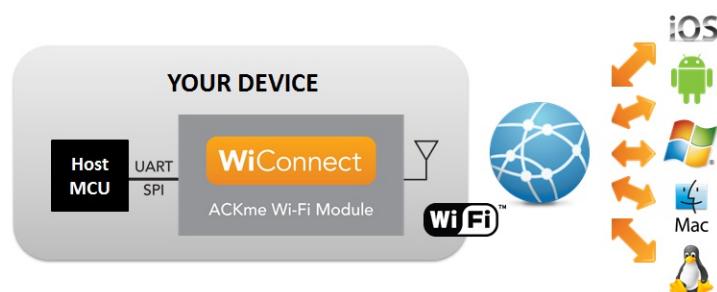




WiConnect Reference Guide

WiConnect v2.1



The official WiConnect Reference Guide is available online at:

<http://wiconnect.ack.me>

This document is provided for convenience ONLY and is likely to be out of date!

Please refer to the online version for the latest information about WiConnect.

Disclaimer

While the information provided in this document is believed to be accurate, it is under development and ACKme Networks reserves the right to make changes without further notice to the product described herein to improve reliability, function, or design, and makes no guarantee or warranty concerning the accuracy of said information, nor shall it be responsible for any loss or damage of whatever nature resulting from the use of, or reliance upon, such information. ACKme Networks makes no warranties of any kind, whether express, implied or arising by custom or course of trade or performance, and specifically disclaims the implied warranties of title, non-infringement, merchantability, or fitness for a particular purpose.

No part of this document may be copied, reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photographic, or otherwise, or used as the basis for manufacture or sale of any items without the prior written consent of ACKme Networks.

Trademarks

ACKme Networks and the ACKme Networks logo are trademarks of ACKme Networks. WICED™ is a trademark of Broadcom® Corporation, Inc. Other trademarks in this document belong to their respective owners.

Copyright © 2014 ACKme Networks, Inc.

Contact

<http://ack.me/contact>

About this User Guide

This guide provides information and usage instructions for the WiConnect serial Wi-Fi application that runs on Wi-Fi modules available from ACKme Networks.

An on-line version of this document is available at:

<http://wiconnect.ack.me>

Organization

This document is organized into the following sections:

- General Overview
- Getting Started with WiConnect
- WiConnect Web App
- Serial Interface
- Wi-Fi Interface
- Networking & Security
- Peripherals
- File System
- Memory
- Configuration and Setup
- System Functions
- Upgrade and Recovery
- Application Examples
- Troubleshooting Guide
- WiConnect Commands
- WiConnect Variables
- Release Notes
- Revision History & Glossary

Contents

Contents	4
General Overview	20
What is WiConnect?	20
Getting Started	21
WiConnect Web App	21
Serial Interface	21
Wi-Fi	21
Networking and Security	22
Peripherals	22
File System	22
Memory	22
Configuration and Setup	22
System Functions	23
Upgrade and Recovery	23
goHACK.me	23
Getting Started with WiConnect	24
Using the WiConnect Web App	24
Opening a WiConnect Terminal	24
Configuring the Module for a Wi-Fi Network	25
Verifying the Internet Connection	25
WiConnect Applications	25
Getting Help	25
WiConnect Web App	27
Setup Web Mode	27
Local Mode	29
Internet Mode	29
Automatically Connecting in Internet Mode	29
Manually Connecting in Internet Mode	29
WiConnect Web App Features	30
Connect Tab	30
Network Tab	31
Console Tab	31
Alias	32
Clear	32
Grep	32
History	32
Remove (rm)	33
System Tab	34
Files Tab	35
GPIO Configuration Tab	35
Firmware Management	36
Cloud Services	37
Customizing the WiConnect Web App	37
Serial Interface	38
UART Protocol	38
SPI Protocol	38
Interface Modes	38

Command Mode	38
Configuration	38
Command Format	39
Response Format	39
Log Format	40
Stream Mode	40
Configuration	41
Breaking out of Stream Mode	41
Packet Mode	41
Wi-Fi Interface	42
WLAN Client Interface	42
Scanning for Networks	42
Joining a Network	42
Wi-Fi Roaming	43
Access Point (SoftAP) Interface	43
Networking and Security	44
DHCP Client and Server	45
DHCP Server	45
DHCP Client	45
TCP, UDP and TLS Client and Server	46
TCP Client Auto Connect	46
UDP Client Auto Connect	46
UDP Server Auto Connect	46
HTTP Library	47
HTTP Post Example	47
HTTP Server with RESTful API	48
Command Request/Response	48
GET Request	48
POST Request	49
HTTP Response Codes	49
Response body	49
Log Request/Response	50
GET Request	50
HTTP Response Codes	50
Response	50
Web Socket Stream	50
HTTP Response Codes	50
HTTP Server Security and Authorization	50
Client Authorization	50
Protecting or Securing a File	51
Securing the REST API, Whitelisting API Calls	51
CORS (Cross Origin Resource Sharing)	51
HTTP Server Security Variables	52
Network Status Indication Using GPIOs	52
Network Connections and Streams	53
Remote Terminal Access	54
Network Discovery	54
Overview	54
OS Support	54

Apple	54
Windows	54
Linux	55
Android	55
In a Nutshell	55
Commands	55
Variables	55
Broadcast Status Announcement	55
Network Time Management	55
Sending Email using SMTP	56
Sensors.com API	56
Peripherals	57
GPIO Commands and Variables	57
GPIO Commands	57
GPIO Variables	57
GPIO Functions and Pins	58
Wallaby Module and Mackerel Evaluation Board	58
Numbat Module and Moray Evaluation Board	59
Viewing GPIO Usage	60
Setting GPIO Function	61
System Indicator Functions	62
Bus Stream Command GPIO	62
GPIO Controlled Network Connection	64
GPIO Controlled Script	64
Network Status GPIO	64
TCP Server Status GPIOs	64
Out-of-Band Interrupts	64
Using a GPIO Alias	64
Setting and Getting GPIO Values	65
ADCs	65
ADC Lookup Tables	66
PWMs	67
Wallaby PWMs	67
Numbat PWMs	67
LEDs	68
GPIO Standard I/O Control	68
PWM control - Blinking and Dimming LEDs	68
Communicating Peripheral Information Over the Network	69
Peripheral GPIO Mapping by Function	70
Wallaby GPIOs by Function	70
Wallaby UART	70
Wallaby ADC	70
Wallaby PWM	70
Wallaby SPI Master (System Use Only)	71
Wallaby DAC	71
Numbat GPIOs by Function	71
Numbat UART	71
Numbat ADC	71
Numbat PWM	72

Numbat SPI Master (System Use Only)	72
Numbat SPI Slave (Numbat v4+ Only)	72
Temperature versus ADC value	73
Module Comparison	76
File System	77
File Listing	77
File Types	78
File Checksum	78
Special Files	79
Writing Files to the File System	79
Writing with the WiConnect Web App File Browser	79
Writing with a WiConnect Terminal or Remote Terminal	80
Writing File Contents to a Stream in Chunks	80
HTTP Download	81
OTA	81
Reading Files from the File System	81
Reading with the WiConnect Web App File Browser	81
Reading from a WiConnect Terminal or Remote Terminal	82
HTTP Upload	82
Internal, Extended and Bulk Flash	82
Memory	83
Memory Management	83
Determining Memory Usage	83
Minimizing Memory Usage	84
Streams	85
Memory Management Variables	85
HTTP clients	85
Network Buffer	85
goHACK.me Cache	85
Configuration and Setup	86
Wi-Fi Setup	86
Setup with a Web browser	87
Starting Web Setup Mode on an ACKme Evaluation Board	87
Starting Web Setup Mode from a WiConnect Terminal	87
Opening the Web Setup Page	87
Web Setup Timeout	88
Setup by WiConnect Terminal	88
Network Setup Option	89
Setup by Remote Terminal	89
Initiating Setup by GPIO	89
Initiating Setup on Boot	90
Setup by WPS Push-button or PIN	90
Setup Configuration Script	90
Automatically Executing a Script	91
On Boot	91
On GPIO Assertion or Reboot	91
Displaying Comments and Command Output	91
Script Format	91
Variable Configuration	93

GPIO Configuration	94
Using a GPIO Configuration File	94
Standard GPIO Functions	95
Sleep State	95
Alternative GPIO Functions	95
System Indicator State Options	95
Example GPIO config file	95
System Functions	97
Configuring WiConnect Command Mode and Response	97
Power management	97
Sleeping and Waking	97
Powersave Mode	98
Monitoring System State	98
Controlling the Boot Application	98
System Identification and Version	98
Upgrade and Recovery	99
Secure OTA Upgrade	99
Safe Mode	99
Factory Reset	101
Application Examples	102
Notes	102
Recent Changes	102
By Topic	102
Serial Interface	102
Stream Mode	103
WLAN	103
TCP Client	103
TCP Server	104
UDP Client	104
HTTP Server with RESTful API, JavaScript API, Python API	104
mDNS Network Discovery	104
Soft AP	104
Peripherals	104
Transmitting ADC Data	104
Firmware Update & Recovery	104
goHACK.me	104
TCP Server + softAP	106
Change Log	108
TCP Client	109
Connecting	109
Writing Data	111
Checking for Data	111
Reading Data	112
Change Log	112
Multiple TCP Clients	113
Connecting	113
Writing Data	115
Checking for Data	115
Reading Data	116

Change Log	116
Wireless Serial Port	117
Features demonstrated	117
Module 1 Server Configuration Settings	117
Module 2 Client Configuration Settings	118
Wireless Magic!	118
Change Log	118
Controlling GPIOs and LEDs	120
GPIO Usage	120
LED Control	120
GPIO Initialization	121
Change Log	121
Multi-client WLAN Communications	122
A Simple Message Protocol	122
Implementation	122
Verify it works	125
Other Options?	125
Change Log	125
Wi-Fi Remote Terminal	127
Features demonstrated	127
Setting up the Soft AP and Remote Terminal	127
View the Default State of the Soft AP and Remote Terminal Variables	128
Change the Soft AP and Remote Terminal Variables	128
View the Changed State of the Soft AP and Remote Terminal Variables	129
Connecting the Remote Terminal	129
Change Log	130
Secure TLS Client	131
Overview	131
TLS Certificate Primer	131
Certificate File Types	131
Certificate Types	131
OpenSSL	132
Generating a Self-Signed CA Certificate	132
Generating a Server Certificate	133
Starting the Python TLS Server	134
Loading the CA cert onto the Module (using WiConnect)	134
Connecting to the Python TLS Server	134
Scripts	135
Notes for Windows & Additional Links	135
Change Log	136
Broadcast UDP Packet	137
Setup	137
Method	137
Open a WiConnect Terminal to the ACKme Module	137
Associate Module to Wi-Fi Network	137
Run Python Script	138
Change Properties in Broadcast UDP Packet	138
Set the IP Address to the Test Computer	139
Set IP address to Target Other than Test Computer	139

Send ADC Values and GPIO Values in the Broadcast UDP Packet	139
Change Log	140
High Speed UART	141
Notes for Settings	145
Using the Module 1 Soft AP	145
Setup Instructions	145
Update Settings	145
Connect UARTs	146
Run Python Script	146
Change Log	147
GPIO-Controlled Network Connection	148
Features Demonstrated	148
Method	148
Setup	149
Performing a GPIO-Controlled Host Connection	150
Using Stream Mode	151
Change Log	152
goHACK.me Slave Mode	153
Factory Reset	153
Connect to Wi-Fi	153
Create a Caps File	153
Set the Caps File	155
Activate	155
Write Streams & Read Controls	155
Send a message to goHACK.me	156
Read a message from goHACK.me	156
Change Log	157
goHACK.me Solo Mode	158
Caps & Solo Mode Setup Files	158
The Caps File Explained	159
The Solo Mode Setup File Explained	160
Let's go SOLO!	161
Factory Reset	161
Connect to Wi-Fi	161
Solo Mode Setup	161
Activate	162
Monitoring & Control with goHACK.me	162
Change Log	162
goHACK.me Triggers	163
Features demonstrated	163
Method	163
Procedure	164
Editing Device Details	164
Verifying goHACKme Connections	165
Adding a Trigger	165
Adding an Action to a Trigger	167
Triggering Actions	169
Change Log	171
goHACK.me Nest Integration	172

Features demonstrated	172
Method	172
Procedure	172
Authorizing goHACK.me with Nest	172
Create a Trigger with a Nest Thermostat Action	173
Trigger the Nest Thermostat Action	176
Change Log	176
Device Discovery and HTTP RESTful API	177
Setup	177
Connect Your Module to the Local Network	177
Enable the mDNS and HTTP Server	177
Syntax of REST API WiConnect Commands	178
Sending Commands from a Web Browser	179
Troubleshooting	179
How does this work?	179
Using cURL	180
WiConnect Version	180
Verbose file list	180
Open and Read a file	181
Change Log	181
Secure HTTP Server	182
Overview	182
Command Line Conventions	182
Command Line Prompt	182
Command Line Continuation	182
TLS Certificate Overview	182
OpenSSL	183
Generating a Self-Signed CA Certificate	183
Generating a Server Certificate	184
Authenticating the Client	186
Generating a Client Certificate and Key	186
Loading the Server cert onto the Module (using WiConnect)	188
Run WiConnect as a Secure HTTPS Server	189
HTTP Server / Client Authentication	190
HTTP Server Username and Password	191
Notes for Windows & Additional Links	191
Change Log	192
HTTP Server Simple WebSocket Demonstration	193
Setup	193
Configure WiConnect Device	193
Download and Launch the HTML page	194
Command Mode	194
Stream Mode	195
JavaScript	195
mDNS Device Discovery	197
Setup	197
Connect Your Module to the Local Network	197
Enable the mDNS Daemon	197
Ping the Module using the mDNS Domain	198

Advertise an mDNS Service	198
Change Log	199
Recovery from Safemode	201
Force Safe Mode	201
Listing Faults	201
Return to Normal Operation	202
Change Log	202
TCP Client and UDP Client Auto-Connect	203
Description	203
Setup	203
TCP Auto Connect	203
UDP Auto Connect	204
Change Log	204
goHACK.me	205
Getting Started with goHACK.me	205
Using WiConnect with goHACK.me	206
Slave vs. Solo Mode	206
Signup and Device Activation	206
Device Capabilities	207
Streams	207
Controls	208
Messages	208
Example Caps File	209
The goHACK.me API	211
Slave Mode	211
Solo Mode	211
goHACK.me Basic Account	212
Troubleshooting Guide	213
Basics	213
My module fails to associate to the network	213
My module has missing commands or variables. What do I do?	213
Recovery from Safe Mode	213
Recovery from Interrupted OTA Upgrade	214
WiConnect commands	215
Nav Tips for Humans	215
Shortcuts	215
Documentation Format	215
command	215
Alphabetical List of Commands	217
Description of Commands	219
adc_take_sample	219
D	219
dac_set_level	219
E	220
exit	220
F	220
factory_reset	220
faults_print	221
faults_reset	221

file_create	222
file_delete	223
file_open	224
format_flash	224
force_safemode	225
G	225
get	225
ghm_activate	226
ghm_capabilities	227
ghm_deactivate	229
ghm_message	230
ghm_read	232
ghm_signup	233
ghm_sync	233
ghm_write	233
gpio_dir	234
gpio_get	235
gpio_set	235
gpios_dir	236
gpios_get	237
gpios_set	237
H	238
help	238
http_add_header	239
http_download	239
http_get	242
http_head	243
http_post	244
http_read_status	245
http_upload	246
L	247
ls	247
load	248
M	248
mdns_discover	248
N	249
network_down	249
network_lookup	250
network_up	250
network_verify	251
ota	252
P	254
ping	254
pwm_update	254
R	255
reboot	255
S	255
save	255
set	256

setup	257
sleep	258
smtp_send	259
stream_close	260
stream_list	260
stream_poll	261
stream_read	261
stream_write	262
T	262
tcp_client	262
tcp_server	263
tls_client	264
tls_server	264
U	265
udp_client	265
udp_server	265
V	267
version	267
W	267
wlan_get_rssi	267
wlan_scan	268
wps	269
WiConnect Variables	270
Shortcuts	270
Documentation Format	270
variable	270
List of Variables	272
Variable Description	277
ALL	277
all	277
Broadcast	277
broadcast.data	277
broadcast.http.host	279
broadcast.interface	279
broadcast.interval	280
broadcast.udp.data	281
broadcast.udp.interface	281
broadcast.udp.interval	281
broadcast.udp.ip	281
broadcast.udp.port	282
Bus	283
bus.data_bus	283
bus.log_bus	284
bus.mode	284
bus.stream.cmd_gpio	285
bus.stream.cmd_seq	285
bus.stream.flush_count	286
bus.stream.flush_time	287
bus.stream.flush_time_reset	288

Email	288
email.name_address	288
email.smtp.host	289
email.smtp.password	290
email.smtp.port	291
email.smtp.username	291
goHACK.me	292
ghm.capabilities	292
ghm.cache_size	293
ghm.solo.echo	293
ghm.solo.enabled	294
ghm.solo.gpio	294
ghm.solo.sync_period	296
ghm.status	297
GPIO	297
gpio.alias	298
gpio.config_file	298
gpio.init	299
gpio.sleep	300
gpio.usage	301
HTTP Server	301
http.server.api_enabled	301
http.server.cors_origin	302
http.server.denied_filename	303
http.server.enabled	303
http.server.interface	304
http.server.max_clients	305
http.server.notfound_filename	305
http.server.password	306
http.server.port	306
http.server.root_filename	307
http.server.tls_cert	308
http.server.tls_enabled	308
http.server.tls_key	309
http.server.tls_verify_peer	309
http.server.username	310
IO Connection	311
ioconn.control_gpio	311
ioconn.enabled	311
ioconn.local_port	312
ioconn.protocol	313
ioconn.remote_host	313
ioconn.remote_port	314
ioconn.status_gpio	315
mDNS	315
mdns.enabled	315
mdns.interface	316
mdns.name	317
mdns.service	318

mdns.ttl	320
Network	321
network.buffer.rxtx_ratio	321
network.buffer.size	321
network.buffer.usage	322
network.ca_cert	323
network.default_interface	323
network.dhcp	324
network.dhcp.enabled	324
network.dhcp.timeout	325
network.dns	326
network.gateway	326
network.ip	327
network.netmask	327
network.status	328
network.status_gpio	328
ntp.enabled	329
ntp.interval	329
ntp.server	330
OTA Upgrade	331
ota.port	331
ota.host	331
Remote Terminal	332
remote_terminal.enabled	332
remote_terminal.interface	333
remote_terminal.password	334
remote_terminal.port	334
remote_terminal.timeout	335
Setup	335
setup.auto.cmd	336
setup.gpio.cmd	336
setup.gpio.control_gpio	337
setup.gpio.level	338
setup.gpio.mode	338
setup.web.client_list	339
setup.web.passkey	340
setup.web.root_filename	341
setup.web.ssid	342
setup.web.url	342
SoftAP Interface	343
softap.auto_start	343
softap.channel	344
softap.client_list	344
softap.dhcp_server	345
softap.dns_server	346
softap.hide_ssid	346
softap.idle_timeout	347
softap.info	348
softap.ip	348

softap.passkey	349
softap.rate.protocol	349
softap.rate.transmit	350
softap.ssid	351
softap.url	351
Static Network Settings	352
static.dns	352
static.gateway	353
static.ip	353
static.netmask	354
Stream Settings	355
stream.auto_close	355
System	355
system.adc.vref	356
system.activity gpio	356
system.activity gpio_level	357
system.bflash.cs_gpio	358
system.boot_app	358
system.cmd.buffered	359
system.cmd.echo	360
system.cmd.header_enabled	361
system.cmd.mode	361
system.cmd.prompt_enabled	362
system.cmd.timestamp	363
system.gotosleep.timeout	364
system.indicator gpio	364
system.indicator.state	365
WLAN Indicator	365
Network Indicator	365
SoftAP Indicator	366
system.memory_usage	366
system.msg	367
system.oob.event_mask	368
system.oob gpio	369
system.oob gpio_level	370
system.oob.rising_edge_mask	371
system.oob.status	371
system.powersave.mode	372
system.print_level	373
system.safemode	373
system.uuid	374
system.version	375
system.wakeup.events	375
system.wakeup.timeout	376
TCP Client	376
tcp.client.auto_interface	377
tcp.client.auto_retries	377
tcp.client.auto_start	378
tcp.client.connect_timeout	379

tcp.client.local_port	379
tcp.client.remote_host	380
tcp.client.remote_port	380
tcp.client.retries	381
tcp.client.retry_period	382
TCP Keepalive	382
tcp.keepalive.enabled	382
tcp.keepalive.initial_timeout	383
tcp.keepalive.retry_count	384
tcp.keepalive.retry_timeout	385
TCP Server	385
tcp.server.auto_interface	385
tcp.server.auto_start	386
tcp.server.connected_gpio	387
tcp.server.data_gpio	387
tcp.server.idle_timeout	388
tcp.server.max_clients	389
tcp.server.port	389
tcp.server.tls_cert	390
tcp.server.tls_enabled	391
tcp.server.tls_key	391
tcp.server.tls_verify_peer	392
Time	393
time.last_set	393
time rtc	393
time.uptime	394
time.zone	394
UART	395
uart.baud	395
uart.data	396
uart.flow	397
uart.parity	397
uart.stop	398
UDP Client	399
udp.client.auto_interface	399
udp.client.auto_start	399
udp.client.remote_host	400
udp.client.remote_port	400
UDP Server	401
udp.server.auto_interface	401
udp.server.auto_start	402
udp.server.data_gpio	402
udp.server.lock_client	403
udp.server.port	404
udp.server.remote_host	405
udp.server.remote_port	405
WLAN Interface	406
wlan.antenna	406
wlan.auto_join.enabled	407

wlan.auto_join.retries	407
wlan.auto_join.retry_delay	408
wlan.hide_passkey	409
wlan.info	409
wlan.join.result	410
wlan.join.retries	411
wlan.join.timeout	412
wlan.mac	412
wlan.passkey	413
wlan.powersave.listen_interval	414
wlan.powersave.mode	414
wlan.powersave.sleep_delay	415
wlan.rate.protocol	416
wlan.rate.transmit	417
wlan.roam.threshold	417
wlan.rssi_average	418
wlan.scan.active_dwell	419
wlan.scan.channel_mask	419
wlan.scan.home_dwell	420
wlan.scan.num_probes	421
wlan.scan.passive_dwell	421
wlan.scan.retries	422
wlan.scan.type	423
wlan.security	423
wlan.ssid	424
wlan.tx_power	425
Release Notes for WiConnect v2.1	426
Versions	426
Known Issues (v2.1.0)	426
Changelog	427
Commands Added	427
Commands Changed	427
Variables Added	427
Variables Changed	429
Revision History	430
Glossary	430

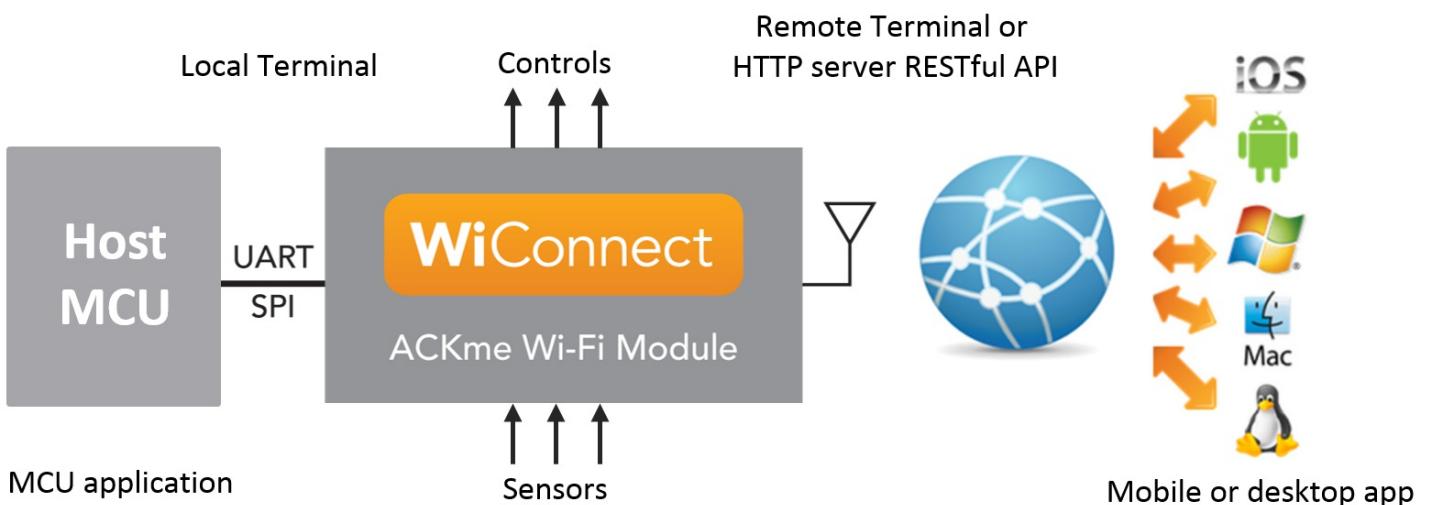
General Overview

What is WiConnect?

WiConnect is an embedded serial Wi-Fi application that provides a host microprocessor with a secure wireless connection to a local network or to any device that is part of the universal Internet of Things. The application is designed to run on various wireless modules provided by ACKme Networks.



For our Bluetooth Low Energy API, see [TruConnect](#).



An ACKme module is an Internet-connected monitoring and control device, operated under local MCU control or under remote control by a mobile or desktop application

A host MCU communicates with WiConnect via a UART or SPI serial interface.

A mobile or desktop app communicates with WiConnect via the HTTP Server RESTful API / Websockets or the remote terminal interface.

The local or remote host sends commands to control the operation of the application and to send data to, and receive data from the network. Application operation may be customized by setting individual configuration variables.

WiConnect provides a powerful set of [commands](#) to enable host control, together with a large number of [variables](#) to enable host configuration. Beyond commands and variables, WiConnect provides a rich feature set for wireless, network, peripheral and file system command and control.

Features include:

- Interfaces: serial, soft AP and WLAN Client
- Servers: TCP/TLS, UDP, HTTP(S), DHCP
- Clients: TCP/TLS, UDP, NTP, SMTP, DHCP, DNS

- Network Discovery with support for mDNS, LLMNR & Netbios
- GPIOs for control, indication and monitoring
- Remote terminal access
- HTTP Server with RESTful API and Websockets
- OTA (over the air) Update to remotely manage firmware via the wireless interface
- Scripts and configuration files
- Automated broadcast and streaming of data
- Local caching of sensor data
- Support for cloud services - goHACK.me and sensors.com
- Support for large file storage in bulk serial flash

All WiConnect commands can be issued manually (human command mode) or under host control (machine command mode). Human command mode is typically used during evaluation and development of a WiConnect application. Machine command mode is used during production.

For human command mode, the interfaces are a [WiConnect Terminal](#), the [WiConnect Web App](#), or a remote terminal.

Getting Started

To start using your WiConnect module in human command mode, see [Getting Started](#).

WiConnect Web App

The WiConnect Web App provides complete monitoring and control of your ACKme module using a web browser.

See [WiConnect Web App](#).

Serial Interface

WiConnect provides one of the easiest to use and most flexible serial Wi-Fi networking interfaces in the industry. The WiConnect serial interface provides support for multiple serial buses including multi-UART and serial-SPI, three different serial communications modes including [command mode](#), [stream mode](#) and [packet mode](#), and is easily configured for use by humans and machines alike.

See [Serial Interface](#).

Wi-Fi

WiConnect provides an easy-to-use serial API allowing you to focus your efforts on application development, rather than the time-consuming complexity and interoperability issues associated with the development of a full-featured secure wireless network stack.

Two Wi-Fi interfaces are supported, including a Wi-Fi client (wlan) and Wi-Fi soft access point (softAP) interface. Each interface may be run independently or concurrently to suit various application needs.

See [Wi-Fi Interface](#).

Networking and Security

WiConnect includes a full IPv4 networking stack and SSL/TLS security suite that supports a range of popular networking protocols including TCP, UDP, DNS, DHCP. Additional network application libraries are provided for native HTTP/S and secure cloud access using the goHACK.me API.

See [Networking and Security](#).

Peripherals

Various peripheral APIs are available to provide access to general purpose IOs (GPIOs), analog functions such as analog-digital and digital-analog converters, and timers including a real time clock. Special system and network functions can be assigned to GPIOs to enable handy features including:

- automatic LED status indicators to show the state of Wi-Fi and network connections;
- an indication of when one or more network clients are connected to a local WiConnect server; and
- the ability to control the connection state of a client or server.

See [Peripherals](#).

File System

Most connected applications require the ability to store configuration information or cache local data. WiConnect provides a reliable read/write filesystem and on-board flash storage to satisfy application storage requirements. With just a few simple commands, WiConnect provides your application with complete access to local and network connected storage.

See [File System](#).

Memory

For management of ACKme module memory, see [Memory](#).

Configuration and Setup

Several options are available to setup a module running WiConnect. The most common option, and also the most convenient when prototyping is to use a command line and serial interface such as a UART. A PC terminal can be quickly connected to a WiConnect evaluation board, and within 60 seconds it is possible to be connected to a Wi-Fi network and downloading HTTP webpages or connecting to a remote TCP server.

Other methods for configuration, setup and Wi-Fi provisioning include:

- a softAP and webserver interface, or use of WPS push-button PBC or PIN for Wi-Fi provisioning;
- an (optionally) password-secured remote terminal that provides command line convenience over one of the WiConnect wireless network interfaces; and
- automatic execution of a custom configuration script on boot-up

See [Configuration and Setup](#).

System Functions

System functions include:

- Configuring WiConnect Command Mode and Response
- Power management
- Monitoring System State
- Controlling the Boot Application
- System Identification and Version

See [System Functions](#).

Upgrade and Recovery

WiConnect is part of a larger sophisticated application framework installed on all ACKme modules. The framework provides a hardened bootloader, read/write file system, safemode recovery mechanism, and over-the-air (OTA) upgrade capability.

ACKme manages a secure on-line OTA server that offers each ACKme module with the ability to securely upgrade individual files, applications or an entire firmware bundle. In the rare event that WiConnect fails to regularly execute correctly, the bootloader switches the boot application to safe mode. The safe mode application provides the host with the ability to upgrade the firmware, switch back to WiConnect or even factory reset the module.

See [Upgrade and Recovery](#).

goHACK.me

The [goHACK.me](#) API provided by WiConnect enables monitoring and control of your ACKme device. Using [goHACK.me commands](#) and [goHACK.me variables](#), you can setup your ACKme device and automatically signup for an account, as well as synchronize data, control and messages with the goHACK.me cloud service.

See [goHACK.me](#).

Getting Started with WiConnect

WiConnect provides a secure wireless network link between your host microcontroller (MCU) and a network.



To join a wireless network, you need to supply the module with network credentials: the wireless network name (SSID) and password. There are several ways to do this.

Let's get started!

Using the WiConnect Web App

The [WiConnect Web App](#) provides a simple way to configure the module network credentials using a web browser. This is particularly convenient for configuring ACKme evaluation boards, which can be put in setup web mode without connecting a WiConnect terminal. See [WiConnect Web App, Setup Web Mode](#).

Opening a WiConnect Terminal

You can control your ACKme module with WiConnect commands via a USB UART serial terminal:

- Plug your ACKme WiConnect evaluation board into a USB port (USB provides power and UART serial connection.)
- Open a communication terminal application e.g. Teraterm
- Find the UART serial (COM) port associated with the eval board
- Set the communication parameters to 115200 8N1 with NO hardware flow control
- Press the enter key. WiConnect responds with a ready message and command prompt:
Ready
>
- At the WiConnect prompt, type `version` and check for a response similar to the text below.
WiConnect-2.0.0.11, Built:2014-10-20 13:35:04 for AMW006.4, Board:AMW006-E03.2

If problems arise, see the WiConnect [Troubleshooting Guide](#).

Configuring the Module for a Wi-Fi Network

You need to provide your module with your Access Point (AP) SSID and password. You can do this via the serial terminal, with the following commands.

WiConnect Commands	Description
set wlan.ssid "YOUR AP NAME"	Name of your Access Point
set wlan.passkey "YOUR AP PASSWORD"	Password for your Access Point

When you issue any WiConnect command that requires network access, WiConnect attempts to bring the network up using these properties.

ACKme evaluation boards have a network indicator LED to help determine successful association. Enter the [network_up](#) command (nup). The network indicator LED changes its blink rate from fast (no IP address), to medium (DHCP in progress) to slow (DHCP successful, IP address obtained).

There are several other ways to configure your module to connect to a Wi-Fi network. See [Configuration and Setup](#).

Verifying the Internet Connection

To demonstrate that the module is connecting to the Internet, try downloading a web page:

WiConnect Commands	Description
http_get http://google.com	<- Google web address
stream_read 0 1000	<- Read 1000-bytes of the web page

WiConnect Applications

Now that you have established control over your ACKme module, you can experiment with the rich set of WiConnect features. There are many application notes demonstrating WiConnect networking applications. See [application examples](#).

Getting Help

As well as this WiConnect documentation, there are various WiConnect help commands:

- At the WiConnect prompt, enter [help](#).

The following help options are available ...

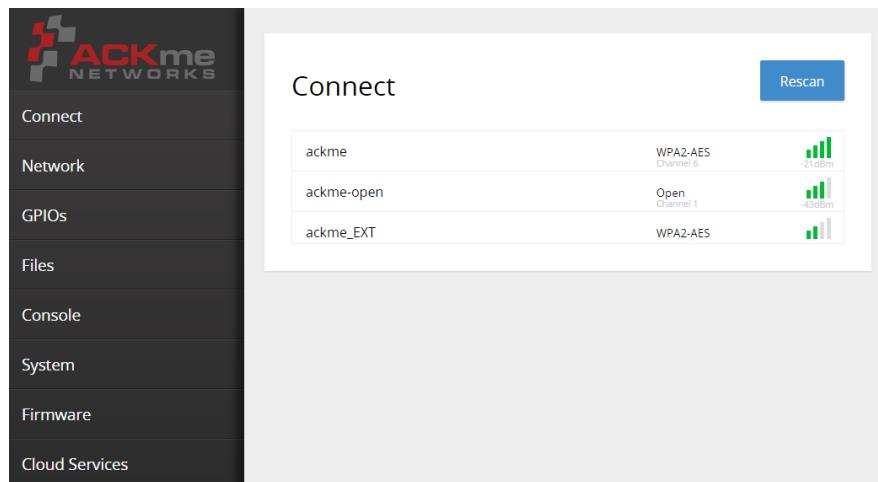
```
help all      -> Print a list of all Commands and Variables
help commands -> Print a list of Commands
help variables -> Print a list of readable Variables
help <command> -> Print help for a specific Command
help <variable> -> Print help for a specific Variable
```

Additional help is available online at <http://wiconnect.ack.me>

- Enter **help all** for a full list of **commands** and **variables**.
- Enter **help commands** to list commands only, or **help variables** to list variables only.
- Enter **help <command_or_variable_name>** to show help for an individual command or variable.

WiConnect Web App

The WiConnect Web App provides complete monitoring and control of your ACKme module using a web browser.



The web app can be run in any one of three modes:

- **Setup web mode**
- **Local mode**, without Internet access
- **Internet mode**, connected to a wireless network with Internet access

The WiConnect Web App serves as a complete demonstration showing how to use WiConnect with the [HTTP RESTful API](#).

Setup Web Mode

In the factory default configuration, your module is not configured to connect with a wireless network and does not have Wi-Fi access.

You can use a Wi-Fi client with a web browser, such as a smartphone, tablet or PC, to connect to the WiConnect web app via the module softAP wireless network to set up access to your wireless network.

WiConnect evaluation boards are by default configured to enter Setup Web mode when you hold down Button 2, press Reset, and continue to hold down Button 2 for at least three seconds.

On any ACKme WiConnect device you can also make the module start Setup Web mode by [opening a WiConnect terminal](#) and issuing the command:

```
setup web
```

On evaluation boards, the red LED blinks when the module soft AP network is turned on and waiting for a connection.

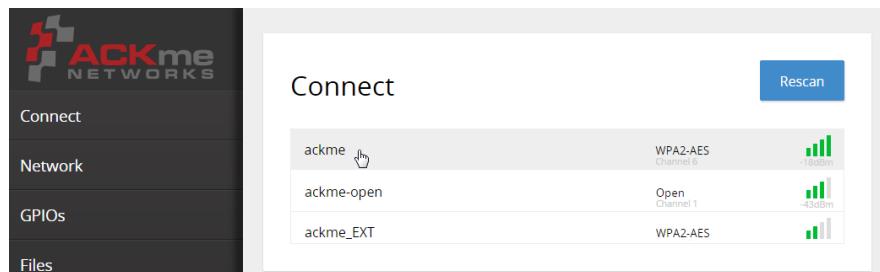
Using the Wi-Fi settings on your computer or mobile device, connect to the module soft AP wireless network. By default, the soft AP network name (SSID) is WiConnect-###, where ### is replaced by the last 3 digits of the module MAC address, e.g. WiConnect-

2D6. The default password is: password.

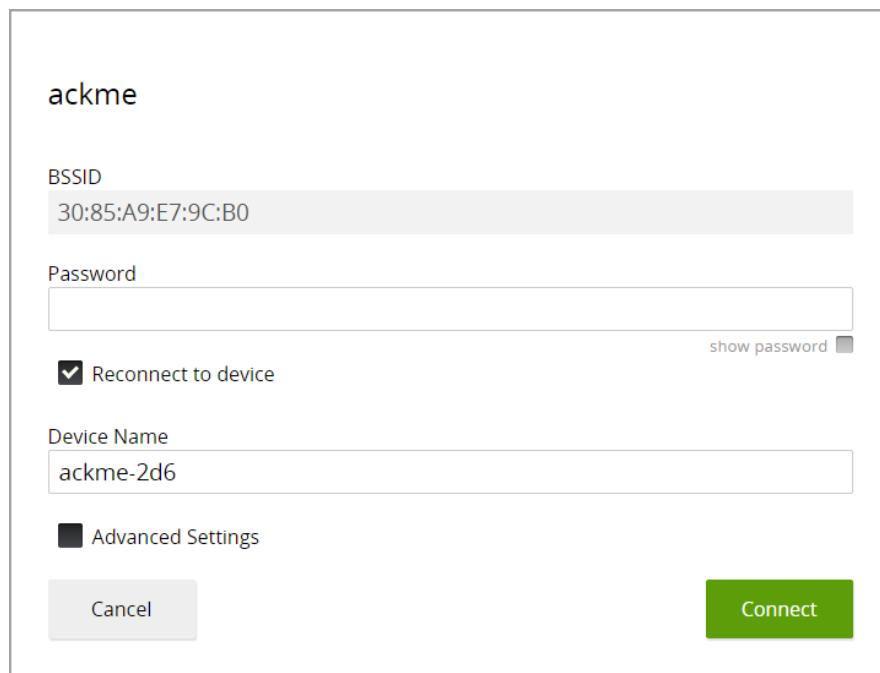
Open a web browser and navigate to `setup.com` or `wiconnect.com`.

The WiConnect Web App opens the Connect tab by default, and scans for Wi-Fi networks in range.

Click the name of your Wi-Fi network to select it.



Supply the network password in the following dialog.



ackme

BSSID
30:85:A9:E7:9C:B0

Password

show password

Reconnect to device

Device Name
ackme-2d6

Advanced Settings

Cancel Connect

The web app configures your module, saves the variables, and instructs the module to connect to your network.

If you choose the **Reconnect to device** option, the web app restarts your module and reconnects to it via your network, in **Internet mode**. This assumes that after disconnecting from the module soft AP network, your computer also reconnects to your network automatically. If the web app fails to reconnect, check the Wi-Fi on your computer or mobile device has roamed back to your network.

Local Mode

Local mode provides all web app features (tabs) that do not require Internet access. Features requiring Internet access are [Firmware Management](#) and [Cloud Services](#).

You can connect in Local mode via the module soft AP interface or via the module [wlan interface](#).

To set up the HTTP server for the WiConnect Web App, see the instructions in [Internet Mode](#).

Internet Mode

Internet mode requires a Wi-Fi network with Internet access. All features (tabs) are available.

Automatically Connecting in Internet Mode

The simplest way to use the WiConnect Web App in Internet mode is to set up the module with the web app in [Setup Web mode](#), and check the [Reconnect to device](#) option. The web app performs all the necessary configuration to reconnect in Internet mode. It sets the module mDNS domain to ackme-###, where ### is replaced by the last three digits of the module MAC address, e.g. ackme-2d6. The web app restarts the module and reconnects to it in Internet mode.

Manually Connecting in Internet Mode

Alternatively, you can manually configure the module to run the WiConnect Web App as described below.

Issue the following commands from a terminal:

```
set wlan.ssid          <YOUR NETWORK NAME>
set wlan.passkey       <YOUR NETWORK PASSWORD>
set wlan.auto_join.enabled 1
set http.server.enabled 1
set http.server.api_enabled 1
set mdns.enabled        1
set mdns.name           <YOUR MODULE NAME>
save
reboot
```

For example:

```
set wlan.ssid          ackme
set wlan.passkey       secretpassword
set wlan.auto_join.enabled 1
set http.server.enabled 1
set http.server.api_enabled 1
set mdns.enabled        1
set mdns.name           mymodule
save
reboot
```

After a reboot, the response from WiConnect is similar to the following:

```
Obtaining IPv4 address via DHCP
IPv4 address: 10.5.6.59
Starting mDNS
mDNS domain: mymodule.local
HTTP and REST API server listening on port: 80
[2015-01-15 | 23:48:42: Associated]
```

The module automatically connects to your network if `wlan.auto_join.enabled` is configured as described above.

With your computer or mobile device also connected to your network, open a web browser and navigate to the address of the Web App. The address is the mDNS domain name <YOUR MODULE NAME>.local (or the module IP address).

In the example above, you can connect a browser to either <http://mymodule.local> or <http://10.5.6.59> to use the WiConnect Web App. Some Windows systems may instead need to connect to <http://mymodule>, for further information see [Networking & Security, Network Discovery](#).

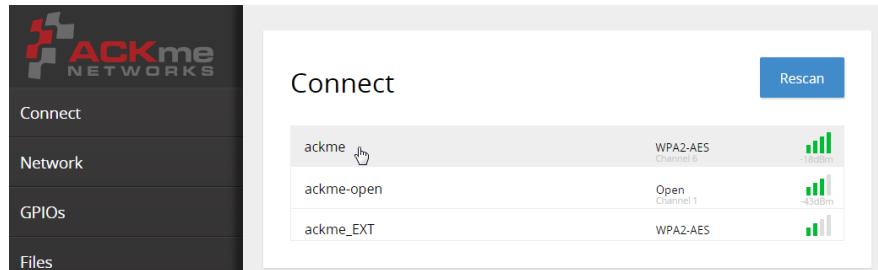
WiConnect Web App Features

The WiConnect Web App provides information and configuration tabs that allow complete monitoring and control of your module.

All features (tabs) of the web app are available in [Internet mode](#), unlike [Setup Web mode](#) and [Local mode](#), in which the [Firmware Management](#) and [Cloud Services](#) tabs are not available.

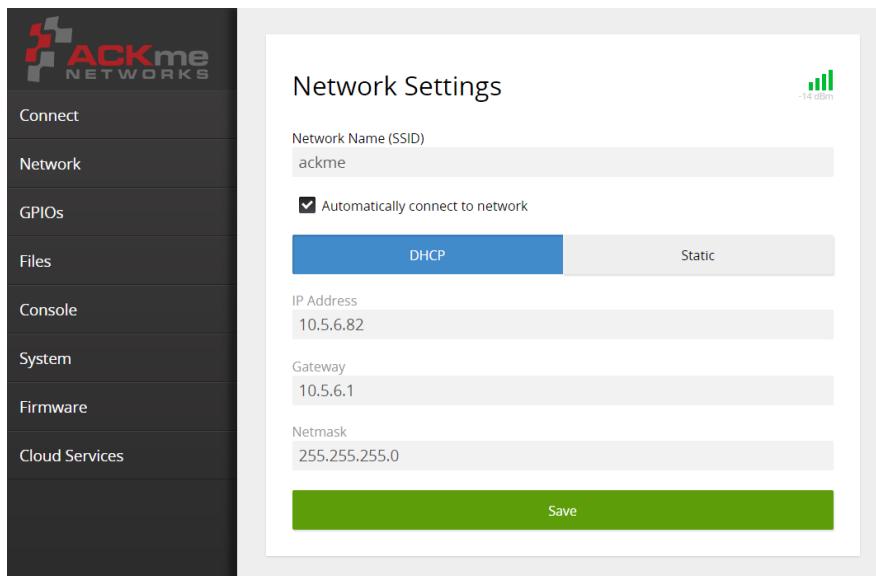
Connect Tab

The Connect tab provides a way to join your module to a Wi-Fi network. When the Connect tab opens, it automatically scans for Wi-Fi networks in range and allows you to select your network and enter a password, as described above in [Setup Web Mode](#).



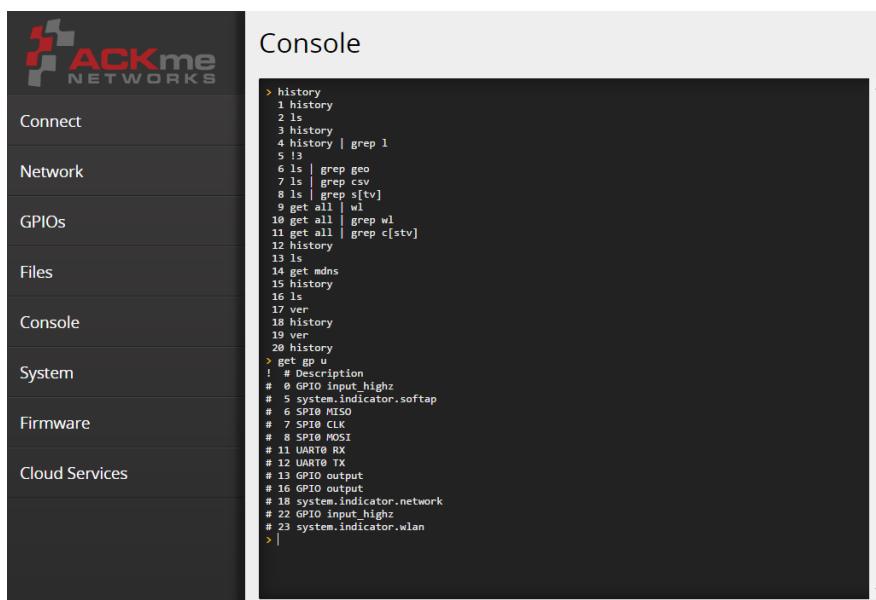
Network Tab

If your module is already connected to your wireless network, then by default the web app displays the Network tab, showing some essential WiConnect **wlan interface** variable settings.



Console Tab

The console tab works like a WiConnect terminal or remote terminal. You can issue any WiConnect command from this page. In the background, the web app uses the WiConnect **HTTP RESTful API** to issue commands to the module and display responses.



As well as standard WiConnect terminal behavior, the Console tab provides additional features, exclusive to the WiConnect Web App. Additional commands mimic Linux shell commands and include:

- alias
- clear
- grep
- history
- rm

Alias

The console alias command has the following syntax:

```
alias <alias_name>=<command_string>"
```

This lets you create a short name for a long command. Once an alias is set, you can type the <alias_name> in the console, in place of the full <command_string>. For example:

```
alias g="get gpio.usage"
```

To remove a previously aliased command, use unalias <alias_name>, e.g.

```
unalias g
```

Clear

The clear command clears all text on the display and returns the cursor to the top line on the console

Grep

The console grep command allows filtering of output. The following example lists only those commands in the history that contain the text http:

```
history | grep http
```

The grep syntax also allows regular expressions. The following example lists only those files on the module file system whose names contain c followed by either s, t or v:

```
ls | grep c[srv]
```

Use the option -v to invert the grep result and display all lines but those that match the regular expression:

```
ls | grep -v c[srv]
```

History

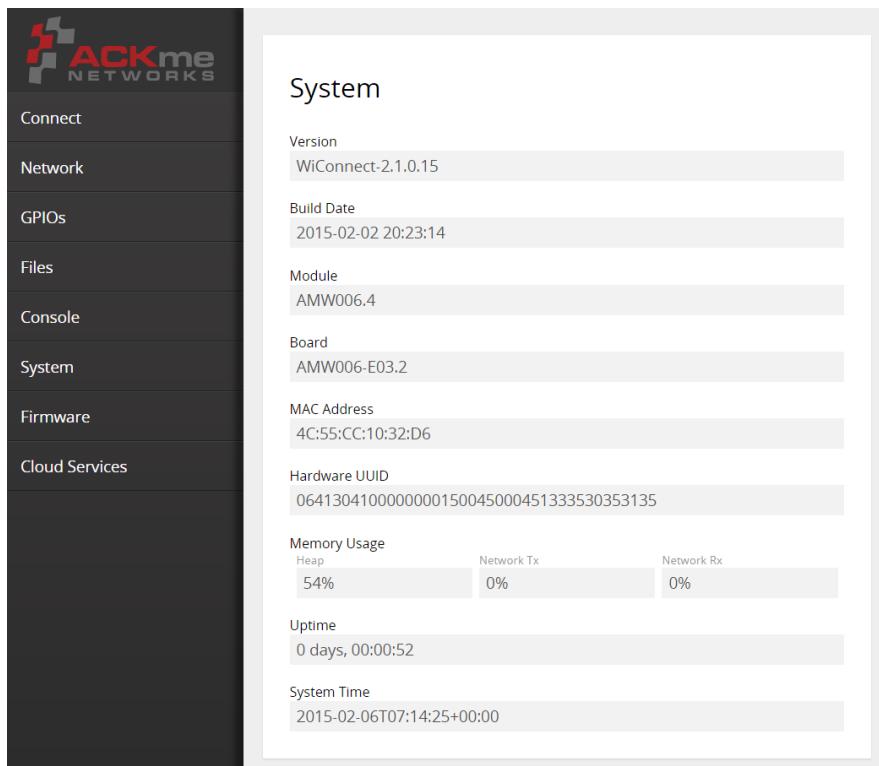
The history command displays a list of past commands issued on the console. You can re-issue a command by copying and pasting from the list, or by typing !i, where i is the index number of the command in the list. To return the last X entries in history, issue the command history X.

Remove (rm)

The `rm` command is identical to the WiConnect `file_delete` command. Either command may be used to delete files from the module file system.

System Tab

The system tab displays a number of system variables, such as version, build date, module, board and MAC address, memory usage and time.



The screenshot shows the 'System' tab selected in the left sidebar. The main content area displays various system parameters:

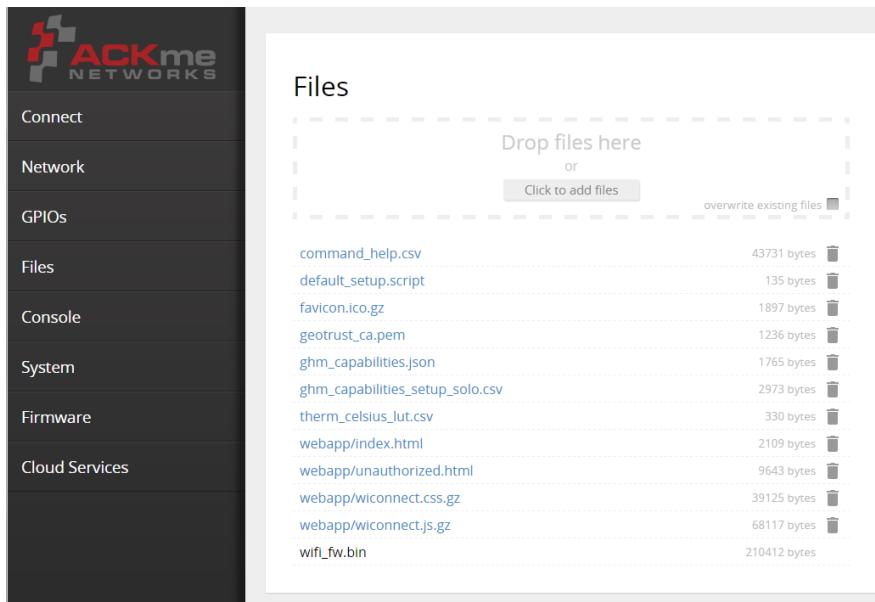
- Version: WiConnect-2.1.0.15
- Build Date: 2015-02-02 20:23:14
- Module: AMW006.4
- Board: AMW006-E03.2
- MAC Address: 4C:55:CC:10:32:D6
- Hardware UUID: 0641304100000000150045000451333530353135
- Memory Usage:

Heap	Network Tx	Network Rx
54%	0%	0%
- Uptime: 0 days, 00:00:52
- System Time: 2015-02-06T07:14:25+00:00

Files Tab

The Files Tab lists files stored on the module file system. You can upload, download and delete files.

- To upload files to the module file system, select **Click to add files** and browse, or drag and drop to the **Drop files here** area.
- To download a file from the module file system, click the file name.
- To delete a file, click the corresponding trash icon.

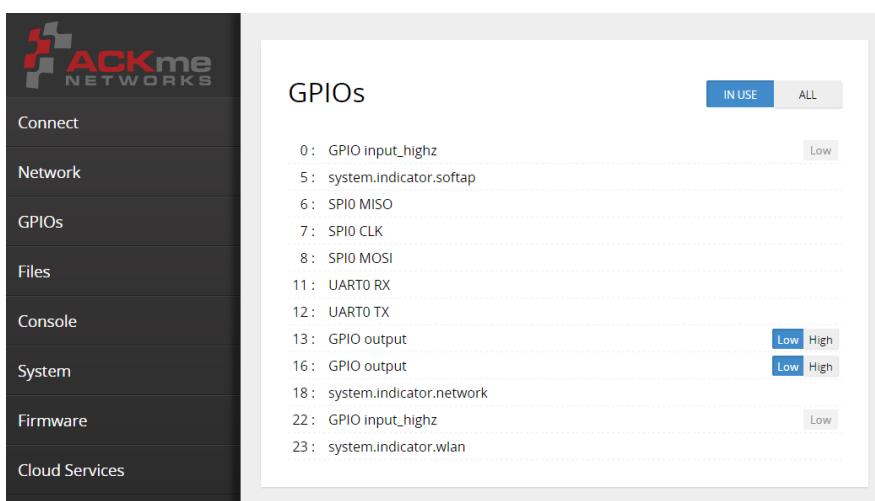


The screenshot shows the 'Files' tab interface. On the left is a sidebar with navigation links: Connect, Network, GPIOs, Files (which is selected and highlighted in blue), Console, System, Firmware, and Cloud Services. The main content area has a title 'Files'. It features a central panel with a dashed border containing the text 'Drop files here' and a button 'Click to add files'. Below this is a table listing files with their sizes and delete icons:

File	Size	Action
command_help.csv	43731 bytes	trash
default_setup.script	135 bytes	trash
favicon.ico.gz	1897 bytes	trash
geotrust_ca.pem	1236 bytes	trash
ghm_capabilities.json	1765 bytes	trash
ghm_capabilities_setup_solo.csv	2973 bytes	trash
therm_celsius_lut.csv	330 bytes	trash
webapp/index.html	2109 bytes	trash
webapp/unauthorized.html	9643 bytes	trash
webapp/wiconnect.css.gz	39125 bytes	trash
webapp/wiconnect.js.gz	68117 bytes	trash
wifi_fw.bin	210412 bytes	trash

GPIO Configuration Tab

The GPIO configuration tab lists the configuration details for GPIOs in use. This is equivalent to the output of the `get gpio.usage` command. For input GPIOs, the current state is listed. For output GPIOs, buttons are provided to set the state to high or low.

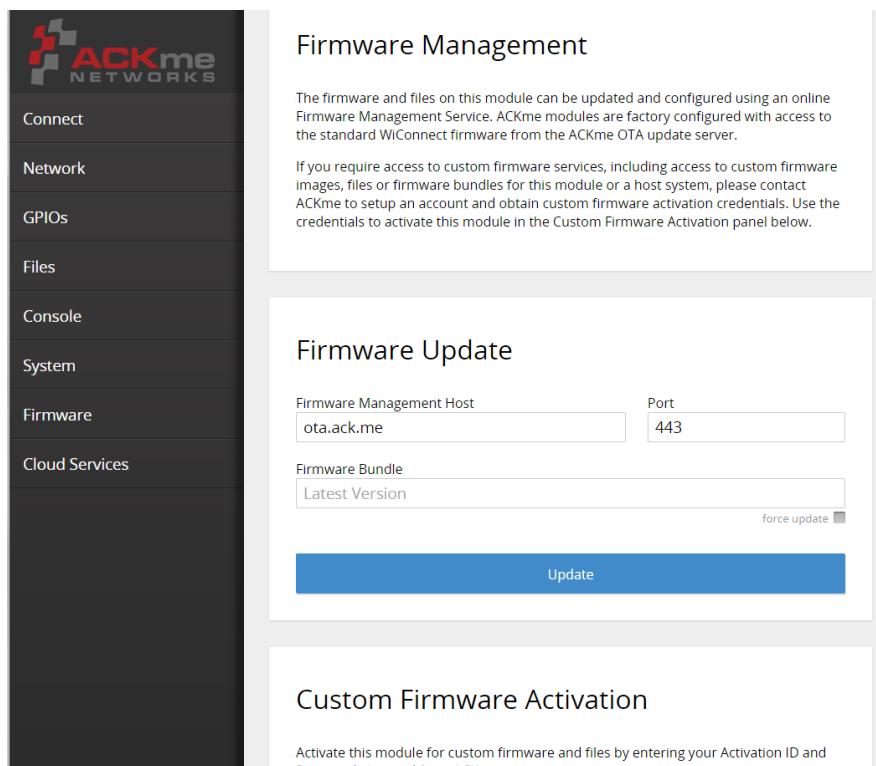


The screenshot shows the 'GPIOs' tab interface. On the left is a sidebar with navigation links: Connect, Network, GPIOs (selected and highlighted in blue), Files, Console, System, Firmware, and Cloud Services. The main content area has a title 'GPIOs' with two buttons above it: 'IN USE' (blue) and 'ALL' (grey). The table below lists GPIO pins with their names and current states:

GPIO	Value
0: GPIO input_highz	Low
5: system.indicator.softap	Low
6: SPI0 MISO	Low
7: SPI0 CLK	Low
8: SPI0 MOSI	Low
11: UART0 RX	Low
12: UART0 TX	Low
13: GPIO output	Low
16: GPIO output	Low
18: system.indicator.network	Low
22: GPIO input_highz	Low
23: system.indicator.wlan	Low

Firmware Management

The firmware management tab allows Over-the-Air (OTA) wireless firmware updates. This is the equivalent of issuing the **ota** command. This tab is available only with Internet access.



The screenshot shows the 'Firmware Management' section of the WiConnect Web App. On the left, there is a vertical navigation menu with the following items: Connect, Network, GPIOs, Files, Console, System, Firmware, and Cloud Services. The 'Firmware' item is currently selected and highlighted in white. The main content area has three sections: 'Firmware Management', 'Firmware Update', and 'Custom Firmware Activation'. The 'Firmware Management' section contains a note about updating files and a link to the standard WiConnect firmware from the ACKme OTA update server. The 'Firmware Update' section contains fields for 'Firmware Management Host' (set to 'ota.ack.me') and 'Port' (set to '443'), a 'Firmware Bundle' field (set to 'Latest Version'), and a 'force update' checkbox. A large blue 'Update' button is at the bottom. The 'Custom Firmware Activation' section contains a note about activating the module for custom firmware and files by entering an Activation ID and Password obtained from ACKme.

Firmware Management

The firmware and files on this module can be updated and configured using an online Firmware Management Service. ACKme modules are factory configured with access to the standard WiConnect firmware from the ACKme OTA update server.

If you require access to custom firmware services, including access to custom firmware images, files or firmware bundles for this module or a host system, please contact ACKme to setup an account and obtain custom firmware activation credentials. Use the credentials to activate this module in the Custom Firmware Activation panel below.

Firmware Update

Firmware Management Host: ota.ack.me Port: 443

Firmware Bundle: Latest Version

force update

Update

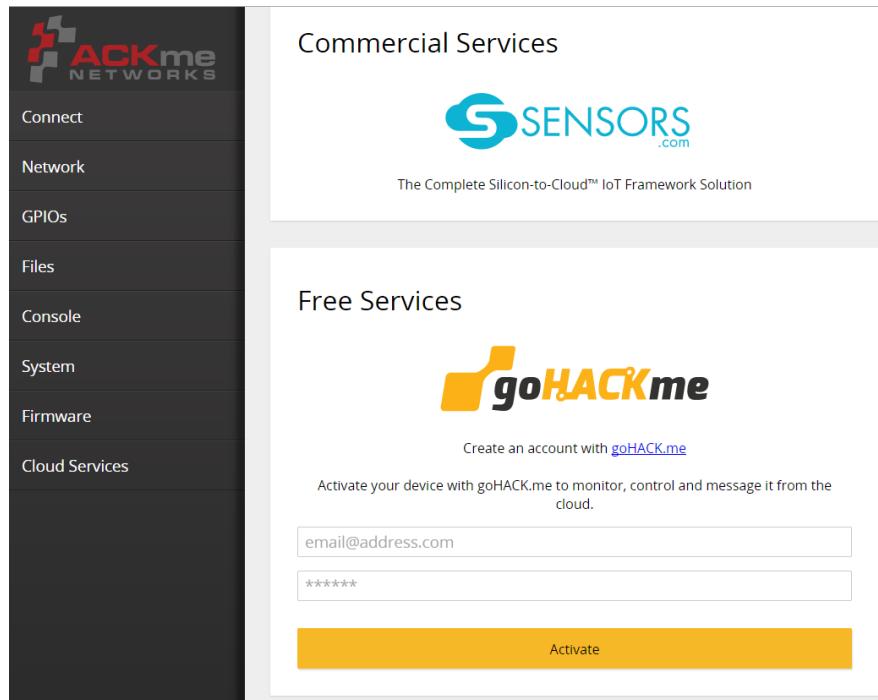
Custom Firmware Activation

Activate this module for custom firmware and files by entering your Activation ID and Password obtained from ACKme.

Cloud Services

The Cloud Services tab provides links and configuration settings related to the Sensors.com commerical cloud service and the goHACK.me free cloud service. This tab is available only with Internet access.

A link is provided to create your free account with goHACK.me. Once you have an account, you can use the activation form on this tab to activate your module with goHACK.me.



Commercial Services

SENSORS.com

The Complete Silicon-to-Cloud™ IoT Framework Solution

Free Services

goHACKme

Create an account with [goHACK.me](#)

Activate your device with goHACK.me to monitor, control and message it from the cloud.

Activate

Customizing the WiConnect Web App

You can create your own modified version of the WiConnect Web App.

The customizable version contains all the features listed above with the exception of the Console tab.

It is available free at: <https://github.com/ackme/WiConnectWebApp>

Detailed development and build instructions are provided.

Serial Interface

WiConnect provides a flexible serial interface. It supports various serial protocols and modes, and is easy to use for both humans and machines alike. WiConnect supports two serial protocols, UART and SPI.

UART Protocol

A single or dual UART is available for use by the host MCU. If a host can support two UARTs, one UART may be used for control and data communication, and the other used for debug logging. If a single UART is available, control, data and debug logs (if enabled) are interleaved.

SPI Protocol

SPI provides a higher speed communications interface for higher bandwidth applications such as audio and video. Full support for a serial-SPI communications interface will be added in an upcoming release of WiConnect.

Interface Modes

The WiConnect serial interface may be used in any of three different modes.

- **Command Mode** provides an asynchronous command interface that a host may use to send and receive control and data information. Command mode is typically used by a host to configure WiConnect, however it may also be used by simple hosts that need ultimate master/slave control over information sent to, and received from the module.
 - **Stream Mode** provides a streaming interface that transparently connects a WiConnect serial interface (UART or SPI) with a network stream such as a UDP or TCP client or server. Stream mode provides a simple 1-1 connection between a physical serial interface and a network stream. Bytes or characters sent from the host to a serial interface are transparently pushed by WiConnect to a network stream via a wireless interface. Conversely, bytes or characters received by a network stream (from a remote server) via a wireless interface are transparently pushed by WiConnect to a serial interface connected to the host. A [wireless serial port](#) is a typical application that uses stream mode.
 - **Packet Mode** Packet mode enables complex connections between multiple serial interfaces and network streams.
-

Command Mode

Command mode is the primary mode used to configure WiConnect. Understanding how command mode works, and the options provided, is the key to mastering the use of WiConnect. The serial interface is designed to cater for humans and machines alike, and the interface is configurable to suit the needs of both.

Configuration

In command mode, WiConnect and the ACKme module effectively provides a slave interface to the host. The host is the master and WiConnect is the slave. The host initiates all transactions which follow the format of “host issues a command, WiConnect provides a response”. WiConnect [commands](#) are used by the host to read and write WiConnect [variables](#), send control information, and send and receive data across network connections.

The variable [system.cmd.mode](#) is provided for convenience to make it easy to switch between human mode and machine mode. Setting [system.cmd.mode](#) conveniently changes the value of each of the following variables to configure the desired command

mode.

Human Friendly Command Mode

By default, the serial interface is configured to be human friendly with the following settings. See the [system variable documentation](#).

```
set system.print_level all      -> Turn on all debug & informational prints
set system.cmd.header_enabled 0  -> Disable a response header
set system.cmd.prompt_enabled 1  -> Turn on the user prompt
set system.cmd.echo on          -> Turn on echo to see what you're typing
```

Machine Friendly Command Mode

To configure the serial interface for machine friendly operation, use the following settings. Detailed information about these settings is available in the [system variable documentation](#).

```
set system.print_level 0        -> Turn off all debug & informational prints
set system.cmd.header_enabled 1  -> Enable a response header (described below)
set system.cmd.prompt_enabled 0  -> Turn off the user prompt
set system.cmd.echo off         -> Turn off character echo
```

Command Format

Commands sent to WiConnect are formatted as shown in the following table. The <command name> is a WiConnect command issued by the host, and [variable name <argument>]\r\n is an (optional) WiConnect variable name and accompanying argument terminated by a carriage return and newline character.

```
<command name> [variable name <argument>]\r\n
```

Response Format

Responses from WiConnect follow the format shown in the text below.

```
RXYYYYY\r\n
<response data>
```

where ...

- R denotes a Response header. **NOTE!** If the module is operating in [Safe Mode](#), R is replaced with S.
- X is the error code (response error codes are listed below).
- YYYYY is the response data length in bytes including a trailing \r\n. The response data length is zero if no data is available, or >2 (including \r\n) if data is available.
- <response data>. If the response data length is >0, the <response data> is the data returned from WiConnect in response to the command.

Response Error Codes

0 = Success
1 = Command failed
2 = Parse error
3 = Unknown command
4 = Too few args
5 = Too many args
6 = Unknown variable or option
7 = Invalid argument
8 = Serial command line buffer overflow

Log Format

If `system.print_level` is greater than 0, a log header together with one or more informational debug logs may be returned in addition to a response header and response data when a command is issued. Log headers follow the format shown in the table below.

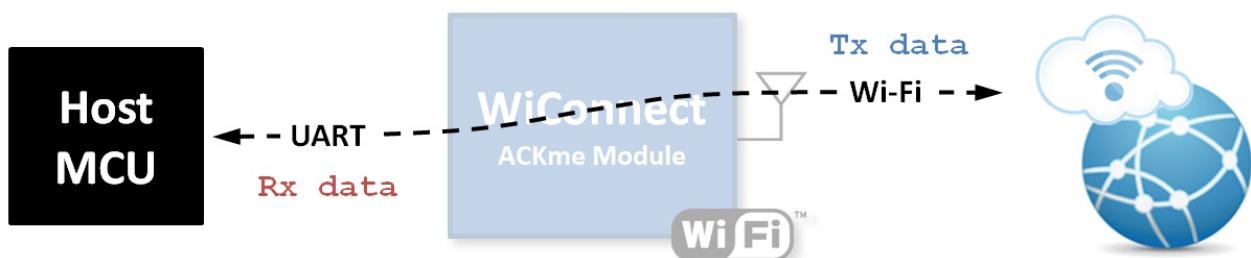
LXXXXXX\r\n<log message>

where ...

- L denotes a Log header.
- X is the log level.
- XXXXX is the log message length in bytes including a trailing \r\n. Log messages are always >2 bytes in length and including terminating \r\ncharacters.
- <log message>. The log message returned by WiConnect in response to the command.

Stream Mode

As depicted in the following diagram, stream mode provides a transparent connection between a serial port and a network stream. It is perfect for implementing applications such as a [wireless serial port](#).



Stream mode is used primarily by host applications that can handle asynchronous data (with the optional use of hardware flow control). It is perfectly suited to applications that only need to use a single network stream at any one time.

Configuration

The simplest way to setup an ACKme module to use Stream mode is to do all the WiConnect configuration first using [command mode](#), then reboot into stream mode. Configuration involves just a few simple steps:

1. Setup a wireless and network interface to use with stream mode e.g. wlan interface and TCP client. This typically requires setting up the wireless interface and network stream to auto-connect, or enable connection using a GPIO control pin.
2. Set the serial `bus.mode` to `stream`.
3. Then [save](#) and [reboot](#) the module.

Both the wlan and softap interfaces have an auto-start on bootup capability using the `wlan.auto_join.enabled` and `softap.auto_start` variables. The TCP server has auto-start capability using the `tcp_server.auto_start` variable. Network clients can be started using a GPIO or manually by breaking out of stream mode as described below.

Breaking out of Stream Mode

When the serial bus is in Stream mode, commands are disabled and every character sent to the serial port is automatically forwarded to a network stream. There are two ways to break out of stream mode and enable command mode entry temporarily. The first method is to send a character sequence, configurable with the `bus.stream.cmd_seq`, after 3 seconds of inactivity on the serial interface. By default, the stream break out sequence is `$$$`. The other option available to break out of stream mode is to use a GPIO configurable with the `bus.stream.cmd_gpio` variable. When you're done entering commands, return to stream mode by simply issuing the [exit](#) command.

Packet Mode

Packet mode, which is planned for an upcoming release of WiConnect, will provide the option to connect one or more serial interfaces to one or more network streams for ultimate data multiplexing flexibility!

Wi-Fi Interface

There are two Wi-Fi interfaces available for use with WiConnect, a wlan client (802.11 STA) interface and a softAP (802.11 AP) interface. Both interfaces share the same physical Wi-Fi radio and antennas. To set the Wi-Fi transmit power for both interfaces, use the common `wlan.tx_power` variable. Likewise, to set the antenna used by both interfaces, use the common `wlan.antenna` variable.

WLAN Client Interface

The wlan client interface enables WiConnect to join with a Wi-Fi access point at home, work or in-between. It is possible to use the wlan and softap interfaces concurrently, however these interfaces share the Wi-Fi radio and hence must operate on the same Wi-Fi radio channel in concurrent mode. The wlan interface has priority, and if the wlan interface starts on a different channel to the softap interface, the softap channel switches to the wlan channel.

WiConnect provides various commands and variables to configure the wlan interface. See the documentation for [wlan commands](#) and [wlan variables](#). An overview of WLAN functionality is provided in the following sections.

Scanning for Networks

There are two methods to scan for networks in range, active scanning and passive scanning. Active scanning involves the Wi-Fi module actively transmitting 802.11 probe request packets. When a Wi-Fi Access Point receives a probe, it replies with an 802.11 probe response. Passive scanning on the other hand avoids actively transmitting, and the Wi-Fi module simply listens for Wi-Fi Access Point beacons for a period of time on each channel scanned.

Each of the parameters discussed in the preceding paragraph (plus more) are configurable with WiConnect. To specify the channels to scan, use the `wlan.scan.channel_mask` variable. The scan type (active or passive) can be set using the `wlan.scan.type` variable. The time spent listening on each channel, in active and passive scan mode respectively, is configurable using the `wlan.scan.active_dwell` and `wlan.scan.passive_dwell` variables. In active mode, the number of 802.11 probe requests is configurable using `wlan.scan.num_probes`.

And finally, to initiate a scan for Wi-Fi networks in range, use the `wlan_scan` command.

Joining a Network

To join a network manually, set the `wlan.ssid` and `wlan.passkey` variables to match the network name and network password the module is attempting to join. If your application is sensitive to latency or power, you should also consider customizing the number of join retry attempts with the `wlan.join.retries`. WiConnect supports all SOHO security types including Open, WEP, WPA & WPA2 together with mixed modes. If you are planning to use WEP (and we sincerely hope you are not since **WEP IS NOT SECURE**), please read the instructions in [wlan security](#).

To hide the passkey after it has been set, simply set the `wlan.hide_passkey` variable. After the passkey is hidden, it is no longer possible to read the passkey until a factory reset is performed. This stops an attacker learning the network password by physically acquiring a product with an ACKme module running WiConnect.

If the application requires the module to auto-join a network after boot up, set the `wlan.auto_join.enabled` variable. The number of automatic retry attempts and the delay between automatic retries is configurable with the variables `wlan.auto_join.retries` and `wlan.auto_join.retry_delay`.

Once the module is joined to a Wi-Fi network, the received signal strength (signal level) of the Wi-Fi Access Point is available with the [wlan_get_rssi](#) command.

Wi-Fi Roaming

If the signal strength of the Wi-Fi Access Point the module is joined with drops too low, the module attempts to scan for (and roam to) another known network in range. The received signal level at which roaming is initiated is configured using the [wlan.roam.threshold](#) variable. While the module is roaming on other radio channels, it may miss packets on the home channel. Depending on your application, it may be necessary to modify one or more of the [wlan scan](#) variables discussed in the [Scanning for Networks](#) section to minimize packet loss during roaming.

Access Point (SoftAP) Interface

The softAP interface enables a Wi-Fi client such as a notebook, smartphone or tablet to join with WiConnect. The softap interface is primarily controlled with the [softap variables](#).

SoftAP features include ...

- Auto start on reboot
- Wi-Fi radio channel selection
- Control of interface settings such as IP address range, DHCP server, DNS redirect server
- Options to adjust AP name, password and URL
- Combined use with a web server for provisioning of the wlan interface
- Plus more ...

Networking and Security

WiConnect includes a full IPv4 networking stack and SSL/TLS security suite that supports a range of popular networking protocols including TCP, UDP, DNS, DHCP. Additional network application libraries are provided for native HTTP/S and secure cloud access using goHACK.me & the Sensors.com API.

Network features include ...

- [DHCP client and server](#)
- [TCP/UDP/TLS client and server](#)
- [HTTP & HTTPS library](#)
- [HTTP Server with RESTful API & Websockets](#)
- [Network status indication using GPIOs](#)
- [Network Connections and Streams](#)
- [Remote Terminal Access](#)
- [Network Discovery including mDNS, LLMNR & Netbios](#)
- [Broadcast status announcement](#)
- [Network time management](#)
- [Sending email using SMTP](#)
- [Sensors.com API](#)

Security features include ...

- [SSL/TLS client and server](#)
 - [HTTPS webserver](#)
 - [HTTPS file upload and download](#)
 - [CA certificate store](#)
-

DHCP Client and Server

DHCP Server

The WiConnect DHCP server supplies IP addresses to clients connecting to the soft AP when the network is brought up with the softap interface.

See command:

- [network_up](#)

See variables:

- [softap.auto_start](#)
- [softap.channel](#)
- [softap.client_list](#)
- [softap.dhcp_server](#)

DHCP Client

The WiConnect DHCP client obtains an IP address from the wlan DHCP server when the network is brought up with the wlan interface.

See command:

- [network_up](#)

See [wlan variables](#)

TCP, UDP and TLS Client and Server

See commands:

- [tcp_client](#)
- [tcp_server](#)
- [tls_client](#)
- [tls_server](#)
- [udp_client](#)
- [udp_server](#)

and:

- [TCP client variables](#)
- [TCP keepalive variables](#)
- [TCP server variables](#)
- [UDP client variables](#)
- [UDP server variables](#)

TCP Client Auto Connect

See variables:

- [tcp.client.auto_interface](#)
- [tcp.client.auto_retries](#)
- [tcp.client.auto_start](#)
- [tcp.client.remote_host](#)
- [tcp.client.remote_port](#)

UDP Client Auto Connect

See variables:

- [udp.client.auto_interface](#)
- [udp.client.auto_start](#)
- [udp.client.remote_host](#)
- [udp.client.remote_port](#)

UDP Server Auto Connect

See variables:

- [udp.server.auto_interface](#)
- [udp.server.auto_start](#)
- [udp.server.remote_host](#)
- [udp.server.remote_port](#)

HTTP Library

HTTP Post Example

The following (fictitious) HTTP post example shows how to post data to an HTTP web server using the WiConnect HTTP API. The HTTP body data posted in this example is a small piece of JSON (sent using `stream_write 0 7`). Since the `-o` option is used with the `http_post` command, a connection stream to the HTTP server is opened, but the HTTP post is queued locally on the module.

Queuing the HTTP post locally provides the ability to add HTTP headers using the `http_add_header` command, and to post data in the HTTP body using the `stream_write` command. Once all headers and body data are queued, the HTTP post is sent to the server and completed using the `http_post` command.

Any response data received from the server may be read using `stream_read`.

```
> http_post -o example.com/hello application/json
[2014-04-23 | 19:40:23: Opening: example.com]
Request POST /hello
Connecting (HTTP): example.com:80
[2014-04-23 | 19:40:23: Opened: 0]
0
> stream_write 0 7
... JSON data goes here ...
Success
> http_read 0
HTTP response: 200
Chunked response
200
> stream_read 0 1000
{
  "response": "howdy!"
}
> stream_close 0
Closing: 0
[2014-04-23 | 19:40:40: Closed: 0]
Success
```

HTTP Server with RESTful API

The WiConnect HTTP webserver may be configured to run as a service on either the softAP or wlan interface. The server supports [HTTP Basic Authentication](#) with (or without) HTTPS security.

WiConnect provides a simple RESTful API on top of the HTTP server. The API allows for a remote HTTP(S) client to issue any WiConnect command. The result of the command is returned in a simple JSON format.

The HTTP Server is configured with the following variables:

- `http.server.api_enabled`
- `http.server.cors_origin`
- `http.server.denied_filename`
- `http.server.enabled`
- `http.server.interface`
- `http.server.max_clients`
- `http.server.notfound_filename`
- `http.server.password`
- `http.server.port`
- `http.server.root_filename`
- `http.server.tls_cert`
- `http.server.tls_enabled`
- `http.server.tls_key`
- `http.server.tls_verify_peer`
- `http.server.username`

A client can use the RESTful API to issue WiConnect commands and receive responses, and also for retrieval of module log messages.

The available requests are as follows:

```
GET /command  
POST /command  
GET /log  
POST /stream
```

The WiConnect Web App provides a complete demonstration of the HTTP Server RESTful API, and can be customized as required. See [Customizing the WiConnect Web App](#).

Command Request/Response

GET Request

The API supports either a simple GET request:

```
GET /command/<wiconnect command>
```

POST Request

Or a slightly more complex POST request:

```
POST /command
{
    "flags"    : <flags>,
    "command"  : "<wiconnect command>",
    "data"      : "<command data>"
}
```

Note: The HTTP server does not check headers.

where:

- <flags>
 - 0x01 - “command” field is base64 encoded
 - 0x02 - base64 encode response data
 - 0x04 - “data” field is base64 encoded
- <wiconnect command> - any wiconnect serial command
- <command data> - optional, specific to certain wiconnect commands that require additional data (write, file_create, etc)

HTTP Response Codes

- 200 - the command transaction executed successfully
- 400 - malformed request
- 500 - server error

Response body

```
{
    "id"        : <unique id>,
    "code"      : <response code>,
    "flags"     : <flags>,
    "response"  : "<command response>"
}
```

where:

- <unique id> - unique id given to each command.
- <response code> - the wiconnect command response code.
- <flags>
 - 0x01 - the command response is base64 encoded
- <command response> - the command response data

Log Request/Response

The API also buffers log messages. This is the request to retrieve the log messages.

GET Request

```
GET /log
```

HTTP Response Codes

- 200 - the command transaction executed successfully
- 400 - malformed request

Response

```
{  
  "logs" : [ "<log data>", "<log data>", .... ] }  
}
```

Note: the log buffer has limited space. Older logs are replaced by newer ones. This should be called periodically to avoid losing logs.

Web Socket Stream

Issuing the API call POST /stream opens a websocket and a 'stream' is created by WiConnect. An MCU can read/write data using the WiConnect [stream_read](#) and [stream_write](#) commands.

```
POST /stream  
<raw data .... >
```

See [HTTP Server Simple WebSocket Demonstration](#).

HTTP Response Codes

- 200 - the command transaction executed successfully
- 400 - malformed request
- 404 - no available stream handles

HTTP Server Security and Authorization

The HTTP server is secured using [HTTP Basic Authentication](#). This requires that a client supplies a username and password. Note that the username/password are sent in the HTTP request header which is encrypted only if the HTTP request itself is encrypted.

Client Authorization

When the http server username/password settings are set, authorization is required to access certain files/api commands. Client authorization uses 'HTTP Basic Authentication'.

Client authorization requires a username and password that matches the [http.server.username](#) and [http.server.password](#) variables. If authorization fails, the server returns a 401 error code and the following response header:

```
WWW-Authenticate: Basic realm=WiConnect
```

The authorization feature is enabled when both the `http.server.username` and `http.server.password` variables are set.

Protecting or Securing a File

When client authorization is enabled, authorization is required to download all files (except unprotected files). To unprotect a file, the `-u (unprotected)` flag must be explicitly specified when the file is created using the `file_create` or `http_download` commands.

When client authorization is disabled, all files may be downloaded from the WiConnect web server.

Securing the REST API, Whitelisting API Calls

When the authorization feature is enabled all REST API calls require authorization. It is possible to 'whitelist' certain API calls. This is done by creating the file: `http_whitelist.csv` which contains a comma separated list of REST API calls that do not require authorization. This file also supports a trailing wildcard character *. Some examples of this file are as follows:

Whitelist ALL REST API calls

*

Whitelist the log and stream API calls:

/log,/stream

Whitelist all get wlan calls:

get wl*

Whitelist the help variables and help commands commands:

help variables,help commands

Whitelist /stream, help, and all getters:

/stream,help *,get *

CORS (Cross Origin Resource Sharing)

The WiConnect HTTP server supports [CORS \(Cross-Origin Resource Sharing\)](#).

The `http.server.cors_origin` variable allows you to specify origins for which the [same-origin policy](#) is relaxed.

This allows control of the module, via the HTTP server, from a remote site provided the module has originally been set up with a `http.server.cors_origin` domain that allows access from that site. Via the WiConnect HTTP server [WiConnect JavaScript API](#), the remote site can issue all WiConnect commands, including `reboot`.

Setting the `http.server.cors_origin` results in the WiConnect HTTP server inserting a corresponding CORS Access-Control-Allow-Origin (ACAO) response header into resources it delivers. It also results in the WiConnect HTTP server responding to an OPTION request with a set of options supporting remote control.

HTTP Server Security Variables

- `http.server.username`
 - `http.server.password`
 - `http.server.cors_origin`
-

Network Status Indication Using GPIOs

See [Peripherals, System Indicator Functions](#).

See variables `system.indicator.gpio` and `system.indicator.state`.

Network Connections and Streams

A maximum of 8 network connections is allowed.

All stream types in the table below use one stream, with the following exceptions and qualifications:

- External UDP clients connecting to a WiConnect UDP server do not use an additional stream. That is, a single UDP server takes only one stream and allows an unlimited number of UDP clients to be connected.
- The TCP server uses no streams, but each external TCP or TLS client connecting to a WiConnect TCP or TLS server uses an additional stream.

When a stream is open, it is assigned a handle number. This handle is used to read/write/poll/close the stream. WiConnect supports many types of streams.

The currently opened streams may be listed with the [stream_list](#) command. Streams are listed with stream number, type, and information about the stream source.

Note that some stream types only support certain operations. The following is a list of stream types, and for each type the command to create it, and the operations it supports.

Type	Command	Operations	Description
UDPC	udp_client	read/write/poll/close	UDP client
UDPS	udp_server	read/write/poll/close	UDP server. Note that one stream is used for all server clients
TCPC	tcp_client	read/write/poll/close	TCP client
TCPS	tcp_server	read/write/poll/close	TCP server client stream. Note that one stream is used *per* client
TLSC	tls_client	read/write/poll/close	TLS client
TLSS	tls_server	read/write/poll/close	TLS server client stream. Note that one stream is used *per* client
HTTP	http_get / http_post / http_head	read/write/poll/close	HTTP client
HTTPS	http_get / http_post / http_head	read/write/poll/close	Secure HTTP client
WEBS	-	read/write/poll/close	WebSocket opened by a network client
FILE	file_open	read/poll/close	File system handle
GHM	ghm_message get /ghm_message post	read/write/poll/close	goHACK.me message
CMD	buffered commands	read/poll/close	Buffered command data. Select commands that are buffered when the variable system.cmd.buffered = '1'

Refer to the following stream commands to use the stream handle:

- [stream_list](#)
- [stream_read](#)
- [stream_write](#)

- [stream_close](#)
 - [stream_poll](#)
-

Remote Terminal Access

See the [Wi-Fi Remote Terminal application note](#) and the [Remote Terminal variables](#).

Network Discovery

Overview

In simple terms, network discovery is used to give the module a name on the local network. So, for instance, if the module has a domain name `mymodule.local`, a remote client on the same network can connect to the module using the domain `mymodule.local` even though the domain `mymodule.local` is not registered with a DNS server. This is useful because the remote client doesn't need to know the IP address of the module.

WiConnect supports three network discovery protocols: mDNS (multicast Domain Name System), LLMNR (Link-Local Multicast Name Resolution) and NetBIOS. The latter two protocols are used by Windows systems only.

OS Support

The remote client must support one or more network discovery protocols for device discovery to work.

Apple

Mac OS X & iOS support mDNS by default, with Bonjour.

Windows

Windows has support for mDNS, but by default it uses the LLMNR and NetBIOS protocols. These protocols are very similar to mDNS so their basic domain resolution features are supported by the module as well. When a domain is entered into the web browser, Windows follows this sequence to resolve the domain:

1. Check local cache
2. Issue LLMNR query
3. Issue NetBIOS query
4. Issue DNS query

The LLMNR and NetBIOS queries are broadcast to the local network only. The DNS query is typically sent to the internet.

If network discovery is enabled on the module, the module will receive the LLMNR and NetBIOS queries, compare the query domain to its `mdns.name`, and if they match respond with the module's IP address. In this way the module can easily be found on the local network from a Windows machine.

A NetBIOS domain can have up to 15 characters (including the '.local'). If `mdns.name` is longer than 15 characters then the NetBIOS protocol is not used.

Finally, some ISPs **hijack** the `.local` domain for their own purposes, including advertising. This may cause a Windows PC to fail

to resolve the domain. The solution (for Windows only) is to drop `.local` from the URL and simply use <http://mymodule/> directly.

Linux

Linux needs an additional package installed for mDNS support. The most common package is Avahi.

Android

Unfortunately, Android does not support these (or apparently any ZeroConf protocol) by default. It recently added a mDNS SDK but most apps don't appear to use it. Consequently third party apps like Chrome on Android do not support mDNS but adding support to a custom app should be straight-forward.

In a Nutshell

Network discovery provides a way for remote clients to resolve the IP address of the module on a local network using a standard protocol. mDNS has other features which allow the module to advertise various services including HTTP, TCP & UDP servers. WiConnect supports configuration of these additional features with the `mdns.service` variable.

Commands

- [mdns_discover](#)

Variables

- [mdns.enabled](#)
- [mdns.interface](#)
- [mdns.name](#)
- [mdns.service](#)
- [mdns.ttl](#)

Broadcast Status Announcement

ACKme WiConnect modules broadcast module properties in JSON format. The properties can be sent either as UDP packets to a UDP host or by a post request to an HTTP host. Properties include by default the IP address and the MAC address.

See [Broadcast](#) variables:

- [broadcast.data](#)
- [broadcast.http.host](#)
- [broadcast.interface](#)
- [broadcast.interval](#)
- [broadcast.udp.ip](#)
- [broadcast.udp.port](#)

See also the [Broadcast UDP Packet Application Note](#).

Network Time Management

ACKme WiConnect modules can obtain time data from an NTP (Network Time Protocol) server.

See [NTP variables](#):

- [ntp.enabled](#)
 - [ntp.interval](#)
 - [ntp.server](#)
-

Sending Email using SMTP

ACKme WiConnect modules can send email messages via an external SMTP server.

See [email SMTP variables](#):

- [email.name_address](#)
 - [email.smtp.host](#)
 - [email.smtp.password](#)
 - [email.smtp.port](#)
 - [email.smtp.username](#)
-

Sensors.com API

The goHACK.me web service provides a free introduction to the Sensors.com services.

See [goHACK.me API](#).

See the [goHACKme commands](#) and [goHACKme variables](#).

Peripherals

Peripheral commands and variables enable access to general purpose input/outputs (GPIOs), analog functions such as analog-digital converters (ADCs) and PWMs (Pulse Width Modulators). In the case of the ACKme Numbat and Mackeral evaluation boards, peripherals include two user LEDs, two user buttons and the thermistor ADC.

GPIO Commands and Variables

Typically, particular GPIOs are hard-wired to components such as LEDs, buttons and sensors. Other GPIOs are configurable for input and output.

GPIOs can be used for special system functions, including (but not limited to):

- assigning a LED as wlan status indicator
- selecting between serial bus modes: COMMAND or STREAM
- pulse width modulator (PWM) function
- general standard IO function
- out-of-band interrupts

GPIO commands and variables allow control of the function and direction of the GPIOs.

GPIO Commands

- `adc_take_sample` - Take a sample of an ADC pin via the connected GPIO
- `dac_set_level` - Set the output level of a DAC (if available)
- `gpio_dir` - Set GPIO direction, as a Standard I/O
- `gpio_get` - Get GPIO value, for a GPIO functioning as Standard I/O
- `gpio_set` - Set GPIO value, for a GPIO functioning as Standard I/O
- `pwm_update` - Assign a PWM to a GPIO, with a specified duty cycle and frequency

GPIO Variables

- `broadcast.udp.data` - Specify GPIOs and ADCs for which state is to be broadcast in UDP data packet
- `bus.stream.cmd_gpio` - Set and get the GPIO used for switching from stream mode to command mode
- `ghm.solo gpio` - Assign a GPIO to a goHACK.me stream or control parameter in solo mode.
- `gpio_alias` - Set and get the alias of a GPIO
- `gpio.config_file` - Set and get the file used for GPIO configuration on bootup
- `gpio.init` - Set and get GPIO initial direction and value
- `gpio.usage` - Get list of GPIO usage
- `ioconn.control_gpio` - Set and get the GPIO used for GPIO controlled network connection (stream)
- `ioconn.status_gpio` - Set and get the GPIO used to indicate when GPIO controlled network feature is enabled (stream)
- `network.status_gpio` - Set and get the GPIO used for network status
- `setup gpio.cmd` - Set and get the command executed when setup gpio.control_gpio asserted
- `setup gpio.control_gpio` - Set and get the GPIO used for initiating a command or web setup sequence
- `setup gpio.level` - Set and get the active logic level that indicates setup gpio.control_gpio is asserted
- `setup gpio.mode` - Set the execution mode for setup gpio.cmd
- `system.activity gpio` - Set and get the GPIO used to indicate specific types of system activity.
- `system.activity gpio_level` - Set and get the logic level corresponding to activity.

- **system.bflash.cs_gpio** - Set and get the bulk flash chip select GPIO
- **system.indicator gpio** - Set and get the GPIO used for a system indicator function
- **system.indicator.state** - Set and get the behavior of the GPIO corresponding to each indicator state
- **system.wakeup.events** - Set and get GPIOs assigned to wake the module from sleep. Available only for GPIOs with wake capability
- **system.oob variables** - Set a GPIO for out-of-band interrupts in response to specified events.
- **tcp.server.connected_gpio** - Set and get the GPIO used to indicate whether a client is connected to the TCP server.
- **tcp.server.data_gpio** - Set and get the GPIO used to indicate whether a connected TCP client has data available to read.
- **udp.server.data_gpio** - Set and get the GPIO used to indicate whether a connected UDP client has data available to read.

GPIO Functions and Pins

GPIO functions pins differ depending on module and evaluation board. For a comparison of modules, see [Module Comparison](#).

Wallaby Module and Mackerel Evaluation Board

The table below shows functions, defaults and pins for the AMW004 Wallaby GPIOs. The table also shows defaults, connections and header pins for the AMW004-E03.3 Mackerel evaluation board.

Wallaby AMW004							Mackerel AMW004-E03.3		
GPIO	Default	ADC	PWM	DAC	Wake	Pin	Default	Connections	Header Pin
GPIO0			pwm3	dac0		7	Standard I/O, factory reset	Button 1	H1-4
GPIO1						8	system.indicator gpio wlan	LED G (Green)	H1-3
GPIO2						9	system.indicator gpio network	LED Y (Yellow)	H1-6
GPIO3		adc4			wake	11	none		H1-8
GPIO4		adc5				12	none		H1-7
GPIO5			pwm0		wake	13	system.indicator gpio softap	LED R (Red)	H1-10
GPIO6			pwm1		wake	14	none		H1-9
GPIO7		adc0	pwm3			15	none	Thermistor	H1-12
GPIO8		adc1				17	none		H1-11
GPIO9		adc2	pwm0		wake	18	none		H1-14
GPIO10			pwm3		wake	19	none		H1-13
GPIO11			pwm2		wake	20	Standard I/O, wake, web setup	Button 2	H1-16
GPIO12		adc3	pwm1		wake	21	none		H1-15
GPIO13	uart1.rx	adc6				23	uart1.rx		UART-2
GPIO14	uart1.tx	adc7				24	uart1.tx		UART-1
GPIO15	uart1.rts		pwm1			26	uart1.rts		UART-3

GPIO16	uart1.cts		pwm2			27	uart1.cts			UART-4
GPIO17	spi0.clk			wake	30	spi0.clk				H2-13
GPIO18	spi0.mosi				31	spi0.mosi				H2-14
GPIO19	spi0.miso				32	spi0.miso				H2-11
GPIO20		pwm0		wake	33					H2-12
GPIO21		adc8	pwm0		34	Standard I/O	User LED2			H2-9
GPIO22		adc9			35	Standard I/O	User LED1			H2-10
GPIO23					36					H2-7
GPIO24				wake	37					H2-8
GPIO25	uart0.tx				38	uart0.tx				H2-5
GPIO26	uart0.rx			wake	39	uart0.rx				H2-6
GPIO27				wake	41					H2-3
GPIO28					42					H2-4

Mackerel power header pins:

- GND: H1-2, H1-19, H2-2, H2-19
- VDD: H1-1, H1-20, H2-1, H2-20

Numbat Module and Moray Evaluation Board

The table below shows functions, defaults and pins for the AMW006 Numbat GPIOs. The table also shows defaults, connections and header pins for the AMW006-E03.2 Moray evaluation board.

Numbat AMW006						Moray AMW006-E03.2				
GPIO	Default	ADC	PWM	Wake	Pin	Default	Alias	Connections	Header Pin	
GPIO0	spi1.mosi				4	Standard I/O, factory reset	button_1	Button 1		H6-4
GPIO1					5	none				
GPIO2	spi1.clk*				6	none				
GPIO3	spi1.cs*				7	none				
GPIO4		N/A			8	none				
GPIO5	spi1.mosi*		pwm5		13	system.indicator.gpio softap	softap_led	LED R (Red)		H8-13
GPIO6	spi0.miso	adc6			14	spi0.miso				H8-14
GPIO7	spi0.clk	adc5			15	spi0.clk				H8-15
GPIO8	spi0.mosi	adc7			16	spi0.mosi				H8-16

GPIO9	uart0.rts				17	none				UART-3
GPIO10	uart0.cts		pwm0		18	none				UART-4
GPIO11	uart0.rx		pwm0		19	uart0.rx				UART-2
GPIO12	uart0.tx		pwm0		20	uart0.tx				UART-1
GPIO13			pwm1		21	Standard I/O	led2	User LED2		H8-21
GPIO14			pwm3		22	none				H8-22
GPIO15			pwm3		23	none				H8-23
GPIO16			pwm6		24	Standard I/O	led1	User LED1		H8-24
GPIO17	uart1.rx	adc3	pwm4		25	SPI CLK				H8-25
GPIO18					28	system.indicator gpio network	network_led	LED Y (Yellow)		H7-28
GPIO19	uart1.tx	adc2	pwm4		29	SPI MISO				H7-29
GPIO20		adc9	pwm2		30	none		Thermistor		H7-30
GPIO21		adc8	pwm2		31	none				H7-31
GPIO22	uart1.cts	adc0	pwm1	wake	32	Standard I/O, wake, web setup	button2	Button 2		H7-32
GPIO23	uart1.rts	adc1	pwm4		33	system.indicator gpio wlan	wlan_led	LED G (Green)		H7-33

* Numbat V4+ only

Moray power header pins:

- GND: H6-2, H6-11, H7-27, H7-36, H8-12, H8-35
- VDD: H6-3, H6-10, H7-35

Viewing GPIO Usage

To see the current GPIO usage, read the `gpio.usage` variable.

The default Mackerel GPIO usage is:

```
get gp u
! # Description
# 0 GPIO input_highz
# 1 system.indicator.wlan
# 2 system.indicator.network
# 5 system.indicator.softap
# 11 setup.control_gpio
```

```
# 13 UART1 RX
# 14 UART1 TX
# 17 SPI0 CLK
# 18 SPI0 MOSI
# 19 SPI0 MISO
# 21 GPIO output
# 22 GPIO output
```

The default Moray GPIO usage is:

```
> get gp u
!
! # Description
# 0 GPIO input_highz (button1)
# 5 system.indicator.softap (softap_led)
# 6 SPI MISO
# 7 SPI CLK
# 8 SPI MOSI
# 11 UART0 RX
# 12 UART0 TX
# 13 GPIO output (led2)
# 16 GPIO output (led1)
# 18 system.indicator.network (network_led)
# 22 Standard I/O (button2)
# 23 system.indicator.wlan (wlan_led)
```

The [gpio_usage](#) command does not list GPIOs set to none: these GPIOs have no function configured.

Note that the aliases shown in the Moray GPIO usage list above are set in the `gpio_config_init.csv` file provided with the Moray evaluation board. See [Configuration and Setup, GPIO Configuration](#).

Setting GPIO Function

GPIO function can be set by commands or variables, or by a configuration file (see [Configuration and Setup, GPIO Configuration](#)).

If there is a `gpio_config_init.csv` file present on the file system, this overrides any other GPIO settings at boot time. Some GPIO function changes are not implemented until reboot, and cannot be implemented by WiConnect command if a `gpio_config_init.csv` file sets the GPIO function.

GPIOs that are already assigned to a function need to be deregistered before reassignment. If the GPIO is assigned to a `system.indicator gpio` function, deregister or reassign the `system.indicator gpio` function. See [System Indicator Functions](#). Otherwise use the `gpio_dir` command with a none argument.

For example, to deregister user LED1 on the Moray board (GPIO13):

```
gdi 13 none
```

For the change to persist after reboot, you need to set the `gpio.init` variable and save:

```
set gpio.init <GPIO#> none
```

```
save
```

To use a GPIO as Standard I/O, use the `gpio_dir` command to set its direction. For example, to set user LED1 on the Moray to an output:

```
gdi 13 out
```

For the change to persist after reboot, you need to set the `gpio.init` variable and save. For example:

```
set gpio.init 13 out
save
```

To use a GPIO with a PWM, use the `pwm_update` command.

You can also use a GPIO for a number of special functions, such as indicators, or inputs to control special features.

System Indicator Functions

Use the `system.indicator gpio` and `system.indicator state` to configure a GPIO for a system indicator function. Save the variables and reboot to complete the configuration change, since these functions are managed by services configured during bootup.

Note: This GPIO function change cannot be implemented by WiConnect command if a `gpio_config_init.csv` file sets the GPIO function.

On the Mackerel and Moray evaluation boards, the red, yellow and green LEDs are configured by default as system indicators, showing soft AP, network and wlan status. See [Wallaby Module and Mackerel Evaluation Board](#), [Numbat Module and Moray Evaluation Board](#).

For example, to switch the network system indicator function to Mackerel user LED 1 (GPIO 22):

```
gdi 22 none
set sy i g network 22
save
reboot
```

Each system indicator function can be associated with only one GPIO. Setting the indicator to a GPIO automatically sets to none any GPIO previously assigned to that system indicator.

To deregister a GPIO currently registered for a system indicator function, either disable the system indicator function or set it to another GPIO. For example, to deregister GPIO 1 on the Mackerel:

```
set sy i g wlan -1
```

Bus Stream Command GPIO

Set the `bus.stream.cmd_gpio` to configure a GPIO to force WiConnect from stream mode to command mode. See [Serial Interface](#) for a discussion of stream mode and command mode.

This feature is intended for automated use with a host MCU. In the following demonstration of this feature on the Mackerel evaluation board, we use Button 2 (GPIO 11).

WiConnect commands	Comments
gdi 11 none	Deregister Mackerel Button 2 GPIO
set bu s g 11	Configure Button 2 GPIO as bus.stream.cmd_gpio
set bu m stream	Set the bus mode to stream
save	Save variables
reboot	The changed bus mode is not implemented until boot

Because the device is in stream mode, characters typed into the serial terminal are not echoed. Press Button 2, and WiConnect displays:

Command Mode Start

While holding down Button 2, WiConnect commands may be typed into the terminal, and characters are echoed. For example, you can set bus mode to command and save, to restore command mode after a reboot. Release Button 2 and the device returns to stream mode.

GPIO Controlled Network Connection

WiConnect allows you to configure full details of a connection to a host, then connect and disconnect under the control of a single GPIO. The connection can be via UDP or TCP, with optional TLS.

Set `ioconn.control_gpio` to assign a GPIO to the control function that makes and breaks the connection.

Set `ioconn.status_gpio` to assign a GPIO to indicate the status of the connection.

See the `ioconn.*` variables for details.

The [GPIO-Controlled Network Connection](#) application note provides further examples.

GPIO Controlled Script

Network Status GPIO

`network.status_gpio` enables you to set and get the GPIO used to indicate network status.

TCP Server Status GPIOs

`tcp.server.connected_gpio` provides a GPIO to indicate whether a client is connected to the TCP server.

`tcp.server.data_gpio` provides a GPIO to indicate whether a connected client has data available to read.

See the [TCP Server + softAP](#) application note for further information.

Out-of-Band Interrupts

The `system.oob variables` allow a GPIO to be assigned for out-of-band (OOB) interrupts in response to specified events.

The OOB GPIO is asserted whenever one or more of the configured events occurs. It is de-asserted when reading the status register.

See in particular `system.oob gpio` for details.

NOTE: The OOB gpio feature can be used by itself. The corresponding special function GPIO does not need to be enabled. For example, the `network.status` OOB event can be triggered without setting the `network.status_gpio` variable.

Using a GPIO Alias

You can set a [GPIO alias](#) for any GPIO. The alias may be used instead of the GPIO number, since an alias is easier to remember. Using aliases can also help to make code more consistent and easier to read.

The default settings for the Moray board configure the alias `led1` for GPIO13, so you can use the alias instead of the GPIO number:

```
gdi led1 none
```

To create an alias for a GPIO, set the `gpio.alias` variable. For example, to set an alias for the Mackerel board user LED1:

```
set gp a 22 led1
```

Now the same command works on both the Moray and Mackerel boards, even though GPIO assignments differ:

```
gdi led1 none
```

Setting and Getting GPIO Values

If a GPIO is set to the Standard I/O function, and the GPIO direction is set as an output with the gdi ([gpio_direction](#)) command, the GPIO value may be configured with the gse ([gpio_set](#)) and gge ([gpio_get](#)) commands. For example:

```
> gdi 12 out
Set OK
> gge 12
0
> gse 12 1
Set OK
> gge 12
1
```

If a Standard I/O GPIO is configured for input, you can read the GPIO value but you cannot set it:

```
> gdi 12 in
Set OK
> gge 12
0
> gse 12 1
GPIO not configured for output
Command failed
```

ADCs

To read the value from an ADC, use the [adc](#) command. For example, to read the value for the Moray thermistor ADC (GPIO20):

```
> adc 20
0x7D7
```

The ADC value can be read regardless of the GPIO function setting.

To convert the ADC_value to mV, use the following formula:

```
adc_mV = (Vref * adc_value) / adc_max_value
```

Vref is nominally 3300 (3.3V) and adc_max_value is 0xFFFF (hex) = 4095 (dec).

You can specify the ADC reference voltage with the [system.adc.vref](#) variable.

For example, using the nominal Vref the ADC value 0x7D7 converts to 1617 mV. This thermistor reading corresponds to a temperature of about 26 degrees Celsius when using the thermistor on the Mackerel or Moray evaluation board.

To determine temperature from the Mackerel and Moray thermistor ADC readings, see [Temperature versus ADC value](#).

You can also send ADC values in broadcast UDP packets. See [Broadcast UDP Packet](#).

ADC Lookup Tables

A LookUp Table (LUT) can be specified for an ADC using the <LUT filename.csv> parameter with the [adc_take_sample](#) command. A lookup table can also be specified when setting a GPIO for an ADC function using the [ghm.solo gpio](#) variable.

The ADC is read and the raw value is used to find the two closest entries in the LUT. Those two entries are used to linearly interpolate the corresponding converted value of the ADC.

The LUT is parsed and loaded into RAM, either the first time a LUT file parameter is supplied with the [adc_take_sample](#) command, or when the [ghm.solo gpio](#) setting is specified. For every subsequent read of that ADC with a lookup table specified, the cached LUT is used for the conversion, unless the [adc_take_sample](#) is issued with a -v (voltage) parameter. Only one LUT for a given ADC may be used. A reboot is required to load another LUT for the same ADC.

Care should be taken when generating the LUT as it is cached in the heap, i.e. the larger it is, the more RAM it takes away from other features.

Up to 3 ADC GPIOs may use a LUT.

The LUT is a CSV file with the following format:

```
<Raw ADC Value (0-4095)>,<Floating point number>\r\n
```

The table entries must follow these requirements:

- Must contain at least three entries
- Entries must be ordered with <Raw ADC Value> in ascending order (i.e. the smallest ADC value is the first entry in the table, the largest ADC value is the last entry)
- The first entry MUST be the <Raw ADC Value> of 0
- The last entry MUST be the <Raw ADC Value> of 4095

For evaluation boards with a thermistor, such as the Mackerel and Moray boards, a thermistor LUT is downloaded automatically to the module when you issue the [ghm_capabilities download -s](#) command in the course of a [goHACK.me](#) solo setup.

PWMs

PWMs (Pulse Width Modulators) can be used for blinking and dimming user LEDs. They can also be used for driving a speaker attached to a GPIO.

The [pwm_update](#) command provides control of the frequency and duty cycle of a PWM output.

Before using with a PWM, a GPIO must be set to the none function with [gpio_dir](#).

PWMs are associated with specific GPIOs. In the GPIO tables on this page ([GPIO Functions and Pins](#)), each PWM has an associated group ID. Only one PWM GPIO in each group may be simultaneously and independently active. The GPIO numbers, **NOT** the PWM ID numbers, are used in the [pwm_update](#) command.

PWM groups, maximum and minimum frequencies are module specific as shown in the tables below:

Wallaby PWMs

Number of independent PWMs (groups)	4
minimum PWM frequency	5Hz
Maximum frequency	1.2MHz

PWM Group ID	GPIOs
pwm0	5, 9, 20, 22
pwm1	6, 12, 15
pwm2	11, 16
pwm3	0, 7, 10

Numbat PWMs

Number of independent PWMs (groups)	7
minimum PWM frequency	1Hz
Maximum frequency	21MHz

Numbat PWM Group ID	GPIOs
pwm0	10, 11, 12
pwm1	13, 22
pwm2	20, 21
pwm3	14, 15
pwm4	17, 19, 23
pwm5	5
pwm6	16

LEDs

The level of a LED can be controlled via a GPIO, either with the Standard I/O output or by using a PWM.

GPIO Standard I/O Control

A LED can be turned on or off by configuring as a Standard I/O output, then setting the GPIO level.

For example, to turn on Mackerel user LED1, use the `gpio_dir` command and the `gpio_set` command:

WiConnect commands	Comments
<code>gdi 22 none</code>	Deregister Mackerel User LED1 GPIO, in case it is assigned to a different function
<code>gdi 22 out</code>	Assign User LED1 GPIO to a Standard I/O output
<code>gse 22 1</code>	Set GPIO output high (LED1 on)
<code>gse 22 0</code>	Set GPIO output low (LED1 off)

PWM control - Blinking and Dimming LEDs

You can use a PWM to blink an LED. A PWM can be used with a GPIO only when no function is assigned to the GPIO. For example, to blink Moray user LED1, use the `gpio_dir` command and the `pwm_update` command:

WiConnect commands	Comments
<code>gdi 16 none</code> <code>pwm 16 50 1</code>	Deregister Moray User LED1 GPIO, in case it is assigned to a different function Run the GPIO PWM at duty cycle 50%, frequency 1Hz

Using `pwm_update` on a GPIO assigns the corresponding PWM function to the GPIO. You can see this with the `gpio_usage` variable:

```
> get gp u
! # Description
...
# 16 PWM6 (led1)
...
```

To turn off blinking, use the `pwm_update` command with the `stop` argument:

```
pwm 16 stop
```

To dim the LED, reduce the duty cycle. To make blinking imperceptible, set the frequency to a rate above 30 Hz. For example, to dim the Moray user LED2:

WiConnect commands	Comments
<code>gdi 16 none</code> <code>pwm 16 5 60</code>	Deregister Moray User LED1 GPIO, in case it is assigned to a different function Run the GPIO PWM at duty cycle 5%, frequency 60Hz

Communicating Peripheral Information Over the Network

The WiConnect TCP and UDP client and server features provide frameworks for building a customised application for transmitting and receiving monitor and control data from your ACKme module. For examples of these applications, see:

- [TCP Client](#)
- [Multiple TCP Clients](#)
- [Secure TLS Client](#)
- [GPIO-Controlled Network Connection](#)
- [TCP Server + SoftAP](#)

WiConnect also provides general systems for handling information transfer.

A WiConnect module can be configured to broadcast GPIO and ADC data at regular intervals, via the `broadcast` group of variables.

The `broadcast.udp.data` variable lets you specify what data is broadcast. GPIOs are specified by GPIO number, and ADCs are specified by ADC number. See the [Broadcast UDP Packet](#) for a demonstration of this feature.

The [goHACK.me](#) system provides a complete solution to remote monitoring and control of your ACKme device, including free web servers, a powerful and easy to use online interface, and a WiConnect API. See:

- [goHACKme](#)
- [goHACK.me Slave Mode](#) application note
- [goHACK.me Solo Mode](#) application note

Peripheral GPIO Mapping by Function

Wallaby GPIOs by Function

Wallaby UART

Name	GPIO
uart0.tx	25
uart0.rx	26
uart0.cts	N/A
uart0.rts	N/A
uart1.tx	14
uart1.rx	13
uart1.cts	16
uart1.rts	15

Wallaby ADC

Name	GPIO
adc0	7
adc1	8
adc2	9
adc3	12
adc4	3
adc5	4
adc6	13
adc7	14
adc8	21
adc9	22

Wallaby PWM

Name	GPIOs
pwm0	5, 9, 20, 21
pwm1	6, 12, 15
pwm2	11, 16
pwm3	0, 7, 10

Wallaby SPI Master (System Use Only)

Name	GPIO
spi0.clk	17
spi0.mosi	18
spi0.miso	19

Wallaby DAC

Name	GPIO
dac0	0

Numbat GPIOs by Function**Numbat UART**

Name	GPIO
uart0.tx	12
uart0.rx	11
uart0.cts	10
uart0.rts	9
uart1.tx	19
uart1.rx	17
uart1.cts	22
uart1.rts	23

Numbat ADC

Name	GPIO
adc0	22
adc1	23
adc2	19
adc3	17
adc4	N/A
adc5	7
adc6	6
adc7	8
adc8	21
adc9	20

Numbat PWM

Name	GPIOs
pwm0	10, 11, 12
pwm1	13, 22
pwm2	20, 21
pwm3	14, 15
pwm4	17, 19, 23
pwm5	5
pwm6	16

Numbat SPI Master (System Use Only)

Name	GPIO
spi0.clk	7
spi0.mosi	8
spi0.miso	6

Numbat SPI Slave (Numbat v4+ Only)

Name	GPIO
spi1.clk	2
spi1.mosi	5
spi1.miso	0
spi1.cs	3
spi1.irq	optional
spi1.rdy	optional

Temperature versus ADC value

The temperature table below shows temperature versus thermistor ADC value (hex and dec) with gain=1, and ADC mv for the thermistor and ADC on the Mackerel AMW004-E03.x and the Moray AMW006-E02.x. See [Peripherals](#), [ADCs](#). The table assumes the nominal Vref of 3300mV.

The conversion to mV uses the formula

$$\text{adc_mV} = (\text{Vref} * \text{adc_value}) / \text{adc_max_value}$$

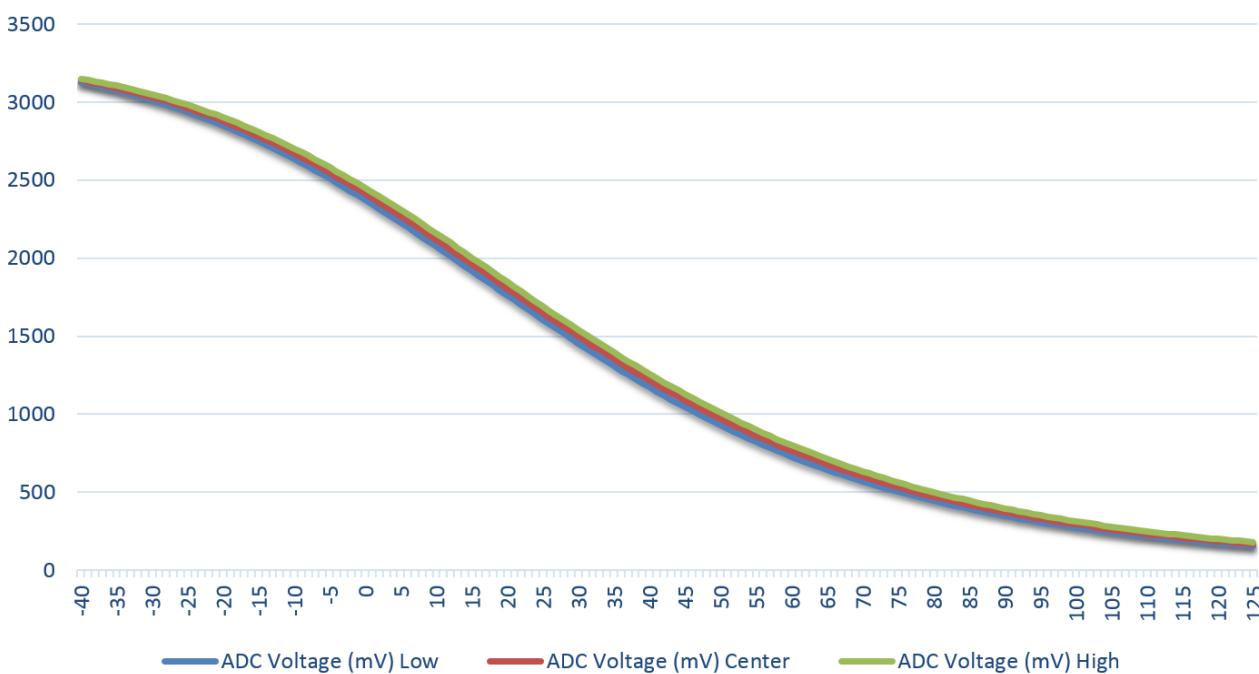
This table is accurate within a few degrees celsius. For full thermistor characteristics search for [MuRata part NCP18XH103J03RB](#).

Note: Adjust as necessary if Vref is not equal to 3300mV. Different thermistors may have a different correspondence between voltage and temperature. To create an accurate table for your particular thermistor you can calibrate with an accurate thermometer.

ADC value hex	ADC value dec	ADC mv	Temp (deg. C)	Temp (deg. F)
0xBB2	2994	2413	0	32.0
0xB8F	2960	2385	1	33.8
0xB6C	2925	2357	2	35.6
0xB48	2889	2328	3	37.4
0xB24	2853	2299	4	39.2
0xAF	2816	2269	5	41.0
0xADB	2780	2240	6	42.8
0xAB6	2742	2210	7	44.6
0xA8F	2704	2179	8	46.4
0xA6A	2667	2149	9	48.2
0xA44	2628	2118	10	50.0
0xA1D	2590	2087	11	51.8
0x9F7	2551	2056	12	53.6
0x9D0	2513	2025	13	55.4
0x9AA	2474	1994	14	57.2
0x983	2436	1963	15	59.0
0x95C	2396	1931	16	60.8
0x935	2358	1900	17	62.6
0x90E	2318	1868	18	64.4
0x8E7	2280	1837	19	66.2
0x8BF	2240	1805	20	68.0

0x899	2201	1774	21	69.8
0x872	2163	1743	22	71.6
0x84C	2124	1712	23	73.4
0x825	2086	1681	24	75.2
0x7FF	2048	1650	25	77.0
0x7D9	2009	1619	26	78.8
0x7B3	1972	1589	27	80.6
0x78D	1933	1558	28	82.4
0x768	1896	1528	29	84.2
0x742	1859	1498	30	86.0
0x71E	1823	1469	31	87.8
0x6F9	1786	1439	32	89.6
0x6D5	1750	1410	33	91.4
0x6B1	1714	1381	34	93.2
0x68E	1679	1353	35	95.0
0x66C	1644	1325	36	96.8
0x649	1609	1297	37	98.6
0x627	1576	1270	38	100.4
0x606	1542	1243	39	102.2
0x5E4	1509	1216	40	104.0
0x5C3	1475	1189	41	105.8
0x5A3	1443	1163	42	107.6
0x584	1412	1138	43	109.4
0x565	1381	1113	44	111.2
0x546	1350	1088	45	113.0

Thermistor ADC (mV) versus Temperature (C)



Module Comparison

Currently available ACKme Wi-Fi modules are:

- AMW004 Wallaby module - see [ACKme site product details](#)
- AMW006 Numbat module - see [ACKme site product details](#)

The AMW004 Wallaby module has a faster microprocessor, has more RAM and flash, has more GPIOs and has a DAC.

The AMW006 Numbat module is smaller, consumes less power, has more independent PWMs and has a SPI slave interface.

Module differences affect GPIO configuration and the running of memory intensive features. See [Peripherals](#) and [Memory](#).

Feature	AMW004 - Wallaby	AMW006 - Numbat
Total RAM	128Kb	96Kb
CPU Speed	120MHz	80MHz
Internal flash (NOT user writable)	1MB	512Kb
Extended (serial) flash (user writable)	1MB	1MB
Power consumption (3.3V supply)	1.85µA to 390mA	0.97µA to 360mA
Max RF Tx Output Power	+18.5 dBm @ 802.11b (EVM < -9 dB) +13.5 dBm @ 802.11n MCS7 (EVM < -28 dB)	+17.5 dBm @ 802.11b (EVM < -9 dB) +13.5 dBm @ 802.11n MCS7 (EVM < -28 dB)
Size	31.8 x 17.8 x 2.7 mm (1.25" x 0.70" x 0.11")	20.3 x 15.2 x 2.7mm (0.80" x 0.60" x 0.11")
GPIOs	29	21
DACs	1 x 12bit	-
ADCs	10 x 12bit	9 x 12bit
Independent simultaneous PWMs	4	7
4-Wire UART interfaces	2	2
SPI interfaces	1 x SPI-master interface	2 x SPI interfaces (1 master, 1 slave)
Wake Inputs	12 x edge/level sensitive wake inputs	1 x ultra-low power wake input

File System

Most connected applications require the ability to store configuration information or cache local data. WiConnect provides a reliable read/write filesystem and on-board extended flash storage to satisfy application storage requirements. Bulk flash, external to the module, can also be used. See [Internal, Extended and Bulk Flash](#).

With just a few simple commands, WiConnect provides your application with complete access to local and network connected storage.

Commands are available to [create, delete and open files](#), and file contents can be easily accessed using one or more [stream commands](#).

Here's a quick example showing how to create, manipulate then delete a file:

WiConnect Commands	Description
file_create hello.txt 12 Hello World!	<- Create file, length 12 bytes <- Enter file contents (echoed if system.cmd.echo = 1)
file_open hello.txt stream_read 0 3	<- Open file for reading <- Read no more than 3 bytes (file stream handle = 0)
stream_read 0 50	<- Read no more than 50 bytes (close file if EOF reached)
file_delete hello.txt	<- Delete file

File Listing

Most ACKme Wi-Fi modules contain a microprocessor with internal flash and an extended (on module) serial flash. An additional (optional) bulk serial flash is also supported. See [Internal, Extended and Bulk Flash](#).

To obtain a verbose listing of all files on the MCU-internal flash, extended (and bulk) serial flash, use the [ls command](#) with the `-v` option. Users are only permitted to write files to serial flash.

```
> ls -v
! # Type Flags Hnd Size Version Filename
# 0 e-FE 0001 73 1853 2.0.0.11 /favicon.ico.gz
# 1 e-FE 0001 68 18067 2.0.0.11 /setup/images.png
# 2 e-FE 0001 52 10525 2.0.0.11 /setup/index.css.gz
# 3 e-FE 0001 65 10155 2.0.0.11 /setup/index.html
# 4 e-FE 0001 55 39247 2.0.0.11 /setup/index.js.gz
# 5 e-FB 0001 74 36511 2.0.1.8 command_help.csv
# 6 e-FD 0001 51 135 2.0.0.11 default_setup.script
# 7 e-03 0001 50 1236 2.0.0.11 geotrust_ca.pem
# 8 e-FE 0021 84 2074 1.0.0.0 my_ca.pem
# 9 i-00 001B 0 212736 2.0.1.8 upgrade_app.exe
# 10 i-81 001B 52 178252 2.0.1.8 wiconnect.exe
```

```
# 11 e-01 0009 0 203261 5.26.230.3 wifi_fw.bin
# 12 b-FE 0021 25 1995261 2.1.0.0 DSC20564.JPG
```

File Types

The file type of files on the MCU internal flash is pre-pended with 'i', likewise 'e' is used for files on extended serial flash (inside the module), and 'b' is used for files on bulk serial flash (external to the module). WiConnect file types are enumerated in the following table.

File type	ID
UPGRADE_APP	0x00
WIFI_FW	0x01
SHARED_LIB	0x02
TLS_CERT	0x03
TXT_LOG	0x04
DCT	0x05
MISC_APP	0x80
WICONNECT_APP	0x81
TEMPORARY	0xF9
GPIO_CONFIG	0xFA
COMMAND_HELP	0xFB
SDC_CAPS	0xFC
SETUP_SCRIPT	0xFD
MISC_FIX_LEN	0xFE
TYPE_INVALID	0xFF

File Checksum

The checksum used on individual files is calculated using a CCITT CRC-16 algorithm with polynomial 0x1021. The full set of CRC parameters is as follows:

Parameter	Value
CRC Order	16
CRC Polynomial	0x1021
Initial Value	FFFF (direct)
Final XOR	0x0
Reverse data bytes	Yes
Reverse CRC	No

See the [crc_util.py](#) python script and the [crccitt_test.c](#) C source for more details of calculating the CRC.

Special Files

Some files with special functions are listed in the following table:

File Name	Description	Notes
default_setup.script	Runs on issuing setup cmd	See Configuration and Setup, Setup Configuration Script
default_config.csv	Variable configuration, loaded after a successful OTA	See Configuration and Setup, Variable Configuration Script
gpio_config_init.csv	Bootup GPIO configuration	See Configuration and Setup, Using a GPIO Configuration File
ghm_capabilities.json	goHACKme capabilities (caps) file	See GoHACKme, Device Capabilities

Writing Files to the File System

There are two WiConnect commands that can write to the file system:

- [file_create](#)
- [http_download](#)

There are several ways to invoke these commands:

- Via the [WiConnect Web App](#). This is the simplest way to manage files manually.
- Manually, via a WiConnect terminal or a remote terminal
- Under programmed MCU host control
- Via the [HTTP Server RESTful API](#)

Writing with the WiConnect Web App File Browser

When you open the WiConnect Web App in a web browser and select the [Files tab](#), you can upload files to the module file system using click and browse, or drag and drop.

In the background, the WiConnect Web App uses the [file_create](#) command to write files to the module file system, using the [HTTP Server RESTful API](#).

There are several ways to activate the WiConnect Web App. See [WiConnect Web App](#).

Writing with a WiConnect Terminal or Remote Terminal

Use the `file_create` command to create the file. Immediately after issuing the command, type the file contents, or alternatively copy and paste the file contents into the terminal after issuing the `fcr` command. The length of the file must be supplied. The character count includes any line termination characters.

In the example below we create a script that can be run using the command `setup cmd -v test.script`. In this case the character count is 35.

```
> fcr test.script 35
help,setup,\r\n#This is a comment

File created
Success
```

Writing File Contents to a Stream in Chunks

You can write the file in chunks by leaving the file open after issuing the `file_create` with the `-o` option. Write the file chunks to the stream returned by the `file_create` command. You must know the total size of the file to be created in advance.

In the example below a small file is created in two chunks:

Commands and Responses	Description
<pre>> fcr -o hello.txt 11 [Opened: 0] 0 > write 0 6 Hello Success > write 0 5 World File created [Closed: 0] Success > fop hello.txt [Opened: 0] 0 > read 0 1000 Hello World [Closed: 0]</pre>	<p>Create file; leave stream open; specify total length</p> <p>Response: open stream</p> <p>Write first chunk; specify chunk length</p> <p>Send chunk content (includes trailing space)</p> <p>Write second chunk; specify chunk length</p> <p>Send chunk content</p> <p>Stream closed</p> <p>Open new file</p> <p>Response: open stream</p> <p>Read contents</p> <p>Stream closed</p>

HTTP Download

You can also provide the file for download from a web server accessible to the device, and use the `http_download` command. For example:

```
> http_download http://www.google.com.au/images/srpr/logo11w.png test1.png
Downloading: test1.png to flash file system
Request GET /images/srpr/logo11w.png
Connecting (http): www.google.com.au:80
HTTP response: 200
Success
> ls
! #  Size  Version  Filename
...
# 4 14022      1.0.0  test1.png
...
```

OTA

The **OTA** system allows you to upgrade the WiConnect system files automatically from the ACKme OTA servers.

Reading Files from the File System

There are two WiConnect commands that can write to the file system:

- `file_open`
- `http_upload`

There are several ways to invoke these commands:

- Via the **WiConnect Web App**. This is the simplest way to manage files manually.
- Manually, via a WiConnect terminal or a remote terminal
- Under programmed MCU host control
- Via the **HTTP Server RESTful API**

Reading with the WiConnect Web App File Browser

When you open the WiConnect Web App in a web browser and select the **Files tab**, you can download a file from the module file system by clicking the file name.

In the background, the WiConnect Web App uses the `file_open` command to read the files from the module file system, using the **HTTP Server RESTful API**.

There are several ways to activate the WiConnect Web App. See **WiConnect Web App**.

Reading from a WiConnect Terminal or Remote Terminal

You can read text files directly from the WiConnect Terminal. Read the file, using a `file_open` followed by a `stream_read` command, specifying the stream index returned from the file open, e.g.:

```
> fop default_setup.script
[Opened: 0]
0
> read 0 1000
network_up,-s      ,Configuration network credentials
set wlan.auto_join.enabled,true,Enable network auto-join
save,-,Saving settings

[Closed: 0]
```

HTTP Upload

You can upload the file to an available web server with file upload capability with the `http_upload` command.

Internal, Extended and Bulk Flash

A WiConnect module contains **internal flash** in the microprocessor and an **extended flash** chip for user storage.

Extended flash is serial flash.

In addition, an external serial flash can be connected to the module using the existing sflash GPIOs. This is referred to as **bulk flash**. To enable access to bulk flash, configure the chip select GPIO with the `system.bflash.cs_gpio` variable.

By default WiConnect can support up to 128MB bulk flash.

Internal flash cannot be manipulated by WiConnect commands.

All OTA files and config files (created by the `save` command) are always stored on extended flash regardless of whether the bulk flash is enabled.

The following commands operate on bulk flash if available, otherwise on extended flash:

- `http_download` - saves file to bulk flash if available, otherwise to extended flash
- `file_create` - saves file to bulk flash if available, otherwise to extended flash
- `file_open` - opens file if found from bulk flash, otherwise from extended flash
- `file_delete` - deletes file if found from bulk flash, otherwise from extended flash

Extended and bulk flash can be formatted with the `format_flash` command.

Memory

WiConnect requires memory (RAM) as working space to hold intermediate calculations and to buffer serial and network data. The amount of memory available in a module impacts the number of features that can be run concurrently, as well as the maximum throughput that is achievable. Modules with more RAM support more features and achieve higher throughput. ACKme offers modules with differing amounts of RAM as shown in the follow table.

Part Number	Name	RAM	CPU Speed
AMW004	Wallaby	128 kB	120 MHz
AMW006	Numbat	96 kB	80 MHz

For a more detailed module comparison, see [Module Comparison](#).

Memory Management

WiConnect features require varying amounts of RAM. Features like the HTTP Server, TLS server and [goHACK.me](#) solo mode are memory intensive. In some cases it may not be possible to run memory intensive features simultaneously.

For example, it is not possible to run simultaneously two TLS sessions, or two HTTPS sessions. On a Numbat module, it is not possible to run the HTTP server and simultaneously activate the module for goHACK.me.

Some websites use large TLS certificate chains that may exceed the RAM capacity of the module resulting in a failed connection.

When memory usage is greater than 90%, the system displays a warning every 30 seconds:

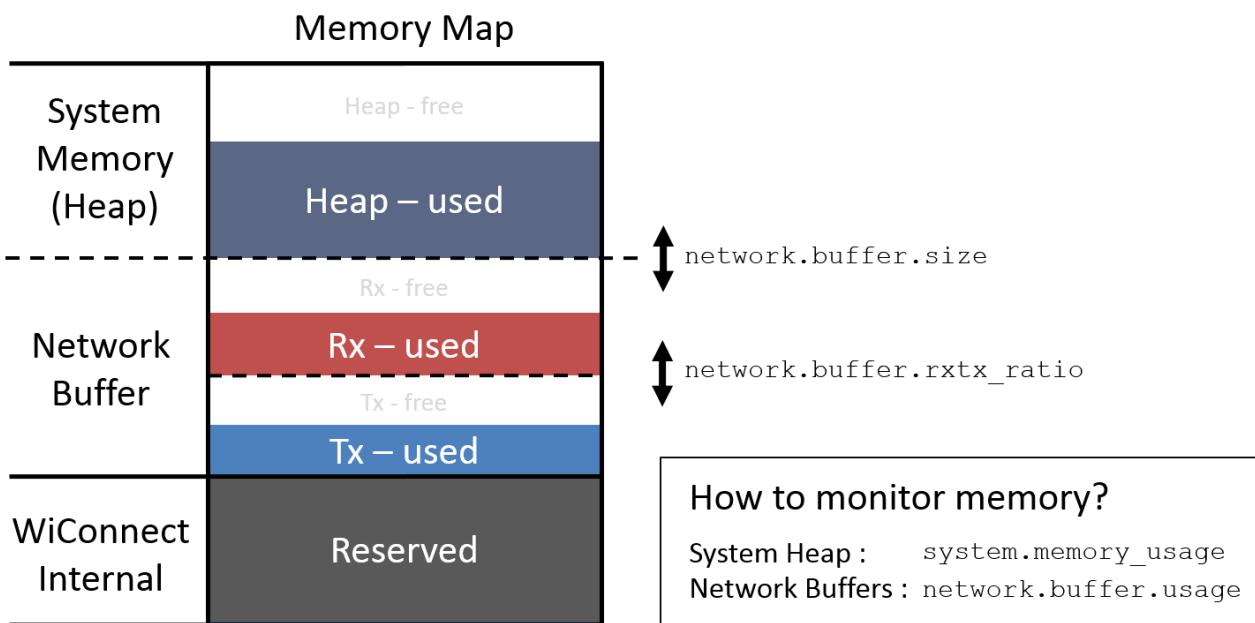
WARN: Low memory, system may become unstable

Determining Memory Usage

Applications use heap memory. To see the percentage of available heap currently allocated, use the `system.memory_usage` variable:

```
get system.memory_usage
```

The diagram below shows a high level map of module memory allocation.



The percentage memory usage returned by `system.memory_usage` is calculated as follows:

`system.memory_usage = 100*heap_used/heap_size`

where

`heap_size = total_memory_size - reserved_size - network.buffer.size`

Notes:

- Adjusting `network.buffer.size` alters the total heap size available.
- Adjusting `network.buffer.rxtx_ratio` changes the proportion of the network buffer available for Rx and Tx. It does not change network buffer size.

Minimizing Memory Usage

To minimize memory usage, disable services that are not in use and set default memory allocations to the minimum.

Default features that consume memory include:

- NTP client - see [NTP variables](#)
- Broadcast of properties - see [Broadcast variables](#)
- RSSI averaging - see [wlan.rssi_average](#)
- Network buffer size set above minimum value - see [network.buffer.size](#)

Of these features, network buffer size has the most impact on memory usage. Setting network buffer size to the maximum can consume 30% or more of available heap. NTP client, broadcast and RSSI averaging each consume less than 1% of available heap.

To minimize memory usage, disable defaults using the following WiConnect commands:

WiConnect commands	Description
set ntp.enabled 0	Disable the NTP client
set broadcast.interval 0	Turn off UDP and HTTP broadcast
set wlan.rssi_average 0	Disable RSSI averaging
set network.buffer.size 10000	Minimize the network buffer size
save	
reboot	

Streams

Applications that use streams consume memory according to the number of streams in use. Streams are used by the file system, and the various servers and clients. For usage and limits on streams, see [Network Connections and Streams](#).

Memory Management Variables

Some variables allow control of the size and proportion of memory allocated to various tasks.

HTTP clients

It may be necessary to restrict the [maximum number of clients](#) the HTTP server allows. Since each HTTP client uses a stream, restricting clients prevents excessive memory usage, and it also avoids web browsers opening multiple streams for a single connection (some browsers do this in an effort to minimize page load time).

- [http.server.max_clients](#)

Network Buffer

Reducing the [network buffer size](#) may prevent excessive memory usage. In some applications that are either receive or transmit intensive, efficiency can be improved by adjusting the [ratio](#) of Rx and Tx buffer usage. Monitor network buffer usage with the [network.buffer.usage](#) variable.

- [network.buffer.rxtx_ratio](#)
- [network.buffer.size](#)
- [network.buffer.usage](#)

goHACK.me Cache

When using goHACK.me in solo mode, samples are cached before sending to the goHACK.me server. You can adjust the [size of the cache](#) to manage memory.

- [ghm_cache_size](#)

Configuration and Setup

Setup, in its simplest form, is [Wi-Fi Setup](#), also called Wi-Fi provisioning: providing WiConnect with your network ssid and passkey. Once these details are provided, WiConnect automatically brings the network up when it is required by any WiConnect activity.

Setup may also involve [Variable Configuration](#) and [GPIO Configuration](#).

To configure your own version of the WiConnect Web App, see [Customizing the WiConnect Web App](#).

Some of the procedures described below use a WiConnect terminal. Connecting a terminal is described in [Getting Started](#).

Wi-Fi Setup

A number of flexible and easy-to-use options are available to set up a module running WiConnect.

WiConnect is so easy to setup, we bet that within just a couple of minutes after plugging your evaluation board in, you'll be connected to a Wi-Fi network. Why not try it right now? With just four WiConnect commands, you can connect to the Internet and download information! Enter the following commands into a WiConnect terminal, substituting the name and password for your access point:

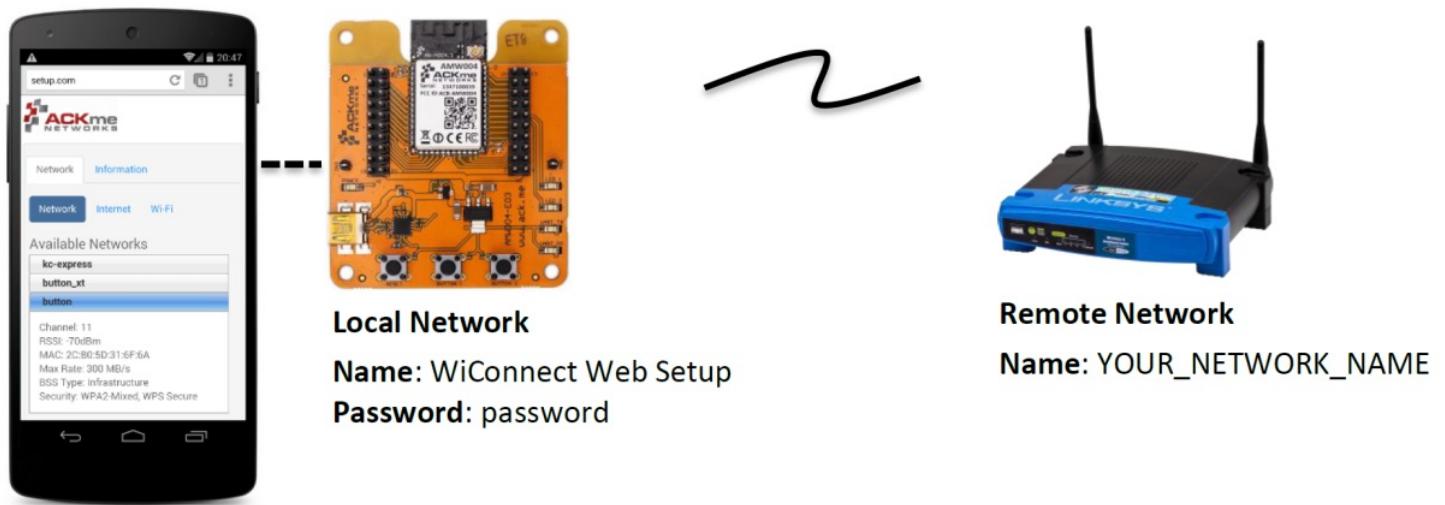
WiConnect Commands	Description
set wlan.ssid "Your AP Name"	<- Name of your Access Point
set wlan.passkey "Your secret password"	<- Password for your Access Point
http_get https://google.com	<- Sample information: get the home page from the Google secure web address
stream_read 0 1000	<- Read 1000-bytes of the sample

The setup steps are the first two lines of the example above - connecting to your local Wi-Fi Access Point. For further quick start information see [Getting Started](#).

Setup with a Web browser

WiConnect provides the option to use a web browser running on a network client (such as a smartphone, tablet or computer) to set the `wlan_ssid` and `wlan_passkey`.

In web setup mode, the ACKme module starts a soft AP and an HTTP server. The network client joins the module network via the soft AP and loads a set up page running on the WiConnect HTTP server.



Starting Web Setup Mode on an ACKme Evaluation Board

On an ACKme evaluation board in the factory default state, you can start web setup mode without connecting via a terminal: hold down Button 2, press Reset, and continue to hold down Button 2 for another three seconds. The evaluation board red LED flashes to indicate the WiConnect soft AP is running. See [Initiating Setup by GPIO](#).

Starting Web Setup Mode from a WiConnect Terminal

Alternatively, to start web setup mode, connect via a terminal and enter the command `setup web`.

WiConnect starts the local network and web server as indicated by the final message: In progress.

```
> setup web
[Disassociated]
IPv4 address: 10.10.10.1
Web setup started with the SSID: "WiConnect-###"
In progress
```

Opening the Web Setup Page

Open the Wi-Fi settings on your smartphone, tablet or computer and join the network called Wiconnect-###. The ### represents a unique ID derived from the last 3 characters of the module **MAC address**, e.g. WiConnect-2D6 for a module with MAC address 4C:55:CC:10:32:D6. The password for the network is simply: password.

The local network name and password, and the web address may be customised to suit your needs. See `setup.web.ssid`, `setup.web.passkey` and `setup.web.url`.

After joining the local network, open a web browser on the network client and direct the browser to setup.com. The WiConnect web page appears, and a scan begins for Wi-Fi access points in range. Select the remote network you wish to join, enter the network password then select Save & Exit and follow the prompts.

Once the settings are successfully saved, WiConnect prints Web Setup Mode exited to the terminal. You can check that the settings were successfully saved using get wlan.ssid and get wlan.passkey.

```
Web Setup Mode exited
> get wlan.ssid
YOUR_NETWORK_NAME
> get wlan.passkey
YOUR_NETWORK_PASSWORD
```

Web Setup Timeout

Web Setup times out after a period of user inactivity. If after web setup starts, the user does not open the web page, WiConnect exits setup mode after five minutes. If after opening the page the user does not click any button on the web page for five minutes, the setup web page displays a warning message. If there is no activity for 60 seconds after the message is displayed, the web page exits and web setup stops.

Setup by WiConnect Terminal

Perhaps the most common option used for general setup (and also the most convenient when prototyping) is to use a WiConnect terminal and set variables using WiConnect commands. See [Getting Started](#).

It is straightforward to set the wlan.ssid and wlan.passkey variables manually. Be sure to save afterwards, or the values will be lost when the module reboots.

```
> set wlan.ssid YOUR_NETWORK_NAME
Set OK
> set wlan.passkey YOUR_NETWORK_PASSWORD
Set OK
> save
Saved
Success
```

Any subsequent command requiring network access, such as an ICMP (Internet Control Message Protocol) ping, automatically results in the module attempting to join the network.

```
> ping -g
[Associating to YOUR_NETWORK_NAME]
Security type from probe: WPA2-Mixed
Obtaining IPv4 address via DHCP
IPv4 address: 192.168.0.31
[Associated]
Ping reply in 25ms
```

Network Setup Option

The **network_up** command provides a **-s** option that prompts for the network and passkey. This simplifies the process by scanning for networks and allowing the desired network to be selected by index number, e.g.:

```
> nup -s
Scanning for networks...
! 5 networks found
! # Ch RSSI MAC (BSSID)      Network (SSID)
# 0  1  -78 00:0E:E8:B2:FE:FC someone-else
# 1  1  -79 18:33:9D:5F:9E:F3 another-network
# 2  6  -53 84:1B:5E:D8:0F:18 ackme_EXT
# 3  6  -28 30:85:A9:E7:9C:B0 ackme
# 4 11  -84 58:BF:EA:D8:C9:D2 not_us

Type the number # that matches your Network: 3
Type the password for your Network      : <passkey>
```

After running **nup -s** you may want to set **wlan.auto-join.enabled**. Save the variables if you want them to persist through a reboot:

```
> set wlan.auto_join.enabled 1
Set OK
> save
Saved
Success
```

Setup by Remote Terminal

An (optionally) password-secured remote terminal provides command line setup convenience (look ma, no wires!) via one of the WiConnect wireless network interfaces. See [Application Examples - Remote Terminal](#). After connecting to the ACKme device via a remote terminal, follow the procedure described above in [Setup by WiConnect Terminal](#).

Initiating Setup by GPIO

Setup can be triggered by changing the level of a specified GPIO. See [Automatically Executing a Script](#).

If **setup gpio mode** is set to **gpio**, and the GPIO specified by **setup gpio control_gpio** is held to the level specified by **setup gpio level** for 3 seconds during and immediately after reset, then the command specified by **setup gpio cmd** is run.

On ACKme WiConnect evaluation boards, the default factory configuration is:

- **setup gpio control_gpio** - GPIO corresponding to Button 2: Wallaby: GPIO11, Numbat: GPIO22
- **setup gpio cmd** - setup web
- **setup gpio level** - 1
- **setup gpio mode** - gpio

Accordingly, on a default configuration eval board, hold down Button 2, press reset, and continue to hold Button 2 for three seconds to start web setup.

Initiating Setup on Boot

Set the value of `setup.auto.cmd` to the required setup cmd. See [Automatically Executing a Script](#).

Setup by WPS Push-button or PIN

WPS is a Wi-Fi provisioning method originally intended to simplify the process of connecting Wi-Fi clients to Wi-Fi Access Points. WPS offers both a push-button and PIN entry method for configuration. In reality, WPS push-button (as opposed to PIN) is the only method that has gained some level of adoption in the industry, however WPS naturally only works when the Wi-Fi AP supports WPS. Many AP vendors choose not to test and certify APs with the Wi-Fi Alliance, and the lack of a standard WPS logo next to the WPS button on an AP often means many users are unaware that WPS is available. The potential for equipment incompatibility and added user confusion mean it is unwise to rely on WPS as the primary method of Wi-Fi provisioning in the real world. Incompatibility and confusion aside, WiConnect provides full support for WPS1.0 & WPS2.0 and the underlying WPS engine has passed Wi-Fi certification. To use WPS in push-button mode, simply enter the `wps` command into WiConnect, then press the WPS button on your router (if the router supports WPS, and it is enabled, and you can find the button), and wait for the magic to happen.

Setup Configuration Script

A custom configuration script can be set up to execute on boot-up. WiConnect provides a default setup script to step you through the process. The setup script is provided as a file called `default_setup.script` on the WiConnect file system. The setup script may be customised as required. See [Script Format](#). You can also create your own configuration script and supply the filename as an argument: `setup cmd <script_file>`.

For example, the `default_setup.script` file contents are as follows:

```
network_up,-s,Configuration network credentials
set wlan.auto_join.enabled,true,Enable network auto-join
save,-,Saving settings
```

On running the `default_setup.script`, the output is similar to the following:

```
> setup cmd
Running setup script: default_setup.script
In progress
> Configuration network credentials
network_up -s
Scanning for networks...
! 9 networks found
! # Ch RSSI MAC (BSSID)      Network (SSID)
# 0  1  -77 18:33:9D:5F:9E:F2 Some AP
# 1  1  -79 18:33:9D:5F:9E:F4 Another AP
# 2  6  -31 30:85:A9:E7:9C:B0 ackme
# 3  6  -66 E8:08:8B:CA:4A:CC Yet Another AP

Type the number # that matches your Network: 2
Type the password for your Network      : secret_passkey
[Associating to ackme]
> In progress
Enable network auto-join
```

```
set wlan.auto_join.enabled true
Set OK
Saving settings
save
Saved
Success
Script executed successfully
```

Automatically Executing a Script

On Boot

To automatically execute a script on booting an unconfigured module, set the value of [setup.auto.cmd](#) to the required setup cmd, e.g.

```
set setup.auto.cmd "setup cmd my.script"
```

Note: The "setup cmd" value must be enclosed in double quotes because it contains spaces.

On GPIO Assertion or Reboot

To execute a script in response to a reboot, or a GPIO being asserted, set the value of [setup gpio.cmd](#), e.g.

```
set setup gpio.cmd "setup cmd my.script.txt"
```

In addition, set the required [setup gpio.control_gpio](#) and [setup gpio.mode](#).

Displaying Comments and Command Output

To display comments and command output, add the -v option after the setup cmd, e.g.

```
set setup gpio.cmd "setup cmd -v my.script"
```

See [setup cmd](#).

Script Format

The configuration script is in .csv format. Each line represents a command, in the form:

```
<command>,<arguments>,<comment>
```

The maximum line length of any line in a setup script is 128 characters (including \r\n).

You can create complex scripts, that run other commands to perform actions such as GPIO configuration. For example, here is part of a goHACK.me setup script:

```
,----- Free up GPIOs
set ,gpio.init 0 none          ,\r\n# Free up GPIO configured for user button 1
set ,gpio.init 22 none         ,\r\n# Free up GPIO configured for user button 2
set ,gpio.init 16 none         ,\r\n# Free up GPIO configured for user LED 1
set ,gpio.init 13 none         ,\r\n# Free up GPIO configured for user LED 2
```

```
set ,ghm.solo.gpio all none          ,\r\n# Free previously configured goHACK.me GPIOs
,,-----
set ,ghm.solo.gpio 0 button1 in rising ,\r\n# Setup Button 1 on GPIO 22 for use with
goHACK.me
set ,ghm.solo.gpio 22 button2 in rising ,\r\n# Setup Button 2 on GPIO 18 for use with
goHACK.me
set ,ghm.solo.gpio 20 thermistor adc ,\r\n# Setup Thermistor on GPIO 20 for use with
goHACK.me
,,set ,ghm.solo.gpio 23 gpio23 inpd ,\r\n# Setup GPIO23 for use with goHACK.me
```

The comment is printed before the corresponding command is executed.

The comment fields in the above example contain \r\n# characters for formatting purposes only. The effect is to skip a line then print the # character before the text of the comment.

Note: To comment out a line completely, insert two commas at the beginning of the line. See examples above.

Note: Each line of the script is terminated by \r\n (<CR><LF>).

Write the script to the WiConnect file system. See [Writing Files to the File System](#).

Now you can run the script as follows:

```
setup cmd -v test.script
```

The -v option results in a verbose display of comments and commands in the script:

```
> setup cmd -v test.script
Running setup script: test.script
In progress
>
#This is a comment
help setup
Usage   : setup <web / status / stop / cmd [script]>
Shortcut: setup
Brief   : Use setup mode to enable device configuration with a softAP and webserver
or a command prompt. Configure the softAP using the setup.web.* variables.
The cmd option runs the default_setup.script on the filesystem, or the [script] specified.
Script executed successfully
```

Variable Configuration

You can save the current variables to flash, so that they persist after reboot, with the [save](#) command. Multiple configurations can be saved by providing a save filename, e.g.:

```
save config1.cfg
```

When you reboot, the module loads the currently saved configuration. You can load a specified save file with the [load](#) command after reboot is complete, e.g.:

```
load config1.cfg
```

You can save a default configuration by saving a file with the name `default_config.csv`:

```
save default_config.csv
```

The default configuration is loaded after a successful [OTA](#), or in response to asserting the factory reset GPIO for more than 5 seconds and less than 10 seconds. See [save](#) and [factory_reset](#).

To return all values to their original values, use the [factory_reset](#) command:

```
fac <mac_address>
```

The [factory_reset](#) command requires a MAC address in order to avoid accidental reset. The MAC address can be obtained from the [wlan.mac](#) variable.

GPIO Configuration

GPIOs are configured in the factory default variable configuration. You can use GPIO commands and variables to configure GPIOs. See [Peripherals, GPIO Commands and Variables](#). It is also possible to use a csv file to configure and initialize GPIOs on bootup.

Using a GPIO Configuration File

The file name is set with the [gpio.config_file](#) variable. On some boards a default GPIO configuration script is provided with the filename `gpio_config_init.csv`.

The GPIO config file format consists of lines, each of which configures a GPIO. The file is of the form:

```
GPIO#,Alias1,Function1,Initialization1
GPIO#,Alias2,Function2,Initialization2
GPIO#,Alias3,Function3,Initialization3
...

```

where:

- GPIO# is any valid GPIO number
- Alias is any unique 1-15 character alias
- Function is a standard or system GPIO function (see below)
- Initialization sets the boot state or option and may be either:
 - a fixed value of 0 or 1; or
 - a System Indicator State Option combination

Note that all fields are optional (except the GPIO# field).

Standard GPIO Functions

- input_pull_up
- input_pull_down
- input_highz
- output
- output_open_drain
- output_open_drain_pull_up

Sleep State

To set the sleep state in the gpio config file add the | in the gpio function column.

For example:

Set GPIO 0 with gpio.alias = "reset", gpio.init = "input_pull_down", gpio.sleep = "input_highz":

0,reset,input_pull_down|input_highz

See the [gpio.sleep](#) variable description for state options.

Alternative GPIO Functions

- bus.stream.cmd_gpio
- ioconn.control_gpio
- ioconn.status_gpio
- network.status_gpio
- tcp.server.connected_gpio
- tcp.server.data_gpio
- system.indicator.wlan
- system.indicator.network
- system.indicator.softap

System Indicator State Options

State Name	Blink Period	Blink Frequency
static_on	-	-
static_off	-	-
slow_blink	T=2s	f=0.5Hz
medium_blink	T=1s	f=1Hz
fast_blink	T=0.250s	f=4Hz

Example GPIO config file

```
10,grn,system.indicator.wlan,static_off|fast_blink|medium_blink
11,yel,system.indicator.network,fast_blink|medium_blink|slow_blink
0,red,system.indicator.softap,static_off|fast_blink|medium_blink
4,button1,input_pull_up,
```

```
3,button2,input_pull_up,  
1,network_status,network.status_gpio,  
2,io_status,ioconn.status_gpio,  
5,alias_only,,  
8,output_high,output,1ds  
7,,output,0
```

System Functions

System functions cover the configuration, status and control of a number of miscellaneous and loosely coupled features. These are managed by the [system variables](#) and other commands and variables, as follows:

Configuring WiConnect Command Mode and Response

WiConnect commands can be entered manually via a WiConnect terminal or by a host MCU system.

There are optimizations for [human](#) and [machine](#) control.

Variables:

- [system.cmd](#) variable group:
 - [system.cmd.buffered](#)
 - [system.cmd.echo](#)
 - [system.cmd.header_enabled](#)
 - [system.cmd.mode](#)
 - [system.cmd.prompt_enabled](#)
 - [system.cmd.timestamp](#)
- [system.print_level](#)
- [system.msg](#)

Power management

Power consumption can be minimized, and battery life extended, by a number of strategies.

Commands and variables:

- [sleep](#) command
- [system.wakeup.events](#) - see also [Peripherals](#), [GPIO functions](#), [wake capability](#)
- [system.wakeup.timeout](#)
- [system.gotosleep.timeout](#)
- [system.powersave.mode](#)

See also Wi-Fi power management variables:

- [wlan.powersave.listen_interval](#)
- [wlan.powersave.mode](#)
- [wlan.powersave.sleep_delay](#)

Sleeping and Waking

When the module is not required to be active, you can put it into a low-power sleep mode with the [sleep](#) command, but don't forget to configure a wake mechanism before putting the module in a sleep state. This wake can be either a timeout ([system.wakeup.timeout](#)) or a GPIO level change event ([system.wakeup.events](#)).

The module may also be awoken from sleep with a power cycle.

It is also possible to set the module to go to sleep after a specified time by setting [system.gotosleep.timeout](#).

Powersave Mode

System powersave mode allows the module to power down when idle, with a rapid power up in response to activity. See [system.powersave.mode](#). Powersave mode is on by default.

Monitoring System State

System state can be monitored via visual indication by LED, system messages and GPIO level indication.

State monitoring variables:

- [system.indicator gpio](#) - see also [Peripherals, System Indicator Functions](#)
- [system.indicator.state](#)
- [system.msg](#)
- [system.activity gpio](#)
- [system.activity gpio_level](#)

Sub-system states can also be associated with GPIOs. See [Peripherals](#).

For a detailed snapshot of system state [get](#) the values of [all](#) the WiConnect variables:

```
get all
```

Controlling the Boot Application

Boot application commands and variables:

- [system.boot_app](#)
- [system.safemode](#) - see also [Upgrade and Recovery, Safe Mode](#)
- [reboot command](#)
- [factory_reset command](#) - see also [Upgrade and Recovery, Factory Reset](#)

The boot application is by default wiconnect.exe, but may be changed for various reasons.

If exceptions occur due to misconfiguration, the module may switch to safe mode, changing the boot application to upgrade_app.exe. See [Upgrade and Recovery, Safe Mode](#).

System Identification and Version

Commands and variables:

- [system.uuid](#)
- [system.version](#)
- [version command](#) - this gives the same result as `get system.version`

Upgrade and Recovery

WiConnect is part of a larger sophisticated application framework installed on all ACKme modules. The framework provides a hardened bootloader, read/write file system, safemode recovery mechanism, and secure over-the-air (OTA) upgrade capability.

Secure OTA Upgrade

ACKme manages a secure online OTA server that offers each ACKme module with the ability to securely upgrade individual files, applications or an entire firmware bundle. The **OTA command** is used to upgrade module firmware using the **OTA server** on this port.

The OTA process is secured by industry standard TLS1.0/HTTPS using server and client certificates that forces

- ACKme devices to verify the identity of the ACKme OTA server; and
 - the ACKme OTA server to verify the identity of each individual ACKme device.
- Each ACKme device is uniquely identified by a 128-bit hardware universally unique identifier (UUID).

HTTPS is the same security your web browser uses to make secure transactions with your bank over the internet. ACKme adds an additional layer of security, since the ACKme OTA server checks the unique security certificate and hardware UUID of each ACKme device that requests an upgrade.

If an OTA upgrade fails to complete for some reason, you can resume it later. Rejoin the network and run the **ota** command again.

Safe Mode

Safe mode operation is guaranteed in the unlikely event the module is configured in a way that causes repeated and/or unexpected reboots. The bootloader automatically switches to safe mode after eight exceptions occur with the offending boot application. Safe mode is indicated by the **system.safemode** variable.

It is easy to return the module to normal operation when the module is in safe mode. Follow the command sequence in the following table and your module will be back to normal in no time.

WiConnect Commands	Description
get system.safemode	<- Check if the module is in safemode? 0 = no, 1 = yes
faults_print	<- Print a list of faults (8 faults should print)
faults_reset	<- Reset faults counter
faults_print	<- Check there are no more faults after reset
reboot	<- Reboot the module

In some cases, a misconfigured module may quickly return to safe mode since the module may be inadvertently setup to invoke a fault. If this occurs, we recommend the module is returned to **factory reset**.

Once in safe mode, the module runs the upgrade application. The upgrade app provides the ability to upgrade the firmware, or switch back to WiConnect using the `upgrade_app faults_reset` command and `system.boot_app` variable.

For a detailed discussion of recovery procedures, see the **Recovery from Safemode** application note.

Factory Reset

The module may be factory reset using the [factory_reset](#) command or by holding the factory reset pin (GPIO 0 on all ACKme modules) high for more than 10 seconds through a hardware reset. GPIO0 is wired to Button 1 on the Wallaby and Moray evaluation boards.

After a successful factory reset, all variables are set to factory defaults and the module reboots. To avoid accidental factory reset, the Wi-Fi MAC address must be provided when calling the [factory_reset](#) command.

Application Examples

Examples on this page provide a great way to get started writing applications that use WiConnect. A number of examples use two ACKme modules running WiConnect to demonstrate end-to-end connected applications. We have lots more application examples on the way. If you have a specific application in mind, please create a [support ticket](#) and provide details. We'll do our best to write up your application and add it here.

Notes

- The application command sequences provided with all examples on this page are intended to be conveniently pasted directly into a WiConnect terminal.
- Unless otherwise indicated, all applications are configured for use with AMW003-E03 Mackerel boards. If you have a different eval board, GPIO settings may need to be adjusted to suit your board. See [Peripherals](#).
- All examples assume the Mackerel board starts from a factory reset state
- Some examples use python scripts to demonstrate connection between a computer and an ACKme module(s). All scripts are based on Python 2.7.

Recent Changes

App Note	WiConnect Version Required	Last Modified
HTTP Server WebSocket Demonstration	v2.1	2015-Feb-17
TCP Client	v1.0	2015-Feb-14
TCP Server + SoftAP	v1.0	2015-Feb-13
TCP Client and UDP Client Auto-Connect	v2.1	2015-Feb-03
Secure HTTP Server, with Client Authentication	v2.1	2015-Feb-03
mDNS Discovery	v2.1	2015-Feb-03
goHACK.me Triggers	v1.2	2015-Feb-03
goHACK.me Nest Integration	v1.2	2015-Feb-03
Recovery from Safe Mode	v1.0	2015-Feb-03
Broadcast UDP Packet	v1.0	2014-Nov-28
GPIO-Controlled Network Connection	v2.0	2014-Nov-28
High Speed UART	v2.0	2014-Nov-28
Secure TLS Client	v1.0	2014-Nov-28

By Topic

Serial Interface

- [Wi-Fi Remote Terminal](#) - Modified: 2014-Oct-20

Stream Mode

- [Wireless Serial Port](#) - Modified: 2015-Feb-03
 - [High Speed UART](#) - Modified: 2014-Nov-28
-

WLAN

- [Multi-client WLAN Messaging](#) - Modified: 2014-May-08
-

TCP Client

- [TCP Client](#) - Modified: 2014-Apr-03
 - [Multiple TCP clients](#) - Modified: 2014-May-08
 - [Secure TLS Client](#) - Modified: 2014-Nov-28
 - [GPIO-Controlled Network Connection](#) - Modified: 2014-Nov-28
 - [Auto-connect](#) - Modified: 2015-Feb-03
-

TCP Server

- [TCP Server + SoftAP](#) - Modified: 2014-Apr-03
-

UDP Client

- [Broadcast UDP Packet](#) - Modified: 2014-Nov-28
 - [Multi-client WLAN Messaging](#) - Modified: 2014-May-08
 - [Auto-connect](#) - Modified: 2015-Feb-03
-

HTTP Server with RESTful API, JavaScript API, Python API

- [HTTP RESTful API](#) - Modified: 2015-Feb-03
 - [Secure HTTP Server, with Client Authentication](#) - Modified: 2015-Feb-03
 - [HTTP Server WebSocket Demonstration](#) - Modified: 2015-Feb-17
 - [WiConnect JavaScript API](#)
 - [WiConnect Python API](#) - Coming soon
-

mDNS Network Discovery

- [mDNS Discovery](#) - Modified: 2015-Feb-03
-

Soft AP

- [TCP Server + SoftAP](#) - Modified: 2015-Feb-13
 - [Wireless Serial Port](#) - Modified: 2015-Feb-03
-

Peripherals

- [Controlling GPIOs & LEDs](#) - Modified: 2015-Feb-03
 - [GPIO-Controlled Network Connection](#) - Modified: 2014-Nov-28
-

Transmitting ADC Data

- [Broadcast UDP Packet](#) - Modified: 2015-Feb-03
-

Firmware Update & Recovery

- [Recovery from Safe Mode](#) - Modified: 2015-Feb-03
-

goHACK.me

- [Slave Mode](#) - Modified: 2014-Oct-20
 - [Solo Mode](#) - Modified: 2014-Oct-20
 - [Triggers](#) - Modified: 2015-Feb-03
 - [Nest Integration](#) - Modified: 2015-Feb-03
-

TCP Server + softAP

This application uses three ACKme modules. Module 1 is a TCP server configured to start its softap interface and its TCP server feature on bootup. Module 2 and Module 3 are TCP clients configured to use their wlan interface and TCP client feature to connect to the Module 1 softap and TCP server.

Once module 2 and module 3 are connected, data is sent to module 1 first from module 2, and then from module 3. Module 1 reads the received data.

The `stream_list` command is used to list open streams, and `stream_poll` is used on the server to check for data received by clients. Two GPIOs are configured to toggle high when a client connects, and also when data is received and waiting.

In this example, the TCP server is configured to use the softAP interface, but it can also easily be configured to use the wlan client interface.

Module 1 - TCP Server Settings

WiConnect Commands (Module 1)	Description
<code>set softap.auto_start true</code>	<- softap auto starts on reboot
<code>set softap.dhcp_server true</code>	<- DHCP server is enabled
<code>set softap.ssid tcp_server_ap</code>	<- Set softAP name
<code>set tcp.server.auto_interface softap</code>	<- TCP server uses softAP interface
<code>set tcp.server.auto_start true</code>	<- TCP server auto starts on reboot
<code>set tcp.server.idle_timeout 300</code>	<- Don't disconnect clients quickly
<code>set tcp.server.connected_gpio 22</code>	<- GPIO indicates client connection
<code>set tcp.server.data_gpio 21</code>	<- GPIO indicates data is available
<code>save</code>	<- Save settings to flash
<code>reboot</code>	<- Reboot the module

Example TCP Session with Module 2

Module 2: Wi-Fi + TCP Client	Module 1: softAP + TCP Server

```
> set wlan.ssid tcp_server_ap
Set OK
> save
Success
> tcp_client 10.10.10.1 3000
[Associating to tcp_server_ap]
[Associated]
[Connecting: 10.10.10.1:3000]
[Connected: 0]
0
> stream_write 0 22
<hello from module 2!>
Success
```

```
> list
! # Type Info
# 0 TCPS 10.10.10.1:3000 10.10.10.2:27192
> stream_poll 0
1
> stream_read 0 100
<<hello from module 2!>
> stream_poll 0
0
```

Example TCP Session adding Module 3

Module 3: Wi-Fi + TCP Client

Module 1: softAP + TCP Server

```
> set wlan.ssid tcp_server_ap
Set OK
> save
Success
> tcp_client 10.10.10.1 3000
[Associating to tcp_server_ap]
[Associated]
[Connecting: 10.10.10.1:3000]
[Connected: 0]
0
> stream_write 0 22
<hello from module 3!>
Success
```

```
> stream_list
! # Type Info
# 0 TCPS 10.10.10.1:3000 10.10.10.2:32193
# 1 TCPS 10.10.10.1:3000 10.10.10.3:33312
> stream_poll all
0,0|1,1
> stream_read 1 100
<hello from module 3!>
> stream_poll all
0,0|1,0
```

Change Log

Modified	Changes	WiConnect Version Required
2014-Apr-03	Created	1.0+
2015-Feb-13	Added second client example	1.0+

TCP Client

This application uses the WiConnect **TCP client** command to open a TCP connection to a TCP echo server on the internet.

The TCP echo server, located on the internet at `test.ack.me:50007`, simply echoes all received data back to the WiConnect TCP client. Once the connection is established, `stream_write` is used to send data to the connection stream. With all data sent, `stream_poll` is used to check for received data on the connection stream, and `stream_read` is used to read data waiting on the stream.

You will only need one ACKme evaluation board for this example.

Connecting

Let's get started by connecting your board to a Wi-Fi network and opening the TCP connection stream. You can simply cut and paste the commands in the following table, but **DON'T FORGET** to replace your Wi-Fi network name and password for the defaults.

WiConnect Commands	Description
<code>set wlan.ssid YOUR_AP_NAME</code>	<- Set the name of your Wi-Fi AP
<code>set wlan.passkey YOUR_AP_PASSWORD</code>	<- Set the password for your Wi-Fi AP
<code>tcp_client test.ack.me 50007</code>	<- Open TCP client session

There is no need to bring up the WLAN network interface prior to opening a TCP connection, the TCP client (just like all commands that need the network) automatically connects to the network if a connection has not already been established.

An example showing what the session looks like if you entered the above commands successfully is shown below. Notice that the new TCP connection is assigned a stream handle, starting at 0 for the first TCP connection.

Session log when opening the TCP client connection

```
> tcpc test.ack.me 50007
[2014-05-01 | 12:34:45: Associating to YOUR_AP_NAME]
Obtaining IPv4 address via DHCP
IPv4 address: 192.168.0.79
[2014-05-01 | 12:34:48: Associated]
Resolving host: test.ack.me
[2014-05-01 | 12:34:48: Opening: test.ack.me:50007]
Connecting (TCP): test.ack.me:50007
[2014-05-01 | 12:34:48: Opened: 0]
0
```

Check the status of open stream before continuing. Use the [stream_list](#) command to get a list of open streams. The number in brackets appended to each line is the local port used for the TCP connection stream.

```
> stream_list
!# Type Info
#0 TCPC test.ack.me:50007 (41713)
```

Writing Data

Writing data to a stream is easy using the `stream_write` command. Copy and paste the following commands to write data to the open stream.

WiConnect Commands	Description
<code>stream_write 0 23</code> <code>Hello from stream 0</code>	<- Prepare to write 32-characters to TCP connection stream 0 <- Data to write

Checking for Data

To check for data on any stream (or all streams at once), use the `stream_poll` command as shown in the following session. If you don't want to poll for data, try assigning a GPIO to a stream to indicate when data is available. The `tcp_client` command describes how to assign a GPIO.

Checking for data

```
> stream_poll 0
1
> stream_poll all
0,1
```

Reading Data

To read available data from any stream, use the `stream_read` command as shown in the following session. The following example reads up to 100 bytes from the stream.

Checking for data

```
> stream_read 0 100
Hello from stream 0
```

Change Log

Modified	Changes	WiConnect Version Required
2014-May-08	Created	1.1+
2015-Feb-14	Changed the example to use the test.ack.me:50007 server	1.1+

Multiple TCP Clients

This application uses the WiConnect **TCP client** command to open 4 simultaneous TCP connections to a TCP echo server on the internet (WiConnect actually supports up to **8 simultaneous** connections).

The TCP echo server, located on the internet at `test.ack.me:50007`, simply echoes all received data back to the WiConnect TCP client. Once each connection is established, **stream_write** is used to send data to each connection stream in turn. With all data sent, **stream_poll** is used to check for received data on connection streams, and **stream_read** is used to read data waiting on a stream.

You will only need one ACKme evaluation board for this example.

Connecting

Let's get started by connecting your board to a Wi-Fi network and opening each of the 4 TCP connection streams in turn. You can simply cut and paste the commands in the following table, but **DON'T FORGET** to replace your Wi-Fi network name and password for the defaults.

WiConnect Commands	Description
<code>set wlan.ssid YOUR_AP_NAME</code>	<- Set the name of your Wi-Fi AP
<code>set wlan.passkey YOUR_AP_PASSWORD</code>	<- Set the password for your Wi-Fi AP
<code>tcp_client test.ack.me 50007</code>	<- Open TCP client session 0
<code>tcp_client test.ack.me 50007</code>	<- Open TCP client session 1
<code>tcp_client test.ack.me 50007</code>	<- Open TCP client session 2
<code>tcp_client test.ack.me 50007</code>	<- Open TCP client session 3

There is no need to bring up the WLAN network interface prior to opening a TCP connection, the TCP client (just like all commands that need the network) automatically connects to the network if a connection has not already been established.

An example showing what the session looks like if you entered the above commands successfully is shown below. Notice that each new TCP connection is assigned a stream handle, starting at 0 for the first TCP connection.

Session log when opening 4 TCP client connections

```
> tcpc test.ack.me 50007
[2014-05-01 | 12:34:45: Associating to YOUR_AP_NAME]
Obtaining IPv4 address via DHCP
IPv4 address: 192.168.0.79
[2014-05-01 | 12:34:48: Associated]
Resolving host: test.ack.me
[2014-05-01 | 12:34:48: Opening: test.ack.me:50007]
Connecting (TCP): test.ack.me:50007
[2014-05-01 | 12:34:48: Opened: 0]
0
> tcpc test.ack.me 50007
Resolving host: test.ack.me
[2014-05-01 | 12:34:49: Opening: test.ack.me:50007]
Connecting (TCP): test.ack.me:50007
[2014-05-01 | 12:34:49: Opened: 1]
1
> tcpc test.ack.me 50007
Resolving host: test.ack.me
[2014-05-01 | 12:34:49: Opening: test.ack.me:50007]
Connecting (TCP): test.ack.me:50007
[2014-05-01 | 12:34:49: Opened: 2]
2
> tcpc test.ack.me 50007
Resolving host: test.ack.me
[2014-05-01 | 12:34:50: Opening: test.ack.me:50007]
Connecting (TCP): test.ack.me:50007
[2014-05-01 | 12:34:50: Opened: 3]
3
```

Check the status of open streams before continuing. Use the [stream_list](#) command to get a list of open streams. The number in brackets appended to each line is the local port used for the TCP connection stream.

```
> stream_list
!# Type Info
#0 TCPC test.ack.me:50007 (41713)
#1 TCPC test.ack.me:50007 (46714)
#2 TCPC test.ack.me:50007 (51715)
#3 TCPC test.ack.me:50007 (56716)
```

Writing Data

Writing data to a stream is easy using the `stream_write` command. Copy and paste the following commands to write data to each open stream.

WiConnect Commands	Description
<code>stream_write 0 23</code> Hello from stream 0	<- Prepare to write 32-characters to TCP connection stream 0 <- Data to write
<code>stream_write 1 23</code> Hello from stream 1	<- Prepare to write 32-characters to TCP connection stream 1 <- Data to write
<code>stream_write 2 23</code> Hello from stream 2	<- Prepare to write 32-characters to TCP connection stream 2 <- Data to write
<code>stream_write 3 23</code> Hello from stream 3	<- Prepare to write 32-characters to TCP connection stream 3 <- Data to write

Checking for Data

To check for data on any stream (or all streams at once), use the `stream_poll` command as shown in the following session. If you don't want to poll for data, try assigning a GPIO to a stream to indicate when data is available. The `tcp_client` command describes how to assign a GPIO.

Checking for data
<pre>> stream_poll 0 1 > stream_poll 2 1 > stream_poll all 0,1 1,1 2,1 3,1 </pre>

Reading Data

To read available data from any stream, use the `stream_read` command as shown in the following session.

Checking for data

```
> stream_read 0 100
Hello from stream 0

> stream_read 1 100
Hello from stream 1

> stream_read 2 100
Hello from stream 2

> stream_read 3 100
Hello from stream 3
```

Change Log

Modified	Changes	WiConnect Version Required
2014-May-08	Created	1.1+

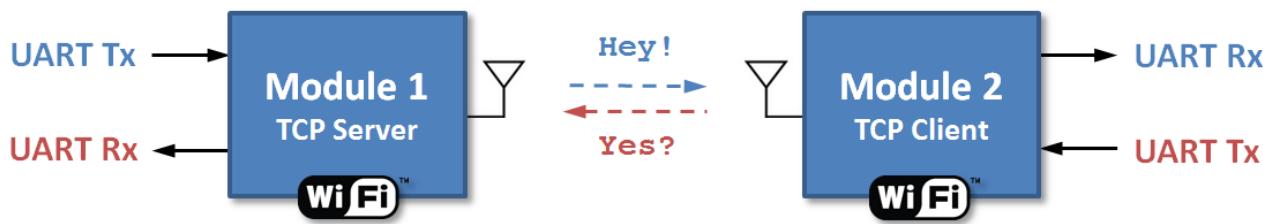


Wireless Serial Port

This application uses two ACKme modules each running WiConnect to demonstrate how to create a wireless serial port. A character typed into the serial port of Module 1 appears instantly at the output of the Module 2 serial port (and vice versa).

Features demonstrated

- Serial Interface Stream Mode
 - Use of the wlan and softap wireless interfaces
 - TCP client and TCP server



Module 1 is a server configured to start the softap interface and the TCP server feature on bootup. Module 2 is a client configured to use the wlan interface and the TCP client feature to connect to the Module 1 softap and TCP server. The final configuration step places both modules into serial interface Stream Mode. Enough chit-chat, let's get started by configuring each of the modules.

Copy and paste the commands in the left hand column of the following table into a WiConnect terminal connected to Module 1.

Module 1 Server Configuration Settings

WiConnect Commands (Module 1)	Description	
set softap.auto_start true	<- softap auto starts on reboot	
set softap.dhcp_server on	<- DHCP server is enabled	
set softap.ssid server_ap	<- Set server softap name	
set softap.url wiconnect.com	<- Set server DNS name	
set tcp.server.auto_interface softap	<- TCP server uses softAP interface	
set tcp.server.auto_start true	<- TCP server auto starts on reboot	
set tcp.server.port 3000	<- TCP server port	
set tcp.server.idle_timeout 300	<- Don't disconnect clients too quickly	
set tcp.server.connected_gpio 22	<- GPIO indicates client connection	
set tcp.keepalive.enabled 1	<- Check the TCP connection is still alive	
set tcp.keepalive.initial_timeout 10	<- Initial TCP check timeout is 10s	
set bus.mode stream	stream	<- Use serial bus stream mode
save	<- Save settings to flash	
reboot	<- Reboot the module	

Now, copy and paste the commands in the left hand column of the following table into a WiConnect terminal connected to Module 2.

Module 2 Client Configuration Settings

WiConnect Commands (Module 2)	Description
set wlan.ssid	<- Set name of Wi-Fi AP to join
server_ap	<- Turn on wlan auto-join
set wlan.auto_join.enabled	
true	
set tcp.client.remote_host	<- Set host for tcp client auto-connect
wiconnect.com	<- Set port for tcp client auto-connect
set tcp.client.remote_port	<- Set interface for tcp client auto-connect
3000	<- No auto retries, TCP keepalive checks the connection
set tcp.client.auto_interface	<- Enable tcp client auto-connect
wlan	
set tcp.client.auto_retries	<- Check the TCP connection is still alive
0	<- Initial TCP check timeout is 10s
set tcp.client.auto_start	
1	<- Use serial bus stream mode
set tcp.keepalive.enabled	
1	<- Save settings to flash
set tcp.keepalive.initial_timeout	<- Reboot the module
10	
set bus.mode	
stream	
save	
reboot	

On rebooting, Module 2 automatically connects to the TCP server running on Module 1 and enters stream mode.

Wireless Magic!

At this point, all characters typed into the Module 1 serial port appear immediately on the Module 2 serial port output (and vice versa). Give it a go, it's like a little piece of wireless magic!

Change Log

Modified	Changes	WiConnect Version Required
2015-02-03	Added tcp.keepalive, additional tcp.client variables	2.1+
2014-04-03	Created	1.0+

Controlling GPIOs and LEDs

This example demonstrates how to use GPIOs with WiConnect in general, and how to use GPIOs to control LEDs on ACKme evaluation boards. For details of using GPIOs and peripherals, see [Peripherals](#).

In WiConnect, a GPIO may have two functions: a **standard IO function** or an **alternative function** such as a system indicator, status GPIO or control GPIO. See [GPIO Configuration](#).

When a GPIO is configured with an alternative function, the standard IO function is NOT available and the `gpio_dir`, `gpio_set`, and `gpio_get` commands are disabled for that GPIO.

Before a GPIO can be used to control an LED, it is necessary to first check whether the GPIO is being used for an alternative function. If a GPIO is not being used for an alternative function, but is already in use as a standard GPIO (for another purpose), the GPIO must first be freed up by setting the direction of the GPIO to none using the `gpio_dir` command.

GPIO Usage

The `gpio.usage` variable makes it easy to get a list of GPIOs that are already in use. The session in the following table shows the GPIO usage after factory reset for a [Mackerel Evaluation Board](#) (schematics: [AMW004-E03 Rev 1](#)).

WiConnect Session (AMW004-E03 Rev 1)

```
> get gpio.usage
! # Description
#10 - Standard I/O
#11 - Standard I/O
#13 - UART1 RX
#14 - UART1 TX
#17 - SPI CLK
#18 - SPI MOSI
#19 - SPI MISO
#21 - system.indicator.network
#22 - system.indicator.wlan
```

LED Control

The following example is based on the first revision of a [Mackerel evaluation board](#) which only has two LEDs connected to GPIO 21 & 22. These LEDs are factory configured as [system indicators](#) to dynamically indicate the state of the Wi-Fi and network connection (as shown in the table above). To repurpose the GPIOs as standard IOs for use with LEDs, try the WiConnect command sequence shown in the table below.

Note! Don't forget to adjust the GPIO numbers to suit your specific board!

WiConnect Commands (for AMW004-E03 Rev1)	Command Description
set system.indicator gpio wlan -1 set system.indicator gpio network -1 save reboot	<- Disable the wlan system indicator <- Disable the network system indicator <- Save & reboot since system indicator functions are setup after a reset
set gpio.alias 21 LED1 set gpio.alias 22 LED2	<- Setup a friendly alias for the LED on GPIO 21 <- Setup a friendly alias for the LED on GPIO 22
gpio_dir LED1 out gpio_dir LED2 out	<- Set the GPIO direction to output <- Set the GPIO direction to output
gpio_set LED1 1 gpio_set LED2 1	<- Turn on LED 1 <- Turn on LED 2
gpio_set LED1 0 gpio_set LED2 0	<- Turn off LED 1 <- Turn off LED 2

To re-instate the system indicators, either factory reset the module, or use the following commands.

WiConnect Commands	Command Description
gpio_dir LED1 none gpio_dir LED2 none	<- Free up the GPIO aliased to LED1 <- Free up the GPIO aliased to LED2
set system.indicator gpio wlan LED1 set system.indicator gpio network LED2 save reboot	<- Set the wlan system indicator GPIO to LED1 <- Set the wlan system indicator GPIO to LED2 <- Save & reboot since system indicator functions are setup after a reset

GPIO Initialization

To simplify GPIO setup (on boot up), WiConnect provides the option to automatically configure GPIOs with a GPIO configuration script set using the [gpio.config_file](#) variable. Alternatively, GPIOs may be individually configured for initialization after reset using the [gpio.init](#) variable. Don't forget to save & reboot to make sure recent GPIO initialization changes take effect.

Change Log

Modified	Changes	WiConnect Version Required
2014-05-08	Created	1.1+

Multi-client WLAN Communications

This example demonstrates how to connect multiple ACKme modules running WiConnect to a network as Wi-Fi clients, and how to configure each module to send messages to one another. The requirements for this kind of application can vary considerably, here's a list of assumptions we made for this particular example.

- Each module has a pre-assigned static IP address. This makes it easy for a module to send a message to another specific module on the network since all modules have pre-assigned addresses
- A message may be sent from one module to one or more modules
- There is no security encryption at the network layer, all modules can read messages from other modules if they choose to do so
- Message delivery is NOT guaranteed. If a message is sent from one module to one or more modules, and the message fails to be received, there are no retries at the network layer (retries at the Wi-Fi layer still work though).

The simplest way to enable modules to send messages to one another on a local network is to send messages in UDP broadcast packets. To receive messages, each module runs a UDP client and listens for messages sent on the UDP broadcast address. While UDP broadcast provides a way to send messages successfully, it lacks the ability to address messages to individual clients.

A Simple Message Protocol

To address messages, a simple application layer message protocol can be used. The requirements of a simple message protocol are listed below.

- Each message must have a sender (the source of the message)
- Each message must have one or more recipients (the destination of the message)
- Each message must have a payload

The message packet structure shown in the following diagram provides everything needed.



where ...

- **Source Address** : Source address of the device sending a message. A 1 byte address provides up to 128 unique addresses, and 128 multicast addresses as explained below.
- **Destination Address** : Destination address of the recipient of the message. If individual destination addresses are limited to 0 through 127, then addresses 128 through 255 can be used as multicast addresses. If the sender wants to send particular types of messages to a group of clients, the sender uses a multicast address. Devices interested in those types of messages can listen on the particular multicast address of interest, in addition to their own address
- **Payload** : The message payload

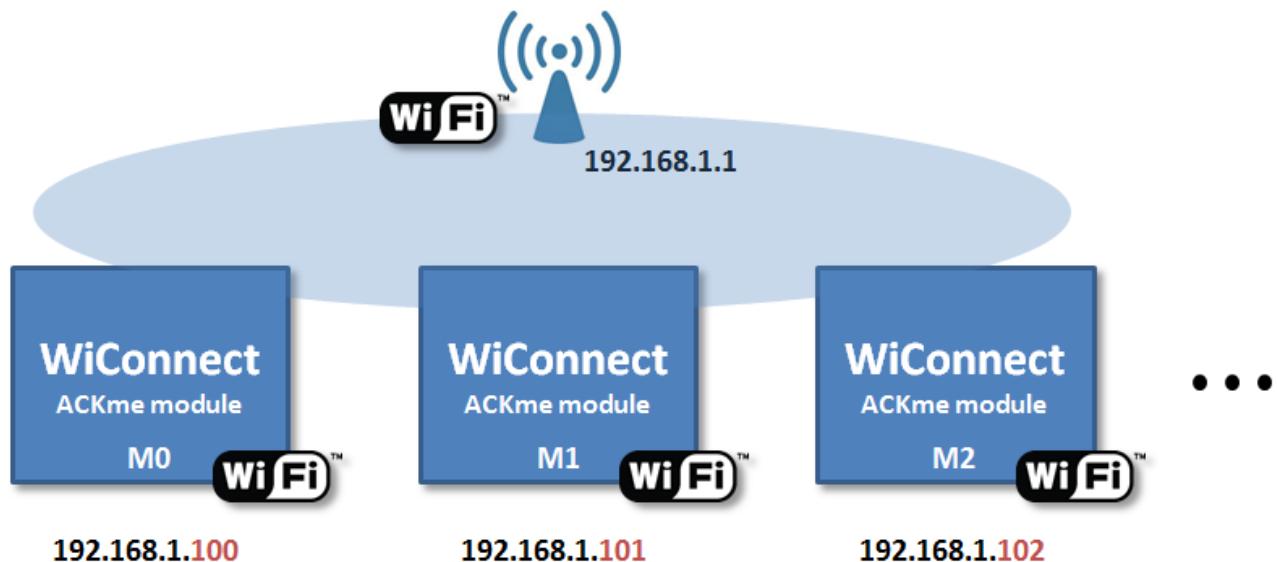
Implementation

This example assumes each Wi-Fi device has a pre-allocated (or static) IP address. There are several ways to achieve this.

- Setup the DHCP server on your Wi-Fi access point to always allocate a particular IP address to each Wi-Fi device based on

- the MAC address of the device (the MAC address of your ACKme module is available by reading the `wlan.mac` variable).
- Ensure that each device that joins the Wi-Fi AP uses a static IP address, and that no two clients use the same IP address. You can set a static IP address for your device using the `static.ip` variable. Note that you will also need to set the `static.netmask` and `static.gateway` variables.

For this example, we assume that IP addresses assigned to clients are numbered in the range 192.168.1.100, 192.168.1.101, 192.168.1.102, and so on. We also assume the Wi-Fi AP (which is the network gateway) has an IP address of 192.168.1.1. The assumed configuration of the network is shown here.



Before continuing, ensure your Wi-Fi AP is configured as described above. Or alternately, if you have a different configuration, ensure you have the IP addresses and ranges of your AP handy (you will need to substitute them below as required).

Commands to setup the first module M0 at address 0 (192.168.1.100) are shown below. Enter these commands now.

WiConnect Commands for Module M0	Command Description
<code>set wlan.ssid</code>	<- Set the name of your network
<code>YOUR_NETWORK_NAME</code>	<- Set the password for your network
<code>set wlan.passkey</code>	<- Set the static IP address of the module
<code>YOUR_NETWORK_PASSWORD</code>	<- Set the network gateway for the module
<code>set static.ip 192.168.1.100</code>	<- Set the network netmask for the module
<code>set static.gateway 192.168.1.1</code>	<- Turn off DHCP (we're using statically assigned IP addresses)
<code>set static.netmask 255.255.255.0</code>	<- Start a UDP client connected to remote & local ports 50000
<code>set network.dhcp.enabled 0</code>	
<code>udp_client 192.168.1.255 50000</code>	
<code>50000</code>	

With module M0 setup and ready, grab the next module (M1), plug it in and set it up using the same commands **EXCEPT** this time, change the `static.ip` variable to 192.168.1.101 so module M1 has address 1.

Verify it works

With both modules setup and connected to the Wi-Fi AP and the UDP client started on both modules, let's test that messages sent from one module are received by the other module.

In the following WiConnect session, the source and destination addresses (separated by commas) have been added to each message according to the simple message protocol described previously. Also, note that any text entered after a `stream_write` command is not echoed to the terminal.

Example WiConnect Session

Module 0	Module 1
<pre>> stream_write 0 12 0,1>Hello-M1 > stream_poll 0 1 > stream_read 0 50 1,0>Hello-M0</pre>	<pre>> stream_poll 0 1 > stream_read 0 50 0,1>Hello-M1 > stream_write 0 12 1,0>Hello-M0</pre>

The `stream_poll` command is used here to check whether data has been received. A more convenient notification method is to setup a GPIO interrupt as described in the [udp_client command](#).

Now you have two modules sending messages to each other, go ahead and add other modules to the network, remembering to increment the address for each new module.

Other Options?

This example describes a simple way to send messages between modules. There are other ways to achieve the same result, but the implementation of alternate methods is more complex.

- TCP client/server. Another option is to run a TCP server and TCP client on each module. Each time a module wants to send a message to another module, it must open a TCP client connection to the module it wishes to message. Each module must also run a separate TCP server to receive messages from other modules. This method provides guaranteed delivery, overcoming the potential for lost messages. Note that each module must solve the address problem described for UDP also.

Change Log

Modified	Changes	WiConnect Version Required

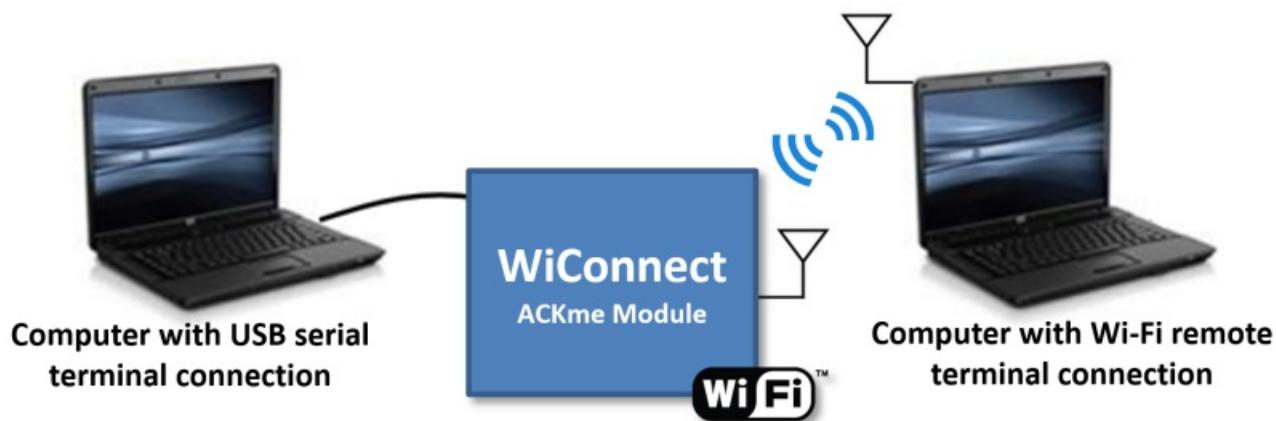
2014-05-08	Created	1.1+
2015-02-19	Corrections to variable names	1.1+

Wi-Fi Remote Terminal

This example demonstrates how to setup WiConnect to enable remote access to your module or evaluation board using telnet via the Wi-Fi interface. The example shows how to use the Wi-Fi softAP interface, but it is also possible to connect remotely using telnet via the Wi-Fi client interface.

Features demonstrated

- Soft AP
- Remote Terminal



Once your evaluation board and WiConnect is configured to enable the Soft AP and Remote Terminal feature, there is no longer any need to use a serial connection to control the board. The board can instead be controlled directly by [Connecting the Remote Terminal](#).

Setting up the Soft AP and Remote Terminal

The example is based on an ACKme Mackerel board, but the process is generic for any ACKme evaluation board with a serial interface and an ACKme module running WiConnect.

Connect a computer to the evaluation board USB connector. Use your terminal program to make a serial connection to the board (see [Getting Started](#)).

View the Default State of the Soft AP and Remote Terminal Variables

This step is for information only. It is NOT necessary to list the variables before you change them.

In the serial terminal console use the `get softap` and `get remote_terminal` command to display the current state of the softap and remote_terminal variables. In the default state the variables are as shown below:

```
> get softap
softap.auto_start: 0
softap.channel: 0
softap.dhcp_server: 1
softap.dns_server: 1
softap.idle_timeout: 7
softap.ip: 10.10.10.1
softap.passkey:
softap.rate.protocol: auto
softap.rate.transmit: auto
softap.ssid: WiConnect-#
softap.url: wiconnect,wiconnect.com,www.wiconnect.com

> get remote_terminal
remote_terminal.enabled: 0
remote_terminal.interface: default
remote_terminal.password:
remote_terminal.port: 2000
remote_terminal.timeout: 60
```

If the variables do not display as shown, you may wish to perform a factory reset to return them to the default state (see [Factory Reset](#)).

Change the Soft AP and Remote Terminal Variables

Setting up the soft Access Point (softap) and remote terminal, from the default state, requires just a few commands:

WiConnect Commands	Description
<code>set softap.auto_start 1</code>	<- Start softap automatically on boot
<code>set remote_terminal.enabled 1</code>	<- Enable the remote terminal
<code>set remote_terminal.interface softap</code>	<- Set the remote terminal to interface via the soft AP
<code>save</code>	<- Save the state of the WiConnect variables to flash memory
<code>reboot</code>	<- Reboot the device

You can reboot the board using the `reboot` command, by pressing the reset button, or by power cycling.

After reboot, WiConnect displays a message similar to the following:

```
> IPv4 address: 10.10.10.1
SoftAP 'WiConnect-098' started
Remote terminal listening on port: 2000
[Ready]
```

View the Changed State of the Soft AP and Remote Terminal Variables

This step is for information only. It is NOT necessary to list the variables after you change them.

After reboot, in the serial terminal, check the state of all softap and remote_terminal variables.

```
> get softap
softap.auto_start: 1
softap.channel: 1
softap.dhcp_server: 1
softap.dns_server: 1
softap.idle_timeout: 7
softap.ip: 10.10.10.1
softap.passkey:
softap.rate.protocol: auto
softap.rate.transmit: auto
softap.ssid: WiConnect-#
softap.url: wiconnect,wiconnect.com,www.wiconnect.com

> get remote_terminal
remote_terminal.enabled: 1
remote_terminal.interface: softap
remote_terminal.password:
remote_terminal.port: 2000
remote_terminal.timeout: 60
```

Connecting the Remote Terminal

On the computer running the remote terminal, connect to the Soft AP. The softap SSID is determined by the **softap.ssid** variable.

The default value of **WiConnect-#** means **WiConnect-**, followed by the last 3 digits of the device WLAN MAC address, e.g.

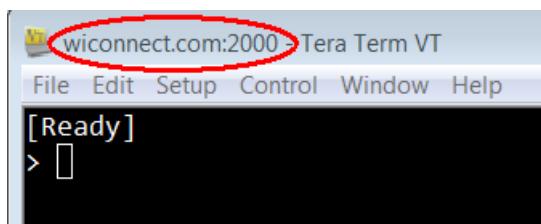
WiConnect-098. The passkey is determined by the **softap.passkey** variable. The default is empty i.e. no passkey.

In your communication terminal program (e.g. Teraterm) create a new **TCP/IP telnet** connection with the Host determined by the **softap.ip** or **softap.url** and the Port determined by **remote_terminal.port**:

Property	WiConnect variable	Default Value
Host:	softap.ip softap.url	10.10.10.1 wiconnect,wiconnect.com,www.wiconnect.com
TCP port:	remote_terminal.port	2000

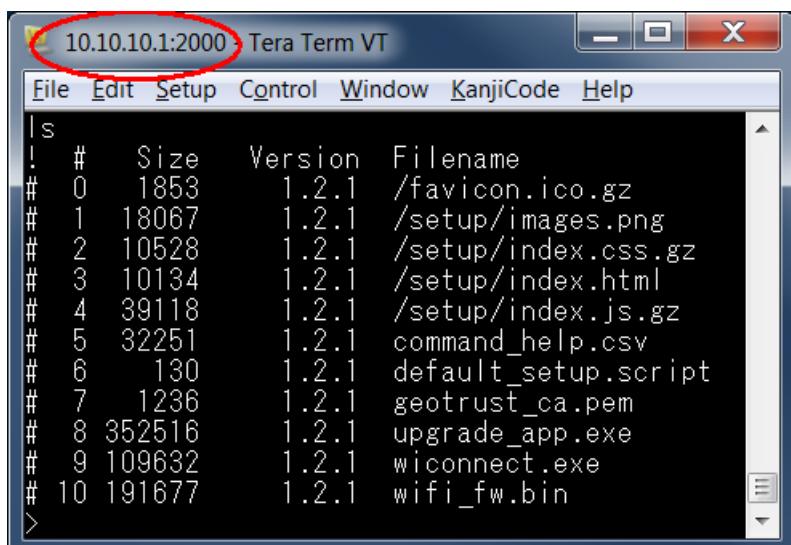
The terminal program should now complete the telnet connection to the eval board.

Note that the softap runs a DNS server by default. Softap urls are determined by the variable `softap.url`. For example, you can connect the remote terminal to `wiconnect.com:2000`, or set a `softap.url` and `remote_terminal.port` as desired.



Before typing commands, ensure that local echo is turned on. For example, in Teraterm, select the checkbox in the menu **Setup | Terminal | Local echo**.

The remote terminal provides full access to all WiConnect commands, exactly as if you were connected via a serial interface. When a telnet client is connected to the remote terminal via Wi-Fi, the serial interface is disabled.



If required, and to add security, a remote terminal password can be added by configuring the `remote_terminal.password` variable.

Change Log

Modified	Changes	WiConnect Version Required
2014-10-20	Created	1.2+

Secure TLS Client

Overview

This example demonstrates how to use the WiConnect `tls_client` command to connect to a secure **TLS** server using a self-signed TLS certificate. The TLS server runs on a computer using python.

The following steps are covered by this example:

- Generate TLS certificates using OpenSSL
- Run a python TLS server
- Load a TLS certificate into the module
- Connect the module to the TLS server using the `tls_client` command

NOTE: The procedures below for running OpenSSL commands, and running the Python TLS server script, will work on Linux, Mac or Windows computers with the correct software packages installed. The examples shown are for a Linux machine. Notes are provided where a variation is required for a different platform, and references for Windows in particular are provided in the final section at the bottom of the page.

TLS Certificate Primer

TLS certificates are used to:

- Encrypt data passing between a client and server
- Verify the identity of the remote side of the connection to avoid 'man in the middle' attacks

Certificate File Types

There are two types of files used in TLS connections:

- **Certificate (.crt or .pem)**

This is essentially the public key. Receiving parties typically keep this file locally in a certificate store. Several formats are used, the two most popular being .crt and .pem. WiConnect accepts only .pem format which is a base64-encoded X.509 certificate.

- **Private Key (.key)**

This is a sensitive file used to generate the certificate and validate connections. The private key must be kept secure.

Certificate Types

- **Certificate Authority (CA) Certificate**

This is essentially the master certificate. All other certificates are generated from this cert. If this certificate is trusted, all certificates generated from it are trusted as well. This certificate may be obtained by either (1) generating a self-signed CA cert or (2) using a CA cert from a 3rd Party.

You can generate a self-signed CA cert using a tool like OpenSSL. It is absolutely imperative the CA private key is secure. If it gets lost or comprised, all other certificates generated from it are vulnerable.

A 3rd party, such as VeriSign or GeoTrust, maintains a CA certificate and private key. The CA.crt is made public (and typically comes built-in to most browsers) and the CA.key is locked down in a secure location by the 3rd party.

■ Server Certificate

A server certificate is provided by a server to a client to verify the server identity to the client. The server cert is 'signed' by the CA certificate. For example, when a web browser connects to a website secured by HTTPS/TLS, the web server provides the browser with the server certificate enabling the browser to verify the identity of the server/website.

■ Client Certificate

A client certificate is provided by a client (device) to a server to verify the client identity to the server. The client cert is 'signed' by the Server Certificate. The client certificate enables the client (i.e. ACKme module) to securely log-in to the server without the need to supply a username and password. The client certificate is optional and not used by standard web browsers when accessing web sites secured by HTTPS/TLS (although a web browser can use a client cert if it is configured to do so). Client certs are not used in this example and are not discussed further here.

OpenSSL

For the following steps in which certificates are generated, it is assumed you have OpenSSL installed on your computer and OpenSSL is in the PATH environment variable i.e. you can execute OpenSSL directly from a command line. OpenSSL may be downloaded here: <https://www.openssl.org/related/binaries.html> or on Ubuntu, run the command:

```
~$ sudo apt-get install openssl
```

The manual for OpenSSL may be found here: <https://www.openssl.org/docs/apps/openssl.html>

In some instances, it may be necessary to set the RANDFILE environment variable to avoid OpenSSL errors such as unable to write 'random state'. To temporarily set the RANDFILE environment variable to a local file, use the following command:

```
set RANDFILE=.rnd
```

Generating a Self-Signed CA Certificate

The first certificate to generate is a self-signed Certificate Authority (CA) certificate. This is the root certificate from which all other certificates are derived.

We start by generating the CA cert key. This is the most sensitive file in your secure TLS system. This should be placed in a secure location.

```
~$ openssl genrsa -aes256 -out my_ca.key 4096
```

After executing this command OpenSSL prompts for a password. This is used to secure this CA key. This is the same password used for the rest of the process.

Next we create the self-signed CA cert:

```
~$ openssl req -new -sha1 -x509 -key my_ca.key -out my_ca.crt
```

OpenSSL prompts for various values. Press enter to accept a default value. At the 'Common Name' prompt, you can enter any name, for example 'my_cert'.

Next, convert `my_ca.crt` to `.pem` format:

First convert to an intermediate `.der` format

```
~$ openssl x509 -in my_ca.crt -out my_ca.der -outform DER
```

Now convert the `.der` to `.pem`:

```
~$ openssl x509 -in my_ca.der -inform DER -out my_ca.pem -outform PEM
```

~\$ Delete the temporary `.der` file:

```
> rm my_ca.der
```

For a Windows platform, use `del` instead of `rm`.

You have now generated the self-signed CA certificate in the required `.pem` format.

Generating a Server Certificate

Now that we have a CA certificate, we need to generate a server certificate. This is what our python server will use.

To generate the server certificate, you need to obtain the IP address of your computer on the local network. This is the IP address the module will use to connect to the python server (running on your computer).

For this example, we will assume the local IP address is: 192.168.1.228

NOTE: At every point where this address appears in the procedure below, you should substitute the IP address of the computer used to run the python TLS server script.

First generate the private key for our server certificate:

```
~$ openssl genrsa -aes256 -out 192.168.1.228.key 1024
```

OpenSSL prompts for another password. This can be the same as the CA cert's or different.

Next we need to create a Certificate Signing Request (CSR).

```
~$ openssl req -new -sha1 -key 192.168.1.228.key -out 192.168.1.228.csr
```

OpenSSL prompts for various values. Press enter to accept default values until reaching the '**Common Name**' prompt, then ...

... AT THE PROMPT 'Common Name': enter the IP address of the computer! ← **This is very important.**

Afterwards, when OpenSSL asks for a 'Challenge password', do NOT enter anything, just press Enter.

Now, remove the passphrase on the server key to avoid the need for manual entry of a passphrase when the server starts:

```
~$ cp 192.168.1.228.key 192.168.1.228.key.tmp  
~$ openssl rsa -in 192.168.1.228.key.tmp -out 192.168.1.228.key  
~$ rm 192.168.1.228.key.tmp
```

For a Windows platform, you may need to use copy instead of cp and del instead of rm.

Next we generate the server certificate:

```
~$ openssl x509 -req -in 192.168.1.228.csr -CA my_ca.crt -CAkey my_ca.key -out 192.168.1.228.crt -  
set_serial 01
```

We now have everything needed to connect the WiConnect module to the python server.

Starting the Python TLS Server

To start the server, run the `echo_tls_server.py` script, supplying the server IP address as an argument:

```
~$ python echo_tls_server.py 192.168.1.228
```

NOTES

- The server IP address supplied must be the same IP address used when generating the server certificate.
 - The server certificate, and the server certificate key (in this example `192.168.1.228.crt` and `192.168.1.228.key`) must be in the same directory as the `echo_tls_server.py` file.
-

Loading the CA cert onto the Module (using WiConnect)

Next we'll load the CA cert onto the ACKme module. Open `my_ca.pem` in a text editor such as `Notepad++` and determine the size of the file (number of characters, including whitespace and carriage returns). Issue the following WiConnect command:

```
> file_create my_ca.pem <file size>
```

Then copy and paste the contents of `my_ca.pem` into the serial terminal. This will write the CA certificate to the flash file system on the module. It is critical that the line endings of the file are correct. The .pem format expects a single \n (0x0A) line terminator, and a single \n character at the very end of the file.

Connecting to the Python TLS Server

We're now ready to connect the ACKme module to the python TLS echo server. Assuming WiConnect already has valid credentials for the local Wi-Fi network, issue the following command (don't forget to specify the port number 3000 and to append `my_ca.pem`):

```
> tls_client 192.168.1.228 3000 my_ca.pem
```

If everything is working, WiConnect responds with text similar to:

```
Resolving host: 192.168.1.228
[2014-11-03 | 05:58:34: Opening: 192.168.1.228:3000]
Connecting (TLS): 192.168.1.228:3000
[2014-11-03 | 05:58:34: Opened: 0]
0
```

Issue the WiConnect command (to write to the python server):

```
> write 0 15
Hello World!!!
```

You should see Hello World!!! in the python server terminal on the PC.

Issue the WiConnect command:

```
> read 0 1000
```

Hello World!!! should be returned.

Scripts

The TLS server is created with a Python script:

- [echo_tls_server.py](#)

The following example BASH scripts are provided to help automate the OpenSSL parts of the process described above if required.

- [generate_client_cert.sh](#)
 - [generate_self_signed_ca.sh](#)
 - [generate_server_cert.sh](#)
 - [openssl.conf](#)
 - [revoke_certificate.sh](#)
-

Notes for Windows & Additional Links

To run this example on Windows, the following software packages are needed:

- [MingGW](#) - Minimum GNU for Windows (aka Linux tools for Windows). After install, don't forget to add **MinGW & msys** to your path. Here's an example of where these directories are located after installation in the default location: C:\MinGW\bin, C:\MinGW\msys\1.0\bin
- [OpenSSL](#) - executable installer for Windows
- [Python](#) - for Windows and other operating systems (be sure to install Python 2.7)

Change Log

Modified	Changes	WiConnect Version Required
2014-11-28	Created	1.0+

Broadcast UDP Packet

ACKme WiConnect modules broadcast module properties in JSON format. The properties can be sent either as UDP packets to a UDP host or by a post request to an HTTP host. Properties include by default the IP address and the MAC address.

See the [broadcast](#) variables.

This application note demonstrates how to send and receive a broadcast UDP packet, and how to extract properties from the JSON format data.

This application uses an ACKme module and a Python Script (Python v2.7).

The ACKme module is programmed with the [factory default](#) configuration.

The Python script listens for UDP packets and validates any packet received as JSON data. It then extracts the IP address from the JSON data.

Setup

The following assumes that the module starts in its default state. To return it to the default state, perform a [factory reset](#).

The python script runs on a test computer on the same Wi-Fi network as the ACKme module.

Copy the following python script to the test computer.

[broadcast_udp_packet.py](#)

The Python script procedure is:

1. While no IP address has been successfully parsed:
 - Wait until the server receives a broadcast UDP packet from the module
 - When a packet is received, parse the packet for the module's IP address
 - If the packet is parsed as valid JSON and the module IP address is received, return.
2. Display the JSON packet and the IP address
3. Go to 1

To break out of the Python script loop, press Ctrl-C (Command-C on a Mac.)

Method

Open a WiConnect Terminal to the ACKme Module

To open a WiConnect terminal connection to the ACKme module, see [Getting Started](#).

Associate Module to Wi-Fi Network

To associate the module to a Wi-Fi network, set the values of [wlan.ssid](#) and [wlan.passkey](#) and run the [network_up](#) command. See [Getting Started](#).

Issue the following commands to join the network automatically and reduce the broadcast UDP interval to 2 seconds:

Module WiConnect commands	Comments
set wlan.auto_join.enabled 1	Join network automatically on reboot
set broadcast.interval 2	Set broadcast interval to 2 seconds
save	Save variables
reboot	Reboot to restart the periodic UDP broadcast service

The module automatically broadcasts UDP packets every 2 seconds after association to an AP has completed.

Run Python Script

Execute the `broadcast_udp_packet.py` python script. The script displays the following text:

Listening for ACKme module broadcast UDP packets...

Within the broadcast interval of 2 seconds, the python script prints a result similar to the following:

Broadcast UDP Packet (json format):

```
{
    "mac"      : "4C:55:CC:10:06:12",
    "bssid"    : "30:85:A9:E7:9C:B0",
    "channel"  : 11,
    "ip"       : "192.168.1.173",
    "ssid"     : "ackme",
    "rss"      : -28,
    "time"     : 1416801734637,
    "version"  : "WiConnect-beta-2.0.1.5,
                  Built:2014-11-06 16:02:35 for AMW004.3,
                  Board:AMW004-E03.2",
    "remote_terminal_port" : 2000
}
```

IP found in broadcast UDP packet: 192.168.1.173

Change Properties in Broadcast UDP Packet

At the WiConnect prompt, enter the following command to change the properties in the broadcast UDP packet:

Module WiConnect Commands	Comments
set broadcast.data ip,mac	Set broadcast properties to IP and MAC address
save	Save variables
reboot	Reboot to restart the periodic broadcast service

The python script now prints text similar to:

```
Listening for ACKme module broadcast UDP packets...
```

```
Broadcast UDP Packet (json format):
```

```
{
    "mac" : "4C:55:CC:10:06:12",
    "ip"   : "192.168.1.173"
}
```

```
IP found in broadcast UDP packet: 192.168.1.173
```

Note that only the mac (module MAC address) and ip (module IP address) properties are broadcast.

Set the IP Address to the Test Computer

The module can be configured to send the broadcast UDP packet to a fixed IP address other than the broadcast address of 255.255.255.255. At the WiConnect prompt, enter the following command to set the target IP for the broadcast UDP packet.

Module WiConnect Commands	Comments
set broadcast.udp.ip <IP address of test computer> save reboot	Restrict broadcast to test computer Save variables Reboot to use updated IP address

The python script continues to print the same result as in the previous step.

Set IP address to Target Other than Test Computer

Set the broadcast.udp.ip variable to some address on your network other than your test computer.

Module WiConnect commands	Comments
set broadcast.udp.ip <not ip of test computer> save reboot	Restrict broadcast to other host Save Reboot

The python script stops printing results, because it never sees the broadcast UDP packet.

Now, while the script is running, follow the steps in [Set the IP Address to Test Computer](#). Within two seconds of the ACKme module network coming up, the script begins to display the extracted IP address again.

Send ADC Values and GPIO Values in the Broadcast UDP Packet

Specify ADCs by ADC number, not by GPIO. See [Peripherals](#) for tables showing ADC numbers on the Wallaby/Mackerel evaluation board and the Numbat/Moray evaluation board.

On the Mackerel evaluation board, the thermistor is on ADC0 (GPIO7). On the Moray evaluation board, the thermistor is on ADC9 (GPIO20).

GPIO0 corresponds to Button 1 on both the Mackerel and Moray evaluation boards.

Module WiConnect commands	Comments
set broadcast.data ip,mac,adc0,adc9,gpio0 save reboot	Add ADC0, ADC9, GPIO0 to broadcast properties Save Reboot

The python script displays a result similar to the following:

Listening for ACKme module broadcast UDP packets...

Broadcast UDP Packet (json format):

```
{
    "mac"      : "4C:55:CC:10:06:12",
    "ip"       : "192.168.1.173",
    "gpio0"    : 0,
    "adc0"     : 1845,
    "adc9"     : 1513
}
```

IP found in broadcast UDP packet: 192.168.1.173

ADC values in the packet are in decimal. To convert to mV, see [Peripherals, ADCs](#). If you are using a Mackerel or a Moray evaluation board, press and continue to hold Button 1. The python script displays a result similar to the following:

Listening for ACKme module broadcast UDP packets...

Broadcast UDP Packet (json format):

```
{
    "mac"      : "4C:55:CC:10:06:12",
    "ip"       : "192.168.1.173",
    "gpio0"    : 1,
    "adc0"     : 1845,
    "adc9"     : 1513
}
```

Release Button 1 and the value for gpio0 returns to 0.

Change Log

Modified	Changes	WiConnect Version Required
2014-11-28	Created	1.0+

High Speed UART

This application note demonstrates how to achieve a serial data rate of 400KBytes/s or more between two ACKme modules running WiConnect.

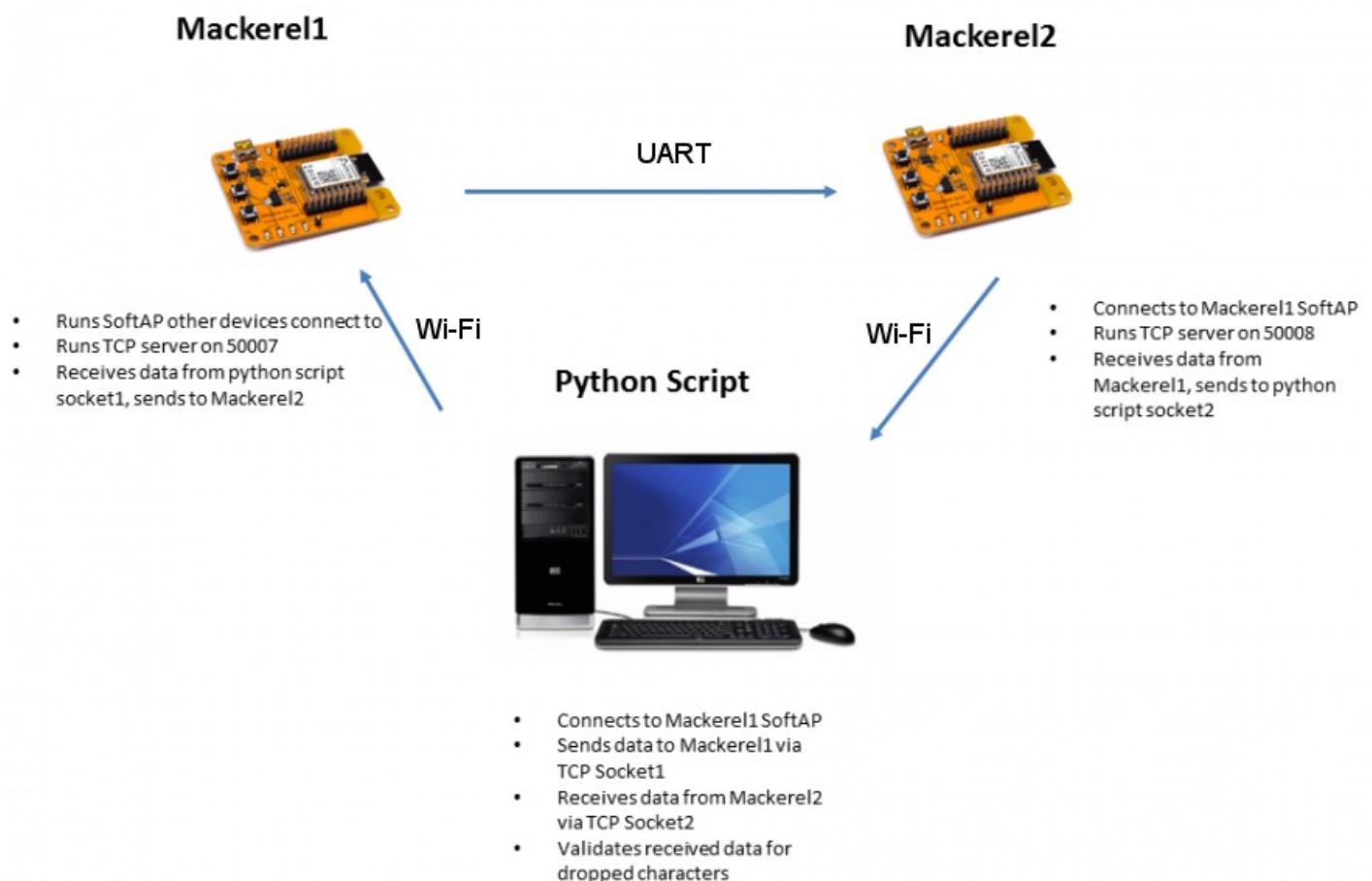
This application uses two ACKme modules and a Python Script running on a PC.

- Module 1 and 2 are configured to use the WLAN interface and the TCP server feature.
- Both modules run in 'stream' mode and are connected to each other via UART.
- The Python Script uses separate TCP connections to the TCP server running on each module.
- The modules and the PC running the python script all connect to the same wireless network.

The flow of data is as follows:

- The Python script transmits data to Module 1 via Wi-Fi (TCP socket)
- Module 1 forwards data to Module 2 via UART
- Module 2 forwards data to Python script via Wi-Fi (TCP socket)
- Python script validates received data and calculates throughput.

The diagram below shows the flow of data:



Module 1 - Settings

WiConnect Commands (Module 1)		Description
set uart.flow	1 on	<- UART1 hardware flow control
set uart.baud	1 1000000 raw	<- UART1 baud rate of 10Mbit/s
set bus.mode	stream	<- Streaming data mode
set bus.data_bus	uart1	<- UART1 is the data bus
set bus.log_bus	uart0	<- UART0 is the logging bus (optional)
set bus.stream.flush_time	0	<- Disable the flush timer (allows for data zero-copy)
set bus.stream.flush_count	1460	<- Maximum flush packet size
set network.buffer.size	55000	<- Network buffer size (max of 70k)
set network.buffer.rxtx_ratio	85	<- 85% of network buffer used for RX packets
set wlan.ssid	<YOUR SSID>	<- Update with name of your network
set wlan.passkey	<YOUR PASSWORD>	<- Update with password to your network
set wlan.auto_join.enabled	1	<- Automatically join network on reboots
set remote_terminal.enabled	1	
set remote_terminal.port	50007	<- Enable remote configuration terminal <- Port remote terminal listens on
set tcp.server.auto_start	1	
set tcp.server.port	40007	<- Automatically start TCP server when network starts <- Port TCP server listens on
set broadcast.udp.interval	10	
set broadcast.udp.port	55555	<- Broadcast module info every 10s <- Broadcast module info on this port
set broadcast.udp.data		<- Broadcast remote terminal's port and module's IP
remote_terminal_port,ip		
set ntp.enabled	0	<- Disable Network Time Protocol (NTP) &
set wlan.rssi_average	0	<- Disable WLAN RSSI averaging to avoid interrupting the onboard processor
save		
reboot		<- Save settings <- Reboot module to start with new settings

Module 2 - Settings

WiConnect Commands (Module 2)		Description
set uart.flow	1 on	<- UART1 hardware flow control
set uart.baud	1 1000000 raw	<- UART1 baud rate of 10Mbit/s
set bus.mode	stream	<- Streaming data mode
set bus.data_bus	uart1	<- UART1 is the data bus
set bus.log_bus	uart0	<- UART0 is the logging bus (optional)
set bus.stream.flush_time	0	<- Disable the flush timer (allows for data zero-copy)
set bus.stream.flush_count	1460	<- Maximum flush packet size
set network.buffer.size	55000	<- Network buffer size (max of 70k)
set network.buffer.rxtx_ratio	15	<- 85% of network buffer used for TX packets
set wlan.ssid	<YOUR SSID>	<- Update with name of your network
set wlan.passkey	<YOUR PASSWORD>	<- Update with password to your network
set wlan.auto_join.enabled	1	<- Automatically join network on reboots
set remote_terminal.enabled	1	<- Enable remote configuration terminal
set remote_terminal.port	50008	<- Port remote terminal listens on
set tcp.server.auto_start	1	<- Automatically start TCP server when network starts
set tcp.server.port	40008	<- Port TCP server listens on
set broadcast.udp.interval	10	<- Broadcast module info every 10s
set broadcast.udp.port	55555	<- Broadcast module info on this port
set broadcast.udp.data		<- Broadcast remote terminal's port and module's IP
remote_terminal_port,ip		
set ntp.enabled	0	<- Disable Network Time Protocol (NTP)
set wlan.rssi_average	0	& <- Disable WLAN RSSI averaging to avoid interrupting the onboard processor
save		<- Save settings
reboot		<- Reboot module to start with new settings

Notes for Settings

- The module receiving data from the network (Module 1) has its `network.buffer.rxtx_ratio` set HIGH so a larger percentage of the network buffer is allocated to the RX packet pool.
- The module transmitting data to the network (Module 2) has its `network.buffer.rxtx_ratio` set LOW so a larger percentage of the network buffer is allocated to the TX packet pool
- For both modules:
 - The `bus.stream.flush_time` is set to 0. This allows for 'zero-copy' of memory between the network interface and the bus
 - UART hardware flow control is enabled. This is *absolutely* essential so data bytes are not dropped
 - The `remote terminal` is enabled. This allows remote configuration of the modules via Wi-Fi. Use a standard telnet client to connect and issue WiConnect commands
 - The WiConnect `UDP broadcast` feature enables the python script to dynamically determine the IP address of each module

Using the Module 1 Soft AP

You can connect the ACKme WiConnect devices and computer via any available Wi-Fi network. As the diagram above suggests, it is also possible to use the soft AP on Module 1 to create a wireless network between the modules and the PC. To use this setup (rather than a standalone Wi-Fi Access Point), add the following to the [Module 1 Settings](#) listed above:

WiConnect Commands (Module 1) - enables use of Module 1 softAP

```
set softap.auto_start      1
set network.default_interface softap
set broadcast.udp.interface default
set tcp.server.auto_interface default
```

Set the Module 2 `wlan.ssid` and `wlan.passkey` according to the Module 1 settings for `softap.ssid` and `softap.passkey`. IP addresses are allocated to connected devices by the Module 1 [soft AP DHCP server](#), if running, according to the setting for `softap.ip`. Connect your computer to the Module 1 soft AP before running the Python script.

Setup Instructions

To run this example, follow these steps.

Update Settings

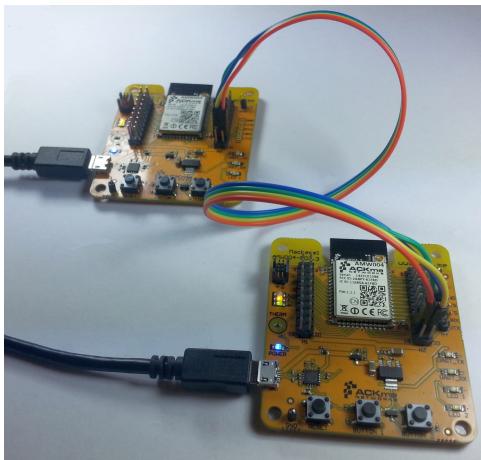
Copy and paste the settings above into a terminal connected to each respective module. Note that after rebooting the module, the module will no longer respond to terminal commands since hardware flow control is enabled and the UART baud rate is set to 10Mbit/s.

Use the `remote terminal` if a setting needs to be updated or `factory reset` the module using the external GPIO.

Connect UARTs

Connect the module UART signals. The signals are connected with Rx/Tx crossover and RTS/CTS crossover, as follows. Header pins shown are for Mackerel boards:

Module 1 UART1 Signal	Module 1 H2 Pin	Module 2 UART1 Signal	Module 2 H2 Pin
RX	18	TX	17
TX	17	RX	18
RTS	16	CTS	15
CTS	15	RTS	16



Run Python Script

Copy the following python script to your computer.

[highspeed_uart.py](#)

Execute the python script. You should see the following

Listening for module UDP broadcast packets...

The Python script procedure is:

1. Wait until the server receives a UDP broadcast packet from each module
2. When a packet is received, parse the packet for the respective module's IP address
3. Connect to the TCP server of each module
4. Start transmitting and receiving data
5. Display unexpected characters received
6. Display upload/download data throughput

Script output is similar to the following:

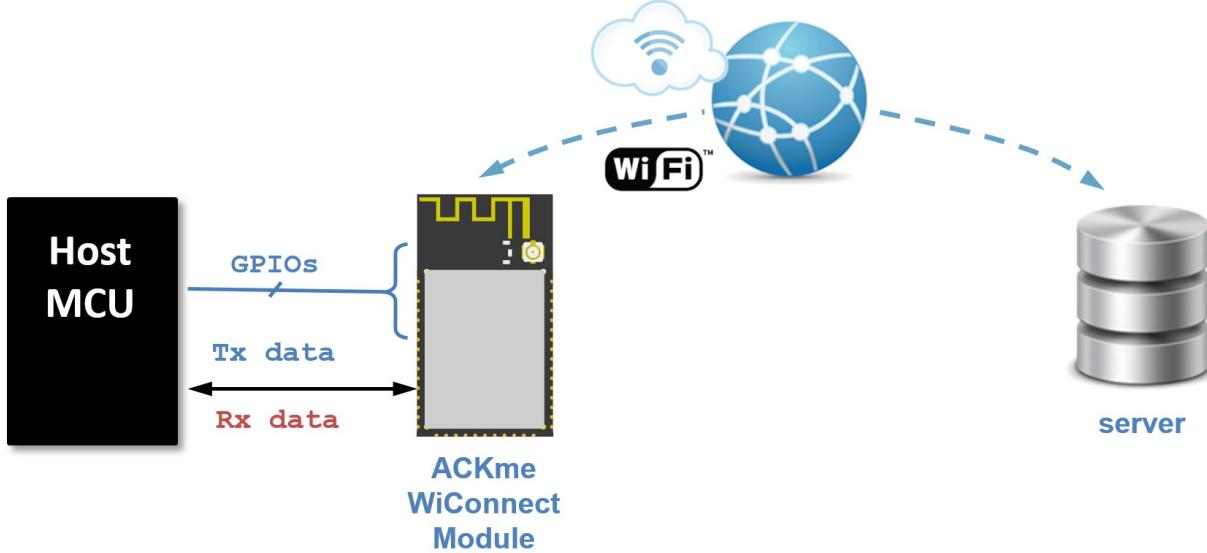
```
Bytes sent: 7.790MB
Bytes received: 7.774MB
TX Rate: 622.414KB/s
RX Rate: 615.221KB/s
Invalid char Found: 0
```

Change Log

Modified	Changes	WiConnect Version Required
2014-11-28	Created	2.0+

GPIO-Controlled Network Connection

This example demonstrates how to use a GPIO to control a network connection to a remote server. Once the network connection is open, data can be automatically streamed between WiConnect and the remote server.



Features Demonstrated

- `ioconn.*` variables
- `gpio_alias` variable

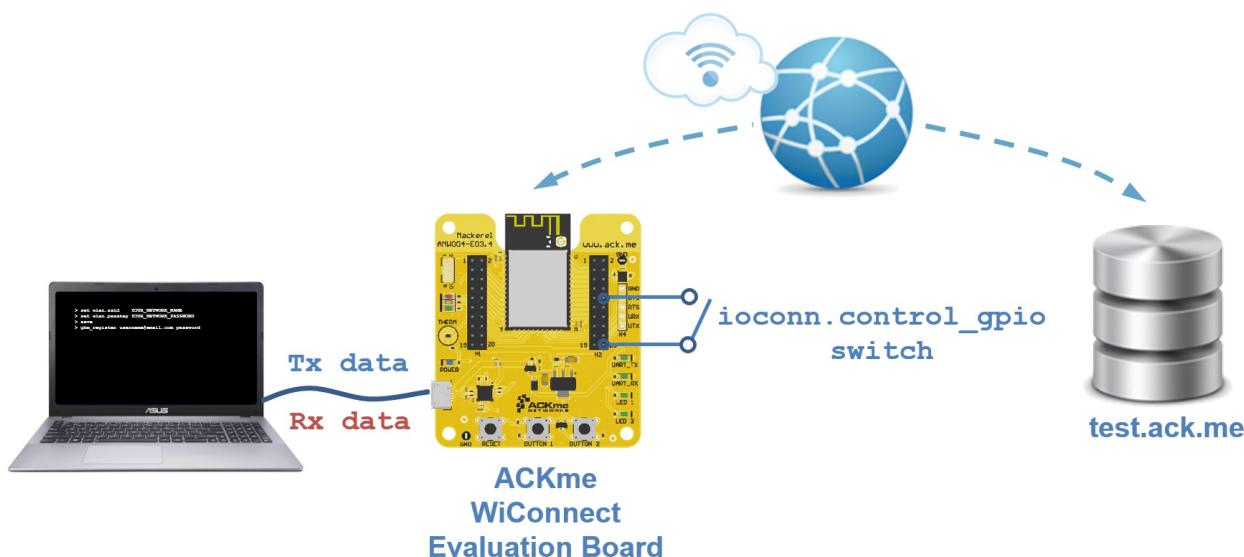
Method

The WiConnect `ioconn` feature makes it easy for an MCU, or blackbox electronic system with a serial output, to add network communication with a remote server. A GPIO is assigned as the `ioconn.control_gpio`. Toggling the `ioconn` control GPIO makes and breaks the network connection to the remote server.

The remote server used in the demonstration is the ACKme test TCP echo server, located online at `test.ack.me`. The TCP echo server simply echoes any messages back to the client.

A computer, connected via USB to the evaluation board, takes the part of the MCU host system.

To set the level of the `ioconn` control GPIO high, various techniques are possible. In this demonstration, a switch is used to connect a logic high (3.3V) to the `ioconn` control GPIO pin. In an actual implementation, the host system toggles the `ioconn` control GPIO.



This demonstration uses the user LED GPIOs on the module for the ioconn control and status GPIOs. The LEDs are configured so that:

- User LED1 turns on when the ioconn control GPIO is asserted
- User LED2 turns on when a connection with the remote server is established

This demonstration works on either a Mackerel or Moray evaluation board.

Setup

Connect the switch between the following pins on the module, depending on your evaluation board:

Mackerel AMW004-E03		Moray AMW006-E03	
Function	Header Pin	Function	Header Pin
VDD_3V3	H2-20	VDD_3V3	H7-35
GPIO22 (LED1)	H2-10	GPIO16 (LED1)	H8-24

Connect the module via USB to your computer and open a serial terminal console. See [Getting Started](#).

The following assumes that the module is in the default state. To return to the default state, perform a [factory reset](#).

Set up the local Wi-Fi network parameters.

WiConnect commands	Comments
set wlan.ssid <YOUR_SSID>	Set the wlan parameters for your local Wi-Fi access point
set wlan.passkey <YOUR_PASSKEY>	Remember to substitute correct values for your local Wi-Fi
set wlan.auto_join.enabled 1	Set up automatic network join

Set up aliases for the **ioconn** variables.

For the AMW004-E03 Mackeral evaluation board:

WiConnect commands	Comments
set gpio.alias 22 IOC_CTRL set gpio.alias 21 IOC_STATUS	Set the alias 'IOC_CTRL' for GPIO22 (user LED1) Set the alias 'IOC_STATUS' for GPIO21 (user LED2)

For the AMW006-E03 Moray evaluation board:

WiConnect commands	Comments
set gpio.alias 16 IOC_CTRL set gpio.alias 13 IOC_STATUS	Set the alias 'IOC_CTRL' for GPIO16 (user LED1) Set the alias 'IOC_STATUS' for GPIO13 (user LED2)

Configure the **ioconn** variables.

WiConnect commands	Comments
gdi IOC_CTRL none	Required if IOC_CTRL GPIO already in use
set ioconn.control_gpio IOC_CTRL	Set the GPIO to initiate connection
set ioconn.enabled 1	Enable GPIO-controlled connection
set ioconn.local_port 5000	Specify local port
set ioconn.protocol tcp	Specify protocol
set ioconn.remote_host test.ack.me	Specify host to connect to
set ioconn.remote_port 50007	Specify port on host
gdi IOC_STATUS none	Set the connection status GPIO
set ioconn.status_gpio IOC_STATUS	
save	Save
reboot	Reboot - changes are initiated at boot time

Performing a GPIO-Controlled Host Connection

Assert the module IOC_CTRL GPIO by setting the switch to ON.

If all goes well, the module terminal console displays something like the following:

WiConnect Response
> Resolving host: test.ack.me
[2014-11-21 03:40:51: Opening: test.ack.me:50007]
> Connecting (TCP): 107.170.222.80:50007
[2014-11-21 03:40:52: Opened: 0]

While the IOC_CTRL GPIO is still asserted, send a message to the server:

WiConnect commands	Comments
write 0 16 hello IoT world!	Send message to stream 0 (server). Specify length of message Enter the message on the next line

WiConnect Response
Success

Now check if a message has been received in reply:

WiConnect commands	Comments
read 0 1000	Read up to 1000 bytes from stream 0 (tcp test server)

WiConnect Response
hello IoT world!

The server's response echoes the message sent.

De-assert the ioconn control GPIO by setting the switch to OFF.

The WiConnect response is similar to the following:

WiConnect Response
> [2014-11-21 03:42:13: Closed: 0]

Using Stream Mode

A typical use of the GPIO Controlled Network Connection feature is to allow a device to stream data to and from a server. To demonstrate this, put the module in stream mode:

WiConnect commands	Comments
set bus.mode stream save reboot	Set the bus mode to stream Save Reboot to bring up in stream bus mode.

The module terminal console can no longer send commands to WiConnect. Try typing at the terminal. There is no response. Characters you type are transparently streamed to the network, which is currently not connected. See [Serial Interface, Stream](#)

Mode.

Now assert the ioconn control GPIO by setting the switch to on:

WiConnect Response

```
> Resolving host: test.ack.me
[2014-11-21 | 03:40:51: Opening: test.ack.me:50007]
> Connecting (TCP): 107.170.222.80:50007
[2014-11-21 | 03:40:52: Opened: 0]
```

All characters received from the remote server appear automatically on the console.

Now type some characters into the module terminal console. Typed characters are echoed (after round-trip network latency) by the ACKme test server.

Change Log

Modified	Changes	WiConnect Version Required
2014-11-28	Created	2.0+

goHACK.me Slave Mode

This example demonstrates how to use the WiConnect goHACK.me API in slave mode with a PC terminal acting as the host (instead of a microprocessor).

By the time you have completed this demonstration, you will be able to ...

- Connect your ACKme device with goHACK.me (and setup a goHACK.me account if you don't already have one)
- Send samples of data streams to goHACK.me for device monitoring
- Receive control parameters from goHACK.me for device control
- Send messages between your device and goHACK.me for messaging

Factory Reset

To get started, let's factory reset your board to ensure that any existing configuration does not get in the way of the demo. Use the commands shown below as a guide to factory reset the board now.

WiConnect Commands	Description
get wlan.mac factory_reset <MAC address>	<- Get the wlan MAC address for your device <- Factory reset the device; paste the MAC address returned by 'get wlan.mac'

If your board is already activated with goHACK.me, we need to deactivate in order to setup a new caps file. Sign in to goHACK.me and deactivate it now. If this is your first time connecting a board to goHACK.me, don't worry, we'll soon have you setup and ready to go.

Connect to Wi-Fi

Connect your board to your Wi-Fi network using the **network_up** command as shown below.

WiConnect Commands	Description
network_up -s	<- Scan & select your network, then enter the network password

Create a Caps File

Create a demo caps file by copy and pasting the following text into the terminal. This is a simple caps file that only has 1 stream and 1 control, and sets up message boxes to enable messaging with goHACK.me.

WiConnect Commands	Description
file_create slave_caps.json 783 { "model": "Demo 1.0",	<- Create the capabilities file and write it to the file system

```

    "title": "gHm Slave
Demo",
    "streams": [
        {
            "slug":
"stream1",
            "title": "My
Stream",
            "unit": "",
            "symbol": "",
            "type": "integer"
        }
    ],
    "controls": [
        {
            "slug":
"control1",
            "title": "My
Control",
            "value": 0,
            "type": "select",
            "options": {
                "0": "0-Red",
                "1": "1-
Green",
                "2": "2-Blue"
            }
        }
    ],
    "messages": [
        {
            "slug": "from-
device",
            "title": "From
Device"
        },
        {
            "slug": "to-
device",
            "title": "To
Device"
        }
    ]
}

```

<- Streams are listed here

<- This caps file has 1 stream called "streams1"

<- This caps file has 1 control called "control1"

<- This caps file has messages enabled

<- The "from-device" box holds messages from the device to goHACK.me

<- The "to-device" box holds messages for the device from goHACK.me

Set the Caps File

Use the **ghm_capabilities** command to set the caps file for use with goHACK.me as shown below.

WiConnect Commands	Description
ghm_capabilities set slave_caps.json	<- Set the caps file to use with goHACK.me

Activate

If you do not already have an account on the goHACK.me website, go to [goHACK.me](#) and create a free account. Activate the board with goHACK.me by using the **ghm_activate** command.

WiConnect Commands	Description
ghm_activate <email@address.com> <yourpassword>	<- Requires a gHm account

Open a web browser and connect to [goHACK.me](#), and login. Select the Device tab in the vertical column at the left of the webpage and your device appears.

Write Streams & Read Controls

Now for some fun! Enter the following commands one-by-one, taking note of the instructions in the right-hand column.

WiConnect Commands	Instructions
get ghm.status get ghm.capabilities ghm_write stream1 239 ghm_sync ghm_read control1 ghm_sync ghm_read control1	1. Verify your device is activated with goHACK.me 2. Verify the name of the capabilities file in use 3. Write the value '239' to stream1 4. Synchronize with goHACK.me; the value of Stream 1 changes on goHACK.me 5. Read the value of control1; it is the same as the value on goHACK.me 6. On the goHACK.me website, change the value of 'My Control' 7. Synchronize with goHACK.me (push stream samples and pull controls) 8. Read the updated value of control1

Send a message to goHACK.me

WiConnect Commands	Description
<pre>ghm_message post stream_write 0 5 Hello stream_write 0 7 there! stream_close 0</pre>	<pre><- Get ready to post a message to goHACK.me <- Send the start of the message "Hello" <- Send the rest of the message " there!" <- Finish & close out the message</pre>

Read a message from goHACK.me

To read messages from goHACK.me, you first have to use goHACK.me to send a message. Setup a message entry board on goHACK.me as follows ...

- Select a dashboard
- Click the + in the top right corner
- Select “gHm Slave Demo”
- Select “Messages” | “To Device”
- Select “Enter Message”
- Enter a name for the message panel (or use the default)
- Select “Done”

In the message panel that appears, enter a text message and click “send”. Use the following commands to read the message. The WiConnect stream API is used to actually read text from the message.

WiConnect Commands	Description
<pre>ghm_message list ghm_message get 0 stream_read 0 100</pre>	<pre><- Get a list of any messages waiting to be read <- Get #0 message; opens with stream handle 0 <- Read up to 100 bytes of stream handle 0</pre>

For completeness, the following text provides an example session log of reading a message from goHACK.me.

WiConnect Session Log - Reading a Message			
> ghm_message list			
Request GET /api/token/messages/to-device/index			
Connecting (https): api.gohack.me:443			
Starting TLS			
HTTP response: 200			
! # Message ID	Timestamp	Length	
# 0 0ee16f11-bff0-42b6-b022-9f93a7b4c768	1407160356842	0x0018	
> ghm_message get 0			
Request GET /api/token/messages/to-device/0ee16f11-bff0-42b6-b022-9f93a7b4c768/body/text			
Connecting (https): api.gohack.me:443			
Starting TLS			
HTTP response: 200			
[2014-08-04 14:10:41: Opened: 0]			
0			
> stream_read 0 100			
Hello from goHACK.me!			
[2014-08-04 14:10:45: Closed: 0]			

Change Log

Modified	Changes	WiConnect Version Required
2014-10-20	Created	1.2+

goHACK.me Solo Mode

This example demonstrates how to use WiConnect in solo mode to work with goHACK.me. The example is based on an ACKme Mackerel board, but the process to setup solo mode is generic and may be used with any ACKme device capable of working with goHACK.me.

In solo mode, WiConnect provides a virtual connection between GPIOs on your device and goHACK.me. GPIO inputs to your device are automatically monitored and can be viewed on the goHACK.me cloud. You can also use the goHACK.me cloud to change the value of GPIOs to enable cloud control of devices.

There are only a couple of steps required to configure this wireless magic.

1. Create/choose a caps file that tells goHACK.me about your device, including the name & type of GPIOs available
2. Configure GPIOs and solo mode variables using WiConnect

For convenience, we provide example caps files for all ACKme devices suitable for use with goHACK.me. For each example caps file, we also provide a corresponding solo mode setup script to configure GPIOs and setup solo mode for use with the caps file. You don't have to use our example caps files of course, but they are great examples that demonstrate how to setup your board with goHACK.me. Different caps and setup files can enable different features of the device. Feel free to create alternative caps files and/or setup files. How you use goHACK.me in solo mode is entirely up to you!

Caps & Solo Mode Setup Files

The default example caps file for each ACKme board is called **ghm_capabilities.json**. For reference, the default caps file & solo mode setup file together with a circuit schematic for an AMW004-E03.3 'Mackerel' board are available for download from the following links.

- [Default Example Caps file for Mackerel](#)
- [Default Example Solo mode setup script for Mackerel](#)
- [Mackerel Circuit Schematic](#)

Alternative caps and solo mode setup files demonstrate how to enable the Mackerel DAC:

- [PWM and DAC example Caps file for Mackerel](#)
- [PWM and DAC example Solo mode setup script for Mackerel](#)

The Caps File Explained

The default example Mackerel caps file is configured with four streams and four controls. It also specifies two message boxes for 2-way messaging with goHACK.me. For more details of caps file format, see [goHACKme Device Capabilities](#). A brief explanation of each of the streams, controls and message boxes follows.

Streams	
button1	<- This stream is used to monitor when Button 1 is pressed
button2	<- This stream is used to monitor when Button 2 is pressed
thermistor	<- This stream is used to monitor the voltage on a thermistor. The voltage responds to temperature changes.
gpio23	<- This stream is used to monitor the digital state of GPIO 23 which is connected to Header 2 Pin 7 on the Mackerel board
Controls	
led1	<- This control is used to turn LED 1 on or off from goHACK.me
led2-duty	<- This control is used to set the duty cycle of a pulse-width modulation (PWM) signal connected to LED 2. The PWM duty cycle is used to change the brightness of the LED.
led2-freq	<- This control is used to set the frequency of a pulse-width modulation signal connected to LED 2. The PWM frequency is used to change how often the LED is turned on and off.
gpio6	<- This control is used to set the digital output state of GPIO 6 which is connected to Header 1 Pin 9 on the Mackerel board
Messages	
from-device to-device	<- This message box is used to send messages from the device to goHACK.me <- This message box is used to send messages to the device from goHACK.me

The Solo Mode Setup File Explained

The corresponding default example Mackerel solo mode setup file configures the Mackerel board to autonomously connect the streams & controls to goHACK.me. It also enables solo mode and sets the solo mode sync period.

The file is formatted with each line as a comma separated list. In each line there are three items:

command, arguments, comment

As it runs each successive line of the script WiConnect emits the comment first, then executes the command with its arguments. The default setup file contains the following commands:

Free up GPIOs that will be used with goHACK.me	
set gpio.init 0 none set gpio.init 11 none set gpio.init 22 none set gpio.init 21 none set ghm.solo.gpio all none	<- Free up GPIO configured for user button 1 <- Free up GPIO configured for user button 2 <- Free up GPIO configured for user LED 1 <- Free up GPIO configured for user LED 2 <- Free previously configured goHACK.me GPIOs
Assign GPIOs to Streams	
set ghm.solo.gpio 0 button1 in rising set ghm.solo.gpio 11 button2 in rising set ghm.solo.gpio 7 thermistor adc set ghm.solo.gpio 23 gpio23 inpd	<- Setup Button 1 on GPIO 0 as a rising edge triggered input <- Setup Button 2 on GPIO 11 as a rising edge triggered input <- Setup Thermistor on GPIO 7 as an ADC input <- Setup GPIO 23 = H2 Pin 7 as a digital input
Assign GPIOs to Controls	
set ghm.solo.gpio 22 led1 output set ghm.solo.gpio 21 led2- duty pwm led2-freq set ghm.solo.gpio 6 gpio6 output	<- Setup LED 1 on GPIO 22 for digital control <- Setup LED 2 on GPIO 21 for control by PWM <- Setup GPIO 6 = H1 Pin 9 digital control
General Config	
set ghm.solo.sync_period 4 set ghm.solo.enabled 1 set system.print_level 3 save	<- Set goHACK.me auto sync period for solo mode <- Enable solo mode for standalone operation. <- Turn off verbose prints which are not needed in solo mode <- Save configuration

Let's go SOLO!

For reference, the variables used to setup solo mode are described here ...

- `gpio.init`
- `ghm.solo.gpio`
- `ghm.solo.sync_period`
- `ghm.solo.enabled`
- `system.print_level`

Let's go SOLO!

Now that you have a basic understanding of what is in a caps file and how WiConnect is configured to work in solo mode with goHACK.me, let's get your board connected and working with goHACK.me.

Factory Reset

Firstly, you should factory reset your board to ensure that any existing configuration does not get in the way of the demo. Use the commands shown below as a guide to factory reset the board now.

WiConnect Commands	Description
<code>get wlan.mac</code> <code>factory_reset <MAC address></code>	<- Get the wlan MAC address for your device <- Factory reset the device; paste the MAC address returned by 'get wlan.mac'

If your board is already activated with goHACK.me, we need to deactivate in order to setup a new caps file. Sign in to goHACK.me and deactivate it now. If this is your first time connecting a board to goHACK.me, don't worry, we'll soon have you setup and ready to go.

Connect to Wi-Fi

Connect to your board to your Wi-Fi network using the `network_up` command as shown below.

WiConnect Commands	Description
<code>network_up -s</code>	<- Scan & select your network, then enter the network password

Solo Mode Setup

Download the default example caps & setup file and run the setup script using the `ghm_capabilities` command as shown below. The `ghm_capabilities` command actually downloads the caps file & the setup file, sets the caps file for use with goHACK.me, and runs the setup file as a script; it's a very handy command.

WiConnect Commands	Description
<code>ghm_capabilities download -s</code>	<- Download the default caps file and setup for solo mode

Let's go SOLO!

If you have your own caps & solo mode setup files and do not want to use example files provided by ACKme, skip the above command and use the following procedure instead ...

1. Write your caps file to the file system using the WiConnect `file_create` command
2. Set the caps file for use with goHACK.me using the `ghm_capabilities set` command
3. Configure goHACK.me solo mode variables 1-by-1, or write a setup script and save it to the file system, then run it using the `setup cmd` command.

Activate

If you do not already have an account on the goHACK.me website, go to [goHACK.me](#) and create a free account. Activate the board with goHACK.me by using the `ghm_activate` command.

WiConnect Commands	Description
<code>ghm_activate <email@address.com> <yourpassword></code>	<- Requires a gHm account

Open a web browser and connect to [goHACK.me](#), and login. Select the Device tab in the vertical column at the left of the webpage and your device appears.

Monitoring & Control with goHACK.me

With your device connected, press button 1 on your board and within a few seconds, you should see the corresponding button 1 on goHACK.me change to 1. Now try clicking the LED 1 switch on goHACK.me, within a few seconds, the LED on your board will turn on just like magic.

Change Log

Modified	Changes	WiConnect Version Required
2014-10-20	Created	1.2+

This demonstrates an end-to-end solution, using only ACKme hardware, WiConnect firmware and sensors.com (SDC) software, for controlling one hardware device by another via the cloud. No coding is required. All the devices need is a Wi-Fi internet connection. They can be on opposite sides of the earth.

Features demonstrated

- goHACK.me triggers



Method

This application uses two ACKme evaluation boards to demonstrate goHACK.me triggers. The buttons on one evaluation board control LED 1 on the other board. Both devices operate in solo mode.

Set up both modules for goHACK.me solo mode, and activate with your free goHACK.me account. On the goHACK.me site, set up a trigger on Module 1 Button 1, on change, to turn on the Module 2 LED 1 on. Set up a trigger on Module 1 Button 2, on change, to turn on the Module 2 LED 1 off. Press Button 1 on Module 1, and Module 2 LED 1 turns on. Press Button 2 on Module 1, and Module 2 LED 1 turns off.

Procedure

First, set up a free goHACK.me account, and for each device download the goHACK.me solo caps file and activate the device with the account. See [Getting Started with goHACK.me](#).

Editing Device Details

Go to the goHACK.me site and log in. The Devices tab shows the two Mackerel boards.

The screenshot shows the goHACK.me Devices page with two Mackerel boards listed. The left sidebar has navigation links: 1, 2, 3, 4, Triggers, Devices, Account, and Editing. The main area shows 'Devices' at the top, followed by 'Mackerel 1' and 'Mackerel 2'. Each board section includes an image, a timestamp ('in a few seconds' or '2014-12-10 13:02:50 +11:00'), and two buttons: 'Clear' and 'Deactivate'. Below the timestamp are four status cards:

BUTTON 1 button1	BUTTON 2 button2	GPIO 23 = H2 PIN 7 gpio23	THERMISTOR thermistor
Off	On	20	1862

Below the first row of cards are two more status cards:

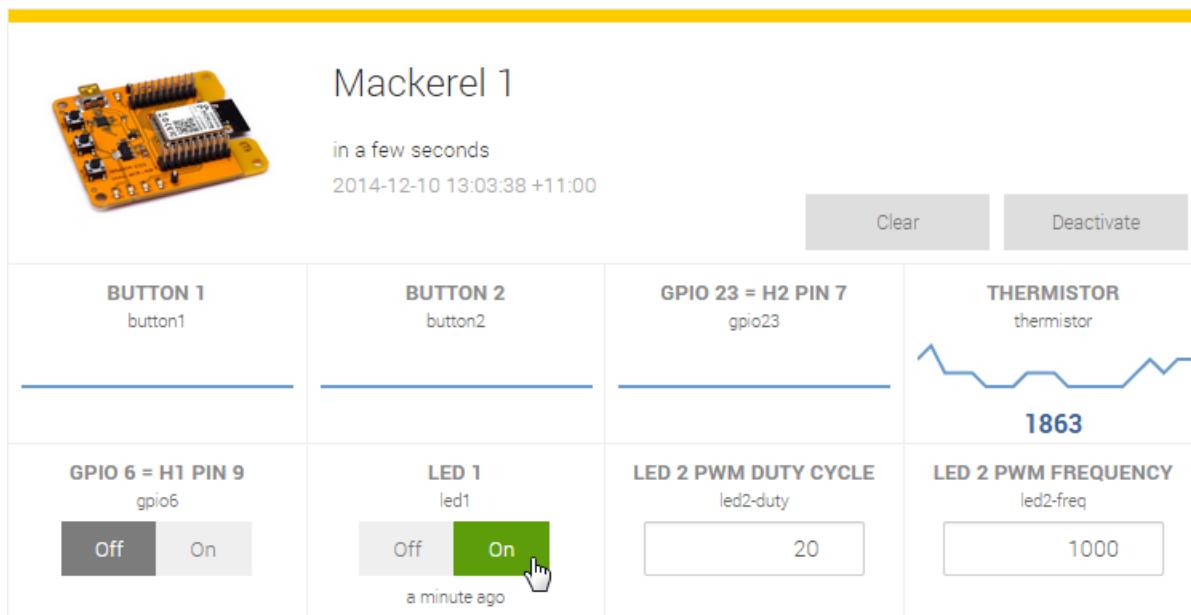
GPIO 6 = H1 PIN 9 gpio6	LED 1 led1	LED 2 PWM DUTY CYCLE led2-duty	LED 2 PWM FREQUENCY led2-freq
Off	On 9 minutes ago	20	1000

The 'Mackerel 2' section follows a similar layout.

Click the title of the first device and change it to Mackerel 1. Change the title of the second device to Mackerel 2.

Verifying goHACKme Connections

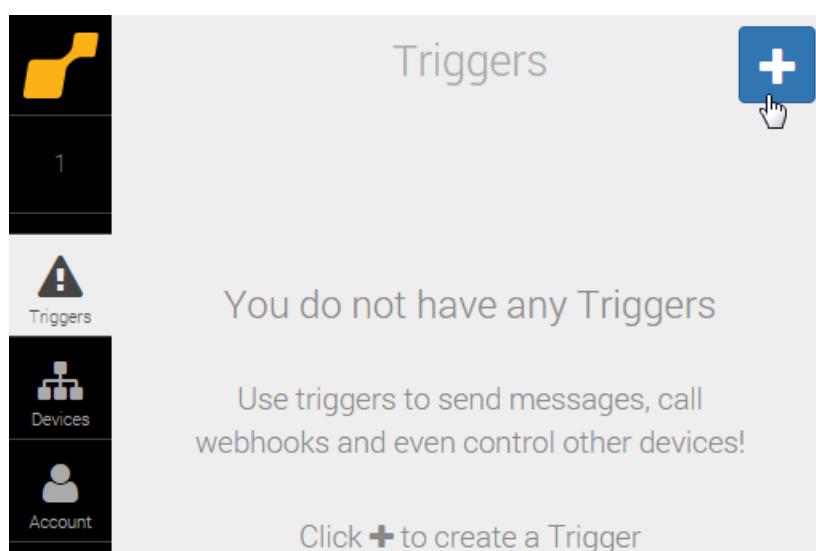
To verify that the goHACK.me connection is working, on the Devices tab, switch the Mackerel 1 LED 1 to On. After a network latency of up to 5 seconds, LED 1 on your Mackerel 1 device switches on. Switch LED 1 off on the Devices tab to verify that it turns off. Allow for network latency.



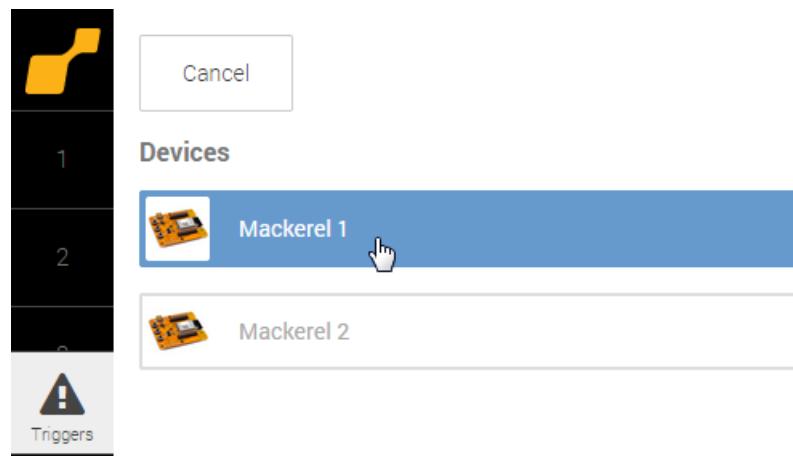
Similarly, verify the Mackerel 2 goHACK.me connection.

Adding a Trigger

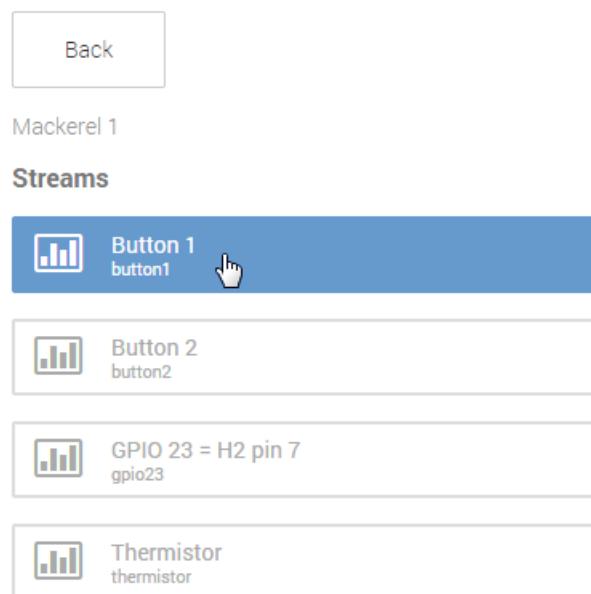
On the Triggers tab, click the + button to add a trigger.



In Devices, select the Mackerel 1 board.



In Streams, select Button 1.



For Condition, select Change.

[Back](#) [Done](#)

Mackerel 1 - Button 1

Title

Mackerel 1 - Button 1

 Mackerel 1

 Button 1
button1

Condition

 Change

 ↑ Rising Edge

 ↓ Falling Edge

Click Done.

The Mackerel 1 Button 1 Trigger appears on the Triggers tab.

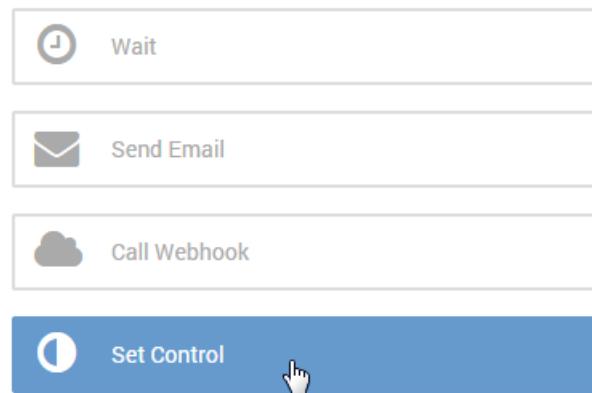
Triggers [+](#)

	Mackerel 1 - Button 1	Disable Enabled
Mackerel 1	Button 1	
+ Add action		
Reset trigger after		 30s  15m  1h  4h  8h  24h

Adding an Action to a Trigger

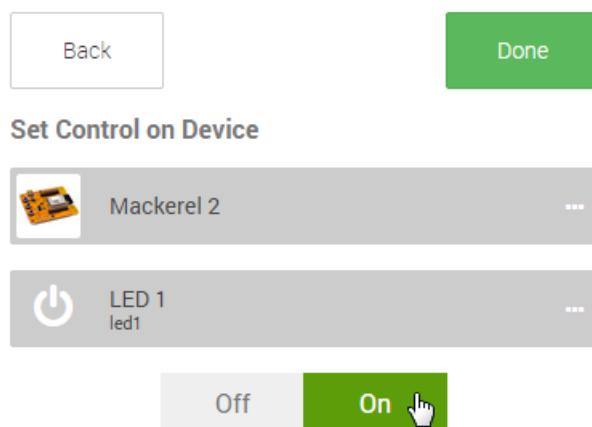
Click the + Add action button on the Trigger

For Actions, select Control.

Actions

In the next steps for Set Control on Device, select Mackerel 2, then LED 1, then On.

Click Done.



The Action appears on the Trigger as: On Mackerel 2 set control LED 1 to 1.

Mackerel 1 - Button 1

Triggered by: On Mackerel 2 set control LED 1 to 1

Add action

Reset trigger after: 1h

Similarly, create a trigger on Mackerel 2, Button 2 Change, with the action: On Mackerel 2, select Control LED 1 to 0.

On both Triggers, select Reset trigger after: 30s.

Select Enabled on both Triggers.

The screenshot shows the WiConnect Triggers interface with two triggers listed:

- Mackerel 1 - Button 2**: Triggered by Mackerel 1 Button 2 Change. Action: On Mackerel 2 set control LED 1 to 0. Status: Enabled. Reset trigger after: 30s.
- Mackerel 1 - Button 1**: Triggered by Mackerel 1 Button 1 Change. Action: On Mackerel 2 set control LED 1 to 1. Status: Enabled. Reset trigger after: 30s.

The triggers are now set up. Both triggers show the green lightning stroke Ready icon.

Triggering Actions

Press Button 1 on Mackerel 1, and after a few seconds the Mackerel 1 Button 1 trigger shows the red alert Triggered icon. A few seconds after that, LED 1 on the Mackerel 2 lights up.

Triggers

Mackerel 1 - Button 1

 Mackerel 1 Button 1 

Actions: On Mackerel 2 set control LED 1 to 1

Add action

Reset trigger after: **30s** 15m 1h 4h 8h 24h

Mackerel 1 - Button 2

 Mackerel 1 Button 2 

Actions: On Mackerel 2 set control LED 1 to 0

Add action

Reset trigger after: **30s** 15m 1h 4h 8h 24h

Press Button 2 on Mackerel 1, and after a few seconds the Mackerel 1 Button 2 trigger shows the red alert Triggered icon. A few seconds after that, LED 1 on the Mackerel 2 turns off.

Triggers

Mackerel 1 - Button 1

Mackerel 1 | Button 1 |

Disable Enabled

On Mackerel 2 set control LED 1 to 1

+ Add action

Reset trigger after 30s 15m 1h 4h 8h 24h

Mackerel 1 - Button 2

Mackerel 1 | Button 2 |

Disable Enabled

On Mackerel 2 set control LED 1 to 0

+ Add action

Reset trigger after 30s 15m 1h 4h 8h 24h

Note that the minimum trigger reset is 30 seconds. After 30 seconds, the trigger resets and shows the Ready icon again. Until the trigger is reset and ready, pressing the button does not trigger the associated actions.

Change Log

Modified	Changes	WiConnect Version Required
2015-02-03	Created	1.2+

goHACK.me Nest Integration

Features demonstrated

- goHACK.me Nest Integration

Method

This application uses an ACKme evaluation boards to demonstrate Nest integration. A button on the evaluation board triggers a Nest thermostat to switch to Heat/Cool mode.

To follow through these steps, you require a Nest account.

Set up the module for goHACK.me solo mode, and activate with your free goHACK.me account.

From your goHACK.me account, authorize goHACKme integration with Nest.

Then set up a trigger on Module 1 Button 1, on change, to switch a Nest thermostat to Heat/Cool mode.

Press Button 1 on the module 1, and the Nest thermostat enters the specified mode.

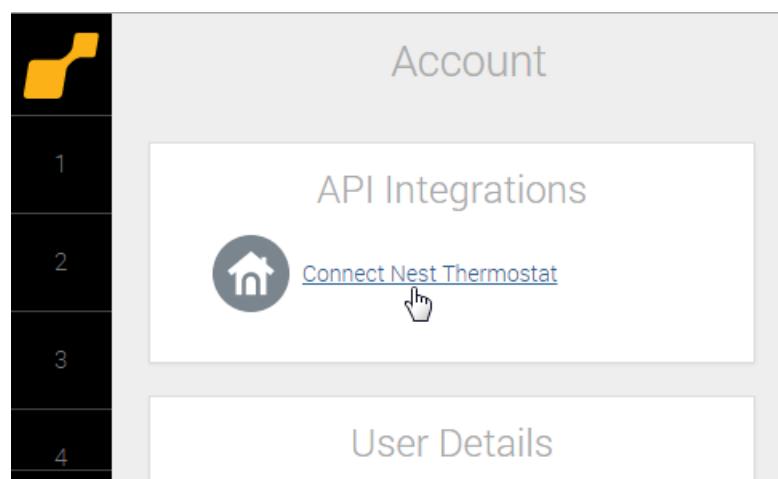
Procedure

First, set up a free goHACK.me account, download the goHACK.me solo caps file and activate the device with the account. See [Getting Started with goHACK.me](#).

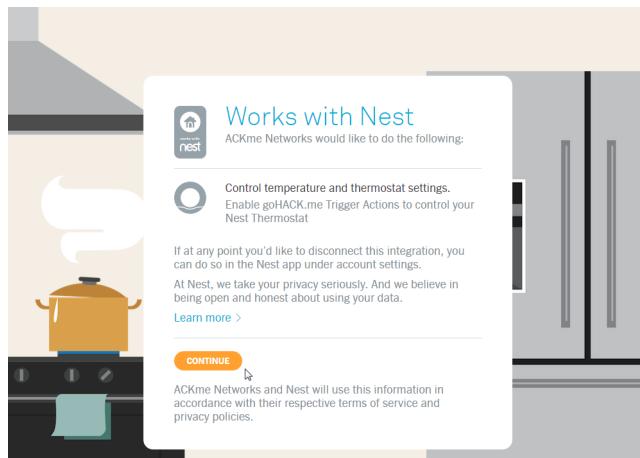
Go to the goHACK.me site and log in.

Authorizing goHACK.me with Nest

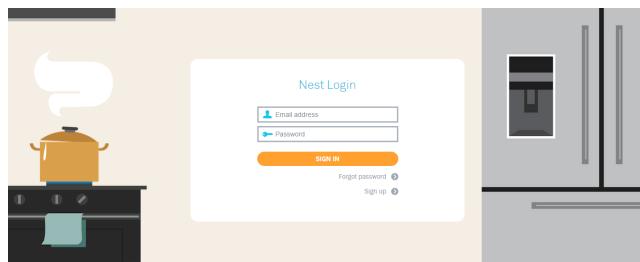
Go to the Account tab and click the Connect Nest Thermostat link.



The Nest site displays the goHACK.me authorization request. Click CONTINUE to authorize your goHACK.me account with Nest.

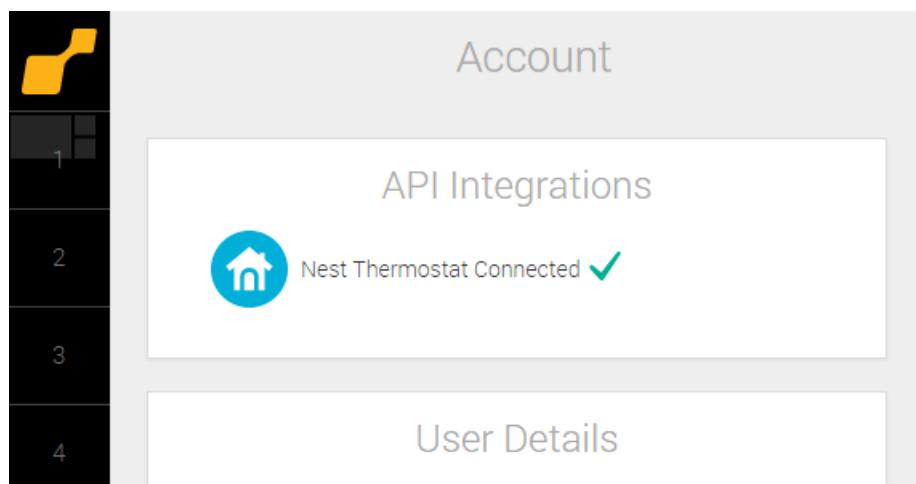


If you are not already logged in to Nest, provide your Nest login details.



Once you have provided the authorization, the Nest account provides goHACK.me with the authorization token to control your Nest devices. You don't have to authorize with Nest again after this point.

You're automatically redirected back to your goHACK.me account page.



The API Integrations panel now shows the Nest connection is authorized.

Create a Trigger with a Nest Thermostat Action

Go to the goHACK.me Triggers tab. Create a trigger for button 1 change. For details of this procedure see [goHACK.me Triggers, Adding a Trigger](#).

The screenshot shows the 'Triggers' section of the WiConnect interface. A trigger named 'Mackerel 1 - Button 1' is listed. It is associated with 'Mackerel 1' and 'Button 1'. The status is 'Enabled'. There is a 'Disable' button and a blue '+' button in the top right. Below the trigger name, there is a 'Reset trigger after' dropdown menu with options: 30s, 15m, 1h (selected), 4h, 8h, and 24h. A blue bar at the bottom has a '+ Add action' button with a cursor icon pointing to it.

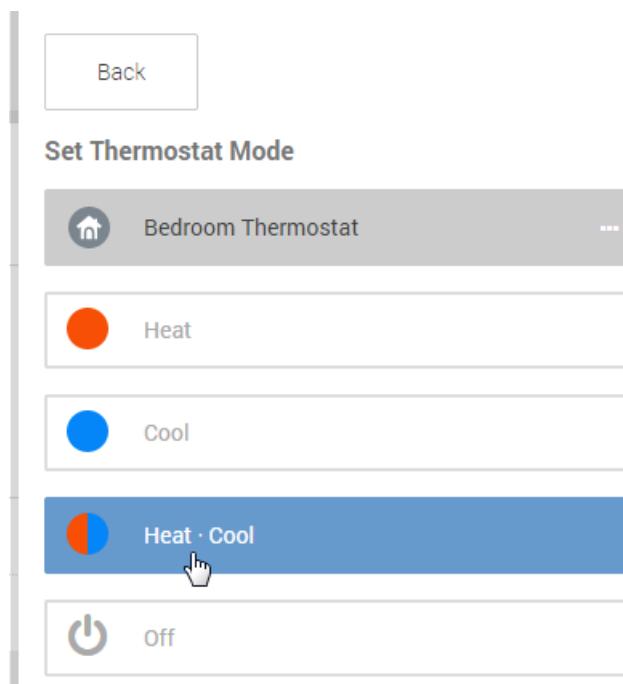
After creating the Trigger on button 1 change, click Add Action. From the list of available actions, choose Control Nest Thermostat.

This screenshot shows a list of available actions. At the top is a 'Cancel' button. Below it is a heading 'Actions'. Four actions are listed: 'Wait' (with a clock icon), 'Send Email' (with an envelope icon), 'Call Webhook' (with a cloud icon), and 'Set Control' (with a gear icon). At the bottom is a blue button labeled 'Control Nest Thermostat' with a house icon, which has a cursor icon pointing to it.

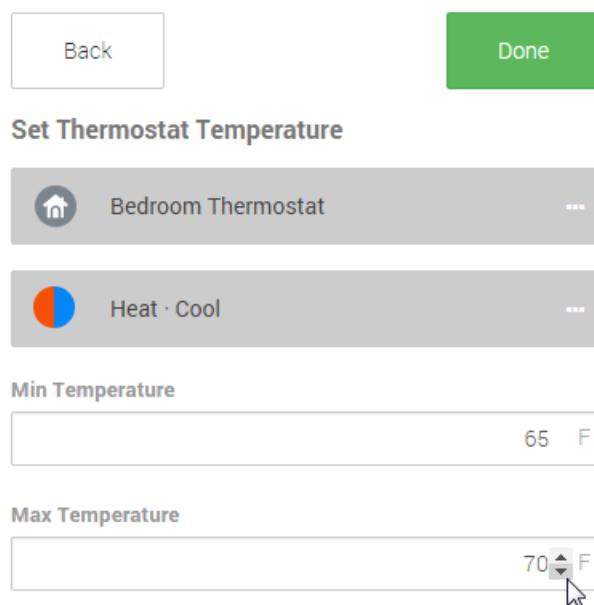
Choose an available thermostat from your Nest thermostats.

This screenshot shows a list of available thermostats. At the top is a 'Back' button. Below it is a heading 'Control Thermostat'. Two thermostats are listed: 'Bedroom Thermostat' (selected, highlighted in blue) and 'Kitchen Thermostat' (not selected).

Choose the Heat . Cool action for the thermostat.

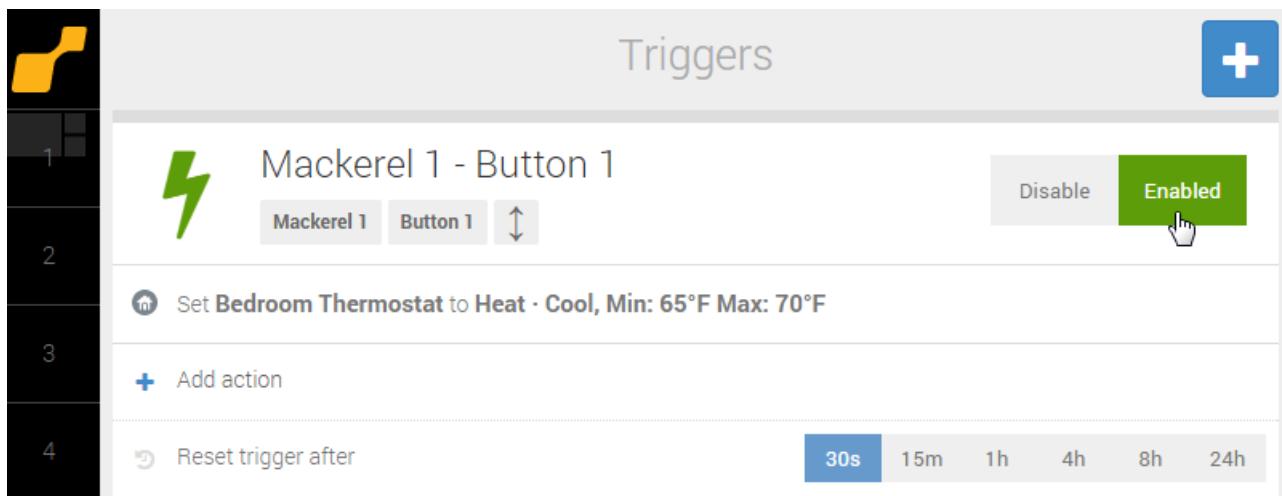


Set the desired maximum and minimum temperatures.



Click DONE.

The trigger now shows the Nest thermostat action. Click Enabled to enable the trigger.



Trigger the Nest Thermostat Action

Press Button 1 on your ACK.me device to trigger the Nest thermostat action.

Change Log

Modified	Changes	WiConnect Version Required
2015-02-03	Created	1.2+

Device Discovery and HTTP RESTful API

This example uses the HTTP RESTful API to interact with the module using HTTP requests. Commands are sent to the module using HTTP REST methods such as GET and POST and a simple request syntax.

Using HTTP requests, you can send any command to the module and receive the response. The [WiConnect Web App](#) offers a full demonstration of the HTTP RESTful API. This application note demonstrates how to use the basic requests of the RESTful API.

Other ways of managing and interacting with an ACKme module include:

- WiConnect Terminal - via physical connection to USB port (see [Getting Started, Opening a WiConnect Terminal](#))
- Wi-Fi Remote terminal - Telnet connection via Wi-Fi (see [Wi-Fi Remote Terminal](#) application note.)
- For more sophisticated scripting, the [WiConnect JavaScript API](#) provides a JavaScript wrapper around the HTTP server REST API
- A WiConnect Python API providing a Python wrapper around the HTTP server REST API - coming soon.

This demo uses the following WiConnect features:

- mDNS discovery - to make it easy to find the module on the local network. See [Networking and Security, Network Discovery](#).
 - WiConnect RESTful API - for executing WiConnect commands directly via HTTP requests. See [Networking and Security, HTTP server with RESTful API](#)
-

This example demonstrates how to:

1. Setup the module to connect to your local network
 2. Start the WiConnect mDNS service and HTTP web server
 3. Use a web browser or cURL to send commands to the module, and receive responses from WiConnect
-

Setup

To run this example follow these steps:

Connect Your Module to the Local Network

Since you'll need to configure mDNS and the web server, the quickest way to setup the module is to use a WiConnect terminal - see [Getting Started](#). Once you have a terminal connected, issue the following command to configure the credentials for the local network:

```
> network_up -s
```

Enable the mDNS and HTTP Server

Enable mDNS and specify the domain the module will respond to using the following commands:

```
set mdns.enabled 1
set mdns.name mymodule
```

Enable the WiConnect HTTP server and the RESTful API using the following WiConnect commands:

```
set http.server.enabled      1
set http.server.api_enabled 1
```

Save the settings, then restart the network:

```
save
network_down
network_up
```

The module automatically starts the mDNS discovery service and the HTTP server on boot after connecting to the network.

The response from WiConnect is similar to the following:

```
> Obtaining IPv4 address via DHCP
IPv4 address: 10.5.6.59
Starting mDNS
mDNS domain: mymodule.local
HTTP and REST API server listening on port: 80
[2015-01-15 | 23:48:42: Associated]
```

Syntax of REST API WiConnect Commands

The format of a REST API WiConnect command GET request is:

```
GET http://mymodule.local/command/<command>
```

For example, to issue the ver command:

```
GET http://mymodule.local/command/ver
```

To issue a GET request from a browser, enter the URL <http://mymodule.local/command/ver> into the browser URL address box.

The format of a REST API WiConnect command POST request is:

```
POST http://mymodule.local/command/<command>
```

followed immediately by a json formatted structure of the form:

```
{
  "flags"    : <flags>,
  "command"  : "<wiconnect command>",
  "data"     : "<command data>"
}
```

The “data” value is required by commands that expect additional data sent immediately after the command, such as `file_create`. Some of the curl examples below demonstrate how to issue an HTTP POST request.

Sending Commands from a Web Browser

Using your favorite web browser (with a network client such as a PC, phone or tablet connected to the same local network as the module), enter the following URL to retrieve the WiConnect version string, or click the following link <http://mymodule.local/command/ver> if this computer is connected to the same network as the module:

The browser responds with the WiConnect version embedded in a JSON object, similar to the following text.

```
{"id":18,"code":0,"flags":0,"response":"WiConnect-2.1.0.0, Built:2015-01-15 ... }
```

Troubleshooting

If you are having problems making this work, try the following

- omit the .local domain in the URL, and use <http://mymodule/command/ver> instead.
- replace mymodule.local with the IP address of the module (available by reading the WiConnect IP address variable get network.ip). The issue may be related to the operation of mDNS which has some operating system caveats. See [Networking & Security, Network Discovery](#)

How does this work?

When the WiConnect command is entered into the web browser address box, the browser translates the URL into an HTTP GET request and sends the request to the module name. The operating system in your PC (or phone / tablet) translates the module name into the module IP address and sends the request to the module. On the module, the HTTP web server receives and processes the request, and then provides a response back to the server.

A few more examples are shown below. For full documentation on the WiConnect HTTP REST API, see [Networking & Security, WiConnect REST API](#).

Read a list of files on the module:

URL	http://mymodule.local/command/ls
WiConnect Response	<pre>{"id":2,"code":0,"flags":0,"response":"! # Size Version Filename\r\n# 0 41741 2.1.0 command_help.csv\r\n# 1 135 2.1.0 default_setup.script\r\n# 2 1897 2.1.0 favicon.ico.gz\r\n# 3 1236 2.1.0 geotrust_ca.pem\r\n# 4 212736 2.1.0 upgrade_app.exe\r\n# 5 215204 2.1.0 wiconnect.exe\r\n# 6 38004 2.1.0 wiconnect_webgui.css.gz\r\n# 7 1827 2.1.0 wiconnect_webgui.html\r\n# 8 61492 2.1.0 wiconnect_webgui.js.gz\r\n# 9 210412 5.26.230 wifi_fw.bin\r\n"}</pre>

Read the raw value of the ADC connected to the thermistor on the ACKme [Mackerel eval board](#) or [Moray eval board](#):

URL	http://mymodule.local/adc_7 (Mackerel - GPIO 7) http://mymodule.local/adc_20 (Moray - GPIO 20)
WiConnect Response	{"id":104,"code":0,"flags":0,"response":"0x835\r\n"}

Open a TCP connection to Google:

URL	http://mymodule/command/tcp_client google.com 80
WiConnect Response	{"id":4,"code":0,"flags":0,"response":"0\r\n"}

Read the response to the TCP client connection above:

URL	http://mymodule.local/log
WiConnect Response	{"logs":[{"HTTP and REST API server listening on port: 80","[2015-01-15 - 01:09:23: Associated]\r\n","Resolving host: google.com","[2015-01-15 - 01:28:53: Opening: google.com:80]","Connecting (TCP): 216.58.220.142:80","[2015-01-15 - 01:28:54: Opened: 0]"}]}

Using cURL

[Curl](#), a command line URL utility, may also be used to connect to the WiConnect webserver. Curl is typically run from a PC or Linux command line for transferring data with a URL syntax. The following examples show how to use cURL with WiConnect.

In some cases, the arguments to curl in the examples below have been put onto separate lines for readability purposes only. Curl commands should be issued on a single command line.

Tip! If you are having trouble using curl, try using the -v verbose flag.

WiConnect Version

Curl	\$> curl http://mymodule.local/command/version
WiConnect Response	{"id":18,"code":0,"flags":0,"response":"WiConnect-2.1.x.x, Built:2015-01-14 21:34:43 for AMW004.3, Board:AMW004-E03.3"}

Curl used with a URL only issues an HTTP GET request.

Verbose file list

This example reads a verbose list of files on the module and uses the more sophisticated [POST request API](#). Note that the double-quotes may need to be escaped with backslash characters (as shown below) on some operating systems, such as Windows.

Curl used with the -d (-data) option issues an HTTP POST request. Note that the Content-Type header must be supplied to specify that the data posted is in json format.

Curl	\$> curl -H "Content-Type: application/json" -d "{\"flags\":0,\"command\":\"ls -v\"}" http://mymodule.local/command
------	---

WiConnect Response	{"id":3,"code":0,"flags":0,"response":"! # Type Flags Hnd Size Version Filename\r\n# 0 e-FB 0001 82 41741 2.1.0.0 command_help.csv\r\n# 1 e-FD 0001 53 135 2.1.0.0 default_setup.script\r\n# 2 e-FE 0001 81 1897 2.1.0.0 favicon.ico.gz\r\n# 3 e-03 0001 52 1236 2.1.0.0 geotrust_ca.pem\r\n# 4 i-00 801B 0 212736 2.1.0.0 upgrade_app.exe\r\n# 5 i-81 801B 52 215204 2.1.0.0 wiconnect.exe\r\n# 6 e-FE 0001 55 38004 2.1.0.0 wiconnect_webgui.css.gz\r\n# 7 e-FE 0001 54 1827 2.1.0.0 wiconnect_webgui.html\r\n# 8 e-FE 0001 65 61492 2.1.0.0 wiconnect_webgui.js.gz\r\n# 9 e-01 0009 0 210412 5.26.230.12 wifi_fw.bin\r\n"}
--------------------	---

The value of the `response` key is the return data from the command. In this case it is the list of files. This is formatted to be read on a terminal in a monospace font, with new lines where each `\r\n` character sequence appears.

Open and Read a file

Open the `default_setup.script` file located on the WiConnect file system:

Curl	\$> curl -H "Content-type: application/json" -d "{\"flags\":0,\"command\":\"file_open default_setup.script\"}" http://mymodule.local/command
WiConnect Response	{"id":14,"code":0,"flags":0,"response":"0\r\n"}

The value of the `response` key is the return data from the command. In this case it is the stream number for the open file stream, 0, which is used as an argument to the `stream_read` command issued below.

Read the file using the `stream_read` command:

Curl	\$> curl -H "Content-type: application/json" -d "{\"flags\":0,\"command\":\"stream_read 0 300\"}" mymodule.local/command
WiConnect Response	{"id":15,"code":0,"flags":0,"response":"network_up,-s ,Configuration network credentials\r\nset wlan.auto_join.enabled,true,Enable network auto-join\r\nnsave,-,Saving settings\r\n"}

Change Log

Modified	Changes	WiConnect Version Required
2015-02-03	Created	2.1+

Secure HTTP Server

Overview

This example demonstrates how to use WiConnect to create a secure **TLS** HTTPS server using a self-signed TLS certificate. Any web browser may then securely connect to the HTTPS server. If required, a client certificate may be used to enable bi-directional client and server authentication.

The following steps are covered by this example:

- Generate TLS certificates using OpenSSL
 - Load a TLS certificate into the module and client web browser.
 - Configure the module to enable the HTTPS server
 - Connect to the HTTPS server with a web browser and view a secure webpage.
-

Command Line Conventions

The procedures below for running OpenSSL commands work on Linux, Mac or Windows computers with the correct software packages installed.

The examples shown are for a Linux machine.

Notes are provided where a variation is required for a different platform, and references for Windows in particular are provided in the final section at the bottom of the page.

Command Line Prompt

In the command line examples below, the beginning part `~$` stands for the command prompt. The actual prompt varies depending on your user name, machine name and platform. Enter only the part of the command line following the command prompt.

Command Line Continuation

This procedure requires long command lines. Where command lines are too long to display, a line continuation character `\` is shown at the end of the first line, and the remainder of the command line is shown on the next line. Type the whole command on the same line. Alternatively, on some operating systems you can type the line continuation character followed by `Enter`, and continue the command on the next line.

On a Linux system the line continuation character is a backslash `\` as shown in the examples below. On a MS Windows command line, the line continuation character is a caret `^`.

TLS Certificate Overview

TLS certificates are used to:

- Encrypt data passing between a client and server
- Verify the identity of the remote side of the connection to avoid 'man in the middle' attacks.

See [TLS Client, TLS Certificate Primer](#) for more information.

OpenSSL

For the following steps in which certificates are generated, it is assumed you have OpenSSL installed on your computer and OpenSSL is in the PATH environment variable i.e. you can execute OpenSSL directly from a command line. OpenSSL may be downloaded here: <https://www.openssl.org/related/binaries.html> or on Ubuntu, run the command:

```
~$ sudo apt-get install openssl
```

The manual for OpenSSL may be found here: <https://www.openssl.org/docs/apps/openssl.html>

In some instances, it may be necessary to set the RANDFILE environment variable to avoid OpenSSL errors such as unable to write 'random state'. To temporarily set the RANDFILE environment variable to a local file, use the following command:

```
~$ set RANDFILE=.rnd
```

Generating a Self-Signed CA Certificate

The first certificate to generate is a self-signed Certificate Authority (CA) certificate. This is the root certificate from which all other certificates are derived.

We start by generating the CA key. This is the most sensitive file in your secure TLS system. This key should be kept in a secure location.

```
~$ openssl genrsa -aes256 -out my_ca.key 4096
```

After executing this command OpenSSL prompts for a password to secure the CA key, enter a password (and a confirmation password). This is the same password used for the rest of the process.

Next, create the self-signed CA cert using the key:

```
~$ openssl req -new -sha1 -x509 -key my_ca.key -out my_ca.crt
```

OpenSSL prompts for various values. Press Enter to accept a default value. At the Common Name prompt, you can enter any name, for example my_ca_cert. Here is an example of what to expect (note the Common Name value):

```
~$ openssl req -new -sha1 -x509 -key my_ca.key -out my_ca.crt
Enter pass phrase for my_ca.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
```

Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:my_ca_cert <-- Common Name
Email Address []:

Generating a Server Certificate

Now that we have a CA certificate, we need to generate a TLS server certificate. This is TLS cert used by the WiConnect HTTPS server.

To generate the server certificate, we need to decide on the host name of our server. The host name is the URL entered into a web browser (e.g <https://mymodule.com>). WiConnect has several options for host names.

First we need to decide, is WiConnect connecting to a network or is it creating the network? i.e will the WiConnect 'wlan' or 'softap' interface be used?

If the 'wlan' interface is used, there are two host name options:

- IP Address - this is the address the network gives the module. The web browser URL would look like: <https://192.168.1.12>
- mDNS domain - using the WiConnect mDNS feature to give the module a host name. The URL would look like:
<https://mymodule.local>

If the 'softAP' interface is used, there are also two host name options:

- Static IP Address - this is the address assigned to the module. The web browser URL would look like: <https://10.10.10.1>
- DNS domain - using the WiConnect DNS feature to give the module a host name. The URL would look like:
<https://mymodule.com>

The following examples uses the wlan interface and mDNS. The host name will be: `mymodule.local`.

When generating the server certificate, it is **CRITICAL** the host name is used since the web browser will verify the name. If the server certificate is generated with the host name: `mymodule.local`, the only valid URL is: <https://mymodule.local>. If any other domain is used the web browser will display a warning indicating the certificate is not valid.

NOTE! At every point where `mymodule.local` appears in the procedure below, you should substitute the host name you have chosen for the module.

First generate the private key for our server certificate:

```
~$ openssl genrsa -aes256 -out mymodule.local.key 1024
```

OpenSSL prompts for another password. This can be the same as the password used for the CA cert above (or a different password if you prefer).

Next we need to create a Certificate Signing Request (CSR).

```
~$ openssl req -new -sha1 -key mymodule.local.key -out mymodule.local.csr
```

OpenSSL prompts for various values. Press Enter to accept default values until the **Common Name** prompt.

Important Note! At Common Name prompt enter the host name of the module. At the OpenSSL Challenge password, do NOT enter anything. Just press Enter. Here is an example of what to expect (note the Common Name value):

```
~$ openssl req -new -sha1 -key mymodule.local.key -out mymodule.local.csr
Enter pass phrase for mymodule.local.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:mymodule.local      <-- Common Name
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:                                <-- Leave this BLANK!
An optional company name []:
```

Now, remove the passphrase on the server key so WiConnect may use it:

```
~$ cat mymodule.local.key | openssl rsa -out mymodule.local.key
```

Next we need to generate the server certificate signed by our self-signed CA certificate:

```
~$ openssl x509 -req -sha1 -in mymodule.local.csr -CA my_ca.crt \
    -CAkey my_ca.key -out mymodule.local.crt -set_serial 01
```

Next, convert mymodule.local.crt to .pem format (converting first to an intermediate DER format):

```
~$ openssl x509 -in mymodule.local.crt -outform DER | \
    openssl x509 -inform DER -out mymodule.local.pem -outform PEM
```

The server certificate has now been generated in the required .pem format.

At this point we have everything needed to run the WiConnect module as a Secure HTTPS server (but not with client auth ... yet). Any client (that has a copy of the module root CA certificate, or a cert signed by the module root CA) can connect and authenticate the WiConnect HTTP server. Once authenticated an encrypted connection can be opened between the client and HTTPS server.

This process is identical to the process used when you open a non-password protected https:// site from your web browser. e.g. <https://google.com>. However, instead of a self-signed cert, your web browser uses a CA cert from a trusted third-party company

GeoTrust Global CA. The Google server certificate is signed by the GeoTrust CA cert, and your web browser uses this CA cert (which is contained in a certificate store on your PC) to authenticate the Google HTTPS server.

Authenticating the Client

If you don't want to use client authentication, you may want to skip to the [next section](#).

What if the WiConnect HTTPS server wants to authenticate the client? WiConnect has two options, both of which may be used simultaneously:

1. Use the `http.server.username` and `http.server.password` variables. Using these variables, the web browser displays a prompt asking for credentials before the WiConnect HTTPS server will return a webpage.
2. Use a client TLS certificate. With this option, a certificate is installed into the client web browser. The WiConnect HTTPS server authenticates the client certificate before a webpage is returned.

Each option has its pros and cons.

Option 1:

1. Pro: Much simpler. The client just needs the username/password.
2. Con: Every client uses the same username/password.

Option 2:

1. Pro: Much more secure. Each client has a unique custom certificate which the server authenticates.
2. Con: Much more complicated. A unique certificate must be generated and loaded into each client.

For this example we'll use both (because WiConnect supports both!).

Setting up HTTPS server username and password is described in [HTTP Server Username and Password](#) below.

Generating a Client Certificate and Key

The remainder of this section describes how to generate a client key and certificate.

Decide on a name for the client. For this example we'll use the name: `name@email.com`.

The first step is to generate a certificate key for the client.

```
~$ openssl genrsa -aes256 -out name@email.com.key 1024
```

OpenSSL prompts for another password (and a confirmation). This can be the same as the password used for the CA cert above, or a different password.

Next we need to create a Certificate Signing Request (CSR).

```
~$ openssl req -new -sha1 -key name@email.com.key -out name@email.com.csr
```

OpenSSL prompts for various values. Press Enter to accept default values until reaching the Common Name prompt. At the prompt Common Name: enter the name of the client, name@email.com for this example. When OpenSSL asks for a Challenge password, do NOT enter anything. Just press Enter. Here is an example of what to expect (note the Common Name value):

```
~$ openssl req -new -sha1 -key name@email.com.key -out name@email.com.csr
Enter pass phrase for name@email.com.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:name@email.com      <-- Common Name
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:                                <-- Leave this BLANK!
An optional company name []:
```

Next we generate the client certificate signed by the mymodule.local.crt server certificate:

```
~$ openssl x509 -req -sha1 -in name@email.com.csr -CA mymodule.local.crt \
-CAkey mymodule.local.key -out name@email.com.crt -set_serial 02
```

Finally we create a '.p12' file from the client certificate and key. A .p12 file makes it much easier to install the client certificate into a web browser.

```
~$ openssl pkcs12 -export -clcerts -in name@email.com.crt \
-inkey name@email.com.key -out name@email.com.p12
```

Simply press enter when prompted for an 'Export Password' (twice). This makes the .p12 file non-password protected.

We now have everything needed to setup a Secure HTTPS server with client authentication.

NOTE: The name@email.com.key file is password protected. If you intend to use the .key/.crt files rather than the .p12 file, the password can be removed with the following command:

```
~$ cat name@email.com.key | openssl rsa -out name@email.com.key
```

Loading the Server cert onto the Module (using WiConnect)

Next we'll load the server cert and key onto the WiConnect module. The easiest way to do this is to use the WiConnect Web App provided with WiConnect 2.1 or later. To start the webapp, issue the following command to the module using a WiConnect terminal:

```
> setup web
```

This starts the WiConnect softAP and webserver. Connect your computer to the WiConnect Wi-Fi network (by default the network name is WiConnect-XXX, and the network password is 'password'). Then open a web browser and enter the URL: <http://setup.com>

Wait for the WiConnect webapp to load, then click on the 'Files' tab. The Files tab enables you to drag and drop files from your computer to the file system on the WiConnect module.

On your computer, find the server key and certificate files that you just created:

- mymodule.local.pem
- mymodule.local.key

Drag these files onto the WiConnect webapp where it says 'Drop files here'. Alternately, click the button labelled 'Click to add files'.

That's it! The server key and certificate are now stored in non-volatile memory on the WiConnect module flash file system.

Issue the following command to stop the softAP and webserver:

```
> reboot
```

Run WiConnect as a Secure HTTPS Server

We're now ready to configure the module for:

- Secure HTTPS server using the wlan interface
- mDNS network discovery
- HTTP server username/password
- HTTP server client authentication

First, setup the secure HTTPS server *without* client authentication or verification. Issue the following commands:

WiConnect Commands	Description
set http.server.enabled 1	<- Enable the HTTP server
set http.server.port 443	<- Standard secure HTTP port
set http.server.tls_enabled 1	<- Enable secure (TLS) HTTP server
set http.server.tls_cert mymodule.local.pem	<- Set the server certificate
set http.server.tls_key mymodule.local.key	<- Set the server certificate key
set http.server.max_clients 1	<- (optional) A TLS connection may use lots memory. This variable ensures only one connection is opened

Next let's enable the mDNS feature with the host name 'mymodule' :

WiConnect Commands	Description
set mdns.enabled 1	<- Enable mDNS daemon
set mdns.name mymodule	<- Set the host name to 'mymodule' (note that '.local' is automatically appended)
save	<- Save the settings

Now, put the module on the network. Issue the following command and follow the prompts:

```
network_up -s
```

That's it! The WiConnect module is now running a secure HTTPS server with mDNS support. Go to your web browser (on a computer connected to the same network as the module) and enter the URL: <https://mymodule.local/>

At this point, your web browser will likely display a security warning message. That's good!

Recall that the `mymodule.local.pem` server certificate was signed by the self-signed CA certificate? Your browser doesn't know about the self-signed CA certificate (yet), as a result, it is unable to authenticate the `mymodule.local.pem` certificate.

To fix this, find and double-click on the `my_ca.crt` file. This should display a dialog providing information about the certificate. There should also be a button that says something like 'Install Certificate'. Click the 'Install Certificate' button and follow the dialogs to install the certificate. Make sure the cert is installed in the '**Trusted Root Certification Authorities**' Certificate Store.

Once installed, you may need to restart your browser for the new `my_ca.crt` cert to be recognized as a valid certificate authority. Once recognized, the browser should not display a security warning when browsing to the module at the URL:
<https://mymodule.local>

If everything is working, you are now viewing secure encrypted HTTP pages served from the WiConnect module.

The WiConnect Web App displays by default, but the default webpage can easily be changed by setting the WiConnect variable:

```
set http.server.root_filename <webpage.html>
```

where `<webpage.html>` is the name of the desired root web page located on the WiConnect file system (you need to put your own page on there first!).

Other default pages that can be changed are:

WiConnect Commands	Description
<code>set http.server.notfound_filename <mypage.html></code>	<- Sets the page that appears if the webpage isn't found
<code>set http.server.denied_filename <mypage.html></code>	<- Sets the page that appears if bad credentials are entered

HTTP Server / Client Authentication

As an extra security measure, let's enable client authentication. This forces a client to supply a validate TLS certificate before the HTTP server returns information requested by the client.

NOTE: This is a memory intensive feature. Other settings may need to be adjusted for it to function properly.

To enable client authentication, issue the following command:

WiConnect Commands	Description
<code>set http.server.tls_verify_peer 1</code> <code>save</code>	<- Enable client (i.e. peer) authentication <- Save the new settings

Toggle the network interface to restart the secure HTTPS server (with client authentication enabled):

```
network_down
network_up
```

Open your web browser and enter the URL: <https://mymodule.local/>

The browser displays nothing. This is because the browser is not configured to supply a valid TLS certificate. We can fix this by installing the .p12 certificate generated previously.

To install the certificate, double-click the file name@email.com.p12 on your computer file system. This should bring up a dialog. Follow the dialog to install the cert into your system (the client certificate does not need to go into the Trusted Cert Authority). Again, you may need to restart your browser for the changes to take effect.

After the .p12 certificate is installed, open your web browser and enter the URL:

`https://mymodule.local/`

This time you are prompted for certificate information. Select the newly installed name@email.com certificate.

You should once again see the WiConnect webapp. This time bi-directional client **AND** server authentication is used. Your browser authenticates the WiConnect HTTPS server and the WiConnect HTTPS server authenticates your browser.

HTTP Server Username and Password

Finally, let's also force users to enter a username and password to access the HTTPS server using **HTTP Basic Authentication**. A valid username/password combination must be supplied to gain access to webpages on the module.

To set the username/password, issue the following commands:

WiConnect Commands	Description
<code>set http.server.username admin</code>	<- Sets the HTTP server username
<code>set http.server.password rootpassword</code>	<- Set the HTTP server password
<code>save</code>	<- Save the new settings

Toggle the network interface to restart the HTTP server with the new username/password settings:

`network_down`
`network_up`

Now go to your web browser and enter the URL:

`https://mymodule.local/`

The browser prompts for a username and password, enter the details to match the settings given above: admin/rootpassword

Notes for Windows & Additional Links

To run this example on Windows, the following software packages are needed:

- **MingGW** - Minimum GNU for Windows (aka Linux tools for Windows). After install, don't forget to add **MinGW & msys** to your path. Here's an example of where these directories are located after installation in the default location: C:\MinGW\bin, C:\MinGW\msys\1.0\bin
- **OpenSSL** - executable installer for Windows

Change Log

Modified	Changes	WiConnect Version Required
2015-02-03	Created	2.1+

HTTP Server Simple WebSocket Demonstration

The WiConnect HTTP Server WebSocket support can be accessed from JavaScript without any special libraries.

To set up a WebSocket on your WiConnect device, and connect to it, you use the standard JavaScript WebSocket object, in conjunction with a HTTP RESTful API call to `ws://deviceName/stream` that causes the device to open a WebSocket. For more on the WebSocket API, see [Mozilla WebSocket API](#).

With the socket open, you can send and receive messages between the client and the WiConnect device.

From the web page client end you use the standard JavaScript library to handle WebSocket input and output.

At the WiConnect device end, in command mode, you can read messages with `stream_read` and send messages with `stream_write`. In stream mode, data streams directly to and from the WebSocket.

Setup

Open a WiConnect terminal.

Configure WiConnect Device

Configure as follows:

WiConnect Commands	Description
<code>set wlan.ssid <your_wlan_SSID></code>	Set wlan ssid - substitute your own
<code>set wlan.passkey <your_wlan_password></code>	Set wlan password - substitute your own
<code>set wlan.auto_join.enabled 1</code>	Enable WLAN auto join
<code>set http.server.enabled 1</code>	Enable HTTP server
<code>set mdns.name mymodule</code>	Set mDNS name
<code>set mdns.enabled 1</code>	Enable mDNS
<code>set network.buffer.rxtx_ratio 25</code>	Increase Tx network buffer (Numbat)
<code>save</code>	Save
<code>reboot</code>	Reboot

Note: The `network.buffer.rtx_ratio` setting is necessary only for the Numbat, which has a smaller memory. See [Memory](#).

After rebooting, the WiConnect Terminal displays messages similar to:

```
[Associating to ackme]
> Obtaining IPv4 address via DHCP
IPv4 address: 10.5.6.55
Starting mDNS
mDNS domain: mymodule.local
HTTP and REST API server listening on port: 80
[Associated]
```

Download and Launch the HTML page

Download the HTTP Server Simple WebSocket Demonstration web page: http_server_ws_simple.html.

Run this page in a web browser to act as the client. You can run it as a file:// URL.

The web page has input boxes for the device IP or name, and for a message to send from the client.

Command Mode

By default, the WiConnect device bus mode is command. You issue commands to read from and write to the WebSocket.

In the web page Device IP or Name input box, specify the mDNS name or the IP address of the module. If you used the command sequence above, the mDNS name is the default, mymodule.

Click the Open WebSocket button to open the WebSocket.

The web page session log indicates a successful connection.

In the WiConnect Terminal, WiConnect indicates a connection has been opened with a response similar to:

```
> [2015-02-12 | 06:12:09: Opened: 0]
```

The [stream_list](#) command displays a list of open streams, similar to:

```
> list
! # Type Info
# 0 WEBS 10.5.6.55:80 10.5.6.60:63745
```

In the web page, enter a message in the Message input box and press Enter to send the message.

In the WiConnect Terminal, view the message by issuing the command:

```
read 0 1000
```

The first argument is the stream id, and should be modified depending on the WebSocket stream. See [stream_read](#). The response is similar to:

```
Hello Wiconnect!
```

To send a message back to the web page via the WebSockets, in the WiConnect Terminal issue the command:

```
write 0 5
Hello
```

Press Enter after typing 'write 0 5', then type the text of the message, hello. The first argument is the stream id, and should be modified depending on the WebSocket stream. See [stream_write](#).

The web page message log shows the message received.

WiConnect WebSocket Demonstration

Device IP or Name:

Session Log

WebSocket open to WiConnect device mymodule

Message Log

Hello WiConnect!

Hello

Message (Press Enter to Send)

Stream Mode

Now, in the WiConnect Terminal, switch to stream mode:

```
set bus.mode stream
save
reboot
```

The WiConnect device reboots in stream mode and displays messages indicating the the HTTP server and mDNS are running.

In the web page, open a new WebSocket.

When the WebSocket opens successfully, send a message to the device from the web page. It appears immediately on the WiConnect Terminal console.

Type a message on the WiConnect Terminal console. Depending on terminal echo settings, it may not appear on the terminal, but each character you type is streamed directly to the the web page message log, via the WebSocket connection.

JavaScript

The demonstration web page uses simple JavaScript to establish the WebSocket, and send and receive messages.

The WebSocket is initialized in the `initWebSocket()` function, with the line:

```
ws = new WebSocket('ws://'+ipName+'/stream')
```

In the same function, calls to the WebSocket `onopen`, `onclose` and `onmessage` functions set callbacks for those events.

In the `sendMessage` function, a call to the WebSocket `send` method sends the message to the device.

The rest of the JavaScript is plumbing to handle the user interface.

mDNS Device Discovery

This example demonstrates how to find the module on the network using a network discovery protocol such as mDNS, LLMNR or Netbios (WiConnect includes support for all of these protocols). Network discovery enables you to connect with the module using a known domain, rather than a dynamically allocated IP address.

This example demonstrates how to:

1. Set up the module to connect to your local network
 2. Start the WiConnect mDNS service
 3. Connect to the module via its mDNS domain
 4. Advertise an mDNS service
-

Setup

To run this example follow these steps:

Connect Your Module to the Local Network

To configure mDNS, use a WiConnect terminal - see [Getting Started](#). Once you have a terminal connected, issue the following WiConnect command to configure the credentials for the local network:

```
network_up -s
```

Enable the mDNS Daemon

Enable mDNS and specify the domain advertised by the module using the following commands:

```
set mdns.enabled 1
set mdns.name mymodule
```

Save the mDNS configuration, then restart the network:

```
save
network_down
network_up
```

The module automatically starts the network discovery service after connecting to the network.

The response from WiConnect is similar to the following:

```
WiConnect-2.1.0.x, Built:2015-01-14 21:34:43 for AMW004.3, Board:AMW004-E03.3
[Ready]
[Associating to mynetwork]
> Obtaining IPv4 address via DHCP
IPv4 address: 10.5.6.59
```

```
Starting mDNS
mDNS domain: mymodule.local
[Associated]
```

Ping the Module using the mDNS Domain

From a command line on a Mac or Windows PC connected to the same local network, send an ICMP ping to the module.

```
C:\> ping mymodule.local
Pinging mymodule.local [10.5.6.59] with 32 bytes of data:
Reply from 10.5.6.59: bytes=32 time=308ms TTL=128
Reply from 10.5.6.59: bytes=32 time=8ms TTL=128
Reply from 10.5.6.59: bytes=32 time=4ms TTL=128
Reply from 10.5.6.59: bytes=32 time=96ms TTL=128
```

NOTE: Some ISPs hijack the .local domain which confuses Microsoft Windows, so it may be necessary to omit .local from the name on Windows platforms.

If there is no response (on a Windows PC), try sending the ping directly to the module name instead.

```
C:\> ping mymodule
Pinging mymodule [10.5.6.59] with 32 bytes of data:
Reply from 10.5.6.59: bytes=32 time=33ms TTL=128
Reply from 10.5.6.59: bytes=32 time=17ms TTL=128
Reply from 10.5.6.59: bytes=32 time=5ms TTL=128
Reply from 10.5.6.59: bytes=32 time=29ms TTL=128
```

Advertise an mDNS Service

This example assumes the network credentials are set up as described in [Connect Your Module to the Local Network](#).

To advertise an mDNS service, you need to start a WiConnect server. This example uses the HTTP server. To enable the HTTP server, issue the WiConnect commands:

```
set http.server.enabled      1
set http.server.api_enabled 1
```

Advertise the service using the `mdns.service` variable:

```
> set md s http "Test Server" "record1=on.record2=off.another=21"
```

The server name and TXT record name=value pairs are arbitrary in this case. For specific applications, there may be a required naming convention and format for the TXT record information.

Restart the network to begin advertising:

```
network_down
network_up
```

The response is similar to the following:

```
[Associating to ackme]
In progress
> Obtaining IPv4 address via DHCP
IPv4 address: 10.5.6.82
Starting mDNS
mDNS domain: mymodule.local
Adding mDNS service: Test Server._http._tcp.local
HTTP and REST API server listening on port: 80
[Associated]
```

Note the line Adding mDNS service: Test Server._http._tcp.local.

This means a device capable of browsing for mDNS services will be able to discover your “Test Server” service in the .local domain, among the _http._tcp. services, and read its name and TXT record for additional information.

If you have a Bonjour browser app you can view the module's advertised services.

To disable advertising of the service, set the server's name to -1:

```
set md s http -1
```

Now bring the network down and back up:

```
network_down
network_up
```

The response is similar to the following:

```
[2015-01-29 | 06:44:48: Associating to ackme]
In progress
> Obtaining IPv4 address via DHCP
IPv4 address: 10.5.6.82
Starting mDNS
mDNS domain: nealemodule.local
HTTP and REST API server listening on port: 80
[2015-01-29 | 06:44:51: Associated]
```

Note that the Adding mDNS service line no longer appears.

If you have a Bonjour browser app you can see that the module service is no longer advertised.

Change Log

Modified	Changes	WiConnect Version Required
2015-02-03	Created	2.1+

Recovery from Safemode

This example demonstrates how to recover the module if it enters [Safe Mode](#).

It is extremely important for a host application to handle safe mode operation. In safe mode, the commands available to the host are purposely restricted to simplify the recovery process. See [Upgrade & Recovery](#).

In this example, the `force_safemode` command is used to enter safe mode. This command exists only to allow demonstration of safe mode recovery procedures. In a real-world situation, the module enters safe mode due to repeated faults.

The example works through each of the following steps:

1. Force the module into safe mode
 2. List and clear faults
 3. Return the module to normal operation
-

Force Safe Mode

Connect to the module using a WiConnect terminal - see [Getting Started](#). Once you have a terminal connected, issue the following WiConnect commands to force the module into safe mode. The first command returns the MAC address of the module. The MAC address is required by the `force_safemode` command.

```
> get wlan.mac
<module MAC address>
> force_safemode <module MAC address>
```

In safe mode, the prompt (if enabled) changes to `SAFEMODE>` in [human friendly command mode](#) and is prefixed with S in [machine friendly command mode](#).

Here's an example demonstrating how to force safe mode:

```
> get wlan.mac
4C:55:CC:10:10:98
> force_safemode 4C:55:CC:10:10:98
Forcing safemode...
Rebooting
[Disassociated]
WiConnect-2.1.0.9, Built:2015-01-25 15:58:14 for AMW004.3, Board:AMW004-E03.3

*** Max faults exceeded. Entering Safe Mode.
SAFEMODE>
```

Listing Faults

The module only enters safe mode if a combined total of 8 watchdog faults or hardfaults occur. Each time a fault occurs, key information is logged to non-volatile memory to help ACKme isolate the fault. To obtain a list of faults, use the `faults_print` command

shown in the example below. Note that this particular example does not show any real faults, since safe mode was forced.

```
SAFEMODE> faults_print
0: NMI Exception, uhandle:00 bhandle:00
1: NMI Exception, uhandle:00 bhandle:00
2: NMI Exception, uhandle:00 bhandle:00
3: NMI Exception, uhandle:00 bhandle:00
4: NMI Exception, uhandle:00 bhandle:00
5: NMI Exception, uhandle:00 bhandle:00
6: NMI Exception, uhandle:00 bhandle:00
7: NMI Exception, uhandle:00 bhandle:00
Success
```

If you have a module that has entered safe mode unintentionally, please open an [ACKme Support ticket](#) and post the output of the [faults_print](#) command.

Return to Normal Operation

To reset all faults in preparation for returning the module to normal operation, use the [faults_reset](#) command.

```
SAFEMODE> faults_reset
Success
SAFEMODE> reboot
Rebooting
WiConnect-2.1.0.9, Built:2015-01-25 15:58:14 for AMW004.3, Board:AMW004-E03.3
[Ready]
>
```

After resetting faults, the module must be either [factory reset](#) or [rebooted](#) before normal operation is returned. The prompt (if enabled) automatically returns to >.

Change Log

Modified	Changes	WiConnect Version Required
2015-02-03	Created	1.0+

TCP Client and UDP Client Auto-Connect

Description

This example shows how to use the TCP and UDP auto connection feature. With this feature the module will automatically connect to a remote server. The following features are demonstrated:

- TCP Client Auto-Connect - the module automatically connects to a remote TCP server
- UDP Client Auto-Connect - the module automatically connects to a remote UDP server
- Stream Mode - data is automatically streamed on the module UART. No need for host MCU interaction.

Setup

First let's configure some basic module settings needed for the rest of the examples. Issue the following command to connect the module to a local network:

```
nup -s
```

Once this command completes, the module should be connected to the local network.

By default the module is in **Command Mode**. Issue the following command to put the module in **Stream Mode**.

```
set bus.mode stream
```

Next configure the module to automatically connect to the network. Then save and reboot:

```
set wlan.auto_join.enabled 1
save
reboot
```

After the module reboots, it's in **Stream Mode**. To issue more commands to the module, a break-in sequence is needed. The default break-in sequence is '\$\$'. Issue '\$\$\$' to break out of stream mode and into Command Mode:

```
$$
Command Mode Start
>
```

Note: if an invalid break-in sequence is issued, wait 3 seconds before sending the break-in sequence again.

TCP Auto Connect

Now that we've issued the break sequence and can issue more commands, let's configure the module to automatically connect to a remote TCP server. Issue the following commands:

```
set tcp.client.remote_host test.ack.me
set tcp.client.remote_port 50007
set tcp.client.auto_start 1
save
```

```
reboot
```

That's it. The module has been configured to automatically connect to the remote TCP server: test.ack.me:50007. When the module reboots, it will connect to the network then connect to the TCP server. Once connected, anything that's typed into terminal will be echoed.

Notes:

- test.ack.me:50007 is an echo server. Anything that's sent to it, it sends back. This is why anything typed into the terminal is echoed back. The data flow is: UART RX → TCP Server TX → TCP Server RX → UART TX.
- For TCP Auto Connect to work, the network must be configured to auto start. This is done with [wlan.auto_join.enabled](#)

UDP Auto Connect

The UDP Auto Connect feature is very similar to the TCP Auto Connect. To configure the module for UDP Auto Connect, first issue the break-in sequence as described above. Once the module is able to receive more commands. Issue the following:

```
set tcp.client.auto_start 0
```

This will disable the TCP Auto connect feature from the previous example. Note that in [Stream Mode](#), only one stream may be opened at a time. Next issue the following commands to enable UDP Auto Connect:

```
set udp.client.remote_host test.ack.me
set udp.client.remote_port 50007
set udp.client.auto_start 1
save
reboot
```

That's it. The module has been configured to automatically connect to the remote UDP server: test.ack.me:50007. When the module reboots, it will connect to the network then connect to the UDP server. Once connected, anything that's typed into terminal will be echoed.

Change Log

Modified	Changes	WiConnect Version Required
2015-02-03	Created	2.1+

goHACK.me

The [goHACK.me](#) API provided by WiConnect enables monitoring and control of your ACKme device. You can set up a goHACK.me account, then using [goHACK.me commands](#) and [goHACK.me variables](#), you can activate your device and synchronize data, control and messages with the goHACK.me cloud service.



NOTE! WiConnect version 1.2 or higher is required to use goHACK.me. To update WiConnect, use the [OTA update](#) command.

Getting Started with goHACK.me

Eager to get started? No problem! Let's jump right in and get your new device connected (we'll get to the details of how goHACK.me works later).

The goHACK.me promise: Connect your device ready for control from the internet in just a few easy steps ...

Step	Description
» 1 Signup	The first step to getting started with goHACK.me is to signup for an account. Go to the goHACK.me site and click the SIGNUP button to create your account.
» 2 Connect	Plug your Mackerel (or other ACKme device) into your computer via USB and open a WiConnect terminal (see Getting Started), then enter the <code>network_up</code> command at the prompt as shown below. The device scans for Wi-Fi networks in range, and offers a selection of available networks. Pick your network and enter the network password. <pre>> network_up -s</pre>
» 3 Setup	When a device activates with goHACK.me, it tells goHACK.me what it can do (or is capable of doing) by sending a capabilities file. To get you started, ACKme provides an example capabilities file that is perfectly suited to your device. And even better, we also provide an example automated configuration script to set up your device to go SOLO . No external host micro needed! To download and setup your device with example default capabilities, enter the <code>ghm_capabilities</code> command at the prompt as shown below. <pre>> ghm_capabilities download -s</pre>

» 4 Activation	<p>Activate the device with goHACK.me using the ghm_activate command. This will associate the device with your account.</p> <p>A valid goHACK.me username and password must be provided.</p> <pre>ghm_activate youremail@address.com yourpassword</pre>
-----------------------	---

Congratulations! Your device is now connected, setup and ready for use with your shiny new goHACK.me account.

To see your device on goHACK.me, open a web browser and login to [goHACK.me](#). After login, your device appears in the goHACK.me device view. Assuming your device has one or more LED lights on it (and all of our devices do) go ahead and press a button and within a few seconds, the corresponding LED on your device will magically turn on!

Using WiConnect with goHACK.me

The following sections provide a brief overview of how to use WiConnect with goHACK.me. We suggest you review the [goHACK.me Getting Started Guide](#) to familiarize yourself with goHACK.me before continuing.

Slave vs. Solo Mode

WiConnect can be used in one of two modes with goHACK.me. Read on and decide which mode works best for your connected application.

- **Slave mode.** In slave mode, WiConnect is a slave to a host microcontroller (or computer). The MCU communicates with goHACK.me using commands built into WiConnect. For example, the host may sync streams and controls with goHACK.me by using the [ghm_sync](#) command.
- **Solo mode.** In solo mode, WiConnect operates standalone (no host MCU needed) and automatically syncs with the goHACK.me cloud. Solo mode is enabled by setting the WiConnect [ghm.solo.enabled](#) variable, and GPIOs used with solo mode are configured with [ghm.solo gpio](#). The solo mode sync period i.e. how often stream samples are sent to goHACK.me, is configured using [ghm.solo.sync_period](#).



Signup and Device Activation

Create a goHACK.me user account and activate your device with goHACK.me as described in [Getting Started](#) above.

If you already have a goHACK.me account, use the `ghm_activate` command to activate additional devices as shown in the following example.

Note: The `ghm_activate` command attempts to read the `device capabilities` file configured with the WiConnect `ghm_capabilities` command. To check whether your device is already activated, read the `ghm.status` variable.

```
> ghm_activate your-email@address.com ****
Request POST /api/activate/login
Connecting (HTTPS): api.gohack.me:443
Starting TLS
HTTP response: 200
Success
```

Device Capabilities

When your device is activated using the `ghm_activate` command, the device tells goHACK.me what it is capable of doing. Device capabilities (stored as a file in JSON format), describe device functionality, preferences, messages and information about connected I/O including the number of I/Os connected, a name for each I/O, a type or range for each I/O, plus more ... In WiConnect & goHACK.me documentation, capabilities file is referred to simply as the "caps" file.

A caps file has the following entries.

Entry	Description
model	The model number of the device
title	The device title or name
description	A short description of the device
streams	An array of (read-only) sampled streams sent to goHACK.me. Examples include the open/close state of a door, the temperature of your wine cellar, etc.
controls	An array of (read/write) control parameters. Control parameters may be changed on goHACK.me and are read by the device. Since controls are used to control physical "things" connected to your device, your porch light for example, controls are also writeable by the device. This enables you to bypass goHACK.me to control your porch light directly from the device if required.
messages	An array of message boxes that enable messages to be sent to, and received from, your device

The goHACK.me `ghm_activate` command reads the caps file (configured using the `ghm_capabilities` command) from the file system.

Streams

goHACK.me devices send streams of data samples to the goHACK.me server. Each stream carries samples of the state or value of an I/O connected to the device. Streams enable you to monitor and measure your "things". Some example streams include the **temperature of your wine cellar**, the **open/close state of the back door**, or the **amount of energy the pool pump is using**.

The caps file contains an array of streams to monitor "things" connected to your device. Each stream in the caps file contains the

following entries.

Entry	Mandatory	Description
slug	x	The name of the stream (space or underscore characters NOT allowed)
title	x	A verbose title for the stream (must be set to "" if not used)
symbol	x	Symbol for the units. Symbols may be a text string e.g. kw or an HTML code , ° for example
unit	x	Units for the stream, e.g. degrees C (must be set to "" if not used)
type		Valid types are: boolean, integer, float. If type is not specified, float is assumed.

Controls

Control parameters allow you to send commands to your device to in turn control “things” connected to it. Some examples include **turn on the porch light**, **open the garage door** or **set the heat to 75 degrees**.

The caps file contains an array of controls to control “things” connected to your device. Each control in the caps file contains the following entries.

Entry	Mandatory	Description
slug	x	The name of the control (space or underscore characters NOT allowed)
title	x	A verbose title for the control (must be set to "" if not used)
symbol		Symbol for the units. Symbols may be a text string e.g. kw or an HTML code , ° for example.
unit		Units for the control, e.g. degrees C,
type	x	The type of control. Available options are: boolean, integer, float, select
options		If type is select, a list of options that may be selected
value	x	The value to assign to the control
value_min		Minimum value. Not valid for controls of type = select. value_min must be less than the value of value_max
value_max		Maximum value. Not valid for controls of type = select. value_max must be greater than the value of value_min

Messages

goHACK.me provides the capability to send messages to your devices and to receive messages from your devices. Messages are currently limited to plain text, however goHACK.me will support a range of message types including audio clips and images in the future.

To enable messaging to and from your device, the caps file must contain a messages section. The messages section is not configurable, and must appear in the caps file exactly as shown in the following example capabilities object. Message box titles may be changed to suit your requirements.

Example Caps File

The following text provides an example caps file for the **Mackerel board v3**, together with a description of how to write the caps file to the WiConnect file system, and set the caps file for use with goHACK.me.

```
{  
    "model": "AMW004-E03.3",  
    "title": "Mackerel",  
    "streams": [  
        {  
            "slug": "button1",  
            "title": "Button 1",  
            "unit": "On/Off",  
            "symbol": "",  
            "type": "boolean"  
        },  
        {  
            "slug": "button2",  
            "title": "Button 2",  
            "unit": "On/Off",  
            "symbol": "",  
            "type": "boolean"  
        },  
        {  
            "slug": "thermistor",  
            "title": "Thermistor",  
            "unit": "Levels",  
            "symbol": "",  
            "type": "integer"  
        }  
    "controls": [  
        {  
            "slug": "led1",  
            "title": "LED 1",  
            "value": 0,  
            "type": "boolean"  
        },  
        {  
            "slug": "led2-duty",  
            "title": "LED 2 PWM Duty Cycle",  
            "value": 20,  
            "value_min": 0,  
            "value_max": 100,  
            "type": "integer"  
        },  
        {  
            "slug": "led2-freq",  
            "title": "LED 2 PWM Frequency",  
        }  
    ]}
```

```
        "value": 1000,
        "type": "integer",
        "value_min": 1,
        "value_max": 1200000
    },
],
"messages": [
{
    "slug": "from-device",
    "title": "From Device"
},
{
    "slug": "to-device",
    "title": "To Device"
}
]
```

To write the caps file (shown above) to the WiConnect file system and configure the caps file for use with goHACKme, use the WiConnect [file_create](#) commands as follows.

```
> file_create ghm_capabilities.json 1445
... paste caps text here ...
> ghm_capabilities set ghm_capabilities.json
```

NOTE: Some terminal applications may incorrectly translate EOL characters, if you're having trouble with the above example, try the following caps file with all non-essential whitespace removed. The length of the caps file must be changed to match the reduced number of characters in the file.

If you are having trouble cutting and pasting the above capabilities object with whitespace included, try using the following version which has all whitespace removed. For reference, a useful Windows editor for removing non-essential whitespace is [Notepad++](#).

```
{ "model": "AMW004-E03.3", "title": "Mackerel", "streams": [ { "slug": "button1", "title": "Button 1", "unit": "On/Off", "symbol": "", "type": "boolean" }, { "slug": "button2", "title": "Button 2", "unit": "On/Off", "symbol": "", "type": "boolean" }, { "slug": "thermistor", "title": "Thermistor", "unit": "Levels", "symbol": "", "type": "integer" } ], "controls": [ { "slug": "led1", "title": "LED 1", "value": 0, "type": "boolean" }, { "slug": "led2-duty", "title": "LED 2 PWM Duty Cycle", "value": 20, "value_min": 0, "value_max": 100, "type": "integer" }, { "slug": "led2-freq", "title": "LED 2 PWM Frequency", "value": 1000, "type": "integer", "value_min": 1, "value_max": 1200000 } ], "messages": [ { "slug": "from-device", "title": "From Device" }, { "slug": "to-device", "title": "To Device" } ] }
```

And here's the corresponding [file_create](#) and [ghm.capabilities](#) set commands (with a reduced character count to match the lack of whitespace) for completeness ...

```
> file_create ghm_capabilities.json 805
> set ghm.capabilities ghm_capabilities.json
```

The goHACK.me API

A common set of WiConnect goHACK.me API commands and variables are used for device activation/deactivation and basic setup, whether the device is used in slave mode or solo mode. These commands/variables are summarized in the following list.

- Commands
 - `ghm_activate` – Activate additional devices with goHACK.me
 - `ghm_capabilities` – Setup device capabilities
 - `ghm_deactivate` – Deactivate a device
- Variables
 - `ghm.cache_size` – Set the local stream cache size
 - `ghm.capabilities` – Read the caps filename
 - `ghm.status` – Read the device activation status

Slave Mode

In slave mode, the host writes streams and controls using `ghm_write`, and reads controls (from a local cache after a sync) using `ghm_read`. To manually force stream samples to be pushed to goHACK.me, and the latest control values to be pulled from goHACK.me, the host uses the `ghm_sync` command. Messages may be sent or received using `ghm_message` and read or written using the `stream_read`, `stream_write` and `stream_close` commands.

When stream samples are written, a local cache is provided to minimize the overhead and additional power drain of connecting to the network to send individual samples. The number of samples cached before automatically pushing to goHACK.me is configurable with the `ghm.cache_size` variable. To immediately push all locally cached samples to goHACK.me, use the `ghm.sync` command with the `push` option. A summary of commands used with slave mode is provided in the following list.

- `ghm_message` – Send & receive messages
- `ghm_read` – Read controls
- `ghm_sync` – Synchronize streams and controls with goHACK.me
- `ghm_write` – Write streams and controls

Solo Mode

In solo mode, streams are automatically synchronized with goHACK.me. Solo mode is enabled by setting the WiConnect `ghm.solo.enabled` variable, and GPIOs used with solo mode are configured with `ghm.solo gpio`. The synchronization period is configured with `ghm.solo.sync_period`. After synchronization, any changes in control parameters are printed to the WiConnect `bus.log_bus` if `ghm.solo.echo` is enabled.

- `ghm.solo.echo` – Echo changes to controls to the `bus.log_bus`
- `ghm.solo.enabled` – Enable/disable solo mode
- `ghm.solo gpio` – Configure GPIOs connected to streams/controls
- `ghm.solo.sync_period` – Configure the period to sync with goHACK.me

In solo mode, WiConnect operates standalone (no host MCU needed) and automatically syncs with the goHACK.me cloud. The solo mode sync period i.e. how often stream samples are sent to goHACK.me, is configured using `ghm.solo.sync_period`.

goHACK.me Basic Account

A basic goHACK.me account is provided entirely free for use by customers of ACKme Networks. The basic account is provided with some limitations as outlined in the following list. If you need an unrestricted commercial cloud service, please contact ACKme at <http://ack.me/contact>

- Devices are limited to a maximum of 4 samples and 4 controls
- The minimum period for synchronisation with goHACK.me is 4 seconds
- Each account is limited to a maximum of 4 devices
- Accounts are limited to a maximum of 4 triggers

Troubleshooting Guide

If you clicked through to here, then we expect you are having some trouble with WiConnect. Please accept our apologies; perhaps our documentation is not clear enough or you have found an operational corner case.

The notes in the following sections should get you up and running again quickly. If you are still having trouble after reading through these notes, please log a [support ticket](#).

Basics

- Check the power LED is on
- When you type into the terminal application (see [Getting Started](#)), the UART LED(s) on the evaluation board should blink.

My module fails to associate to the network

Please check each of the following points carefully to help resolve this issue.

- Double-triple check the spelling of the network name assigned to `wlan.ssid`. It is VERY easy to incorrectly enter the name of a network, and especially so if the network name is long and unwieldy.
- Check the network password or security key assigned to `wlan.passkey`. If your network uses WEP (and we sincerely hope it does not since WEP is **NOT** secure), follow the setup procedure for WEP keys documented in the [wlan variable documentation](#).
- Check the value of `wlan.security` is correctly set. For most networks, this variable should be set to Auto. If you plan to use WEP, `wlan.security` must be set to WEP.

My module has missing commands or variables. What do I do?

If you enter a valid command or variable and see an Unknown command or Unknown variable or option error, your module may be in [Safe Mode](#). Alternatively an [OTA upgrade](#) may have been interrupted before completion.

Recovery from Safe Mode

In safe mode, `system.safemode` is set to 1, and there is a reduced set of commands and variables (see [help all](#)).

To recover from Safe Mode, follow the instructions in [Upgrade & Recover, Safe Mode](#). The essential recovery commands are summarised below.

WiConnect Commands	Description
<code>faults_reset</code>	<- Reset faults counter
<code>reboot</code>	<- Reboot the module

Once the module has successfully returned to normal operation, it is advisable to ensure you have the latest WiConnect firmware

version. Upgrading to the latest firmware provides your module with the latest fixes to problems, one of which may have caused the module to enter safemode. To upgrade, simply enter the `ota` command.

Recovery from Interrupted OTA Upgrade

If an OTA upgrade is interrupted, you can resume it and complete the upgrade. You may have to set the `wlan.ssid` and `wlan.passkey` variables to rejoin your WLAN. Then run the `ota` command. When the OTA upgrade completes, your module returns to normal operation.

WiConnect commands

This page provides a list of WiConnect commands with a full description of how to use each command.

Nav Tips for Humans

WiConnect command mode is friendly for humans too. Use the following tips and you'll be an expert WiConnect terminal jockey in no time flat!

Tip	Description
Tab complete	Type part of a command or variable name then hit the tab key. The command completes or partially completes, just like a terminal in the linux or DOS world.
History	Up to 5 commands of history are available. Press the up-arrow key to access recently entered commands.
Edit	The left-arrow , home , end and backspace keys may be used to edit or navigate commands already present on the command line.

Shortcuts

A shortcut is assigned to each command. Common sense dictates the rules for shortcuts. Commands that are already short do not need a shortcut other than their name, shortcuts for longer commands sometimes take the first three letters of the command name, and other times take the first letter of the name plus several other letters contained in the name. Shortcuts are no longer than six characters and are easy to guess, the ultimate reference however is command help and the documentation on this page.

Documentation Format

Many of the WiConnect responses shown in the examples on this page were captured with `system.print_level = all`, and `system.cmd.header_enabled = true`. These settings are provided to make it easy for a host microcontroller to parse responses by examining `response.headers`. Information about machine parsing is available in [Serial Interface](#), [MCU Config](#).

Documentation for each command is provided in the format shown below.

command

Abbreviation

cmd

Description

A description of how to use the command with, together with notes about available options and arguments.

Syntax

Formal command syntax with a listing of all available options and arguments.

Example

One or more examples demonstrating how the command is used.

Alphabetical List of Commands

- **A**
 - adc_take_sample
- **D**
 - dac_set_level
- **E**
 - exit
- **F**
 - factory_reset
 - faults_print
 - faults_reset
 - file_create
 - file_delete
 - file_open
 - force_safemode
 - format_flash
- **G**
 - get
 - ghm_activate
 - ghm_capabilities
 - ghm_deactivate
 - ghm_message
 - ghm_read
 - ghm_signup
 - ghm_sync
 - ghm_write
 - gpio_dir
 - gpio_get
 - gpio_set
 - gpios_dir
 - gpios_get
 - gpios_set
- **H**
 - help
 - http_add_header
 - http_download
 - http_get
 - http_head
 - http_post
 - http_read_status
 - http_upload
- **L**
 - ls
 - load
- **M**
 - mdns_discover
- **N**

- [network_down](#)
- [network_lookup](#)
- [network_up](#)
- [network_verify](#)
- **O**
 - [ota](#)
- **P**
 - [ping](#)
 - [pwm_update](#)
- **R**
 - [read](#)
 - [reboot](#)
- **S**
 - [save](#)
 - [set](#)
 - [setup](#)
 - [sleep](#)
 - [smtp_send](#)
 - [stream_close](#)
 - [stream_list](#)
 - [stream_poll](#)
 - [stream_read](#)
 - [stream_write](#)
- **T**
 - [tcp_client](#)
 - [tcp_server](#)
 - [tls_client](#)
 - [tls_server](#)
- **U**
 - [udp_client](#)
 - [udp_server](#)
- **V**
 - [version](#)
- **W**
 - [wlan_get_rssi](#)
 - [wlan_scan](#)
 - [wps](#)
 - [write](#)

Description of Commands

adc_take_sample

Abbreviation

adc

Description

Take an analog sample of a pin using an onboard Analog-Digital Converter.

To set the ADC reference voltage, see [system.adc.vref](#).

Syntax

```
> adc_take_sample <gpio#> [<gain>] [-v]/[<LUT Filename.csv>]
```

- <gain> - optional, defaults to 1 if not specified. The valid values for the adc gain are 1, 2, 4.
- <-v> - optional, return ADC value as voltage. Must NOT be specified with the LUT parameter
- <LUT Filename.csv> - optional, filename of Lookup Table (LUT) to use for converting ADC values. See [Peripherals, ADC Lookup Tables](#) for details.

Example

```
> adc 1 1
R000007
0x805
> adc 20 thermistor.csv
R000009
80.0223
```

D

dac_set_level

Abbreviation

dac

Description

Set the output level of a DAC. The DAC output voltage level is set to $3.3V \times <\text{level}>/4096$.

Supported hardware: GPIO 0 on AMW004 (Wallaby).

Syntax

```
> dac_set_level <level>
```

Example

```
> dac 0 2048
Success
```

E

exit

Abbreviation

exit

Description

Exits **command mode** immediately and returns to stream mode. Only valid when **stream mode** is operational.

Syntax

```
> exit
```

Example

```
> exit
Command Mode Stop
[Ready]
```

F

factory_reset

Abbreviation

fac

Description

Reset the application configuration to factory default values. The WLAN MAC address wlan.mac must be provided as an argument to avoid accidental factory reset. Note! The application may also be factory reset by asserting GPIO0 for 10 seconds through a module reset as described in [Upgrade and Recovery, Factory Reset](#).

Syntax

```
> factory_reset <MAC address>
```

Example

```
> get wl m  
R000019  
4C:55:CC:10:03:44  
> fac 4C:55:CC:10:03:44  
Success
```

faults_print**Abbreviation**

faup

Description

List system faults. After 8 faults, the system enters Safe Mode. See [Upgrade and Recovery, Safe Mode](#).

Syntax

```
> faults_print
```

Example

```
> faup  
Success
```

faults_reset**Abbreviation**

faur

Description

Reset system fault counter. See [Upgrade and Recovery, Safe Mode](#).

Syntax

```
> faults_reset
```

Example

```
> faur
Success
```

file_create

Abbreviation

fcr

Description

Create a new file <size> bytes in length. After the command is issued, the ensuing <size> number of bytes is written to the file system.

Files can be written in chunks smaller than the specified length, using the -o option.

Additional options include a version, [file type](#) and [CRC](#), or [checksum](#).

See [Writing Files to the File System](#).

Syntax

```
> file_create [-[e][u]] [-o] <filename> <size> [<version> [<type> [<crc>]]]
```

Option	Description
-e	Optional flag, must be first argument if used. Specifies that the file is essential and must not be deleted during an OTA. Specified together with -u flag (i.e. -eu for both. Flags can occur in any order). Note: Files are deleted during OTA only if space is required. OTA may fail if files marked essential leave insufficient space.
-u	Optional flag. Set HTTP Server unprotect flag for file. Specified together with -e flag (i.e. -eu for both. Flags can occur in any order). See HTTP Server Security and Authorization .

-o	Optional. Leave file open for writing. If the -o option is not specified, then all the file data must come immediately after the command. If the -o option is specified, the file can be written in chunks. A stream handle is returned after the command. The write command is used to write data to the file. The stream closes automatically after the specified <size> bytes of data is written. The file is not considered 'valid' until it automatically closes. If the module is rebooted or the close command is issued before all data is written, the file data is lost. It is not possible to restart writing the file after the stream is manually closed.
filename	Name of file to be written.
size	Length of file in bytes. After the command is issued, write the file content to the serial interface.
version	Specify version number. Default 1.0.0
type	See File System, File Types
crc	optional, CRC16-CCITT checksum of the file data, in ASCII hex format. See File System, Checksum

Example

In the example below, the file content , Hello World! is 12 bytes in length and is entered or sent by the user.

```
> fcr myfile.txt 12
Hello World!
File created
```

Example

Create file for writing in chunks

```
> file_create -o my_file.txt 100
3
> write 3 50
... <50 bytes of data> ...
> write 3 50
... <50 bytes of data> ...
File created
```

file_delete

Abbreviation

fde

Description

Delete a file from the file system.

Syntax

```
> file_delete <filename>
```

Example

```
> fde myfile.txt  
L000014  
File deleted  
R000009  
Success
```

file_open**Abbreviation****fop****Description**

Open a file and return a file stream handle if successful. Once open, the file contents may be read with the [stream_read](#) command.

Syntax

```
> file_open <filename>
```

Example

```
> fop myfile.txt  
R000003  
0
```

format_flash**Abbreviation****format****Description**

Format (i.e. erase all contents) of bulk or extended flash chip. If the extended flash is specified, ALL files BUT the wifi_fw.bin file are erased.

For details on enabling and using bulk flash, see [File System, Internal, Extended and Bulk Flash](#). See also the [system.bflash.cs_gpio](#) variable.

Syntax

```
> format_flash <bulk/extended> <module MAC address>
```

Example

```
> format bulk 4C:55:CC:10:10:98
Formatting flash...
```

force_safemode**Abbreviation****force_safemode****Description**

Force the module into safe mode. This command is used by a host to test recovery procedures if the module enters safe mode.
See [Application Notes, Recovery from Safe Mode](#) for further information.

This is a hidden command that does not appear in the list of commands returned by [help commands](#).

Syntax

```
> force_safemode <module MAC address>
```

Example

```
> force_safemode 4C:55:CC:10:10:98
Forcing safemode...
Rebooting
[Disassociated]
WiConnect-2.1.0.9, Built:2015-01-25 15:58:14 for AMW004.3, Board:AMW004-E03.3

*** Max faults exceeded. Entering Safe Mode.
SAFEMODE>
```

G**get****Abbreviation****get**

Description

Get the value of a WiConnect variable or group of variables.

Syntax

```
> get <[variable name] / [variable group]> [options]
```

Example 1

```
> get time rtc utc
R000029
2014-03-23T23:54:33.021464Z
```

Example 2

```
> get bus
R000166
bus.data_bus: uart0
bus.log_bus: uart0
bus.mode: command
bus.stream.cmd_gpio: 0
bus.stream.cmd_seq: $$$
bus.stream.flush_count: 1500
bus.stream.flush_time: 20
```

ghm_activate

Abbreviation

gac

Description

Activate the device with [goHACK.me](#). A valid goHACK.me username and password must be provided. If a capabilities filename is provided, the `ghm.capabilities` variable is updated with the specified filename. See [goHACK.me Device Capabilities](#).

Need a [goHACK.me](#) account? Signup at [goHack.me](#) and then use `ghm_activate` to activate your device. See [goHACK.me, Basic Account](#) for details of restrictions.

Note: If `ghm.solo.enabled` is set, the device reboots and automatically connects to goHACK.me after activation completes successfully.

Syntax

```
> ghm_activate <username> <password> [capabilities filename]
```

Example

```
> get ghm.capabilities
ghm_capabilities.json
> ghm_activate your-email@address.com ***** mycaps.json
Request POST /api/activate/login
Connecting (https): gohack.me:443
Starting TLS
HTTP response: 200
Success
> get ghm.capabilities
mycaps.json
```

ghm_capabilities**Abbreviation****gca****Description**

The `ghm_capabilities` command enables users to download and manipulate goHACK.me device capabilities files. Options include the ability to list example capabilities files on the goHACK.me server, download a specific capabilities file and setup the device to use it, or independently configure which capabilities file to use when the device activates with goHACK.me.

Note! After device activation, the capabilities file is read only. To modify or delete the file, first deactivate the device using `ghm_deactivate`.

Syntax**Listing Capabilities**

```
> ghm_capabilities <list>
```

where:

- `<list>` returns a list of example capabilities files for use with the device

Downloading Capabilities

```
> ghm_capabilities download [<filename / URL> [-s] [-d] [<version>]]
```

where:

- `<filename>` - The name of the capabilities file to download. If no filename is specified, download the default caps file, typically `ghm_capabilities.json`, and set it for use with goHACK.me

Note:

- Example caps files downloaded from ACKme overwrite existing files with the same name.
- Do not add a JSON extension to the filename
- <URL> - URL specifying the network location of the file. Local caps files with an identical name to a downloaded caps file will not be overwritten. In this case, the command fails. Delete the version of the file manually first.
- [-s] - Download a setup script that works with the capabilities, then run the script to setup the device to run 'solo' in a standalone mode (external host not needed!).
- [-d] - Download the file only, do not set the file for use.
- [version] - Provide a version number for the file.

All parameters are optional. If no parameters are provided, the default capabilities file for the device is downloaded and configured for use with goHACK.me.

Note:

- Device activation is NOT required to use this command
- The capabilities filename on the server is used as the filename on the local file system. If the file already exists on the filesystem, the command fails.

Setup & Querying Capabilities

```
> ghm_capabilitiess <get/set>
```

where:

- <get> returns the filename of the capabilities used with goHACK.me. The ghm.capabilities variable may also be used to return the capabilities filename.
- <set> sets the filename of the capabilities to be used with goHACK.me

List Example

```
> ghm_capabilitiess list
Request GET /1.2/ghmcaps/amw004_e03.4/capabilities_list?do=export_raw
Connecting (http): wiconnect.ack.me:80
HTTP response: 200
=====
Capabilities Filename Description
=====
ghm_capabilities      Default capabilities
-----
          NAME      TITLE
Streams   --> button1    Button 1
          --> button2    Button 2
          --> thermistor Thermistor
Controls  <-> led1       LED 1
          <-> led2       LED 2
Messages   --> from-device
          <-> to-device
```

Download Example

Download the default capabilities file and run the accompanying setup script for **solo** mode operation:

```
> ghm_capabilities download -s
Connecting (http): wiconnect.ack.me:80
HTTP response: 200
Setup file: ghm_capabilities_setup.csv created
Running setup script
> Setup GPIO for use with goHACK.me
set      ghm.solo.GPIO      0      button1
Set OK
> Setup GPIO for use with goHACK.me
set      ghm.solo.GPIO      10     led1
Set OK
> Set goHACK.me auto sync period for solo mode
set ghm.solo.sync_period          4
Set OK
> Enable solo mode for standalone operation
set      ghm.solo.enabled        1
Set OK
> save
Saved
Success
> get      ghm.solo.GPIO      all
! # Description
# 0 - stream.button1 - in
# 10 - control.led1 - out
```

Download user-generated capabilities file, set the version to 2.3.0:

```
> ghm_capabilities download http://myserver.com/my_caps.json 2.3.0
Request GET /1.2/ghmcaps/amw004_e03.4/ghm_capabilities?do=export_raw
Connecting (http): wiconnect.ack.me:80
HTTP response: 200
Capabilities file: ghm_capabilities.json created
Success
```

ghm_deactivate

Abbreviation

gde

Description

Deactivate the device from goHACK.me. A valid goHACK.me username and password must be provided for successful deactivation.

Syntax

```
> ghm_deactivate <username> <password>
```

Example

```
> ghm_deactivate your-email@address.com ****
Request POST /api/token/deactivate
Connecting (https): gohack.me:443
Starting TLS
HTTP response: 200
Success
```

ghm_message**Abbreviation****gme****Description**

goHACK.me provides the ability for devices to get messages from a message box called **to-device** and post messages to a message box called **from-device**. The **ghm_message** API is used to manipulate messages. The primary message options are **list**, **get**, **post** and **delete**.

Syntax**Listing Messages**

```
> ghm_message <list> [<-t time> / <-c count>]
```

where:

- < -t time> - gets all messages since the specified time
- < -c count> - gets 'count' messages, latest first

Getting Messages

```
> ghm_message <get> [<list_index / msg_id>] [<all> / <body>]
```

where:

- <list_index / msg_id> is optional, the latest message is returned if omitted
- <list_index> is the message index returned by **ghm_message <list>**
- <msg_id> is the ID of a specific message
- <all / body> is optional, only the message text is returned if omitted

- <all> returns the entire message including the message header
- <body> returns the JSON formatted message body

If successful, ghm_message <get> returns a stream handle, the message may then be read using the stream_read command.

Posting Messages

```
> ghm_message <post>
```

If successful, the ghm_message <post> returns a stream handle. The message may then be written using the [stream_write](#) command. To complete and send the message, use the [stream_close](#) command.

Deleting Messages

```
> ghm_message <delete> <index / msg ID>
```

where:

- <index / msg ID> is either the message index returned by ghm_message <list> or the ID of a message

List Example

```
> ghm_message list
Request GET /api/token/messages/to-device/index
Connecting (https): api.gohack.me:443
Starting TLS
HTTP response: 200
! # Message ID           Timestamp      Length
# 0 09b6a12e-162f-435d-86cd-14463ff5e8a4 1404832747366 0x0010
# 1 6d47011e-ac4d-4501-80fe-4fe8921cabb0 1404832741419 0x000f
# 2 990df780-7d5d-4032-a8d5-f3810c89e459 1404832704239 0x0020
# 3 4b93708b-e387-4a74-8f1f-6dd30b9c84c8 1404830371994 0x0019
# 4 f3f3b532-ef6f-49be-ba41-8db035572beb 1404830267929 0x0017
# 5 fe1bfd74-f1b9-437e-ae8f-7edfd7f56492 1404830260260 0x0013
```

Get Example

```
> ghm_message get 2
Request GET /api/token/messages/to-device/990df780-7d5d-4032-a8d5-f3810c...
Connecting (https): api.gohack.me:443
Starting TLS
HTTP response: 200
[2014-07-08 | 15:21:16: Opened: 0]
0
> stream_read 0 100
Hello from goHACK.me!
[2014-07-08 | 15:21:18: Closed: 0]
```

Post Example

```
> ghm_message post
Request POST /api/token/messages/from-device
Connecting (https): api.gohack.me:443
Starting TLS
[2014-07-08 | 15:21:55: Opened: 0]
0
> stream_write 0 6
Hello!
Success
> stream_close 0
HTTP response: 200
[2014-07-08 | 15:22:25: Closed: 0]
Success
```

Delete Example

```
> ghm_message delete 990df780-7d5d-4032-a8d5-f3810c89e459
Request DELETE /api/token/messages/to-device/990df780-7d5d-4032-a8d5-f3810c89e459
Connecting (https): api.gohack.me:443
Starting TLS
HTTP response: 200
Success
```

ghm_read

Abbreviation

gre

Description

Read the value of control parameter(s) from a local cache after using **ghm_sync** to pull the latest values from goHACK.me. If a control parameter name is not provided, the value of all control parameters is returned. The timestamp is returned only if it is non-zero.

Syntax

```
> ghm_read [control name]
```

Example

```
> ghm_read
fanspeed,3|porchlight,on
> ghm_read fanspeed
3
> ghm_read porchlight
on
```

ghm_signup

Abbreviation

gsi

Description

Signup from devices is no longer supported. Sign up for a goHACK.me account at <http://goHACK.me>. Once you have an account set up, use [ghm_activate](#) to activate your device. See [Getting Started with goHACKme](#) for details.

ghm_sync

Abbreviation

gsy

Description

Immediately push locally cached stream samples and control parameters to goHACK.me, and pull control parameters from goHACK.me. If the push or pull option is provided, only a push or pull is performed. If no options are provided, both a push and pull is performed. Control parameters pulled from goHACK.me are stored in a local cache and may be read using [ghm_read](#).

Syntax

```
> ghm_sync [push/pull]
```

Example

```
> ghm_sync
Request GET /api/token
Connecting (https): gohack.me:443
Starting TLS
HTTP response: 200
Success
```

ghm_write

Abbreviation

gwr

Description

Write the value of one or more stream samples or control parameters. Samples are written to a local cache and `y pushed to goHACK.me when the cache size is exceeded. Control parameters are immediately pushed to goHACK.me.

Use the first example template if only one value is written, and the second example template if more than one value is written. If the optional timestamp is not provided, values are automatically timestamped by the server.

Syntax

```
> ghm_write <name> <value> [timestamp]
> ghm_write <name1>,<value1>[,timestamp1]<name2>,<value2>,...
```

where:

- * <name> is the name of the stream or control parameter
- * <value> is the value to be written
- [timestamp] is an optional timestamp

Control Example

```
> ghm_write_control led1 1
Request POST /api/token
Connecting (https): gohack.me:443
Starting TLS
HTTP response: 200
Success
```

Stream Example

```
> ghm_write dooropen 1
Success
> ghm_write temperature,75,1404833404122|temperature,79,1404833404302
Success
```

gpio_dir

Abbreviation

gdi

Description

Set the direction of a general purpose I/O pin. To deregister the GPIO, set the direction to either -1 or none.

Note : This is a run-time only setting: the value is not saved with the [save](#) command. To set the direction of a GPIO after a reboot, set the variable [gpio_init](#). See [Peripherals, Setting GPIO Function](#).

Syntax

```
> gpio_dir <GPIO#> <direction>
```

where <direction> may be one of the following values. For each type there are several equivalent alias values, listed with comma separation:

Direction Value	I/O Type Description
in, input, inhz, in_highz, input_hz	Input, high-impedance
ipd, inpd, input_pd, input_pulldown	Input, pull-down
ipu, inpu, input_pu, input_pullup	Input, pull-up
out, output, opp, out_pp, output_pp	Output, push-pull
ood, out_od, ouput_od	Output, open-drain no-pull
oodpu, out_odpu, output_odpu	Output, open-drain pull-up
-1, none	Deregister the GPIO

Example

```
> gdi 12 in_highz
R000008
Set OK
```

gpio_get

Abbreviation

gge

Description

Get the current value of a general purpose I/O pin.

Syntax

```
> gpio_get <GPIO#>
```

Example

```
> gge 12
R000003
0
```

gpio_set

Abbreviation

gse**Description**

Immediately set the value of a general purpose I/O pin. When setting a GPIO, the GPIO direction must be also correctly set using the command `gpio_dir` or the command will fail.

Syntax

```
> gpio_set <GPIO#> <value>
```

Example

```
> gse 12 1
L000032
GPIO not configured for output
R100016
Command failed
```

gpios_dir**Abbreviation****gdis****Description**

Set the direction of all general purpose I/O pins. A direction for all GPIOs must be provided with the direction of GPIO 0 in the first (left-most) position in the array.

Note : This is a run-time only setting: the direction values are not saved with the `save` command. To set the direction of a GPIO after a reboot, set the variable `gpio_init`. See [Peripherals, Setting GPIO Function](#).

Syntax

```
> gpios_dir <direction array>
```

where a `<direction array>` element may be one of the values in the following table.

Enumerator	I/O Type Description
0	Input, with pull-up
1	Input, pull-down
2	Input, high-impedance
3	Output, push-pull

4	Output, open-drain no-pull
5	Output, open-drain pull-up
6	Deregister the GPIO

Example

```
> gdis 10011011333530011000222226622
R000008
Set OK
```

gpios_get

Abbreviation**gges****Description**

Get the current value of all general purpose I/O pins. The value of standard GPIOs is returned as either 0 or 1. Other GPIO types not configured for GPIO access (such as UART pins or system indicators) are returned as X.

Syntax

```
> gpios_get
```

Example

```
> gpios_get
R000031
0XX00X0000000XX10XXX000000000
```

gpios_set

Abbreviation**gses****Description**

Immediately set the value of all general purpose I/O pins. When setting a GPIO, the GPIO direction must be also correctly set using [gpio_dir](#) or [gpios_dir](#) or the command will fail. The value for all GPIOs must be provided, with the value of GPIO 0 in the first (left-most) position in the array.

Syntax

```
> gpios_set <GPIO value array>
```

Example

The examples shown below are for the AMW004 Wallaby module which has 29 GPIOs, the first example command failed because the GPIO value array contained less than 29 values.

```
> gses 100110100101101001001011
```

```
L000022
```

```
Must supply 29 gpios
```

```
R700018
```

```
Invalid argument
```

```
> gses 10011011001011110100100101111
```

```
R000008
```

```
Set OK
```

H**help****Abbreviation**

help, ?

Description

Return a list of commands or variables; or return help for a specific command or variable.

Syntax

```
> help [all/commands/variables/<command>/<variable>]
```

Example

```
> help
```

```
Usage   : help [all/commands/variables/<command>/<variable>]
```

```
Shortcut: help
```

```
Brief   : Return a list of commands or variables; or return  
           help for a specific command, variable or group of  
           variables.
```

http_add_header

Abbreviation

had

Description

Add a custom HTTP header to the HTTP stream specified by <handle>.

Syntax

```
> http_add_header <handle> <key> <value>
```

where:

Option	Description
<handle>	the stream handle supplied in response to the http_get , http_head or http_post commands (opened with -o option)
<key>	header key
<value>	header value

Options must be provided in the order shown.

Example

```
> http_get -o ack.me
[2014-03-29 | 01:37:17: Connecting: http://ack.me]
Request GET /
Connecting (HTTP): ack.me:80
[2014-03-29 | 01:37:18: Connected: 0]
0
> had 0 Content-type application/text
Success
> hre 0
HTTP response: 200
200
```

http_download

Abbreviation

hdo

Description

Download one or more files from a remote http server and save to the extended or bulk flash.

Syntax

Download a Single File

```
> http_download [-i <wlan/softap>] [-m <json_size>] [-d] [-[e][u]] [-c <CRC>] <url>
  [<flash_file_name> [<version> [<type> [<cert_filename> ] ] ] ]
```

where parameters are as follows:

Option	Description
-i	Optional - Specify interface, MUST come first. If omitted uses network.default_interface .
-m <json_size>	Optional - Download multiple files with a json specification of size <json_size>. MUST come first if -i option not used. No other parameters are valid with -m option. See Downloading Multiple Files below for more information.
-d	Don't delete duplicate files. By default all duplicate file names are overwritten; with this option the command will fail on duplicates.
-e	Optional flag - Indicates file is essential . Specified together with -u flag (i.e. -eu for both). Flags can appear in any order).
-u	Optional flag - Set HTTP Server unprotected flag for file. Specified together with -e flag (i.e. -eu for both. Flags can appear in any order). See HTTP Server Security and Authorization .
-c <CRC>	Optional, hex string, specifies CRC for file, upon download, the file is read back and the calculated CRC is compared against this value. The file is not set 'valid' unless the CRCs match. See File System, File Checksum .
<url>	Full path to a file on a http server. See below for more info about URL.
<flash_file_name>	Optional - name to save file as. Use url filename if omitted
<version>	Optional - default 1.0.0 if omitted, set 0 to use default
<type>	Optional - the type of file, default to miscellaneous text file. Set 0 to use default
<cert_filename>	Optional - TLS certificate filename

Downloading Multiple Files

```
> http_download -m <json_size>
```

Immediately after issuing this command, send <json_size> bytes of a JSON formatted manifest file. An example of the manifest file is as follows:

```
{
  "path" : "https://myserver.com/path/to/my/files/",
  "cert" : "mycert.pem",
```

```
"files" : [
    {
        "remote" : "name_of_remote_file.html",
        "local" : "name_of_flash_file.html", // optional
        "version" : "1.0.0", // optional
        "type" : 150, // optional
        "flags" : "eu", // file is essential and unprotected
        "crc" : 23423 // optional
    },
    { ... }
]
}
```

String Replacement

WiConnect allows string replacement in the URL of the file for download, using a C-like or pythonesque syntax.

If the url contains %s, then the supplied filename is substituted for %s.

Example:

```
hdo 192.168.1.110:50007?id=%s&action=download my_file.txt
```

generates the URL:

```
http://192.168.1.110:50007?id=my_file.txt&action=download
```

and creates the file: my_file.txt in the module file system.

String replacement is also available in the manifest. For example:

```
{
    "path" : "http://myserver.com?file=%s&user=name&pass=1234",
    "cert" : "mycert.pem",
    "files" : [
        {
            "remote" : "my_file1.txt",
        },
        {
            "remote" : "my_file2.txt",
        },
    ]
}
```

generates the following URLs for download:

```
http://myserver.com?file=my_file1.txt&user=name&pass=1234
http://myserver.com?file=my_file2.txt&user=name&pass=1234
```

Example

Save goHACK.me home page as 'goHACKme.html' on extended or bulk flash:

```
> http_download http://gohack.me goHACKme.html
Downloading: goHACKme.html to flash file system
Request GET /
Connecting (http): gohack.me:80
HTTP response: 301
Redirected to https://gohack.me
Request GET /
Connecting (https): gohack.me:443
Starting TLS
HTTP response: 200
Success
> ls -v
! # Type Flags Hnd Size Version Filename
...
# 19 e-FE 0021 10 13936 1.0.0.0 goHACKme.html
...
```

http_get

Abbreviation**hge****Description**

Perform an HTTP(S) GET request and return a network stream handle if successful.

The protocol is assumed to be HTTP unless otherwise specified in the <url> parameter.

To allow custom headers, use the -o option.

Syntax

```
> http_get [-i <wlan/softap>] [-o] <url> [ca_cert_filename]
```

where:

Parameter	Description
-i	Network interface. Possible values are wlan or softap. Overrides default specified by network.default_interface

-o	Open the HTTP request but do not send. This enables custom header(s) to be added to the request using the <code>http_add_header</code> command.
<url>	URL to which request is sent. The URL prefix <code>http://</code> is optional for standard HTTP transactions, however <code>https://</code> must be added for secure HTTPS transactions.
[ca_cert_filename]	CA certificate file. Overrides default specified by <code>network.ca_cert</code>

Options must be provided in the order shown.

Example

```
> hge https://www.google.com.au
[2014-04-02 | 13:27:20: Connecting: https://www.google.com.au]
Request GET /
Connecting (HTTP): www.google.com.au:443
Starting TLS
[2014-04-02 | 13:27:21: Connected: 0]
HTTP response: 200
Status: 200
0
> hge -o example.com:80
[2014-04-11 | 10:48:25: Opening: example.com:80]
Request GET /
Connecting (HTTP): example.com:80
[2014-04-11 | 10:48:25: Opened: 1]
1
```

http_head

Abbreviation

hhe

Description

Perform an HTTP(S) HEAD request and return a network stream handle if successful.

The protocol is assumed to be HTTP unless otherwise specified in the <url> parameter.

To allow custom headers, use the -o option.

Syntax

```
> http_head [-i <wlan/softap>] -o <url> [ca_cert_filename]
```

where:

Parameter	Description
-i	Network interface. Possible values are wlan or softap. Overrides default specified by network.default_interface
-o	Open the HTTP request but do not send. This enables custom header(s) to be added to the request using the http_add_header command.
<url>	URL to which request is sent. The URL prefix http:// is optional for standard HTTP transactions, however https:// must be added for secure HTTPS transactions.
[ca_cert_filename]	CA certificate file. Overrides default specified by network.ca_cert

Options must be provided in the order shown.

Example

```
> hhe ack.me
[2014-04-02 | 13:32:28: Connecting: ack.me]
Request HEAD /
Connecting (HTTP): ack.me:80
[2014-04-02 | 13:32:29: Connected: 0]
HTTP response: 200
Status: 200
0
```

http_post

Abbreviation

hpo

Description

Perform an HTTP(S) POST request and return a network stream handle if successful.

The protocol is assumed to be HTTP unless otherwise specified in the <url> parameter.

To allow custom headers, use the -o option.

The HTTP Content-type must be provided.

Syntax

```
> http_post [-i <wlan/softap>] -o <url> <content type> [ca_cert_filename]
```

where:

Parameter	Description
-i	Network interface. Possible values are wlan or softap. Overrides default specified by network.default_interface
-o	Open the HTTP request but do not send. This enables custom header(s) to be added to the request using the <code>http_add_header</code> command.
<url>	URL to which request is sent. The URL prefix http:// is optional for standard HTTP transactions, however https:// must be added for secure HTTPS transactions.
<content_type>	The Content-type, (Internet media type, MIME type), of the content to be posted (e.g. application/json).
[ca_cert_filename]	CA certificate file. Overrides default specified by <code>network.ca_cert</code>

Options must be provided in the order shown.

Example

```
> hpo http://example.com application/json
[2014-04-02 | 13:35:19: Connecting: http://example.com]
Request POST /
Connecting (HTTP): example.com:80
[2014-04-02 | 13:35:20: Connected: 0]
HTTP response: 411
Status: 411
0
> hpo -o example.com:80
[2014-04-11 | 10:45:50: Opening: example.com:80]
Request POST /
Connecting (HTTP): example.com:80
[2014-04-11 | 10:45:50: Opened: 1]
1
```

http_read_status

Abbreviation

hre

Description

Read the HTTP response code returned by the last HTTP transaction on the stream specified by <handle>. If the HTTP request is pending and the transaction is incomplete, `http_read_status` sends the request and forces completion.

Syntax

```
> http_read_status <handle>
```

Example

```
> hre 1  
411
```

http_upload

Abbreviation**hup****Description**

Upload file(s) from the device flash to a remote HTTP server using HTTP POST and file upload.

Syntax**Upload a Single File**

```
> http_upload [-i <softap/wlan>] <url> <local filename>  
[<remote filename> [<content type> [<cert filename>]]]
```

where:

- -i - optional, specify interface, value 'wlan' or 'softap' MUST come first. If omitted uses [network.default_interface](#).
- <url> - URL of remote HTTP server
- <local filename> - name of file on sflash to upload
- <remote filename> - optional, name of file on remote server, use local filename if omitted.
- <content type> - optional, HTTP header content-type, e.g. image/jpeg
- <cert filename> - name of certificate on local serial flash

Upload Multiple Files using JSON Manifest

```
> http_upload [-i <softap/wlan>] -m <json size>
```

where:

- -i - optional, specify interface, value 'wlan' or 'softap' MUST come first
- -m - use JSON manifest for multi-file upload. MUST come first if -i option not used

The manifest must be input directly after the command is issued. Manifest format:

```
{  
  "path" : "https://myserver.com/path/to/save/files",  
  "cert" : "mycert.pem", // existing TLS on local sflash  
  "files" : [ {
```

```
"local" : "my_file_to_load.txt",
"remote" : "file_name_on_server.txt",    // optional
"name"    : "file",                      // optional
"type"    : "application/octet-stream",  // optional
},
{ ... }
]
}
```

Defaults for optional properties are:

- “remote” - The name of the file on the local file system.
- “name” - Populated in the ‘name’ field of form-data, the default is “file”
- “type” - “application/octet-stream”

Example

```
> hup http://myserver.com/uploadpage ghm_capabilities.json testfile.json
Uploading: ghm_capabilities.json to server
Request POST /uploadpage
Connecting (http): myserver.com:80
HTTP response: 200
Success
```

The file is saved on the upload server at myserver.com:80 as testfile.json

L

ls

Abbreviation

ls

Description

Return a list of available files located in internal, extended and bulk flash. Use the -v (or equivalent -l) option for full details.

Syntax

```
> ls [-v | -l]
```

Example

```
> ls -l
```

```
R000717
! # Type Flags Hnd Size Version Filename
# 0 i-00 801B 0 212736 2.1.0.0 upgrade_app.exe
# 1 i-81 801B 52 214776 2.1.0.0 wiconnect.exe
# 2 e-FB 0001 75 36578 2.1.0.0 command_help.csv
# 3 e-FD 0001 53 135 2.1.0.0 default_setup.script
# 4 e-FE 0001 74 1897 2.1.0.0 favicon.ico.gz
# 5 e-03 0001 52 1236 2.1.0.0 geotrust_ca.pem
# 6 e-FE 0021 84 422 1.0.0.0 thermistor.csv
# 7 e-FE 0001 55 20250 2.1.0.0 wiconnect_webgui.css.gz
# 8 e-FE 0001 54 1741 2.1.0.0 wiconnect_webgui.html
# 9 e-FE 0001 60 56176 2.1.0.0 wiconnect_webgui.js.gz
# 10 e-01 0009 0 210412 5.26.230.12 wifi_fw.bin
```

load

Abbreviation

load

Description

Load a configuration from a file previously saved by the **save** command.

Syntax

```
> load <config_file>
```

Example

```
> load config1.cfg
> Configuration successfully loaded
```

M

mdns_discover

Abbreviation

mdns

Description

Discover mDNS services on the local network. See variables `mdns.enabled`, `mdns.interface`, `mdns.name`, `mdns.service`, `mdns.ttl`.

Syntax

```
> mdns_discover [-v] [<service type>]
```

where:

- `-v` - optional, verbosely print the results
- `<service type>` - optional, a specific service type to discover (e.g. `_http._tcp`). If omitted, all found services are returned.

Example

```
> set wlan.ssid ackme
Set OK
> set wlan.passkey secretkey
Set OK
> set mdns.enabled 1
Set OK
> mdns_discover
[Associating to ackme]
Security type from probe: WPA2-AES
Obtaining IPv4 address via DHCP
IPv4 address: 10.5.6.115
Starting mDNS
mDNS domain: wiconnect-102f1a.local
[Associated]
Services found: 4
Service: _cisco-api._tcp.local
Name: _jenkins._tcp.local

Service: _https._tcp.local
Name: _workstation._tcp.local

Service: _http._tcp.local
Name: _http._tcp.local

Service: _cisco-sb._tcp.local
Name: routera296b6._cisco-api._tcp.local
Server: routera296b6._cisco-api._tcp.local
Address: 10.5.6.1
```

N

network_down

Abbreviation**ndo*****Description***

Bring down a network interface. If provided, the network interface overrides the default specified by the variable `network.default_interface`. All open streams on the interface will be closed.

Syntax

```
> network_down [-i <wlan/softap>]
```

Example

```
> ndo
L000040
[2014-03-24 | 00:08:48: Disassociated]
R000009
Success
```

network_lookup***Abbreviation*****nlo*****Description***

Perform a DNS lookup for a domain using the wlan interface.

Syntax

```
> network_lookup
```

Example

```
> nlo google.com
R000016
216.58.220.142
```

network_up***Abbreviation***

nup

Description

Start the process to bring up a network interface. The `-s` option prompts the user to select from a list of available APs. The `-i` option overrides the default network interface specified by [network.default_interface](#).

NOTE! Every command that needs access to the network will automatically bring up the network interface. The [network_up](#) command is non-blocking (except when the `-s` option is used) and provided for convenience only.

Syntax

```
> network_up [<-s> / <-i [wlan]/[softap]>]
```

Example

```
> network_up -s
Scanning for networks...
! 3 networks found
! # Ch RSSI MAC (BSSID)      Network (SSID)
# 0  6  -27 84:1B:5E:29:9D:F7 Take the blue pill
# 1 11  -68 EC:1A:59:36:5B:6C button_xt
# 2 11  -70 2C:B0:5D:31:6F:6A button

Type the number # that matches your Network: 0
Type the password for your Network      : welcome-to-kansas
[Associating to Take the blue pill]
In progress
[Associated]
```

network_verify

Abbreviation

nve

Description

Verify the module can connect to a Wi-Fi network with specific network credentials. This command provides a fast way test the network credentials entered by a user are valid when web setup with softAP is operational, see [Configuration & Setup, Web Setup](#). At present, only the Wi-Fi credentials are verified. No attempt is made to verify whether the module can obtain an IP address using DHCP.

The network verify command can also be used to quickly test network credentials at any time, however if the module is already connected to a network on the wlan client interface, it will be automatically disconnected from that network first.

If the verification attempt is unsuccessful, the command responds with Timeout. Verification may be unsuccessful if either the

credentials provided are incorrect, or the verification attempt takes longer than 1 second.

Notes

- The `wlan_scan -v` command returns detailed information (including the BSSID, AP channel and security type) of Wi-Fi APs in range. A relevant subset of this information may be passed to the `network_verify` command.
- For WPA/WPA2, the PSK/PMK is calculated by applying the PBKDF2 key derivation function to the WPA/WPA2 passphrase using the SSID as the salt and 4096 iterations of HMAC-SHA1, see [here](#) for additional information.

Syntax

```
> network_verify wifi <ssid> <bssid> <ap channel> [<security> <psk>]
```

where:

- `<ssid>` - the name of the Wi-Fi network
- `<bssid>` - the MAC address of the Wi-Fi Access Point (colons in the MAC are optional)
- `<ap channel>` - the radio channel of the AP
- `<security>` - an enumeration of the security type used by the AP:
 - 1 = WEP
 - 2 = WPA-AES
 - 3 = WPA-TKIP
 - 4 = WPA2-AES
 - 5 = WPA2-Mixed
 - 6 = WPA2-TKIP
- `<psk>` - 64 hex character pre-shared key (aka PMK) for WPA/WPA2 **or** WEP key

Example (Open Security)

```
> network_verify wifi mynetwork 4C55CC102F1A 1
Success
```

Example (WEP40 Security)

```
> network_verify wifi mynetwork 4C55CC102F1A 1 1 1122334455
Success
```

Example (WPA/WPA2 Security)

```
> network_verify wifi mynetwork 4C55CC102F1A 1 4
1205089613366e362586ee5f9ac8f485188fb0bdab30f2f1c23295eb863122de
Success
```

ota

Abbreviation

ota**Description**

Initiate a **secure over-the-air update** using the ota server specified by the variables ota.host, ota.port. The secure OTA protocol uses industry standard TLS/HTTPS security with both client-side and server-side certificate verification. As an additional measure of security, the server authenticates each device using the universally unique **system.uuid** embedded inside the hardware of each device.

Optionally force an update with the -f option. Download a specific firmware bundle version with the -b option.

To activate a module to receive custom firmware & files, use the -a option along with your activation code and password. All modules are activated and licensed for standard WiConnect firmware. If you have firmware customised by ACKme for your application, or you would like ACKme to manage your host firmware or host application files on our secure servers, please [contact ACKme](#) to receive an activation code and password.

To maintain a saved configuration through an OTA upgrade, save the configuration as `default_config.csv`. See [save](#).

Syntax

```
> ota [-f] [-b <bundle name>] [-a <activation code> <password>]
```

Example

```
> ota
Connecting to network
Request POST /ota/05b320df003134534e394b8330333231323a3533/0
Connecting (HTTP): ota.ack.me:443
Starting TLS
HTTP response: 200
Downloading new firmware...
Bundle version: wiconnect-1.0.0.1, Built:2014-04-02 02:23:15 for AMW004.3
Downloading: command_help.csv-1.0.0.1 (25822, 0xFB, 0x1)
Downloading: default_setup.script-1.0.0.1 (189, 0xFD, 0x9)
Downloading: /setup/index.html-1.0.0.1 (9266, 0xFE, 0x1)
Downloading: /setup/images.png-1.0.0.1 (18067, 0xFE, 0x1)
Downloading: upgrade_app.exe-1.0.0.1 (352804, 0x0, 0x801F)
Downloading: wifi_fw.bin-1.0.0.1 (191677, 0x1, 0x8009)
Downloading: wiconnect.exe-1.0.0.1 (83848, 0x80, 0x801F)
Downloading: geotrust_ca.pem-1.0.0.1 (1162, 0x3, 0x9)
Downloading: /favicon.ico.gz-1.0.0.1 (1853, 0xFE, 0x1)
Downloading: /setup/index.css.gz-1.0.0.1 (10539, 0xFE, 0x1)
Downloading: /setup/index.js.gz-1.0.0.1 (38771, 0xFE, 0x1)
Updating Upgrade App to version: 1.0.0.1
Updating firmware files...
Updating file: wifi_fw.bin to version: 1.0.0.1
Updating file: wiconnect.exe to version: 1.0.0.1
Found new bootable app: wiconnect.exe (47), booting to it now!
OTA completed successfully
```

[Ready]

P

ping

Abbreviation

ping

Description

Send one or more ICMP pings to an IP address or the network gateway using the -g option. If provided, the network interface overrides the default specified by the variable `network.default_interface`. Options must be provided in the order shown.

Syntax

```
> ping [-i <wlan/softap>] <[IP address]/[-g]> [# retries]
```

Example

```
> ping example.com 3
R000021
Ping reply in 183ms
R000021
Ping reply in 178ms
R000021
Ping reply in 177ms
```

pwm_update

Abbreviation

pwm

Description

Drive a GPIO with a pulse width modulated signal with `duty_cycle` in the range 0-100%. The PWM frequency has a module-dependent default and may be optionally set with the `frequency` option. Use the `stop` option to turn the PWM off.

See [Peripherals, PWMs](#) for frequency defaults and PWM groups for each module. GPIOs in the same PWM group cannot be used independently at the same time.

Syntax

```
> pwm <GPIO#> <duty_cycle / stop> [frequency]
```

Example

```
> pwm_update 10 30
R000009
Success
```

R**reboot****Abbreviation****reboot****Description**

Reboot the application.

Syntax

```
> reboot
```

Example

```
> reboot
R000082
wiconnect-1.0.0.1, Built:2014-04-02 02:23:15 for AMW004.3, Board: AMW004.3-E03.1
```

S**save****Abbreviation****save****Description**

Save all WiConnect variables to non-volatile flash memory.

If a file name is supplied, the saved configuration can be restored with the [load](#) command.

Save as `default_config.csv` to create a default settings file. If a file of this name is found, WiConnect loads the configuration in `default_config.csv` after a successful OTA or if the factory reset GPIO is asserted for more than 5 seconds but less than 10 seconds. See [Upgrade and Recovery, Factory Reset](#).

Syntax

```
> save [<config_file>]
```

Example

```
> save
L000007
Saved
R000009
Success
```

Example 2

```
save config1.cfg
R000009
Success
ls
! # Size Version Filename
...
# 5 3868    1.0.0 config1.cfg
...
```

set

Abbreviation

set

Description

Set the value of a WiConnect variable.

Syntax

```
> set <variable> <args>
```

Note: The `set` command displays the `Too many args` error only if the number of space-separated arguments exceeds the maximum for all variables (5). However the number of arguments for the `set` command depends on the variable being set, and the form in which the variable is specified: abbreviations for variables vary from one to three arguments in length, and variable settings may require one or two arguments. If you specify too many arguments when setting a specific variable, surplus arguments are

ignored. Refer to the variable documentation for the number of arguments for a specific variable.

Example

```
> set wlan.ssid "My Home Network"  
R000008  
Set OK
```

setup

Abbreviation

setup

Description

The setup command simplifies the process of connecting an ACKme module to a Wi-Fi network. It can also be used to automate device configuration and setup.

The setup command allows for set up from the WiConnect terminal, using a command script, or via a web browser.

See [Configuration and Setup, Configuration Script](#) and [Configuration and Setup, Setup with a Web Browser](#).

The setup command can be run automatically on boot or in response to asserting a GPIO. See [Configuration and Setup, Automatically Executing a Script](#).

Syntax

```
> setup <web/status/stop/cmd [-v] [<script_file>]>
```

Option	Description
web	When called with the web option, setup starts a Wi-Fi Access Point and a web server, the Wi-Fi AP credentials are configured using the variables <code>setup.web ssid</code> and <code>setup.web passkey</code> . Once connected to the WiConnect Setup AP, a Wi-Fi client uses a web browser to setup WiConnect variables and connect to your local Wi-Fi network. Web setup is non-blocking.
status	The status option allows an MCU host to determine whether setup is in-progress. Response is 1 if setup is in progress, otherwise 0.
stop	The stop option terminates any in-progress setup activity.
cmd	When called with the cmd option, setup runs the script <code>default_setup.script</code> located on the filesystem, or alternatively the optional command <code><script_file></code> provided. The setup command blocks until completion when used with the cmd option.
-v	Execute script verbosely, displaying all script comments, commands and output. The <code>default_setup.script</code> is always executed verbosely.

<script_file>	Default when omitted: default_setup.script The setup script can be used to run WiConnect commands in sequence. See Configuration and Setup, Configuration Script .
---------------	--

Example

```
> setup cmd
> Scan and select a network:
network_up -s
Scanning for networks...
! 4 networks found
! # Ch RSSI MAC (BSSID)      Network (SSID)
# 0 06  -35 84:1B:5E:29:9D:F7 GameOverMan
# 1 06  -88 28:C6:8E:70:93:18 Philips Home Network_EXT
# 2 11  -65 EC:1A:59:36:5B:6C laboite_xt
# 3 11  -73 2C:B0:5D:31:6F:6A laboite

Type the number # that matches your Network: 0
Type the password for your Network      : pullyourselftogetherhicks
[Associating to GameOverMan]
In progress
> Enabling network auto-join
set wlan.auto_join.enabled true
Set OK
> Saving settings
save
Saved
Success
> Security type from probe: WPA2-Mixed
Obtaining IPv4 address via DHCP
IPv4 address: 192.168.0.32
[Associated]
> Exiting Cmd Setup Mode
```

sleep

Abbreviation

sleep

Description

Put the application into a low-power sleep state. The application sleeps until the wakeup timeout ([system.wakeup.timeout](#)) expires or a wakeup event ([system.wakeup.events](#)) occurs.

Syntax

```
> sleep
```

Example

```
> set system.wakeup.events uart0|gpio5
```

```
Set OK
```

```
> set system.wakeup.timeout 10
```

```
Set OK
```

```
> sleep
```

```
... wait 10 seconds ...
```

```
wiconnect-1.1.1.0, Built:2014-04-02 02:23:15 for AMW004.3, Board:AMW004.3-E03.1
```

smtp_send

Abbreviation

smtp

Description

Send email to a specified email address. See SMTP variables:

- `email.name_address`
- `email.smtp.host`
- `email.smtp.password`
- `email.smtp.port`
- `email.smtp.username`

Syntax

```
> smtp <to address> <subject> <content length>  
[<content length> bytes of data ...]
```

where:

- `<to address>` - email address to send email to
- `<subject>` - the subject of the email, must be enclosed in double-quotes if spaces are used in the subject.
- `<content length>` - the length in bytes/characters of the email contents.

Immediately after issuing the command, `<content length>` characters of data should be sent. This is the body of the email.

Note that most SMTP servers require SSL/TLS encryption. Some servers, including [gmail](#), use additional authentication methods like OAUTH by default. These additional authentication methods are not supported and may need to be disabled server-side.

Example

```
> smtp_send first.last@youremail.com "World greetings" 5
Hello
[Associating to MY_NETWORK]
Security type from probe: WPA2-AES
Obtaining IPv4 address via DHCP
IPv4 address: 10.5.6.59
[Associated]
Success
```

stream_close

Abbreviation

close

Description

Close an open network or file stream specified by [handle], or alternatively close all open streams.

Syntax

```
> stream_close <[handle]/[all]>
```

Example

```
> close all
[2014-04-02 | 13:32:04: Disconnected: 0]
[2014-04-02 | 13:32:04: Disconnected: 1]
Success
```

stream_list

Abbreviation

list

Description

Return a list of open streams.

Syntax

```
> stream_list
```

Example

```
> list
!# Type Info
#0 TCPC google.com:80 (32288)
#1 HTTP ack.me:80 (37289)
```

stream_poll

Abbreviation

poll

Description

Poll a stream to check if data is available. Returns 0 if open with no data to read, 1 if data is available and 2 if the stream has been closed remotely. Use the `all` option to return the status of all open streams. To avoid polling, use a GPIO (may be assigned when the stream is opened) as an interrupt line to the host.

Syntax

```
> stream_poll [all] <handle>
```

Example

```
> poll 0
1
> read 0 100
hello!
> poll 0
0
> poll all
0,0|2,1|3,1
```

stream_read

Abbreviation

read

Description

Read up to `<size>` bytes from a network or file stream. Returns data immediately (if available), does not wait to receive data. For network streams, WiConnect may buffer up to seven packets with each packet containing up to 1440 (MTU) data bytes. To avoid receiving data fragmented across packets, `<size>` should be set to 1440 bytes. To check for data, use the `stream_poll` command or the GPIO (assigned when the stream was opened) as an interrupt line to the host. For file streams, the file is automatically closed if the end of file is reached.

Syntax

```
> stream_read <handle> <size>
```

Example

```
> read 0 200
R000202
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage"
 lang="en-AU"><head><meta content="/images/google_favicon_128.png"
 itemprop="image"><title>Google</title><script>(function(){
wind
```

stream_write**Abbreviation****write****Description**

Write data to a network or file stream. As described in [Serial Interface, Command Format](#), the command must be terminated by \r\n. After issuing the command, WiConnect waits for <size> number of data bytes then immediately sends the data to the stream **before** returning a response.

Syntax

```
> stream_write <handle> <size>
```

Example

```
> write 3 10
Success
```

T**tcp_client****Abbreviation****tcpc****Description**

Open a TCP stream to a remote server at the network location <host>:<port>. If provided, the network interface overrides the default specified by the variable network.default_interface, and GPIO# specifies which GPIO indicates buffered data is waiting to be read. Options must be provided in the order shown.

Syntax

```
> tcp_client [-i <wlan/softap>] [-g GPIO#] <host> <port>
```

Example

```
> tcpc google.com 80
L000028
Resolving host: google.com
L000052
[2014-03-24 | 01:07:12: Connecting: google.com:80]
L000036
Connecting (TCP): 74.125.237.96:80
L000039
[2014-03-24 | 01:07:12: Connected: 0]
R000003
0
```

tcp_server

Abbreviation

tcps

Description

Start/stop a TCP server specified by the variables `tcp.server.*`. If provided, the network interface and port override defaults specified by the variables `network.default_interface` and `tcp.server.port` respectively. Options must be provided in the order shown.

Syntax

```
> tcp_server [-i <wlan/softap>] <start/stop> [port]
```

Example

```
> tcps start
L000036
TCP server listening on port: 3000
R000009
Success
```

tls_client

Abbreviation

tlsc

Description

Open a secure TCP stream to a remote server at the network location <host>:<port>. If provided, the network interface and CA cert override defaults specified by the variables `network.default_interface` and `network.ca_cert` respectively. The optional `GPIO#` argument specifies which GPIO indicates buffered data is waiting to be read. Options must be provided in the order shown.

Syntax

```
> tls_client [-i <wlan/softap>] [-g GPIO#] <host> <port> [ca_cert_filename]
```

Example

```
> tlsc www.google.com 443
L000032
Resolving host: www.google.com
L000057
[2014-03-24 | 01:15:53: Connecting: www.google.com:443]
L000038
Connecting (TLS): 74.125.237.211:443
L000039
[2014-03-24 | 01:15:55: Connected: 1]
R000003
1
```

tls_server

Abbreviation

tlss

Description

Start/stop a TLS server on the specified port.

If starting the server and no port is specified, the default is the port specified by `tcp_server_port`.

The server certificates must be pre-configured.

Syntax

```
tls_server <start/stop> [port]
```

Example

```
> tlss start 5000
```

U**udp_client****Abbreviation****udpc****Description**

Open a UDP stream to a remote server at the network location <host>:<port>. If provided, the network interface overrides the default specified by the variable `network.default_interface`, and `GPIO#` specifies which GPIO indicates buffered data is waiting to be read. The [local port] may also be provided if desired. Options must be provided in the order shown.

Syntax

```
> udp_client [-i wlan/softap] [-g GPIO#] <host> <remote port> [local port]
```

Example

```
> udpc 192.168.45.67 3000
L000057
[2014-03-24 | 01:16:40: Connecting: 192.168.45.67:3000]
L000031
Resolving host: 192.168.45.67
L000032
Connecting: 192.168.45.67:3000
L000039
[2014-03-24 | 01:16:40: Connected: 2]
R000003
2
```

udp_server**Abbreviation****udps****Description**

Start and stop the UDP server.

When the UDP server is started the read and write sub-commands are available.

When the UDP server is started, it is assigned a stream handle, indicated in the WiConnect response by Opened: <stream>. This allows for the equivalent stream 'read' and 'write' commands.

The read command is functionally the same as: udps read.

The write command is functionally the same as udps write without the <address> and <port> options.

Syntax

```
> udp_server <start/stop> [port]
```

Start and stop the UDP server. Port may be optionally specified. If no port is specified the default is **udp.server.port**.

```
> udp_server read [-q] [<byte_count>]
```

where :

- -q - optionally prepend the client's IP address and port before the data bytes, in a comma-separated list: ip, port, data
- <byte_count> - The number of bytes to read. Note that only a single packet's worth of data is returned
i.e. data returned = MIN(<byte_count>, len(packet))

```
> udp_server write <byte_count> [<address> <port>]
```

where :

- <byte_count> - the number of data bytes to write
- <address>, <port> - optional, the remote client's address and port. If omitted, write to last client whose data has been read.

Example

```
> udp_server start 5000
[2014-11-28 | 07:14:49: Opened: 0]
0
UDP server listening on port: 5000
Success
> udps stop
[2014-11-28 | 07:15:31: Closed: 0]
UDP server stopped
Success
> udp_server write 0 5
Hello
Success
> udp_server read
> udp_server read -q 100
10.5.6.60,52333>Hello World!
> write 0 8
Hi there
```

Success

> read 0

V

version

Abbreviation

ver

Description

Returns the WiConnect firmware bundle version. This is a convenience command that is equivalent to reading the system.version variable.

Syntax

version

Example

```
> ver
R000069
wiconnect.exe v1.0.0.0, Built:Apr 03 2014 for AMW004.3-E03.1
```

W

wlan_get_rssi

Abbreviation

rssi

Description

Get the received signal level (RSSI, in dBm) of the Access Point the wlan interface is connected to. The value is averaged to improve accuracy by setting the variable wlan.rssi_average.

Syntax

```
> wlan_get_rssi
```

Example

```
> rssi  
R000005  
-47
```

wlan_scan**Abbreviation****scan****Description**

Initiate a Wi-Fi scan and return results; optionally specify a channel and AP SSID to scan for. For verbose scans, -v must be the first argument. Information returned by scan results is described in the following table.

Name	Description
Ch	The WLAN radio channel the AP is operating on
RSSI	Signal strength of the AP in units of dBm
BSSID	MAC address of the AP
Rate	The maximum data rate supported by the AP in kilobits/s e.g. 54000 = 54Mbit/s
Security	Security type used by the AP e.g. Open, WPA2-AES, etc.
Mode	AP mode. Inf = Infrastructure, Ad = Ad-hoc
Len	Number of characters in the SSID
SSID	AP name. APs with a hidden SSID are displayed as <ssid hidden>

Syntax

```
> wlan_scan [-v] [<channel> [ssid]]
```

Example

```
> scan -v  
R000196  
! 2 found  
! # Ch RSSI BSSID Rate Security Mode Len SSID  
# 0 01 -24 EC:1A:59:36:5B:6C 144.4 Open Inf 13 Ch1_Nearby_AP  
# 1 06 -85 84:1B:5E:29:9D:F7 450.0 WPA2-AES Ad 14 Ch6_Distant_AP
```

wps***Abbreviation*****wps*****Description***

Runs Wi-Fi Protected Setup (WPS) to enable the wlan interface to connect with an Access Point. PIN and push-button (PBC) modes are supported, PBC mode is default. To specify a timeout for PBC mode, use \0 as the first argument.

Syntax

```
> wps [<PIN code>/<\0>] [timeout in seconds] [stop]
```

Example

```
> wps
L000023
Starting WPS PBC mode
R000013
In progress
```

WiConnect Variables

This page provides a list of WiConnect variables with a full description of the function of each variable together with example usage.

Variables are cached in volatile RAM and must be saved to non-volatile flash memory to persist between reboots. To save variables to flash, use the `save` command. Some variables impact the operation of the entire system, wake and sleep timers for example. A `save` and `reboot` is required before new settings for these types of variables take effect.

Shortcuts

A shortcut is assigned to each variable. The format for shortcuts is shown in the following table. The shortcut is a hybrid of the first two characters of the first word of the variable name, plus (typically) the first character of the second word (if a second word exists), plus (typically) the first character of the third word of the variable name (if a third word exists). Each of these characters is separated by a space character. In some instances, it was not possible to use the first letter of the second or third word in the variable name, in these cases another letter has been somewhat arbitrarily chosen.

	Variable Name	Shortcut
Template	first_word.second_word.third_word	fi s t
Example 1	time.uptime	ti u
Example 2	wlan.scan.retries	wl s r

Documentation Format

Many WiConnect responses shown in the examples on this page were captured with `system.print_level = 0`, and `system.cmd.header_enabled = 1`. These **machine friendly settings** make it easy for a host MCU to parse responses by examining **response headers**. Documentation for each variable is provided in the format shown below.

variable

Abbreviation	<code>var</code>
Access	get/set

Description

A description of variable function.

Arguments

A full list of mandatory and optional arguments.

Default

The factory reset default value.

Get example

An example of how to read the variable, including response codes.

Set example

An example of how to write the variable, including response codes (for writeable variables).

NOTE! Don't forget to check out [command navigation tips](#) to make it easier to find and type specific variable names.

List of Variables

- **ALL Variables**
 - all
- **Broadcast**
 - broadcast.data
 - broadcast.http.host
 - broadcast.interface
 - broadcast.interval
 - broadcast.udp.data
 - broadcast.udp.interface
 - broadcast.udp.interval
 - broadcast.udp.ip
 - broadcast.udp.port
- **Bus**
 - bus.data_bus
 - bus.log_bus
 - bus.mode
 - bus.stream.cmd_gpio
 - bus.stream.cmd_seq
 - bus.stream.flush_count
 - bus.stream.flush_time
 - bus.stream.flush_time_reset
- **Email**
 - email.name_address
 - email.smtp.host
 - email.smtp.password
 - email.smtp.port
 - email.smtp.username
- **goHACK.me**
 - ghm.capabilities
 - ghm.cache_size
 - ghm.solo.echo
 - ghm.solo.enabled
 - ghm.solo gpio
 - ghm.solo.sync_period
 - ghm.status
- **GPIO**
 - gpio.alias
 - gpio.config_file
 - gpio.init
 - gpio.sleep
 - gpio.usage
- **HTTP Server**
 - http.server.api_enabled
 - http.server.cors_origin
 - http.server.denied_filename
 - http.server.enabled
 - http.server.interface

- `http.server.max_clients`
- `http.server.notfound_filename`
- `http.server.password`
- `http.server.port`
- `http.server.root_filename`
- `http.server.tls_cert`
- `http.server.tls_enabled`
- `http.server.tls_key`
- `http.server.tls_verify_peer`
- `http.server.username`
- **IO Connection**
 - `ioconn.control_gpio`
 - `ioconn.enabled`
 - `ioconn.local_port`
 - `ioconn.protocol`
 - `ioconn.remote_host`
 - `ioconn.remote_port`
 - `ioconn.status_gpio`
- **mDNS**
 - `mdns.enabled`
 - `mdns.interface`
 - `mdns.name`
 - `mdns.service`
 - `mdns.ttl`
- **Network**
 - `network.buffer.rtxb_ratio`
 - `network.buffer.size`
 - `network.buffer.usage`
 - `network.ca_cert`
 - `network.default_interface`
 - `network.dhcp`
 - `network.dhcp.enabled`
 - `network.dhcp.timeout`
 - `network.dns`
 - `network.gateway`
 - `network.ip`
 - `network.netmask`
 - `network.status`
 - `network.status_gpio`
- **NTP**
 - `ntp.enabled`
 - `ntp.interval`
 - `ntp.server`
- **OTA Upgrade**
 - `ota.port`
 - `ota.host`
- **Remote Terminal**
 - `remote_terminal.enabled`
 - `remote_terminal.interface`

- `remote_terminal.password`
- `remote_terminal.port`
- `remote_terminal.timeout`
- **Setup**
 - `setup.auto.cmd`
 - `setup.gpio.cmd`
 - `setup.gpio.control_gpio`
 - `setup.gpio.level`
 - `setup.gpio.mode`
 - `setup.web.client_list`
 - `setup.web.passkey`
 - `setup.web.root_filename`
 - `setup.web.ssid`
 - `setup.web.url`
- **SoftAP Interface**
 - `softap.auto_start`
 - `softap.channel`
 - `softap.client_list`
 - `softap.dhcp_server`
 - `softap.dns_server`
 - `softap.hide_ssid`
 - `softap.idle_timeout`
 - `softap.info`
 - `softap.ip`
 - `softap.passkey`
 - `softap.rate.protocol`
 - `softap.rate.transmit`
 - `softap.ssid`
 - `softap.url`
- **Static Network Settings**
 - `static.dns`
 - `static.gateway`
 - `static.ip`
 - `static.netmask`
- **Stream Settings**
 - `stream.auto_close`
- **System**
 - `system.adc.vref`
 - `system.activity gpio`
 - `system.activity gpio_level`
 - `system.bflash.cs_gpio`
 - `system.boot_app`
 - `system.cmd.buffered`
 - `system.cmd.echo`
 - `system.cmd.header_enabled`
 - `system.cmd.mode`
 - `system.cmd.prompt_enabled`
 - `system.cmd.timestamp`
 - `system.gotosleep.timeout`

- system.indicator gpio
- system.indicator.state
- system.memory.usage
- system.msg
- system.oob.event_mask
- system.oob.gpio
- system.oob gpio.level
- system.oob.rising_edge_mask
- system.oob.status
- system.powersave.mode
- system.print_level
- system.safemode
- system.uuid
- system.version
- system.wakeup.events
- system.wakeup.timeout
- **TCP Client**
 - tcp.client.auto_interface
 - tcp.client.auto_retries
 - tcp.client.auto_start
 - tcp.client.connect_timeout
 - tcp.client.local_port
 - tcp.client.remote_host
 - tcp.client.remote_port
 - tcp.client.retries
 - tcp.client.retry_period
- **TCP Keepalive**
 - tcp.keepalive.enabled
 - tcp.keepalive.initial_timeout
 - tcp.keepalive.retry_count
 - tcp.keepalive.retry_timeout
- **TCP Server**
 - tcp.server.auto_interface
 - tcp.server.auto_start
 - tcp.server.connected_gpio
 - tcp.server.data_gpio
 - tcp.server.idle_timeout
 - tcp.server.max_clients
 - tcp.server.port
 - tcp.server.tls_cert
 - tcp.server.tls_enabled
 - tcp.server.tls_key
- **Time**
 - time.last_set
 - time rtc
 - time.uptime
 - time.zone
- **UART**
 - uart.baud

- `uart.data`
- `uart.flow`
- `uart.parity`
- `uart.stop`
- **UDP Client**
 - `udp.client.auto_interface`
 - `udp.client.auto_start`
 - `udp.client.remote_host`
 - `udp.client.remote_port`
- **UDP Server**
 - `udp.server.auto_interface`
 - `udp.server.auto_start`
 - `udp.server.data_gpio`
 - `udp.server.lock_client`
 - `udp.server.port`
 - `udp.server.remote_host`
 - `udp.server.remote_port`
- **WLAN Interface**
 - `wlan.antenna`
 - `wlan.auto_join.enabled`
 - `wlan.auto_join.retries`
 - `wlan.auto_join.retry_delay`
 - `wlan.hide_passkey`
 - `wlan.join.result`
 - `wlan.join.retries`
 - `wlan.join.timeout`
 - `wlan.info`
 - `wlan.mac`
 - `wlan.passkey`
 - `wlan.powersave.listen_interval`
 - `wlan.powersave.mode`
 - `wlan.powersave.sleep_delay`
 - `wlan.rate.protocol`
 - `wlan.rate.transmit`
 - `wlan.roam_threshold`
 - `wlan.rssi_average`
 - `wlan.scan.active_dwell`
 - `wlan.scan.channel_mask`
 - `wlan.scan.home_dwell`
 - `wlan.scan.num_probes`
 - `wlan.scan.passive_dwell`
 - `wlan.scan.retries`
 - `wlan.scan.type`
 - `wlan.security`
 - `wlan.ssid`
 - `wlan.tx_power`

Variable Description

ALL

all

Abbreviation	all
Access	get

Description

Returns a list of all variables

Arguments

Default

Get example

```
> get all
R003978
broadcast.data: mac,bssid,channel,ip,ssid,...
broadcast.http.host: None
broadcast.interface: default
broadcast.interval: 10
broadcast.udp.data: mac,bssid,channel,ip,ssid,...
broadcast.udp.interface: default
broadcast.udp.interval: 10
broadcast.udp.ip: 255.255.255.255
broadcast.udp.port: 55555
bus.data_bus: uart0
...
```

Broadcast

broadcast.data

Abbreviation	br d
--------------	-------------

Access	get/set
--------	---------

Description

A comma separated list of information fields to be periodically transmitted in a UDP packet or sent to a remote HTTP server. See the [Broadcast UDP Packet](#) application note.

Arguments

<csv list>

where <csv list> is a comma separated value list constructed from:

List Item	Description
adcX	Current value of ADC X
bssid	wlan interface → BSSID of the AP
channel	Wi-Fi radio channel
gpioX	Current state of GPIO X
ip	Current IP address
mac	wlan MAC address
rssi	wlan interface → RSSI of the AP
ssid	wlan interface → SSID of the AP
time	WiConnect local time
uuid	Hardware ID
version	Firmware version
remote_terminal_port	inbound network port for the remote terminal

Default

mac,bssid,channel,ip,ssid,rssi,remote_terminal_port,time,uuid,version

Get example

```
> get br d
R000033
mac,bssid,channel,version,gpio0
```

Set example

```
> set br d mac,bssid,channel
R000008
Set OK
```

broadcast.http.host

Abbreviation	br h h
Access	get/set

Description

When specified, periodically issue JSON formatted data to the specified host as an HTTP(S) post request. If null, the HTTP post function is disabled. The post may be to an unencrypted or HTTPS secure webserver. If a non-standard HTTP/HTTPS port is required, the port may be appended to the end of the URL, for example myserver.com:3002. The URL must be prefixed with https:// for HTTPS secure connections on the default port 443.

Arguments

<http_host>

Default

Null

Get example

```
> get br h h  
myhttpserver.com  
Set OK
```

Set example

```
> set br h h myhttpserver.com  
Set OK
```

Set example 2

```
> set br h h https://mysecureserver.com  
Set OK
```

broadcast.interface

Abbreviation	br i
Access	get/set

Description

The default interface used to transmit informational packets.

If set to default, uses the interface specified in `network.default_interface`

Arguments

<wlan/softap/default>

Default

default

Get example

```
> get br i
R000006
wlan
```

Set example

```
> set br i softap
R000008
Set OK
```

broadcast.interval

Abbreviation	br t
Access	get/set

Description

The time between transmission of informational packets. Units are in seconds.

To disable both UDP and HTTP broadcast, set the interval to 0.

Arguments

<seconds>

Default

10

Get example

```
> get br t
```

R000004
10

Set example

```
> set br t 20  
R000008  
Set OK
```

broadcast.udp.data

Abbreviation	br u d
Access	get/set

This variable is deprecated and will be removed in the next release of WiConnect. Instead use [broadcast.data](#).

broadcast.udp.interface

Abbreviation	br u i
Access	get/set

This variable is deprecated and will be removed in the next release of WiConnect. Instead use [broadcast.interface](#).

broadcast.udp.interval

Abbreviation	br u t
Access	get/set

This variable is deprecated and will be removed in the next release of WiConnect. Instead use [broadcast.interval](#).

broadcast.udp.ip

Abbreviation	br u a
Access	get/set

Description

Informational UDP packets are sent to this IP address.

To enable general UDP broadcast, set the IP address to 255.255.255.255 (default), i.e

```
set broadcast.udp.ip 255.255.255.255
```

To disable UDP broadcast, set the IP address to 0.0.0.0, i.e

```
set broadcast.udp.ip 0.0.0.0
```

Alternatively, set **broadcast.interval** to 0 to disable both UDP and HTTP broadcast.

Arguments

<IP address>

Default

255.255.255.255

Get example

```
> get br u a
R000017
255.255.255.255
```

Set example

```
> set br u a 192.168.1.255
R000008
Set OK
```

broadcast.udp.port

Abbreviation	br u p
Access	get/set

Description

Informational UDP packets are sent to this port.

Arguments

<port>

Default

55555

Get example

```
> get br u p  
R000007  
55555
```

Set example

```
> set br u p 50007  
R000008  
Set OK
```

Bus**bus.data_bus**

Abbreviation	bu d
Access	get/set

Description

The serial data bus to use for communication with a host.

Arguments

<uart0/uart1>

Default

<platform dependent>

Get example

```
> get bu d  
R000007  
uart1
```

Set example

```
> set bu d uart0  
R000008  
Set OK
```

bus.log_bus

Abbreviation	bu l
Access	get/set

Description

All log messages are sent to this serial bus regardless of bus mode.

Arguments

<uart0/uart1>

Default

<platform dependent>

Get example

```
> get bu l
R000007
uart1
```

Set example

```
> set bu l uart0
R000008
Set OK
```

bus.mode

Abbreviation	bu m
Access	get/set

Description

The serial bus mode.

Arguments

<command/stream>

Default

command

Get example

```
> get bu m  
R000009  
command
```

Set example

```
> set bu m command  
R000008  
Set OK
```

bus.stream.cmd_gpio

Abbreviation	bu s g
Access	get/set

Description

The GPIO used to force the module into command mode (from stream mode). A value of -1 disables this feature.

Arguments

<[GPIO#]/[-1]>

Default

0

Get example

```
> get bu s g  
R000003  
0
```

Set example

```
> set bu s g 6  
R000008  
Set OK
```

bus.stream.cmd_seq

Abbreviation	bu s s
Access	get/set

Description

Break-in sequence for stream mode. After 3 seconds of inactivity on the serial interface, sending the break-in sequence changes stream mode to command mode; send 'exit' to return to stream mode.

Note: The length of the break-in sequence affects the allowed minimum value of **bus.stream.flush_count**.

Arguments

<1-15 characters>

Default

\$\$\$

Get example

```
> get bu s s
R000005
$$$
```

Set example

```
> set bu s s ####
R000008
Set OK
```

bus.stream.flush_count

Abbreviation	bu s c
Access	get/set

Description

Number of bytes received on the serial bus before flushing to the network stream.

Arguments

<count>

where <count> can range from **minimum value** to **maximum value**:

- **minimum value:** MAX(16, len(bus.stream.cmd_seq)*2)
i.e. the minimum value is the maximum of 16 and 2 x the length of the **bus.stream.cmd_seq** variable
For example, if bus.stream.cmd_seq is the default \$\$\$, its length is 3, so minimum value is the maximum of 16 and 2*3 = 16.
- **maximum value:** 1460

Default

1460

Get example

```
> get bu s c
R000006
1460
```

Set example

```
> set bu s c 1450
R000008
Set OK
```

bus.stream.flush_time

Abbreviation	bu s t
Access	get/set

Description

Timeout in milliseconds before bytes are automatically flushed to the network stream.

If set to 0, automatic flush is disabled.

To achieve optimal throughput, set flush time to 0. WiConnect transfers data with 'Zero Copy'. i.e. a data packet is directly transferred from the bus to network or network to bus.

Arguments

<0-65535>

Default

50

Get example

```
> get bus t  
R000004  
20
```

Set example

```
> set bus t 10  
R000008  
Set OK
```

bus.stream.flush_time_reset

Abbreviation	busr
Access	get/set

Description

Flag whether to reset the flush timer if data is received before the flush timeout expires. When this variable is set and data is streaming, data is flushed only when the buffer is full. This allows for efficient use of the packet buffer and prevents packet fragmentation. When the variable is reset, the buffer is flushed at intervals of **bus.stream.flush_time** and also when the buffer is full. This allows data to be sent at consistent intervals.

Arguments

<1/0>

Default

0

Get example

```
> get bus.stream.flush_time_reset  
0
```

Set example

```
> set bus.stream.flush_time_reset 1  
Set OK
```

Email**email.name_address**

Abbreviation	em n
Access	get/set

Description

The email name and/or address registered with the SMTP server used by the **smtp_send** command. This is shown as the name and/or email address of the email sender. It may be overridden by the SMTP server.

The name and/or email address is supplied in the standard format for Internet Address Specifications. For further information see [Section 3.4, RFC-2822](#).

The following get example shows an address only format, and the set example shows an example of a name and address.

Arguments

<name_address>

in either of the forms:

- local-part@domain
- display-name <local-part@domain>

Default

None

Get example

```
> get email.name_address
no.name@example.com
```

Set example

```
> set em n "Firstname Lastname <first.last@mydomain.com>"
Set OK
> get em n
Firstname Lastname <first.last@mydomain.com>
```

email.smtp.host

Abbreviation	em s h
Access	get/set

Description

The SMTP server host (e.g. `smtp.myemail.com`) used by the `smtp_send` command.

Arguments

<host string>

Default

None

Get example

```
> get email.smtp.host
smtp.example.com
```

Set example

```
> set em s h smtp.myemail.com
Set OK
```

email.smtp.password

Abbreviation	em s w
Access	get/set

Description

The SMTP server login password used by the `smtp_send` command. This is optional depending on the SMTP host: `email.smtp.host`.

Arguments

<password>

Default

None

Get example

```
> get email.smtp.password
abc123
```

Set example

```
> set em s w loooong_SECR3T_password
```

Set OK

email.smtp.port

Abbreviation	em s p
Access	get/set

Description

The SMTP server port used by the **smtp_send** command. Standard non-encrypted port is 25. Standard encrypted (TLS) port is 587.

Arguments

<port_number>

Range: 1 - 65535

Default

587

Get example

```
> get email.smtp.port
25
```

Set example

```
> set em s p 587
Set OK
```

email.smtp.username

Abbreviation	em s u
Access	get/set

Description

The SMTP server login username used by the **smtp_send** command. This is optional depending on the SMTP host:
email.smtp.host.

Arguments

```
<user_name>
```

Default

None

Get example

```
> get email.smtp.username
No Name
```

Set example

```
> set email.smtp.user "Firstname Lastname"
Set OK
> get em s u
Firstname Lastname
```

goHACK.me**ghm.capabilities**

Abbreviation	gh c
Access	get

Description

Returns the name of the goHACK.me capabilities file used by the device. Use the **ghm_capabilities** command to manipulate the capabilities file.

Arguments

-

Default

ghm_capabilities.json

Get example

```
> get ghm.capabilities
R000023
ghm_caps_pwm_adc.json
```

ghm.cache_size

Abbreviation	gh s
Access	get/set

Description

The maximum number of stream samples to store locally (per stream) before samples are automatically pushed to goHACK.me. All cached stream samples are automatically pushed to goHACK.me when the number of cached samples for any stream exceeds the cache size.

Arguments

<#samples>

Default

50

Get example

```
> get ghm.cache_size
R000004
50
```

Set example

```
> set ghm.cache_size 20
R000008
Set OK
```

ghm.solo.echo

Abbreviation	gh o c
Access	get/set

Description

After the device synchronizes with goHACK.me, any control parameter changes are echoed to the **bus.log_bus** in a comma separated list format.

Arguments

<0/off/false/1/on/true>

Default

0

Get example

```
> get ghm.solo.echo
0
```

Set example

```
> set ghm.solo.echo 1
Set OK
```

ghm.solo.enabled

Abbreviation	gh o e
Access	get/set

Description

In solo mode, the device automatically monitors and controls GPIOs without help from an external host. GPIOs connected to streams are automatically sampled and pushed to goHACK.me, and control parameters are automatically pulled from goHACK.me and new values are applied to control GPIOs.

Arguments

<0/off/false/1/on/true>

Default

0

Get example

```
> get ghm.solo.enabled
0
```

Set example

```
> set ghm.solo.enabled 1
Set OK
```

ghm.solo.gpio

Abbreviation	gh o g
Access	get/set

Description

Assign a GPIO to a goHACK.me stream or control parameter in solo mode. Streams are GPIO inputs, controls are GPIO outputs.

Arguments

<GPIO#> <slug/none> [<function> [config]]

where parameters are as follows:

- <GPIO#>: The GPIO number of an unassigned GPIO
- <slug/none> The name of a slug in the capabilities file. If a stream and control have the same slug name, then the slug must be prefixed with stream or control
e.g. control.door_open, stream.door_open
Set to none to free the GPIO from a previous assignment.
- [<function>] Specify a valid function for the GPIO (optional)

Available functions are:

Function	Description
in	Input, high impedance (default for streams)
inpu	Input, pull up
inpdu	Input, pull down
out	Output (default for controls)
out_od	Output, open drain
out_od_pu	Output open drain, pull up
adc	Analog-Digital Converter. The default value is specified in the capabilities file.
dac	Digital-Analog Converter. The default duty cycle and (optional) default frequency are specified in the capabilities file.
pwm	Pulse Width Modulator output. The default value is specified in the capabilities file.
pulse	Pulsed GPIO output. The inactive level is specified in the capabilities file.

- [config]: Optional configuration specific to GPIO functions.

Available configurations include:

GPIO type	Configuration Specification	Description
Input GPIOs (streams only)	<code>in <rising / falling / both>	Edge detection. A stream sample is taken each time a rising or falling (or both) edge occurs on the GPIO input.
Output GPIOs (controls only)	pulse [length]	Pulsed output. If pulse is specified, the GPIO output is pulsed once each time the control parameter is set. The pulse length is specified in multiples of 10 milliseconds and defaults to 50 ms if the [length] option is omitted.
ADC lookup table	adc [<LUT filename>]	Analog-Digital Converter lookup table. See Peripherals, ADC Lookup Tables . Example: <code>set ghm.solo gpio 7 thermistor adc therm_celsius_lut.csv</code>
Pulse Width Modulation	pwm [<frequency name>]	The name of a control in the capabilities file that may be used to set the PWM frequency. If the frequency is not specified, it defaults to 1000 Hz.
Pulsed GPIO	pulse [<pulse length>]	Set the one-shot pulse length in units of tens of milliseconds.

Default

-

Get example

```
> get ghm.solo gpio 10
control.led1 - opp
```

Set example

```
> set gpio.alias 10 GPIO10
Set OK
> set ghm.solo gpio GPIO10 led1
Set OK
```

ghm.solo.sync_period

Abbreviation	gh o s
Access	get/set

Description

Period (in seconds) to automatically sync with goHACK.me.

Arguments

[push/pull]

Default

20

Get example

```
> get ghm.solo.sync_period  
20
```

Set example

```
> set ghm.solo.sync_period 4  
Set OK
```

ghm.status

Abbreviation	gh t
Access	get

Description

Get the device activation status. Local query only, does not query the goHACK.me server.

Arguments

-

Default

0

Get example

```
> get ghm.status  
Device is activated  
1
```

GPIO

gpio.alias

Abbreviation	gp a
Access	get/set

Description

A GPIO alias name between 1-15 characters. GPIO name aliasing avoids the need to remember which GPIO is connected where. You can give the GPIO a recognisable name instead! Retrieve aliases with `get gp a all`. Remove an alias by setting to an empty double quoted string ("").

Arguments

`<all> / <GPIO#> <alias name>`

Default

Get example

```
> get gp a 6
R000008
MyGPIO
```

Set example

```
> set gp a 6 FrontDoor
R000008
Set OK
```

gpio.config_file

Abbreviation	gp c
Access	get/set

Description

A file used to configure and initialize GPIOs on bootup. The file may have multiple lines and must follow a specified format. Further information about GPIO configuration using system indicators is provided in [Configuration and Setup, GPIO Configuration](#).

Arguments

<filename>

Default

gpio_config_init.csv

Get example

```
> get gp c
R000022
gpio_config_init.csv
```

Set example

```
> set gp c my_gpio_config.csv
R000008
Set OK
```

gpio.init

Abbreviation	gp i
Access	get/set

Description

The initial direction and value of a GPIO after bootup/reset. Retrieve a list of GPIOs that will be initialized with the all option.

Arguments

<all> / <GPIO#> <direction> <initial value>

Default

<none for unassigned GPIOs>

Get example

```
> get gp i 0
R000021
output_open_drain 1
```

Set example

```
> set gp i 0 out 1
R000008
Set OK
```

gpio.sleep

Abbreviation	gp s
Access	get/set

Description

Set the state of the specified GPIO when sleeping.

GPIOs are initialized as follows:

- On startup, all unused pins (i.e. MCU pins not assigned a module GPIO) are set to the default state for the MCU (WALLABY= input-pullup, NUMBAT=Analog input)
- Then module GPIOs are set to either their `gpio.init` value if specified, else the MCU default state
- On sleep, module GPIOs are set to their `gpio.sleep` value if specified, else the MCU default state.

Sleep state can also be set using a GPIO configuration file. See [Configuration and Setup, Using a GPIO Configuration File](#).

Arguments

`<all> / <GPIO#> <state>`

where state is one of the following:

State	Description
default	factory default state
input_highz	input high impedance
input_pull_down	input pull down
input_pull_up	input pull up
output_high	output high
output_low	output low

Default

default

Get example

```
> get gp s 0
input_pull_up
```

Set example

```
> set gp s 0 output_low
Set OK
```

gpio.usage

Abbreviation	gp u
Access	get

Description

Returns information about which GPIOs have been registered for functions, and which have standard I/O options set. GPIOs may be de-registered by setting the GPIO to -1 using the appropriate variable. e.g. set ioconn.status_gpio -1. See [Peripherals](#).

Arguments

-

Default

<platform dependent>

Get example

```
> get gp u
! # Description
# 0 GPIO input_highz
# 1 system.indicator.wlan
# 2 system.indicator.network
# 5 system.indicator.softap
# 11 setup.control_gpio
# 13 UART1 RX
# 14 UART1 TX
# 17 SPI0 CLK
# 18 SPI0 MOSI
# 19 SPI0 MISO
# 21 GPIO output
# 22 GPIO output
```

HTTP Server

http.server.api_enabled

Abbreviation	hts a
--------------	-------

Access	get/set
--------	---------

Description

Enable the HTTP Server API. See [Networking and Security, HTTP Server RESTful API](#).

Arguments

<0/1>

Default

1

Get example

```
> get http.server.api_enabled  
1
```

Set example

```
> set ht s a 0  
Set OK
```

http.server.cors_origin

Abbreviation	ht s c
Access	get/set

Description

Specifies origins for which the web browser same-origin policy is relaxed.

If <origin> is set to an empty string, then CORS is completely disabled.

For requests without credentials, a value of * acts as a wildcard, allowing any origin to access the resource.

See [HTTP Server RESTful API, CORS](#).

Arguments

<origin>

Default

Get example

```
get http.server.cors_origin
```

Set example

```
set ht s c *
Set OK
```

http.server.denied_filename

Abbreviation	ht s d
Access	get/set

Description

Specify the file name of the page to be displayed in the event that the HTTP server denies authentication.

Arguments

```
<filename>
```

Default

..

Get example

```
get http.server.denied_filename
```

Set example

```
set ht s d access_denied.html
Set OK
```

http.server.enabled

Abbreviation	ht s e
Access	get/set

Description

Enable the HTTP server. See [Networking and Security, HTTP Server RESTful API](#).

Arguments

<1/0>

Default

0

Get example

```
> get ht s e  
0
```

Set example

```
> set ht s e 1  
Set OK
```

http.server.interface

Abbreviation	hts i
Access	get/set

Description

Specifies the network interface used by the HTTP server. See [Networking and Security, HTTP Server RESTful API](#).

Arguments

<default/softap/wlan>

Default

default

Get example

```
> get http.server.interface  
default
```

Set example

```
> set ht s i wlan  
Set OK
```

http.server.max_clients

Abbreviation	ht s m
Access	get/set

Description

Specifies the maximum number of clients the HTTP server allows to connect.

Restricting connected clients may be necessary to limit memory usage. See [Memory](#).

Arguments

<clients>

Range: 1 - 8

Default

3

Get example

```
> get http.server.max_clients
3
```

Set example

```
> set ht s m 1
Set OK
```

http.server.notfound_filename

Abbreviation	ht s n
Access	get/set

Description

Specifies the filename of the web page to be displayed in the event that the HTTP server reports a 404 - not found error.

<filename>

Default

Get example

```
get http.server.notfound_filename
```

Set example

```
set ht s n notfound.html  
Set OK
```

http.server.password

Abbreviation	ht s w
Access	get/set

Description

Set the password of the http server. The **username** MUST be set as well for the http server to start.

See [HTTP Server Security and Authorization](#).

Arguments

```
<password>
```

Default

```
null
```

Get example

```
> get ht s w  
secretpassword
```

Set example

```
> set ht s w secretpassword  
Set OK
```

http.server.port

Abbreviation	ht s p
--------------	---------------

Access	get/set
--------	---------

Description

Set the port on which the HTTP server listens. See [Networking and Security, HTTP Server RESTful API](#).

Arguments

<listening port>

Default

80

Get example

```
get ht s p  
80
```

Set example

```
set http.server.port 8080  
Set OK
```

http.server.root_filename

Abbreviation	ht s r
Access	get/set

Description

This sets or gets the root file of the HTTP server. If a plain request to `http://<host>` is issued, with no url specified, this file is returned.

See [Networking and Security, HTTP Server RESTful API](#).

Arguments

<filename>

Default

/setup/index.html

Get example

```
> get ht s r  
/setup/index.html
```

Set example

```
> set http.server.root_filename /setup/home.html  
Set OK
```

http.server.tls_cert

Abbreviation	ht s l
Access	get/set

Description

The server TLS CA certificate filename.

Arguments

<filename>

Default

null

Get example

```
> get ht s c  
mycert.pem
```

Set example

```
> set ht s c mycert.pem  
Set OK
```

http.server.tls_enabled

Abbreviation	ht s t
Access	get/set

Description

Enable server Transport Level Security (TLS).

Arguments

<1/0>

Default

0

Get example

```
> get ht s t  
0
```

Set example

```
> set ht s t 1  
Set OK
```

http.server.tls_key

Abbreviation	ht s k
Access	get/set

The server TLS certificate key filename.

Arguments

<filename>

Default

null

Get example

```
> get ht s k  
mycert.key
```

Set example

```
> set ht s k mycert.key  
Set OK
```

http.server.tls_verify_peer

Abbreviation	ht s v
Access	get/set

Description

When this variable is set, a connecting client must provide a valid TLS certificate that the server validates. The client is disconnected if the provided certificate is invalid. **Note:** The [http.server.tls_cert](#) variable must be set if using this feature.

Arguments

<0/1>

Default

0

Get example

```
> get ht s v
0
```

Set example

```
> set ht s v 1
Set OK
```

http.server.username

Abbreviation	ht s u
Access	get/set

Description

Set the username of the http server. The [password](#) MUST be set as well for the http server to start.

See [HTTP Server Security and Authorization](#).

Arguments

<username>

Default

null

Get example

```
> get http.server.username
cyrano
```

Set example

```
> set ht s u cyrano
Set OK
```

IO Connection**ioconn.control_gpio**

Abbreviation	io c
Access	get/set

Description

The GPIO used for control of the GPIO controlled network connection (stream). See the [GPIO-Controlled Network Connection](#) application note.

Arguments

<GPIO#>

Default

-1

Get example

```
> get io c
R000004
-1
```

Set example

```
> set io c 6
R000008
Set OK
```

ioconn.enabled

Abbreviation	io e
Access	get/set

Description

Enable/disable the GPIO controlled network connection feature.

Arguments

<0/off/false/1/on/true>

Default

0

Get example

```
> get io e
R000003
0
```

Set example

```
> set io e true
R000008
Set OK
```

ioconn.local_port

Abbreviation	io l
Access	get/set

Description

The local port associated with the GPIO controlled network connection, If 0, a random port is assigned.

Arguments

<port>

Default

0

Get example

```
> get io 1  
R000003  
0
```

Set example

```
> set io 1 30001  
R000008  
Set OK
```

ioconn.protocol

Abbreviation	io p
Access	get/set

Description

The network protocol associated with the GPIO controlled network connection.

Arguments

<udp/tcp/tls>

Default

tcp

Get example

```
> get io p  
R000005  
tcp
```

Set example

```
> set io p udp  
R000008  
Set OK
```

ioconn.remote_host

Abbreviation	io h
Access	get/set

Description

The remote host associated with the GPIO controlled network connection. An IP address or hostname may be specified.

Arguments

<IP address/hostname>

Default

-

Get example

```
> get > get io h
R000013
10.11.176.12
```

Set example

```
> set io h example.com
R000008
Set OK
```

ioconn.remote_port

Abbreviation	io r
Access	get/set

Description

The remote port associated with the GPIO controlled network connection.

Arguments

<port>

Default

0

Get example

```
> get io r
R000003
0
```

Set example

```
> set io r 55000
R000008
Set OK
```

ioconn.status_gpio

Abbreviation	io s
Access	get/set

Description

The GPIO used to indicate connection status when the GPIO controlled network connection feature is enabled.

Arguments

<GPIO#>

Default

-1

Get example

```
> get io s
R000004
-1
```

Set example

```
> set io s 13
R000008
Set OK
```

mDNS**mdns.enabled**

Abbreviation	md e
Access	get/set

Description

Enable the mDNS daemon (i.e. background process). The process is started when the configured interface is brought up (see [mdns.interface](#)). If the network interface is already up when mDNS is enabled, bring the network interface down and up again to start mDNS (see [network_down](#), [network_up](#)). See [Networking and Security, Network Discovery](#).

Note: When mDNS is enabled, .local domains are first attempted to be resolved using mDNS. If the attempt fails, standard DNS is then attempted.

Arguments

<0/1>

Default

0

Set example

```
> set mdns.enabled 1
Set OK
```

Get example

```
> get md e
1
```

mdns.interface

Abbreviation	md i
Access	get/set

Description

The network interface on which the mDNS daemon runs. See [Networking and Security, Network Discovery](#).

Arguments

<default/softap/wlan>

Default

default

Set example

```
> set mdns.interface wlan
Set OK
```

Get example

```
> get md i
wlan
```

mdns.name

Abbreviation	md n
Access	get/set

Description

The mDNS domain name of the module.

Note:

- .local is appended to the name. See [Networking and Security, Network Discovery](#).
- If the specified name conflicts with a pre-existing mDNS name, a number is automatically appended to the name, e.g. mymodule-2.local.

Arguments

<name>

- Only valid domain characters are supported: a-z, A-Z, 0-9, -, ..
- If the # character is the last character in the name, then it is replaced by the last 6 hex characters of the module's mac address, e.g. wiconnect-1032d6.

Default

wiconnect-#

Set example

```
> set mdns.name MyDomain_#
Set OK
```

Get example

```
> get md n
MyDomain_#
```

mdns.service

Abbreviation	md s
Access	get/set

Description

Advertise a module server via mDNS. See [Networking and Security, Network Discovery](#).

The corresponding server must be enabled and configured to auto-start before the service is advertised via mDNS.

Arguments

set syntax

```
<server name> [<name> [<txt> [<service type>] ] ]
```

where:

<server name> - the name of a wiconnect server. The following servers are available:

- http - http server
- tcp - TCP/TLS server
- udp - UDP server
- remote_terminal - remote terminal server

<name> - optional, an additional name/description of the service. For example: Sensor HTTP Server

NOTE: To disable advertising of an mDNS service, set the <name> argument to -1

<txt> - optional, additional configuration settings for the service. This is a Bonjour style TXT Record, consisting of a string of <name>=<value> pairs, separated by an ASCII . character (code 0x2E). e.g.

```
set mdns.service http "Test Server" "record1=on.record2=off.another=21"
```

<service type> - optional, the service registration type. (Note: .local is appended to the end) By default, the following server/service type is used:

- http server - _http._tcp.local
- tcp server - _wiconnect._tcp.local
- udp server - _wiconnect._udp.local
- remote_terminal - _telnet._tcp.local

get syntax

```
<server name>
```

where: <server name> is one of:

- http
- tcp
- udp
- remote_terminal
- all - list all servers

The response lists the attributes as follows:

```
"<name>" "<txt>" <type>
```

e.g.

```
>get md s http
"Test Server" "record1=on.record2=off.another=21" _http._tcp.local
```

If all is specified to list status of all servers, the response lists the servers and attributes as follows:

```
<server name> - <name>.type <txt>
...
```

e.g.

```
>get md s all
http - Test Server._http._tcp.local record1=on.record2=off.another=21
tcp - my_tcp_server._wiconnect._tcp.local
udp - Disabled
remote_terminal - Disabled
```

Default

-

Set example

Advertise tcp service as my_tcp_server

```
> set md s tcp my_tcp_server
Set OK
> get md s all
http - Disabled
tcp - my_tcp_server._wiconnect._tcp.local
udp - Disabled
remote_terminal - Disabled
```

Disable advertising of tcp service

```
> set md s tcp -1
Set OK
> set md s tcp -1
Set OK
> get md s all
http - Disabled
tcp - Disabled
udp - Disabled
remote_terminal - Disabled
```

Get example

```
> get md s tcp
"my_tcp_server" "" _wiconnect._tcp.local
```

mdns.ttl

Abbreviation	md t
Access	get/set

Description

Sets the timeout of the module's mDNS response in seconds: the “Time-to-Live” of a mDNS response from the module. This is essentially a timeout of the validity of a mDNS record. Once expired, a remote client will reprobe the module for an updated response. See [Networking and Security, Network Discovery](#).

Arguments

<seconds>

where <seconds> is in the range 1 - 65535.

Default

300

Set example

```
> set mdns.ttl 1000
```

Get example

```
> get mdns.ttl
1000
```

Network

network.buffer.rxtx_ratio

Abbreviation	ne b r
Access	get/set

Description

The percentage of the network buffer allocated to Rx.

Rx buffer pool size = network.buffer.size x (network.buffer.rxtx_ratio / 100)

Tx buffer pool size = network.buffer.size x ((100 - network.buffer.rxtx_ratio) / 100)

Value limits (%): Min: 10 Max: 90

Arguments

<percent_rx>

Default

50

Set example

```
> set ne b r 20
R000008
Set OK
```

Get example

```
> get ne b r
R000004
20
```

network.buffer.size

Abbreviation	ne b s
Access	get/set

Description

The total size of the network buffers, in bytes.

This is important for applications that have asymmetric data throughput requirements. If your app is heavily Tx (or Rx) centric, you

can adjust the Tx and Rx buffer sizes accordingly to achieve maximum throughput.

This buffer is divided between the rx/tx buffers based on the `network.buffer.rxtx_ratio` variable.

This value is the amount of memory allocated from the dynamic memory section. The larger this value, the less memory available for other features.

This is an advanced feature. Care should be given when adjusting this value. The system may become unstable with certain values in certain applications.

Value limits (KB = 1024 bytes):

Wallaby : Min: 10KB = 10240 Max: 70KB = 71680

Numbat: Min: 10KB = 10240 Max: 40KB = 40960

Arguments

<bytes>

Default

default: numbat: 14KB = 14336, wallaby: 20KB = 20480

Set example

```
> set ne b s 22528
R000008
Set OK
```

Get example

```
> get ne b s
R000007
22528
```

network.buffer.usage

Abbreviation	ne b u
Access	get

Description

The approximate usage of the network TX & RX buffers as an integer percentage.

Arguments

-

Default

-

Get example

```
> get ne b u
R000013
RX:13,TX:78
```

network.ca_cert

Abbreviation	ne c
Access	get/set

Description

The default CA TLS certificate used with TLS network streams.

Note: PEM file format requires Unix format line termination: LF (\n), not Windows format: CR-LF (\r\n). Some terminal applications may append unwanted end-of-line characters. Incorrect line termination may result in TLS connection failure.

Arguments

```
<CA TLS cert filename>
```

Default

geotrust_ca.pem

Get example

```
> get ne c
R000017
geotrust_ca.pem
```

Set example

```
> set ne c globalsign_ca.pem
R000008
Set OK
```

network.default_interface

Abbreviation	ne f
Access	get/set

Description

The default network interface used by various network commands and variables.

Commands include: [http_upload](#), [http_download](#), [network_up](#), [network_down](#), [ping](#), [tcp_server_start](#), [udp_server_start](#), [tls_server_start](#).

Variables include: [broadcast.udp.interface](#), [remote_terminal.interface](#), [tcp.server.auto_interface](#), [udp.server.auto_interface](#)

Arguments

<wlan/softap>

Default

wlan

Get example

```
> get ne f  
R000006  
wlan
```

Set example

```
> set ne f softap  
R000008  
Set OK
```

network.dhcp

Description

Renamed to [network.dhcp.enabled](#) in WiConnect V2.0.0

network.dhcp.enabled

Abbreviation	ne d e
Access	get/set

Description

Select whether DHCP is used to automatically obtain an IPv4 IP address on the wlan interface.

Note: Renamed from `network.dhcp` in WiConnect v1.2 and earlier.

Arguments

<0/off/false/1/on/true>

Default

1

Get example

```
> get ne d e
R000003
1
```

Set example

```
> set ne d e off
R000008
Set OK
```

network.dhcp.timeout

Abbreviation	ne d t
Access	get/set

Description

Amount of time in seconds the module waits as a client for an IP from a DHCP server. See [network.dhcp.enabled](#).

Arguments

<timeout_seconds>

Range: 1-255 seconds

Default

15

Get example

```
> get ne d t
R000004
15
```

Set example

```
> set ne d t 25
R000008
Set OK
```

network.dns

Abbreviation	ne d
Access	get

Description

Return a comma separated list of the DNS server addresses that the wlan interface is using.

Arguments

Default

0.0.0.0

(i.e. No DNS addresses in use)

Get example

```
> get ne d  
10.5.6.1
```

network.gateway

Abbreviation	ne g
Access	get

Description

Returns the wlan interface IPv4 gateway IP address resolved by DHCP.

Arguments

-

Default

0.0.0.0

Get example

```
> get ne g  
R000013
```

192.168.0.1

network.ip

Abbreviation	ne i
Access	get

Description

Returns the wlan interface IPv4 IP address resolved by DHCP.

Arguments

-

Default

0.0.0.0

Get example

```
> get ne i
R000014
192.168.0.79
```

network.netmask

Abbreviation	ne n
Access	get

Description

Returns the wlan interface IPv4 netmask resolved by DHCP.

Arguments

-

Default

0.0.0.0

Get example

```
> get ne n  
R000015  
255.255.255.0
```

network.status

Abbreviation	ne s
Access	get

Description

Returns the wlan network status. 0 = down, 1 = WLAN connected, 2 = IP network connected (with a valid IP address)

Arguments

-

Default

0

Get example

```
> get ne s  
R000003  
2
```

network.status_gpio

Abbreviation	ne o
Access	get/set

Description

The GPIO used to indicate network connectivity status. GPIO low = not connected, GPIO high = the default network interface is connected and has an IP address.

Arguments

<GPIO#>

Default

-1

Get example

```
> get ne o  
R000004  
-1
```

Set example

```
> set ne o 4  
R000008  
Set OK
```

ntp.enabled

Abbreviation	nt e
Access	get/set

Description

Enable/disable synchronization of local time with network time.

Arguments

<0/off/false/1/on/true>

Default

1

Get example

```
> get nt e  
R000003  
1
```

Set example

```
> set nt e off  
R000008  
Set OK
```

ntp.interval

Abbreviation	nt i
Access	get/set

Description

The synchronization interval used when updating local time with network time.

Arguments

<seconds>

Default

3600

Get example

```
> get nt i
R000006
3600
```

Set example

```
> set nt i 14400
R000008
Set OK
```

ntp.server

Abbreviation	nts
Access	get/set

Description

The IP address or host name of the NTP server used to obtain network time.

Arguments

<IP address / host name>

Default

pool.ntp.org

Get example

```
> get nt s  
R000014  
pool.ntp.org
```

Set example

```
> set nt s 203.26.72.7  
R000008  
Set OK
```

OTA Upgrade**ota.port**

Abbreviation	ot p
Access	get/set

Description

Network port of the remote OTA upgrade server

Arguments

<port><port>

Default

443

Get example

```
> get ot p  
R000005  
443
```

Set example

```
> set ot p 443  
R000008  
Set OK
```

ota.host

Abbreviation	ot h
Access	get/set

Description

IP address or host name to use for the OTA server

Arguments

<[IP address]/[host name]>

Default

ota.ack.me

Get example

```
> get ot h
R000012
ota.ack.me
```

Set example

```
> set ot h ota2.ack.me
R000008
Set OK
```

Remote Terminal**remote_terminal.enabled**

Abbreviation	re e
Access	get/set

Description

Enable/disable remote terminal at bootup.

Arguments

<0/off/false/1/on/true>

Default

0

Get example

```
> get re e  
R000003  
0
```

Set example

```
> set re e true  
R000008  
Set OK
```

remote_terminal.interface

Abbreviation	re i
Access	get/set

Description

Network interface the remote terminal will listen on.
If set to default, uses the interface specified in [network.default_interface](#)

Arguments

<wlan/softap/default>

Default

default

Get example

```
> get re i  
R000009  
default
```

Set example

```
> set re i wlan  
R000008  
Set OK
```

remote_terminal.password

Abbreviation	re w
Access	get/set

Description

Client login password for the remote terminal. Note! The terminal emulator must be configured for passive password negotiation.

Arguments

<password>

Default

""

Get example

```
> get re w
R000000
```

Set example

```
> set re w mypassword
R000008
Set OK
```

remote_terminal.port

Abbreviation	re p
Access	get/set

Description

The remote terminal listens on this port.

Arguments

<port>

Default

2000

Get example

```
> get re p  
R000006  
2000
```

Set example

```
> set re p 55000  
R000008  
Set OK
```

remote_terminal.timeout

Abbreviation	re t
Access	get/set

Description

Remote terminal client idle timeout in seconds.

Arguments

<# seconds>

Default

60

Get example

```
> get re t  
R000004  
60
```

Set example

```
> set re t 120  
R000008  
Set OK
```

Setup

setup.auto.cmd

Abbreviation	se a c
Access	get/set

Description

The WiConnect command specified by setup.auto.cmd is run after boot up if:

- `setup gpio.control_gpio` is NOT asserted AND
- `wlan.ssid` is NOT configured.

This corresponds to the 'out-of-the-box' default state of a module. You can set `setup.auto.cmd` to `setup cmd` to run a setup script file. See [Setup Configuration Script](#).

Arguments

<command>

Default

-

Get example

```
> get se a c
R000005
wps
```

Set example

```
> set se a c "setup web"
R000008
Set OK
```

setup gpio.cmd

Abbreviation	se g c
Access	get/set

Description

The specified command is executed in response to a reboot, or a specified GPIO being asserted, depending on the setting of the `setup gpio.control_gpio` and the `setup gpio.mode`.

Arguments

<command>

Default

setup web

Get example

```
> get se g c
R000011
setup web
```

Set example

```
> set se g c "setup web"
R000008
Set OK
```

setup gpio control_gpio

Abbreviation	se g g
Access	get/set

Description

Asserting the GPIO specified by `setup gpio control_gpio` for 3 seconds during and immediately after reset causes the WiConnect command specified by `setup gpio cmd` to run. A value of -1 disables this feature.

Arguments

<GPIO#>

Default

The default GPIO is platform dependent and is typically mapped to button 2 on evaluation boards (the wake button)

Get example

```
> get se g g
R000003
3
```

Set example

```
> set se g g 12
R000008
Set OK
```

setup gpio.level

Abbreviation	seg l
Access	get/set

Description

The active logic level used when **setup gpio.control_gpio** is asserted.

Arguments

<GPIO#>

Default

1

Get example

```
> get se g 1
R000003
1
```

Set example

```
> set se g 1 0
R000008
Set OK
```

setup gpio.mode

Abbreviation	seg m
Access	get/set

Description

The mode setting determines what events result in running the command string specified by **setup gpio.cmd**.

If the **setup gpio.cmd** is set, the command is run as follows:

Mode	Event Causing Command to Run	Note
boot	The command is run on PoR (Power on Reset, or reboot)	The operation ignores the control GPIO.
gpio	If <code>setup gpio control_gpio</code> is set, the command runs when the GPIO is asserted for 3 seconds through a PoR.	This corresponds to the legacy behavior of running setup web using Button 2 on evaluation boards. Control GPIO is not registered so may be used as another function.
any	If <code>setup gpio control_gpio</code> is set, the command runs any time the GPIO is asserted.	Control GPIO is registered so cannot be used as another function.

Otherwise the command does not run.

Arguments

<boot/gpio/any>

Default

Get example

```
> get se g m
gpio
```

Set example

```
> set se g m boot
R000008
Set OK
```

setup.web.client_list

Abbreviation	se w c
Access	get

Description

List MAC address, IP address and OS of connected websetup clients

Note: 'Invalid' means the client has not yet connected to the webpage (but is connected to the softap).

The additional info comes from the browser's 'User-Agent' field in its HTTP request header.

A list of possible 'User-Agent' values is found here: <http://www.useragentstring.com/pages/useragentstring.php>.

The following is a list of supported OSs:

Windows

Windows Mobile

Linux
Mac
Android
iPhone
iPod
iPad
Blackberry
Chrome
Symbian
Unknown

Arguments

-

Default

-

Get example

```
> get setup.web.client_list
R000143
! Connected: 2
! # MAC           IP Address   OS
# 1 90:18:7C:34:CB:97 10.10.10.2   Android
# 2 F8:16:54:35:0F:8E 0.0.0.0     Windows
```

setup.web.passkey

Abbreviation	se w p
Access	get/set

Description

The softAP WPA2-AES passphrase used for web setup, must be between 8-32 characters in length.

Arguments

<passphrase>

Default

password

Get example

```
> get we p  
R000010  
password
```

Set example

```
> set we p my-new-password  
R000008  
Set OK
```

setup.web.root_filename

Abbreviation	se w r
Access	get/set

Description

The root filename of the web setup: in response to a request for <http://wiconnect.com>, the web server returns http://wiconnect.com/<root_filename>.

Note: The file must be a supported type for it to appear correctly in the browser. The type is determined by the file extension. The following types are supported:

Extension	MIME
.css	text/css
.js	application/javascript
.html/.htm	text/html
.png	image/png
.ico	image/x-icon
.gif	image/gif
.jpg/.jpeg	image/jpeg
.txt	text/plain

Arguments

<root_filename>

Default

/setup/index.html

Get example

```
> get se w r  
/setup/index.html
```

Set example

```
> set se w r home.htm  
R000008  
Set OK
```

setup.web.ssid

Abbreviation	se w s
Access	get/set

Description

The softAP SSID used for web setup. If the SSID ends in a # character, the last 6 digits of the wlan MAC address are automatically appended to the SSID.

Arguments

<ssid>

Default

WiConnect Web Setup #

Get example

```
> get we s  
R000021  
WiConnect Web Setup #
```

Set example

```
> set we s "This is my softAP!"  
R000012  
Set OK
```

setup.web.url

Abbreviation	se w u
Access	get/set

Description

A comma separated list of host names the webserver responds to in web setup mode.

Arguments

```
<curl list>
```

Default

wiconnect.com, www.wiconnect.com, setup.com, www.setup.com

Get example

```
> get we u
R000057
wiconnect.com, www.wiconnect.com, setup.com, www.setup.com
```

Set example

```
> set we u setup,start,lets.go,hello
R000008
Set OK
```

SoftAP Interface**softap.auto_start**

Abbreviation	so a
Access	get/set

Description

Auto start softap at boot up

Arguments

```
<0/off/false/1/on/true>
```

Default

0

Get example

```
> get so a  
R000003  
0
```

Set example

```
> set so a off  
R000008  
Set OK
```

softap.channel

Abbreviation	so c
Access	get/set

Description

The softap channel. If 0, use the default channel. If a wlan interface is already active, the channel variable is ignored since the wlan interface has channel set priority.

Arguments

<channel>

Default

0

Get example

```
> get so c  
R000003  
0
```

Set example

```
> set so c 6  
R000008  
Set OK
```

softap.client_list

Abbreviation	so l
--------------	-------------

Access	get
--------	-----

Description

Return list of connected softap clients

Arguments

-

Default

-

Get example

```
get so l
! Connected: 2
! # MAC           IP Address
# 1 F8:16:54:35:0F:8E 10.10.10.2
# 2 11:22:33:44:55:66 10.10.10.17
```

softap.dhcp_server

Abbreviation	so d
Access	get/set

Description

Enable DHCP server on the softap interface.

Arguments

<0/off/false/1/on/true>

Default

1

Get example

```
> get so d
R000003
1
```

Set example

```
> set so d true
R000008
Set OK
```

softap.dns_server

Abbreviation	so n
Access	get/set

Description

Enable DNS redirect server on the softap interface.

Arguments

<0/off/false/1/on/true>

Default

1

Get example

```
> get so n
R000003
1
```

Set example

```
> set so n 0
R000008
Set OK
```

softap.hide_ssid

Abbreviation	so h
Access	get/set

Description

When enabled, this variable disables broadcast of the softap's SSID, for security purposes.

Arguments

<0/1>

Default

0

Get example

```
> get so h  
0
```

Set example

```
> set so h 1  
Set OK
```

softap.idle_timeout

Abbreviation	so t
Access	get/set

Description

Maximum client idle time before the client is automatically disassociated.

Arguments

<# seconds>

Default

7

Get example

```
> get so t  
R000003  
7
```

Set example

```
> set so t 20  
R000008  
Set OK
```

softap.info

Abbreviation	so o
Access	get

Description

Soft AP connection status parameters: state, SSID, channel and clients.

Arguments

-

Default

<varies>

Get example

```
> get softap.info
state: up
SSID: WiConnect-#
channel: 0
clients: 1
```

softap.ip

Abbreviation	so i
Access	get/set

Description

The IP address used for the softap network interface. The gateway is set to the IP and the netmask is 255.255.255.0. Note! The softap.ip settings are also used for web setup.

Arguments

<IP address>

Default

10.10.10.1

Get example

```
> get so i  
R000013  
10.10.10.1
```

Set example

```
> set so i 192.168.10.1  
R000008  
Set OK
```

softap.passkey

Abbreviation	so p
Access	get/set

Description

The WPA2-AES passphrase for the softap interface, or none to disable security.

Arguments

<passphrase>

Default

""

Get example

```
> get so p  
R000000
```

Set example

```
> set so p "My softAP password"  
R000008  
Set OK
```

softap.rate.protocol

Abbreviation	so r p
--------------	---------------

Access	get/set
--------	---------

Description

softAP data rate protocol selection. Use 11b to restrict 802.11 PHY data rates to 802.11b, 11g for 802.11g data rates, and auto to automatically select the best rate from all 802.11b/g/n rates.

Arguments

<11b/11g/auto>

Default

auto

Get example

```
> get softap.rate.protocol
auto
```

Set example

```
> set softap.rate.protocol 11b
Set OK
```

softap.rate.transmit

Abbreviation	sor t
Access	get/set

Description

Force the softAP interface to transmit at a fixed data rate. The transmit rate is chosen from the subset specified by **softap.rate.protocol**.

Arguments

<rate>

where:

<rate> value	Transmit Rate
-1	auto rate selection
1, 2, 5.5, 11	protocol = 802.11b

6, 9, 12, 18, 24, 36, 48, 54	protocol = 802.11g
mcs0, mcs1, mcs2, ... , mcs7	protocol = 802.11n 1x1

Default

auto

Get example

```
> get so r x
5.5
```

Set example

```
> set softap.rate.transmit mcs3
Set OK
```

softap.ssid

Abbreviation	so s
Access	get/set

Description

The SSID of the softAP. If the SSID ends with #, the last 6 digits of the WLAN MAC address are appended to the SSID

Arguments

<ssid>

Default

WiConnect-#

Get example

```
> get
```

Set example

```
> set
```

softap.url

Abbreviation	so u
Access	get/set

Description

Comma separated list of domains to resolve. Maximum of 63 characters.

Arguments

<comma separated list>

Default

wiconnect,wiconnect.com,www.wiconnect.com

Get example

```
> get so u
R000043
wiconnect,wiconnect.com,www.wiconnect.com
```

Set example

```
> set so u mydevice,mydevice.com
R000008
Set OK
```

Static Network Settings

static.dns

Abbreviation	st d
Access	get/set

Description

DNS server address if static network settings are used for the wlan interface.

Arguments

<[IP address]/[host name]>

Default

8.8.8.8

Get example

```
> get st d  
R000009  
8.8.8.8
```

Set example

```
> set st d 8.8.4.4  
R000008  
Set OK
```

static.gateway

Abbreviation	st g
Access	get/set

Description

Gateway address if static network settings are used for the wlan interface.

Arguments

<IP address>

Default

0.0.0.0

Get example

```
> get st g  
R000009  
0.0.0.0
```

Set example

```
> set st g 192.168.0.1  
R000008  
Set OK
```

static.ip

Abbreviation	st i
Access	get/set

Description

Local IP address if static network settings are used for the wlan interface.

Arguments

<IP address>

Default

0.0.0.0

Get example

```
> get st i
R000009
0.0.0.0
```

Set example

```
> set st i 192.168.0.2
R000008
Set OK
```

static.netmask

Abbreviation	st n
Access	get/set

Description

Netmask if static network settings are used for the wlan interface.

Arguments

<network mask>

Default

0.0.0.0

Get example

```
> get st n  
R000009  
0.0.0.0
```

Set example

```
> set st n 255.255.255.0  
R000008  
Set OK
```

Stream Settings

stream.auto_close

Abbreviation	sta
Access	get/set

Description

This setting is intended for remote TCP/TLS clients that may asynchronously close a connection.
When FALSE, a stream handle is not reused until it is manually closed.
When TRUE, a stream handle is automatically reused when closed.

Arguments

<0/off/false/1/on/true>

Default

true

Get example

```
> get st a  
1  
>
```

Set example

```
> set st a false  
Set OK
```

System

system.adc.vref

Abbreviation	sy a v
Access	get/set

Description

Specify the ADC reference voltage, in millivolts, used by the [adc_take_sample](#) command.

Arguments

<mv>

Default

3300

Get example

```
> get system.adc.vref
3300
```

Set example

```
> set sy a v 3200
Set OK
```

system.activity.gpio

Abbreviation	sy a g
Access	get/set

Description

Map GPIO to activity type. When activity is detected, the corresponding GPIO is momentarily set to the level specified by [system.activity gpio_level](#).

Arguments

' '[`<GPIO#> <type>`](#)

where:

- <GPIO#> - the mapped GPIO. set to -1 to disable the <type> for all configured GPIOs.
- <type> - a comma separated list of activity types.

The following types are available:

- softap_rx
- softap_tx
- wlan_rx
- wlan_tx
- uartX_rx - where X is the uart number
- uartX_tx - where X is the uart number

Default

none - no GPIOs are mapped to activity types

Get example

```
> get system.activity gpio 1
none
```

Set example

```
> set system.activity gpio 7 softap_rx,softap_tx,wlan_tx
Set OK
> get system.activity gpio 7
softap_rx,softap_tx,wlan_tx
>
Ready
> set system.activity gpio -1 softap_rx
Set OK
> get system.activity gpio 7
softap_tx,wlan_tx
```

system.activity gpio_level

Abbreviation	sy a l
Access	get/set

Description

Specify the GPIO level corresponding to activity. See [system.activity gpio](#).

Arguments

<1/0>

Default

1

Get example

```
> get system.activity gpio_level 1
1
```

Set example

```
> set system.activity gpio_level 0
```

system.bflash.cs_gpio

Abbreviation	sy f g
Access	get/set

Description

Specify the bulk flash's chip select gpio. Set to -1 to disable the bulk flash feature.

See [File System, Internal, Extended and Bulk Flash](#).

Arguments

<GPIO #>

Default

-1

Get example

```
> get system.bflash.cs_gpio
-1
```

Set example

```
> set sy f g 4
Set OK
```

system.boot_app

Abbreviation	sy b
Access	get/set

Description

Default boot application. The <filename> must be a valid boot application available on the filesystem.

Arguments

<filename>

Default

wiconnect

Get example

```
> get sy b
R000016
wiconnect.exe
```

Set example

```
> set sy b wiconnect.exe
R000008
Set OK
```

system.cmd.buffered

Abbreviation	sy c b
Access	get/set

Description

Flag to enable/disable buffering of certain command responses. This is intended for use when reading command output by machine.

If enabled, a command's response is buffered and a stream handle along with the response size in bytes is returned.

The host must use the 'read' command to retrieve the command's response data.

Note that if headers are enabled, the header is returned along with the stream handle and response size.

The following commands are buffered if the system.cmd.buffered flag is set:

- scan
- ls
- help
- group getters
- stream_list
- get softap.client_list
- get setup.web.client_list
- all getters with all option

Arguments

<0/off/false/1/on/true>

Default

0

Get example

```
> get sy c b  
0
```

Set example

```
> set sy c b on  
Set OK
```

system.cmd.echo

Abbreviation	sy c e
Access	get/set

Description

Enable/disable character echo. Only applies to command mode.

Arguments

<0/off/false/1/on/true>

Default

1

Get example

```
> get sy c e
R000003
1
```

Set example

```
> set sy c e on
R000008
Set OK
```

system.cmd.header_enabled

Abbreviation	sy c h
Access	get/set

Description

Enable/disable a response header for commands. Only applies to command mode. Response headers make it easy to parse responses with a host MCU.

Arguments

<0/off/false/1/on/true>

Default

0

Get example

```
> get sy c h
R000003
1
```

Set example

```
> set sy c h on
R000008
Set OK
```

system.cmd.mode

Abbreviation	sy c m
--------------	---------------

Access	set
--------	-----

Description

Puts the WiConnect command interface into human or machine mode, further details are provided in [Serial Interface, Command Mode](#). **Note!**: This variable is NOT readable.

Arguments

<human | machine>

Default

human

Set example

```
> set sy c m human
R000008
Set OK
```

system.cmd.prompt_enabled

Abbreviation	sy c p
Access	get/set

Description

Enable/disable terminal command prompt. Only applies to command mode. A prompt makes it easy for humans to interact with WiConnect.

Arguments

<0/off/false/1/on/true>

Default

1

Get example

```
> get sy c p
R000003
1
```

Set example

```
> set sy c p on
R000008
Set OK
```

system.cmd.timestamp

Abbreviation	sy c t
Access	get/set

Description

When set to value other than none, adds a millisecond resolution timestamp to each WiConnect response. The timestamp appears before other output, and is followed by a colon and a space, e.g.

rtc output:

```
> get network.ip
1409819199621: 10.5.6.71
```

utc output:

```
> get network.ip
2014-09-04 | 08:27:45: 10.5.6.71
```

Arguments

<none/rtc/utc>

where:

- none - no timestamp appended
- rtc - numeric value of RTC with millisecond resolution
- utc - human-readable RTC value with millisecond resolution</code>

Default

none

Get example

```
> get sy c t
2014-09-04 | 08:32:11: utc
```

Set example

```
> set sy c t rtc
```

system.gotosleep.timeout

Abbreviation	sy s t
Access	get/set

Description

The module automatically goes to sleep after a timeout period of <seconds>. The timeout countdown restarts when a wake event occurs and is held off if data is available on open network connections, the minimum timeout is 10 seconds. 0 disables gotosleep. A save and reboot is required before the sleep timeout is enabled.

Arguments

<seconds>

Default

0

Get example

```
> get sy s t
R000003
0
```

Set example

```
> set sy s t 12
R000008
Set OK
```

system.indicator.gpio

Abbreviation	sy i g
Access	get/set

Description

The GPIO used for a particular system indicator. System indicators drive LEDs to indicate the state of the wlan, network and softap interfaces. -1 to disable.

Arguments

```
<wlan/network/softap> <GPIO#>
```

Default

```
<varies>
```

Get example

```
> get sy i g network
R000004
11
```

Set example

```
> set gpio.alias 11 netstat_gpio    <- Alias GPIO 11 for simplicity
R000008
Set OK
> set sy i g network netstat_gpio
R000008
Set OK
```

system.indicator.state

Abbreviation	syis
Access	get/set

Description

System indicator option for each state as described in the following tables.

WLAN Indicator

Wi-Fi Client status, by default GREEN LED on eval boards

State #	State Description	Default Behavior
state 1	Wi-Fi off	static_off
state 2	Error	fast_blink
state 3	Connecting to AP	medium_blink
state 4	Joined to AP	slow_blink

Network Indicator

Network status, by default YELLOW LED on eval boards

State #	State Description	Default Behavior
state 1	No IP address	fast_blink
state 2	DHCP in-progress	medium_blink
state 3	DHCP successful	slow_blink

SoftAP Indicator

SoftAP status, by default RED LED on eval boards

State #	State Description	Default Behavior
state 1	SoftAP off	static_off
state 2	SoftAP error	fast_blink
state 3	SoftAP active	medium_blink
state 4	Client connected	slow_blink

For blink rates, see [Configuration and Setup, System Indicator State Options](#)

Arguments

<wlan/network/softap> <csv list>

Default

<see above>

Get example

```
> get sy i s wlan
R000047
static_off|fast_blink|medium_blink|slow_blink
```

Set example

```
> set sy i s network static_off|medium_blink|static_on
R000008
Set OK
```

system.memory_usage

Abbreviation	sy o
Access	get

Description

Returns the percentage of available heap memory currently in use. See [Memory](#).

Arguments

Default

Get example

```
> get sy o  
58
```

system.msg

Abbreviation	sy m
Access	get/set

Description

Customize system messages for various events: initialized, sleep, wlan_joining, wlan_joined, wlan_leave, wlan_failed, softap_joined, softap_leave, network_opening, network_opened, network_closed, network_failed. Tags may be included to provide dynamic information as shown below.

Tag	Description	Tag is available for ...
@t	Timestamp	All messages
@s	SSID	All wlan messages
@c	Stream handle	network_opened, network_closed
@h	Connection host/port	network_opening, network_failed
@m	Client MAC Address	softap_joined, softap_leave

Arguments

<message name> <message value>

Default

<see get example below>

Get example

```
> get system.msg all
R000397
initialized - [@tReady]
network_closed - [@tDisconnected: @c]
network_failed - [@tConnect failed]
network_opened - [@tConnected: @c]
network_opening - [@tConnecting: @h]
sleep - [@tSleep]
wlan_failed - [@tJoin failed]
wlan_joined - [@tAssociated]
wlan_joining - [@tAssociating to @s]
wlan_leave - [@tDisassociated]
softap_joined - [@t@m associated]
softap_leave - [@t@m disassociated]
```

Set example

```
> set sy m wlan_leave "[@t: Bye for now]"
R000008
Set OK
```

system.oob.event_mask

Abbreviation	sy o e
Access	get/set

Description

The mask of events which assert the OOB GPIO. See [system.oob gpio](#).

For event/mask mapping, see the [system.oob gpio event mask table](#).

Arguments

<hex mask>

Mask is specified in the form 0xHH where H is a hexadecimal digit 0-F.

Default

0x0

Get example

```
> get sy o e
0x1F
```

Set example

```
> set sy o e 0x11
Set OK
```

system.oob gpio

Abbreviation	sy o g
Access	get/set

Description

The `system.oob` variables allow creation of an out-of-band interrupt (OOB) GPIO. This GPIO is asserted whenever one or more of the configured events occurs. It is de-asserted when reading the status register.

Note: The OOB GPIO feature can be used by itself. The corresponding special function GPIO does not need to be enabled. i.e. The `network.status` OOB event can be triggered without setting the `network.status_gpio` variable.

Specify the GPIO with the `system.oob gpio` variable.

Specify the active level of the OOB GPIO with `system.oob gpio_level`.

Specify the events that cause the GPIO to be asserted with `system.oob.event_mask`.

Specify events that trigger the GPIO only on the rising edge (i.e. when the event is asserted) with `system.oob.rising_edge_mask`.

Read the events that triggered an OOB interrupt from the `system.oob.status` variable.

Only one GPIO can be specified as the OOB GPIO. Determine the OOB GPIO by reading the `gpio.usage` variable or the `system.oob gpio` variable.

Events are mapped to the `event mask`, `rising edge mask` and `status mask` as shown in the following table:

Event Name	Mask Value
ioconn.status_gpio	0x01
network.status gpio	0x02
tcp.server.connected_gpio	0x04
tcp.server.data_gpio	0x08
udp.server.data_gpio	0x10
tcp.client.data_gpio	0x20

udp.client.data_gpio	0x40
----------------------	------

Arguments

<GPIO #>

The number of the OOB GPIO.

Default

No GPIO assigned to OOB

Get example

```
> get sy o g
22
```

Set example

```
> set system.oob gpio 22
Set OK
```

system.oob gpio_level

Abbreviation	sy o l
Access	get/set

DescriptionThe active level of the OOB GPIO. See [system.oob gpio](#).**Arguments**

<0/1>

Default

1

Get example

```
> get sy o l
1
```

Set example

```
> set sy o 1 0
Set OK
```

system.oob.rising_edge_mask

Abbreviation	sy or
Access	get/set

Description

The mask of events which should trigger the OOB GPIO only on the rising edge (i.e. when an event is asserted). See [system.oob gpio](#).

If the mask is cleared, then the OOB GPIO is asserted on both rising and falling edges.

For instance, when the module connects, the network status event event is considered asserted. When the module disconnects, the event is considered de-asserted.

For event/mask mapping, see the [system.oob gpio event mask table](#).

Arguments

<hex mask>

Mask is specified in the form 0xHH where H is a hexadecimal digit 0-F.

Default

0x0

Get example

```
> get sy o r
0x1F
```

Set example

```
> set sy o r 0x11
Set OK
```

system.oob.status

Abbreviation	sy os
--------------	--------------

Access	get
--------	-----

Description

The mask of triggered events. When an event is triggered, the OOB GPIO is asserted and the status mask is updated with the triggering event. Reading this variable returns a mask of events that caused the OOB gpio to be asserted.

Reading this variable de-asserts the OOB GPIO and clears the status mask.

For event/mask mapping, see the [system.oob gpio event mask table](#).

Arguments

Default

0x0

Get example

```
get sy o e  
0x0
```

system.powersave.mode

Abbreviation	sy p m
Access	get/set

Description

Set the system powersave mode where 0 = no powersave, 1 = core powered down on idle with <5 cycles power up latency. Additional powersave modes will be added in an upcoming release. Until then, use [system.gotosleep.timeout](#) to put the module into deep sleep.

Note! Wi-Fi powersave is independently configured using [wlan.powersave.mode](#).

Arguments

<mode>

Default

1

Get example

```
> get sy p m  
R000003  
1
```

Set example

```
> set sy p m 0  
R000008  
Set OK
```

system.print_level

Abbreviation	sy p
Access	get/set

Description

Print levels: 0=None, 1=Synchronous system msgs, 2=Synchronous logging msgs, 3=Asynchronous system msgs, 4/all=Asynchronous logging msgs.

Arguments

<[0-4]/[all]>

Default

all

Get example

```
> get sy p  
R000005  
all
```

Set example

```
> set sy p 0  
R000008  
Set OK
```

system.safemode

Abbreviation	sy s
--------------	-------------

Access	get
--------	-----

Description

Returns a flag to indicate whether the system is in safe mode. The module switches to safe mode after eight exceptions occur with the boot application.

Arguments

-

Default

0

Get example

```
> get sy s
R000003
0
```

system.uuid

Abbreviation	sy u
Access	get

Description

Returns the hardware UUID of the module.

Arguments**Default****Get example**

```
> get sy u
R000042
05B3203FA03133434E39423330363031B2303533
```

system.version

Abbreviation	sy v
Access	get

Description

Returns the WiConnect firmware bundle version.

Arguments

-

Default

<varies>

Get example

```
> get sy v
R000069
wiconnect.exe v1.0.0, Built:Apr 03 2014 for AMW004.3-E03.1
```

system.wakeup.events

Abbreviation	sy we
Access	get/set

Description

Control which events wake the module from sleep. Multiple options are separated by the " character. \0 disables wake events. Some UART interfaces do not have wake capability. See [Peripherals, GPIO Functions and Pins](#) for tables showing GPIOs with wake capability.

Note: On ACKme evaluation boards the Button 2 GPIO (GPIO11 Wallaby, GPIO22 Numbat) is by default set as one of the system.wakeup.events and also associated with the [setup gpio control gpio](#) function. When using Button 2 as a wakeup event, either disable the [setup gpio control gpio](#) function, or ensure the GPIO is asserted for less than 3 seconds, otherwise the module will enter [web setup mode](#) after waking.|

Arguments

<[<uartX>|<gpioX>] / [\0]>

Default

uart0

Get example

```
> get sy w e
R000007
uart0
```

Set example

```
> set sy w e gpio5|uart0
R000008
Set OK
```

system.wakeup.timeout

Abbreviation	sy w t
Access	get/set

Description

The module automatically wakes from sleep after timeout <seconds>. Wake events are configured with system.wakeup.events. 0 disables auto-wakeup.

Arguments

<seconds>

Default

0

Get example

```
> get sy w t
R000003
0
```

Set example

```
> set sy w t 60
R000008
Set OK
```

TCP Client

tcp.client.auto_interface

Abbreviation	tc c i
Access	get/set

Description

Interface to auto-start TCP client connection.

Arguments

default

Default

<default/wlan/softap>

Get example

```
> get tcp.client.auto_interface
default
```

Set example

```
> set tc c i wlan
Set OK
```

tcp.client.auto_retries

Abbreviation	tc c e
Access	get/set

Description

Determines number of attempts to reconnect after remote server disconnects client.

- 0 - disables feature (i.e. don't reconnect after server disconnects)
- 255 - infinite retries (i.e. continuously try to reconnect after server disconnects)
- 1-254 - retry specified number of times then stop

Note: This counter is reset when the interface is brought down.

To control how the client connects to the server, use **tcp.client.retries**, **tcp.client.connect_timeout** and **tcp.client.retry_period**.

Arguments

<0-255>

Default

0

Get example

```
> get tcp.client.auto_retries  
0
```

Set example

```
set tc c e 20  
Set OK
```

tcp.client.auto_start

Abbreviation	tc c a
Access	get/set

Description

Enable/disable TCP client auto-start feature. If enabled, TCP client automatically connects to remote server when interface is brought up.

Arguments

<0/1>

Default

0

Get example

```
> get tcp.client.auto_start  
0
```

Set example

```
> set tc c a 1  
Set OK
```

tcp.client.connect_timeout

Abbreviation	tc c t
Access	get/set

Description

TCP client connect timeout in milliseconds.

Arguments

<milliseconds>

Default

5000

Get example

```
> get tc c t
R000006
5000
```

Set example

```
> set tc c t 6500
R000008
Set OK
```

tcp.client.local_port

Abbreviation	tc c p
Access	get/set

Description

TCP client local port. 0 = automatic.

Arguments

<port>

Default

0

Get example

```
> get tc c p  
R000003  
0
```

Set example

```
> set tc c p 30009  
R000008  
Set OK
```

tcp.client.remote_host

Abbreviation	tc c h
Access	get/set

Description

Domain or IP address of remote tcp server to auto connect.

Arguments

<domain/ip address>

Default

null

Get example

```
> get tcp.client.remote_host  
mydomain.com
```

Set example

```
> set tc c h mydomain.com  
Set OK
```

tcp.client.remote_port

Abbreviation	tc c o
Access	get/set

Description

Port of remote tcp server to auto connect.

Arguments

<port>

Default

3000

Get example

```
> get tcp.client.remote_port
```

Set example

```
> set tc c o 5000
Set OK
```

tcp.client.retries

Abbreviation	tc c r
Access	get/set

Description

TCP client connection retry attempts

Arguments

<max #retries>

Default

3

Get example

```
> get tc c r
R000003
```

3

Set example

```
> set tc c r 5
R000008
Set OK
```

tcp.client.retry_period

Abbreviation	tc c w
Access	get/set

Description

TCP client wait time between connection retry attempts

Arguments

<milliseconds>

Default

1000

Get example

```
> get tc c w
R000006
1000
```

Set example

```
> set tc c w 2000
R000008
Set OK
```

TCP Keepalive**tcp.keepalive.enabled**

Abbreviation	tc k e
--------------	---------------

Access	get/set
--------	---------

Description

Enable/disable TCP keepalive feature of network stack.

Note: These settings apply to ALL TCP connections (TCP client, TLS, TCP server)

Keepalive is a system for determining whether a connected peer is still connected. If the peer reboots or otherwise drops out without notice, keepalive detects the problem. It works by sending an empty message at a specified interval and receiving an ACK in reply from the peer. If the ACK is not received with an initial expiration period, the keepalive exchange is periodically retried at a specified retry period until the retry count is exceeded.

To determine the initial expiration period, see [tcp.keepalive.initial_timeout](#).

To determine the retry period, see [tcp.keepalive.retry_timeout](#).

To determine the maximum retry count, see [tcp.keepalive.retry_count](#).

Arguments

<true/false>

Default

false

Get example

```
> get tc k e
0
```

Set example

```
> set tc k e 1
Set OK
```

tcp.keepalive.initial_timeout

Abbreviation	tc k i
Access	get/set

Description

Number of seconds for initial keepalive expiration.

See [tcp.keepalive.enabled](#).

Arguments

<seconds>

Range: 1 - 65535

Default

60

Get example

```
> get tc k i  
60
```

Set example

```
> set tc k i 20  
Set OK
```

tcp.keepalive.retry_count

Abbreviation	tc k c
Access	get/set

Description

Maximum number of times to retry keepalive. Once all retries are expired, close connection.

See [tcp.keepalive.enabled](#).

Arguments

<seconds>

Range: 1 - 255

Default

5

Get example

```
> get tc k c
```

5

Set example

```
> set tc k c 10
Set OK
```

tcp.keepalive.retry_timeout

Abbreviation	tc k r
Access	get/set

Description

After initial expiration, period of retries in seconds.

See [tcp.keepalive.enabled](#).

Arguments

<seconds>

Range: 1 - 255

Default

5

Get example

```
> get tc k r
5
```

Set example

```
> set tc k r 10
Set OK
```

TCP Server**tcp.server.auto_interface**

Abbreviation	tc s i
--------------	---------------

Access	get/set
--------	---------

Description

Wireless interface used for the TCP server if auto-start is enabled.
If set to default, uses the interface specified in [network.default_interface](#)

Arguments

<wlan/softap/default>

Default

default

Get example

```
> get tc s i
R000009
default
```

Set example

```
> set tc s i softap
R000008
Set OK
```

tcp.server.auto_start

Abbreviation	tc s a
Access	get/set

Description

Enable/disable TCP server auto-start on bootup.

Arguments

<0/off/false/1/on/true>

Default

0

Get example

```
> get tc s a
R000003
0
```

Set example

```
> set tc s a true
R000008
Set OK
```

tcp.server.connected_gpio

Abbreviation	tc s c
Access	get/set

Description

The GPIO used to indicate whether a client is connected to the TCP server.

Arguments

<[GPIO#]/[-1]>

Default

-1

Get example

```
> get tc s c
R000004
-1
```

Set example

```
> set tc s c 12
R000008
Set OK
```

tcp.server.data_gpio

Abbreviation	tc s d
Access	get/set

Description

The GPIO used to indicate whether a connected client has data available to read.

Note: Disable by setting to -1.

Arguments

<GPIO#>

Default

-1

Get example

```
> get tc s d
R000004
-1
```

Set example

```
> set tc s d 13
R000008
Set OK
```

tcp.server.idle_timeout

Abbreviation	tc s t
Access	get/set

Description

Maximum client idle time in seconds before the client is automatically disconnected.

Arguments

<seconds>

Default

60

Get example

```
> get tc s t
R000004
```

60

Set example

```
> set tc s t 30
R000008
Set OK
```

tcp.server.max_clients

Abbreviation	tc s m
Access	get/set

Description

Specify maximum number of clients that may be simultaneously connected to the tcp server.

Arguments

<max #> - maximum number of TCP server clients, valid range: 0-8.

Default

0

Get example

```
> > get tc s m
0
```

Set example

```
> set tc s m 5
Set OK
```

tcp.server.port

Abbreviation	tc s p
Access	get/set

Description

TCP server port to listen on

Arguments

<port>

Default

3000

Get example

```
> get tc s p
R000006
3000
```

Set example

```
> set tc s p 3001
R000008
Set OK
```

tcp.server.tls_cert

Abbreviation	tc s n
Access	get/set

Description

The server TLS CA certificate filename.

Arguments

<filename>

Default

null

Get example

```
> get tc s n
R000012
mycert.pem
```

Set example

```
> set tc s n mycert.pem
```

```
R000008
Set OK
```

tcp.server.tls_enabled

Abbreviation	tc s u
Access	get/set

Description

Enable server TLS certificate (used with **auto_start** option)

Arguments

<true/false>

Default

false

Get example

```
> get tc s u
R000003
0
```

Set example

```
> set tc s u false
R000008
Set OK
```

tcp.server.tls_key

Abbreviation	tc s k
Access	get/set

Description

The server TLS certificate key filename.

Arguments

```
<filename>
```

Default

```
null
```

Get example

```
> get tc s k
R000012
mycert.key
```

Set example

```
> set tc s k mycert.key
R000008
Set OK
```

tcp.server.tls_verify_peer

Abbreviation	tc s v
Access	get/set

Description

When this variable is set, a connecting client must provide a valid TLS certificate that the server validates. The client is disconnected if the provided certificate is invalid. **Note:** The **tcp.server.tls_cert** variable must be set if using this feature.

Arguments

```
<0/1>
```

Default

```
0
```

Get example

```
> get tc s v
0
```

Set example

```
> set tc s v 1
Set OK
```

Time

time.last_set

Abbreviation	ti l
Access	get

Description

Return the number of seconds (or milliseconds) since the time was set (either manually or via NTP), or -1 if the time has never been set.

Get arguments

[ms]

Default

-

Get example

```
> get ti l
R000005
239
```

time.rtc

Abbreviation	ti r
Access	get/set

Description

The local time. Read returns epoch seconds by default, or (optionally) epoch milliseconds or UTC. Write takes an <epoch seconds> argument.

Get arguments

[ms/utc]

Set arguments

<epoch seconds>

Default

-

Get example

```
> get ti r utc
R000029
2014-03-23T08:13:07.010248Z
```

Set example

```
> set ti r 1395562166
R000008
Set OK
```

time.uptime

Abbreviation	ti u
Access	get

Description

Returns the number of seconds (or milliseconds) since reset, guaranteed not to roll over in your lifetime!

Arguments

[ms]

Default

<varies>

Get example

```
> get ti u ms
R000006
3578576
```

time.zone

Abbreviation	ti z
Access	get/set

Description

Timezone offset from GMT in hours.

Arguments

[+/-]H[:M]

where H is an integer between 0 and 12, and M (if present) = 30
Offset ranges from -12:30 to 12:30.

Default

0

Get example

```
> get ti z
R000007
-4:30
```

Set example

```
> set ti z -4:30
R000008
Set OK
```

UART**uart.baud**

Abbreviation	ua b
Access	get/set

Description

Sets the UART baud rate. Valid baud rates are <baud rate> must be one of the following standard rates:

110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600, 115200, 128000, 153600, 230400, 256000, 460800, 921600.

If the optional <raw> parameter is given, then the rate may be any value.

Get arguments

```
<uart#>
```

Set arguments

```
<uart#> <baud rate> [raw]
```

Default

115200

Get example

```
> get ua b 0
R000008
115200
```

Set example

```
> set ua b 0 460800
R000008
Set OK
```

uart.data

Abbreviation	ua d
Access	get/set

Description

The number of UART data bits to use.

Arguments

```
<uart#> <# data bits>
```

Default

8

Get example

```
> get ua d 0
R000003
8
```

Set example

```
> set ua d 0 8
R000008
Set OK
```

uart.flow

Abbreviation	ua f
Access	get/set

Description

Turn on/off UART hardware flow control.

Arguments

<uart#> <uart#> <off/on>

Default

Get example

```
> get ua f 1
R000005
off
```

Set example

```
> set ua f 1 off
R000008
Set OK
```

uart.parity

Abbreviation	ua p
Access	get/set

Description

Configure UART parity selection.

Arguments

```
<uart#> <none/even/odd>
```

Default

none

Get example

```
> get ua p 1  
R000006  
none
```

Set example

```
> set ua p 1 odd  
R000008  
Set OK
```

uart.stop

Abbreviation	ua s
Access	get/set

Description

The number of UART stop bits to use.

Arguments

```
<uart#> <# stop bits>
```

Default

1

Get example

```
> get ua s 0  
R000003  
1
```

Set example

```
> set ua s 0 1  
R000008  
Set OK
```

UDP Client

udp.client.auto_interface

Abbreviation	ud c i
Access	get/set

Description

Interface to auto-start UDP client connection.

Arguments

default

Default

<default/wlan/softap>

Get example

```
> get udp.client.auto_interface
default
```

Set example

```
> set ud c i wlan
Set OK
```

udp.client.auto_start

Abbreviation	ud c a
Access	get/set

Description

Enable/disable UDP client auto-start feature. If enabled, UDP client automatically connects to remote server when interface is brought up.

Arguments

<0/1>

Default

0

Get example

```
> get udp.client.auto_start  
0
```

Set example

```
> set ud c a 1  
Set OK
```

udp.client.remote_host

Abbreviation	ud c h
Access	get/set

Description

Domain or IP address of remote udp server to auto connect.

Arguments

<domain/ip address>

Default

null

Get example

```
> get udp.client.remote_host  
mydomain.com
```

Set example

```
> set ud c h mydomain.com  
Set OK
```

udp.client.remote_port

Abbreviation	ud c p
Access	get/set

Description

Port of remote UDP server to auto connect.

Arguments

<port>

Default

3000

Get example

```
> get udp.client.remote_port
```

Set example

```
> set ud c p 5000
Set OK
```

UDP Server**udp.server.auto_interface**

Abbreviation	ud s i
Access	get/set

Description

If 'auto-start' is enabled, this is the interface used when the server is set to auto-start.
If set to default, uses the interface specified in [network.default_interface](#)

Arguments

<wlan/softap/default>

Default

default

Get example

```
> get ud s i  
R000009  
default
```

Set example

```
> set ud s i softap  
R000008  
Set OK
```

udp.server.auto_start

Abbreviation	ud s a
Access	get/set

Description

Start server on power-up

Arguments

<true|on|1/false|off|0>

Default

0

Get example

```
> get ud s a  
R000003  
0
```

Set example

```
> set ud s a true  
R000008  
Set OK
```

udp.server.data_gpio

Abbreviation	ud s d
Access	get/set

Description

Specify a GPIO to be asserted HIGH when client data is available to be read.

Note: Disable by setting to -1.

Arguments

<GPIO#>

Default

-1

Get example

```
> get ud s d
R000004
-1
```

Set example

```
> set ud s d 3
R000008
Set OK
```

udp.server.lock_client

Abbreviation	ud s l
Access	get/set

Description

If this flag is set, the UDP server is locked to the first client from which it receives a packet. It can send data to, or receive data from, only that client.

If this flag is not set, the udp server receives data from all clients and sends to the last client from which it received a packet.

This flag is useful in stream mode when streaming to only one udp client whose IP is unknown.

Note: The variables `udp.server.remote_host` and `udp.server.remote_port` may be used to override this flag. If these variables are set then the udp server replies only to the specified host:port.

Arguments

<0/1>

Default

0

Get example

```
> get ud s 1  
0
```

Set example

```
> set ud s 1 1  
Set OK
```

udp.server.port

Abbreviation	ud s p
Access	get/set

Description

Default port for UDP server to bind.

Arguments

<port>

Default

3000

Get example

```
> get ud s p  
R000006  
3000
```

Set example

```
> set ud s p 5000  
R000008  
Set OK
```

udp.server.remote_host

Abbreviation	ud s h
Access	get/set

Description

Domain or IP address of remote udp client to auto connect. This is used in stream mode when the udp server is configured to auto-start. The stream sends data only to the configured remote host:port. See also [udp.server.remote_port](#).

To disable, set null, i.e.

```
set udp.server.remote_host ""
```

Note : If the remote_host is not configured, then the UDP server writes to the last client from which it read data (unless the destination address is specified using the udp_server write command).

Arguments

<domain/ip address>

Default

null

Get example

```
> get udp.server.remote_host
mydomain.com
```

Set example

```
> set ud c s mydomain.com
Set OK
```

udp.server.remote_port

Abbreviation	ud s r
Access	get/set

Description

Port of remote UDP client to auto connect. This is used in stream mode when the udp server is configured to auto-start. The stream

sends data only to the configured remote host:port. See also [udp.server.remote_host](#).

Note: If the port is 0 then use the UDP server's listening port. See [udp.server.port](#).

Arguments

<port>

Default

3000

Get example

```
> get udp.server.remote_port
```

Set example

```
> set ud s r 5000
Set OK
```

WLAN Interface

wlan.antenna

Abbreviation	wl a
Access	get/set

Description

The default antenna used by the Wi-Fi radio. 1 or int:internal, 2 or ext:external, or auto. NOTE: An external antenna must be connected if external or auto mode are selected.

Arguments

<1|int/2|ext/auto>

Default

1

Get example

```
> get wl a
R000003
```

1

Set example

```
> set wl a auto
R000008
Set OK
```

wlan.auto_join.enabled

Abbreviation	wl o e
Access	get/set

Description

Enable/disable network auto-join using the wlan interface on bootup.

Arguments

<0/off/false/1/on/true>

Default

0

Get example

```
> get wl o e
R000003
0
```

Set example

```
> set wl o e true
R000008
Set OK
```

wlan.auto_join.retries

Abbreviation	wl o r
Access	get/set

Description

Maximum number of attempts to auto-join a network. 0 = retry indefinitely.

Arguments

<0-255>

Default

3

Get example

```
> get wl o r
R000003
3
```

Set example

```
> set wl o r 5
R000008
Set OK
```

wlan.auto_join.retry_delay

Abbreviation	wl o d
Access	get/set

Description

The period between auto-join attempts in seconds.

Arguments

<0-255>

Default

10

Get example

```
> get wl o d
R000004
10
```

Set example

```
> set wl o d 5
R000008
Set OK
```

wlan.hide_passkey

Abbreviation	wl h
Access	get/set

Description

Hide the variable wlan.passkey. Once hidden, a factory reset must be issued to reset this variable and reveal the (reset value) of the passkey. **NOTE:** Don't forget to save before reboot!

Arguments

<0/off/false/1/on/true>

Default

0

Get example

```
> get wl h
R000003
0
```

Set example

```
> set wl h true
R000008
Set OK
> save
R000009
Success
> get wlan.passkey
R000008
*****
```

wlan.info

Abbreviation	wl i
--------------	-------------

Access	get
--------	-----

Description

WLAN connection status parameters: state, SSID, BSSID, channel and datarate

Arguments

-

Default

<varies>

Get example

```
> get wlan.info
state: up
SSID: ackme
BSSID: 30:85:A9:E7:9C:B0
channel: 6
datarate: 450.0
```

wlan.join.result

Abbreviation	wlj s
Access	get

Description

Result of most recent join attempt. Returns a code:

Code	Description
0	Haven't tried to join
1	Success - Joined network
2	Joining network
3	SSID not set
4	No password set and AP requires security
5	Specified security not supported by AP
6	Network not found
7	Failed to join network. All attempts to join have failed

8

Join aborted. network_down called while trying to join

Arguments**Default**

<varies>

Get example

```
> get wl j s  
1
```

wlan.join.retries

Abbreviation	wljr
Access	get/set

Description

Specifies the maximum number of attempts to join a WLAN network after issuing the [network_up](#) command (or any other command requiring network access when the network is down). If the [network_down](#) command is issued while attempting to join a network, the subsequent attempts are canceled.

Note that [wlan.join.retries](#) is separate from the [wlan.auto_join.retries](#) variable. If [wlan.auto_join.retries](#) > 0, then the total number of retries is calculated by the multiplication of [wlan.join.retries](#) with [wlan.auto_join.retries](#).

Arguments

<retries>

where '<retries>' is an integer ranging 1-255.

Default

3

Get example

```
> get wl jr  
R0000003  
5
```

Set example

```
> set wl j r 5
R000008
Set OK
```

wlan.join.timeout

Abbreviation	wljt
Access	get/set

Description

The maximum amount of time to wait while the module attempts to join a network.

Arguments

<timeout in ms>

Default

7000 ms

Get example

```
> get wlan.join.timeout
R000006
7000
```

Set example

```
> set wl j t 5000
R000008
Set OK
```

wlan.mac

Abbreviation	wl m
Access	get

Description

Returns the wlan MAC address.

Arguments

-

Default

varies

Get example

```
> get wl m
R000019
4C:55:CC:01:23:45
```

wlan.passkey

Abbreviation	wl p
Access	get/set

Description

The WPA passphrase, WPA PSK or WEP key used when connecting to an AP.

The WEP key is replicated across each of the four possible keys.

Note! **wlan.security** must be manually set to WEP to use WEP security. ACKme **STRONGLY** discourages the use of WEP - it is **NOT** secure!

Arguments

<[<passphrase>/<WEP key>] / [\0]>

- WPA passphrase - length: 8 - 63 characters
- WPA PSK - length: 64 characters
- WEP key - A 40-bit WEP key is specified as a 5-byte, for example: 1122334455
A 104-bit WEP key is specified as 13-byte hex string, for example: 112233445566778899aabccdd
- \0 sets a NULL passphrase.

Default

""

Get example

```
> get wl p
R000000
```

Set example

```
> set wl p "Use the force!"  
R000008  
Set OK
```

wlan.powersave.listen_interval

Abbreviation	wl p l
Access	get/set

Description

The Wi-Fi sub-system remains in low power mode for a period specified in units of DTIM intervals (typically multiples of 100ms).

Arguments

<sleep time>

- <sleep time> is in the range 1-10. If <sleep time> = 0, the WLAN subsystem wakes to listen to every DTIM.

Default

0

Get example

```
> get wl p l  
R000003  
0
```

Set example

```
> set wl p l 90  
R000008  
Set OK
```

wlan.powersave.mode

Abbreviation	wl p m
Access	get/set

Description

Wi-Fi powersave mode. For lowest poweruse mode = 1.

Arguments

<mode>

<mode> value	Description
0	no powersave
1	powersave with PS-Poll. Lowest power mode
2	powersave optimised to maintain throughput

Default

0

Get example

```
> get wl p m
R000003
0
```

Set example

```
> set wl p m 2
R000008
Set OK
```

wlan.powersave.sleep_delay

Abbreviation	wl p s
Access	get/set

Description

Return to sleep delay in milliseconds when **wlan.powersave.mode** = 2 is used. Longer delays increase power consumption but may improve throughput performance. <delay> is in the range 10-100ms in increments of 10ms.

Arguments

<delay>

Default

10

Get example

```
> get wl p s
R000004
10
```

Set example

```
> set wl p s 40
R000008
Set OK
```

wlan.rate.protocol

Abbreviation	wl r p
Access	get/set

Description

WLAN client data rate protocol selection.

Arguments

<11b/11g/auto>

Value	Description
11b	Restrict 802.11 PHY data rates to 802.11b
11g	Restrict 802.11 PHY data rates to 802.11g
auto	Automatically select the best rate from all 802.11b/g/n rates

Default

auto

Get example

```
> get wl r p
auto
```

Set example

```
> set wlan.rate.protocol 11b
Set OK
```

wlan.rate.transmit

Abbreviation	wl rx
Access	get/set

Description

Force the WLAN interface to transmit at a fixed data rate. The transmit rate is taken from valid rates specified by [wlan.rate.protocol](#).

Arguments

<rate>

where:

<rate> value	Description
auto	auto rate selection
1, 2, 5.5, 11	protocol = 802.11b
6, 9, 12, 18, 24, 36, 48, 54	protocol = 802.11g
mcs0, mcs1, mcs2, ... , mcs7	protocol = 802.11n 1x1

Default

auto

Get example

```
> get wl rx
5.5
```

Set example

```
> set wlan.rate.transmit mcs3
Set OK
```

wlan.roam.threshold

Abbreviation	wl rt
Access	get/set

Description

The wlan interface searches for a new AP to connect to once the signal from the AP it is currently associated with drops below the roam trigger level. The trigger level is specified in dBm in the range -1 to -99.

Arguments

<dBm>

Default

-99

Get example

```
> get wl r t
R000005
-99
```

Set example

```
> set wl r t -70
R000008
Set OK
```

wlan.rssi_average

Abbreviation	wl r
Access	get/set

Description

The number of consecutive wlan RSSI readings to average. Averaging smooths variations in the measured signal strength of the AP. Set to 0 to disable averaging.

Arguments

<# readings>

Default

5

Get example

```
> get wl r
R000003
5
```

Set example

```
> set wl r 12
R000008
Set OK
```

wlan.scan.active_dwell

Abbreviation	wl s a
Access	get/set

Description

Time spent listening on a channel when actively scanning for APs.

Arguments

<milliseconds>

Default

75

Get example

```
> get wl s a
R000004
75
```

Set example

```
> set wl s a 90
R000008
Set OK
```

wlan.scan.channel_mask

Abbreviation	wl s m
Access	get/set

Description

Channels to scan. Specified by a hex bitmask.

Arguments

<channel_bitmask>

where bits correspond to channels, as shown in the following examples:

Channels :	11	10	9	8	7	6	5	4	3	2	1	channel_bitmask
Channel 1 only	0	0	0	0	0	0	0	0	0	0	1	0x001
Channels 11/6/1	1	0	0	0	0	1	0	0	0	0	1	0x421
All channels	1	1	1	1	1	1	1	1	1	1	1	0x7FF

Default

0x7FF

Get example

```
> get wl s m
R000007
0x7FF
```

Set example

Set channels 11 & 1 only

```
> set wl s m 0x401
R000008
Set OK
```

wlan.scan.home_dwell

Abbreviation	wl s h
Access	get/set

Description

Time spent listening on the softap home channel when scanning in softap mode.

Arguments

<milliseconds>

Default

50

Get example

```
> get wl s h  
R000004  
50
```

Set example

```
> set wl s h 70  
R000008  
Set OK
```

wlan.scan.num_probes

Abbreviation	wl s n
Access	get/set

Description

Number of 802.11 probes to send when actively scanning.

Arguments

<# probes>

Default

3

Get example

```
> get wl s n  
R000003  
3
```

Set example

```
> set wl s n 5  
R000008  
Set OK
```

wlan.scan.passive_dwell

Abbreviation	wl s p
Access	get/set

Description

Time spent listening on a channel when passive scanning.

Arguments

<milliseconds>

Default

110

Get example

```
> get wl s p
R000005
110
```

Set example

```
> set wl s p 130
R000008
Set OK
```

wlan.scan.retries

Abbreviation	wl s r
Access	get/set

Description

Number of times to repeat a full scan when the **scan command** is issued.

Arguments

<# retries>

Default

3

Get example

```
> get wl s r  
R000003  
3
```

Set example

```
> set wl s r 5  
R000008  
Set OK
```

wlan.scan.type

Abbreviation	wl s t
Access	get/set

Description

Type of scan. Passive is listen-only, active transmits 802.11 probe management frames.

Arguments

<passive/active>

Default

active

Get example

```
> get wl s t  
R000008  
active
```

Set example

```
> set wl s t passive  
R000008  
Set OK
```

wlan.security

Abbreviation	wl c
Access	get/set

Description

The wlan security type. If set to Auto, the security type is automatically populated when joining a network (except WEP). Alternatively, the security type may be forced to one of the following options: Open, WEP, WPA-AES, WPA-TKIP, WPA2-AES, WPA2-Mixed, WPA2-TKIP.

Note! WEP security MUST be set to use WEP. We recommend against WEP, as it is **not** secure.

Arguments

<security type>

Default

Auto

Get example

```
> get wl c
R000006
Auto
```

Set example

```
> set wl c WPA2-AES
R000008
Set OK
```

wlan.ssid

Abbreviation	wls
Access	get/set

Description

The name of the AP the wlan client interface will attempt to join.

Arguments

<ssid>

Default

""

Get example

```
> get wl s  
R000012  
My Home AP
```

Set example

```
> set wl s "Obi-wan Kenobi"  
R000008  
Set OK
```

wlan.tx_power

Abbreviation	wl t
Access	get/set

Description

The Wi-Fi RF transmit power in units of dBm. Range is 0-31. The maximum achievable transmit power is dependent on the module.

Arguments

<dBm>

Default

31

Get example

```
> get wl t  
R000004  
31
```

Set example

```
> set wl t 15  
R000008  
Set OK
```

Release Notes for WiConnect v2.1

WiConnect 2.1 provides the following new functionality as well as general improvements and stability enhancements.

- **Secure HTTP webserver** with [RESTful API and Websockets](#) that can be used on the wireless softAP or WLAN client interface. The web server enables 100% wireless control and data exchange with the module using a web client such as a phone, tablet or PC. See [Networking and Security, HTTP Server with RESTful API](#).
- A **responsive web application** based on the [WiConnect JavaScript API](#) library. The web app may be 100% customized to suit end-product requirements. The new WiConnect web app includes a web console and graphical drag and drop file browser.
- **Device discovery** to enable network clients to discover ACKme devices and WiConnect services (such as the HTTP webserver), and enables WiConnect to discover services offered by network servers. The discovery protocol supports various protocols including [mDNS/Zeroconf](#), [LLMNR](#) and [Netbios](#). See [Networking and Security, Network Discovery](#).
- **SMTP email client** enabling the module to send secure email
- Support for external **bulk serial flash** up to 128 MBytes. See [File System, Internal, Extended and Bulk Flash](#).
- Out-of-band GPIO to indicate a status change to one or more configured events e.g. TCP client data available, Network status change, etc
- Auto-connect feature for TCP/UDP clients and UDP server. See [TCP Client Auto Connect](#), [UDP Client Auto Connect](#) and [UDP Server Auto Connect](#).
- API for **TCP keepalive**, see [Wireless Serial Port](#) example.
- Improved power supply monitoring and brownout performance.
- Force safe mode option to aid validation of recovery, see [App Note, Recovery from Safe Mode](#).
- API to dynamically read and write all GPIOs simultaneously.
- Automatic HTTP POST of dynamic module information to a remote HTTP server.
- ADC lookup table automatically converts raw ADC values into meaningful pre-calibrated data.
- Individual control of GPIO direction and output values in ultra-low power sleep mode

Versions

Version	Release Date
wiconnect-2.1.0.15	Feb 3, 2015

Known Issues (v2.1.0)

Last Updated : Feb 3, 2015

All known issues in v2.0.0 have been fixed in the v2.1.0 release, except those listed below.

1. TBD	
Description	N/A
Solution	N/A

Changelog

v2.1.0 : Changes for v2.1.0 are listed in the following tables.

Commands Added

Change	Description
<code>faults_print</code>	Display system faults (previously available in safe mode only).
<code>faults_reset</code>	Reset system faults (previously available in safe mode only).
<code>force_safemode</code>	Force a module into safe mode.
<code>format_flash</code>	Format extended or bulk flash.
<code>gpios_dir</code>	Set the direction of all GPIOs simultaneously.
<code>gpios_get</code>	Get the value of all GPIOs simultaneously.
<code>gpios_set</code>	Set the value of all GPIOs simultaneously.
<code>mdns_discover</code>	Discover mDNS services on the local network.
<code>network_verify</code>	Quickly verify the credentials of a WLAN network.
<code>smtp_send</code>	Send an email via an external SMTP server.

Commands Changed

Change	Description
<code>adc_take_sample</code>	voltage and lookup table options added
<code>file_create</code>	-u file unprotect option added. -o stream open option added.
<code>http_download</code>	-u file unprotect option added
<code>ls</code>	-l option added, equivalent to -v, for linux compatibility

Variables Added

Change	Description
<code>broadcast.data</code>	Specify information fields to be periodically transmitted in a packet (deprecates <code>broadcast.udp.data</code>).
<code>broadcast.http.host</code>	Periodically issue JSON formatted data to the specified host as an HTTP post request
<code>broadcast.interface</code>	Set default interface used to transmit informational packets (deprecates <code>broadcast.udp.interface</code>),
<code>broadcast.interval</code>	Set time between transmission of informational packets (deprecates <code>broadcast.udp.interval</code>)
<code>bus.stream.flush_time_reset</code>	Flag whether to reset the flush timer if data is received before flush timeout expires.
<code>gpio.sleep</code>	Set the state of the GPIO when sleeping.
<code>http.server.api_enabled</code>	Enable the HTTP Server API.

<code>http.server.denied_filename</code>	Set the filename of the HTTP Server unauthorized page.
<code>http.server.cors_origin</code>	Specify allowed origins for HTTP cross-site requests.
<code>http.server.enabled</code>	Enable the HTTP Server.
<code>http.server.interface</code>	Specify HTTP Server interface.
<code>http.server.max_clients</code>	Specify maximum clients HTTP server allows to connect.
<code>http.server.notfound_filename</code>	Set the name of the HTTP Server 404 not found error page.
<code>http.server.password</code>	Set HTTP Server password.
<code>http.server.port</code>	Set HTTP Server port.
<code>http.server.root_filename</code>	Set HTTP Server root filename.
<code>http.server.tls_cert</code>	Set HTTP Server TLS certificate filename.
<code>http.server.tls_enabled</code>	Enable HTTP Server TLS.
<code>http.server.tls_key</code>	Set HTTP Server TLS key filename.
<code>http.server.tls_verify_peer</code>	Client connecting to HTP server must provide a valid TLS certificate for validation.
<code>http.server.username</code>	Set HTTP Server root username.
<code>mdns.enabled</code>	Enable mDNS.
<code>mdns.interface</code>	Specify the network interface on which the mDNS daemon runs.
<code>mdns.name</code>	Set the mDNS domain name of the module.
<code>mdns.service</code>	Advertise a module server via mDNS.
<code>mdns.ttl</code>	Set the timeout of the module's mDNS response.
<code>network.dns</code>	Return a comma separated list of the DNS server addresses that the wlan interface is using.
<code>setup gpio mode</code>	Specify when to run the command configured with <code>setup gpio cmd</code> .
<code>email.name_address</code>	Set the name and/or address of the email client.
<code>email.smtp.host</code>	Set external SMTP host.
<code>email.smtp.password</code>	Set SMTP password.
<code>email.smtp.port</code>	Set SMTP port.
<code>email.smtp.username</code>	Set SMTP user name.
<code>softap.hide_ssid</code>	Hide or show the SSID (name) of the softAP.
<code>system.adc.vref</code>	Set ADV voltage reference.
<code>system.bflash_cs_gpio</code>	Set bulk flash chip select GPIO
<code>system.memory_usage</code>	Display the percentage of available heap memory in use
<code>system.oob.event_mask</code>	Set Out-of-band GPIO event mask
<code>system.oob gpio</code>	Set Out-of-band GPIO
<code>system.oob gpio_level</code>	Set Out-of-band GPIO level
<code>system.oob.rising_edge_mask</code>	Set Out-of-band GPIO rising edge mask

<code>system.oob.status</code>	Set Out-of-band GPIO status
<code>tcp.client.auto_interface</code>	Interface to auto-start TCP client connection.
<code>tcp.client.auto_retries</code>	Determine number of attempts to reconnect after remote server disconnects client.
<code>tcp.client.auto_start</code>	Enable/disable TCP client auto-start feature.
<code>tcp.client.remote_host</code>	Set domain or IP address of remote tcp server to auto connect.
<code>tcp.client.remote_port</code>	Set port of remote tcp server to auto connect.
<code>tcp.keepalive.enabled</code>	Enable/disable TCP keepalive feature of network stack.
<code>tcp.keepalive.initial_timeout</code>	Set number of seconds for initial keepalive expiration.
<code>tcp.keepalive.retry_count</code>	Set maximum number of times to retry keepalive.
<code>tcp.keepalive.retry_timeout</code>	Set period of keepalive retries.
<code>tcp.server.tls_cert</code>	Set the server TLS CA certificate filename.
<code>tcp.server.tls_enabled</code>	Enable server TLS.
<code>tcp.server.tls_key</code>	Set the server TLS certificate key filename.
<code>tcp.server.tls_verify_peer</code>	Validate TLS certificate of connecting clients.
<code>udp.client.auto_interface</code>	Set interface to auto-start UDP client connection.
<code>udp.client.auto_start</code>	Enable/disable UDP client auto-start feature.
<code>udp.client.remote_host</code>	Set domain or IP address of remote udp server to auto connect.
<code>udp.client.remote_port</code>	Set port of remote UDP server to auto connect.
<code>udp.server.lock.client</code>	Lock UDP server to first client from which it receives a packet.
<code>udp.server.remote_host</code>	Set domain or IP address of remote udp client to auto connect.
<code>udp.server.remote_port</code>	Set port of remote UDP client to auto connect.

Variables Changed

Change	Description
<code>broadcast.udp.data</code>	Renamed to <code>broadcast.data</code> . Old name deprecated and will be removed in future releases.
<code>broadcast.udp.interface</code>	Renamed to <code>broadcast.interface</code> . Old name deprecated and will be removed in future releases.
<code>broadcast.udp.interval</code>	Renamed to <code>broadcast.interval</code> . Old name deprecated and will be removed in future releases.
<code>gpio.usage</code>	Now returns GPIO direction settings for each Standard I/O GPIO, in addition to information returned previously

Revision History

Revision	Date	Change Description
ARG-WiConnect-200R	Oct, 2014	Release for WiConnect v2.0.0
ARG-WiConnect-201R	Nov, 2014	Release for WiConnect v2.0.1
ARG-WiConnect-210R	Feb, 2015	Release for WiConnect v2.1.0

Glossary

In most cases, acronyms and abbreviations are defined on first use. A comprehensive list of acronyms and other terms used in ACKme Networks documents are provided on the ACKme Networks website at ACK.me/FAQs/Glossary of Terms.

ACKme reserves the right to make changes without further notice to any products or data herein to improve reliability, function, or design. Information furnished by ACKme is believed to be accurate and reliable. However, ACKme does not assume any liability arising out of the application or use of this information, nor the application or use of any product described herein, neither does it convey any license under its patent rights nor the rights of others.

ACKme Networks
US Headquarters
2 North Santa Cruz Ave
Suite #207
Los Gatos CA 95030
© 2015 ACKme Networks Inc. All rights reserved.
ARG-WiConnect-210R • WiConnect Reference Guide February 19, 2015

Contact Information
+1 (408) 402 5708
<http://ack.me/contact>

