**Note**: *Arguments in [ ] are optional and those in <> are mandatory. Command response column contains only response description and payload fields.*

## Generic AT Commands

### AT – Check State

|  |  |
| --- | --- |
| **Command** | at |
| **Response** | Success: OK  Failure: Error |
| **Parameters** | NA |
| **Example** | at |

### AT+SYSNAME – Set System Name

|  |  |
| --- | --- |
| **Command** | at+sysname=<Host Name> |
| **Response** | Success: OK  Failure: Error |
| **Parameters** | Host Name: host name to configure at+sysname=T2 |
| **Example** | at+sysname=T2 |

### AT+WAKEUPCFG – Configures System Wakeup

|  |  |
| --- | --- |
| **Command** | at+wakeupcfg=<wakeup pin>, < wakeup level>, <host wakeup pin> , <host wakeup mode> |
| **Response** | Success: OK  Failure: Error |
| **Parameters** | wakeuppin: GPIO pin to be used for Talaria TWO wakeup. 0 to disable.  wakeup level: Talaria TWO wakeup level. 1: High ; 0: Low  host wakeup pin: GPIO pin to be used for interrupt to wake up the host.  host wakeup mode: Interrupt mode to be used to wake up the host: 0: None ; 1: Low ; 2: High |
| **Example** | at+wakeupcfg=4,1,5,1  **Note**:   1. Ensure command execution is successful in the minicom terminal and verify the same in the serial console logs as well. 2. All the GPIOs are pulled high internally by default except GPIO18 (pulled low). 3. To generate a low-level interrupt, GPIO must be pulled low externally. 4. To generate a high-level interrupt, pull down that particular GPIO externally through a pull down register. Only then the high interrupt will be detected. 5. Since there is always a pull down on high level interrupt GPIO, some amount of small current is always sinking through that pin, which will add onto the power save suspend current. 6. Hence it is recommended to use low-level interrupt for low power use case. |

### AT+SYSSLEEP – Puts System in Suspend Mode

|  |  |
| --- | --- |
| **Command** | at+syssleep=<suspend time> |
| **Response** | Success: OK  Failure: Error |
| **Parameters** | suspend time: Time in milliseconds, 0 indicates infinite suspend time. |
| **Example** | at+syssleep=100  **Note**:   1. In the case of infinite suspend time, it is mandatory to set at+wakeupcfg (refer [AT+WAKEUPCFG – Configures System Wakeup](#_AT+WAKEUPCFG_–_Configures) for more details) command to use at+syssleep command. 2. Talaria TWO will not enter suspend/deep sleep mode when it is in active state (traffic timeout set to zero). 3. Send a break character on UART to wake up Talaria TWO temporarily. In such cases, Talaria TWO will wait for a new command from host and goes back to suspend state immediately after executing the command. |

### AT+ECHO – Enable/Disable the Command Echo Feature

|  |  |
| --- | --- |
| **Command** | at+echo=<flag> |
| **Response** | Success: OK  Failure: Error |
| **Parameters** | Flag: Enable/Disable,   * 1: Enable the echo (default) * 0: Disable the echo |
| **Example** | at+echo=1 |

### AT+VER – Prints Software Version Number

|  |  |
| --- | --- |
| **Command** | at+ver |
| **Response** | Success: Version number followed by OK  Failure: Error |
| **Parameters** | NA |
| **Example** | at+ver |

### AT+RESET – Resets the Talaria TWO Module

|  |  |
| --- | --- |
| **Command** | at+reset |
| **Response** | Success: Upon successful reset, module responds with command prompt ‘>’, which implies Talaria TWO is ready to accept commands.  Failure: Error |
| **Parameters** | NA |
| **Example** | at+reset |

**Note**: This command forcefully resets the Talaria TWO module core and comes out with a fresh boot.

## Wi-Fi AT Commands

### AT+WCON - Connect to WLAN Network as a Station

|  |  |
| --- | --- |
| **Command** | at+wcon=<SSID>,[Passphrase],[Timeout],[Enterprise\_security\_type],[Identity],[ca\_certificate path],[client\_certificate path],[private\_key path],[private\_key\_password],[Identity2],[password],[phase2auth] |
| **Response** | Success: OK  Failure: Error |
| **Parameters** | SSID: Name of the Access Point  Passphrase: Password (optional for open security)  Timeout: Timeout in seconds, default being 180 seconds  Enterprise\_security\_type:  1 - EAP-PSK  2 - EAP-TLS  3 - EAP-PEAP  Identity: Identity for enterprise security  ca\_certificate path: Path to CA certificate in Talaria TWO filesystem  client\_certificate path: Path to client certificate in Talaria TWO filesystem  private\_key path: Path to private key file in Talaria TWO filesystem  private\_key\_password: Password of private key  Identity2: Phase 2 identity  password: Phase 2 password  phase2auth: Phase 2 authentication |
| **Example** | **Open security**  at+wcon=rr,,30  **Personal security**  at+wcon=rr,abcd@123,30  **Enterprise security**  **TLS**:  at+wcon=ssid,abc,30,2,eap-tls@innophaseinc.com,/sys/ca.pem,/sys/client.pem,/sys/client.key,password  **PSK**:  at+wcon=ssid,0123456789abcdef0123456789abcdef,30,1,psk,,,,,  **PEAP**:  at+wcon=ssid,abc,30,3,anonymous,/sys/ca.pem,,,,eap-peap,password,MSCHAPv2 |

### AT+WDIS - Disconnect the Connection

|  |  |
| --- | --- |
| **Command** | at+wdis |
| **Response** | Success: OK  Failure: Error |
| **Parameters** | NA |
| **Example** | at+wdis |

### AT+WSTATUS - Get WLAN Status

|  |  |
| --- | --- |
| **Command** | at+wstatus=<ID> |
| **Response** | Success: Respective status values followed by OK  Failure: Error |
| **Parameters** | ID: status identifier  The following ID values are used to get the parameters   * 0: IP Address. Prints node IP address, subnet mask and gateway address * 1: RSSI value * 2: Wi-Fi counters (Tx packet count, packet count, and so on.) * 3: WLAN MAC address * 4: Tx power |
| **Example** | at+wstatus=0  at+wstatus=1  at+wstatus=2  at+wstatus=3  at+wstatus=4 |

### AT+WSCAN - Get the WLAN Scan Operation Results

|  |  |
| --- | --- |
| **Command** | at+wscan=[SSID],[BSSID],[Channel] |
| **Response** | Success: Scan results followed by OK  Failure: Error  Scan results are printed in following order  <BSSID>:<channel>:<RSSI>:<SSID> |
| **Parameters** | SSID: Name of the Access Point  BSSID: Access Point MAC address (optional)  Channel: WLAN channel number(optional) |
| **Example** | at+wscan=InnoPhase,,  at+wscan=InnoPhase,,6 |

### AT+WCFGSET - Set WLAN Configuration

|  |  |
| --- | --- |
| **Command** | at+wcfgset=<ID>, <value-1>, ………, <value-N> |
| **Response** | Success: OK  Failure: Error |
| **Parameters** | ID: Configuration Identifier  Value: Configuration Value  The following ID values are used to set the parameters   * 0: TX power in dBm (0 to 20) * 1: Set interface IPv4, netmask, gateway and DNS addresses   The following parameters can be set   * ipaddr4: IP address, as big-endian integer * netmask: netmask, as big-endian integer * gw: default-route address, as big-endian integer. * dns\_server: DNS server address, as big-endian integer. * 2: Scan configuration parameters   The following parameters can be set:   * num\_probes : Number of probe request to send (default is 0) * idle\_slots: Maximum number of idle slots to decide if we should keep listening (default value is 3). * Txrate: Rate to use for sending probe requests (default value is 0) * min\_listen\_time: Minimum amount of time (in milliseconds) to listen for probe responses on the channel after transmitting the probe request (default value is 8000). * max\_listen\_time: Maximum amount of time (in milliseconds, including listen and probe requests) to stay on the channel (default value is 24000). * wait\_time: Idle time between each channel (giving other parties access to the media) (default value is 0). * probe\_tx\_time: Time (in milliseconds) after which a probe request is aborted if transmission did not happen.(default value is 8000). * 3: Set PMK Path   The PMK Path can be set using the following parameter:   * pmk\_path: pmk\_path, path should start with ‘/’ |
| **Example** | at+wcfgset=0,10  at+wcfgset=0,9  at+wcfgset=0,20  at+wcfgset=0,0  at+wcfgset=1,192.168.1.1,255.255.255.0,192.168.1.3,192.168.1.3  at+wcfgset=2,0,3,0,8000,24000,0,8000  at+wcfgset=3,/data/pmkpath.data |

### AT+WCFGGET- Get WLAN Configuration

|  |  |
| --- | --- |
| **Command** | at+wcfgget=<ID> |
| **Response** | Success: OK  Failure: Error |
| **Parameters** | ID: Configuration Identifier  The following ID values are used to set the parameters:   * 0: TX power in dBm ( -10 to 20) * 1: Get interface IPv4, netmask, gateway and DNS addresses   The following parameters can be used for Get   * + ipaddr4: IP address, as big-endian integer   + netmask: netmask, as big-endian integer   + gw: default-route address, as big-endian integer.   + dns\_server: DNS server address, as big-endian integer. * 2: Scan configuration parameters   The following parameters can be modified to get WLAN configuration   * + num\_probes : Number of probe request to send (default is 0)   + idle\_slots: Max number of idle slots to decide if we should keep listening (default value is 3).   + txrate: Rate to use for sending probe requests (default value is 0)   + min\_listen\_time: Minimum amount of time (in milliseconds) to listen for probe responses on the channel after transmitting the probe request (default value is 8000).   + max\_listen\_time: Maximum amount of time (in milliseconds, including listen and probe requests) to stay on the channel (default value is 24000)   + wait\_time: Idle time between each channel (giving other parties access to the media) (default value is 0).   + probe\_tx\_time: Time (in milliseconds) after which a probe request is aborted if transmission did not happen (default value is 8000). * 3: Get PMK Path: Called to retrieve PMK Path. |
| **Example** | at+wcfgget=0  at+wcfgget=1  at+wcfgget=2  at+wcfgget=3 |

### AT+WPMCFG – Set WLAN Power Management Configuration

|  |  |
| --- | --- |
| **Command** | at+wpmcfg=<listen interval>,<traffic timeout>, <pspoll>, <dyn\_listen\_int>[,<starx\_nap>, <sta\_only\_bc>, <txps>, <mcast\_dont\_care>,<dtim>] |
| **Response** | Success: OK  Failure: Error |
| **Parameters** | |  |  |  | | --- | --- | --- | | **Parameter** | **Description** | **Min/Max values** | | listen interval | Beacon listen interval | 0/integer range | | traffic timeout | Traffic timeout in milliseconds. The traffic timeout parameter specifies the amount of time (in milliseconds) that the device should stay awake with the radio powered-up after a transmission (to quickly receive any replies that may be the result of the transmission) | 0/integer range | | pspoll | Use PS-poll | 0/1 (True/False) | | dyn\_listen\_int | Dynamic listen interval | 0/1 | | starx\_nap | Turn off receiver for inappropriate frames for station | 0/1 | | sta\_only\_bc | Do not receive multicast frames that are not applicable | 0/1 | | txps | Send outgoing frames without leaving Wi-Fi  on power save | 0/1 | | mcast\_dont\_care | Ignore the multicast flag in beacons | 0/1 | | dtim | Wakes up only at effective listen interval and does not switch to listen every beacon in case of beacon miss | 0/1 | | |
| **Example** | at+wpmcfg=3,100,1,1,0,0,0,0,1 |

### AT+WREGDOMAIN – Set WLAN Regulatory Domain

|  |  |
| --- | --- |
| **Command** | at+wregdomain=<regulatory domain> |
| **Response** | Success: OK  Failure: Error |
| **Parameters** | regulatory domain: Regulatory domain configuration.  The valid values are:   * 0: FCC * 1: ETSI * 2: TELEC * 3: KCC * 4: SRCC * ?: Get regulatory domain |
| **Example** | at+wregdomain=0  at+wregdomain=? |

### AT+SOCSRV - Create Server Socket

|  |  |
| --- | --- |
| **Command** | at+socsrv=<Domain>,<Type>,<Protocol>, <Port> |
| **Response** | Success: “CONNECT:<Socket identifier>” message followed by OK  Failure: Error  Refer: [Command Response Description](#_Command_Response_Description_1) for more details |
| **Parameters** | Domain: Specifies the protocol family of the created socket  0: INET: For network protocol IPv4  Type: Specifies the communication semantics  0: STREAM: Reliable stream-oriented service or Stream Sockets  1: DGRAM: Datagram service or Datagram Sockets  Protocol: Specifies a transport to be used with the socket:  0: TCP  1: UDP  Port: Specifies the port number to create the server |
| **Example** | at+socsrv=0,0,0,9000  at+socsrv=0,1,1,9000 |

### AT+SOCCLI - Create Client Socket

|  |  |
| --- | --- |
| **Command** | at+soccli=<Domain>,<Type>,<Protocol>, <Port>, <Host> |
| **Response** | Success: “CONNECTED:<Socket identifier>” message followed by OK  Failure: Error  Refer section [Command Response Description](#_Command_Response_Description_1) for more details |
| **Parameters** | Domain: Specifies the protocol family of the created socket  0: INET: For network protocol IPv4.  Type: Specifies the communication semantics  0: STREAM: Reliable stream-oriented service or Stream Sockets  1: DGRAM: Datagram service or Datagram Sockets  Protocol: Specifies a transport to be used with the socket:  0: TCP  1: UDP  Port: Specifies the port number to create the server  Host: server IP address or host name |
| **Example** | at+soccli=0,0,0,23,192.168.2.184 |

### AT+SOCSEND - Write Data to Socket

|  |  |
| --- | --- |
| **Command** | at+socsend=<Socket ID>,<Type>,<Length>, <Data>, [Timeout] |
| **Response** | Success: OK  Failure: Error |
| **Parameters** | Socket ID: Socket identifier  Type: Data format, binary/ASCII  Length: Number of bytes to send (in decimal). Length is limited to 1024 bytes  Data: Data to send  Timeout: Timeout value in seconds. If no value is provided, 90s is taken as default (applicable only for TCP socket) |
| **Example** | at+socsend=1,ASCII,5,data\_1,40  at+socsend=1,ASCII,1024, data |

### AT+SOCCLOSE - Close Socket

|  |  |
| --- | --- |
| **Command** | at+socclose=<Socket ID> |
| **Response** | Success: OK  Failure: Error |
| **Parameters** | Socket ID: Socket identifier |
| **Example** | at+socclose=0 |

### AT+HCSTART - Start HTTP Client

|  |  |
| --- | --- |
| **Command** | at+hcstart=<Host Name>, <Port>, [<Secured>], [Certificate Name] |
| **Response** | Success: HTTP client identifier followed by OK  Failure: Error  Refer section [Command Response Description](#_Command_Response_Description_1) for more details |
| **Parameters** | Host Name: Remote server host name. It is either the domain name or the IP address.  Port: Server port  Secured:  0: Non-Secured (HTTP),  1: HTTPS without server certificate validation,  2: HTTPS with server certificate validation  **Note**: If no secured parameter is provided, 0 will be taken as the value by default.  Certificate Name: SSL certificate. This is needed only when Secured=2. |
| **Example** | at+hcstart=192.168.2.184,80 |

### AT+HCREQSND – Send HTTP Client Request

|  |  |
| --- | --- |
| **Command** | at+hcreqsnd=<HCID>,<Method>,<URI>, <Length>,<Data> |
| **Response** | Success:  200 - HTTP status code  Failure: ERROR |
| **Parameters** | HC ID: http client identifier  Method: operation   * 1: GET * 2: HEAD * 3: POST * 4: PUT * 5: DEL   URI: URI request string  Length: Number of bytes to send  Data: Data to send |
| **Example** | at+hcreqsnd=1,1,http://192.168.2.184/index.html  at+hcreqsnd =at+hcreqsnd=0,3,/post,6,MAC\_ID  Note: Recommended practice of retry upon HTTP send failure will be:   * Open HTTP client connection * Set required headers * Execute HTTP GET/POST * If ERROR occurs, close the connection * Reopen HTTP client connection * Execute HTTP GET/POST   so on and so forth... |

### AT+HCHDRSET – Set HTTP Client Header

|  |  |
| --- | --- |
| **Command** | at+hchdrset=<Header ID>,<Value> |
| **Response** | Success:  OK  Failure: ERROR |
| **Parameters** | Header ID: header ID  value: header value  The following values are valid header IDs and its corresponding header:   |  |  | | --- | --- | | **ID** | **Header Name** | | 1 | Allow | | 2 | Authorization | | 3 | Connection type | | 4 | Content encoding | | 5 | Content length | | 6 | Content range | | 7 | Content type | | 8 | Http cookie | | 9 | Http cookie2 header | | 10 | Date and time | | 11 | Header expire date and time | | 12 | Email address of user making the request. | | 13 | Domain name of server | | 14 | Resource modified date and time | | 15 | Date and time which resource was last modified | | 16 | Redirect URL | | 17 | Implementation specific header | | 18 | Request only part of remote resource | | 19 | Address of previous page from where it requested current page | | 20 | Address of server generated response | | 21 | Header to send cookie form server | | 22 | Encoding Used | | 23 | String to specify the client | | 34 | Authentication method used for access | |
| **Example** | at+hchdrset=3,keep-alive  at+hchdrset=13,192.168.2.184 |

### AT+HCHDRDEL – Delete HTTP Client Header

|  |  |
| --- | --- |
| **Command** | at+hchdrdel=<HC ID> |
| **Response** | Success:  OK  Failure: ERROR |
| **Parameters** | <if applicable> |
| **Example** | at+hchdrdel=0 |

### AT+HCCLOSE – Close HTTP Connection

|  |  |
| --- | --- |
| **Command** | at+hcclose=<HC ID> |
| **Response** | Success:  OK  Failure: ERROR |
| **Parameters** | HC ID: HTTP client identifier |
| **Example** | at+hcclose=0 |

### AT+MDNSREG – Register MDNS Service

|  |  |
| --- | --- |
| **Command** | at+mdnsreg=<Service Name>,<Service type>, <Service Proto>,<Service Port>,<Service Description> |
| **Response** | Success:“SER-ID:<Serviced ID>” message  followed OK  Failure: ERROR |
| **Parameters** | Service Name: The service name  Service type: Type of service (HTTP)  Service type: The service protocol  0: UDP  1: TCP  Service Port: Port number used  Service Description: Service description |
| **Example** | at+mdnsreg=<servicename>,\_HTTP,0,6553,<servicedesc> |

### AT+MDNSDREG – De-Register MDNS Service

|  |  |
| --- | --- |
| **Command** | at+mdnsdreg=<Service ID> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | Service ID: The service identifier |
| **Example** | at+mdnsdreg=0 |

### AT+MDNSSTART – Start MDNS

|  |  |
| --- | --- |
| **Command** | at+mdnsstart |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | NA |
| **Example** | at+mdnsstart |

### AT+MDNSSTOP – Stop MDNS

|  |  |
| --- | --- |
| **Command** | at+mdnsstop |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | NA |
| **Example** | at+mdnsstop |

### AT+NHOSTIPGET – Get Host IP by Name

|  |  |
| --- | --- |
| **Command** | at+nhostipget=<Host name>,[Family] |
| **Response** | Success: “IP<family>:<ip address>” message followed by OK  Failure: ERROR  Refer section [Command Response Description](#_Command_Response_Description_1) for more details |
| **Parameters** | Host Name: The host name  Family: protocol family  0: IPv4  1: IPv6  Default: It will be trying to resolve IPv4 first. If that fails, then tries IPv6. |
| **Example** | at+nhostipget=www.google.com,0 |

### AT+NPING – Send Ping to Host

|  |  |
| --- | --- |
| **Command** | at+nping=<IP address> |
| **Response** | Success: standard ping response followed by OK  Failure: ERROR  Refer section [Command Response Description](#_Command_Response_Description_1) for more details |
| **Parameters** | IP address: IP address to ping. |
| **Example** | at+nping=192.168.2.184 |

### AT+CERTADD – Stores Certificate

|  |  |
| --- | --- |
| **Command** | at+certadd=<Cert Name>,<Cert Len> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | Cert Name: Name of certificate.  Cert Len: Certificate length in bytes  Procedure to send certificate   * After command validation Talaria TWO will send “<” as response to command. * Send certificate after receiving the command response * OK/ERROR status message will send as response.   If certificate name already exists in T2, it will be overwritten.  **Certificate will be stored in RAM.** |
| **Example** | at+certadd=ssl.pem,2614 |

### AT+CERTDEL – Deletes Certificate

|  |  |
| --- | --- |
| **Command** | at+certdel=<Cert name> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | Cert Name: Name of certificate |
| **Example** | at+certdel=ssl.pem |

### AT+HSSTART – Start HTTP Server

|  |  |
| --- | --- |
| **Command** | at+hsstart=<Port number> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | Port number: port number |
| **Example** | at+hcstart=192.168.2.184,80 |

### AT+MQTTCONF – Set MQTT configurations.

One or more MQTT configurations can be set using this command. <key> is the name of the configuration and <val> is the value for the <key> to be set.

|  |  |
| --- | --- |
| **Command** | at+mqttconf=<key>,<val>,<key>,<val>…. |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | key: The name of the parameter to be set.  The supported parameters are:   * clientid : MQTT client id (Connection will fail if this is not set) * username : MQTT username * password : MQTT password * cleansession : 0/1 (Default 1) * kainterval : Keep alive interval (Default 60 Sec) |
| **Example** | at+mqttconf=clientid,123456  at+mqttconf=username,admin  at+mqttconf=password,xyz  at+mqttconf=kainterval,10 /\*keepalive interval of 10 sec\*/ |

### AT+MQTTCONN – Connect to MQTT Broker

|  |  |
| --- | --- |
| **Command** | at+mqttconn<host name/IP address>,<port number>,[<transport>],[cert verify],[<CA cert file name>],[<ws url>],[<client cert file name>],[<client key file name>],[<connection timeout>] |
| **Response** | Connection ID followed by command response  Success: OK  Failure: ERROR |
| **Parameters** | host name/IP address: It is either Fully Qualified Domain name (FQDN) of the server or the IP address of the server to which the MQTT client opens the connection.  For example: mqtt.eclipseprojects.io or 137.135.83.217  port number: This gives the port number of the server to which the MQTT client opens the connection to. The client can specify the port when the server is running on a non-standard port.  Default standard port: 1833: MQTT client, 8883: MQTT over SSL.  transport:  0: MQTT over TCP (Default)  1: MQTT over TLS (Secured MQTT)  2: WebSocket  3: MQTT over secured WebSocket  cert verify: Used to enable certificate verification in case SSL is enabled.  CA cert file name: Name of the CA certificate to be used for server certificate authentication in case SSL is enabled. The CA certificate must be provisioned before authentication.  ws url: WebSocket URL (Default is “/”).  client cert file name: Name of the client certificate to be used for client authentication. Client certificate must be provisioned before authentication.  client key file name: Name of the client key file to be used for client authentication. Client key must be provisioned before authentication.  connection timeout: Maximum time to wait for the connection to go through. Maximum time being ~300s. |
| **Example** | at+mqttconn=test.mosquitto.org,8884,1,1,/data/mosquitto.org.crt,/mqtt,/data/client.crt,/data/client.key |

### AT+MQTTDISCONN – Disconnect MQTT Client Connection

|  |  |
| --- | --- |
| **Command** | at+mqttdisconn=<nwid > |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | nwid: Network ID |
| **Example** | at+mqttdisconn=0 |

### AT+MQTTSUB – Subscribe to the MQTT topic

|  |  |
| --- | --- |
| **Command** | at+mqttsub=<nwid >,<topic>,<qos> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | nwid: Network ID  topic: Topic to subscribe  qos: Qos of the topic |
| **Example** | at+mqttsub=0,inno/test,0 |

### AT+MQTTUNSUB – Un-Subscribe to the MQTT Topic

|  |  |
| --- | --- |
| **Command** | at+mqttunsub=<nwid >,<topic> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | nwid: Network ID  topic: Topic to un-subscribe |
| **Example** | at+mqttunsub=0,inno/test |

### AT+MQTTPUB-Publish the MQTT Topic with Payload

|  |  |
| --- | --- |
| **Command** | at+mqttpub=<nwid >, <topic>, <qos>, <len>,<payload> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | nwid: Network id  topic: Topic to publish  qos: Qos of the topic  len: Data length  payload: Message payload |
| **Example** | at+mqttpub=0,inno/test,0,5,Hello |

### AT+NTPTIMEGET – Gets time from NTP server

|  |  |
| --- | --- |
| **Command** | at+ntptimeget |
| **Response** | Epoch time in µ seconds followed by:  Success: OK  Failure: ERROR  **Note**: Default server address is pool.ntp.org |
| **Parameters** | NA |
| **Example** | at+ntptimeget |

### AT+FOTA – Start FOTA Operation

|  |  |
| --- | --- |
| **Command** | at+fota=<option> |
| **Response** | Success: Upon successful firmware upgrade, module undergoes reset and provides AT command serial-to-wireless prompt  Failure: ERROR |
| **Parameters** | Option: Currently supported value is 1.  This parameter is used to start firmware upgrade. It uses fota\_config.json file and starts the HTTP connection, downloads the new firmware and starts upgrading the firmware. |
| **Example** | at+fota=1 |

### AT+FOTACFGADD – Update FOTA Configuration File (fota\_config.json)

|  |  |
| --- | --- |
| **Command** | at+fotacfgadd=<len> |
| **Response** | Success:  OK  Failure: ERROR |
| **Parameters** | len: size of the fota\_config.json file in bytes.  Procedure to send the certificate:   * Execute at+fotacfgadd=<len> command on the serial terminal. Now the AT command application will be waiting to receive the config file. * Send fota\_config.json file from the serial terminal. * OK/ERROR response is sent upon success/failure of the command respectively. * The fota\_config.json file sent using this command will replace the existing fota\_config.json file in root fs. |
| **Example** | at+fotacfgadd=652 |

## BT/BLE Commands

### AT+BTNIT – Initialize BLE Stack

|  |  |
| --- | --- |
| **Command** | at+btinit |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | NA |
| **Example** | at+btinit |

### AT+BLECFG – Configure BLE Device

|  |  |
| --- | --- |
| **Command** | at+blecfg=<address>,<address type>,<Device Name> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | Address: BLE mac address.  Address type: BLE address type  0: BLE public address  1: BLE random address  Device Name: BLE device name |
| **Example** | at+blecfg=02:03:04:04:03:02, 0,testble |

### AT+BLEADVCFG – Configure Advertisement Parameters

|  |  |
| --- | --- |
| **Command** | at+bleadvcfg=<Fast adv interval>,<Fast adv Duration>,<Slow adv interval>,<Slow adv Duration> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | Fast adv interval: Fast adverting interval in 625µs, range: -0x0020 to 0x4000 (default:160)  Fast adv duration: Fast advertisement duration in µs  Slow adv interval: Slow advertising interval in 625µs, range: -0x0020 to 0x4000  Slow adv duration: Slow advertisement duration in µs. After this time advertisement will be disabled |
| **Example** | at+bleadvcfg=160, 10, 160, 10 |

### AT+ BLEEXTADVCFG – Creates BLE Extended Advertisement

|  |  |
| --- | --- |
| **Command** | at+bleextadvcfg= <use>, <adv\_pri\_phy>, <adv\_sec\_phy>, <adv\_sid>, <conn\_phy>, <conn\_len> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | use: Extended (5.0) usage: 1-legacy, 2-extended, 3-legacy+extended (default: 1)  adv\_pri\_phy: Primary PHY: 1-LE 1M, 3-LE Coded S2, 4-LE Coded S8 (default: 3)  adv\_sec\_phy: Secondary PHY: 1-LE 1M, 2-LE 2M, 3-LE Coded (S2 or S8 according to adv\_pri\_phy) (default: 3)  adv\_sid: Advertising SID (0..15) (default: 13)  conn\_phy: Preferred phy(s) during connection - bit0: 1M, bit1: 2M, bit2: CodedS2, bit3: CodedS8, 0=no preference (default: 2M)  conn\_len: Maximum length of transmitted data during connection (27..251) (default: 27) |
| **Example** | at+bleextadvcfg=2,1,1,1,1,251 |

### AT+BLEADVSTART – Start BLE Advertisement

|  |  |
| --- | --- |
| **Command** | at+bleadvstart=[Adv data] |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | Adv data: Advertisement data  **Note**: Advertisement data needs to be added according to the Bluetooth SIGS assigned numbers for the AD types |
| **Example** | at+bleadvstart=02010618ff55aa0100686f6e657977656c6c00686f6e657977656c6c |

### AT+BLEADVSTOP – Stop BLE Advertisement

|  |  |
| --- | --- |
| **Command** | at+bleadvstop |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | NA |
| **Example** | at+bleadvstop |

### AT+BLESERVCFG – Creates BLE GATT Service

|  |  |
| --- | --- |
| **Command** | at+bleservcfg=<UUID> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | UUID: Universal unique identifier, it can be either 16 bit, 32 bit or 128 bit. |
| **Example** | at+bleservcfg=0x1111 |

### AT+BLESERVADD – Adds BLE GATT Service

|  |  |
| --- | --- |
| **Command** | at+bleservadd=<UUID > |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | UUID: Universal unique identifier, it can be either 16 bit, 32 bit or 128 bit. |
| **Example** | at+bleservadd=0x1111 |

### AT+BLESERVDEL – Removes BLE GATT Service

|  |  |
| --- | --- |
| **Command** | at+bleservdel=<UUID > |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | UUID: Universal unique identifier, it can be either 16 bit, 32 bit or 128 bit. |
| **Example** | at+bleservdel=0x1111 |

### AT+BLECHRADD - Add Characteristic to Service

|  |  |
| --- | --- |
| **Command** | at+blechradd= <Services uuid>,<Char uuid>, <Properties>,<Permission> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | Services uuid: Service UUID  Char uuid: Characteristic Universal unique identifier  Properties: Properties for the characteristic. The valid values are:   * 0x01: broadcast * 0x02: read * 0x04: write without response * 0x08: write * 0x10: notify * 0x20: indicate * 0x40: signed write (not supported) * 0x80: extended properties (not supported)   **Note**: To set multiple properties, pass the logical or the above values and set the required properties.  Permission: Permissions for the characteristic. The valid values are:   * 0x01: Read * 0x02: Write * 0x03: Read and Write * 0x04: Encrypt * 0x08: Authenticate * 0x10: Authorize * 0x20: Encode with key size 128 * 0x80: Signed * 0x100: Signed MITM   **Note**:   * 1. To set multiple permissions, pass the logical or the above values and set the required permissions.   2. The values for permission and properties must be configured in hexadecimal format. |
| **Example** | at+blechradd=0x1111,0x2a19,0x0c,0x03  at+blechradd=0x1111,0x2a19,0x5c,0x03 |

### AT+BLEDESCADD - Add Descriptor to Characteristics

|  |  |
| --- | --- |
| **Command** | at+bledescadd=<characteristics uuid>,<descriptor uuid>, <Properties>,<Permission > |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | characteristics uuid: Characteristics UUID  descriptor uuid: UUID of descriptor  Properties: Properties for the characteristic. The valid values are:   * 0x01: broadcast * 0x02: read * 0x04: write without response * 0x08: write * 0x10: notify * 0x20: indicate * 0x40: signed write (not supported) * 0x80: extended properties (not supported)   **Note**: To set multiple properties, pass the logical or the above values and set the required properties.  Permission: Permissions for the characteristic. The valid values are   * 0x03: Read and write   **Note**: Permission and properties values must be configured in hexadecimal format. |
| **Example** | at+bledescadd=0x2a19,0x2901,0x0c,0x03  at+bledescadd=0x2a19,0x2901,0x5c,0x03 |

### AT+BLESRVSTART – Start BLE GATT Server

|  |  |
| --- | --- |
| **Command** | at+blesrvstart |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | NA |
| **Example** | at+blesrvstart |

### AT+BLESRVSTOP – Stop BLE GATT Server

|  |  |
| --- | --- |
| **Command** | at+blesrvstop |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | NA |
| **Example** | at+blesrvstop |

### AT+BLENOTIFY – Notify BLE GATT Characteristic

|  |  |
| --- | --- |
| **Command** | at+blenotify=<char uuid>,<len>, <data> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | Char uuid: Characteristic UUID  Len: Length of data, in hexadecimal format  Data: Notification data, in ASCII format |
| **Example** | at+blenotify=0x1234,a,Hello12345 |

### AT+BLEIND – Indicates BLE GATT Characteristic

|  |  |
| --- | --- |
| **Command** | at+bleind=<char uuid>,<len>,<data> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | char uuid: Characteristic UUID  len: Length of data, in hexadecimal format  data: Indication data, in ASCII format |
| **Example** | at+bleind=2a19,a,12345hello |

### AT+BLECHARRDDATA – Sends data for BLE Characteristic Read Request

|  |  |
| --- | --- |
| **Command** | at+blecharrddata=<uuid>,<data len>,<data> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | uuid: Characteristic UUID  data len: Length of data in hexadecimal format  data: Actual data for characteristic read request, in ASCII format  If the data contains a special character, then it has to be pre-appended with a slash (0x5C) |
| **Example** | at+blecharrddata=2a29,a,6162636465 |

### AT+BLEDESCRDDATA – Sends data for BLE Descriptor Read Request

|  |  |
| --- | --- |
| **Command** | at+bledescrddata=<uuid>,<data len>,<data> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | uuid: Descriptor UUID  data len: Length of data in hexadecimal format  data: Actual data for characteristic read request, in ASCII format |
| **Example** | at+bledescrddata=2902,a,6162636465 |

### AT+BLECHARWRDATA – Acknowledges BLE Characteristic Write Request

|  |  |
| --- | --- |
| **Command** | at+blecharwrdata=<uuid>,<data len> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | uuid: Characteristic UUID  data len: Length of data written in hexadecimal format |
| **Example** | at+blecharwrdata=2a29,4 |

### AT+BLEDESCWRDATA – Acknowledges BLE Descriptor Write Request

|  |  |
| --- | --- |
| **Command** | at+bledescwrdata=<uuid>,<data len>,<data> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | uuid: Descriptor UUID  data len: Length of data in hexadecimal format  data: Actual data for characteristic read request (ASCII format) |
| **Example** | at+bledescwrdata=2a29,4,1234 |

### AT+BLESCANCFG – Configure BLE Scan Parameters

|  |  |
| --- | --- |
| **Command** | at+blescancfg=<scan\_period>,<scan\_interval>,<scan\_win>,<bscan\_interval>,<bscan\_win>,<scan\_filter> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | scan\_period: Foreground scan period in ms (default: 10240)  scan\_interval: Scan interval in 625 µs, range: 4 to 16384 (default: 96)  scan\_win: Scan window in 625 µs, range: 4 to 16384 (default: 48)  bscan\_interval: Background scan interval in 625 µs, range: 4 to 16384 (default: 2048)  bscan\_win: Background scan window in 625 µs, range: 4 to 16384 (default: 18)  scan\_filter: Filter duplicates (1=True, 0=False) (default: 1)  **Note**: Background scan is used if the device has existing connections. All the above parameters are in decimal. |
| **Example** | at+blescancfg=5000,96,48,96,24,1 |

### AT+BLECONCFG – Configure BLE Connection Parameters

|  |  |
| --- | --- |
| **Command** | at+bleconcfg=<con\_interval>, <con\_latency>, <con\_timeout>, <con\_storeparam>, <con\_interval\_min>, <con\_interval\_max> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | con\_interval: Connection interval in 1.25ms, range: 6 to 3200 (default: 80)  con\_latency: Connection latency in interval, range: 0 to 499 (default: 0)  con\_timeout: Connection timeout in ms, range: 10 to 3200 (default: 2000)  con\_storeparam: Rejects parameter update (1=True, 0=False) (default: 0)  con\_interval\_min: Minimum connection interval in 1.25ms (default: 6)  con\_interval\_max: Maximum connection interval in 1.25ms (default: 800)  **Note**: All above parameters are in decimal. |
| **Example** | at+bleconcfg=80,0,2000,0,6,800 |

### AT+BLESCAN – Start/Stop BLE Scan

|  |  |
| --- | --- |
| **Command** | at+blescan |
| **Response** | Success: OK  Failure: ERROR  Following scan results are sent:  <mac address>:<RSSI>:<address type>:<data len>:<data> |
| **Parameters** | NA |
| **Example** | at+blescan |

### AT+BLECON – Connects to BLE Device

|  |  |
| --- | --- |
| **Command** | at+blecon=<Peer address>,<Peer address type> |
| **Response** | Success: connection id with OK  Failure: ERROR |
| **Parameters** | Peer address: Address of remote BLE device.  Peer address type: Peer address type  0: Public address  1: Random address  **Note**: If BLE connection issues are observed due to noisy environments or extended distance, BLE Tx power can be increased to a maximum of 10dBm. |
| **Example** | at+blecon=00-01-02-03-04-05,0 |

### AT+BLEDISCON – Disconnects the BLE Device

|  |  |
| --- | --- |
| **Command** | at+blediscon=<connection id> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | Connection id: Connection identifier |
| **Example** | at+blediscon=0 |

### AT+BLESERDIS – Discover All Services

|  |  |
| --- | --- |
| **Command** | at+bleserdis=<connection id> |
| **Response** | Success: service information followed by OK  Failure: ERROR  Service information is sent as described:  <start handle>:<end handle>:<len>:<uuid>  **Note**: All parameters are to be sent in hexadecimal format |
| **Parameters** | Connection id: Connection identifier |
| **Example** | at+bleserdis=0 |

### AT+BLECHARDIS – Discover All GATT Characteristic of a Service

|  |  |
| --- | --- |
| **Command** | at+blechardis=<Connection id>, <Start handle>, <End Handle> |
| **Response** | Success: Characteristic information followed by OK  Failure: ERROR  Characteristic information is sent as described:  <handle>:<properties>:<value handle>:length>:<uuid>  **Note**: All parameters are to be sent in hexadecimal format |
| **Parameters** | Connection id: Connection identifier  Start handle: Attribute start handle of the service  End handle: Attribute end handle of the service |
| **Example** | at+blechardis=0,31,ffff |

### AT+BLEDESDIS – Discover All GATT Characteristic Descriptors

|  |  |
| --- | --- |
| **Command** | at+bledesdis=<connection id>, <Start handle>, <End Handle> |
| **Response** | Success: Characteristic descriptor information followed by OK  Failure: ERROR  Characteristic descriptor information is sent as:  <handle>:<len>:<uuid>  **Note**: All parameters are to be sent in hexadecimal format |
| **Parameters** | Connection id: Connection identifier  Start handle: Attribute start handle of the characteristic  End handle: Attribute end handle of the characteristic |
| **Example** | at+bledesdis=0,31,ffff |

### AT+BLECHARRD – Reads GATT Characteristic Value

|  |  |
| --- | --- |
| **Command** | at+blecharrd=<connection id>,<Handle> |
| **Response** | Success: Characteristic value followed by OK.  Failure: ERROR. |
| **Parameters** | Connection id: Connection identifier  Handle: Attribute handle, in hexadecimal format |
| **Example** | at+blecharrd=0,0033 |

### AT+BLECHARWR – Writes GATT Characteristic Value

|  |  |
| --- | --- |
| **Command** | at+blecharwr=<connection id>,<type>,<Handle>, <Length>, <data> |
| **Response** | Success: OK followed by characteristic value  Failure: ERROR |
| **Parameters** | Connection id: Connection identifier  Type: Write type  0: Write  1: Write with response  2: Signed write (currently not supported)  Handle: Attribute handle, in decimal format  Length: Number of bytes to write, in decimal format  Data: Data to write, in ASCII format |
| **Example** | at+blecharwr=0,1,51,1,1 |

### AT+BLESMPCFG – Configure the SMP (Security)

|  |  |
| --- | --- |
| **Command** | at+blesmpcfg=<io cap>,<oob>,<bondable>,<mitm>,<sc>,<keypress>,< key size min>,<encrypt> |
| **Response** | Success: OK followed by characteristic value.  Failure: ERROR. |
| **Parameters** | Iocap: IO capabilities (default: display\_only=0)  oob: OOB data present (default: 0).  bondable: Bondable (1=True, 0=False) (default: 0)  mitm: MITM (Man In The Middle) protection requested (1=True, 0=False) (default: 0)  sc: Secure Connection supported (1=True, 0=False) (default: 0)  keypress: Generate keypress notifications (1=True, 0=False) (default: 0)  key size min: Minimal key size (bytes) that is accepted (7..16) (default: 16)  encrypt: Automatically encrypt link at connection setup if key exists: 1=True, 0=False (default: 0) |
| **Example** | at+blesmpcfg=0,0,1,0,0,0,16,1 |

### AT+BLEAUTH – Configure the SMP (Security)

|  |  |
| --- | --- |
| **Command** | at+bleauth=< Connection id>,<oob>, <bondable>, <mitm>, <sc>, <key> |
| **Response** | Success: OK followed by characteristic value  Failure: ERROR |
| **Parameters** | connection id: Connection identifier.  oob: OOB data present (default: 0)  bondable: Bondable (1=True, 0=False) (default: 0)  mitm: MITM (Man In The Middle) protection requested (1=True, 0=False) (default: 0)  sc: Secure Connection supported (1=True, 0=False) (default: 0)  keypress: 128-bits key required (1=True, 0=False) |
| **Example** | at+bleauth=0,0,0,0,1,128 |

### AT+BLFOTACONF – BLE FOTA Configuration

|  |  |
| --- | --- |
| **Command** | at+blefotaconf=<Service id>,<Characteristics id> |
| **Response** | Success: OK followed by characteristic value  Failure: ERROR |
| **Parameters** | Service id: Service identifier.  Characteristics id: Characteristics identifier |
| **Example** | at+blefotaconf=0x1111,2Af9 |

**Note**: The characteristics id used for FOTA service should not be shared with any other characteristics.

### AT+BTBONDLIST – BLE BOND Information Display

|  |  |
| --- | --- |
| **Command** | at+btbondlist |
| **Response** | Success: Each entry of the bonded information followed by OK. Each entry will be in the format: macaddress,type. Multiple entries are separated by a colon.  Failure: ERROR |
| **Parameters** | None. |
| **Example** | at+blebondlist |

**Note**: The current BLE bond feature supports storage of 10 keys. In case of more keys, it will overwrite the oldest one created so that at any given time, the maximum number of keys stored is 10.

### AT+BTBONDDEL – Delete a BLE BOND Information

|  |  |
| --- | --- |
| **Command** | at+btbonddel=<mac address> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | macaddress: The mac address of the bonded device. The mac address should be provided with colons. |
| **Example** | at+btbonddel=01:02:fe:a2:22:88 |

### AT+PROVSTART – BLE PROVISIONING

|  |  |
| --- | --- |
| **Command** | at+provstart=<0/1> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | 0: after provisioned, the provision info stores in Talaria TWO flash, set the provisioned flag to 1 and system restarts and connect to provided SSID  1: after provisioned, the provision info sends to Host |
| **Example** | at+provstart =1 |

**Note**: part.json file needs to be loaded from the prov example application.

### AT+PROVSTOP – STOP BLE PROVISIONING

|  |  |
| --- | --- |
| **Command** | at+provstop |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | None |
| **Example** | at+provstop |

### AT+BTTXPOWSET – Set BLE TX Power

|  |  |
| --- | --- |
| **Command** | at+bttxpowset=<tx\_power> |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | tx\_power: TX power in dBm (-20 to 10) |
| **Example** | at+bttxpowset=10 |

### AT+BTTXPOWGET – Get BLE TX Power

|  |  |
| --- | --- |
| **Command** | at+bttxpowget |
| **Response** | Success: OK  Failure: ERROR |
| **Parameters** | NA.  Gets the set BLE TX power. |
| **Example** | at+bttxpowget |