. Current SW release supports filtering out the WMI_DISCONNECTED_EVENT with reason = BSS_DISCONNECTED when host or target reconnects with the AP. The reconnection sequence with AP (e.g. RECONNECT WMI command) causes target to send a WMI_DISCONNECTED_EVENT with reason = CSERV_DISCONNECT first followed by WMI_DISCONNECTED_EVENT with reason = BSS_DISCONNECTED. If the host is not interested in the second WMI_DISCONNECTED_EVENT with reason = BSS_DISCONNECTED, host can use this command to do so.
Commands	**`wmiconfig –eth1 --settgtevt <event value>`**
Argument Type	<event value> This is enum parameter of type TARGET_EVENT_REPORT_CONFIG `event value = 0` Send WMI_DISCONNECT_EVENT with disconnectReason = BSS_DISCONNECTED after re-connection with AP `event value = 1` Do NOT send WMI_DISCONNECT_EVENT with disconnectReason = BSS_DISCONNECTED after re-connection with AP. Reconnection with the AP already sends DISCONNECT_EVENT with disconnectReason = CSERV_DISCONNECT first.
Reset Value	event value = 0
Restrictions	None
IOCTL Type	Set
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_SET_IP**
Description	Conveys the IP binding to target. The IP parameter value should be in nework byte order. This command is meant for ARP offloading feature
Commands	**`wmiconfig -i eth1 --ip x.x.x.x`**

Argument	`struct {`
	`A_UINT32 ips     [WOW_MAX_ARP_IP_ADDRS];`
	`} WMI_SET_IP_CMD;`

IOCTL Type	Set
Class	iwpriv (private IOCTLs)

Name	**AR6000_XIOCTL_AP_SET_ACL_POLICY**
Description	Sets policy for ACL. Depending on the policy STAs in the ACL list will be either allowed or denied to connect. If ACL policy is disabled, any STA can connect to AP and ACL list will not be used in this case.
Commands	**`wmiconfig -i eth1 --aclpolicy <policy> <retain>`**

Argument	`<Policy>`
	`0 Disable`
	`1 Allow MAC in the ACL list`
	`2 Deny the MAC in the ACL list`
	`<Retain>`
	`0 Clear the existing ACL list`

```
  1 Retain the existing ACL list
#define AP_ACL_DISABLE          0x00
#define AP_ACL_ALLOW_MAC        0x01
#define AP_ACL_DENY_MAC         0x02
#define AP_ACL_RETAIN_LIST_MASK 0x80
struct {
    A_UINT8     policy;
} WMI_AP_ACL_POLICY_CMD;
Policy1 = Either AP_ACL_DISABLE or
AP_ACL_ALLOW_MAC or AP_ACL_DENY_MAC
To retain, policy = policy1 |
AP_ACL_RETAIN_LIST_MASK
To clear, policy = policy1
```

IOCTL Type Set

Class iwpriv (private IOCTLs)

Name **AR6000_XIOCTL_AP_INTRA_BSS_COMM**

Description Enables/disables communication between connected STAs

Commands **`wmiconfig -i eth1 --intrabss <control>`**

Argument `<Control>`

0	Disable
1	Enable

IOCTL Type Set

Class iwpriv (private IOCTLs)

Event Notifications

Table H-4 shows the AR6000_IOCTL_EXTENDED Linux private IOCTLs.

Table H-4. **AR6000_IOCTL_EXTENDED**

IOCTL	Description	Page
WMI_READY_EVENT	The AR6001 is prepared to accept commands	page H-56
WMI_CONNECTED_EVENT	The AR6001 has connected to a wireless network	page H-57
WMI_DISCONNECTED_EVENT	The AR6001 has lost connectivity with the network	page H-57
WMI_BSSINFO_EVENT	Contains information describing one or more BSSs	page H-57
WMI_CMDERROR_EVENT	Indicates an error report by the target	page H-58
WMI_REGDOMAIN_EVENT	Used by the target to indicate the regulatory domain	page H-58
WMI_PSTREAM_TIMEOUT_EVENT	Indicates a prioritized stream has been idle	page H-58
WMI_NEIGHBOR_REPORT_EVENT	Indicates the APs in the neighborhood	page H-58
WMI_TKIP_MICERR_EVENT	Reports a TKIP MIC error to host	page H-59
WMI_SCAN_COMPLETE_EVENTID	Signals that a scan has been completed	page H-59
WMI_REPORT_STATISTICS_EVENTID	Reply to the GET_STATISTICS command	page H-59
WMI_RSSI_THRESHOLD_EVENTID	Indicates that certain RSSI threshold was hit	page H-61
WMI_ERROR_REPORT_EVENT	Signals that a type of error has occurred	page H-61
WMI_REPORT_ROAM_TBL_EVENTID	Signals that a type of error has occurred	page H-62
WMI_EXTENSION_EVENTID	Used for a collection of events (not generic)	page H-62
WMI_CAC_EVENTID	Indicates signalling events from the AR6001	page H-63
WMI_SNR_THRESHOLD_EVENTID	Indicates that certain SNR threshold was hit	page H-63
WMI_LQ_THRESHOLD_EVENTID	Indicates that certain Link Quality threshold was hit	page H-63
WMI_TX_RETRY_ERR_EVENTID	Indicates that maximum Tx retries time was exceeded	page H-64
WMI_REPORT_ROAM_DATA_EVENTID	Generated with a special build; reports the roam time calculations made by the AR6000	—
WMI_TEST_EVENTID	Refers to events generated by the TCMD	page H-64
WMI_APLIST_EVENTID	Not supported; available for customers with CCX license	—
WMI_GET_WOW_LIST_EVENTID	Lists the number of patterns that have been programmed in the AR6000	page H-64
WMI_PEER_NODE_EVENTID	The AR6000 accepted an IBSS joiner or peer node connection	page H-65
WMI_PSPOLL_EVENTID	The AR6000 receives PSPOLL requests in SoftAP mode	page H-65
WMI_DTIMEXPIRY_EVENTID	The AR6000 DTIM expired in SoftAP mode	page H-65
WMI_WLAN_VERSION_EVENTID	Sends the AR6000 version info details to the host	page H-65
WMI_APLIST_EVENTID	Sends the neighbor AP information to the host	page H-66

Name **WMI_READY_EVENT**

Description Indicates that the AR6000 is prepared to accept commands

Event ID 0x1001

Argument Type

```
struct {
A_UINT8      macaddr[ATH_MAC_LEN];
A_UINT8      phyCapability;
} WMI_READY_EVENT;
```

A_UINT8	macaddr[ATH_MAC_LEN];	AR6001 MAC address
A_UINT8	phyCapability;	Indicates the capabilities of the AR6001 wireless modules's radio. ■ 1 = 802.11a ■ 2 = 802.11b/g ■ 3 = 802.11a/b/g

Name	**WMI_CONNECTED_EVENT**
Description	Signals that the AR6001 has connected to a wireless network
Event ID	0x1002
Argument Type	`struct {`

`A_UINT16`	`channel;`	
`A_UINT8`	`bssid[ATH_MAC_LEN];`	
`A_UINT16`	`listenInterval;`	
`A_UINT16`	`beaconInterval;`	
`A_UINT8`	`assocReqLen;`	Length (in bytes) of the assocReq array
`A_UINT8`	`assocRespLen;`	Length (in bytes) of the assocResp array
`A_UINT8`	`assocInfo[1];`	Pointer to the association request/ response frame
`}WMI_CONNECT_EVENT;`		

Name	**WMI_DISCONNECTED_EVENT**
Description	Signals that the AR6001 has lost connectivity with the wireless network
Event ID	0x1003
Argument Type	`struct {`

`A_UINT8`	`disconnectReason;`	A value to indicate the reason of disconnect
`A_UINT8`	`bssid[ATH_MAC_LEN];`	Indicate which BSS the AR6001 was disconnected from
`A_UINT8`	`assocRespLen;`	Length (in bytes) of the assocResp array
`A_UINT8`	`assocInfo[assocRespLen];`	Contain the complete association response frame
`} WMI_DISCONNECT_EVENT;`		

Name	**WMI_BSSINFO_EVENT**
Description	Contains information describing one or more BSSs as collected during a scan operation
Event ID	0x1004
Argument Type	`struct {`

`A_UINT16`	`channel;`	
`A_UINT8`	`frameType;`	■ 1 = Beacon frame ■ 2 = Probe response frame
`A_UINT16`	`rssi;`	
`A_UINT8`	`bssid[ATH_MAC_LEN];`	
`} WMI_BSS_INFO_HDR;`		

Name	**WMI_CMDERROR_EVENT**
Description	Indicates an error report by the target. The target only checks for a common violations and it is the responsibility of the host to do all error checking. Behavior of target after wmi error is undefined.
Event ID	0x1005
Argument Type	`struct {`

`A_UINT16`	`commandId;`	WMI command in error
`A_UINT8`	`errorCode;`	{INVALID_PARAM, ILLEGAL_STATE, INTERNAL_ERROR}

`} WMI_CMD_ERROR_EVENT;`

Name	**WMI_REGDOMAIN_EVENT**
Description	Used by the target to indicate the regulatory domain
Event ID	0x1006
Argument Type	`struct {`

`A_UINT32`	`regDomain;`	Regulatory domain

`} WMI_REG_DOMAIN_EVENT;`

Name	**WMI_PSTREAM_TIMEOUT_EVENT**
Description	Indicates that a priority stream that got created as a result of priority-marked data flow (priority marked in IP TOS) being idle for the default inactivity interval period (specified in the **“CREATE_PSTREAM”** command) used for priority streams created implicitly by the driver. This event is not indicated for user-created priority streams. User-created priority streams exist until the users delete them explicitly. They do not timeout due to data inactivity.
Event ID	0x1007
Argument Type	`struct {`

```
A_UINT8       trafficClass;
} PSTREAM_TIMEOUT_EVENT;
```

Name	**WMI_NEIGHBOR_REPORT_EVENT**
Description	Indicates the APs in the neighborhood along with their properties
Event ID	0x1008
Argument Type	`struct {`

`A_INT8`	`numberOfAps;`	Number of APs being reported
`WMI_NEIGHBOR_INFO`	`neighbor[1];`	Pointer to an array of neighbors of type WMI_NEIGHBOR_INFO

`} WMI_NEIGHBOR_REPORT_EVENT;`

`struct {`

`A_UINT8`	`bssid[ATH_MAC_LEN];`	BSSID of the AP
`A_UINT8`	`bssFlags;`	Indicates properties of type WMI_BSS_FLAGS

`} WMI_NEIGHBOR_INFO;`

Name	**WMI_TKIP_MICERR_EVENT**
Description	Reports a TKIP MIC error to host
Event ID	0x1009
Argument Type	`struct {`

```
struct {
A_INT8              keyid;                    802.11 key ID
WMI_NEIGHBOR_INFO   ismcast;                  Multicast or unicast
} WMI_TKIP_MICERR_EVENT;
```

Name	**WMI_SCAN_COMPLETE_EVENTID**
Description	Signals that a scan has been completed
Event ID	0x100A
Argument Type	None

Name	**WMI_REPORT_STATISTICS_EVENTID**
Description	Reply to the GET_STATISTICS command issued by the host. It arguments are a variety of statistics.
Event ID	0x100B
Argument Type	`struct {`

```
struct {
wlan_net_stats_t   txrxStats;
cserv_stats_t      cservStats;
pm_stats_t         pmStats;
A_INT16            noise_floor_calibration;
} WMI_TARGET_STATS;
struct {
A_UINT32           power_save_failure_cnt;
} pm_stats_t;
struct {
A_UINT32           cs_bmiss_cnt;
A_UINT32           cs_lowRssi_cnt;
A_UINT16           cs_connect_cnt;
A_UINT16           cs_disconnect_cnt;
A_UINT8            cs_aveBeacon_rssi;
A_UINT8            cs_lastRoam_msec;
} cserv_stats_t;
struct {
tx_stats_t         tx_stats;
rx_stats_t         rx_stats;
tkip_ccmp_stats_t  tkipCcmpStats;
} wlan_net_stats_t;
struct {
A_UINT32           tx_packets;
A_UINT32           tx_bytes;
A_UINT32           tx_unicast_pkts;
A_UINT32           tx_unicast_bytes;
A_UINT32           tx_multicast_pkts;
A_UINT32           tx_multicast_bytes;
```

... WMI_REPORT_STATISTICS_EVENTID, continued

```
A_UINT32            tx_broadcast_pkts;
A_UINT32            tx_broadcast_bytes;
A_UINT32            tx_rts_success_cnt;
A_UINT32            tx_packet_per_ac[4];
A_UINT32            tx_errors;
A_UINT32            tx_failed_cnt;
A_UINT32            tx_retry_cnt;
A_UINT32            tx_rts_fail_cnt;
} tx_stats_t;
struct {
A_UINT32            rx_packets;
A_UINT32            rx_bytes;
A_UINT32            rx_unicast_pkts;
A_UINT32            rx_unicast_bytes;
A_UINT32            rx_multicast_pkts;
A_UINT32            rx_multicast_bytes;
A_UINT32            rx_broadcast_pkts;
A_UINT32            rx_broadcast_bytes;
A_UINT32            rx_fragment_pkt;
A_UINT32            rx_errors;
A_UINT32            rx_crcerr;
A_UINT32            rx_key_cache_miss;
A_UINT32            rx_decrypt_err;
A_UINT32            rx_duplicate_frames;
} rx_stats_t;
struct {
A_UINT32            tkip_local_mic_failure;
A_UINT32            tkip_counter_measures_
                    invoked;
A_UINT32            tkip_replays;
A_UINT32            tkip_format_errors;
A_UINT32            ccmp_format_errors;
A_UINT32            ccmp_replays;
} tkip_ccmp_stats_t;
A_UINT32
A_UINT32
A_UINT32
```

Name	**WMI_RSSI_THRESHOLD_EVENTID**
Description	Indicates that certain RSSI threshold was hit
Event ID	0x100C

Argument Type

```
struct {
WMI_RSSI_LOWTHRESHOLD_BELOW_LOWERVAL    =1
WMI_RSSI_LOWTHRESHOLD_LOWERVAL
WMI_RSSI_LOWTHRESHOLD_UPPERVAL
WMI_RSSI_HIGHTHRESHOLD_LOWERVAL
WMI_RSSI_HIGHTHRESHOLD_HIGHERVAL
}WMI_RSSI_THRESHOLD_VAL;
struct {
A_UINT8           range;
} WMI_RSSI_THRESHOLD_EVENT;
```

Argument

- 1 = RSSI is below the lower value of the low threshold
- 2 = RSSI has crossed lower value of the low threshold
- 3 = RSSI has crossed upper value of the low threshold
- 4 = RSSI has crossed lower value of the high threshold
- 5 = RSSI has crossed higher value of the high threshold

Name	**WMI_ERROR_REPORT_EVENT**
Description	Signals that a type of error has occurred for which the host previously requested notification
Event ID	0x100D

Argument Type

```
struct {
A_UINT32          errorVal;
```

- 0x00000001 = Power save fails
- 0x00000002 = No cipher key
- 0x00000004 = Decryption error
- 0x00000008 = Beacon Miss
- 0x00000010 = A non-power save disabled node has joined the PS-enabled network

```
}WMI_TARGET_ERROR_REPORT_EVENT;
```

Name	**WMI_REPORT_ROAM_TBL_EVENTID**
Description	Signals that a type of error has occurred for which the host previously requested notification
Event ID	0x100F
Argument Type	see below

```
struct {
A_UINT8            bssid[ATH_MAC_LEN];
A_INT8             rssi;
A_INT8             rssidt;
A_INT8             last_rssi;
A_INT32            roam_util;
A_INT8             util;
A_INT8             bias;
} WMI_BSS_ROAM_INFO;
struct {
A_UINT8            roamMode;
```

Current roam mode

- 1 = RSSI based roam
- 2 = Host bias based roam
- 3 = Lock to the current BSS
- 4 = Autonomous roaming disabled

```
A_UINT8            numEntries;
WMI_BSS_ROAM_INFO  bssRoamInfo[1];
} WMI_TARGET_ROAM_TBL;
```

Name	**WMI_EXTENSION_EVENTID**
Description	Used for a collection of events that are not generic wireless events and may be implementation or platform specific or even optional.
Event ID	0x1010
Argument Type	see below

```
enum {
WMIX_DSETOPENREQ_EVENTID               = 0x3001,
WMIX_DSETCLOSE_EVENTID,
WMIX_DSETDATAREQ_EVENTID,
WMIX_GPIO_INTR_EVENTID,
WMIX_GPIO_DATA_EVENTID,
WMIX_GPIO_ACK_EVENTID,
} WMIX_EVENT_ID;
```

Name	**WMI_CAC_EVENTID**
Description	Indicates signalling events from the AR6001. Events are generated when admission is accepted, deleted, rejected or not responded.
Event ID	0x1011
Argument Type	`enum {`

`A_UINT8`	`ac;`	Access class
`A_UINT8`	`cac_indication;`	CAC_INDICATION
`A_UINT8`	`statusCode;`	Status code
`A_UINT8`	`tspecSuggestion[63];`	Suggested tspec from AP

`} WMI_CAC_EVENT;`

cac_indication enum {

```
CAC_INDICATION_AXDMISSION                    = 0x00
CAC_INDICATION_ADMISSION_RESP                = 0x01
CAC_INDICATION_DELETE                        = 0x02
CAC_INDICATION_NO_RESP                       = 0x03
} CAC_INDICATION;
```

Name	**WMI_SNR_THRESHOLD_EVENTID**
Description	Indicates that certain Link Quality threshold was hit.
Event ID	0x1012
Argument Type	`struct {`

```
A_UINT8            range;                  WMI_SNR_THRESHOLD_VAL
A_UINT8            snr;
} WMI_SNR_THRESHOLD_EVENT;
```

Argument

`range =`	`1`	`SNR threshold 1 above`
	`2`	`SNR threshold 1 below`
	`...`	`...`
	`7`	`SNR threshold 4 above`
	`8`	`SNR threshold 4 below`
`lq`		`Current link quality value`

Name	**WMI_LQ_THRESHOLD_EVENTID**
Description	Indicates that certain Link Quality threshold was hit.
Event ID	0x1013
Argument Type	`struct {`

```
A_UINT8            range;
A_UINT8            lq
} WMI_LQ_THRESHOLD_EVENT;
```

Argument

`range =`	`1`	`SNR threshold 1 above`
	`2`	`SNR threshold 1 below`
	`...`	`...`
	`7`	`SNR threshold 4 above`
	`8`	`SNR threshold 4 below`
`lq`		`Current link quality value`

Name **WMI_TX_RETRY_ERR_EVENTID**

Description Indicates that maximum Tx retries time was exceeded.

Event ID 0x1014

Argument Type
```
struct {
A_UINT8           retrys;
} WMI_TX_RETRY_ERR_EVENT;
```

Argument
```
retrys =          retried themes
```

Name **WMI_TEST_EVENTID**

Description Refers to events generated by the TCMD

Event ID 0x1016

Argument Type
```
struct {
A_UINT8         testCmdId;
A_UINT32        act;
A_UINT32        enANI;
    union {
        struct TCMD_CONT_RX_PARA {
            A_UINT32    freq;
            A_UINT32    antenna;
        }__ATTRIB_PACK para;
        struct TCMD_CONT_RX_REPORT {
            A_UINT32    totalPkt;
            A_INT32    rssiInDBm;
        }__ATTRIB_PACK  report;
        struct TCMD_CONT_RX_MAC {
            A_UCHAR    addr[ATH_MAC_LEN];
        }__ATTRIB_PACK mac;
    }u;
} TCMD_CONT_RX;
```

Name **WMI_GET_WOW_LIST_EVENTID**

Description Lists the number of patterns that have been programmed in the AR6000.

Event ID 0x1019

Argument Type
```
struct {
A_UINT8           /* number of patterns in reply */
A_UINT8           /* this is filter # x of total num_filters */
A_UINT8
A_UINT8
WOW_FILTER
} WMI_GET_WOW_LIST_REPLY;
```

Name	**WMI_PEER_NODE_EVENTID**
Description	The AR6000 has accepted IBSS joiner or peer node connection
Event ID	0x101B
Argument Type	`struct {`

```
A_UINT8           eventCode;      PEER_NODE_JOIN_EVENT 0x00
                                  PEER_NODE_LEAVE_EVENT 0x01
                                  PEER_FIRST_NODE_JOIN_EVENT 0x10
                                  PEER_LAST_NODE_LEAVE_EVENT 0x11
A_UINT8           peerMacAddr[ATH_MAC_LEN];
 }
```

Name	**WMI_PSPOLL_EVENTID**
Description	The AR6000 receives PSPOLL request in SoftAP mode
Event ID	0x101C
Argument Type	`struct {`

```
A_UINT16 aid;                                   Connected STA AID
}
```

Name	**WMI_DTIMEXPIRY_EVENTID**
Description	The AR6000 DTIM expired in SoftAP mode
Event ID	0x101D
Argument Type	None

Name	**WMI_WLAN_VERSION_EVENTID**
Description	Sends the AR6000 version info details to the host
Event ID	0x101E
Argument Type	`struct {`

```
A_UINT32          version
}
```

Name	**WMI_APLIST_EVENTID**
Description	Sends the neighbor AP information to the host
Event ID	0x1017

Argument Type

```
enum {
APLIST_VER1        = 1,
} APLIST_VER;
struct {
A_UINT8            bssid;
A_UINT16           channel;
} WMI_AP_INFO_V1;
union {
WMI_AP_INFO_V1     apInfoV1;
} WMI_AP_INFO;
struct {
A_UINT8            apListVer;
A_UINT8            numAP;
WMI_AP_INFO        apList[1];
}
```

I

WinMobile

Architecture

The AR6000 client device interfaces with the host system through an SDIO interface. The AR6000 host driver is an NDIS Miniport driver. Figure I-I-1 describes how the driver interfaces with the Windows Mobile (WinMobile)/ WinCE components subsystem.

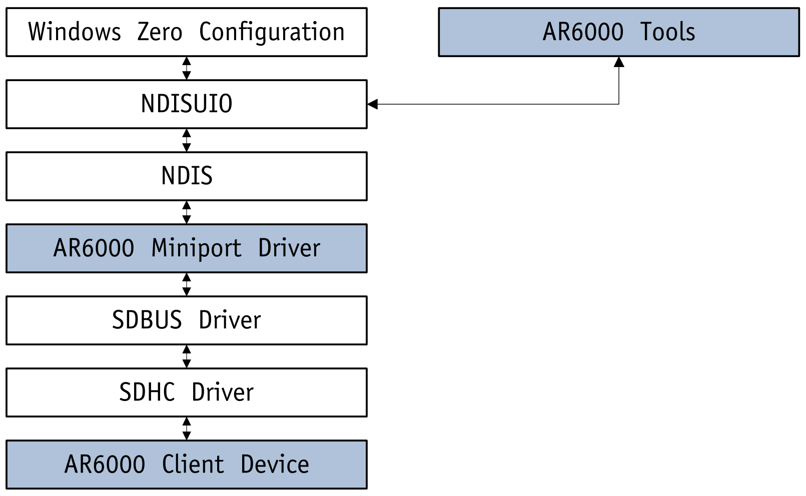

Figure I-1. **Architecture**

The AR6000 Miniport driver, AR6000 tools and AR6000 client device are Atheros-developed components and WZC, NDISUIO, NDIS and SDBUS driver are part of Windows Mobile OS. The SD Host controller is either from Windows Mobile OS or from a Vendor BSP component.

DEVICE IOCTLs

When an AR6000 card is inserted, a stream interface (DRG) is created and an NDIS interface (AR6K_SD1) is derived from the DRG stream interface. The AR6000 stream interface provides BMI-specific IOCTLS and generic IOCTLS to get driver and target versions.

Generic IOCTLs

The generic IOCTLs include:

- IOCTL_CAR6K_NDIS_REGISTER
- IOCTL_CAR6K_GET_FIRMWARE_VERSION
- IOCTL_CAR6K_GET_HOST_VERSION
- IOCTL_CAR6K_GET_TARGET_TYPE
- IOCTL_CAR6K_GET_DBGLOG
- IOCTL_CAR6K_CONFIGURE_AR6000

BMI-Specific IOCTLs

The BMI-specific IOCTLs include:

- IOCTL_CAR6K_BMI_DONE
- IOCTL_CAR6K_BMI_READ_MEMORY
- IOCTL_CAR6K_BMI_WRITE_MEMORY
- IOCTL_CAR6K_BMI_EXECUTE
- IOCTL_CAR6K_BMI_SET_APP_START
- IOCTL_CAR6K_BMI_READ_SOC_REGISTER
- IOCTL_CAR6K_BMI_WRITE_SOC_REGISTER
- IOCTL_CAR6K_BMI_ROMPATCH_INSTALL
- IOCTL_CAR6K_BMI_ROMPATCH_UNINSTALL
- IOCTL_CAR6K_BMI_ROMPATCH_ACTIVATE
- IOCTL_CAR6K_BMI_ROMPATCH_DEACTIVATE

HTC-Specific IOCTLs

The HTC-specific IOCTLs include:

- IOCTL_CAR6K_HTC_RAW_READ
- IOCTL_CAR6K_HTC_RAW_WRITE
- IOCTL_CAR6K_HTC_RAW_OPEN

- IOCTL_CAR6K_HTC_RAW_CLOSE

OIDs

The NIC is controlled through NDIS OIDs and Atheros Private OIDs; OIDs are implemented through AR6000 WMI commands and map to one or more WMI commands, which are sent to the AR6000 target. The target sends control information through events and the host driver generates appropriate NDIS events to inform the status change. See Figure I-I-2.

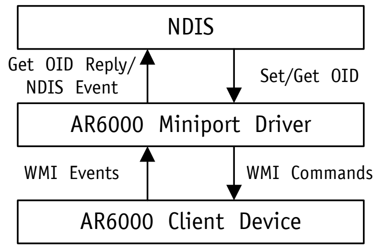

Figure I-2. **OIDs**

NDIS OIDs

The AR6000 host driver supports Microsoft 802.11 NIDS query and set OIDs. See Microsoft documentation for details about these OIDs.

Query OIDs

- OID_802_11_ASSOCIATION_INFORMATION
- OID_802_11_AUTHENTICATION_MODE
- OID_802_11_BSSID
- OID_802_11_BSSID_LIST
- OID_802_11_CAPABILITY
- OID_802_11_CONFIGURATION
- OID_802_11_ENCRYPTION_STATUS
- OID_802_11_INFRASTRUCTURE_MODE
- OID_802_11_NETWORK_TYPE_IN_USE
- OID_802_11_NETWORK_TYPES_SUPPORTED
- OID_802_11_PMKIDOID_802_3_PERMANENT_ADDRESS
- OID_802_11_POWER_MODE
- OID_802_11_RSSI
- OID_802_11_SSID
- OID_802_11_STATISTICS
- OID_802_11_SUPPORTED_RATES
- OID_802_3_CURRENT_ADDRESS
- OID_802_3_MAC_OPTIONS
- OID_802_3_MAXIMUM_LIST_SIZE
- OID_802_3_MULTICAST_LIST
- OID_802_3_RCV_ERROR_ALIGNMENT
- OID_802_3_XMIT_MORE_COLLISIONS
- OID_802_3_XMIT_ONE_COLLISION
- OID_GEN_CURRENT_LOOKAHEAD
- OID_GEN_CURRENT_PACKET_FILTER
- OID_GEN_DRIVER_VERSION
- OID_GEN_HARDWARE_STATUS
- OID_GEN_LINK_SPEED
- OID_GEN_MAXIMUM_FRAME_SIZE
- OID_GEN_MAXIMUM_LOOKAHEAD

- OID_GEN_MAXIMUM_SEND_PACKETS
- OID_GEN_MAXIMUM_TOTAL_SIZE
- OID_GEN_MEDIA_CONNECT_STATUS
- OID_GEN_MAC_OPTIONS
- OID_GEN_MEDIA_CAPABILITIES
- OID_GEN_MEDIA_IN_USE
- OID_GEN_MEDIA_SUPPORTED
- OID_GEN_PHYSICAL_MEDIUM
- OID_GEN_RCV_ERROR
- OID_GEN_RCV_NO_BUFFER
- OID_GEN_RCV_OK
- OID_GEN_RECEIVE_BLOCK_SIZE
- OID_GEN_RECEIVE_BUFFER_SPACE
- OID_GEN_SUPPORTED_LIST
- OID_GEN_TRANSMIT_BLOCK_SIZE
- OID_GEN_TRANSMIT_BUFFER_SPACE
- OID_GEN_VENDOR_DESCRIPTION
- OID_GEN_VENDOR_DRIVER_VERSION
- OID_GEN_VENDOR_ID
- OID_GEN_XMIT_OK
- OID_GEN_XMIT_ERROR
- OID_PNP_CAPABILITIES
- OID_PNP_QUERY_POWER

Set OIDs

- OID_802_11_ADD_KEY
- OID_802_11_ADD_WEP
- OID_802_11_ASSOCIATION_INFORMATION
- OID_802_11_AUTHENTICATION_MODE
- OID_802_11_BSSID
- OID_802_11_BSSID_LIST_SCAN
- OID_802_11_CONFIGURATION
- OID_802_11_DISASSOCIATE
- OID_802_11_ENCRYPTION_STATUS
- OID_802_11_INFRASTRUCTURE_MODE
- OID_802_11_NETWORK_TYPE_IN_USE
- OID_802_11_PMKID
- OID_802_11_POWER_MODE
- OID_802_11_REMOVE_KEY
- OID_802_11_REMOVE_WEP
- OID_802_11_RSSI_TRIGGER
- OID_802_11_SSID
- OID_802_11_TEST
- OID_802_11_TX_POWER_LEVEL
- OID_802_3_MULTICAST_LIST
- OID_GEN_CURRENT_PACKET_FILTER
- OID_GEN_CURRENT_LOOKAHEAD
- OID_GEN_PROTOCOL_OPTIONS
- OID_PNP_SET_POWER

Atheros OIDs

Atheros-defined private OIDs are implemented to support features that not supported by Microsoft and NIC control commands that are not supported by Microsoft OIDs.

Query OIDs

- OID_CAR6K_GET_PNP_POWER
- OID_CAR6K_FIRMWARE_VERSION
- OID_CAR6K_GET_WOW_LIST
- OID_CAR6K_DBGLOG_GET_DEBUG_LOGS
- OID_CAR6K_GET_QOS_QUEUE
- OID_CAR6K_RESUME_WLAN_STATE
- OID_CAR6K_802_11_AUTH_ALG
- OID_CAR6K_802_11_CCKM_AUTHENTICATION_MODE

Set OIDs

- OID_CAR6K_RESUME_WLAN_STATE
- OID_CAR6K_TCMD_CONT_TX
- OID_CAR6K_TCMD_CONT_RX
- OID_CAR6K_TCMD_PM
- OID_CAR6K_SET_HOST_SLEEP_MODE
- OID_CAR6K_SET_WOW_MODE
- OID_CAR6K_ADD_WOW_PATTERN
- OID_CAR6K_DEL_WOW_PATTERN
- OID_CAR6K_DBGLOG_CFG_MODULE
- OID_CAR6K_CREATE_QOS
- OID_CAR6K_DELETE_QOS
- OID_CAR6K_SET_WMM
- OID_CAR6K_SET_TXOP
- OID_CAR6K_SET_BT_STATUS
- OID_CAR6K_SET_BT_PARAMS
- OID_CAR6K_IBSS_CHANNEL
- OID_CAR6K_SET_ATIM
- OID_CAR6K_SET_IBSSPM
- OID_CAR6K_SET_PM
- OID_CAR6K_DUMP_HTC
- OID_CAR6K_SET_TXPOWER
- OID_CAR6K_SET_SCAN_PARAMS
- OID_CAR6K_SET_CHANNEL_PARAMS
- OID_CAR6K_SET_WPS_ENABLE
- OID_CAR6K_ADD_KRK
- OID_CAR6K_802_11_AUTH_ALG
- OID_CAR6K_802_11_CCKM_AUTHENTICATION_MODE
- OID_CAR6K_SET_ROAM_CTRL
- OID_CAR6K_ABORT_SCAN

- OID_CAR6K_RSSI_THRESHOLD

SoftAP OIDs

This section lists the Windows Mobile-specific SoftAP supported OIDs.

Query OIDs

- OID_CAR6K_GET_STA
- OID_CAR6K_GET_ACL
- OID_CAR6K_GET_NUM_STA

Set OIDs

- OID_CAR6K_SET_HIDDEN_SSID
- OID_CAR6K_SET_NUM_STA
- OID_CAR6K_SET_ACL_POLICY
- OID_CAR6K_SET_INTRABSS
- OID_CAR6K_SET_BG_PROTECTION
- OID_CAR6K_SET_DTIM
- OID_CAR6K_SET_RD
- OID_CAR6K_ADD_DEL_ACL
- OID_CAR6K_SET_APMODE

WinMobile Registry Entries

AR6000 Registry Entries

The following driver parameters can be configured through the registry. These parameters can be specified under the registry key **HKLM/Comm/ AR6K_SD1/Parms**.

Table I-1. **Registry Entries**

Key Name	Key Type	Default Value if Not Present	Description	
bkScanEnable	DWORD	1	Sets this key to 1 to enable background scanning	
bkScanPeriod	DWORD	60	Sets this key to configure the background scan interval; used only when background scanning is enabled.	
BtCoexAntConfig	DWORD	0	Specifies BT coexistence antenna configuration	
			0	No existence
			1	Dual Antenna
			2	Single Antenna with splitter
			3	Single Antenna with switch
BtDevType	DWORD	0	Specifies the collocated BT device type	
			0	BTS4020
			1	CSR
clkFreq	DWORD	26000000	Specifies the reference clock in Hz	
discTimeout	DWORD	10	Specifies the disconnect timeout interval; indicates the time the driver waits before indicating a disconnect up the network stack	
eepromFile	STRING	NULL	Specifies the file to use for EEPROM content; if not present EEPROM in the board will be used	
enableDbglog	DWORD	1	Enable target side debug log (default is enabled)	
enableUARTprint	DWORD	1	Enables the target console prints when set to 1	
fgScanEnable	DWORD	1	Foreground scanning enabled by default; configure this entry to disable it	
ibssChannel	DWORD	0	Channel to use when starting ad hoc mode	
ibssPSEnable	DWORD	0	Controls Atheros proprietary ad hoc power save mode	
			0	Disable
			1	Enable
intrabssEnable	DWORD	1	Intra BSS support enabled by default; disable this feature using this entry only when the AR6000 is in SoftAP mode	
NetworkAddress	STRING	NULL	Specifies the software MAC address to use; if the MAC address is AA:BB:CC:DD:EE:FF, specify the string as AABBCCDDEEFF. If NetworkAddress is not present, the eepromFile Key is present, and /DGENERATE_MAC_ADDRESS is defined, a MAC address of 00:03:7F:xx:xx:xx generates automatically.	
nodeAge	DWORD	12000	Specifies the node cache aging timeout in milliseconds	
powerSaveMode	DWORD	0	Specifies WLAN power save mode	
			0	Fully ON
			1	Max Sleep
			2	Auto

Table I-1. **Registry Entries (continued)**

Key Name	Key Type	Default Value if Not Present	Description		
rekeyingPeriod	DWORD	0	Rekeying interval can be configured using this entry when it is the DUT in SoftAP mode		
resumeDelay	DWORD	0	Delay to introduce when coming from D3 state. The device comes out of D3 state either during suspend/resume or WiFi ON/OFF operation.		
resetPowerState	DWORD	1	Controls the state of the device after driver initialization		
			0	Place the device into OFF state upon initialization	
			1	Place the device into ON state upon initialization	
			2	Place the device into last WiFi state (ON/OFF) that the device was in upon initialization	
HIF\SDIO\ SlotPowerDevName	REG_SZ	NULL	Host controller driver name that controls WiFi SDIO slot power ON/OFF		
suspendDelay	DWORD	0	Delay to introduce when system goes to D3 state		
suspendMode	DWORD	1	WLAN power management mode during suspend/resume		
			0	CUT_PWR	Power to device is cut in this mode
			1	WARM_RESET	Power to device is present; device/ firmware it put into initial state by doing a WARM_RESET
			2	WLAN_DEEP_ SLEEP	Power to device is present; WLAN firmware is kept in low power by disconnecting from the current AP and disabling scans
tcmd	DWORD	0	Enables TCMD support		
tpsecCompliance	DWORD	1	Enables validating TSpec parameters provided by the user against Atheros-defined values		
			0	Disable	
			1	Enable	
HIF\SDIO\ WifiModuleOffIoctl	DWORD	0	Powers OFF WiFi SDIO slot using this IOCTL code using the device specified in HIF\SDIO\SlotPowerDevName		
HIF\SDIO\ WifiModuleOnIoctl	DWORD	0	Powers ON WiFi SDIO slot using this IOCTL code using the device specified in HIF\SDIO\SlotPowerDevName		
WifiOnOffMode	DWORD	0	WLAN power management mode during WiFi ON/OFF from wireless manager		
			0	CUT_PWR	Power to device is cut in this mode
			1	WARM_RESET	Power to device is present; device/ firmware it put into initial state by doing a WARM_RESET
			2	WLAN_DEEP_ SLEEP	Power to device is present; WLAN firmware is kept in low power by disconnecting from the current AP and disabling scans
wmmConfig	DWORD	1	Sets this key to enable(1)/disable(0) WMM		
wowEnable	DWORD	0	Enables the Wake-on-Wireless (WoW) feature		
			0	Disable	
			1	Enable	

Modifying Registry Entries

When Platform Builder is used, use the Remote Registry Editor tool to modify the registry entry. In a phone device, use the Registry Editor Tool to modify the registry entry.

In a device where AR6000 device is always present, the registry entry corresponding to the AR6000 device is always present. Reset the device to make the modified values effective. In a device, where the AR6000 client device can be inserted and ejected, registry entries are shown in the registry editor only after inserting the AR6000 device. In this case, to modify the registry entry, insert the AR6000 device, modify the value, then eject and insert to make the new values effective. In a device where the AR6000 device is inbuilt and cannot be removed, use the Registry Editor tool to modify the values.

AR6000 Host Driver Sources

These source directories are present in the source package. The NDIS miniport driver module is described in this document.

- **AR6K_DRV\include**
  Contains the definitions and structures common to the host driver and target firmware
- **AR6K_DRV\host\bmi**
  Implements the Bootloader Message Interface layer
- **AR6K_DRV\host\hif**
  Implements the Host Interface Function layer
- **AR6K_DRV\host\htc**
  Host-Target Communication module
- **AR6K_DRV\host\os\wince\ndis**
  Implements the NDIS miniport 802.11 driver
- **AR6K_DRV\host\include**
  Contains the definitions and structures common to all host drivers
- **AR6K_DRV\host\os\wince\include**
  Contains the definitions and structures specific to wince
- **AR6K_DRV\host\os\wince\stream**
  Implements the streams driver
- **AR6K_DRV\host\os\wince\common**
  Contains the configuration, EEPROM and initialization functions common to both streams and NDIS driver
- **AR6K_DRV\host\miscdrv**
  Contains miscellaneous functions common to all host drivers
- **AR6K_DRV\host\wlan**
  Implements WLAN support functions
- **AR6K_DRV\host\wmi**
  Implements Wireless Message Interface functions

- **AR6K_DRV\host\tools\wince**
  Contains the WinMobile/WinCE tools source

Private OID/IOCTL Structures

Atheros-specific private OIDs assist users to get and set WMI/WMIX / TCMD-specific commands to the target and those OID input/output structures are defined in the **WMI.H**/**WMIX.H**/**TCMD.H** files. Host application and driver also use OID-specific structures.

AR6000 Generic IOCTLs

This section describes the Atheros-defined proprietary IOCTLs.

IOCTL_CAR6K_GET_FIRMWARE_VERSION

Parameter	Type	Direction	Description
dwFirmwareVersion	DWORD	OUT	Returns the firmware version

IOCTL_CAR6K_GET_HOST_VERSION

Parameter	Type	Direction	Description	
dwHostVersion	DWORD	OUT	Get the host driver version information	
			bits [31:28]	Major version
			bits [27:24]	Minor version
			bits [23:16]	Patch version
			bits [15:0]	Build number

IOCTL_CAR6K_GET_TARGET_TYPE

Parameter	Type	Direction	Description	
TargetType	DWORD	OUT	Get the target type	
			TARGET_TYPE_AR6001	1
			TARGET_TYPE_AR6002	2

IOCTL_CAR6K_NDIS_REGISTER

Parameter	Type	Direction	Description
—	—	—	Issues *NdisRegisterAdapter* to bring up AR6000, initiating NDIS 5.1 initialization routines

IOCTL_CAR6K_CONFIGURE_AR6000

Parameter	Type	Direction	Description
Ar6000Param	CONFIGURE_ AR6000_PARAM	IN	Aids users in configuring the AR6000 target; can be used by the application to load its own firmware and use the other stream IOCTLs to communicate with its firmware

CONFIGURE_AR6000_PARAM

Parameter	Type	Direction	Description
enableUART	BOOL	IN	Enables UART print
timerWAR	BOOL	IN	Enables timer work-around
clkFreq	UINT32	IN	Clock frequency
filePath	wchar	IN	Input file path
fileRoot	wchar	IN	File root path
bCompressed	BOOL	IN	Checks whether image is compressed
isColdBoot	BOOL	IN	Checks for cold boot configuration
eepromFile	wchar	IN	Checks for EEPROM file if needed

IOCTL_CAR6K_BMI_WRITE_MEMORY

Parameter	Type	Direction	Description
writeParams	BMI_WRITE_ MEMORY_PARAM	IN	BMI write memory parameters; write data to specified memory address in BMI phase

BMI_WRITE_MEMORY_PARAM

Parameter	Type	Direction	Description
Address	UINT32	IN	BMI address
Length	UINT32	IN	Address length
Buffer	UCHAR[]	IN	Input buffer of variable length

IOCTL_CAR6K_BMI_READ_MEMORY

Parameter	Type	Direction	Description
readParams	BMI_READ_ MEMORY_PARAM	IN	BMI read memory parameters Read data from specified memory address in BMI phase

BMI_READ_MEMORY_PARAM

Parameter	Type	Direction	Description
Address	UINT32	IN	BMI address
Length	UINT32	IN	Address length
Buffer	UCHAR[]	OUT	Variable length response buffer

IOCTL_CAR6K_BMI_EXECUTE

Parameter	Type	Direction	Description
executeParams	BMI_EXECUTE _PARAM	IN	BMI execute parameters Execute the firmware code in BMI phase starting at the address specified

BMI_EXECUTE_PARAM

Parameter	Type	Direction	Description
Address	UINT32	IN	Starting address
Param	UINT32	IN	Parameters (if needed)

IOCTL_CAR6K_BMI_SET_APP_START

Parameter	Type	Direction	Description
appStartParam	BMI_SET_APP START_PARAM	IN	BMI application start parameters The host sets the starting address of the target application

BMI_SET_APP_START_PARAM

Parameter	Type	Direction	Description
targetAddress	UINT32	IN	Starting address of the application

IOCTL_CAR6K_BMI_DONE

Parameter	Type	Direction	Description
—	—	—	Indicate the firmware to come out of BMI and execute the application

IOCTL_CAR6K_BMI_READ_SOC_REGISTER

Parameter	Type	Direction	Description
readSocRegParam	BMI_READ_SOC REG_PARAM	IN	Read SOC register parameters Read a 32-bit target SOC register in BMI phase

BMI_READ_SOC_REG_PARAM

Parameter	Type	Direction	Description
targetAddress	UINT32	IN	BMI address
value	UINT32	OUT	Output register value

IOCTL_CAR6K_BMI_WRITE_SOC_REGISTER

Parameter	Type	Direction	Description
targetAddress	UINT32	IN	BMI address
value	UINT32	IN	Input register value

IOCTL_CAR6K_BMI_ROMPATCH_INSTALL

Parameter	Type	Direction	Description
romPatchInstall Param	BMI_ROMPATCH _INSTALL _PARAM	IN	ROM patch install parameters Host indication of ROM patch installation to target side

BMI_ROMPATCH_INSTALL_PARAM

Parameter	Type	Direction	Description
romAddr	UINT32	IN	ROM address
ramAddr	UINT32	IN	Mapped ROM address
nBytes	UINT32	IN	Length
doActivate	UINT32	IN	Activated or not
romPatchId	UINT32	OUT	ROM unique patch ID

IOCTL_CAR6K_BMI_ROMPATCH_UNINSTALL

Parameter	Type	Direction	Description
romPatch UninstallParam	BMI_ROMPATCH _UNINSTALL _PARAM	IN	Host indication of ROM patch uninstallation to target side

BMI_ROMPATCH_UNINSTALL_PARAM

Parameter	Type	Direction	Description
romPatchId	UINT32	IN	BMI_ROMPATCH_UNINSTALL_PARAM

IOCTL_CAR6K_BMI_ROMPATCH_ACTIVATE

Parameter	Type	Direction	Description
romPatch ActivateParam	BMI_ROMPATCH _ACTIVATE _PARAM	IN	ROM patch activate parameters Host indication of ROM patch activation to target side

BMI_ROMPATCH_ACTIVATE_PARAM

Parameter	Type	Direction	Description
romPatchCount	UINT32	IN	Patch count
buffer	UINT32[]	IN	Variable size input buffer

IOCTL_CAR6K_BMI_ROMPATCH_DEACTIVATE

Parameter	Type	Direction	Description
romPatch Deactivate Param	BMI_ROMPATCH_ DEACTIVATE _PARAM	IN	ROM patch activate parameters Host indication of ROM patch deactivation to target side

BMI_ROMPATCH_DEACTIVATE_PARAM

Parameter	Type	Direction	Description
romPatchCount	UINT32	IN	Patch count
buffer	UINT32[]	IN	Input buffer

IOCTL_CAR6K_HTC_RAW_OPEN

Parameter	Type	Direction	Description
—	—	—	The HTC_RAW mechanism provides a way for the application to directly interact with the target, bypassing the WMI layer

IOCTL_CAR6K_HTC_RAW_CLOSE

Parameter	Type	Direction	Description
—	—	—	Closes the HTC RAW interface

IOCTL_CAR6K_HTC_RAW_READ

Parameter	Type	Direction	Description
htcRaw ReadParam	HTC_RAW_READ_ PARAM	IN	HTC raw read parameters Reads a set of bytes from the specified mailbox; prior to using this IOCTL, the HTC_RAW_OPEN IOCTL must be issued

HTC_RAW_READ_PARAM

Parameter	Type	Direction	Description
endPointId	UINT32	IN	Endpoint handle
Length	UINT32	IN	Length of receive buffer
Buffer	CHAR[]	OUT	Response buffer

IOCTL_CAR6K_HTC_RAW_WRITE

Parameter	Type	Direction	Description
htcRawWrite Param	HTC_RAW _WRITE_PARAM	IN	HTC Raw write parameters Writes a set of bytes to the specified mailbox; prior to using this IOCTL, the HTC_RAW_OPEN IOCTL must be issued

IOCTL_CAR6K_HTC_RAW_WRITE

Parameter	Type	Direction	Description
htcRawWrite Param	HTC_RAW _WRITE_PARAM	IN	HTC Raw write parameters Writes a set of bytes to the specified mailbox; prior to using this IOCTL, the HTC_RAW_OPEN IOCTL must be issued

HTC_RAW_WRITE_PARAM

Parameter	Type	Direction	Description
endPointId	UINT32	IN	Endpoint handle
Length	UINT32	IN	Length of receive buffer
Buffer	CHAR[]	IN	Response buffer

IOCTL_CAR6K_GET_DBGLOG

Parameter	Type	Direction	Description
Buffer	UCHAR	OUT	Host issues this command to get debug logs which are available/accumulated by the host driver

IOCTL_CAR6K_MACADDR_UPDATE

Parameter	Type	Direction	Description
Buffer	UCHAR[6]	IN	Update the MAC address in the board EEPROM with this address

IOCTL_CAR6K_RD_UPDATE

Parameter	Type	Direction	Description
rd	UINT32	IN	Update the regulatory domain in the board EEPROM with this value

NDIS Query OIDs

This section describes the Atheros-defined proprietary NDIS OIDs.

OID_CAR6K_RESUME_WLAN_STATE

Parameter	Type	Direction	Description	
Buffer	UINT32	OUT	Get the WLAN states	
			0	OFF
			1	ON

OID_CAR6K_FIRMWARE_VERSION

Parameter	Type	Direction	Description
version	UINT32	IN	Return the firmware version

OID_CAR6K_GET_PNP_POWER

Parameter	Type	Direction	Description	
power	UINT32	IN	Get the WLAN states	
			0	OFF
			1	ON

OID_CAR6K_GET_QOS_QUEUE

Parameter	Type	Direction	Description
qosQueueReq	Struct ar6000_queuereq	OUT	Returns the queue number to the traffic class

AR6000_QUEUEREQ

Parameter	Type	Direction	Description
trafficClass	UINT8	IN	Traffic class ID
activeTsids	UINT16	OUT	TSpec ID

OID_CAR6K_802_11_AUTH_ALG_1 / OID_CAR6K_802_11_AUTH_ALG

Parameter	Type	Direction	Description
authAlg	CAR6K_802_11 _AUTH_ALG	OUT	Return current authentication algorithm

CAR6K_802_11_AUTH_ALG		
Parameter	**Type**	**Description**
car6k802_11AuthAlgOpen	enum:1	Open authentication algorithm
car6k802_11AuthAlgShared	enum:2	Shared authentication algorithm
car6k802_11AuthAlgReserved	enum:3	—
car6k802_11AuthAlgLeap	enum:4	Leap authentication algorithm

OID_CAR6K_802_11_CCKM_AUTHENTICATION_MODE			
Parameter	**Type**	**Direction**	**Description**
cckmAuth Mode	CAR6K_802_11 _CCKM_ AUTHENTICATION _MODE	OUT	Return the current CCKM authentication mode

CAR6K_802_11_CCKM_AUTHENTICATION_MODE ENUM		
Parameter	**Type**	**Description**
car6k802_11AuthModeDisableCCKM	enum:1	CCKM disabled
car6k802_11AuthModeWPACCKM	enum:2	CCKM + WPAv1 enabled
car6k802_11AuthModeWPA2CCKM	enum:3	CCKM + WPAv2 enabled

NDIS Set OIDs

This section describes the NDIS Set OIDs.

OID_CAR6K_RESUME_WLAN_STATE				
Parameter	**Type**	**Direction**	**Description**	
powerState	ULONG	IN	Set WLAN power state	
			0	OFF
			1	ON

OID_CAR6K_IBSS_CHANNEL			
Parameter	**Type**	**Direction**	**Description**
wchannel	USHORT	IN	Set IBSS channel

OID_CAR6K_SET_ATIM

Parameter	Type	Direction	Description
wAtim	USHORT	IN	Set ATIM windows

OID_CAR6K_SET_WPS_ENABLE

Parameter	Type	Direction	Description
enableWPS	UINT32	IN	Enable wireless simple configuration mode

OID_CAR6K_ADD_KRK

Parameter	Type	Direction	Description
keyMaterial	NDIS_802_11 _KEY	IN	16-byte key added to target in CCX mode

OID_CAR6K_802_11_AUTH_ALG

Parameter	Type	Direction	Description
authAlg	CAR6K_802_11 _AUTH_ALG	IN	Set the authentication algorithm

OID_CAR6K_802_11_CCKM_AUTHENTICATION_MODE

Parameter	Type	Direction	Description
cckmAuth Mode	CAR6K_802_11 _CCKM _AUTHENTICATION _MODE	IN	Set CCKM authentication mode

OID_CAR6K_ABORT_SCAN

Parameter	Type	Direction	Description
None	None	N/A	Abort the current host initiated scan

OID_CAR6K_RSSI_THRESHOLD			
Parameter	**Type**	**Direction**	**Description**
rssiParams	USER_RSSI_PARAMS	IN	Set RSSI above/below thresholds; when a certain threshold is met, corresponding user defined tags are sent to the upper layer application.

USER_RSSI_THOLD		
Parameter	**Type**	**Description**
tag	USHORT	Tag to send back to application
rssi	USHORT	RSSI in dBm

USER_RSSI_PARAMS		
Parameter	**Type**	**Description**
weight	UCHAR	Range in [1, 16], used in the formula to calculate average RSSI
pollTime	ULONG	RSSI sampling frequency in seconds; if polltime=0, RSSI sampling is disabled
tholds	USER_RSSI _THOLD[12]	User-defined RSSI thresholds (in dbM). The first six elements are ABOVE thresholds and can be set in any order, the last six elements are BELOW thresholds, and can also be set in any order.

WMI/TCMD OID Structures

Table I-2 lists the OID commands used for executing WMI/TCMD instructions. The OIDs input/output parameters are WMI/TCMD data structures.

Table I-2. **WMI/TCMD OID Structures**

OID	WMI Tag	Reference
OID_CAR6K_TCMD_CONT_TX	TCMD_CONT_TX	\include\Testcmd.h
OID_CAR6K_TCMD_CONT_RX	TCMD_CONT_RX	\include\Testcmd.h
OID_CAR6K_TCMD_PM	TCMD_PM	\include\Testcmd.h
OID_CAR6K_SET_HOST_SLEEP_MODE	WMI_SET_HOST_SLEEP_MODE_CMD	\include\Wmi.h
OID_CAR6K_SET_WOW_MODE	WMI_SET_WOW_MODE_CMD	\include\Wmi.h
OID_CAR6K_ADD_WOW_PATTERN	WMI_ADD_WOW_PATTERN_CMD	\include\Wmi.h
OID_CAR6K_DEL_WOW_PATTERN	WMI_DEL_WOW_PATTERN_CMD	\include\Wmi.h
OID_CAR6K_GET_WOW_LIST	WMI_GET_WOW_LIST_CMD	\include\Wmi.h
OID_CAR6K_DBGLOG_CFG_MODULE	WMIX_DBGLOG_CFG_MODULE_CMD	\include\Wmix.h
OID_CAR6K_CREATE_QOS	WMI_CREATE_PSTREAM_CMD	\include\Wmi.h
OID_CAR6K_DELETE_QOS	WMI_DELETE_PSTREAM_CMD	\include\Wmi.h
OID_CAR6K_SET_WMM	WMI_SET_WMM_CMD	\include\Wmi.h
OID_CAR6K_SET_TXOP	WMI_SET_WMM_TXOP_CMD	\include\Wmi.h
OID_CAR6K_SET_BT_STATUS	WMI_SET_BT_STATUS_CMD	\include\Wmi.h
OID_CAR6K_SET_BT_PARAMS	WMI_SET_BT_PARAMS_CMD	\include\Wmi.h
OID_CAR6K_GET_WOW_LIST	WMI_GET_WOW_LIST_REPLY	\include\Wmi.h
OID_CAR6K_SET_IBSSPM	WMI_IBSS_PM_CAPS_CMD	\include\Wmi.h
OID_CAR6K_SET_PM	WMI_POWER_PARAMS_CMD	\include\Wmi.h
OID_CAR6K_SET_TXPOWER	WMI_SET_TX_PWR_CMD	\include\Wmi.h
OID_CAR6K_SET_SCAN_PARAMS	WMI_SCAN_PARAMS_CMD	\include\Wmi.h
OID_CAR6K_SET_CHANNEL_PARAMS	WMI_CHANNEL_PARAMS_CMD	\include\Wmi.h
OID_CAR6K_SET_ROAM_CTRL	WMI_SET_ROAM_CTRL	\include\Wmi.h

WinCE NDIS Startup Architecture

The NDIS driver is dynamically loaded by the parent bus driver (such as the SDIO bus driver) upon device/card detection. The NDIS miniport is registered after any final bus interface initializations. The miniport initialization handles firmware download and configuration (patch, datasets and EEPROM transfer).

J

Bluetooth Coexistence

Overview

The AR6003 coexists with a Bluetooth implementation with which it shares common ISM frequency band. AR6003 supports three-wire slotted mode coexistence protocol that has been the industry standard. WLAN and BT must coexist if the devices are placed close by (isolation less than 20 dB). The AR6003 supports these front end antenna configurations:

- Dual antenna
- Single antenna with splitter
- Single antenna with switch

Coexistence Protocol

In three-wire slotted protocol, packet arbitration is driven from the AR6003 coexistence engine. The Bluetooth device requests transmission/reception over the BT_ACTIVE line. The priority of the BT transmission/reception is indicated with the BT_PRIORITY device. The AR6003 coexistence logic, based on multiple inputs (WLAN state and Bluetooth priority) stomps Bluetooth transmission over WL_ACTIVE or allows the BT transmission to go through (WL_ACTIVE stays low). Contact Atheros for details on timing sequences of the coexistence interface.

Table J-1. **Signaling interface between BT device and the AR6003**

Signal	Direction	PIN on AR6003 (JP 37 Labeled as BT header)	Description
BT_ACTIVE	From BT	GPIO_2 (pin #5)	Asserted for BT TX/RX

Table J-1. **Signaling interface between BT device and the AR6003**

BT_PRIORITY	From BT	GPIO_0 (pin #1)	Priority of BT transmission and direction (Tx or Rx)
WL_ACTIVE	To BT from AR6003	GPIO_1(Pin #9)	Decision of coexistence logic for BT frame. A logic high indicates BT to stop transmission. A logic low allows BT to continue with Tx/Rx.

RF Front End

Single Antenna with RF Switch

In principle with coexistence protocol, either BT or WLAN has the medium. A single front end RF switch should suffice for both the devices. The AR6003 coexistence engine arbitrates antenna owner based on the coexistence signal.

Table J-2. **Antenna Control Signals**[1]

Antenna	Pin	Description
ANT_A	Pin 11	Logic high indicates WLAN Tx has the medium
ANT_C	Pin 15	Logic high indicates WLAN Rx has the medium
ANT_D	Pin 17	Logic high indicates Bluetooth has the medium

[1] Only one signal is asserted at any point of the time.

Table J-3. **Switch Logic**

WLAN State	ANTD	ANTA	ANTC
Off	1	0	0
On (Idle)	1	0	0
Coex Disable	1	0	0
Coex Disabled (WLAN Tx)	1	0	0
Coex Disabled (WLAN Tx)	1	0	0
Coex Enabled (WLAN Tx)	0	1	0
Coex Enabled (WLAN Rx)	0	0	1
Network Sleep	1	0	0
Deep Sleep	1	0	0
Reset	1	0	0

This software release supports only a single antenna with RF switch.

Software Configuration

AR6003 firmware drives the AR6003 coexistence engine. This software release supports three coexistence modes based on Bluetooth traffic type.

SCO Coexistence

Firmware relies on the host to indicate the start/end of SCO traffic (using WMI_SET_BTSTATUS_CMD). Based on WLAN traffic, the AR6003 can be in coexistence power-save mode or coexistence active mode.

SCO Coexistence Power-Save Mode

In this mode, the AR6003 comes out of power save mode and sends trigger frames on the next falling edge of SCO frame, thus ensuring the presence of the AP as well as retrieving any downlink packet queued up for the STA. During coexistence, software does not rely on the beacon for TIM information or to dictate the presence of the AP. In the absence of any data activity, software returns to power-save mode. In power save, software allows all Bluetooth traffic to proceed without stomping.

SCO Coexistence Active Mode

On identifying data activity, the AR6003 comes out of power-save mode and queues trigger frames at the falling edge of SCO (a trigger frame is sent only if no uplink frame is queued or no pending downlink data exists). In this mode, firmware stomps any low priority Bluetooth. AR6003 firmware performs some of the adaptation to achieve good throughput and adapts under bad RSSI conditions to reduce the stomping of Bluetooth. In addition, AR6003 firmware maintains statistics of channel conditions and past packets. Based on these statistics, it turns on/off certain internal optimizations to maximize the throughput performance without compromising on BT audio quality.

A2DP Coexistence

Firmware relies on the host to indicate the start/end of SCO traffic (using WMI_SET_BTSTATUS_CMD). Similar to SCO, based on WLAN traffic, the AR6003 can be in coexistence power-save mode or coexistence active mode.

A2DP Coexistence Power-Save

This case works similar to "SCO Coexistence Power-Save Mode".

A2DP Coexistence Active Mode

In this mode, software uses the inherent bursty nature of A2DP traffic. At the end of A2DP burst, software begins queueing trigger frames until the next burst activity is identified. Any Bluetooth traffic is stomped while the WLAN is waiting for downlink traffic. This algorithm scales according to changing A2DP burst duration. In addition, AR6003 firmware maintains statistics of channel conditions and past packets. Based on these statistics, it turns on/off certain internal optimizations to maximize the throughput performance without compromising on BT audio quality.

ACL Coexistence

Based on activity seen on BT-active lines, software determines if any ACL-based traffic (non-A2DP traffic) is running. On ACL frames the firmware triggers ACL coexistence. In this mode, software equally shares the medium between Bluetooth and the WLAN. During WLAN time, if no downlink traffic exists, software returns the medium to Bluetooth. Note that software only monitors for Bluetooth traffic when it is fully awake.

WMI Commands

AR6003 coexistence software uses these WMI commands:

- SET_BT_STATUS (see "SET_BT_STATUS" on page A-29)
  This API indicates the status of collocated Bluetooth device.
- SET_BT_PARAMS (see "SET_BT_PARAMS" on page A-27)
  This API is used to pass configuration details for different BT_STREAMs as well as to indicate front end RF configuration.

K

WLAN Power Save Options

Overview

This section provides a high-level description of four methods for power savings in Wireless Local Area Network (WLAN) clients. Three are industry-standard methods, while the fourth is an Atheros-patented scheme. The performance of these four techniques is compared when the client is engaged in a Voice-over-Internet Protocol (VoIP) conversation. The performance is measured in terms of average power consumption by the client chip, as well as the worst-case delay experienced on the wireless link by the VoIP packets. All power consumption numbers are derived based on the measured power consumption of the Atheros AR6000.

The three power saving methods compared are Legacy Power Save (PS) Polling, Powers Save with Power Management (PM) Bit, and Unscheduled Automatic Power Save Delivery (U-APSD). PS-Polling is the oldest technique. Legacy Power-Save with PM Bit was developed later to improve throughput when transferring large files. U-APSD is part of the more recent 802.11e standard, developed especially to improve the performance when serving streaming traffic with strict delay constraints, such as VoIP conversations. It is shown in this document that U-APSD, indeed, gives the best performance when serving VoIP traffic.

However, the 11e extension of the 802.11 standard is relatively recent; therefore, the majority of Access Points (APs) currently deployed do not yet support U-APSD. To address this issue, Atheros Communications has developed a client-side solution to approximate the behavior and performance of U-APSD even when the client is associated with legacy APs. This solution is called Simulated U-APSD. A description of simulated U-APSD is provided at the end of this document along with a comparison of its performance to the three industry-standard schemes.

Power Consumption of Atheros AR6000

Please refer to the individual data sheets for the typical power drain on each of the on-chip power supply domains as a function of the AR6000's operating mode.

Power Saving Techniques

Legacy Power Save (PS)-Poll Scheme

In the PS-Poll scheme, the access point (AP) does not send any data packets to the client until the client requests them. In this way, the client may conserve energy by going into sleep mode without the danger that it will miss packets sent to it by the AP while the client is in sleep mode. Instead, the AP buffers any data packets intended for the client until the client notifies the AP that it is awake and ready to receive packets. This notification is done using a PS-Poll packet.

Because the AP sends out periodic beacons carrying information about the network (typically every 100 ms), the client can ensure that it wakes up in time to receive every beacon. In the Traffic Indication Message (TIM) field, the AP notifies clients which ones have data intended for them. A client that has data intended for it then sends a PS-Poll packet to the AP, letting the AP know that the client is awake and ready to receive data.

Upon receiving a PS-Poll, the AP sends the first of the data packets for that client. Any more data packets that exist for that client are indicated in the more bit of the previous data packet. The client keeps sending PS-Polls to the AP after every downlink data packet that has the More bit set. Once the client receives a downlink data packet with the More bit not set, the client knows that there is no more data destined for it, so it sleeps until it is time to receive the next beacon. The typical operation of the PS-Poll scheme is shown in Figure K-1.

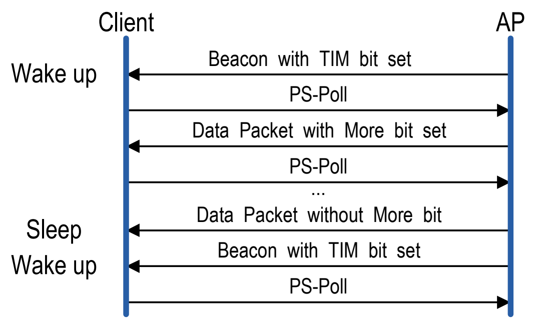

Figure K-1. **PS-Polling Scheme**

Legacy Power Save with Power Management (PM) Bit

The PS-Poll scheme works well when the amount of traffic from the AP to the client is small, because it allows the client to go to sleep quickly after the beacon and any downlink data packets that the AP may have destined for it. However, if the wireless link is used for downloading large files, the PS-Poll scheme becomes very inefficient because it requires the client to send a PS-Poll to pull down every data packet. To address this problem, an alternative power management method was developed, which is now part of WiFi Alliance testing.

The idea behind this power management scheme is for a client to announce that it is awake and have the AP push down all the data it has that is destined for the client. The diagram in Figure 2 shows the typical operation of this scheme. As in the PS-Poll scheme, the client wakes up in time to receive the beacon from the AP. If the TIM field indicates that there is data destined for the client, the client sends a "null packet" with the Power Management (PM) bit off, indicating that it is awake and ready to receive data.

Upon receiving this null packet, the AP pushes down any data it has that is destined for the client, including data that may have arrived since the last beacon. The client tracks the activity on the wireless link, and if the link is idle (no data arrives from the AP) for some period of time (usually set to 200ms), the client decides to go back to sleep and sends another null packet with the PM bit on, indicating that it is going back to sleep. From that point on, the AP buffers any new packets that arrive for the client. These packets are stored in the AP until the client announces that it is awake and ready to receive data again (after the client receives a beacon with the TIM bit set). See Figure K-2.

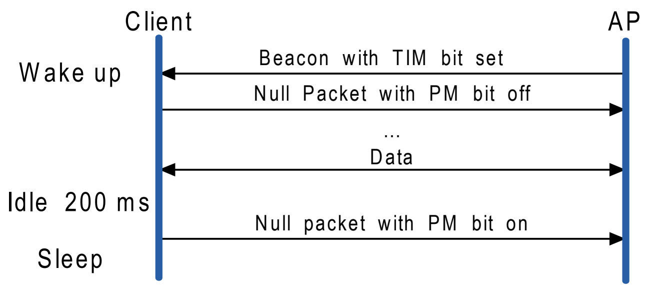

Figure K-2. **Legacy Power Save with PM Bit**

Unscheduled Automatic Power Save Delivery (U-APSD)

Consider the case of periodic data traffic consisting of short messages, such as a Voice over Internet Protocol (VoIP) call. The typical data pattern of a VoIP call consists of short messages (typically 200 Bytes) sent every 20 ms. In this case, the PM Bit scheme would not provide any power savings at all because the client's timer of 200 ms would never time-out, thus never allowing the client to go to sleep.

To address use cases such as VoIP calling over Wireless Local Area Networks (WLAN), the 802.11e standard provides Quality of Service (QoS) enhancements such as prioritized queuing, as well as power saving enhancements. The power saving method specified in 802.11e is called Unscheduled Automatic Power Save Delivery (U-APSD).

U-APSD allows for the client to schedule in advance the communication pattern between the client and AP based on the predictable traffic pattern of VoIP calls. For example, the client can negotiate with the AP to exchange VoIP packets every 20 ms. Once the negotiation is complete the client may sleep between the packet exchanges without notifying the AP.

Power Saving Techniques and VoIP Traffic

VoIP Using U-APSD

This analysis assumes that power consumption in various states of the WLAN chip as well as wake up time of the WLAN chip is based on the measured performance of the Atheros AR6000 device. See the AR6003 data sheets for more details. Because voice traffic follows a deterministic period, the WLAN module alternates between voice and sleep activity during a VoIP call. The duration of the voice period depends on the voice CODEC packetization rate (typically 20 ms). Accordingly, the WLAN module wakes up every 20 ms to send and receive a voice CODEC packet. Figure K-3 shows the communication and sleep pattern of a client engaged in a VoIP call using U-APSD.

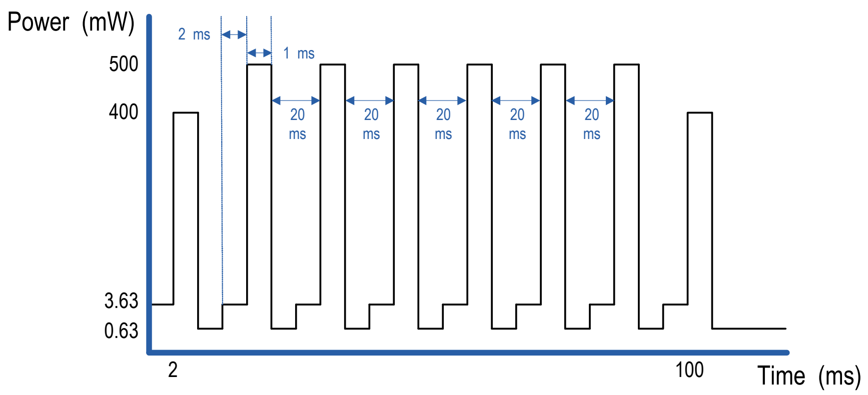

Figure K-3. **Communication and Sleep Pattern of a Client Engaged in a VoIP Call Using U-APSD**

For APs supporting U-APSD operation for QoS-aware and power-efficient support of voice traffic, an average awake period of 1 ms to exchange this data may be assumed. The WLAN module needs ~2 ms to warm-up on exiting the sleep mode - so the module needs to wake up at least 2 ms prior to the 20 ms period.

The WLAN module also needs to wake up every beacon period to listen for the network beacons. Assuming equal Tx and Rx duty cycles during voice data exchange, we use an average WLAN power consumption of 500 mW. During the *sleep* period, the majority of the system is turned off and a very low amount of power (0.4 mW) is consumed.

During the beacon listen period, the WLAN module will stay in the WLAN Rx mode and the estimated power consumption is 400 mW. During warm up period, estimated power consumption is 3.63 mW. Accordingly, based on typical *voice/sleep* duty cycles the average power consumption will be:

30 mW (500 * 5/100 + 400 * 1/100 + 3.63 * 12/100 + 0.4 * 82/100).

VoIP Using PS-Polling

In the legacy PS-Poll scheme, the data is buffered at the AP until the next beacon period. In the next beacon, the AP notifies the client that there is data destined for it by setting a bit in the TIM filed of the beacon. Upon hearing the beacon with the TIM bit set, the client sends a PS-Poll to request the data from the AP.

One drawback of this scheme is that the client must send a PS-Poll for each of the five downlink packets that accumulated for it during a 100 ms beacon period. The energy spent on transmitting the PS-Polls and waiting for the AP to process and respond to the PS-Polls by sending the downlink voice packets is slightly higher than the total energy consumed for both uplink and downlink communication in U-APSD:

The same amount of data is sent on the downlink, while the transmission of the PS-Polls is roughly equivalent to the transmission of the uplink packets. However, the time it takes the AP to respond to the PS-Poll is longer than the time between the transmission of uplink and downlink packets in U-APSD.

Therefore, acquiring the downlink packets with PS-Polling is slightly higher than the total energy required for transmitting both uplink and downlink packets with U-APSD. In addition, with PS-Polling, the uplink packets must be transmitted as they arrive from the application. This will require about half of the total amount of energy that is spent for uplink and downlink communication in U-APSD. The average power consumption of the WLAN chip with power properties of the AR6000 would be approximately 45 mW depending on the specifics of the VoIP call and WLAN network conditions.

The bigger drawback of using the legacy PS-Poll scheme for VoIP calls is the delay incurred by the downlink voice packets. Because the client only sends PS-Polls after it has been notified that there is data waiting for it via the TIM field in the beacon, the worst-case delay between the time a downlink packet arrives at the AP and the time it is actually sent to the client is on the order of 100 ms.

VoIP Using Legacy Power Save (PS) with PM Bit

A VoIP call will prevent a client using Power Save (PS) with PM Bit from ever going to sleep because the client's timer for going to sleep will never expire. Therefore, the power consumption of the client in this mode will be more than 400 mW (it will always either be listening at 400 mW or transmitting at 600 mW). Though, in this scheme the voice packets experience very low latency (<1 ms) because they never have to wait for the client to wake up.

Performance Comparison of PS Schemes for VoIP

Please refer to the individual data sheets for the delay and power consumption of Atheros industry-standard PS schemes for VoIP traffic. U-APSD is clearly the best option because it provides the lowest average power consumption while ensuring a reasonable delay to the VoIP packets over the wireless link. When the client is associated with an AP that supports U-APSD, it is advisable for the client to negotiate a U-APSD connection.

However, because most APs currently deployed do not support 802.11e, there will be many cases when the client cannot negotiate a U-APSD connection. Atheros Communications has developed a client-side solution to approximate the behavior and performance of U-APSD even when the client is associated with legacy APs. This distinct advantage only available in the Atheros WLAN solution ensures high quality VoIP experience for new and legacy APs whether or not they support U-APSD.

Atheros Simulated U-APSD

To provide low power consumption while maintaining acceptable worst case delay for a VoIP call even when the client is associated to a legacy AP (that does not support U-APSD), Atheros has developed a scheme called Simulated U-APSD. This scheme approximates the behavior of U-APSD by using a previously overlooked feature of PS-Polling. Namely, in the PS-Polling scheme, the client does not need to wait for any notification from the AP before sending a PS-Poll (typically, the client only sends PS-Polls upon receiving a beacon with the TIM bit set). See Figure K-4.

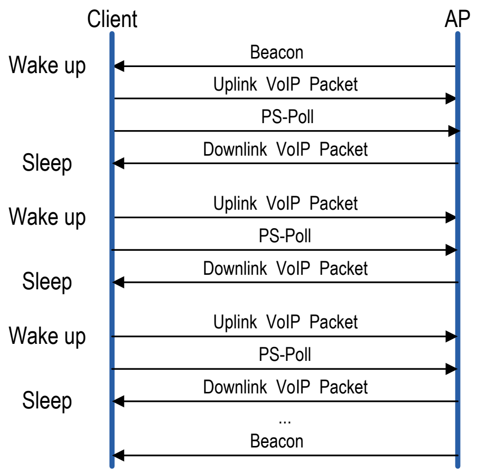

Figure K-4. **VoIP Communication Using Atheros-Patented Simulated U-APSD**

In simulated U-APSD, a client engaged in a VoIP call makes use of the fact that the application layer knows that there will be packets on both the uplink and downlink every 20 ms. The communication pattern, shown in Figure K-4, is as follows: the client is in sleep mode most of the time. Whenever an uplink VoIP packet is generated (every 20 ms), the client wakes up and sends the uplink packet to the AP.

Before going to sleep, the client also sends a PS-Poll to the AP because it knows that the traffic of a VoIP call is symmetric (i.e. downlink packets are generated every 20 ms, just like uplink packets). The AP responds to the PS-Poll by sending the downlink VoIP packet, and the client goes back to sleep until the next uplink packet is generated. During a 100 ms beacon period, the client will wake up five times (every 20 ms) to exchange VoIP packets.

Therefore, the communication and sleep pattern of a client engaged in a VoIP call using Simulated U-APSD would look very much like the pattern for a VoIP call using U-APSD shown in Figure K-3. The difference is that the awake periods last longer (up to 1 ms longer, depending on network conditions) in Simulated U-APSD than in U-APSD, because when using PS-Polling to pull down packets from the AP, a delay may occur in how quickly the AP responds to the PS-Poll due to any other downlink packets that are already in the queue waiting to be transmitted to other clients.

U-APSD does not experience this additional delay because it includes QoS support that ensures that VoIP packets process ahead of other traffic.

Simulated U-APSD approximates the behavior of U-APSD when the client is associated with a legacy AP that does not support 802.11e. The performance of Simulated U-APSD is close to that of U-APSD even though it does not require anything more than legacy features from the AP. Therefore, Simulated U-APSD should be used whenever the client is engaged in a VoIP call over a legacy network because it gives much better performance than the two standard legacy power save protocols.

L
Windows Mobile SoftAP

Architecture

The AR6003 SoftAP allows an associated STA to access a back-bone network gateway, so that the associated client can achieve 3G/GPRS access as soon as it associates with a AR6003 based wi-fi enabled device. Therefore, an AR6003-based wi-fi enabled device can be configured to function as a SoftAP. That is, 3G smart phones with AR6003-based wi-fi can function as a 3G gateway by connecting the STAs in the AR6003-based AP. Figure L-L-1 illustrates the network diagram of 3G gateway usage.

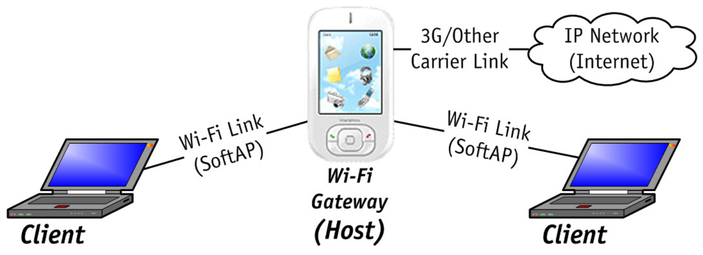

Figure L-1. **3G Gateway Usage Network Diagram**

As Figure L-L-1 shows, the 3G gateway allows clients to connect to the internet via the host. It also establishes a GPRS/public connection and enables the IP router. The architecture controllers include:

- Wi-fi controller
  Checks the wi-fi device state and creates SoftAP Mode (ATHR-TOOL)
- IP controller
  Configures the wi-fi device IP and enables the IP router; this controller reads the IP address and DHCP state information from registries then assigns an IP address based on DHCP state
- NAT controller
  Enables the DHCP allocator (which is provided by Microsoft)
- 3G /VMINI controller
  Establishes a 3G/GPRS/VMINI connection and calls the connection manager functions to establish or release a 3G/GPRS connection; note that a valid 3G /GPRS/VMINI connection profile must be available

Features

These features are supported on Linux and Windows Mobile 6.5:

- Function as an Access Point or a Station
- Dynamic switching between AP and STA mode
- Support for six wi-fi clients in TKIP mode and eight wi-fi clients in other modes
- Open and shared authentication methods
- WEP, WPA-PSK, and WPA2-PSK security methods
- Configuring AP parameters
- Legacy power save method
- Distribution system support for forwarding data to another interface
- Support for forwarding data to and from 3G/GPRS network (support from the customer is required for testing with 3G networks)
- Blocking/allowing of STAs from connecting to the AP
- 802.11b/g

These limitations exist:

- Only eight STAs are supported
- WMM and uAPSD are not supported
- BT Coexistence is not supported in the AP mode

M
Debug Logging

Introduction

Debug logging enables users to expedite the debugging process through enhancements to the current debugging infrastructure to:

- Reduce the time spent in identifying the problems reported
- Reduce the code space occupied by a single debug statement, enabling the inclusion of more debugging information without increasing the overall size of the image
- Compress the information communicated by the debug statement during execution and decompressing the same offline, reducing the memory requirement at runtime
- Be useful in the event of serial port support being absent on the target

Theory of Operation

The basic principle is that debug information generated by target code is buffered in a pre-allocated set of debug buffers. Once a buffer reaches a threshold where it is unable to store any additional debug information, it hands over the buffer to the application using the debug service, which communicates debug information to the host to store it for later analysis.

In the meantime, the debug module switches over to the next buffer in the list of buffers provided by the application during initialization. Once the application is done transmitting the buffer contents, it marks it as free to be used by the debug module later. If the debug module runs out of buffer space and no free buffers are available for it to store debug information, it follows a head drop policy and overwrites the contents at the beginning of the buffer.

The module keeps track of the number of times it wrapped around and communicates the number to the host during the next debug log event. The debug log event in the current implementation assumes the form of a WMI extended event.

For fatal errors (e.g., asserts), on the other hand, because firmware might not be in a state to frame an event, minimal work is performed before the chip goes down, generating an error interrupt to the host. The host uses this as a trigger to pull the information from the target's memory via the diagnostic window.

Once the logs are available to the host side, they are communicated to the user space via the wireless events mechanism. There is a restriction on the size of the message associated with the event. For this reason, once the debug log message is received on the host, it is divided into chunks of manageable size and sent to the application. The 'recEvent' daemon waiting for these events then feeds this information into a parser that scans the relevant headers file to interpret the debug identifiers and captures the output in an easily readable format into a log file.

Debug Primitives

Table M-1 lists the available debug primitives.

Table M-1. **Debug Primitives**

Name	Description	
DBGLOG_INIT	Primitive	DBGLOG_DEBUG_LOG_EVENT_DB callback, void *cookie
	Description	Used by the application (e.g., athwlan) during initialization to register the debug event callback, a cookie associated with the callback
DBGLOG_DEBUG_ LOG_EVENT_CB	Primitive	Registered_callback (void *cookie, A_UINT8 *buffer, A_UINT32 length, A_UINT32 dropped)
	Description	The callback is used by the debug module to communicate debug logs to the application once the current buffer is full or the message threshold is exceeded. The cookie passed is the one registered using the dbglog_init function. The buffer points to the start of the buffer containing the debug log messages. The length parameter indicates the size of the buffer containing valid data. The 'dropped' indicates the number of times the debug module had to reuse this buffer because of the unavailability of a free buffer.
DBGLOG_QUEUE_ BUFFER	Primitive	dbglog_queue_buffer (A_UINT8 *buffer, A_UINT32 size)
	Description	Takes a pointer to the buffer and the size of the buffer as an argument. The API is used by the application to queue a set of buffers to the debug module during initialization. Also, used to requeue the used buffer handed over by the debug module once the application is done transmitting its contents to the host.
DBGLOG_ARG0_ RECORD	Primitive	dbglog_arg0_record (A_UINT32 id)
	Description	Used to log a debug statement with no associated data. Takes a unique identifier as one of the parameters to identify the debug statement.
DBGLOG_ARG1_ RECORD	Primitive	dbglog_arg1_record (A_UINT32 id, A_INT32 arg1)
	Description	Used to log a debug statement with data. Takes a unique identifier as one of the parameters to identify the debug statement and a single 32 bit data argument.
DBGLOG_ARG2_ RECORD	Primitive	dbglog_arg2_record (A_UINT32 id, A_INT32 arg1, A_INT32 arg2)
	Description	Logs a debug statement with data. Takes a unique identifier as one of the parameters to identify the debug statement and a two 32 bit data arguments.

Table M-1. **Debug Primitives (continued)**

Name	Description	
DBGLOG_SEND_ DEBUG_LOGS	Primitive	dbglog_send_debug_logs ()
	Description	Can be used by the application anytime to force the debug module to return the buffer containing the latest set of debug information.
DBGLOG_CONFIG_ DEBUG_MODULE	Primitive	dbglog_config_debug_module (struct dbglog_config_s *config)
	Description	Takes a pointer to the dbglog config data structure as a parameter. Can be used by the application to configure the behavior of debug module. In particular, it can be used to: ■ Selectively enable logging for specific modules by setting the corresponding bit in the module mask. The bit position of a particular module in the mask is indicated by its module id defined in the header file **dbglog_id.h**. The default is to enable logging for all defined modules. ■ Determine whether the debug module should report logs through the event mechanism. If the reporting is disabled, the debug module does not generate a debug event once its current buffer is full, but simply continues with the next buffer in the list. To get debug information, the application can either use the DBGLOG_SEND_DEBUG_LOGS primitive or have the host pull data through the diagnostic window. Reporting is enabled by default. ■ Configure the resolution of time stamps used to tag the debug statement. The resolution can be changed in powers of 2 with 32.5 μs as the minimum. The default resolution is 1 ms. Refer to the wmiconfig documentation for the corresponding command.

Debug identifiers are organized on a module basis and are unique within a particular module. To add support for a new module:

1. Add the hash-defined value for the new module in the **/include/dbglog.h** file between these defines: DBGLOG_MODULEID_START and DBGLOG_MODULEID_END.
2. Add the debug identifiers for the new module in the file **/include/ dbglog_id.h** protected by the defines: <NEW_MODULE_NAME>_DBGID_DEFINITION_START and <NEW_MODULE_NAME>_DBGID_DEFINITION_END.
3. Define the debug primitives for the new module in the file **/target/include/dbglog_api.h** for the module specific header file. Refer to the WMI/CSERV section in the relevant file for help.
4. Include the **dbglog_api** file in the module in which to use the primitives.

Each debug statement results in a log trace in this format:

1. Debug statement header (A_UINT32, mandatory): The lower 16 bits (bits [15:0]) carry the timestamp information. Bits [25:16] store the debug identifier unique only within a particular module. Bits [29:26] indicate the module ID. The upper two bits (bits [31:30]) reflect the number of arguments in the debug statement.
2. Parameter 1 (A_UINT32, optional based on the debug statement header).
3. Parameter 2 (A_UINT32, optional based on the debug statement header).

For specific details on low-level debug commands, see the commands section pertaining to the OS of interest.

N
Address Resolution Protocol Offloading

Address resolution protocol (ARP) is used to resolve an IP address into a physical address (i.e., MAC), such as an Ethernet address. Previously resolved addresses are maintained in an ARP cache. IP to MAC mapping function checks the cache before performing the ARP protocol exchanges.

The proxy ARP technique allows a network entity to answer to ARP queries on behalf of another network host. Proxy ARP is transparent to hosts and learns and unlearns the address bindings by inspecting the IP/ARP packets. The AR6000 implements ARP offloading to reduce host power consumption by avoiding unnecessary host wakeups during sleep mode. In this scheme, the NIC can respond to ARP queries when the host enters power save state. This feature is disabled when the host is awake, and the host-IP stack ARP takes over for normal functionality.

The advantages for ARP offloading include:
- Enhanced host power save; the host does not need to wake up for ARP
- No overhead dynamic IP, as IP binding does not change during host sleep
- The STA-only feature may be supported with any AP

Its disadvantages include:
- No benefits for power save-disabled host OS or platform
- No gains in awake mode operation

ARP offloading works this way:
- When the host informs the AR6000 that the host is going to sleep, ARP offloading is enabled
- When the host informs the AR6000 that the host is going to wake, ARP offloading is disabled
- When ARP offloading is enabled, the AR6000 processes ARP requests and responds with appropriate ARP responses; if no ARP response is needed, the ARP request packet is dropped and not sent to the sleeping host

0 Licensing Issues

The software source and binaries included in this development package are licensed, not sold. The AR6000 SDK is provided under one or more license agreements. The rights granted are specifically listed in these license agreement(s). All other rights remain with Atheros or the respective owner. Distribution of any portion of this package must be in strict compliance with the license agreement(s) terms.

Atheros Communications, Incorporated
5480 Great America Parkway
Santa Clara, CA 95054
tel: 408.773.5200
fax: 408.773.9940
www.atheros.com