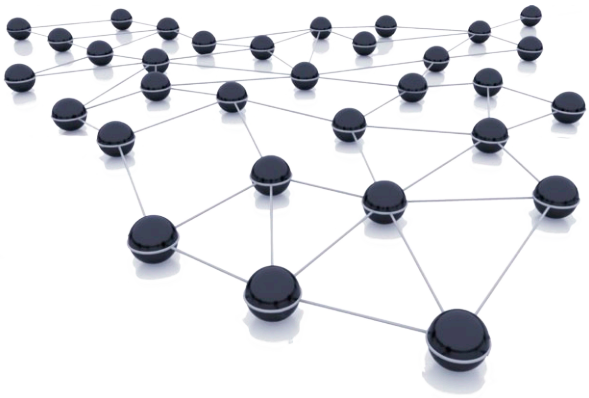
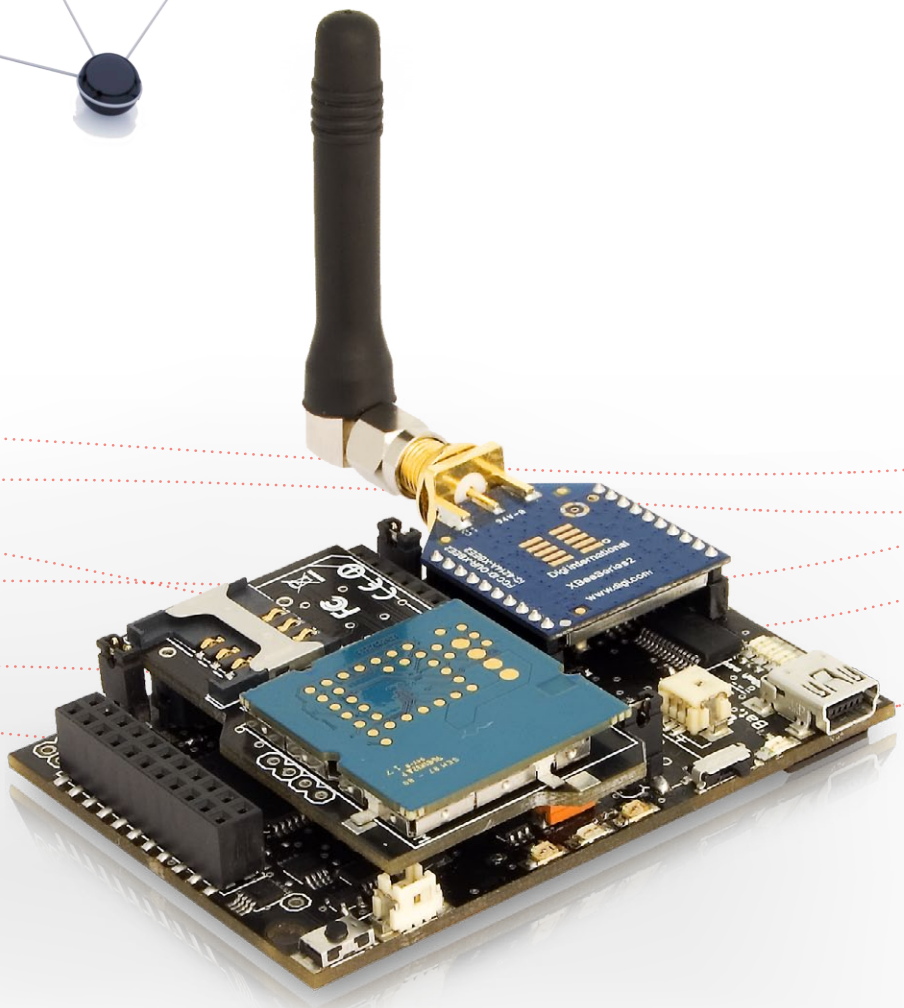
Waspmote Utilities
Programming Guide





libelium

waspmote

Document Version: v0.2 -06/2012

INDEX

1. Introduction

Waspmote provides some libraries to manage USB interaction and other general tasks (blinking LEDs, some string conversions, etc).

To manage these tasks, some functions have been developed and two libraries have been created: 'USB Library' and 'Utils Library'.

2. USB Library

2.1. Waspmote Libraries

2.1.1. Waspmote USB Files

WaspUSB.h ; WaspUSB.cpp

2.1.2. Constructor

To start using Waspmote USB library, an object from class 'WaspUSB' must be created. This object, called 'USB', is created inside the Waspmote USB library and it is public to all libraries. It is used through the guide to show how the Waspmote USB library works.

When creating this constructor, one variable is initialized. This variable specifies the number of the UART that USB is going to use.

2.1.3. Pre-Defined Constants

There are some constants defined in 'WaspUSB .h' related with the different kind of numbers that can be printed on the screen.

2.2. Initialization

Two functions have been created to open and close the UART used to communicate via USB.

Example of use

```
{
  USB.begin(); // Opens the UART at 38400 bps by default
  USB.close(); // Closes the UART
}
```

2.3. Reading data

Two functions have been developed for reading data or checking if data is available in the UART. One more function has been developed to free the UART buffer.

Example of use

```
{
  int data_read=0;
  if(USB.available()){ // If data is available '1' is returned
        data_read=USB.read(); // Reads data from UART0
  }
  USB.flush(); // Frees the UART buffer. All the data unread are lost.
}
```

2.4. Writing data

Some functions have been created to write data to the UART.

Example of use

```
{
 char charac='a';
 USB.print(charac); // Writes the char 'a' to the UART
 USB.println(charac); // Writes the char 'a' to the UART adding an EOL
 USB.println(); // Writes an EOL
 char* string="Hello";
 USB.print(string); // Writes a string to the UART
 USB.println(string); // Writes a string to the UART adding an EOL
 uint8_t unsigned=3;
 USB.print(unsigned); // Writes the number '3' to the UART
 USB.println(unsigned); // Writes the number '3' to the UART adding an EOL
 int integer=54345;
 USB.print(integer); // Writes the number '54345' to the UART
 USB.println(integer); // Writes the number '54345' to the UART adding an EOL
 long long_int=1234567;
 USB.print(long_int); Writes the number '1234567' to the UART
 USB.println(long_int); Writes the number '1234567' to the UART adding an EOL
 long_int= 0x14;
 USB.print(long_int,16); // Writes the number 0x14 in base 16
 USB.println(long_int,16); // Writes the number 0x14 in base 16 adding an EOL
}
```

3. Utilities Library

3.1. Waspmote Libraries

3.1.1. Waspmote USB Files

WaspUtils.h ; WaspUtils.cpp

3.1.2. Constructor

To start using the Waspmote Utilities library, an object from class 'utils' must be created. This object, called 'Utils', is created inside the Waspmote Utilities library and it is public to all libraries. It is used through the guide to show how the Waspmote Utilities library works.

When creating this constructor, no variables are initialized.

3.1.3. Pre-Defined Constants

There are some constants defined in 'utils .h' used to make it easier the understanding of the code.

3.2. Getting Free Memory

It gets the amount of free memory available.
It returns this free memory expressed in bytes and stores it in variable 'freeMemory' .
This function can not be used more than once in the same function, to prevent the code crash.

Example of use

```
{
  int memory=0;
  memory=Utils.getFreeMemory(); // Gets the free memory available
}
```

3.3. Using LEDs

These functions are capable of changing the state of the LEDs. There are one function to change their state, other one to get their state and another to blink LEDs.

Example of use

```
{
  Utils.setLED(LED0,LED_ON); // Sets the LED0 ON
  uint8_t state=Utils.getLED(LED1); // Gets the state of LED1
  Utils.blinkLEDs(1000); // Blink LEDs using a delay of 1000ms for blinking
}
```

3.4. Using EEPROM

These functions manage the writing and reading in EEPROM.

Example of use

```
{
 Utils.writeEEPROM(325,'B'); // Write the value 'B' in the address 325
 uint8_t data=Utils.readEEPROM(325); // Reads the value stored in the address 325
}
```

3.5. GPS String manipulation

There are some functions for managing strings related with GPS sentences. These functions are capable of parsing strings related with GPS sentences.

Example of use

```
{
 long number=0;
 number=Utils.parse_decimal("100.35"); // Parses the string, getting the number 10035
 number=Utils.parse_degrees("00053.175"); // Parses the string, getting out the number
 number=Utils.gpsatol("100.35"); // Parses the string, getting the integer part out
 number=Utils.parse_latitude("00053.175"); // Parses the string, getting the number in rad
}
```

3.6. Numbers Handling

There are some functions to convert numbers.
Example of use

```
{
 uint8_t number=0;
 number=Utils.dec2hex(17); // Converts decimal '17' to hex '11'
 long number2=0;
 number=Utils.array2long("1356"); // Gets the number '1356' out of the string
 char* number3;
 number3=Utils.long2array(1356); // Gets the number '1356' into the string
}
```

3.7. String Handling

There are some functions for handling strings.

Example of use

```
{
 long number=0;
 number=Utils.strtolong("1356"); // Gets the number '1356' out of the string
 number=Utils.sizeOf("Hello!"); // Gets the size of the string
 number=Utils.strCmp("Hello!","Hello!",6); // Compare two strings
 char* destination;
 Utils.strCp("Hello!", destination); // Copy first string into second string
}
```

3.8. General

There are some functions for clearing or filling some 'Utils' variables.
Example of use

```
{
  Utils.clearArguments(); // Clear the variable 'arguments'
  Utils.strExplode("1234,5678",','); // Store in 'arguments' '1234' and '5678'
}
```

4. Analog inputs

4.1. Reading analog inputs

It gets the value read by the corresponding analog input.
It returns a value in the [0-1023] range.

Example of use

```
{
  int val = 0;
  val = analogRead(ANALOG1); // Read the input ANALOG1 and store its value in val
}
```

5. Digital I/O

5.1. Setting pin mode

It configures the specified pin as an input or an output.
It returns nothing.

Example of use

```
{
  pinMode(DIGITAL1, INPUT); // Sets DIGITAL1 as an input
  pinMode(DIGITAL4,OUTPUT); // Sets DIGITAL4 as an output
}
```

5.2. Reading Digital Inputs

It reads the value from the specified digital pin.
It returns '0' or '1'.

Example of use

```
{
  int val=0;
  val=digitalRead(DIGITAL1); // Reads the value from Digital 1
}
```

5.3. Writing Digital Outputs

It writes a 'High' or 'Low' value to a digital pin.
Its voltage will be set to 3.3V for a 'High' value, and 0V for a 'Low' value.

Example of use

```
{
  digitalWrite(DIGITAL4,HIGH); // Writes 'High' to Digital 4
}
```

6. PWM output

There is one pin that can be used as an analog or digital pin.
This pin is called DIGITAL1.
It returns nothing.

Example of use

```
{
  analogWrite(DIGITAL1, 128); // Writes a value comprised between [0-255] in Digital1
}
```

7. Code examples and extended information

For more information about the Waspmote hardware platform go to the Support section:
http://www.libelium.com/support/waspmote

Extended information about the API libraries and complete code examples can be found at:
http://www.libelium.com/development/waspmote