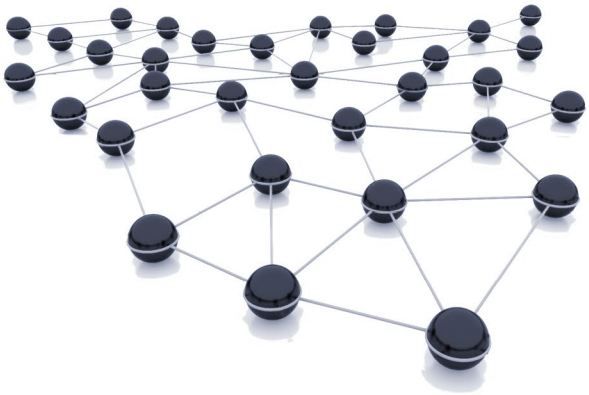
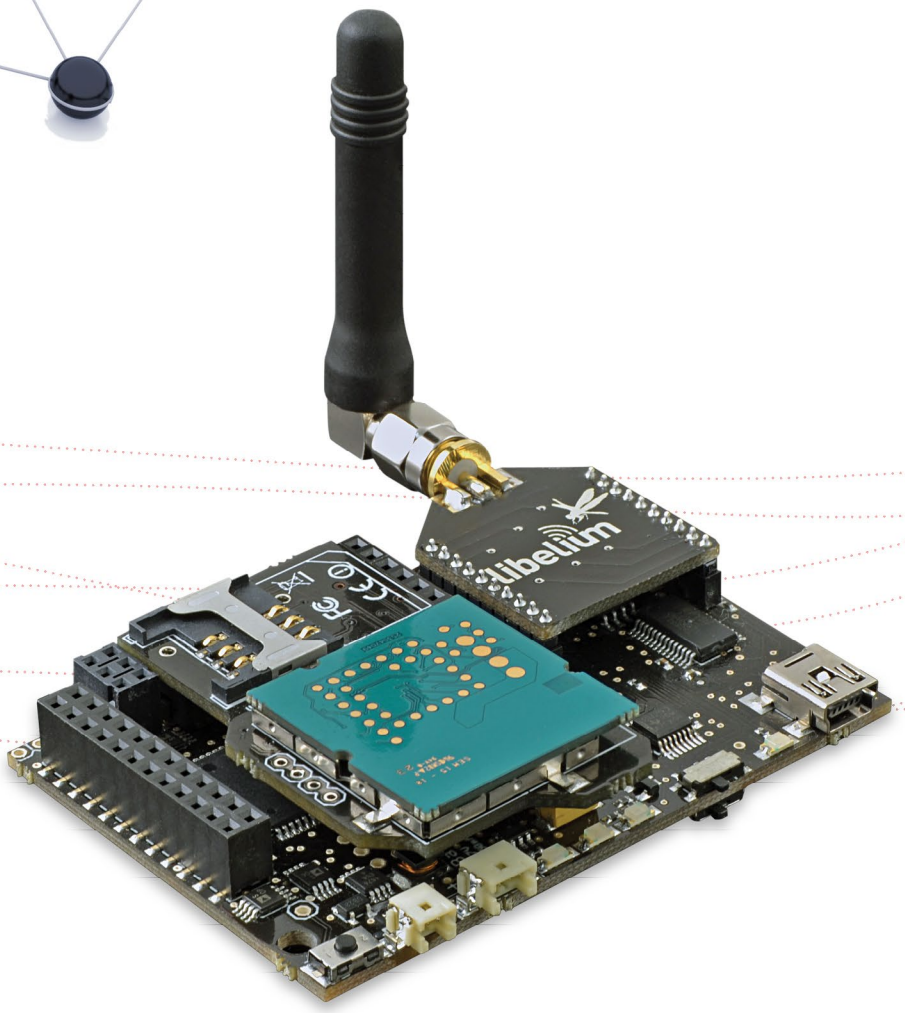
Waspmote
Programming Guide





libelium

waspmote

Document Version: v4.5 - 10/2015

INDEX

1. Introduction

This Guide contains advice to get good, solid and successful codes or projects. They are **practical tips** or small ideas which were not suitable to be in the other Guides.

All the tricks explained here have been learned from the experience. The Waspmote or Waspmote Plug & Sense! developer will find interesting to read this guide to avoid problems **in real life**.

Actually, the best tip is "**Read the Guides**"... this Guide does not replace the careful study of the rest of Guides; it is a practical complement for them. We advise to read this Guide at the same time that the user advances in the development of Waspmote.

2. Program structure

The structure of the ".pde" codes is divided into 2 basic functions: `setup()` and `loop()`. The `setup()` is the first part of the code, which is **only** run once when the code is initialized (or Waspmote is reset). In this part it is recommendable to include the initialization of the modules which are going to be used, as well as the part of the code which is only important when Waspmote is started.

On the other hand, the `loop()` function runs **continuously**, forming an infinite loop. The goal of this function is to measure, send the information and save energy by entering a low consumption state.

The program skeleton:

```
// 1. Include Libraries
// 2. Definitions
// 3. Global variables declaration

void setup()
{
   // 4. Modules initialization
}

void loop()
{
   // 5. Measure
   // 6. Send information
   // 7. Sleep Waspmote
}
```

And just take a look to our hundreds of examples. They try to show a certain order.

3. Libraries includes

The Waspmote API is divided into two folders: "core" and "libraries". The libraries inside "core" are invoked **automatically** so there is no need to add them to the ".pde". Some examples are "Utils.h" or "WaspACC.h". However, it is **mandatory** to manually include to the ".pde" a library which is inside the "libraries" folder (if we need to use it).

The libraries used in the programs must be included at the beginning of the code by writing the inclusion of the corresponding header. The list of classes which have to be included when they are used is:

```
#include <WaspSX1272.h>
```

4. Variable declaration

It is recommended to declare **global variables** before the setup function of the code. For example:

```
// Global variables declaration
char* str = "This is a string";
uint8_t number = 0;
int counter = 0;

void setup()
{
}

void loop()
{
  counter++;
  USB.println(str);
  number = 121;
}
```

- Variable types defined in <stdint.h>:

```
typedef signed char int8_t
typedef unsigned char uint8_t
typedef signed int int16_t
typedef unsigned int uint16_t
typedef signed long int int32_t
typedef unsigned long int uint32_t
typedef signed long long int int64_t
typedef unsigned long long int uint64_t
```

5. Code style

In this section some recommendations are given in order to make a **clear** and easy **readable** code. Try to write beautiful code! You can take a look to all of our examples as a reference. We try to keep some programming rules.

However, each programmer has his own way to write code and could not like some of the given recommendations. The purpose of this section is just to give a guideline to keep things tidy.

Comment your code.

This will help your colleagues to understand what you try to do. Also, it will help the Libelium Technical Service if they needed to analyze the code. It could even help you if you read your own code months after you created it. The recommended styles are:

```
<code line>; // this is a comment; this code line does 'xx'
```

or

```
// this is a comment; this code line does 'xx' :
<code line>;
```

You can even add long comments; they will clear things up:

```
/* now we will perform 'xx' action
   because we need to 'yy'
  in order to get 'zz' done
*/
<code>
```

Use English for comments.. it is the most common language! If you need to share your codes with other developers or clients in foreign countries, it is better to start in English.

Number the parts of your code.

This is basic to keep code organized and improve the legibility. The recommended style is:

```
// 1 - configure module
// 2 - read sensor
// 3 - xxx
```

Variables and constants.

Variables should be lower case (upper case for 2nd, 3rd word). Use meaningful words and try to give a default value:

```
int counter1 = 0;
float newTemperature = 0.0;
```

Constants should be upper case. Use meaningful words:

```
#define TIMEOUT_GPRS 1000
```

Indentation

Default indentation is 2 whitespaces. Use indentation to make code flow more visual, readable.

'if', 'while' and 'for' loops

"Sparse is better than dense". (A) is probably more readable than (B) or (C):

(A)

```
if (<condition>)
{
  <code_line_1>;
}
else
{
  <code_line_2>;
}
```

(B)

```
if (<condition>) {<code_line_1>;}
else {<code_line_2>;}
```

(C)

```
if (<condition>){
  <code_line_1>;
}
else{
  <code_line_2>;
}
```

Always prefer explicit and sparse ways of code writing.

6. General advice

Plan your project and divide it into the **parts** of Waspmote you need to use. With "one part" we mean a feature (RTC, sleep modes,...), a sensor (CO_2, temperature, ...) or a communication module (802.15.4, GPRS, ...).

Always go **step by step**. First, study one feature/sensor/module, read the manual, read the examples, experiment with the examples in the IDE and even play with the Code Generator. Once you understand it, do the same with the next feature/sensor/module you need. Once you understand all the parts and you have working codes for all of them, then try to add one code with another. Once they fit, add the following working code. At the end you will have the final code working, ready to pass the testing phase. Don't try to put all together from the beginning.

"Keep it **simple**". Do you really need OTA operations? Do you really need interruptions? Any feature you are adding to your project will improve it, but bear in mind it will also add complexity.

The **testing phase** is crucial: for debugging tasks, we advise the use of the USB line, and not the communications module.

Place as many **debug** messages as you need. This will help to know which part of the code (or which code line) is an error source. You can see it via the USB line, in the IDE's Serial Monitor. You can use logger programs (CuteCom or Hyperterminal for example) to save the serial monitor output or the packets sent to allow you analyze the information afterwards. You can also use the SD card to store logs, or even the EEPROM for very specific data.

Related to the tip above, you can use the `#ifdef` + `#endif` flow instructions to enable (or not) an advanced and thorough debug mode, as done in some of our libraries (GPRS Pro or 3G).

A **global loop counter** is very useful. It will say how many times the loop has been executed.

Also, a **frame counter** is useful. It will say how many times Waspmote has built a frame (and how many times Waspmote has tried to send a frame). The number of received frames will be obviously equal or lower than the number of sent frames. Due to communication problems, some frames will never arrive to destination, but at least you will know they were sent.

With the global counter and the frame counter you can get the packet loss rate easily and check the health of each link.

Use the `if()` condition to **control the code flow** and make more efficient. For example, you will not want to send an SMS with the GPRS module if the connection to the GPRS network was not possible. You can build nested `if()` structures to do things only when you need to do it. Take a look to the API, many functions deliver 0/1 as an output to show if the execution was successful.

In order to save battery, you may avoid sending frames in certain situations. Anyway, a good code will send frames always after a certain number of loops or after a certain time. This avoids the possibility of never sending frames, which could lead to the doubt "Is this node not working, or it is up but just does not send frames?".

Never use functions or conditions which could be waiting **forever**. For example, any `while()` loop is suitable to be stuck because the exit condition is never satisfied. Instead of putting a single exit condition, add a second one with the OR (`||`) operator. This second condition can be a counter which is incremented in each while loop. This will help to implement a kind of timeout: "if this while loop was executed 100 times, then exit". A second strategy to avoid infinite while loops is to use `break` inside the while loop.

If your code does something that you want to check, it advised to do it **before** the sleep part. This will avoid waiting for the sleep time to expire to see if you get the expected results/action. This means that the sleep mode should be always placed at the end of your code.

We advise to use the RTC in almost any project, it is needed to set timeouts or alarms for the sleep modes. However, we suggest that the RTC time is considered in a **relative** way, and **not absolute**. That is to say, it does not matter if it is 07:27 or 15:28 inside one Waspmote; the important thing is that the sleep alarm will ring within 10 minutes. Time-stamping should be done in reception (Gateway's PC or Meshlium): time will always be more accurate there (internet time). Considering relative time avoids the need of setting the exact time before running a code. Also, bear in mind that if you reset Waspmote and you set the RTC time in the first initialization, then your will lose the exact time.

The `millis()` function measures the **execution time** since Waspmote is turned ON. It can be very useful to measure how much time your functions spend carrying out their tasks (subtracting the value after something with the value before). It can be a sign of malfunction: if a loop normally takes 4 seconds to be executed and now it took 10 seconds... something could go wrong.

You can use the **LEDs** to signal important events in the developing or debugging phase. LED signaling is not recommended for the final code since LEDs have a significant energy consumption.

All the software system is **Open Source**, we are proud of that. The libraries are available for anyone who wants to study them and even modify them for a better performance. However, unless the developer is an expert in Waspmote and C coding, we do **NOT** advise to make changes in the libraries. That practice can lead to unexpected behaviours, data collisions, memory leaks and many other issues. We only recommend to use the official libraries created and verified by Libelium.

As said all along the documentation, please remember to work with the battery always connected and charged. Always.

You should attach the battery to Waspmote with a nylon clamp (and never remove it) to avoid broken battery cables.

7. Flash memory

The microcontroller Flash (128KB) contains both the uploaded program and the bootloader. The bootloader is a small program which is executed at the beginning and proceeds to run the uploaded program. Libelium provides the Waspmote IDE which won't permit rewriting the bootloader.

Do **NOT** use other IDEs, **only** the Waspmote IDE is recommended.

Libelium does not recommend to implement **Watchdog** timers as some users have had some problems and the microcontroller has needed to be re-flashed.

If the bootloader is overwritten by using any of the previous practices, the warranty will be **voided.**

8. EEPROM

To store values permanently when Waspmote is switched off or when it goes to **hibernate** mode, it is necessary to use the microcontroller's EEPROM (4KB) non-volatile memory.

- Addresses from 0 to 1023 are **reserved**. Do not try to write at this addresses. Factory values can be overwritten and your device can become unresponsive.
- Addresses from 1024 to 4095 are available.
- Use the EEPROM to save variables if the program is designed to reset Waspmote or to enter in Hibernate state (in which cases, all variables will be reset too). The use of the EEPROM can help to keep current conditions or current variables even if we **reset** or **hibernate** Waspmote.
- WaspStackEEPROM can use the EEPROM memory as a LIFO stack to allow save frames or other kind or data.

9. RAM

Waspmote's microcontroller includes a 8KB SRAM which is shared between initialized and uninitialized variables, the dynamic memory allocator, and the stack that is used for calling subroutines and storing local variables.

- It is a good practice to delete variables which have already been used. This will avoid confusion, like "is this old or new?".
- Also, once a vector has been used, it is good to delete all the fields in this vector, by writing 0's or voids, for example.
- Avoid using **strings** so as to save memory. When printing debug messages use the Flash memory to allocate those messages:

WRONG: `USB.println("This is a message");`

OK: `USB.println(F("This is a message"));`

- Avoid using **dynamic** memory allocation in order to prevent memory fragmentation. It is preferable to use controlled static variables.
- It is possible to get the free memory by calling the function `freeMemory`:

```
USB.print(F("Free Memory:"));
USB.println(freeMemory());
```

10. Power

Sleep modes should be present in almost any code. They will help to enlarge the battery life. They must be handled with care:

It can be good idea to have 2 types of loops in your code: the normal loop and the "low-cost" loop. This "**low-cost**" loop will be executed if the battery level is detected to be low (you can place a first `if()` in your code to make the code flow as you wish). In this case the battery is low (i.e., <10%), you may want:

- implement just part of the normal loop
- notify of low battery status via a special frame
- implement the normal loop, but setting a longer sleep time at the end
- do nothing but sleep as fast as possible to enable the battery is charged

- As said above, you can have different types of loops depending on the current conditions. Depending on the situation, you perform certain parts of the code, but not others.
- Try to write **fast loops**. The faster the loop, the better energy performance.
- Choose the **duty-cycle** with care. The duty-cycle is the percentage of time a device is on, compared to the total time. It is normally defined by the number of times we want to read sensors each hour. Most of the projects can have their Waspmotes sleeping most of the time. For example, there is no reason to measure temperature every 1 minute: it just does not change in such a quick way! Depending on the parameters you want to control, you will have different duty cycles. Most of our clients measure each 5-10 minutes, which means sleep times of 5-10 minutes and loops of few seconds.
- A good developer will check the **energy consumption** of the written codes (time to go from 100% to 0% battery level or from 80% to 60%, for example). This will help to have a battery life estimation. It could lead to the need of re-doing the code to make it more efficient and have a longer life.
- Remember any battery has a **self-discharge** rate. It can go from 3% to 10% per month in a rechargeable battery.
- Always consider the possibility of adding **external power supply** to your Waspmote (a solar panel for example). The "energy budget" will probably be positive in any project which has external power supply: the energy constraint just disappears.
- Bear in mind that most of the Waspmote power consumption is due to the **communication module** (dozens or hundreds of mA!!). You may find interesting not to send frames in every single loop. You can send "summary frames" each hour, for example, with the latest measured values. It depends on how important it is to the project to have "real time information". Another tip is that you can send one frame only in the case that certain parameter is above a threshold. Or you can send one frame only in the case the value you just read has changed in a significant way compared to the last sample sent.
- Above, we gave tips to avoid sending redundant information and save battery, but anyway any code should send one frame **ALWAYS** after a certain number of loops or after a certain time (hours).
- GPRS and 3G modules are the **most** energy-costly modules; the main reason is because they take several seconds to connect to the cellular network. Bear this fact in mind when using this module.
- It is possible to control the power supply of several modules in Waspmote:
  - XBee
  - Sigfox
  - LoRa
  - Bluetooth_Pro
  - Bluetooth Low Energy
  - GPS
  - GPRS
  - RFID
  - WiFi
  - Sensor Boards
  - RTC
  - SD card
  - 3G

These are some recommendations depending on the module used:

After switching on the XBee modules, it is recommended to **wait** for some seconds in order to establish the network in the case of ZigBee and DigiMesh protocols.

After powering the GPRS, 3G and GPS module, please **check the correct connection** to the network and satellites respectively.

11. Sensor Boards

General sensor advice

Before turning on the board make sure all the connections are correct according to the indications given in the Technical Guides.

Try to minimize the time the sensors are powered, keeping it as close as possible to their response time, in order to minimize consumption.

Read carefully the Technical Guide corresponding to the sensor board in use for a detailed explanation about sensor operation and configuration.

When handling the value read from a given sensor take always into account the format of the output (integer, floating point, string of characters...) and the units of the resulting conversion, shown in the corresponding Technical Guide.

To improve sensor accuracy, it is a good practice to implement a "software filter". This can be easily done with a `while()` loop where the user measures the sensor 8 times, and gets the mean or average value. This is specially suggested for analog sensors, this tip will avoid problems like noise. You can also place a `delay(100)` inside the `while()` to ensure time non-dependence.

Besides that, if ADC channel is changed, the first read must be dismissed.

Gases Board 2.0

Remember that when turning off the board the configuration of the sensor stages will be lost.

Please refer to sections "General considerations in the use of the sensors" and "Starting with the gas sensors" for more information about how to estimate the configuration parameters of gain and load resistance.

Try to minimize the time that sensors are kept powered to that required by the application and, if possible, turning them on separately.

When handling the Atmospheric pressure sensor, a software filtering (such as a median filter) may be advisable if a noisy output is shown.

Events Board 2.0

Disconnect the sensor supply voltage for the unused sockets through the manual switch of the board in order to prevent excessive consumption and a potential interference the interruption flag.

When handling piezoelectric sensors take into account that some current peaks may be too narrow to be measured with the analog-to-digital converter, so detecting the interruption will be a better strategy to read this sensors.

Agriculture Board 2.0

Deactivate the pluviometer interruptions along intensive rain periods in order to prevent a continuous consumption owed to re-triggering. Instant measurement is advised in these cases.

When handling the Atmospheric pressure sensor, a software filtering (such as a median filter) may be advisable if a noisy output is shown.

Use the `sleepAgr()` function of the library when putting the mote to sleep when using the board interruptions for a proper configuration.

Smart Metering Board 2.0

When combining the Current Sensor with USB port transmissions, current peaks may lead to a brief interruption in the serial communication, thus switching the port of the computer to which the board is connected. There is no effect when using other communication modules.

Smart Cities Board

Remember it is necessary a calibrated Smart Cities Board to be able to use the noise sensor reading functions of the library. Beware not to delete the calibration coefficients stored in the EEPROM.

Smart Parking Board

Remember the Smart Parking Board is supplied calibrated to operate properly, with the calibrations coefficients stored in the Waspmote EEPROM. Beware not to overwrite those positions, which would lead to an irreparable error.

For a detailed explanation about a Smart Parking application deployment, please take a look at sections "Application considerations", "Installation of the mote" and "Powering the mote" in the Smart Parking Technical Guide.

Due to the fact that a Smart Parking system will be installed under the ground, bear in mind the maintenance or reprogramming of the node will be difficult or impossible. We suggest to implement many extra tests before the final installation to ensure the code and configuration will work. You can design strategies to face bad conditions. Also, we suggest to masterize the installation process.

Radiation Board

Radiation board contains high voltage parts (500V and above). Do not touch them directly with hand or with any object under any circumstances. There is a high voltage risk.

Led bar of Radiation board is a useful indicator. Adjust it manually to fit your requirements just changing the threshold values.

12. Networking

When using communication modules some tips might be useful:

General advice

- The deployment of transmitters and receivers should be planned carefully. The ideal situation would be to have line-of-sight between antennas, and a good clearance from obstacles like buildings, trees, etc (1st radius of Fresnel zone should be respected). There are free web-applications that allow to plot the RF path in each link.
- Obviously, outdoors performance is better than indoors since there are less obstacles.
- You should always prefer elevated locations (specially for repeaters or central nodes). Mounts or high rooftops are good options. If installing in a lamppost, 4 metres is better than 3.. and 5 is better than 4. In any case, a pole is a good investment to gain some height.
- Always experiment the placement in each deployment spot. Moving the node few cm can lead to a 5 or 10 dB improvement. To do so, a special mode with frequent transmissions can be done.
- The deployment should avoid places with interference sources, like cellular base stations, high-voltage cables, etc.
- A rule of thumb in telecommunications is to keep a 10 dB margin in every link between the mean RSSI and the receiver sensitivity. This will protect the link from unwanted, unpredictable fading effects, interference, moving obstacles, weather effects, etc.

XBee modules

- XBee ZigBee modules need to join the network when they are powered on. So it is needed to check the association indication before try sending for better performances.
- XBee DigiMesh needs a short period of time to create the routing tables when it is powered on. So it is recommended to wait for a couple of seconds before sending.
- Generally, it is really useful to include retries in your Waspmote codes so as to ensure the correct transmissions.
- When a packet is received via XBee, some space is reserved in memory to store it. Due to this matter, after treating this packet, the reserved space must be freed. If the packets are not freed, the code will crash when no more memory is available.

Sigfox module

- The user must keep in mind the ETSI limitation: 140 packets of 12 bytes, per object per day.
- The sending process of a single packet takes several seconds so this has implications about power consumption. If ACK is used this process takes much longer.

LoRa module

- As explained in the the LoRa Guide, the time to transmit one packet depends on the selected mode, and can be as long as 6 seconds. This has huge implications about power consumption, channel availability, maximum number of nodes in the network, etc. The user should explore what mode can be used, in a good balance range VS time of transmissión.

GPRS and 3G

- The GPRS and 3G modules need a lot of current (in comparison with Waspmote or XBee modules) to work. It's very important to do an efficient code to save the maximum of battery. Limit the connection time. If the module cannot connect to the network, power off the module and wait to the next send cycle.
- Reduce the use of the module. For example, upload a file at day instead upload a file every hour.
- If is possible, install the Waspmote with GPRS or 3G with an external power source (for example the solar panel) to recharge the battery of the Waspmote. This increases the life of the mote before substitute the empty battery with other with charge.
- Contact with your network provider to know the APN of your connection, user name and password may be required.

WIFI

- The WIFI module requires that the battery is connected. Do not let the battery discharge completely.
- When using encrypted access points, be careful with the length of the keys (WEP = 13 characters) and in the case of passphrases (WPA1, WPAMIX, WPA2) set them in plain text (not hexadecimal).
- To check if your WIFI module works correctly, connect it to a gateway and open CuteCom, Hyperterminal or a similar program. Send the string $$$ with no carriage return to the module; if the module responds "CMD", it means that it works properly.

Bluetooth Pro

- Bluetooth inquiries need some time to be performed. If you are planning to discover a large number of Bluetooth devices, it is recommended to set inquiry time above 10 seconds.
- The "friendly name" of each Bluetooth device requires a lot of time for being read. If you want to speed up your inquiries, avoid asking this parameter with the corresponding function.

Bluetooth Low Energy

- Last scan is saved on the EEPROM by default. When a new scan is performed, previous scan data is overwritten. it is recommendable to save the scanning data in other way, like SD card or send it through a radio module, to avoid missing the data.
- If you are sending your custom commands to the BLE module, remember to parse the command answer and the possible events. The command answer should arrive in less than 100 ms, but the events can take more time.

RFID

- Introduce a little delay after read-write operations (a few milliseconds). This will avoid the hang of the module.
- Do not try to read or write unless the authentication operation was passed. You can use several if's to control the code flow.
- Don't write address 0, 3, 7, 11, 15, ... if you are not an advanced user. You could leave your tag unaccessible.
- Networking can be the most complicated part of Waspmote, we suggest to read the Guides thoroughly and experiment with the examples.

13. Frame class

There is a class included in Waspmote API which allows the user to create formatted frames easily.

- Use this API tool so as to create your sensor frames. A project should use the **official frame** described in the Frame Guide. It is worth to take the time to use the official frame.
- **Plan** carefully what information you are sending in your frame. Maybe you just need to know one parameter, but in the future you may need to know the battery level or the available RAM. Normally, too much information is better than few information. As a matter of fact, the energy consumption to send 60-byte frames is almost the same than the energy needed to send 90-byte frames.. the difference is negligible.
- Send sensor data frames, but also consider sending "**status frames**" (battery, RAM, loop counter, etc). They will help the network administrator to know about the current conditions inside each node. Besides, they will act as "I am node 'x' and I am alive" frames.
- The Frame class enables to send different type frames, you can use them to notify special **events**.
- For beginners, create **ASCII** frames which are easy to read by the user.
- Use Binary frames which permit to include more sensor fields than ASCII frames.

For further information, please check the Data Frame Guide where all features of this useful tool are described deeply.

14. Interfacing with Meshlium

Before start transmitting to Meshlium it is recommended to start using one **Gateway** in order to validate the Waspmote's code. It is **easier** and faster for the developer to do the first testings with the Gateway.

Meshlium is not a simple Gateway, it is a **full Linux computer**. It is important to **study the manuals** thoroughly in order to get a good project. All the information is found there.

It is specially interesting to use the **official data frame** in projects with Meshlium, since it can receive and parse each frame.

15. Real deployment advice

Before real deployments there are several points to study:

- **Test** your final code for **several days** "in laboratory" before the final installation. This will help to discover hidden bugs in your code or possible error sources.
- In the testing process, rise the number of your cycles by decreasing the time to sleep to a **few seconds** (instead of sleeping 10 minutes, sleep 10 seconds). This way the user can simulate many "virtual days" in few hours. Anyway, we suggest to simulate as much time as possible with the real sleep time.
- The final code is written and tests were successful. Alright. But did the developer consider things which could go wrong and did not happen (or could not be reproduced) in the performed tests? For example, it is highly improbable that an XBee module, once it is configured (normally in the `setup()` function), is mis-configured, but it could happen. A good way to prevent that can be to configure the XBee in every loop or after a certain number of loops. Really robust code works well not only in normal conditions, but also **when strange effects happen**. Be sure to add code to check if everything goes as planned; and if not, you should have an alternative piece of code for it.
- Another example for the tip above could be the strange possibility that the destination Meshlium is temporally unavailable (link down because of interferences? electrical supply fail?). In this case, frames sent towards it would be lost. In this case, you can write a piece of code to write in the **SD** card the frames which are not being received. Afterwards, you can go to that node and get the SD card in person. You could even write all the frames in the SD card as a security measure.
- It is impossible to know if 2 nodes will be in range or not by reading a map. There are so many variables: distance, obstacles, materials, weather, reflection, multipath effect,... The only way to know the best places to deploy nodes for a project are **real tests**: try the nodes on the field. Maybe the lab tests you performed said 'x' meters, but you will get other distance on the street.
- It is also difficult to know the life expectancy of the battery for each node in real life. Take into account that there could be retransmissions, high/low temperatures affect to the battery performance, batteries have a significant self-discharge rate, etc. All that makes that battery life estimations obtained in the lab may not suit with **real deployments results**. Consider a certain factor which divides the estimated life expectancy.
- In the final code which will be executed by the installed nodes, we advise to do the first 10 loops (for example) with a very **short sleep time** (seconds and not minutes). To do so, you can have a loop counter and place an `if()` to make a shorter sleep time if the loop counter is lower than 10. This practice enables the people installing the nodes to check in few minutes if the first 10 frames are arriving to destination correctly, which is a good sign (better than just 1-2 received frames!). If some incident may occur, they are still in place to take measures.
- You can also use the LEDs to signal important events inside those first 10 loops.
- For maximal robustness in final codes, the programmer may find interesting to experiment with the `PWR.reboot()` function. As a matter of fact, many code errors happen when the code has been executed many times, due to low-level, memory or weird microcontroller reasons. This is why it could be good to reset Waspmote in a software way when a certain number of loops have been executed. After the software reset, the microcontroller will be fresh to start again, just like in many electronic devices. It is not an elegant practice, but it could be an **effective measure** for your project.
- Anyway, the best way to get really robust and reliable nodes is to perform periodic hardware-type resets. This can be done using the hibernate sleep mode and is more effective than the process explained above. In practice, the hibernate function acts switching off everything in Waspmote (except the RTC) flushing all variables and processes. After hibernate, everything starts from 0, it is like pressing the physical reset button. This avoids possible problems due to bad programming or long-term microcontroller issues, etc: we can focus on just the first loop. Just remember hibernate is only available for Waspmote, but not for Plug & Sense!.

16. API changelog

When creating the API for Waspmote PRO (v12) and in every new version of this API, Libelium tries to improve the software system by making libraries more stable, more robust and more easy to work with. Libelium tries to leave functions as they were. However, in many cases, some changes were advisable; so we make changes.

Inside each Programming Guide it is possible to find an API changelog link referring to the changes made inside the library explained in the Guide.

In this link we are showing all those changelogs together. It is a nice aid for those Waspmote owners willing to **migrate** their codes to newer API versions.

www.libelium.com/development/waspmote/documentation/changelog/

17. Documentation changelog

From 4.4 to 4.5:

- References to the new Sigfox module

From 4.3 to 4.4:

- References to the new LoRa module
- More advice about networking
- Link to the new online API changelog

From 4.2 to 4.3:

- Added references to the new GPRS+GPS module
- Added references to StackEEPROM
- More tips about variables and hibernate

From 4.1 to 4.2:

- Added references to the new Bluetooth Low Energy module

From 4.0 to 4.1:

- Added references to 3G/GPRS Board in chapters: libraries includes, general advice, power and networking.

18. GitHub Project

GitHub is a web-based source code repository. It acts as a centralized location for software developers to control and manage free and open source software development. Inside GitHub, Libelium has created a project **Waspmote API**, the user can found this project in https://github.com/Libelium/waspmoteapi

> ***Note:*** *The use of Waspmote-API GitHub Project is optional. It is great for those users who have detected some bug and want to collaborate with the Community. Also users can add their own improvements.*

Important, all user that want use GitHub, must be registered in http://www.github.com

This new feature is aimed at users to want improve, correct bug or add functionality to share with the community must:

- Clone the repository (to fork) to be copied to your account.
- Making changes in your copy.
- Finish issuing a pull request to the owner of the original repository.

Libelium maintenance two repositories:

- Waspmote-API repository: https://github.com/Libelium/waspmoteapi
- Waspmote-API Unstable repository: https://github.com/Libelium/waspmoteapi_unstable

Waspmote-API repository is the official version of Waspmote API. The user only have read access. Nobody can push. The repository may only be downloaded.

Waspmote-API Unstable repository is the official version of Waspmote API. Users who want to work with this repository, should clone the Waspmote-API Unestable repository. The user must work about own repository, when a user wants to share their improvements, detect and fix any errors, their must carry out a "Pull Request" about **Waspmote-API Unstable**.

18.1. Install Git

Windows

Download and install the last release of Git, can be found in: http://code.google.com/p/msysgit/downloads/list

Linux

There are multiple ways of installing Git on Linux. Depending on which Linux distribution you're using and whether it provides binary packages for Git, you can either install a binary package or install from source.

- Debian/Ubuntu

```
$ apt-get install git-core
```

- Fedora

```
$ yum install git
```

- Gentoo

```
$ emerge –ask –verbose dev-vcs/git
```

- FreeBSD

```
$ cd /usr/ports/devel/git
$ make install
```

- Solaris 11 Express

```
$ pkg install developer/versioning/git
```

- OpenBSD

```
$ pkg_add git
```

Mac OS

One way it is download from http://git-scm.com/download/mac, other options is use MacPorts(www.macports.org) and install executing:

```
sudo port install git-core +svn +doc +bash_completion +gitweb
```

18.2. Create GitHub account

Create free account in https://github.com/signup/free.

After install Git, the user must generate an SSH key. This is the key used for accessing your GitHub repository. Open a terminal window on Linux/OS X or the Git Bash on Windows.

You can find more information on how to generate SSH Key in http://help.github.com/articles/generating-ssh-keys

18.3. Install Git Client

Libelium has chosen a SmartGit/Hg like a graphical Git client which runs on all major platforms. Git is a distributed version control system (DVCS). SmartGit/Hg's target audience are users who need to manage a number of related files in a directory structure, to coordinate access to these files in a multi-user environment, and to track changes to them. Typical areas of application include software projects, documentation projects and website projects.

The user can found SmartGit/Hg in http://www.syntevo.com/smartgithg/index.html.

Windows

The steps for install SmartGit/Hg in Windows are:

1.- Download the Smartgithg from www.syntevo.com/smartgithg/download.htlm

2.- Unzip the downloaded file

3.- Execute setup-version-jre.exe and follow the installation instructions.

Linux

If there is no JRE present on your system yet, use the package manager of your distribution to install Java, e.g. from OpenJDK, or download from oracle.com.

The steps for install SmartGit/Hg in Linux are:

1.- Download the executable from www.syntevo.com/smartgithg/download.htlm

2.- Unzip the downloaded file, execute next command:

```
$ tar xvf smartgithg-generic-version.tar.gz
```

3.- SmartGit/Hg are inside smartgithg-version folder-

Mac OS 10.5 +

For Mac OS X, version 10.5 or newer, the necessary JRE is already installed on your system or will be downloaded automatically the first time you launch the application.

The steps for install SmartGit/Hg in Linux are:

1.- Download the executable from www.syntevo.com/smartgithg/download.htlm

2.- Unzip the downloaded file, execute next command:

```
$ tar xvf smartgithg-generic-version.tar.gz
```

3.- SmartGit/Hg are inside smartgithg-version folder-

18.3.1. Configure Git Client to clone repository

All steps to configure git client are equals for all platforms.

Accept the license for SmargGit/Hg

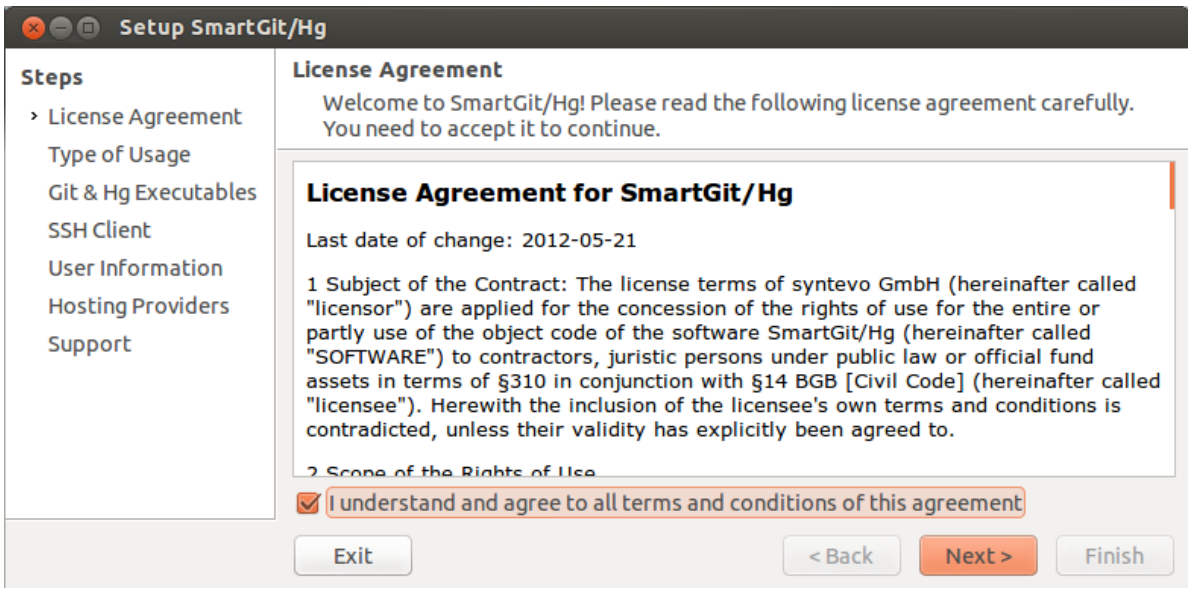


Figure : Step 1 of the installation

Select Non-commercial use only, Waspmote API is open source, with GPLv3 license.

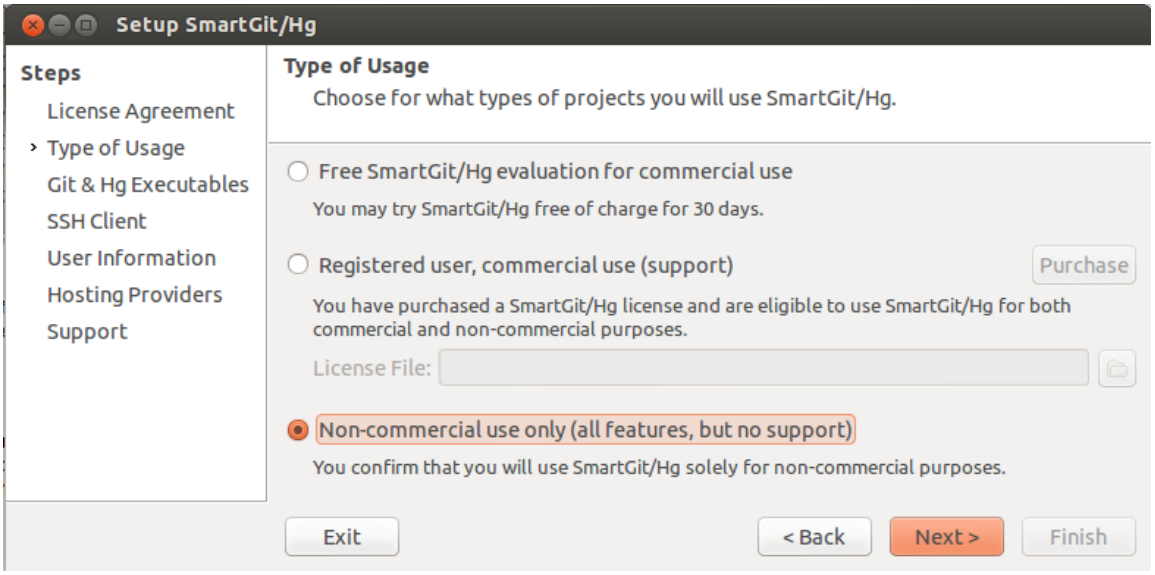


Figure : Step 2 of the installation

Splecify the path to your "git" executable

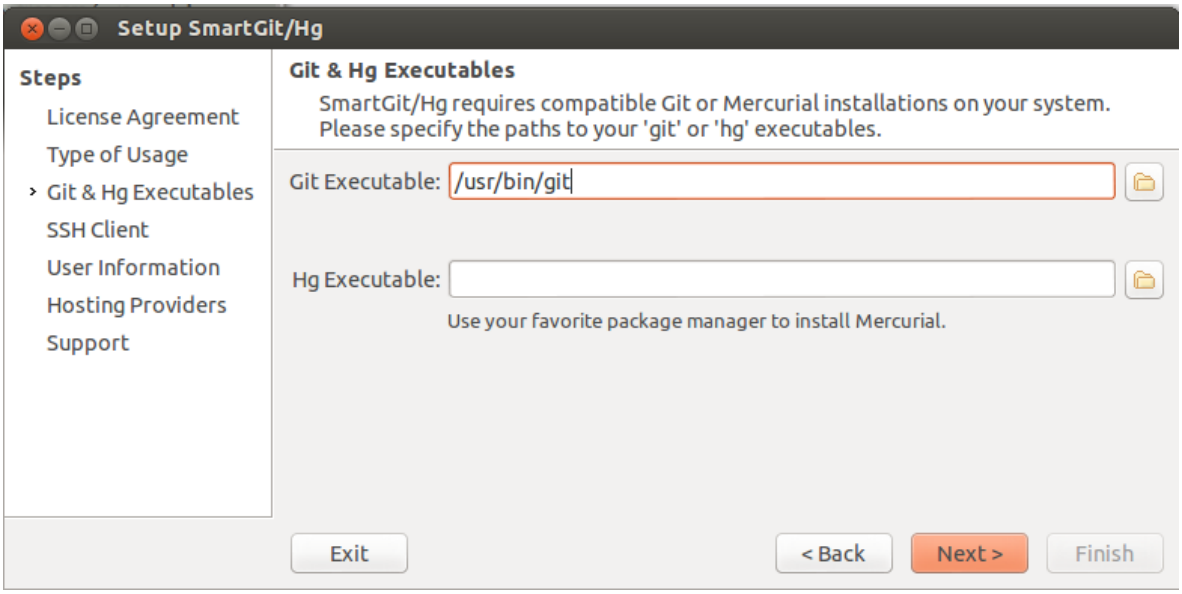


Figure : Step 3 of the installation

Select use SmartGit/Hg as SSH client.

Figure : Step 4 of the installation

Specify your user name and your mail to store your commits.

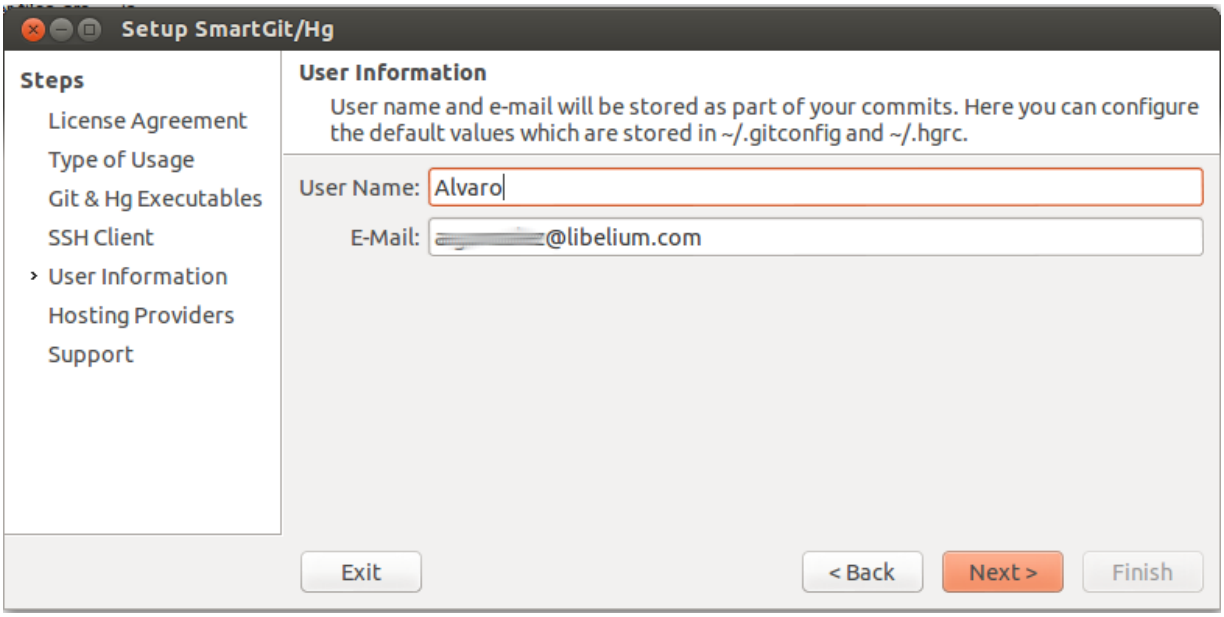


Figure : Step 1 of the configuration

Select "My name main hosting provider is GitHub", and complete the fields with name and pass of GitHub user.

Figure : Step 2 of the configuration

Select "Don't use a master password"

Figure : Step 3 of the configuration

Specify the Git or SVN repository to clone. The repository of Waspmote API Unestable project is:

git@github.com:Libelium/waspmoteapi_unstable.git

> ***Note:*** *To work with this repository the user must be collaborate. Therefore, each user must fill this field with own repository, create after doing fork.*

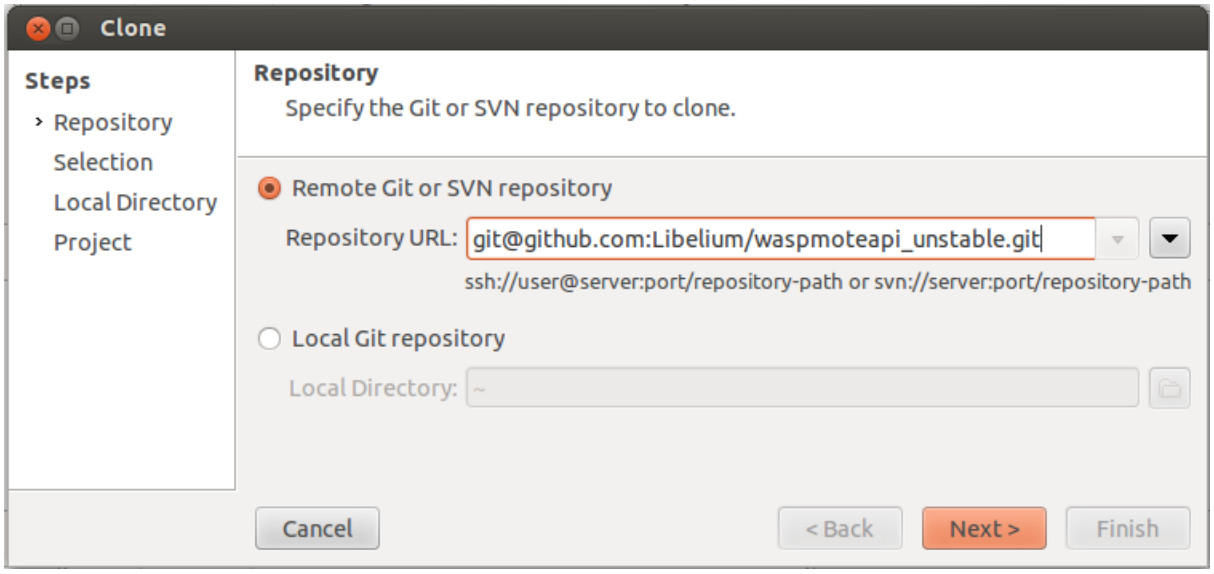


Figure : Step 4 of the configuration

Select authentication private key type and write private key file and passphrase use in generation SSH key.

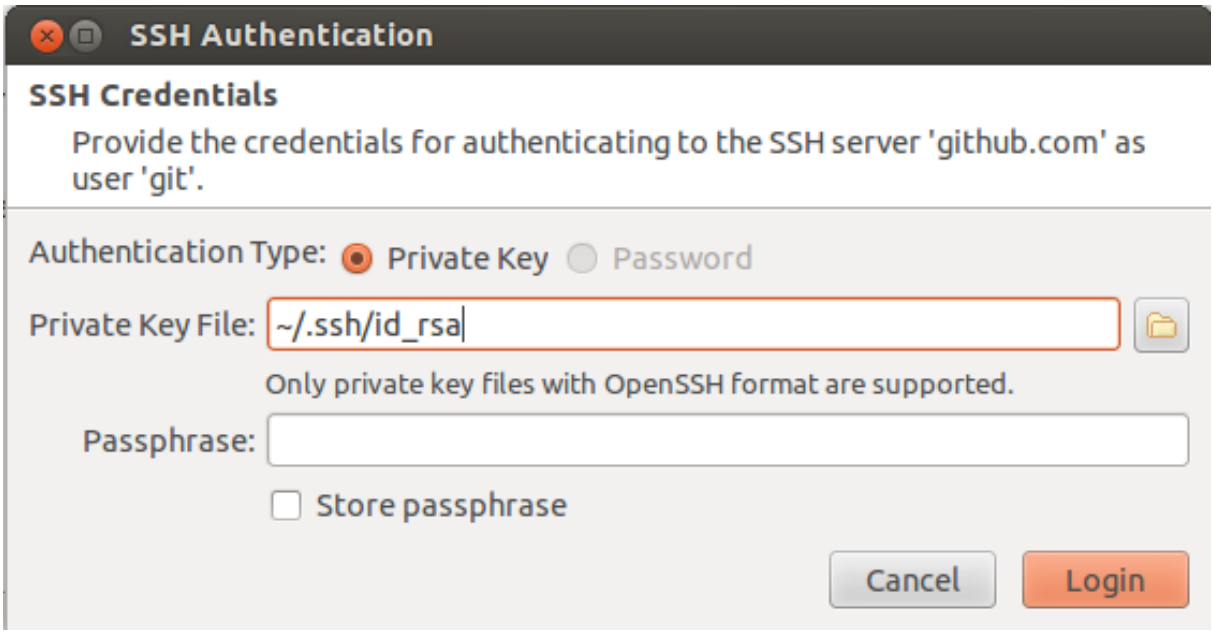


Figure : Step 5 of the configuration

Specify the local directory for the new repository.

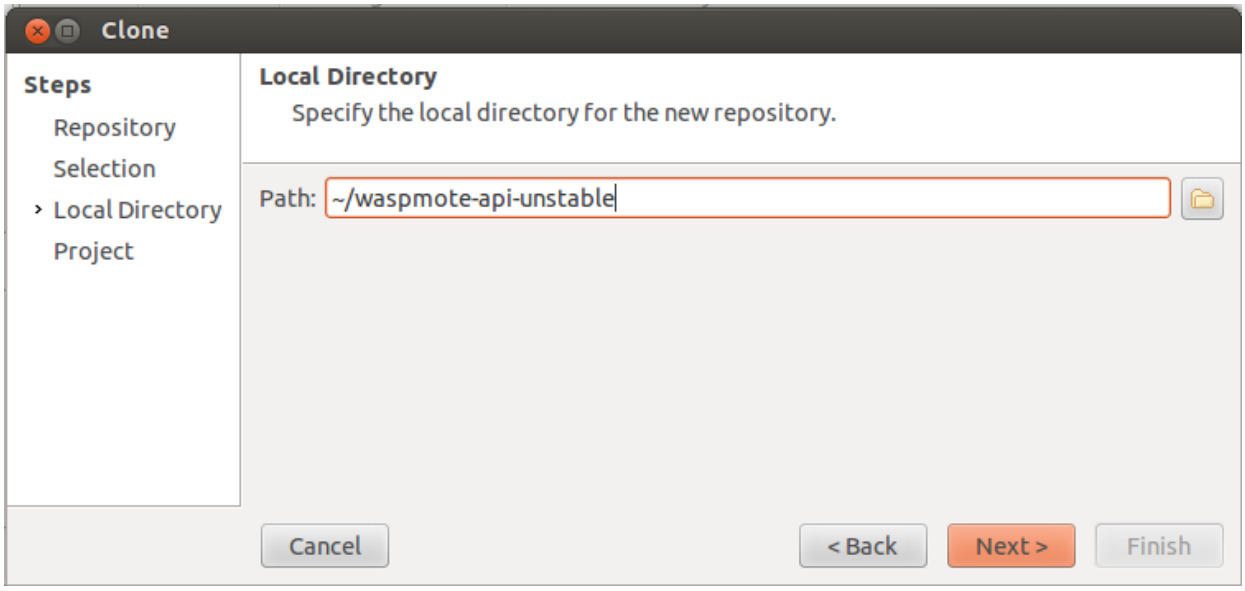


Figure : Step 6 of the configuration

Last, specify the name of the project.

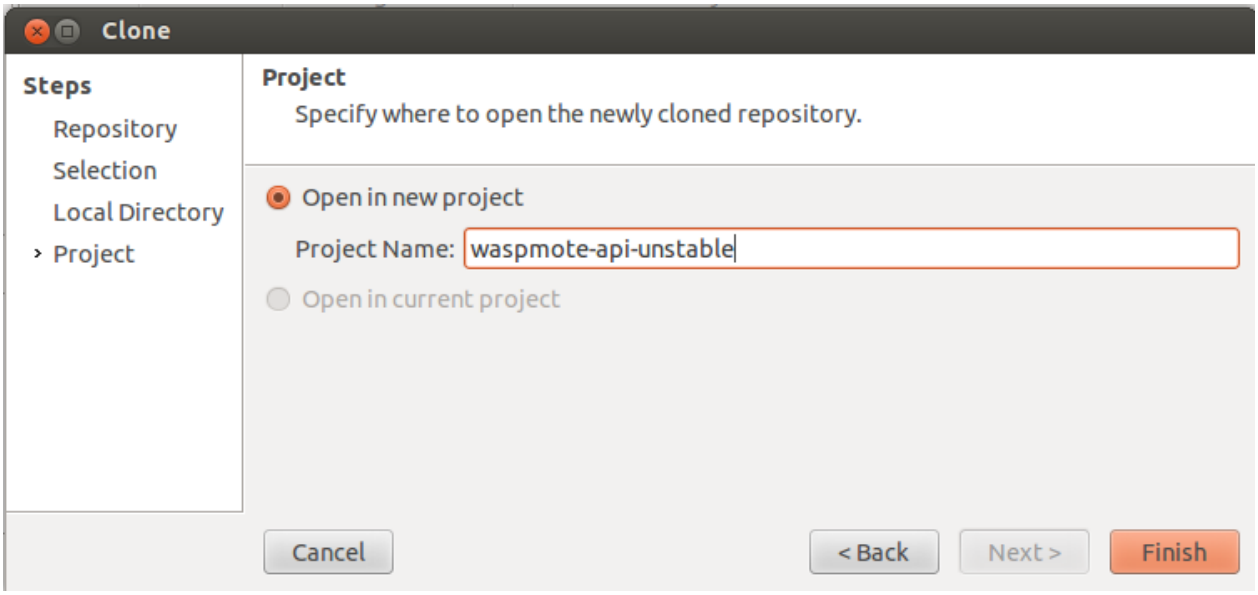


Figure : Step 7 of the configuration

18.4. Thread "Repository Code for Developers" on Forum

Inside Forum Libelium (http://www.libelium.com/forum/), exist a new thread named "Repository Code for Developers".

In this threads users can try and answer your questions with the Libelium technical team.

It is also mandatory for a "Pull Request" to be reviewed and accepted. Users must explain in the forum, what modifications want to commit to the Waspmote API Unstable Repository.

18.5. How upload your code to the Waspmote API Unstable repository

18.5.1. Create new fork

Create new fork means clone a repository on your own account. Thus each user can make their own changes. All user of GitHub can create their own fork. It is possible to make fork of Waspmote-API repository and Waspmote-API Unstable repository.

In this guide http://help.github.com/articles/fork-a-repo can be found how create new fork for own project.

Libelium recommend that each user create a new fork and work on their own repositories.

Create a new repository can be done from the web. To create new repository press the button "fork".

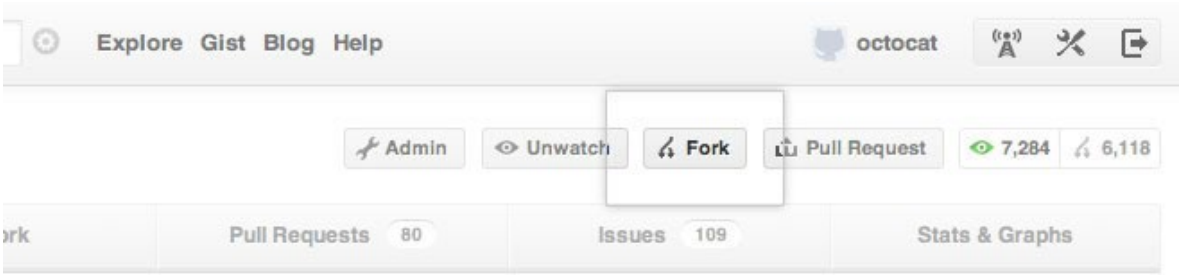


Figure : Fork button on GitHub

18.5.2. Create new branch

Branch is a separate line of code with its own history. You can create a new branch of an existing one and change the code independently of other branches.

Branch management is an important part to the git workflow. The user can also choose to manage branches directly on www.github.com

More information about how create and delete branches on repository can be found in:
https://help.github.com/articles/creating-and-deleting-branches-within-your-repository

18.5.3. Commit to your own GitHub account

Commit is like committing changes. This creates a new revision, which can be retrieved later, for example, if you want to view or retrieve the source code for an older version. Each commit contains the author and committer or who makes the changes, making it possible to identify the source of change. The author and the committer could be different people.

When a user performs commit, this change will apply to your local repository pressing "commit". In order to upload to the user's account in GitHub you must run "commit & push" from the program SmartGit/Hg.

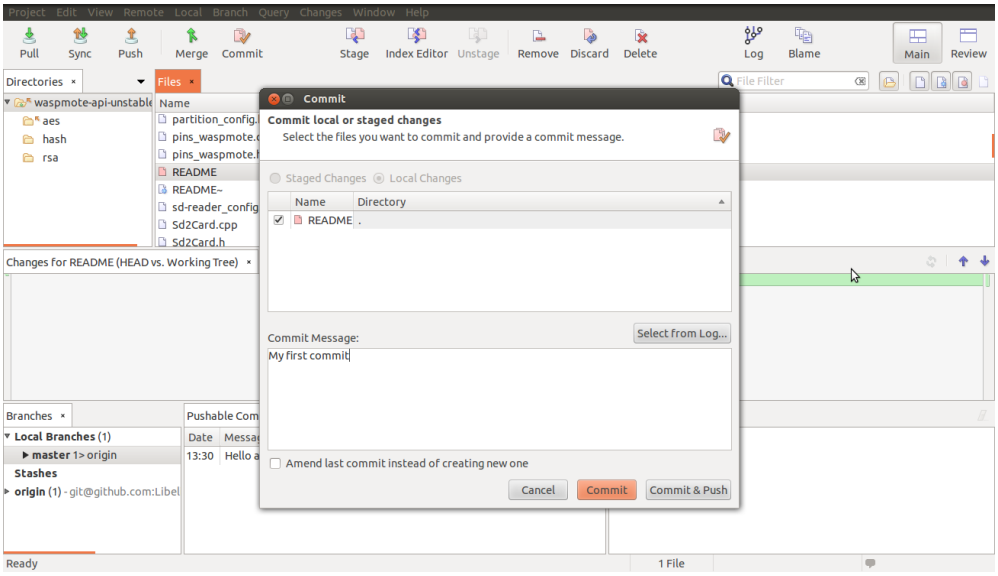


Figure : Commit & push

18.5.4. Commit to the Waspmote API Unstable

To add your code to the Waspmote API Unstable repository, must execute "Pull Request" for the changes to be reviewed and accepted.

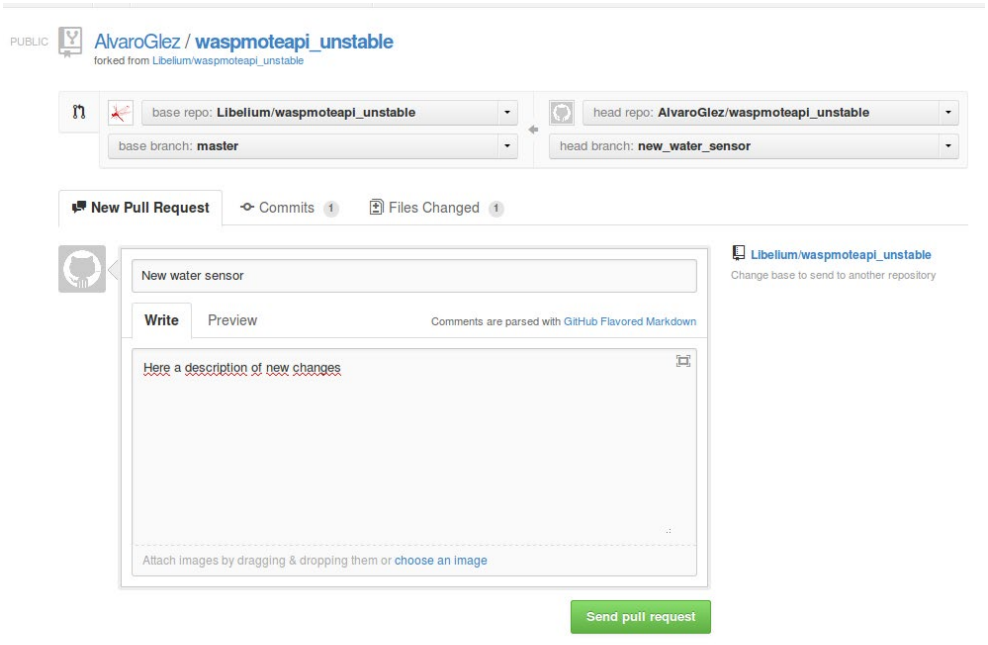


Figure : Pull Request

Can find detailed information on how to perform a "Pull Request" http://help.github.com/articles/using-pull-requests.

Once the code is submited, the Libelium Dev Team will review it and will back to you with any doubt or question in order to review it properly.

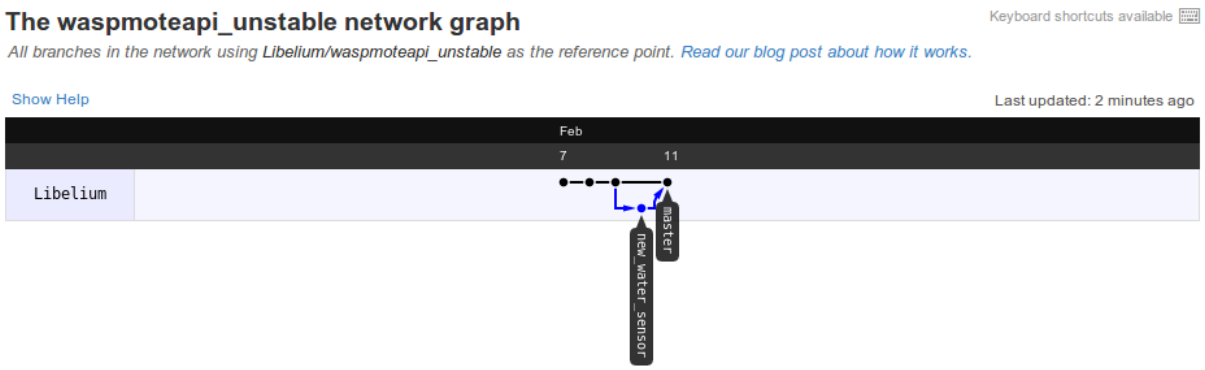


Figure : Pull Request accepted

> ***Important:*** *Addition it is also mandatory for a "Pull Request" to be reviewed and accepted. Users must explain in the Libelium Forum, what modifications made to Waspmote API Unstable code.*

Note: *Code must be commented so that it keeps understandable and clear for all users.*