# Zongle™

***ZigBee RFD NWK/APS layer firmware for UZBee USB adapter***

## Summary

Zongle allows USB-enabled devices to implement the ZigBee communications protocol for low data-rate wireless mesh networks. It is ideal for OEMs who need to add USB capability to their ZigBee networks. It incorporates an FCC / CE certified IEEE 802.15.4 transceiver, a ZigBee RFD stack and USB interface.

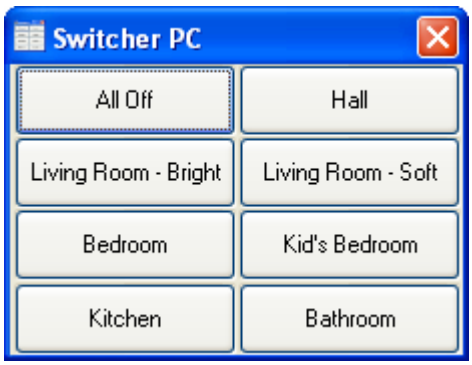
Zongle provides a gateway between ZigBee and Microsoft Windows PC software. As a derivative of the FlexiPanel USBoot family, its firmware may be reprogrammed interchangeably with any other compatible firmware via the USB port.

***Firmware Features:***

- *Incorporates ZigBee RFD stack.*
- *Up to 8 endpoints with up to 8 input and 8 output clusters each.*
- *Fully compatible with free-of-charge Switcher PC software for Windows PCs.*
- *NWK-layer services include:*
  - *Network Discovery*
  - *Joining & Leaving*
  - *Synchronization with parent*
  - *Set & Get attributes*
- *APS-layer services include:*
  - *Binding*
  - *Endpoint data messaging*
- *Device-layer services include:*
  - *ZDO profile ID*
  - *ZDO Endpoint descriptions*
  - *RSSI of last packet*
  - *LQI of last packet*
  - *LED control*
  - *Pushbutton indication*
  - *Error report*
  - *Device information report*
  - *One-time MAC address configuration*

***Hardware Features:***

- *USBoot bootloader allows firmware interchange between:*
  - ***MACdongle™*** *IEEE 802.15.4 MAC layer and sniffer*
  - ***Zongle™*** *ZigBee RFD APS layer*
  - ***StarLite USB™*** *transparent IEEE 802.15.4 communications*
- *2.4GHz IEEE 802.15.4 RF module*
- *FCC / CE / IC compliant*
- *Signature 'G' antenna, free-space range 120m, compact, low 'hand-effect' design*
- *Bind button, status LED*
- *56mm x 20mm x 9mm*




*Zongle operating with Flexipanel's free Switcher PC software*

***Ordering Information***

| ***Table 1. Ordering information*** | |
|---|---|
| ***Part No*** | ***Description*** |
| UZBee | USB IEEE 802.15.4 radio adapter |

Manufactured to ISO9001:2000

# Contents


***Summary*** ..... ***1***

***Contents*** ..... ***2***

***Overview*** ..... ***3***

Stack Profile Implementation 3
Release notes, version 0B400112103520020706 3
Bibliography 3

***Installation Procedures*** ..... ***4***

Service Pack Installation 4
Zongle Installation 4
.inf File Customization 5

***Usage examples*** ..... ***6***

Presence Detection 6
Setting the MAC Address 6
Setting ZigBee Profile and Endpoints 6
Joining a New Network 7
Rejoining a Network 8
Binding 8
Sending Endpoint Data 9
Receiving Endpoint Data 9

***Message Reference*** ..... ***10***

Format 10
Device Messages 10
*Error indication (DERI)* *10*
*Enquire MAC Address request (DMCR)* *10*
*Enquire MAC Address confirm (DMCC)* *11*
*Set MAC Address request (DSMR)* *11*
*Set MAC Address confirm (DSMC)* *11*
*Version request (DVRR)* *11*
*Version confirm (DVRC)* *11*
*LED Display request (DLDR)* *12*
*LED Display confirm (DLDC)* *12*
*Pushbutton indication (DPBI)* *12*
*Device Get request (DGTR)* *12*
*Device Get confirm (DGTC)* *12*
*Device Set request (DSTR)* *13*
*Device Set confirm (DSTC)* *14*
*Device Set request (DRSR)* *14*
*Device Reset confirm (DRSC)* *14*
NWK Messages 15
*NLME-NETWORK-DISCOVERY.request (NNDR)* *15*
*NLME-NETWORK-DISCOVERY.confirm (NNDC)* *15*
*NLME-JOIN.request (NJNR)* *16*
*NLME-JOIN.confirm (NJNC)* *16*
*NLME-LEAVE.request (NLVR)* *17*
*NLME-LEAVE.indication (NLVI)* *17*
*NLME-LEAVE.confirm (NLVC)* *17*
*NLME-SYNC.request (NSYR)* *17*
*NLME-SYNC.confirm (NSYC)* *17*
APS Messages 18
*APSDE-DATA.request (ADAR)* *18*
*APSDE-DATA.confirm (ADAC)* *18*
*APSDE-DATA.indication (ADAI)* *19*
*APSME-BIND.request (ABDR)* *19*
*APSME-BIND.confirm (ABDC)* *19*
*APSME-UNBIND.request (AUBR)* *20*
*APSME-UNBIND.confirm (AUBC)* *20*

***Reference*** ..... ***21***

*Radio Frequency* *21*
*Electrical* *21*
*Mechanical* *21*
*Regulatory* *21*

***Contact Information*** ..... ***21***

# Overview

Zongle is a 2.4GHz IEEE 802.15.4 transceiver with ZigBee RFD stack and a USB interface for connection to PCs. It is one of several firmware applications for FlexiPanel's UZBee USB adapter.

The IEEE 804.15.4 protocol provides services for transceiver devices to discover each other and then exchange packets of data in a reliable, error-free manner. The ZigBee layer allows multi-hop communications across mesh networks. Due to device limitations, only an End Device is implemented, which is a Reduced Function Device. It is capable of participating in a ZigBee network but it cannot forward messages on behalf of other devices.

This data sheet does not attempt to comprehensively explain the ZigBee communications protocol. You will need to have to hand the ZigBee Specification for any significant applications development. For further information, please read the specification documentation available from *www.zigbee.org*.

The ZigBee specification uses the IEEE 802.15.4 MAC layer for single-hop communications. If multi-hop mesh messaging is not required, a MAC-layer-only solution called MACdongle is also available for the same UZBee device.

### *Stack Profile Implementation*

ZigBee devices can only network together if their stack profile parameters match. Unless otherwise requested, the stack profile implemented is Profile Identifier 0x1, as used by HC-L applications. Refer to the ZigBee Specification section E.3 for further details.

### *Release notes, version 0B400112103521150507*

In this release, security is not supported. Changing *RxOnWhenIdle* from its default value of *true* is not supported due to lack of ROM. Changing *TxPower* from its default value of *0xFF* (full power) is not supported due to lack of ROM.

### *Bibliography*

***IEEE 802.15.4 specification***, downloadable from *www.ieee.org*.

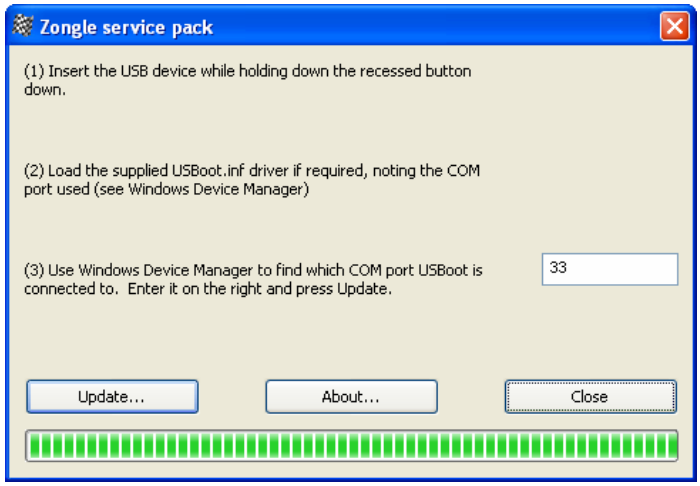
***ZigBee for Applications Developers***, white paper downloadable from *www.flexipanel.com*.

***ZigBee Specification***, downloadable from *www.zigbee.org*.

# Installation Procedures

### *Service Pack Installation*

Zongle is firmware for the UZBee USB IEEE 802.15.4 transceiver. This section explains how to load the Zongle firmware onto UZBee using FlexiPanel's *USBoot* bootloading procedure. You can skip the *Service Pack Installation* section if you know Zongle is already loaded. To install Zongle:




1. Insert the UZBee into a USB port while holding the recessed pushbutton down. The LED should flash. If this is the first time you have run any USBoot product, you will be asked for the `USBoot.inf` driver information file, which may be downloaded from *www.flexipanel.com.*

2. Download `ZongleServicePack.exe` from *www.flexipanel.com*. Run the application.

3. To determine which COM port number has been assigned to USBoot:

    a. Click on *Start*
    b. Right click on *My Computer*, select *Properties*
    c. Click on the *Hardware* tab, press the *Device Manager* button.
    d. Open the *Ports (COM & LPT)* section. You will see the device name *USBoot* and the COM port number.

4. Enter the COM port number in the box provided.

5. Press the Update button. The firmware will load automatically and then start running as if you had just inserted Zongle in a USB slot.

Note that when installed in this way, the MAC address is not preloaded and you may have to specify it using the ***+DSMR*** command. (Application software such as Switcher PC should do this for you.) If you use service pack when you load Zongle into OEM products for distribution to your customers, it is your responsibility to set the MAC address on behalf your customers. Contact us to obtain an allocation of addresses if you wish to have a contiguous block.

### *Zongle Installation*

If this is the first time you have run Zongle, you will be asked for the `Zongle.inf` driver information file, which may be downloaded from *www.flexipanel.com.* The file will be required by users of your products, so it should be packaged with the product or be made readily available on a web site.

When the USB device is first plugged a new USB port, Windows will request the `.inf` file. A dialog box will note that the driver is not certified, to which you should select *Continue Anyway*. The USB device will be assigned an unused COM port number. To determine which number has been assigned:

1. Click on *Start*
2. Right click on *My Computer*, select *Properties*
3. Click on the *Hardware* tab, press the *Device Manager* button.
4. Open the *Ports (COM & LPT)* section. You will see the device name Zongle and the COM port number.

Your users will also need these instructions.

If you plug the Zongle into a different socket, the driver may need to reinstall itself a second time, but this time the computer should find `.inf` file by itself. A different COM port will be assigned, but otherwise the Zongle will function as normal.

Zongle should be plugged in before the application attempts to open the COM port. Failure to do so may require the application to be closed and Zongle to be removed.

### *.inf File Customization*

Zongle uses the standard CDC Windows drivers for data transfer. During installation, only a `.inf` driver information file is needed. This is called `Zongle.inf`, and can be downloaded from *www.flexipanel.com*.

If you wish to develop your own-brand applications for Zongle, the `.inf` driver information file may be customized. It is a text file that may be modified in Windows Notepad as follows. (These steps are optional – you may distribute the `Zongle.inf` file as-is.)

- Modify MFGNAME and DESCRIPTION to suit your application.
- The filename may be modified but the `.inf` suffix should remain.

# Usage examples

The following examples show how Zongle can be used. To test these functions, use the HyperTerminal terminal emulator available in Microsoft Windows to communicate with Zongle.

Application software interfacing with Zongle should treat it as a COM port. Since data flow is asynchronous and unpredictable, data transmission should not block data reception and vice versa. For Microsoft Windows programming, for example, it is vital that overlapped file I/O be used.

### *Presence Detection*

The presence of Zongle can be checked by issuing a version number request:

| | |
|---|---|
| **+DVRR** | (request version) |
| ***+DVRC=0B400112103521200906*** | (confirmation – version number may vary) |

### *Setting the MAC Address*

If presence detection generates an error indicating the MAC address is not set, the MAC address should be set.

| | |
|---|---|
| **+DVRR** | (request version or any command) |
| ***+DERI=04*** | (confirmation – error, MAC address not set) |
| **+DSMR=0500004138C81500** | (request to set MAC address) |
| ***+DSMC*** | (confirmation – confirms MAC address set) |

Note that the MAC address is specified Little-Endian, i.e. the MAC address specified in the example would normally be written as ***0015C83841000005***.

### *Setting ZigBee Profile and Endpoints*

Before you can join a network you should specify the ZigBee profile and endpoints you will be providing. The profile ID is a word value, for example 0x0100 for the HC-L switching profile. The following example sets the profile ID:

| | |
|---|---|
| **+DSTR=040000** | (request Profile ID 0000) |
| ***+DSTC=0004*** | (confirmation – profile ID set) |

Up to 8 endpoints may be specified with up to 8 input clusters and 8 output clusters each. The lowest valued endpoints should be specified only. For example, if 2 endpoints are specified, they should be *EndpointDescriptor0* and *EndpointDescriptor1*. The following example specifies a single Switch Remote Control (Device ID FFFE) On/Off output cluster (value 13), with endpoint number 11, for *EndpointDescriptor0.* The Stack Profile ID value should match the stack profile of the firmware. (Unless otherwise specified, this will be 1.) The application device version number should match the profile definition (0 in this case). This example comes from our Switcher PC switch remote control device.

**+DSTR=81110000FEFF0100000000000000000011300000000000000**
(Set endpoint)

***+DSTC=0081*** (confirmation – endpoint set)

The following example specifies a complementary On/Off input cluster for a Switch Load Controller (Device ID FFFB), with endpoint number 21, for *EndpointDescriptor0.* The Stack Profile ID value should match the stack profile of the firmware. (Unless otherwise specified, this will be 1.) The application device version number should match the profile definition (0 in this case).

**+DSTR=812110000FBFF0101130000000000000000000000000000000**
(Set endpoint)

***+DSTC=0081*** (confirmation – endpoint set)

### *Joining a New Network*

To join a network for the first time, you must first discover it. Then you can request to join it.

Network discovery scans for devices on different channels to determine which networks are operating on which frequencies:

**+NNDR=00000000000000800F8FF07** (request network discovery scan)

***+NNDC=EA000000*** (confirmation – no networks found)

***+NNDC=0001410694090C11FF000000***
(confirmation – 1 network found, PAN 0994, channel 0C)

To join the network, issue a join request specifying the PAN ID to join and with the 'rejoin' parameter set to false. If you set 'Rx is on when idle' as true, the network will assume it can send you messages at any time. If set to false, the network will wait for you to send a NLME-SYNC.request (NSYR) to receive messages.

Example:

**+NJNR=0000000100000100000000000000000000000000000000B000**
(request join new PAN 00B0)

***+NJNC=C30000010000010800F8FF0700000000000000000000000B000***
(confirmation – network is not permitting you to join)

or:

***+NJNC=00005C06B00074795000000000C8150006000041 38C81500B000***
(confirmation – you have joined with short address 7974)

Note that to join a network, the network must permit joining during both network discovery and joining.

### *Rejoining a Network*

Once you have joined a network, you should request to rejoin it rather than joining anew. The main differences are:

- A network discovery scan is not required.
- A PAN ID is not specified. The operating frequency and PAN ID may have changed; you will be told what the new PAN ID is.
- The network does not need to be permit joining in order to rejoin.

To join the network, issue a join request with the 'rejoin' parameter set to true. You must also specify which channels are to be checked for the presence of the parent.

Example:

**+NJNR=0100000000000000F8FF0700000000000000000000000000**
(request rejoin)

***+NJNC=0000000000000303FFE7FF0700C8150006000041 38C81500B000***
(confirmation – you have rejoined and the PAN ID is 00B0)

### *Binding*

Once you have joined or rejoined a network, you can bind endpoints to endpoints on other devices. A bind request is sent as an *END_DEVICE_BIND_request* message to the ZDO endpoint on the coordinator. (Refer to the ZigBee Specification section 1.4.3.2.1.)

Example, binding a single output cluster 13 on endpoint number 11, using transaction ID 55:

**+ADAR=0B0000000000000010A0100000000000000000000000000000020…**
(APSDE-DATA.request header)

**…215508000011000000113** (*END_DEVICE_BIND_request* payload)

***+ADAC=000001016E1802017E790000000000000000000000000000***
(APSDE-DATA.confirm)

***+ADAI=04005A0600000100000000000000000076790000000000000A0…***
(APSDE-DATA.indication header)

***…2100010000*** (*END_DEVICE_BIND_response* payload – success)

Another example, binding a single input cluster 13 on endpoint number 21, using transaction ID 57:

**+ADAR=0B0000000000000010A0100000000000000000000000000000020…**
(APSDE-DATA.request header)

**…215708000021000011300** (*END_DEVICE_BIND_request* payload)

***+ADAC=00000101851C02017F790000000000000000000000000000***
(APSDE-DATA.confirm)

***+ADAI=04005A0600000100000000000000007F790000000000000A0…***
(APSDE-DATA.indication header)

***…2101010000*** (*END_DEVICE_BIND_response* payload – success)

### *Sending Endpoint Data*

Once you have bound an endpoint with an output cluster, you can send data to its corresponding endpoint.

The following example sends a toggle (F0) for a single On/Off output cluster 13 from endpoint number 11, using transaction ID 56. Note that addressing is indirect (i.e. the destination is stored in the binding table). Thus the destination addressing mode is "Address not present" (00) and the destination endpoint is not specified. Refer to the ZigBee specification section 1.3.4 for details of the payload format.

**+ADAR=06000000000100000A0100000000000000000000000000110013…**
(APSDE-DATA.request header)

**…1156110000F0** (*OnOffSRC_CLUSTER* KVP payload)

***+ADAC=00000101641D020080790000000000000000000000000000***
(APSDE-DATA.confirm header)

### *Receiving Endpoint Data*

Once you have bound an endpoint with an input cluster, you might receive data for its corresponding endpoint at any time as an ADAI indication. The following example shows a toggle message received for endpoint 21. Refer to the ZigBee specification section 1.3.4 for details of the payload format.

***+ADAI=06005906000100000000000000000007F790000000000002113…***
(APSDE-DATA.request header)

***…110311000000F0*** (*OnOffSRC_CLUSTER* KVP payload)

# Message Reference

## *Format*

Messages sent to and received from Zongle take the form of ASCII message strings with the following general format:

***+XXXX=hhhhhhhh*<CR><LF>**

All messages begin with the **+** character followed by the four-letter command or response code ***XXXX***, followed by the **=** character. If any additional data accompanies the message, it usually follows as a series of bytes each represented two hexadecimal digits ***hh***. Multi-byte integers are parsed *little-Endian*, i.e. least significant byte first. If no additional data accompanies the message, the **=** character may be omitted. Finally, the string is terminated with a carriage return character **<CR>** and optionally a linefeed **<LF>** character. Extra **<CR>** and/or **<LF>** characters are permitted between messages. Inline editing (*e.g.* pressing backspace) is not supported.

Do not send a message until processing of the previous message is complete. After processing a command, Zongle ignores all characters until a **+** character is received. To abandon after starting it by entering **+**, press ***Z*** until a ***DERI*** syntax error is generated.

## *Device Messages*

Messages starting with a ***D*** character relate to device settings and values.

### Error indication (DERI)

DERI indicates a device-level error occurred. Example:

***+DERI=01***

The following error values may be reported:

| Value | Interpretation |
|---|---|
| 01 | Syntax error – message not recognized |
| 02 | Syntax error – parameters syntax invalid |
| 03 | Syntax error – parameters count invalid |
| 04 | MAC address not valid – send **+DRMC=** message first |
| 05 | MAC address has already been set, cannot be changed |
| 06 | Message not currently supported |

### Enquire MAC Address request (DMCR)

DMCR enquires the MAC address of Zongle. Zongle will generate a DMCC confirmation containing the MAC address. Example:

***+DMCR***

#### Enquire MAC Address confirm (DMCC)

DMCC confirms the 8-byte MAC address of the Zongle. Example:

***+DMCC*=0500004138C81500**

Note that the MAC address is specified Little-Endian, i.e. the MAC address specified in the example would normally be written as ***0015C83841000005***.

#### Set MAC Address request (DSMR)

DSMR specifies a MAC address for Zongle. This command may only be completed once, and since it will normally be set for you, this command will usually generate an error. If you use a bootloader to program the Zongle firmware, you should first identify and note down the current the MAC address and then re-set it using this command. You may only use the MAC address which the hardware has been allocated. For R&D purposes, MAC addresses in the range `0015C83841000000` to `0015C8384100FFFF` may be specified.

Zongle will generate a DSMC confirmation in response. Example:

***+DSMR*=0500004138C81500**

Note that the MAC address is specified Little-Endian, i.e. the MAC address specified in the example would normally be written as ***0015C83841000005***.

#### Set MAC Address confirm (DSMC)

DSMC confirms that a DSMR request completed without error. It has no arguments. Example:

***+DSMC***

#### Version request (DVRR)

DVRR requests firmware version information from Zongle. Zongle will generate a DVRC confirmation in response. Example:

***+DVRR***

#### Version confirm (DVRC)

The DVRC confirmation is generated by Zongle in response to a DVRR request. Example:

***+DVRC=0B400112103521200906***

The response takes the form ***+DMCC=UUUUPPPPVVVVVVDDMMYY***, where the digits are as follows:

| | |
|---|---|
| ***UUUU*** | USB Vendor ID (0B40 always) |
| ***PPPP*** | USB Product ID (0112 indicates Zongle firmware is loaded) |

***VVVVVV*** Zongle version number
***DDMMYY*** Firmware release date

#### LED Display request (DLDR)

DLDR sets the state of the Zongle LED. Zongle will generate a DLDC confirmation in response. Examples:

***+DLDR=00*** (LED off – initial state)

***+DLDR=01*** (LED on)

#### LED Display confirm (DLDC)

DLDC confirms that a DLDR request completed without error. It has no arguments. Example:

***+DLDC***

#### Pushbutton indication (DPBI)

DPBI indicates a change of state of the pushbutton. It has a one byte argument, which is nonzero if the button was pressed and zero if the button was released. Examples:

***+DPBI=01*** (Pushbutton was pressed)

***+DPBI=00*** (Pushbutton was released)

#### Device Get request (DGTR)

DGTR requests device-level attribute data. Refer to Device Set request for a list of attributes.

| **Command format** | | ***+DGTR=****{1 byte}* |
|---|---|---|
| *AttrID* | 1 byte | Attribute requested |

#### Device Get confirm (DGTC)

DGTC confirms device-level attribute data.

| **Command format** | | ***+DGTC=****{varies according to attribute}* |
|---|---|---|
| *status* | 1 byte | Nonzero if not supported |
| *AttrID* | 1 byte | Attribute (see section **DSTR**) |
| *AttributeValue* | *(see* **DSTR***)* | Attribute value |

**Device Set request (DSTR)**

DSTR requests to set device-level attribute data.

| **Command format** | | ***+DSTR=****{length varies with attribute}* |
|---|---|---|
| *AttrID* | 1 byte | Attribute requested – see below |
| *AttributeValue* | *(see below)* | Attribute value |

The following attributes are implemented. Those values which may be set should not be changed once the device has joined the network. They are non-volatile; however, repeatedly setting them to the same values will not exhaust the Flash memory.

| **Attribute** | **Bytes** | **AttrID** | **Settable?** |
|---|---|---|---|
| *RSSI* | 1 byte | 0x01 | No |
| *LQI* | 1 byte | 0x02 | No |
| *RxOnWhenIdle* | 1 byte | 0x03 | Yes ‡§ |
| *ProfileID* | 2 bytes | 0x04 | Yes ‡ |
| *TransmitPower* | 1 byte† | 0x05 | Reset required †‡§ |
| *EndpointDescriptor0* | 1 byte | 0x81 | Yes ‡ |
| *EndpointDescriptor1* | 1 byte | 0x82 | Yes ‡ |
| *EndpointDescriptor2* | 1 byte | 0x83 | Yes ‡ |
| *EndpointDescriptor3* | 1 byte | 0x84 | Yes ‡ |
| *EndpointDescriptor4* | 1 byte | 0x85 | Yes ‡ |
| *EndpointDescriptor5* | 1 byte | 0x86 | Yes ‡ |
| *EndpointDescriptor6* | 1 byte | 0x87 | Yes ‡ |
| *EndpointDescriptor7* | 1 byte | 0x88 | Yes ‡ |

† Format as specified in CC2420 PA_LEVEL specification, e.g. FF = 0dBm, EF = -7dBm, etc.
‡ Set operation supported only
§ Temporarily disabled due to lack of ROM available

***RSSI*** Signal strength of last packet received. The value should be interpreted as a signed byte. Subtract 45 to obtain the signal strength in dBm.

***LQI*** Link quality of last packet received. The most significant bit should be ignored, then subtract 10 to obtain the link quality as a percentage varying from approximately 40% (worst receivable) to 100%.

***RxOnWhenIdle*** If nonzero, the stack will tell other devices that they may send messages to it at any time without waiting for sync requests. The value specified is used in ZDO power descriptor requests. It should match the value specified when joining.

***ProfileID*** Application profile ID supported.

***EndpointDescriptor0 - EndpointDescriptor7*** Up to 8 endpoint descriptors may be specified, each with up to 8 input and 8 output clusters. Each endpoint descriptor comprises the following information (24 bytes total):

| Name | Bytes | Description |
|---|---|---|
| *Endpoint* | 1 byte | Endpoint number |
| *AppProfId* | 2 bytes | Application Profile ID |
| *AppDevId* | 2 bytes | Application Device ID |
| *Flags* | 1 byte | Bits 0-3: Application device version number<br>Bits 4-7: Profile flags |
| *AppInClusterCount* | 1 byte | Number of input clusters (up to 8) |
| *AppInClusterList* | 8 bytes | Input cluster list (set values beyond *AppInClusterCount* to zero). |
| *AppOutClusterCount* | 1 byte | Number of output clusters (up to 8) |
| *AppOutClusterList* | 8 bytes | Output cluster list (set values beyond *AppOutClusterCount* to zero). |

If an endpoint is unused, assign it an unallocated endpoint number and set *AppInClusterCount* and *AppOutClusterList* to zero.

#### Device Set confirm (DSTC)

DSTC confirms device-level attribute data setting.

| **Command format** | | ***+DSTC=****{2 bytes}* |
|---|---|---|
| *status* | 1 byte | Result as enumeration |
| *AttrID* | 1 byte | Attribute |

#### Device Reset request (DRSR)

DRSR requests that the ZigBee stack be reinitialized. Example:

***+DRSR***

#### Device Reset confirm (DRSC)

DRSC confirms the ZigBee stack has been reinitialized. Example:

***+DRSC***

### *MailBox Messages*

Messages starting with an ***N*** character relate to the NWK layer.

#### NLME-NETWORK-DISCOVERY.request (NNDR)

NNDR requests that a scan be performed to discover networks operating within range.

| **Command format** | | ***+NNDR=****{12 bytes}* |
|---|---|---|
| *Filler1* | 7 byte | *(fill with 00)* |
| *ScanDuration* | 1 byte | Scan duration |
| *ScanChannels* | 4 bytes | Scan channels |
| | | *ZigBee 1.0 specification 2.3.2.1* |

#### NLME-NETWORK-DISCOVERY.confirm (NNDC)

NNDC responds to an NLME-NETWORK-DISCOVERY.request.

| **Command format** | | ***+NNDC=****{2 + 8 × NetworkCount bytes}* |
|---|---|---|
| *status* | 1 byte | Result as enumeration |
| *NetworkCount* | 1 byte | Network count |
| *Filler1* | 2 bytes | *(ignore)* |
| *NetDescrList* | 8 × *NetworkCount* | Network descriptor list |
| | | *ZigBee 1.0 specification 2.3.2.2* |

The Network Descriptor List elements have the following format:

| **Networn Descriptor List** | | *{8 bytes}* |
|---|---|---|
| *PAN ID* | 2 bytes | PAN ID |
| *LogicalChannel* | 1 byte | Logical channel |
| *Profile / version* | 1 byte | Bit 0-3: Stack profile<br>Bit 4-7: ZigBee version |
| *Beacon / superframe order* | 1 byte | Bit 0-3: Beacon order<br>Bit 4-7: Superframe order |
| *Flags* | 1 byte | Bit 0: Joining permitted |
| *Filler1* | 2 bytes | *(ignore)* |

#### NLME-JOIN.request (NJNR)

NJNR requests that this device attempt to join the network specified.

| **Command format** | | ***+NJNR=****{26 bytes}* |
|---|---|---|
| *RejoinNetwork* | 1 byte | Nonzero if rejoining |
| *MACSecurity* | 1 byte | Nonzero if MAC security enabled |
| *JoinAsRouter* | 1 byte | Nonzero if joining as router |
| *PowerSource* | 1 byte | `0x00` if battery powered<br>`0x01` if mains powered |
| *Filler1* | 2 bytes | *(fill with 00)* |
| *RxOnWhenIdle* | 1 byte | `0x00` if receiver is off when idle†<br>`0x01` if receiver is on when idle |
| *ScanDuration* | 1 byte | Scan duration |
| *ScanChannels* | 4 bytes | Scan channels |
| *Filler2* | 12 bytes | *(fill with 00)* |
| *PAN ID* | 2 bytes | PAN ID |
| † *Receiver is on when idle. However,* off *may be specified, e.g. if the PC is not always on and/or present* | | |
| | | *ZigBee 1.0 specification 2.3.6.1* |

#### NLME-JOIN.confirm (NJNC)

NJNC responds to an NLME-JOIN.request, indicating whether or not the attempt to join was successful.

| **Command format** | | ***+NJNC=****{26 bytes}* |
|---|---|---|
| *status* | 1 byte | Result as enumeration |
| *Filler1* | 23 bytes | *(ignore)* |
| *PAN ID* | 2 bytes | PAN ID |
| | | *ZigBee 1.0 specification 2.3.6.3* |

#### NLME-LEAVE.request (NLVR)

NLVR requests that a device should leave the network.

| **Command format** | | ***+NLVR=****{24 bytes}* |
|---|---|---|
| *RemoveChildren* | 1 byte | Remove Children |
| *Filler1* | 15 bytes | *(ignore)* |
| *DeviceAddress* | 8 bytes | Long address of device leaving |
| | | *ZigBee 1.0 specification 2.3.8.1* |

#### NLME-LEAVE.indication (NLVI)

NLVI indicates that a device has been removed from the network by its parent.

| **Command format** | | ***+NLVI=****{24 bytes}* |
|---|---|---|
| *status* | 1 byte | Result as enumeration |
| *Filler1* | 15 bytes | *(ignore)* |
| *DeviceAddress* | 8 bytes | Long address of device leaving |
| | | *ZigBee 1.0 specification 2.3.8.2* |

#### NLME-LEAVE.confirm (NLVC)

NLVC responds to an NLME-LEAVE.request, indicating whether or not the request to leave was successful.

| **Command format** | | ***+NLVC=****{x bytes}* |
|---|---|---|
| *status* | 1 byte | Result as enumeration |
| *Filler1* | 15 bytes | *(ignore)* |
| *DeviceAddress* | 8 bytes | Long address of device leaving |
| | | *ZigBee 1.0 specification 2.3.8.3* |

#### NLME-SYNC.request (NSYR)

NSYR requests to synchronize and/or extract data from a coordinator or router.

| **Command format** | | ***+NSYR=****{1 byte}* |
|---|---|---|
| *Track* | 1 byte | Track |
| | | *ZigBee 1.0 specification 2.3.10.1* |

#### NLME-SYNC.confirm (NSYC)

NSYC responds to an NLME-SYNC.request, indicating whether or not the request was successful.

| **Command format** | | ***+NSYC=****{1 byte}* |
|---|---|---|
| *status* | 1 byte | Result as enumeration |
| | | *ZigBee 1.0 specification 2.3.10.3* |

### *APS Messages*

Messages starting with an ***A*** character relate to the APS layer.

#### APSDE-DATA.request (ADAR)

ADAR requests a data packet be transmitted.

| **Command format** | | ***+ADAR=****{asduLength + 27 bytes}* |
|---|---|---|
| *asduLength* | 1 byte | Length of payload *asdu* to follow |
| *Filler1* | 3 bytes | *(fill with 00)* |
| *Profile ID* | 2 bytes | Profile ID |
| *Filler2* | 1 byte | *(fill with 00)* |
| *DstAddrMode* | 1 byte | Destination addressing mode |
| *RadiusCounter* | 1 byte | Radius Counter |
| *DiscoverRoute* | 1 byte | Route discovery options |
| *TxOptions* | 1 byte | Bit 0: securityEnabled<br>Bit 1: useNWKKey<br>Bit 2: acknowledged |
| *Filler3* | 5 bytes | *(fill with 00)* |
| *DstAddr* | 8 bytes | Destination address. (If *DstAddrMode* specifies short addresses, ignore last 6 bytes.) |
| *SrcEndpoint* | 1 byte | Source endpoint |
| *DstEndpoint* | 1 byte | Destination endpoint |
| *ClusterID* | 1 byte | Cluster ID |
| *asdu* | *asduLength* bytes | Payload to be transmitted |
| | | *ZigBee 1.0 specification 1.2.4.1.1* |

#### APSDE-DATA.confirm (ADAC)

ADAC responds to an APSDE-DATA.request.

| **Command format** | | ***+ADAC=****{24 bytes}* |
|---|---|---|
| *status* | 1 byte | Result as enumeration |
| *Filler1* | 6 bytes | *(ignore)* |
| *DstAddrMode* | 1 byte | Destination addressing mode |
| *Filler2* | 8 bytes | *(ignore)* |
| *DstAddr* | 8 bytes | Destination address. (If *DstAddrMode* specifies short addresses, ignore last 6 bytes.) |
| | | *ZigBee 1.0 specification 1.2.4.1.2* |

#### APSDE-DATA.indication (ADAI)

ADAI indicates a data packet was received.

| **Command format** | | ***+ADAI=****{27 + asduLength bytes}* |
|---|---|---|
| *asduLength* | 1 byte | Length of payload *asdu* |
| *SecurityStatus* | 1 byte | Security status |
| *Filler1* | 2 bytes | *(ignore)* |
| *Profile ID* | 2 bytes | Profile ID |
| *SrcAddrMode* | 1 byte | Source addressing mode |
| *WasBroadcast* | 1 byte | Was broadcast |
| *SrcAddr* | 8 bytes | Source address. (If *SrcAddrMode* specifies short addresses, ignore last 6 bytes.) |
| *Filler2* | 8 bytes | *(ignore)* |
| *SrcEndpoint* | 1 byte | Source endpoint |
| *DstEndpoint* | 1 byte | Destination endpoint |
| *ClusterID* | 1 byte | Cluster ID |
| *LinkQuality* | 1 byte | RSSI in dBm as returned by CC2420 chip. (Not part of ZigBee standard.) Added in version 103521. |
| *asdu* | *asduLength* bytes | Payload received |
| | | *ZigBee 1.0 specification 1.2.4.1.3* |

#### APSME-BIND.request (ABDR)

ABDR requests that two endpoints are bound.

| **Command format** | | ***+ABDR=****{27 bytes}* |
|---|---|---|
| *Filler1* | 8 bytes | *(fill with 00)* |
| *DstAddr* | 8 bytes | Destination long address. |
| *SrcAddr* | 8 bytes | Source long address. |
| *SrcEndpoint* | 1 byte | Source endpoint |
| *DstEndpoint* | 1 byte | Destination endpoint |
| *ClusterID* | 1 byte | Cluster ID |
| | | *ZigBee 1.0 specification 1.2.4.3.1* |

#### APSME-BIND.confirm (ABDC)

ABDC responds to an APSME-BIND.request.

| **Command format** | | ***+ABDC=****{24 bytes}* |
|---|---|---|
| *status* | 1 byte | Result as enumeration |
| *Filler1* | 7 bytes | *(fill with 00)* |
| *DstAddr* | 8 bytes | Destination long address. |
| *SrcAddr* | 8 bytes | Source long address. |
| *SrcEndpoint* | 1 byte | Source endpoint |
| *DstEndpoint* | 1 byte | Destination endpoint |
| *ClusterID* | 1 byte | Cluster ID |
| | | *ZigBee 1.0 specification 1.2.4.3.2* |

#### APSME-UNBIND.request (AUBR)

AUNR requests that two endpoints are unbound.

| **Command format** | | ***+AUBR=****{27 bytes}* |
|---|---|---|
| *Filler1* | 8 bytes | *(fill with 00)* |
| *DstAddr* | 8 bytes | Destination long address. |
| *SrcAddr* | 8 bytes | Source long address. |
| *SrcEndpoint* | 1 byte | Source endpoint |
| *DstEndpoint* | 1 byte | Destination endpoint |
| *ClusterID* | 1 byte | Cluster ID |
| | | *ZigBee 1.0 specification 1.2.4.3.3* |

#### APSME-UNBIND.confirm (AUBC)

AUBC responds to an APSME-UNBIND.request.

| **Command format** | | ***+AUBC=****{24 bytes}* |
|---|---|---|
| *status* | 1 byte | Result as enumeration |
| *Filler1* | 7 bytes | *(fill with 00)* |
| *DstAddr* | 8 bytes | Destination long address. |
| *SrcAddr* | 8 bytes | Source long address. |
| *SrcEndpoint* | 1 byte | Source endpoint |
| *DstEndpoint* | 1 byte | Destination endpoint |
| *ClusterID* | 1 byte | Cluster ID |
| | | *ZigBee 1.0 specification 1.2.4.3.4* |

# Reference

### Radio Frequency

| | |
|---|---|
| Max RF output power | 1mW = 0dBm |
| RF frequency range | 2400MHz to 2485MHz |
| Communications protocol | IEEE 802.15.4 (DSSS O-QPSK chip encoding) |
| Raw data rate | 250kbit/s |
| RF channels | 16 |
| Free space range with integral antenna | Approx 120m |

### Electrical

| | |
|---|---|
| USB specification version | 2.0 |
| USB driver | `usbser.sys` (Supplied with Microsoft Windows) |
| USB plug | Type A |
| Maximum current draw | 30mA |

### Mechanical

| | |
|---|---|
| Max operating/storage temperature | -40ºC to +85 ºC |
| Dimensions L×W×H mm | 55.7 × 20.3 x 10.5 (note 1) |

1. Excludes USB plug

### Regulatory

| | |
|---|---|
| FCC compliance | G-antenna version compliant, awaiting certificate |
| CE compliance | G-antenna version compliant, awaiting certificate |
| IC (Industry Canada) compliance | G-antenna version compliant, awaiting certificate |

# Contact Information

*FlexiPanel*


Developed by:

FlexiPanel Ltd
2 Marshall St, 3rd Floor,
London W1F 9BB, United Kingdom
*email: support@flexipanel.com*
*www.flexipanel.com*



Manufactured and distributed by:

R F Solutions Ltd
Unit 21, Cliffe Industrial Estate,
Lewes, BN8 6JL, United Kingdom
*email : sales@rfsolutions.co.uk*
*http://www.rfsolutions.co.uk*
*Tel: +44 (0)1273 898 000*