FINAL PROJECT

SSH CLIENT FOR IPHONE IN IOS 4.3

STUDENT : *JAVIER FLORES FONT*
CAREER: *COMPUTER ENGINEERING*
***CREDITS**: 12 ECTS*
TUTOR: *ALEXANDRU MIHNEA MOUCHA*
TUTOR-UPV : *JUAN CARLOS RUIZ GARCÍA*
DATE: *JANUARY 2011 – JULY 2011*

INDEX:

1) Introduction

1.1) INTRODUCTION

In the department of CVUT E305 (Prague University) there are some servers which are frequently used by members of this office and around it.

Due to almost of this members have an iPhone device, one of them (Alexandru Mihnea Moucha) proposed us an application to manage this servers simply using this apple device. This application would have two functions: First, to check which servers are online at the present time in the office, and the second one, be able to send some commands to manage the network.

Besides, is a good idea to develop an application for iPhone because we must say that IOS is one of the Operating Systems for mobile phones (iPhone) / tables (iPad) more commonly used nowadays, offers a lot of applications to the mobile sector.

Furthermore, we must say that there are an extensive documentation of this device.

Aditionally, developing an application will be helpful in the future for us if our future work will be relationated with mobile devices or others (iPad,iPod) thanks to the compatibility between devices of Apple.

Summarizing, due to this purpose and due to the constant rise of mobile devices sellings, and the success of the apple store nowadays, we have decided to develop an application to this phone. In particular, we are going to develop a SSH client for IOS 4.3 . As this way, any user of this department could access to some server of this area and order some commands in the terminal via SSH, simply having the iPhone device in their hands.

1.2) GOALS

This project have two goals:

-The first one, to create a SSH client for iphone which will be useful for the members of the department E305.

-The second one, to develop an iPhone application on a new platform, with a new programming language, and SDK and iOS (iPhone OS), which never have been studied in the career.

We are going to start from a basic idea: what is a SSH client and which tools we have to build in our project in order to solve all the problems that arise while developing it.

Later on in this project, we will enter more deeply all the SSH client's objectives, details, connections,database and other concepts that will help us understand the rest of the document.

1.3) FEATURES OF THE SSH CLIENT

Now that we have a clear and basic idea of the PFC goal, we will briefly describe its functionality , this is, which things our SSH client must do.

The application when launched will consist of a main window , in which we can see three empty fields asking about the identification of the server: IP, User and Password. Once we are identified, we will press the button "connect" and the application will connect remotely to our server and once this is done, we will access to a second window in which the application shall run the commands introduced on the remote server and return the output to our device.
Besides, in this view we have this options:

- Send the answer of some command via SMS to some phone.
- Send the answer of some command via EMAIL to some computer.

For finishing, the client will be able to check before connecting the list of it servers and to know which servers are online at the current time

1.4) IPHONE / IPOD TOUCH

IOS SDK 4.3 for iPhone is the platform that we are going to use. This kind of mobile device, together with Android, are the most interesting attractions that are now on the market.

It must be said that this project is developed for iPhone, but in fact, is fully compatible with iPod Touch due to they share the same features except the phone.

To learn more about the iPhone ,what offers the iPhone device, all of its technical characteristics , we can find more information on the Apple's official website. Highlight specially the Multi-Touch display,the library CoreData to save information on Iphone, and the Internet connection via WiFi or 3G, which we will use in our client SSH to connect to other computer.

This information is regarding the iPhone 2G, 3G, 3GS and 4, and shows their great potential for development, all these functions have their own library ready to be used in our project using the iPhone SDK.

Summarizing, we have decided on which platform to implement and what we offer. Later, in the next pages, we will discuss the design and development of our ssh client.

1.5) SECURE SHELL

Since we wanted to execute commands on remote servers, the protocol chosen to perform this procedure is SSH (Secure Shell):

Secure Shell or SSH is a network protocol that allows data to be exchanged using a secure channel between two networked devices.The two major versions of the protocol are referred to as SSH1 or SSH-1 and SSH2 or SSH-2. Used primarily on Linux and Unix based systems to access shell accounts, SSH was designed as a replacement for Telnet and other insecure remote shells which send information, notably passwords , in plaintext, rendering them susceptible to packet analysis The encryption used by SSH is intended to provide confidentiality and integrity of data over an unsecured network, such as the Internet.

SSH uses public-key criptography to the authenticate remote computer and allow the remote computer to authenticate the user, if necessary. Anyone can produce a matching pair of different keys (public and private). The public key is placed on all computers that must allow access to the owner of the matching private key (the owner keeps the private key in secret). While authentication is based on the private key, the key itself is never transferred through the network during authentication.

SSH only verifies if the same person offering the public key also owns the matching private key. Hence in all versions of SSH, it is important to verify unknown public keys before accepting them as valid. Accepting an attacker's public key without validation would simply authorize an unauthorized attacker as a valid user.

The problem we found is that among the libraries that contain XCode (we will develop the application with this program, which we will describe it later) to develop for iPhone, does not exist any library to connect the device via SSH. Therefore, we had to search in the network something to fix this problem and we found a library called: libssh2 . With this library we can perform this functionality and it's accepted by the device.

1.6) SUMMARY

Once we have described the purpose of the PFC, the mobile device chosen, which platform has been chosen to develop the app, which protocol(ssh) is the best to connect to some Computers to manage instrucctions…, so in the rest of chapters we will explain:

- Chapter 2: In this chapter will study everything about the iPhone SDK, how to design and program for iPhone, in which language, how to test an aplication and finally which prerequisites need us for program in this platform

- Chapter 3: In this chapter we will discuss the specification of this project, specifically the name of the project and the Use Case Diagram

- Chapter 4: Here we will explain the implementation used to perform the specifications of the previous chapter. How internally we can connect to a remotely server, how to manage the servers and their answers will be our topic.

- Chapter 5: This chapter explains how long time have been pass since beginnig till the end of the project, so to explain it we will show our final Gantt Diagramm.

- Chapter 6: This chapter is the conclusions of the PFC.

- Chapter 7: Describes possible future work.

- Chapter 8: Acknowledgements

- Chapter 9: Bibliography

2) Developing in iPhone

In this chapter we will focus on how to develop for iPhone, what tools are needed, what design patterns are used to design iPhone applications, what is the life cycle of an application, how to manage memory and in which language we must programme.

2.1) iPHONE SDK

To program on the iPhone platform will need certain tools, such as a compiler, a work environment, debuggers, and so on.

For this reason, and to promote the development of applications for iPhone, Apple prepared a specific SDK6 to program on this platform. An SDK , which stands for Software Development Kit, is a package of software tools to develop, in this case, software for the iPhone.

Apple also prepared a standard model to develop in iPhone. This is called Model-View-Controller, this model encapsulates the application data, view displays, and driver software and acts as an intermediary between them. This pattern of use is not required, but greatly facilitates the programming and the SDK is prepared to implement this pattern.

This SDK, once downloaded and installed, contains three essential parts of development for iPhone, which are the XCode Tools, iPhone Simulator API and Cocoa Touch.

a) XCode Tools

This is the most important part of the SDK. The XCode Tools are a collection of the necessary tools to develop any application for iPhone or Mac. The most important and which we will use for the project are the IDE XCode and Interface Builder. In addition, Apple provides a complete and extensive documentation for each tool.

As a curiosity, this same setting is used by Apple to develop Mac OS X.

In the next page we will explain each of these tools::

a.1) XCode IDE

This tool is a development environment like Eclipse or NetBeans, which allows us to program, compile, debug and optimize any application. This IDE, besides being a professional code editor, allows us to use easily all Cocoa Touch frameworks included in the API Cocoa Touch which is presented below.

The compiler used by Xcode is a modified version of GNU Compiler Collection, a compiler open source makes compatible XCode C + +, C #, C, Objective-C, Objective-C + +, Java, Fortran, Python, Ruby, etc. Thus, within our code programmed in Objective-C, we can add functions and methods in C + + and then call them without recompiling. This feature is essential because our application will have some code in C + + (connection to another server) and part of code in Objective-C (window management, classes, etc ...)

Also included a very useful debugger that allows us to put breakpoints anywhere in the code to check at that time the value of all the variables and where they are located in memory. In this way we help you find bugs in our applications easily and conveniently.

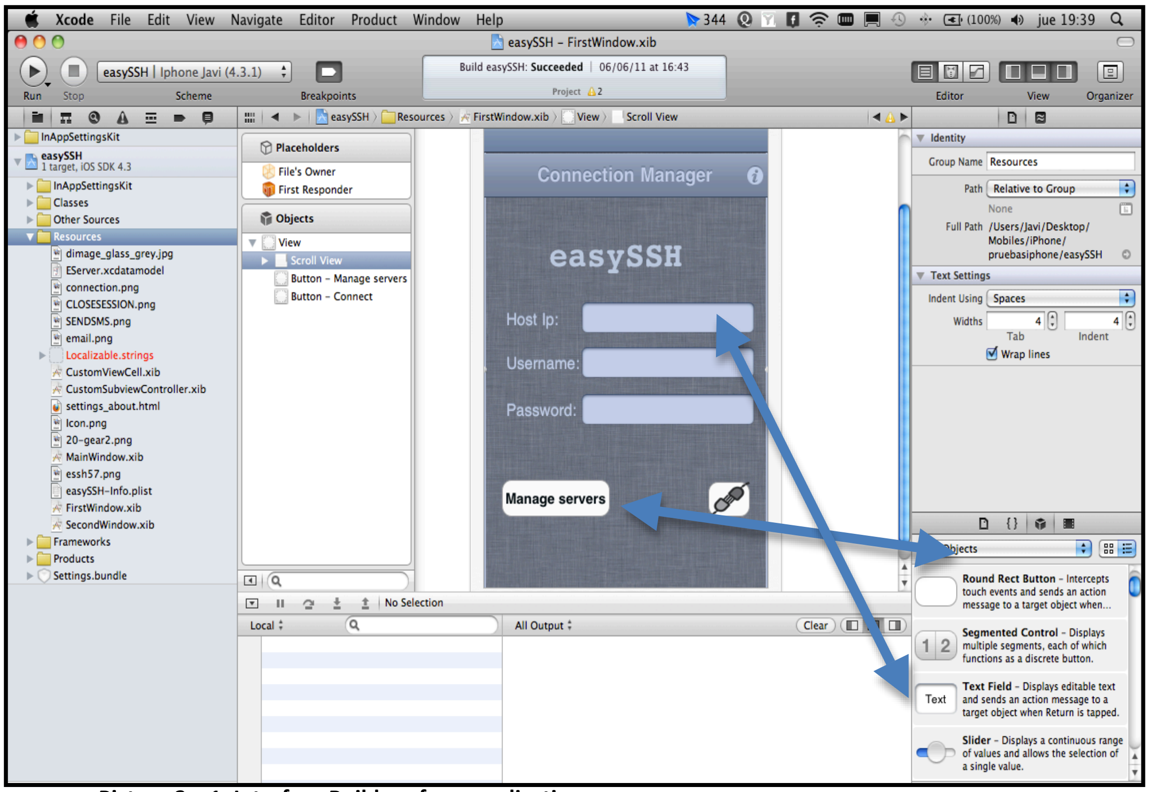
The program will be developed in XCode version 4.0.1 (4A2006), last updated as of today.

a.2) Interface Builder

The Interface Builder is a tool that was included in the SDK a bit later than the XCode IDE. This tool helps us create the graphical interface of our applications easily.

Each view created with Interface Builder generates a file to us. Xib that contains all the coded information for subsequent compilation. These files. Xib need to form a view of a controller to manage data input and output generated in each view.

In the next page we can see our Interface Builder:



Xcode File Edit View Navigate Editor Product Window Help
easySSH – FirstWindow.xib
Run Stop Scheme Breakpoints
easySSH | Iphone Javi (4.3.1)
Build easySSH: Succeeded | 06/06/11 at 16:43
Editor View Organizer
easySSH › Resources › FirstWindow.xib › View › Scroll View
InAppSettingsKit
easySSH
1 target, iOS SDK 4.3
InAppSettingsKit
Classes
Other Sources
Resources
dimage_glass_grey.jpg
EServer.xcdatamodel
connection.png
CLOSESESSION.png
SENDSMS.png
email.png
Localizable.strings
CustomViewCell.xib
CustomSubviewController.xib
settings_about.html
Icon.png
20-gear2.png
MainWindow.xib
essh57.png
easySSH-Info.plist
FirstWindow.xib
SecondWindow.xib
Frameworks
Products
Settings.bundle
Placeholders
File's Owner
First Responder
Objects
View
Scroll View
Button - Manage servers
Button - Connect
Connection Manager
easySSH
Host Ip:
Username:
Password:
Manage servers
No Selection
Local
All Output
Clear
Identity
Group Name Resources
Path Relative to Group
Full Path /Users/Javi/Desktop/Mobiles/iPhone/pruebasiphone/easySSH
Text Settings
Indent Using Spaces
Widths 4 Tab 4 Indent
Wrap lines
Objects
Round Rect Button – Intercepts touch events and sends an action message to a target object when...
Segmented Control – Displays multiple segments, each of which functions as a discrete button.
Text Field – Displays editable text and sends an action message to a target object when Return is tapped.
Slider – Displays a continuous range of values and allows the selection of a single value.

Picture 2 – 1. Interface Builder of our application

Figure 2 -1 we can observe the FirstWindow.xib, it is the file. Xib described above.

To create the look of the view we must to drag the different components of the library, such as buttons, labels, tables, or navigation bars directly to the screen in order to visually create our interface. Everything we put on the screen is what you see in the iPhone.

Then each component should relate to the objects on the screen controller (File's Owner) through connections. For example, if you create a button on the controller object (UIButton) with a particular behavior and a method associated with the press of this, we have to connect it to the button we place on the screen of the Interface Builder.

Finally, we have the Inspector, which will serve to set up visual details of the various components, such as color, text, size, placement and so on.

Therefore, with this tool, you can implement more easily the entire interface.

b) iPhone Simulator

Besides the XCode Tools, Apple's SDK also contains the iPhone Simulator, which is like an iPhone integrated into the Mac OS X, it allows to test our code easily because it does not need the physical device.

Nevertheless, any simulator has limitations. For example, the architecture used to build for iPhone is based on ARM7 and ARM6 and architecture used to build for the simulator is the Intel x86. This at first does not seriously affect the development of the application, because when you test the project on the actual device can easily detect how things change.

To debug an iPhone application it is necessary simultaneously use the debugger provided by the XCode Tools and simulator. That is, the application must be run on the simulator to using the XCode Tools debugger to find the bugs. The simulator allows us to turn to enter the inputs similarly to real and see how iPhone runs the application execution in real time. This makes it much easier to debug, since using the two tools together we can pause the application, return to the previous statement, and resume, to detect bugs.

c) API Cocoa Touch

The iPhone SDK also includes the Cocoa Touch APIs, mentioned previously, based on the Cocoa APIs used in Mac OS X.

Cocoa Touch provides an abstraction layer of the iPhone OS, iPhone OS. Thus, we can access to the following features that iPhone gives us:

1. Controls and Multi-Touch events

2. Accelerometers

3. Structure View

4. Locator

5. Camera

Most of these frameworks are grouped into UIKit Framework and Framework Foundation. In UIKit, we can find all the buttons, tables, etc.. these are identified by the UI code, for example UIButton. In Foundation we can find all the basic classes customized by Apple, such as the NSString (typical string in C), NSObject, and so on. All of them begin with the prefix NS.

Cocoa Touch API is programmed in Objective-C, this provides more flexibility, for example, if we create a view with Interface Builder (. Xib) with corresponding components from the Cocoa Touch APIs (UIButton, UILabel, etc. ...), these can be linked to the view controller also programmed in Objective-C, allowing the connection dynamically between the elements of view . xib and controller objects doing all at runtime without recompiling .

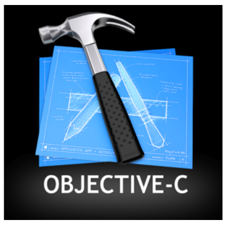
Also Cocoa Touch, mentioned before, allows you to mix code with other languages, especially C and C + +, Objective-C because it is based on these.
Another thing to note is that Cocoa uses the design pattern Model-View-Controller. This together with the Interface Builder allows us to save many lines of code.

We have talked about the main Tools that the iPhone SDK offers, but we must also mention other services that it offers. The SDK also includes the following frameworks that will be highlighted among others, as they are to use:

1. TCP / IP, sockets, etc..
2. Coredata database: To handle data from an application.

However, we had a problem: iPhone SDK does not include the SSH library, which will have to manually add and generate the appropriate code for the server connection, command execution and closing session

2.2) OBJECTIVE-C



Objective-C can be considered as an extension of C, it means that your own Objective-C compiler can compile C without any problem. The sintax is the same for object-oriented code, using the same construction, types of variables, expressions, pointers, etc.. than C, and takes the Smalltalk-style messaging.

What are the messages?
Objective-C is based on sending messages to instances of objects. In Objective-C does not call a method of an object, it sends a message to the object instance. Send a message involves sending a name of a method to the object and this object is responsible to interpreting the message at runtime. If the object which receive the message hasn't a method to respond to this message name is ignored and returns null. We must to clarify that if this happen, the compiler will not give any error.
We will observe the differences in the code:

```
C++: object ->method (parameter); Objective-C: [object method: parameter];
```

These two statements are the same, they have the same purpose : the object must to execute the method with the indicated parameters. As you can see, to send a message in Objective-C we must to use brackets. We can also change the properties of the objects using messages.

Example:
```
[object count:0]; object.count = 0;
```

These two instructions are specific to Objective-C, and do exactly the same: put the counter variable to 0. But they have a difference, in the second, we assign the value 0 to the counter variable directly, and in the first, we sent a message to the object equivalent to calling the method setCount. That means, in the first, being equivalent to a setter method , it can be considered protected from errors doing impossible to assign values not desired. For example, in the 1, as an accountant is an integer, we could not assign the value null as the object would reject the message and not change the value. However in the 2nd, we could assign the value zero because there are not kind of protection.

But, what are the advantages of messages? The answer is in the type checking at runtime: Dynamic Typing. As has been referenced above, we send messages to objects, and these are interpreted at runtime. If the message can be interpreted, it executes the method of that name, but is ignored and null is returned. This at first glance seems to have no advantage, is important with the protocols and delegates.

We are going to detail what they are:

- Delegates: In Objective-C, when you create an object (called Object1 , for example) we can assign a delegate using the method setDelegate. A delegate is any other object(Object2), and is programmed in the same way. Delegates are used to answer messages sent to the Object1, if this one has failed to respond. That is, if we send a message to the main object with a name of a method that is not implemented, do not allow auto reply message and is sent to the delegate to respond to it. This offers several advantages, including the possibility that the same delegate has other delegate, and create chains of messages between objects. If the main answer can not be sent to the delegate object, if the delegate does not, it sends the delegate of the delegate, and so on.

- Protocols: A protocol in Objective-C is a set of methods that can be assigned to an object.Similarly to the delegates, if a message can not be answered by the object, it attempts to answer with some of the methods associated with the protocol.

Objective-C, like C, uses two types of files, the file header. H and the file implementation. M.

Let's take an example to identify both parts:

```
#import <UIKit/UIKit.h>

@interface Person : NSObject {

     int count;
     Person * pair;

}

@property (nonatomic,retain) pair * question; @property int count;

- (void) addPair: (Person *) pair; @end
```

Picture 2 – 2. File Pair.h

```

#import "person.h"

@implementation Person

@synthesize count, pair;

- (void) addPair: (Person *) pair {

[self pair:pair];

} @end
```

Picture 2 - 3. File Implementation.m

With this fragment we will discuss other important aspects of Objective-C.

In Objective-C to specify the header file used the @ interface - @ end, and the implementation file used the @ implementation - @ end. In the header, you specify the class name (Person) and a colon followed by classes which inherit the protocols as well as in our example inherits from NSObject Person. Most times, always inherit the basic properties of the superclass base called NSObject, which contains methods and attributes of a single object belonging to the iPhone OS Framework Foundation. After we must to declare the attributes of the class. If it is a simple type variable, integer, boolean, etc.. is declared normally like: int count, but if the variable is an object is declared with pointers (Person * pair).

Then you have to define the properties of these attributes. Using the @ property we tell the compiler how to treat these attributes. This directive contains options in brackets if the attribute is an object and not a single variable

With Retain we retain the object in memory, i.e.,we create an object reference and nonatomic specify that the object can be accessed from different threads. Having defined the properties, it is necessary to synthesize in the implementation file with @ synthesize. This line of code, we automatically create getter and setter methods for these variables.

It is not necessary to define the properties for all attributes, for example, is not necessary for simple typed variables that are only accessed from the same class nor for variables that we want to create your own getter and setter. If the property for some attribute is not defined, isn't necessary to synthesize.

After defining the properties of attributes declared in the header file we must to declare the functions and methods and then implement them in the implementation file. The function syntax in Objective-C is as follows: first, a dash (-), second, the return type of variable in parentheses ((void)), third, the name of the function (addPair) fourth, two points separate the different parameters of the function. The syntax of these parameters is the parameter type in parentheses followed by the name ((Person *) pair).

Objective-C also includes selectors. The selectors are defined with the @ selector (nameofmethod). A selector is an identifier of functions and methods by name. Thus, we can assign buttons, timers, etc.. different selectors. For example, if we create a button, we can assign a switch-when pressed. Pressing the button, the program will look which selector is associated with the action of pressing and will execute the method with the same name. This would be equivalent in C + + like a pointer to a function.

Finally, discuss the existence of the type "id" variable, which allows to transform any type of variable dynamically. That is, we can declare a variable "id variable" and assign values to integers, floats, or other more complex objects without casting or recompilation or redefinition of the variable.

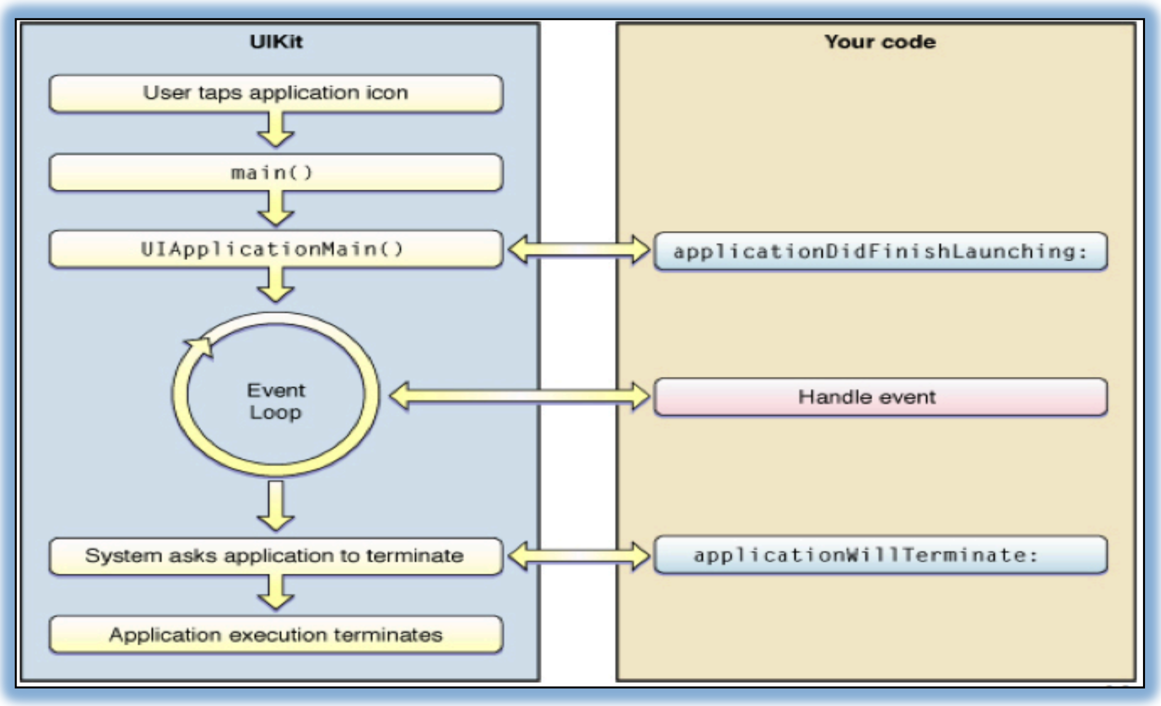
Once we have understood the syntax of Objective-C, we can see the great flexibility offered by the messaging system by the Cocoa Touch APIs. Is also programmed in Objective-C, we can send messages between objects and API objects, as well as override these methods by inheritance or modify the delegates of the API objects to customize their operation. That is, thanks to the Cocoa Touch API (also in Objective-C), we can modify the behaviour of its various objects such as UIButton.

2.3) CICLE OF LIVE OF THE APPPLICATION

In this section we will explain what happens since we click the application icon until it closes. The life cycle is important to know how to initialize the app so you can configure it the same way, and with this, we will know the overall performance of memory management and events.

The UIKit framework is responsible for keeping the application running. During the execution of the application it receives events, and these have to respond. These events are received by the UIApplication object, integrated into the framework UIKit, and are answered in our code.
Let us detail step by step what and who works when the user presses the button on the application for running.

After pressing the icon, the application proceeds to run by calling its main function. Once executed, the UIKit will initialize the application, first loading the user interface, and then giving way to the event loop, the main application loop. It is in this loop where the UIKit is responsible for coordinating all events received the answers that our program wants to give. When you press the Home button to exit out of the application, the UIKit calls methods in our code to signal that the application is closed, so we can save the status of your application if you want. After running your code, the application closes and frees the memory of the iPhone.


UIKit
User taps application icon
main()
UIApplicationMain()
Event Loop
System asks application to terminate
Application execution terminates
Your code
applicationDidFinishLaunching:
Handle event
applicationWillTerminate:

Picture 2 - 4. Cicle of life of the application

Figure 2 - 4 shows exactly the process followed by our application to be executed in iPhone.

We are going to clarify which things are part of UIKit:

Main

The main code is always the same for all iPhone applications.

```
int main(int argc, char *argv[]) {

NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init]; int retVal =

UIApplicationMain(argc, argv, nil, nil);

[pool release];

return retVal;

}
```

Picture 2 - 5. main.m

The code in picture 2 -5 start running the application, begin creating the main application itself (note the UI prefix is part of the framework UIKit) and then create a "memory space "for the application NSAutoreleasePool. The latter is also responsible for managing objects in memory.

UIApplicationMain

This is the main class of all iPhone app. After running the main, it creates its own iPhone application, ie, the UIApplicationMain, you specify who will be the leader delegate, the main class, and that view is loaded in the first instance.

Why not specify the delegate?
In this chapter we talked that our application responds to messages, and that is why we create the delegates to be responsible for answering. In this way we avoid doing UIApplication subclasses, as being part of UIKit, these have a fairly high complexity and is much more comfortable delegating than overwrite methods.

Event Loop

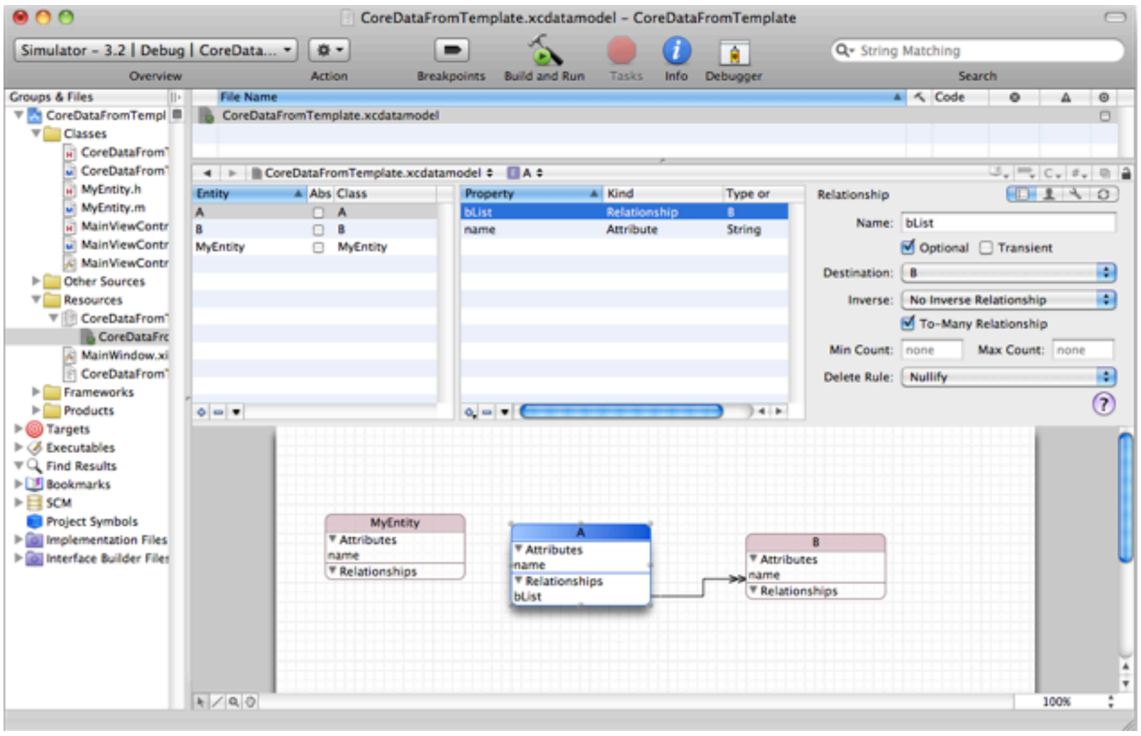
The event loop or main loop is one of the most important features of iPhone, and one of the distinguishing features of other styles of programming such as Java or C + +. And that is completely transparent to the programmer, to run an application on iPhone is trapped in this loop automatically capturing all user events, whether tactile, accelerometers, websites etc..
With the event loop we don't have to worry about creating the loop as would be in Java or C, UIApplicationMain manages automatically.

This method is implemented on iPhone and we dont need to lock it under any circumstances. That is, if we need to do an operation that takes a long time, such as downloading something from internet, we never have to do it inside this loop, because if not, the iPhone is locked and all events that are captured are not resolved.

2.4) MANAGING THE MEMORY

We are going to use the Framework Coredata to manage and save the servers on iPhone.
Core Data is a framework Apple provides to developers that is described as a "schema-driven object graph management and persistence framework." What does that actually mean? The framework manages where data is stored, how it is stored, data caching, and memory management. It was ported to the iPhone from Mac OS X with the 3.0 iPhone SDK release.
The Core Data API allows developers to create and use a relational database, perform record validation, and perform queries using SQL-less conditions. It essentially allows you to interact with SQLite in Objective-C and not have to worry about connections or managing the database schema, and most of these features are familiar to people who have used object-relational mapping (ORM) technologies like those implemented in Ruby on Rails, CakePHP, LINQ or other libraries and frameworks that abstract access to the database. The main benefit to this approach is that it eliminates the development time otherwise necessary to

write complex SQL queries and manually handle the SQL and output of those operations.
This is a picture of how to build a database in Xcode with the Framework Coredata:



Picture 2 - 6. Picture using Coredata Framework

Furthermore, we will use a class called NSCoder. This class will be very helpful because with this we will able to manage the data from last connection. It means that we will have the option "autoconnect" and it will allow the user close the app and some time after run it again, being able to connect automatically as long as this option is on.

What is NSCoder?
The NSCoder abstract class declares the interface used by concrete subclasses to transfer objects and other Objective-C data items between memory and some other format. This capability provides the basis for archiving (where objects and data items are stored on disk) and distribution (where objects and data items are copied between different processes or threads). The concrete subclasses provided by Foundation for these purposes are NSArchiver, NSUnarchiver, NSKeyedArchiver, NSKeyedUnarchiver, and NSPortCoder. Concrete subclasses of NSCoder are referred to in general as coder classes, and instances of these classes as coder objects (or simply coders). A coder object that can only encode values is referred to as an encoder object, and one that can only decode values as a decoder object.
NSCoder operates on objects, scalars, C arrays, structures, and strings, and on pointers to these types. It does not handle types whose implementation varies across platforms, such as union, void *, function pointers, and long chains of pointers. A coder object stores object type information along with the data, so an

object decoded from a stream of bytes is normally of the same class as the object that was originally encoded into the stream.

Specifically we will use this methods:

*NSKeyedUnarchiver- to get the data stored.
*NSKeyedArchiver- to store the data.

2.5) CONNECTION TO SERVER

We have seen on Internet some pages about connecting iphone via SSH with a laptop and following the instructions of the next link we allow the connection:
http://sites.google.com/site/olipion/cross-compilation/libssh2

Summarizing we have to incluye the following libraries:

- libssh2.a
- libgpg-error.a
- libgcrypt.a
- libz.a

The problem is that if we use this library, the app did not work in iphone's that do not have the Jailbreak done. This happens because apple does not allow the use of libraries which are not implemented by them, so that they control and guarantee a better security of the applications in the applestore.

2.6) iPHONE DEVICE TESTING

When we programmed to iPhone, we need to test our applications on the device. So, to test it, iPhone SDK provides us an iPhone simulator to test and debug your application without having the physical device.

Due to the use of a simulator that has several differences from the physical device, one of them, the most important is that the code is compiled on different architectures. The iPhone uses ARM, and the simulator, Intel x86. Also an important difference is that the simulator does not have accelerometers, and therefore, this feature can only be tested on a real iPhone. But, fortunately we are not going to use it.

The SDK offers to try the iPhone like a Simulator in our laptop. This means we can debug and show the display log of the application on the computer screen while running the application on the iPhone. Thus, we can pause the application and go between instrucctions finding errors. Of course, this requires a USB connection between device and computer.

This feature is possible having the iPhone developer certificate. If you want to try the app in the iPhone device, you must pay the Apple license, which also includes the ability to upload applications to the App Store.

In short, developing for iPhone have an easy and another difficult part.
The easy one, without a doubt, the SDK provided by Apple and the vast amount of documentation and information specific by Apple and third parties.
On the other hand, the hard part is the new programming language and its rules of memory management, threads, etc.. required to spend many more hours than planned to learn and use.

2.7) REQUIREMENTS

HARDWARE REQUERIMENTS:

iPhone application development has minimum hardware requirements without which iPhone programming cannot begin.
In the next paragraphs we will list the essential hardware needed for iPhone development and also mentions the alternatives wherever possible.

Mac Machine:

One of the first hardware any budding iPhone developer should procure is an Intel-based Mac machine or Mac Book because iPhone applications can only be developed using Apple X OS. Does this mean non-Intel based Mac machines cannot be used for iPhone application development? Yes, spot on.

Will a Power PC Mac work for iPhone development?

A Power PC Mac will work but the output will be sluggish, to say the least and result in extremely low productivity. An iBook will be worse, so better stick to a powerful Intel-based Mac PC for your iPhone application development. In spite of this, if you still want to use PowerPC make sure it is running Leopard 10.5.4 or higher.

Old or new Mac?

You can take your pick between a Mac Book (or Mac Mini) and a Mac PC as XCode and Interface Builder run smoothly on both machines. Whichever Mac you choose, ensure it has at least 2GB RAM for smooth performance.

iPhone:

iPhone applications can also be developed using the iPhone Simulator that comes bundled with the iPhone SDK (Software Development Kit) but the problem arises when you want to test the application's GPS functionality or access the internet

from within the application. In such cases the iPhone simulator is not enough and you have no option left but to purchase an iPhone. But if you can somehow make do without GPS and internet, the iPhone Simulator for iPhone application development will do just fine.
Once you have this minimum hardware ready you are all set to create great iPhone applications and games.

SOFTWARE REQUERIMENTS:

- XCode is the complete toolset for building Mac OS X and iOS applications — and with Xcode 4, the tools have been redesigned to be faster, easier to use, and more helpful than ever before.

- Adobe Photoshop (or similar bitmap and/or vector based image editing software)

OTHER REQUIREMENTS:

- $90 to get the developer license and check the app in our device.

3) SSH Client Specification : eSSH

3.1) NAMING THE PROJECT

We have decided to name our application as “easySSH”, in short, “eSSH”.

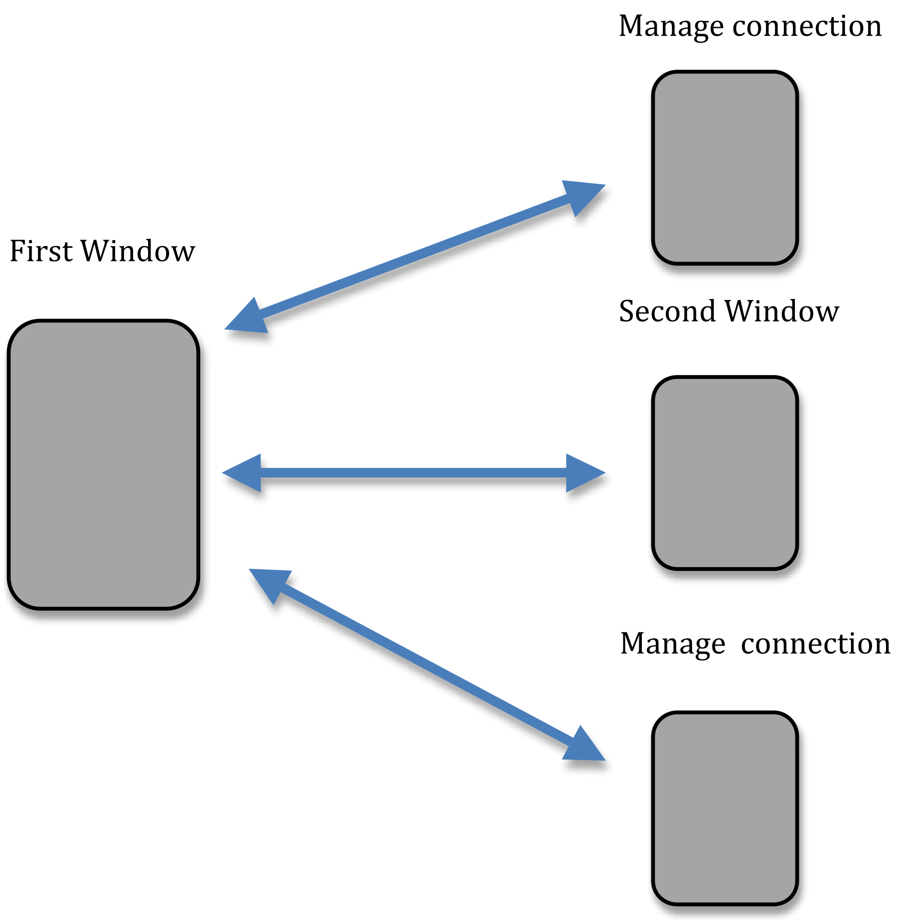
Furthermore, to make eSSH more beautiful and attractive, we have edited the following icons (128X128) using photoshop:

Picture 3 - 1. Icon 1

Picture 3-2. Icon 2

Which one is better? We did a small poll among some potencial users and we have finally decided to choose the first picture.

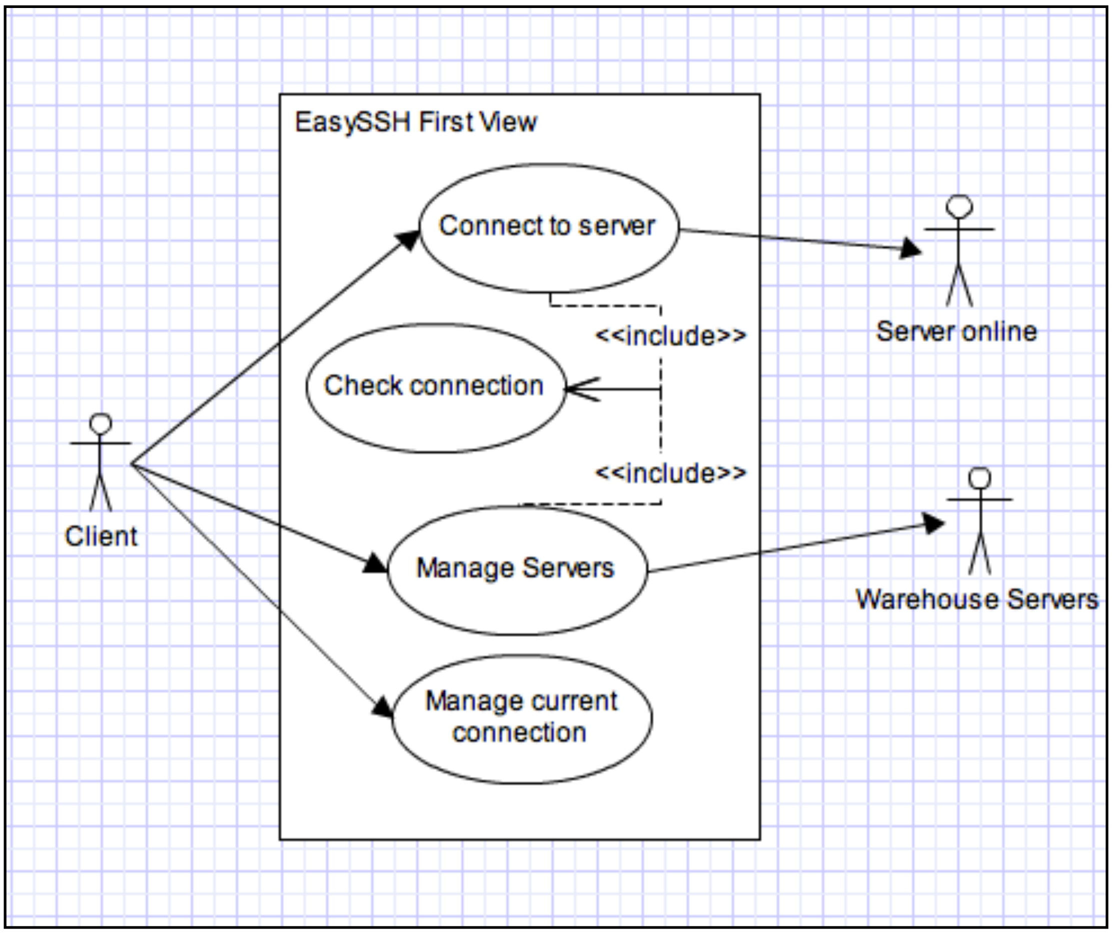
Now we are going to make a sketch to know how much views we will need:



3.1) USE CASE DIAGRAM

We are going to build an use case diagram for all of this views and show which functions we are able to use in each one.

First View diagram:



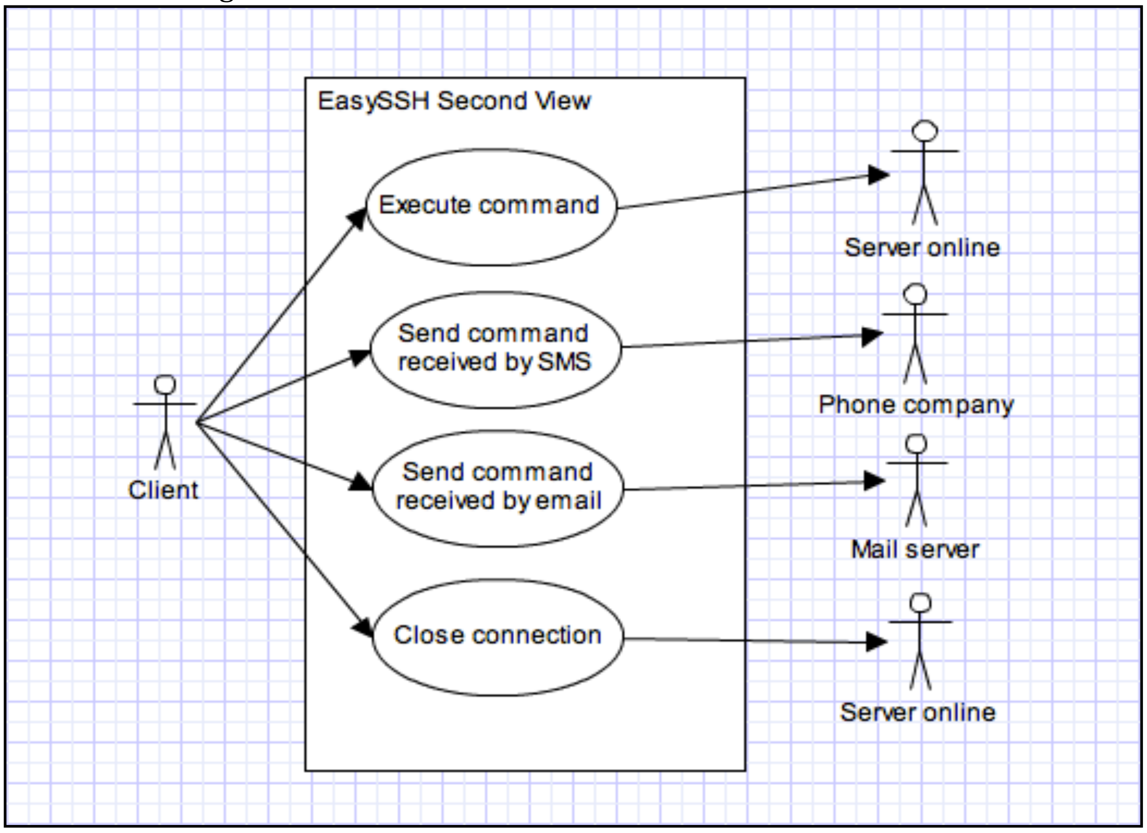
Picture 3 – 3. First view case use diagram.

If the client clicks on "Connect to server", the application will check the connection online and once connected in the correct way, the iphone will show the SecondView. In the next page we will show the diagram of this one.

Furthermore, the user have the option to manage the previously servers stored in a Warehouse of servers and manage them (inserting, deleting and cheking online). So in the next page also, we will show the diagram of Manage Servers.

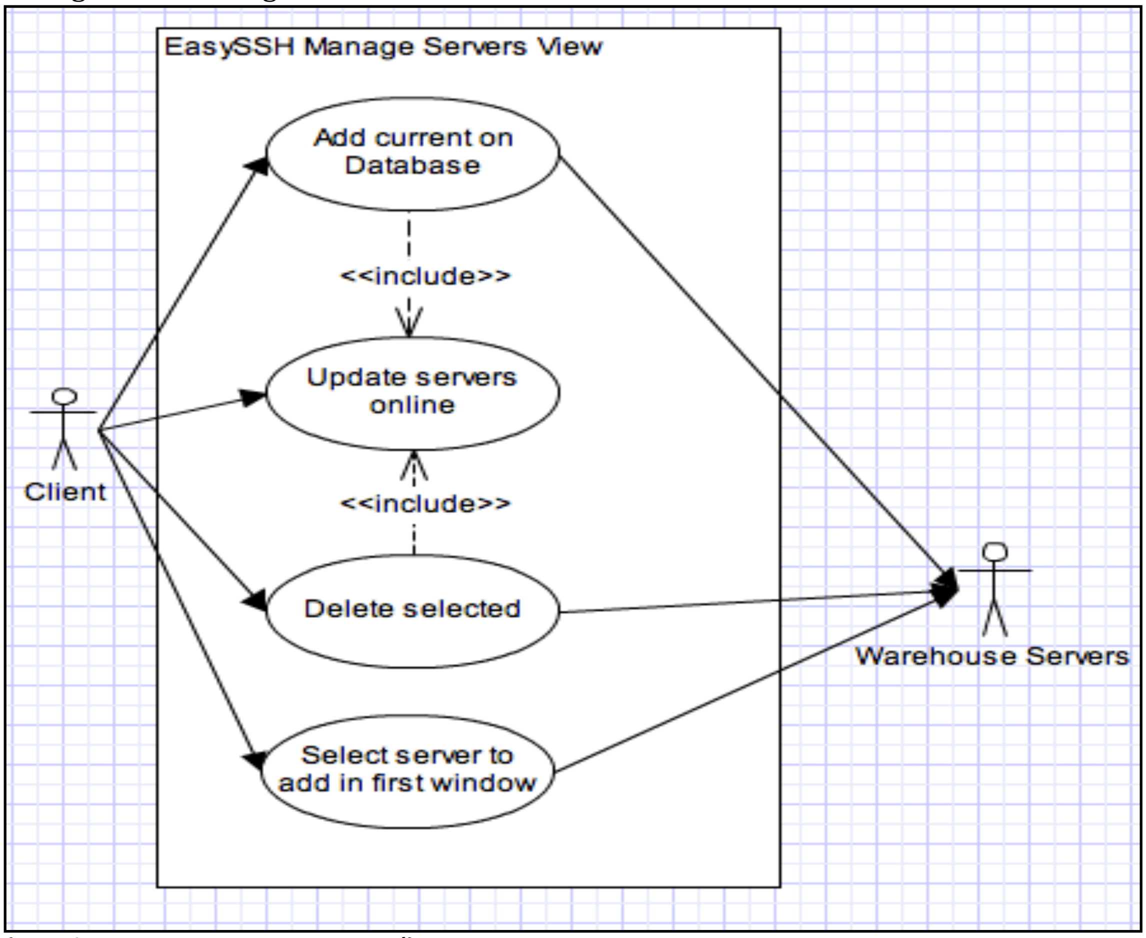
Finally, also in this window, the user can configure the options of the connection. This happen because we don't know the properties of the server: For example which operating system, ...etc This must to be configured because in Windows we use for example "ipconfig" and in Linux "ifconfig", so we must choose one of them before connecting because we will have buttons of quickly running like “ifconfig” and the app must to know if it’s possible to execute it.

As we can see, each paragraph above belong to one view showed below.
Second View diagram:



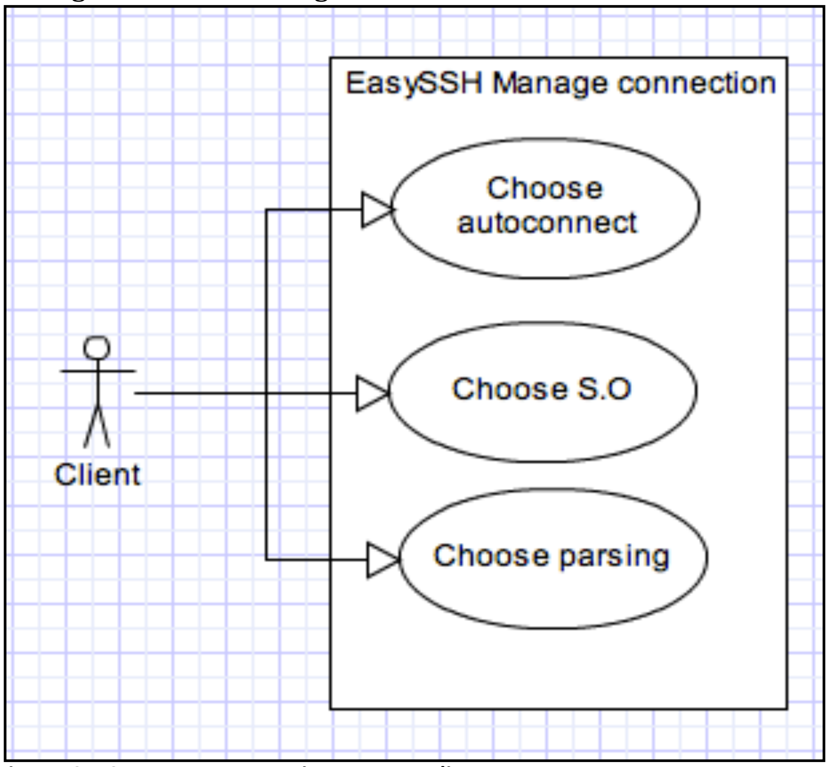
Picture 3 – 4. Second view case use diagram.

Manage Servers diagram:



Picture 3 – 5. Manage servers case use diagram.

Manage Connection diagram:

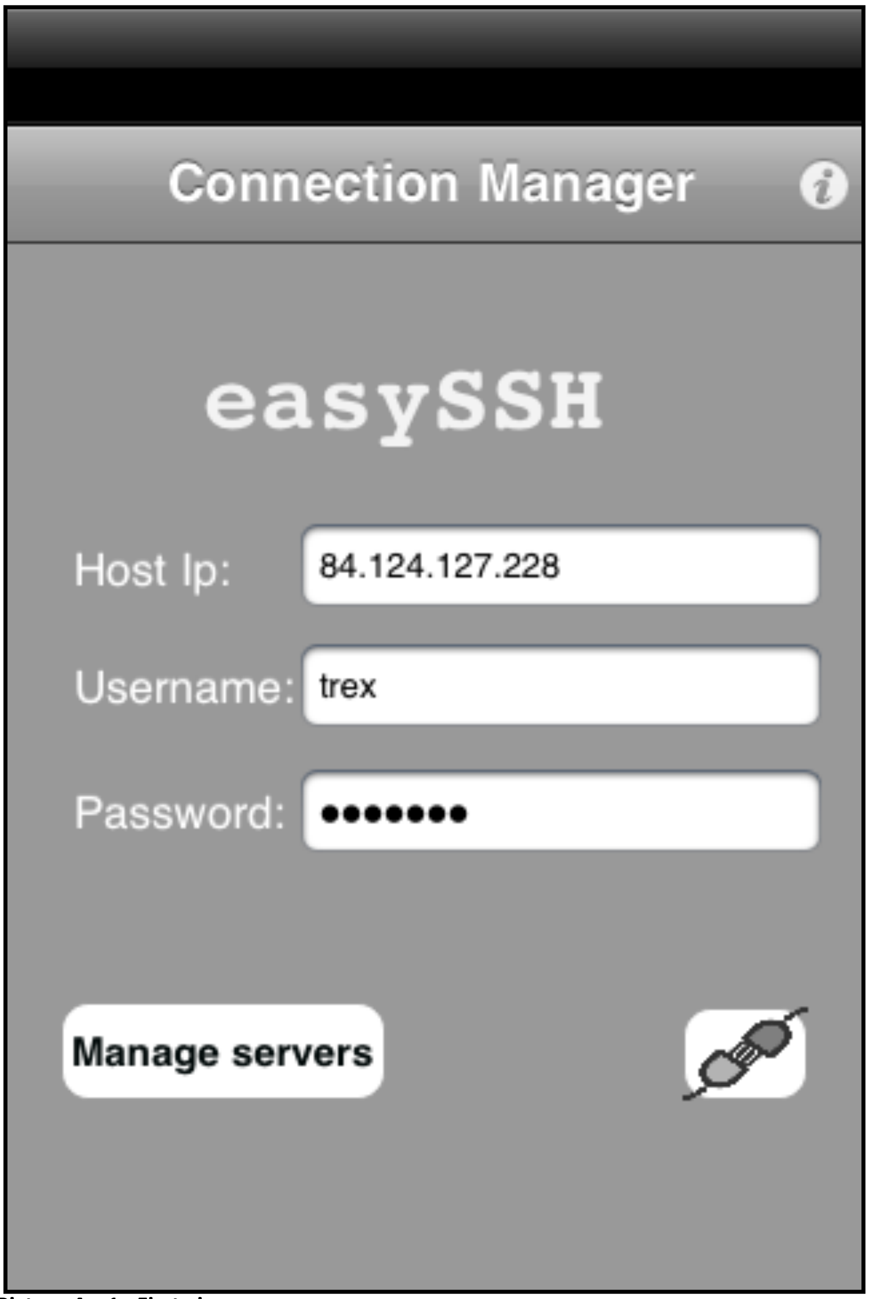


Picture 3 – 6. Manage connection case use diagram.

4) Implementation

4.1) VIEWS

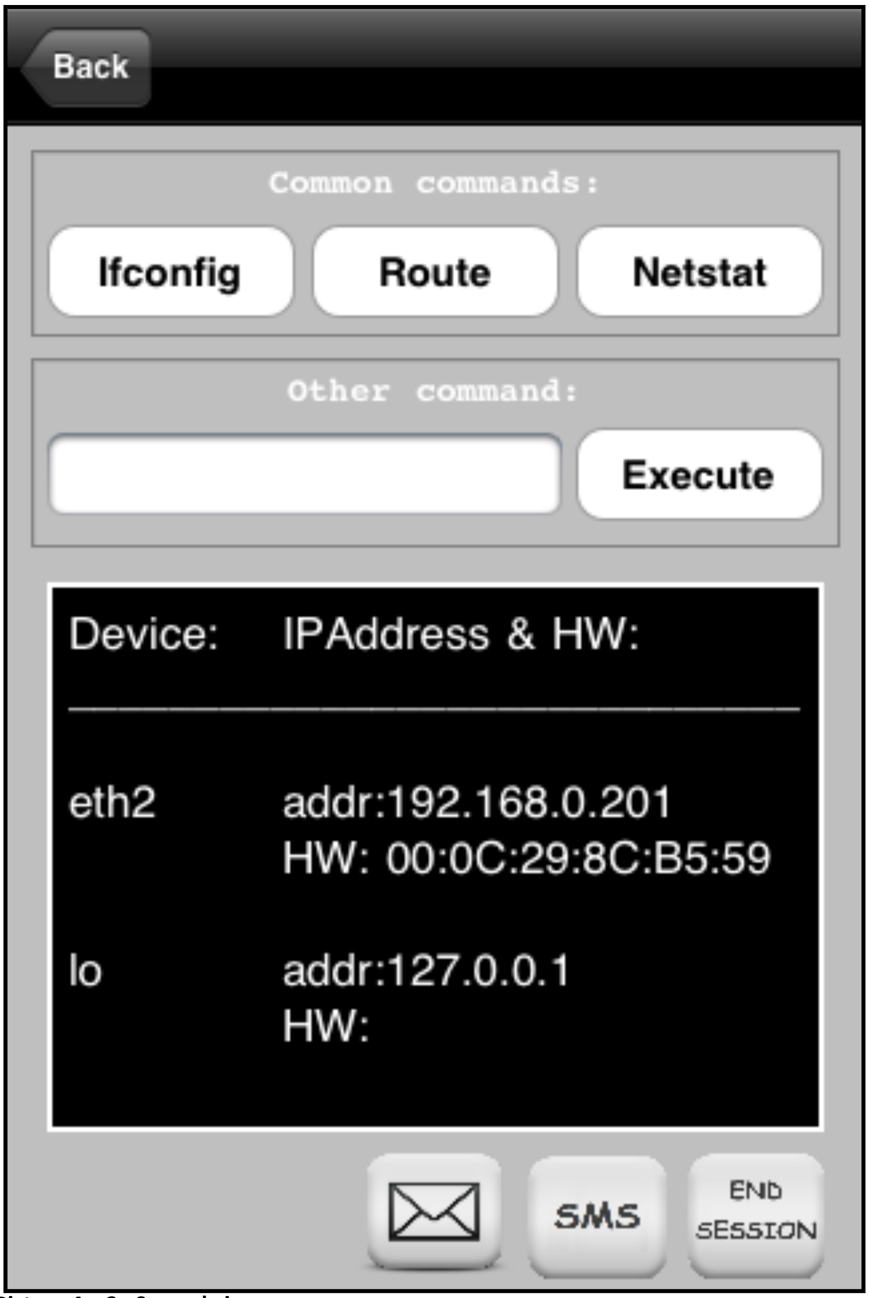
-First Window- This is the first view showed alter launching our app. We can see the title of our app followed by three Textfields (ip, user and password) which are used to identify our connection. After this, we have added three UIbuttons (connect, manage servers and server options which have been talked above in UML diagrams).



84.124.127.228
trex
●●●●●●●●

Picture 4 – 1. First view.

- Second Window - This view will appear alter connecting to any server online (pressing connect). Here we can see some UIButtons. The three buttons in the upside, each one, referente some command (ifconfig, route or netstat) or we have one button joined with a UITextField which both referentes some other command introduced by desire of the user, which is different from the other three previous commands.
Besides, the most part of the view is busy by the UITextView, which is used to show the answers of the server.



```
Device:     IPAddress & HW:
______________________________

eth2        addr:192.168.0.201
            HW: 00:0C:29:8C:B5:59

lo          addr:127.0.0.1
            HW:
```

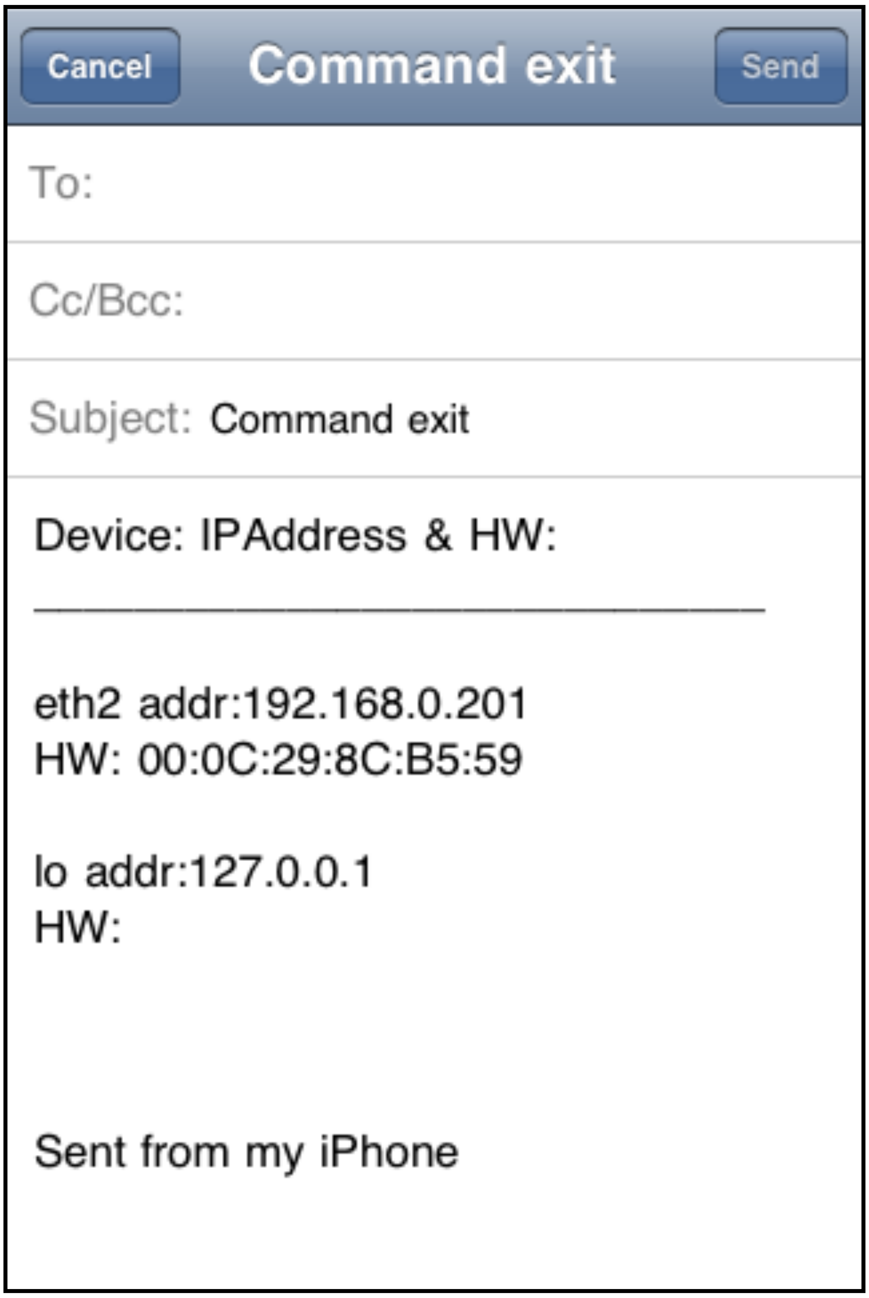
Picture 4 – 2. Second view.

Also, in the bottom, we can see other three UIButtons. Each one have some function which we are going to specifify in the next page.

Picture 4 – 3. Icon mail.

This button is used to send the answer received by email.

Let's see an example when we click on this button.



Command exit

Device: IPAddress & HW:
____

eth2 addr:192.168.0.201
HW: 00:0C:29:8C:B5:59

lo addr:127.0.0.1
HW:

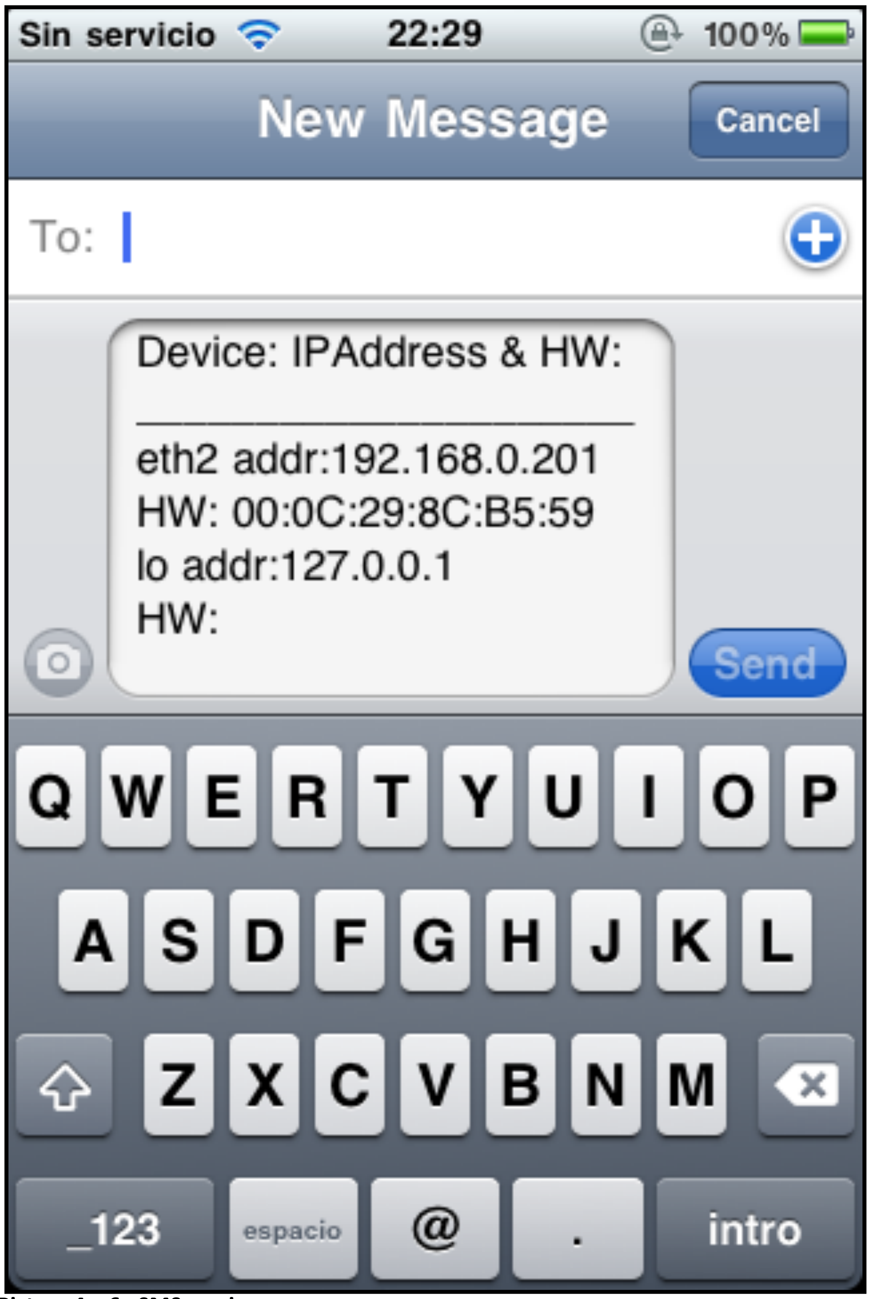
Sent from my iPhone

Picture 4 – 4. Mail service.

Picture 4 – 5. Icon SMS.

This is also the same , but the difference is the protocol to send the answer: via SMS.

Let's take an example:



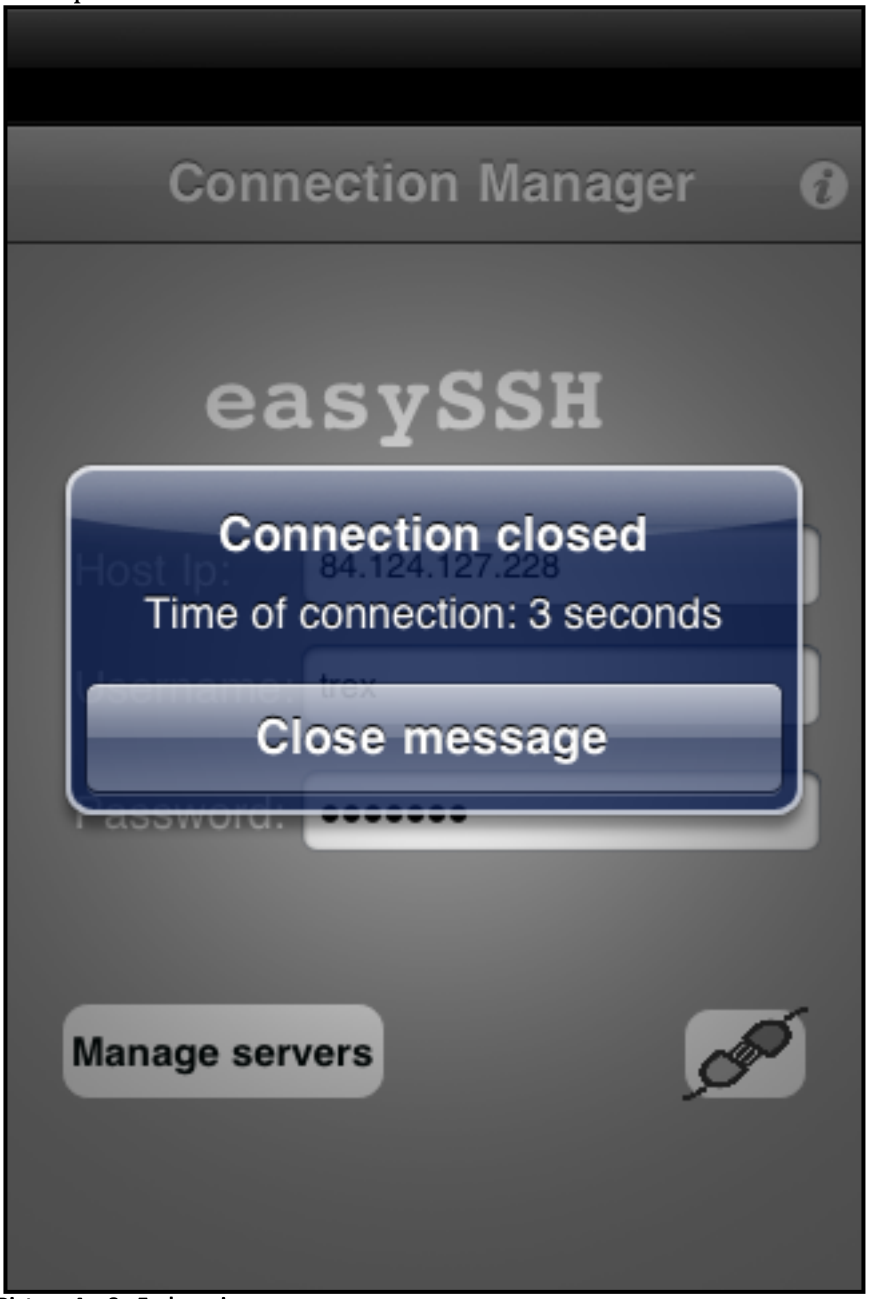
```
Device: IPAddress & HW:
________________________
eth2 addr:192.168.0.201
HW: 00:0C:29:8C:B5:59
lo addr:127.0.0.1
HW:
```

Picture 4 – 6. SMS service.

Picture 4 – 7. Icon end session.

The functionality of the third and the last button is simple: is used to close the connecton, show the connection time and access to the first window.

Example:



Connection Manager

easySSH

Connection closed

Time of connection: 3 seconds

Host Ip: 84.124.127.228

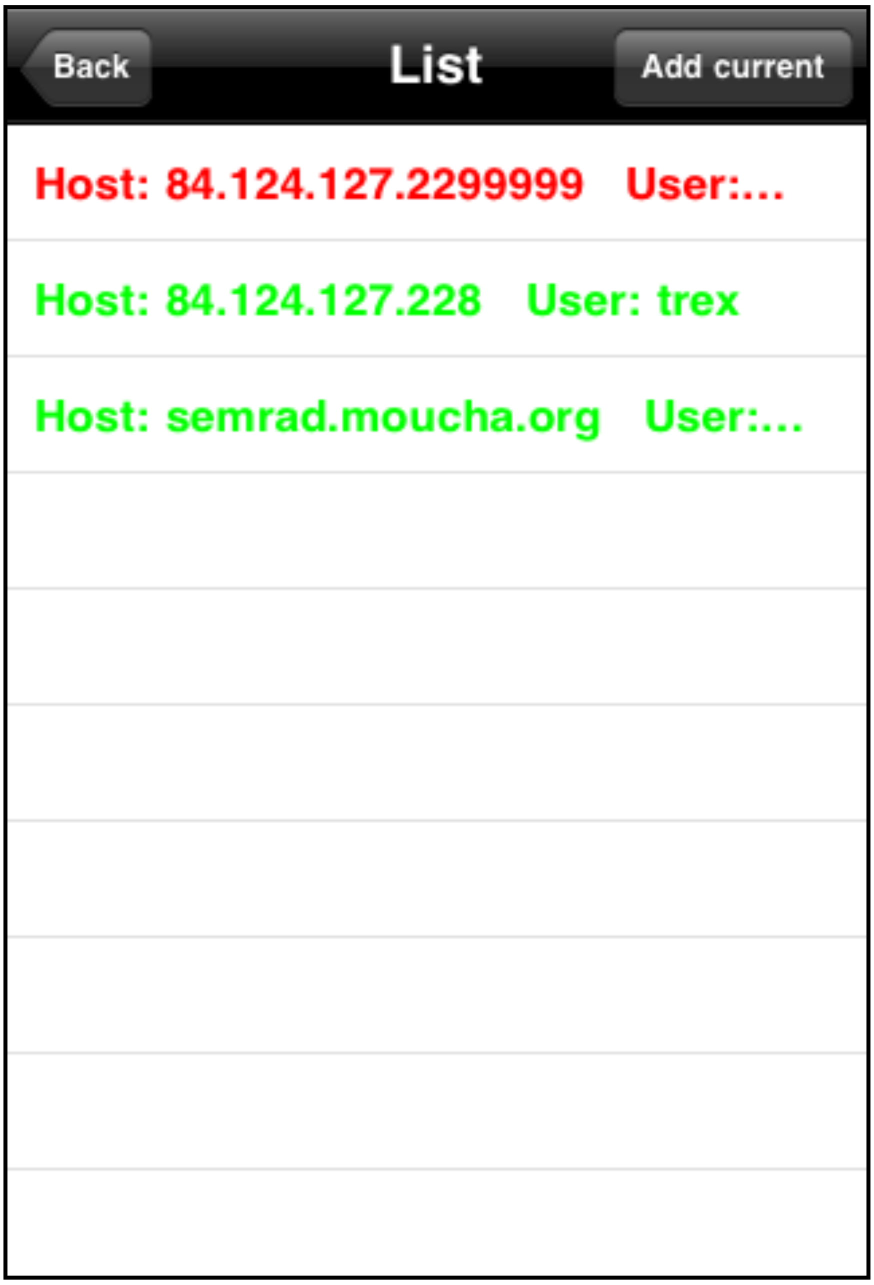
Username: trex

Password: ●●●●●●●

Picture 4 – 8. End session.

-Manage Servers- This view is used to manage the servers addings views, deleting old servers, and selecting someone with the aim to put the information in the TextFields(ip,user pass) allocated on the Firstview, to avoid to introduce him manually. Furthermore, we can see in green color the servers online at the current time and in red color the servers offline. This way, we avoid to connect to servers offline or we can check if a server is online quickly.

Here we can see only the button "add current" which adds the server introduced in the firstview. All fields need to be written, not empty. If its empty will show an error.
To delete a server the client must slide the finger in the line of some server and will appear a delete button in red color.
To select one, we only must click on him.



Host: 84.124.127.2299999 User:...

Host: 84.124.127.228 User: trex

Host: semrad.moucha.org User:...

Picture 4 – 9. Listing servers stored.

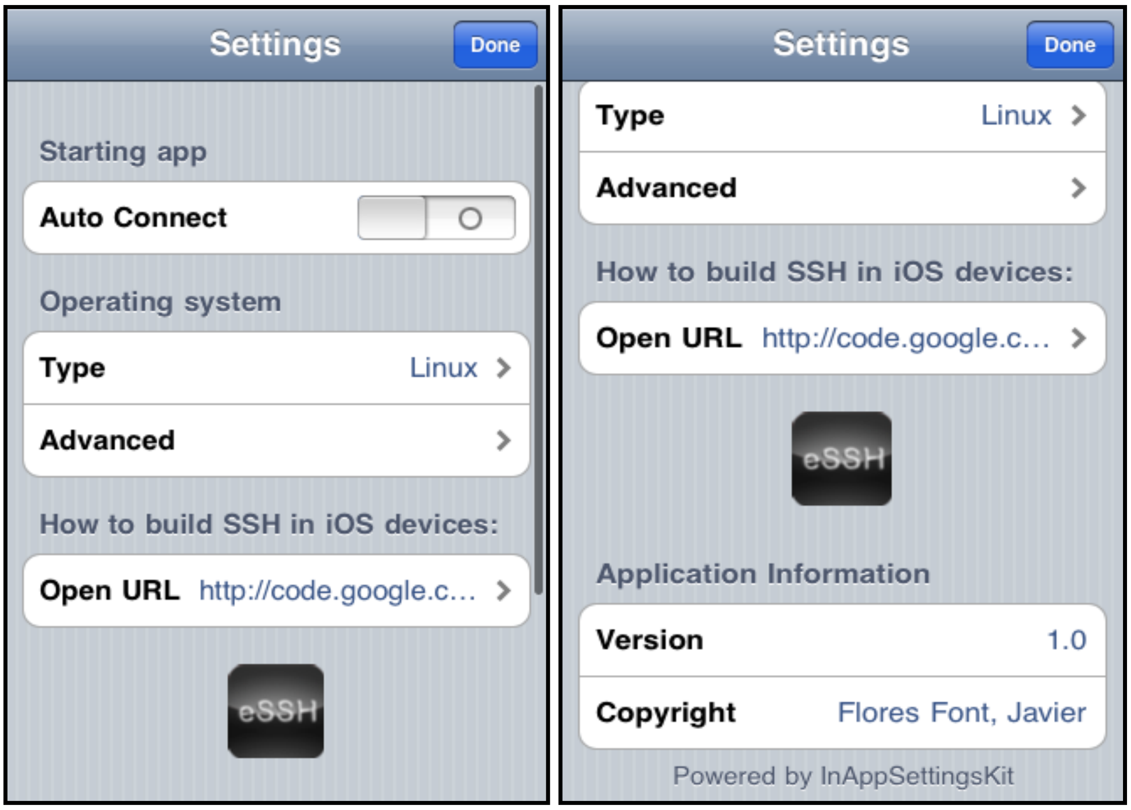
-Manage Connection-

This view is used to configure the settings of the current connection. To manage them we have used a open source code called InAppSettingsKit.

InAppSettingsKit will be explained in the point 4.5)

This is the result of using InAppSettingsKit:

If in the first picture we slide the finger from bottom to top we can see the second picture.



Starting app

Auto Connect

Operating system

Type Linux

Advanced

How to build SSH in iOS devices:

Open URL http://code.google.c…

Application Information

Version 1.0

Copyright Flores Font, Javier

Powered by InAppSettingsKit

Picture 4 – 10. Settings.

4.2) CONNECTION

To get the ssh connection working properly in our iPhone, this code have been the worst part of developing in this project. Install the correct libraries, put the correct path's , and once done, read which code to use, all of this have taken most part of the time. We think this part is so important so we are going to try to explain some functions of the code which we consider are the most remarkables.

4.2.1) ESTABLISHING THE CONNECTION

```
- (IBAction) ConnectbySSH
     {
 NSString *localPath1 = @"Documents/my1.archive";
 NSString *fullPath1 = [NSHomeDirectory()
stringByAppendingPathComponent:localPath1];
[NSKeyedArchiver archiveRootObject:Ip toFile:fullPath1];

 NSString *localPath2 = @"Documents/my2.archive";
 NSString *fullPath2 = [NSHomeDirectory()
stringByAppendingPathComponent:localPath2];
[NSKeyedArchiver archiveRootObject:Name toFile:fullPath2];

 NSString *localPath3 = @"Documents/my3.archive";
 NSString *fullPath3 = [NSHomeDirectory()
stringByAppendingPathComponent:localPath3];
[NSKeyedArchiver archiveRootObject:passwordField
toFile:fullPath3];

sshConnection1 = [[SSHConnection alloc] init];
int x;
x=[sshConnection1 connectToHost:Ip.text port:22 user:Name.text
password:passwordField.text];

if(x==-1){
     UIAlertView *progressAlert = [[UIAlertView alloc]
     initWithTitle: @"Connection failed"
     message: @"Try other server."
     delegate: self
     cancelButtonTitle: @"Ok"
     otherButtonTitles: nil];
     [progressAlert show];
             [progressAlert release];
     }else {
         SecondWindow *controller = [SecondWindow alloc];
         [[self navigationController]
         pushViewController:controller
     animated:YES];
         [controller release];
         }
}}
```

In the code above we can observe three kinds of color. Each one representes some functionality which is going to be explained in the next pharagraps:

Color green: it's used to save the data of the last server used in memory. With this code we are able to use the setting "autoconnect". If the boolean of "autoconnect" is True, the application access to the information of the last server used, stored previously in the iphone with this code.

Color blue: Once we have stored the data of the last server used, the next step is to check the connection. Thus, we create an object SSHConnection and we try the connection passing the arguments: Ip,User and password.

Color grey: The variable "x" store the result of the connection. If the value of "x" is -1 , the server is not avaliable or some data is not correct. So, if this happens we print an UIDialogView which shows "Connection failed". If the cheking of the connection is ok, we will show the next view with the function
[[self navigationController] pushViewController: controller animated:YES];

4.2.1) EXECUTING A COMMAND

Once connected, we are able to execute commands which will be showed in a UITextView.

Executing some command in the remote server is easy, we only must execute this function:

```
NSString *salida=[sshConnection executeCommand:@"Some command"];
```

Furthermore, we need to show the answer in the UITextView(called response), so to save the answer in response the code is quite simple:

```
response.text =salida;
```

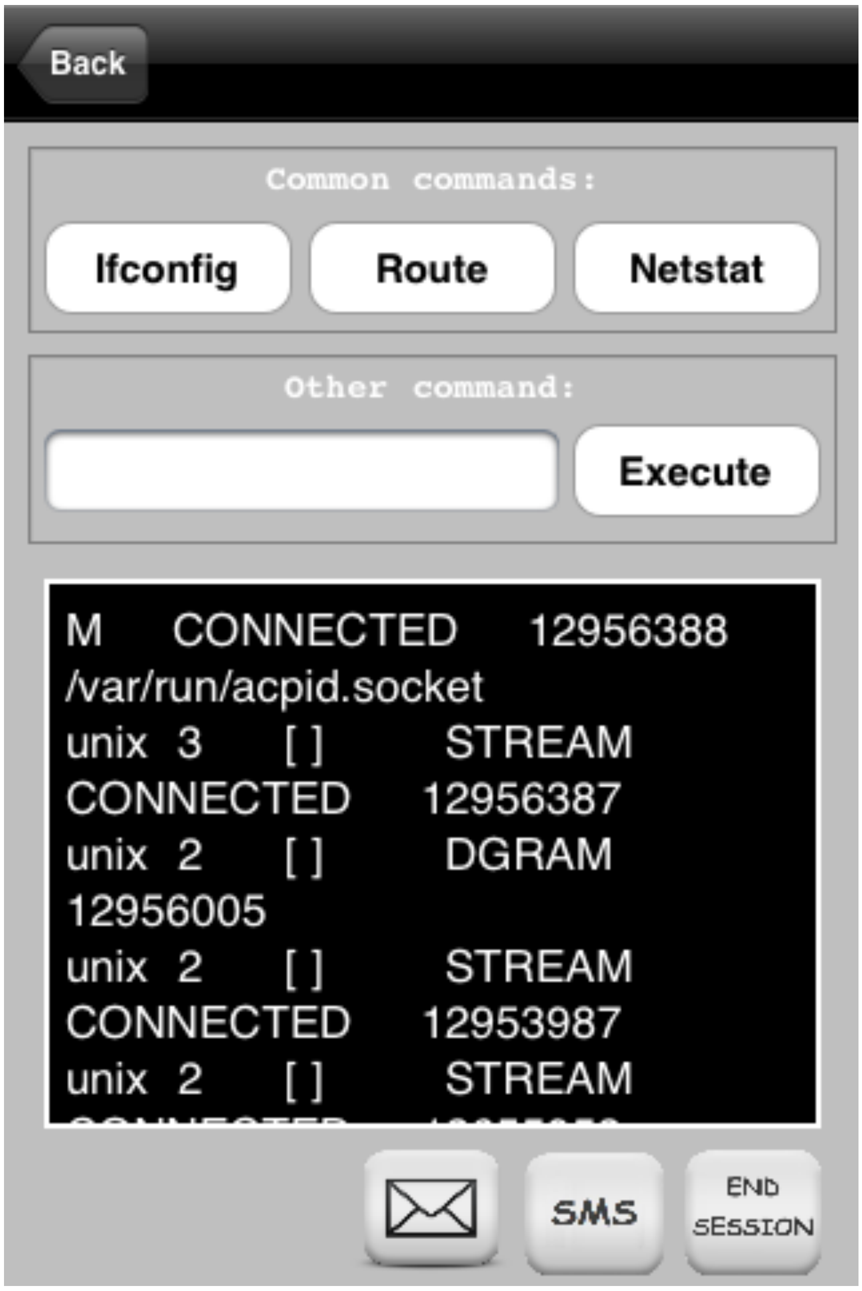
In the next pharagraps we will show the application executing some commands and the answers of them, specifically the command "ifconfig" or "ipconfig".

4.2.1.1) Command: NETSTAT

```
- (IBAction) Netstat
{

 NSString *salida=[sshConnection executeCommand:@"netstat"];
response.text =salida;

}
```



```
M     CONNECTED      12956388
/var/run/acpid.socket
unix  3      [ ]        STREAM
CONNECTED     12956387
unix  2      [ ]        DGRAM
12956005
unix  2      [ ]        STREAM
CONNECTED     12953987
unix  2      [ ]        STREAM
```

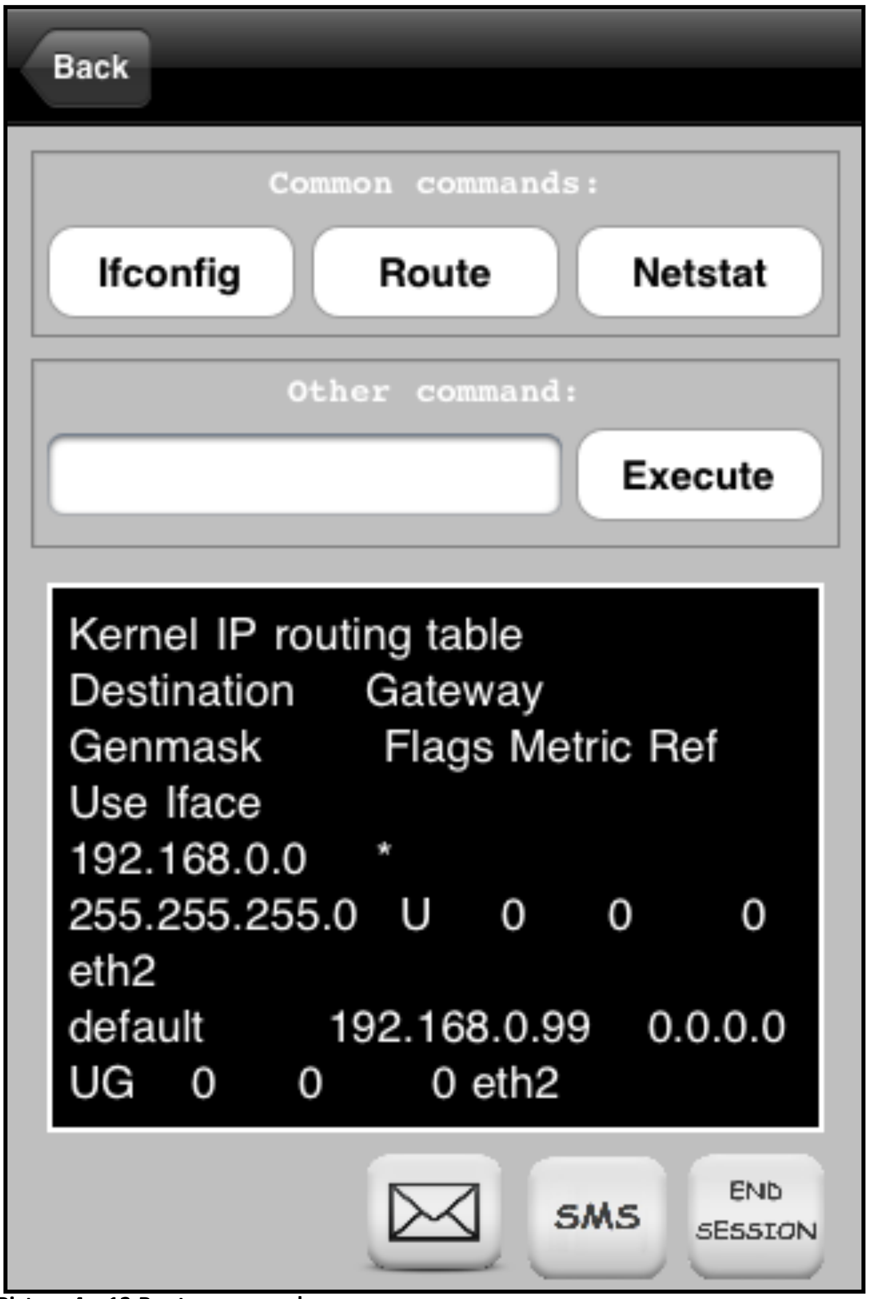
Picture 4 – 11. Netstat command

4.2.1.2) Command: ROUTE

```
- (IBAction) Route
{

 NSString *salida=[sshConnection executeCommand:@"route"];
response.text =salida;

}
```



```
Kernel IP routing table
Destination     Gateway
Genmask         Flags Metric Ref
Use Iface
192.168.0.0     *
255.255.255.0   U     0      0      0
eth2
default         192.168.0.99   0.0.0.0
UG     0      0      0 eth2
```

Picture 4 – 12.Route command.

4.2.1.3) command: IFCONFIG or IPCONFIG

In this command, the answer can change independing in one property of the settings: the parsing.

If the parsing is active we will show only each device , the IP address and HW of this device.
If not, we will show the answer like executing in some computer.

How to parse? We have done manually the code and is quite complex. This code is inside of the funcion ifconfig, this is the result:

```
- (IBAction) Ifconfig
{
     int i=0;
    int k=0;
    char dispositivo[100];
    NSString *sistemValue = [[NSUserDefaults
    standardUserDefaults] stringForKey:@"mulValue"];
    NSString *salida;
if([sistemValue isEqualToString:@"0"]){
    salida=[sshConnection executeCommand:@"ipconfig"];
    NSLog(@"sistemvalue = %@", sistemValue);
    }else
    salida= [sshConnection executeCommand:@"ifconfig"];
    NSString *toogleValue = [[NSUserDefaults
standardUserDefaults] stringForKey:@"switchOther"];
    if([toogleValue isEqualToString:@"1"]){
    const char *salida2 = [salida UTF8String];

    char palabra_siguiente[100];
    int longpal=0;

    int numero_disps=0;

    char dispositivos[100][100];
    char inetaddr[100][100];
    Boolean checkinetaddr[100];
    char HW[100][100];
    Boolean checkHW[100];

    for(i=0;i<100;i++){
        checkinetaddr[i]=true;
        checkHW[i]=true;
    }


    bool palabra=false; //para ver si esta leyendo palabra
    for(k=0;k<strlen(salida2);k++) {
```

```
    if(salida2[k]==' ' || salida2[k]=='\t' ||
salida2[k]=='\n'){ //no se lee palabra
    if(palabra==true){ //si estaba leyend palabra
    palabra=false; //deja de leer palabra
    palabra_siguiente[longpal]='\0';
    if(!strcmp("Link", palabra_siguiente)){
     printf("dispositivo: %s\n", dispositivo);

strcpy(dispositivos[numero_disps],dispositivo);
                    numero_disps++;

     }
else if(strlen(palabra_siguiente)>10 &&
palabra_siguiente[0]=='a'
&& palabra_siguiente[1]=='d' && palabra_siguiente[2]=='d'
    && palabra_siguiente[3]=='r' && palabra_siguiente[4]==':'){
      strcpy(inetaddr[numero_disps-1],palabra_siguiente);
      checkinetaddr[numero_disps-1]=true;
    }
else if(!strcmp("inet", palabra_siguiente) &&
  strlen(dispositivo)==17){
      strcpy(HW[numero_disps-1],dispositivo);
      checkHW[numero_disps-1]=true;
    }
    strcpy(dispositivo,palabra_siguiente);
    longpal=0;
    }
    }
    else{
            if(palabra==false){
                palabra_siguiente[longpal]=salida2[k];
                longpal++;
                palabra=true;}
            else {
                palabra_siguiente[longpal]=salida2[k];
                longpal++; }
        }
    }
    NSString *aa;
    [response setText:@" "];
char dispositivoaux[500]="Device:\tIPAddress &
HW:\n_____________________________\n\n";

    for(i=0;i<numero_disps;i++){
        printf("%d",i);
        printf("%s",dispositivos[i]);
        strcat(dispositivoaux,dispositivos[i]);
        strcat(dispositivoaux,"\t\t");
        if(checkinetaddr[numero_disps]==true){
            strcat(dispositivoaux,inetaddr[i]);
```

```
                strcat(dispositivoaux,"\n");}

            if(checkHW[numero_disps]==true){
                strcat(dispositivoaux,"\t\tHW: ");
                strcat(dispositivoaux,HW[i]);
            }
            strcat(dispositivoaux,"\n\n");
        }

        printf("%s",dispositivoaux);
        aa = [NSString stringWithCString: (const char
        *)dispositivoaux encoding: NSASCIIStringEncoding];
        response.text = aa;
        }
        else {
            [response setText:@" "];
            response.text=salida;
        }

}
```

Grey code:

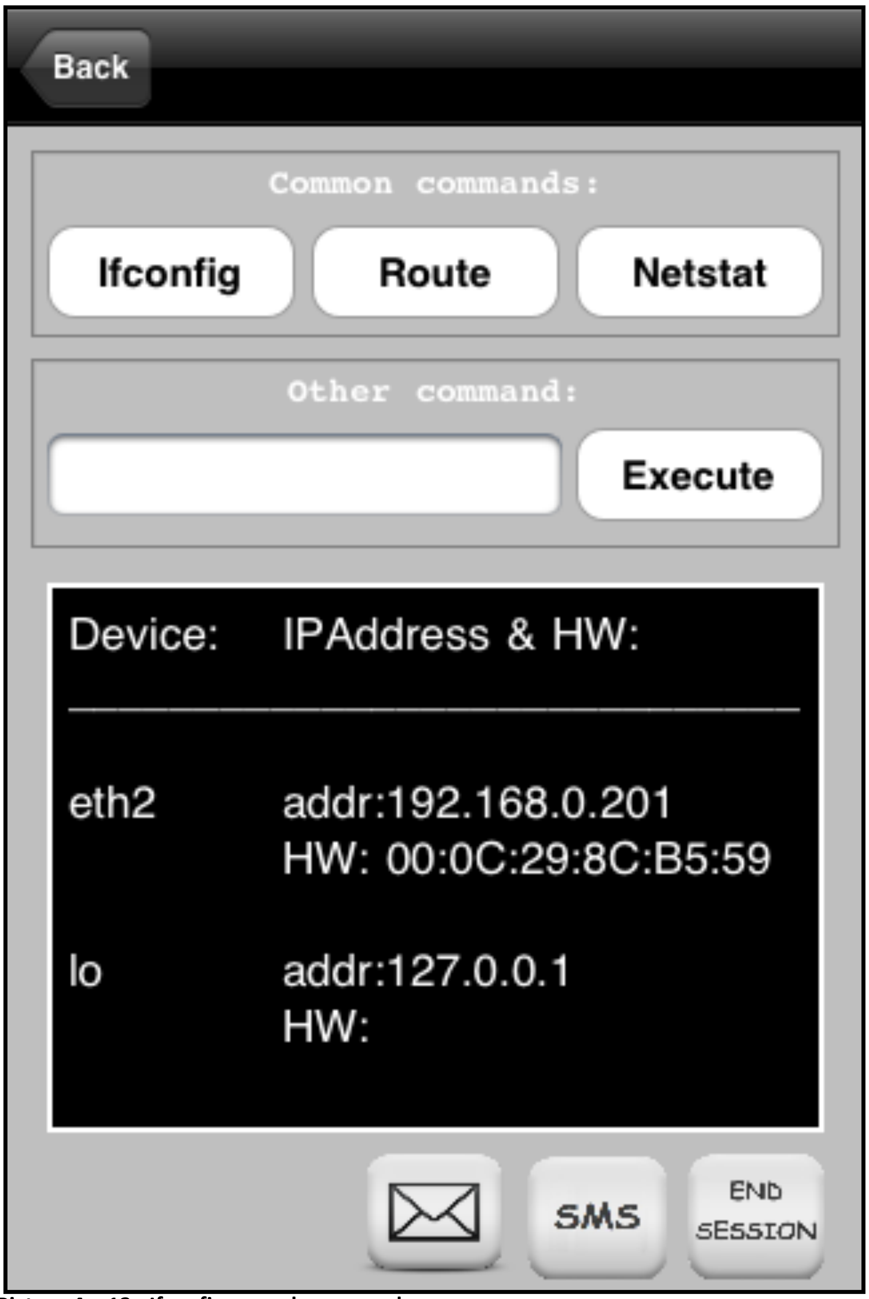
This part is used to check in settings which operating system have been stored by the user previously. This S.O it supposed that is stored correctly by the user. When we say correctly it means that is true that this S.O is exactly the same like the S.O of the server online actually connected.
If the value stored isEqualToString:@"0" it means that the S.O belong to a Windows, so we must execute "ipconfig".
In other hand, if the values are "1" or "2" it means that the current S.O belong to a Linux or Mac Os X, so we must execute "ifconfig".

Green code:

ToggleValue check in settings if the user wants to parse the result or wants to execute ifconfig like always. So, if toogleValue isEqualToString:@"1" we will parse the result, if not, we will not.

In the next page we will show a print screen of the command ifconfig parsed .

Command ifconfig parsed:



```
Device:    IPAddress & HW:
_______________________________

eth2       addr:192.168.0.201
           HW: 00:0C:29:8C:B5:59

lo         addr:127.0.0.1
           HW:
```

Picture 4 – 13. Ifconfig parsed command.

Things to comment of this print screen:

-Here the information is more clever than not parsing.

-We show each device with his own IP address and HW.

-The device Io do not have HW address, so this appear empty.

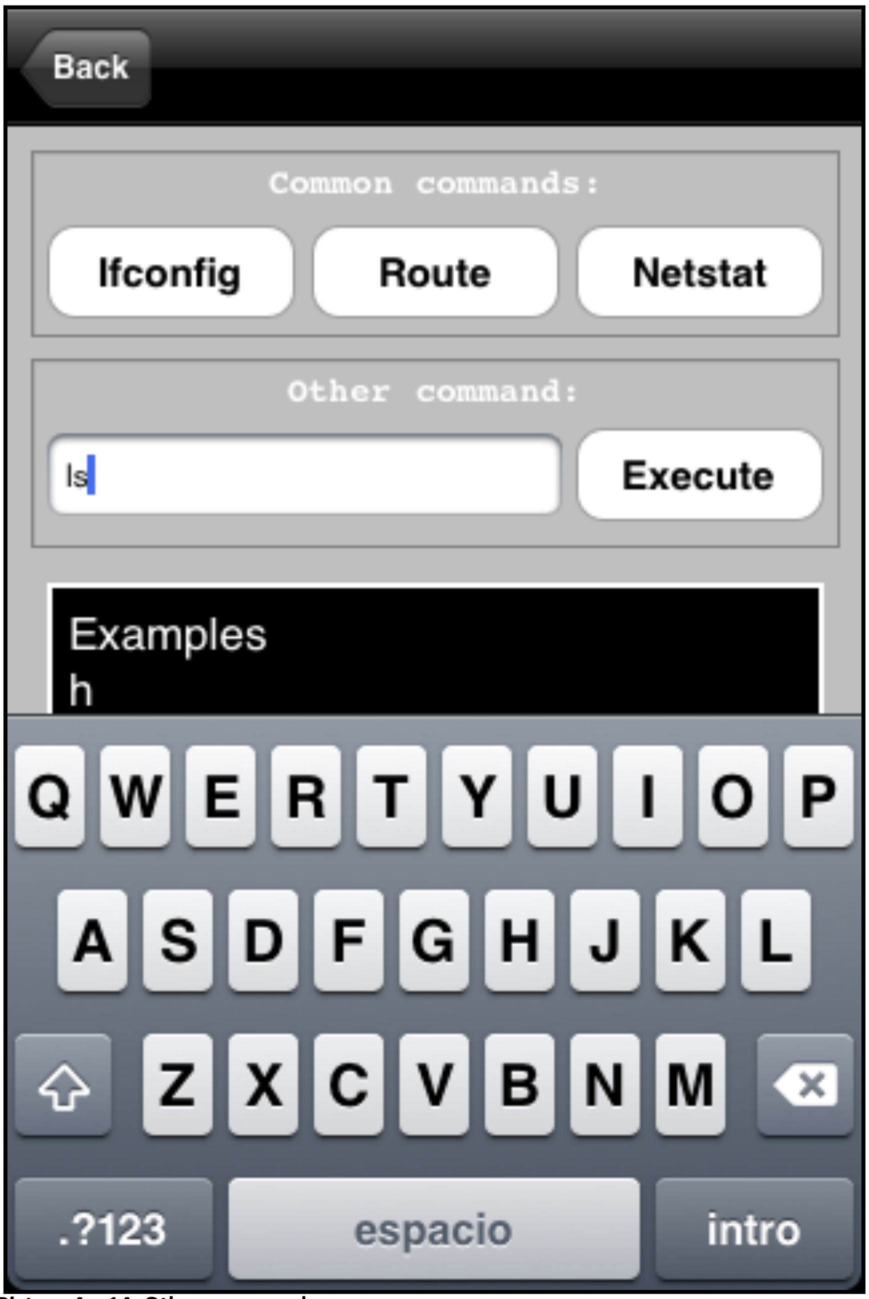
4.2.1.4) Other commands

We have added an UITextField in the second view to introduce manually some command desired by the user.
We can see how the function “OtherCommand” reads this UITextField to get the command introduded by the user:

```
- (IBAction) OtherCommand
{

            NSString *salida=[sshConnection
executeCommand:othercommand.text];;
    response.text =salida;
}
```

The result is this one:



```
Examples
h
```

Picture 4 – 14. Other command.

We have chosen the command “ls” which shows the files or folders in the current directory. In the example we can see that we have two files or folders: “Examples” and “h”.

4.2.1) CLOSING CONNECTION

```
- (IBAction) CloseConnection

{
   [sshConnection closeConnection];
   [sshConnection release];

    NSString *intString = [NSString stringWithFormat:@"%d",
timercount];
    NSString *closemessage = [NSString
stringWithFormat:@"%@%@%@", @"Time of connection: ",
intString,@" seconds"];
    UIAlertView *progressAlert = [[UIAlertView alloc]
initWithTitle: @"Connection closed"

message: closemessage

delegate: self

cancelButtonTitle: @"Close message"

otherButtonTitles: nil];

    [progressAlert show];
    [progressAlert release];
    [[self
navigationController]popToRootViewControllerAnimated:YES];

}
```

This function is quite simple,

the code call the `closeConnection` method to end session,

show an UIAlertView with connection time: `[progressAlert show];`
`[progressAlert release];`

and change the view for the previously: FirstView

```
 [[self
navigationController]popToRootViewControllerAnimated:YES];
```

4.3) MANAGING THE SERVERS

As we mentioned some pages above, to manage the servers we must know how to save the information of the servers in iPhone. To make this possible we have used the library Coredata. Coredata is the most frequently library used in iPhone actually in application which use databases.

We are going to explain how we can get this library working on iPhone.

First of all, we must add the Framework "Coredata.framework" in the folder Frameworks.

Once done, we only need to store the information of the servers, thus we need only one entity. This entity must to be created in the folder Classes. We have called him like EServer.

```
//  EServer.h
```

```
#import <CoreData/CoreData.h>

@interface EServer :  NSManagedObject
{
}

@property (nonatomic, retain) NSString * kIp;
@property (nonatomic, retain) NSString * kUser;
@property (nonatomic, retain) NSString * kPass;

@end
```

```
//  EServer.m
```

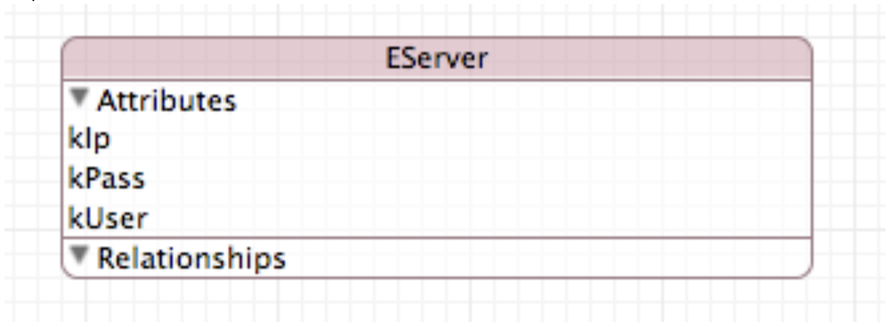
```
#import "EServer.h"

@implementation EServer

@dynamic kIp,kUser,kPass;

@end
```

In the folder resources is necessary to create a xcdatamodel. In this file we shall write how much entities , atributes and other things must to have our database. So, we must create this table:



How to show the servers?

When we click on the UIButton "Manage servers" should appear a table showing in each line the different servers stored. This table will be managed by a controller called UITableViewController.

UITableViewController is a controller pre-configured with a UITableView and with itself set as the table view's delegate and data source.

Here is the definition of this code:

```
#import <UIKit/UIKit.h>
#import "EServer.h"
#import "SSHConnection.h"

@interface ServerTableController : UITableViewController {

    IBOutlet UITextField *Ip;
    IBOutlet UITextField *Name;
    IBOutlet UITextField *passwordField;
    NSManagedObjectContext *managedObjectContext;
    NSMutableArray *serverArray;
    SSHConnection *sshConnection2;

}

@property (nonatomic, retain) NSManagedObjectContext
*managedObjectContext;
@property (nonatomic, retain) NSMutableArray *serverArray;
@property (nonatomic, retain) IBOutlet UITextField *Ip;
@property (nonatomic, retain) IBOutlet UITextField *Name;
@property (nonatomic, retain) IBOutlet UITextField
*passwordField;

- (void)fetchRecords;
- (void)addServer:(id)sender;
- (void)comebackfirstmenu:(id)sender;

@end
```

Things to note here:

- fetchRecords method: used to update all lines of the table every time we open the “manage servers” window , or delete or add some server.

- addServer method: this function access to the database and add the current server introduced in the previous view. All fields must to be completed.

- comebackfirstmenu method: Used to go back to the first view.

In the page above we have showed the methods to addserver,fetch records or comebackfirstmenu, but where is the method used to delete a server? And, where is the method to select some server and put the data in the textfields at the first view?

METHOD USED TO DELETE A SERVER: We want to delete the server when the user slide the finger on some line, so this function is previously defined in `UITableViewController`, and we can take a look at the code below:

```
(void)tableView:(UITableView *)tableView
commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
forRowAtIndexPath:(NSIndexPath *)indexPath
{

NSUInteger row = [indexPath row];
EServer *server = [serverArray objectAtIndex: [indexPath row]];

NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];

NSEntityDescription *entity = [NSEntityDescription
entityForName:@"EServer"
inManagedObjectContext:managedObjectContext];

 [fetchRequest setEntity:entity];

NSError *error;

NSArray *items = [managedObjectContext
executeFetchRequest:fetchRequest error:&error];
[fetchRequest release];

for (NSManagedObject *managedObject in items) {
     if([[managedObject valueForKey:@"kIp"]
     isEqualToString:server.kIp] && [[managedObject
     valueForKey:@"kUser"] isEqualToString:server.kUser]){
        [managedObjectContext deleteObject:managedObject];
      }

}
[serverArray removeObjectAtIndex:row];

if (![managedObjectContext save:&error]) {
          NSLog(@"Error");
     }
     [tableView reloadData];
}
```

The yelow color code is used to delete from the database the selected server.

The green color code is used to delete from the array the selected server.This array is used by the table to show the stored servers more quickly.
METHOD USED TO SELECT A SERVER: Here happens the same as above, the method is predefined by the class UITableViewController.

```
- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath {

    int row = indexPath.row;
    [tableView deselectRowAtIndexPath:indexPath animated:YES];
    NSLog(@"fila seleccionada: %d",row);
    NSFetchRequest *fetchRequest = [[NSFetchRequest alloc]
init];
    NSEntityDescription *entity = [NSEntityDescription
entityForName:@"EServer"
inManagedObjectContext:managedObjectContext];
    [fetchRequest setEntity:entity];
NSError *error;
    NSArray *items = [managedObjectContext
executeFetchRequest:fetchRequest error:&error];
    [fetchRequest release];
    int i=0;

    for (NSManagedObject *managedObject in items) {

        if(i==row){
            NSString *x = [[NSString alloc]init];
            NSString *x1 = [[NSString alloc]init];
            NSString *x2 = [[NSString alloc]init];
            x= [managedObject valueForKey:@"kIp"];
            x1= [managedObject valueForKey:@"kUser"];
            x2= [managedObject valueForKey:@"kPass"];
NSString *localPath1 = @"Documents/my4.archive";
            NSString *fullPath1 = [NSHomeDirectory()
stringByAppendingPathComponent:localPath1];
            [NSKeyedArchiver archiveRootObject:x
toFile:fullPath1];
NSString *localPath2 = @"Documents/my5.archive";
            NSString *fullPath2 = [NSHomeDirectory()
stringByAppendingPathComponent:localPath2];
            [NSKeyedArchiver archiveRootObject:x1
toFile:fullPath2];
            NSString *localPath3 = @"Documents/my6.archive";
            NSString *fullPath3 = [NSHomeDirectory()
stringByAppendingPathComponent:localPath3];
            [NSKeyedArchiver archiveRootObject:x2
toFile:fullPath3];
}
            i++;

    }

[[selfnavigationController]popToRootViewControllerAnimated:YES];
   }
```

As we said some pages above, the method NSKeyedArchiver is used to store data in the iphone, and we have used this method to store the last user,ip and password selected in the table. This is represented in color yelow in the code above.

When we come back to the first view, the system will call to the viewdidload method of the firstview and this method have been edited by us as this way:

```
- (void)viewDidLoad {
    [super viewDidLoad];

    NSString *localPath1 = @"Documents/my1.archive";
    NSString *fullPath1 = [NSHomeDirectory()
stringByAppendingPathComponent:localPath1];

    NSString *localPath2 = @"Documents/my2.archive";
    NSString *fullPath2 = [NSHomeDirectory()
stringByAppendingPathComponent:localPath2];

    NSString *localPath3 = @"Documents/my3.archive";
    NSString *fullPath3 = [NSHomeDirectory()
stringByAppendingPathComponent:localPath3];
   UITextField *objetos1=[NSKeyedUnarchiver
unarchiveObjectWithFile:fullPath1];
    UITextField *objetos2=[NSKeyedUnarchiver
unarchiveObjectWithFile:fullPath2];
    UITextField *objetos3=[NSKeyedUnarchiver
unarchiveObjectWithFile:fullPath3];

Ip.text = objetos1.text;
Name.text=objetos2.text;
passwordField.secureTextEntry = YES;
passwordField.text=objetos3.text;
…
}
```

We use the method NSKeyedUnarchiver and the effect is that we are able to extract the data of the last server selected or used and then copied into the UITextFields (green code).

4.4) MANAGING THE CONNECTION

As we said some pages above, the user prior to start the connection must introduce some properties of the server. Furthermore , we have explained that we are going to use the open source code called InAppSettingsKit.

The aim of this point is how to manage the settings of some connection, how to create them and access to their values using InAppSettingsKit.

First of all, What is InAppSettingsKit?

InAppSettingsKit is an open source solution to easily add in-app settings to your iPhone apps. It uses a hybrid approach by maintaining the Settings.app pane. So the user has the choice where to change the settings.

How does it work?

To support traditional Settings.app panes, the app must include a Settings.bundle with at least a Root.plist to specify the connection of settings UI elements with NSUserDefaults keys. InAppSettingsKit basically just uses the same Settings.bundle to do its work. This means there's no additional work when you want to include a new settings parameter. It just has to be added to the Settings.bundle and it will appear both in-app and in Settings.app. All settings types like text fields, sliders, toggle elements, child views etc. are supported.

Before starting to use InAppSettings, we should decide which properties manage.

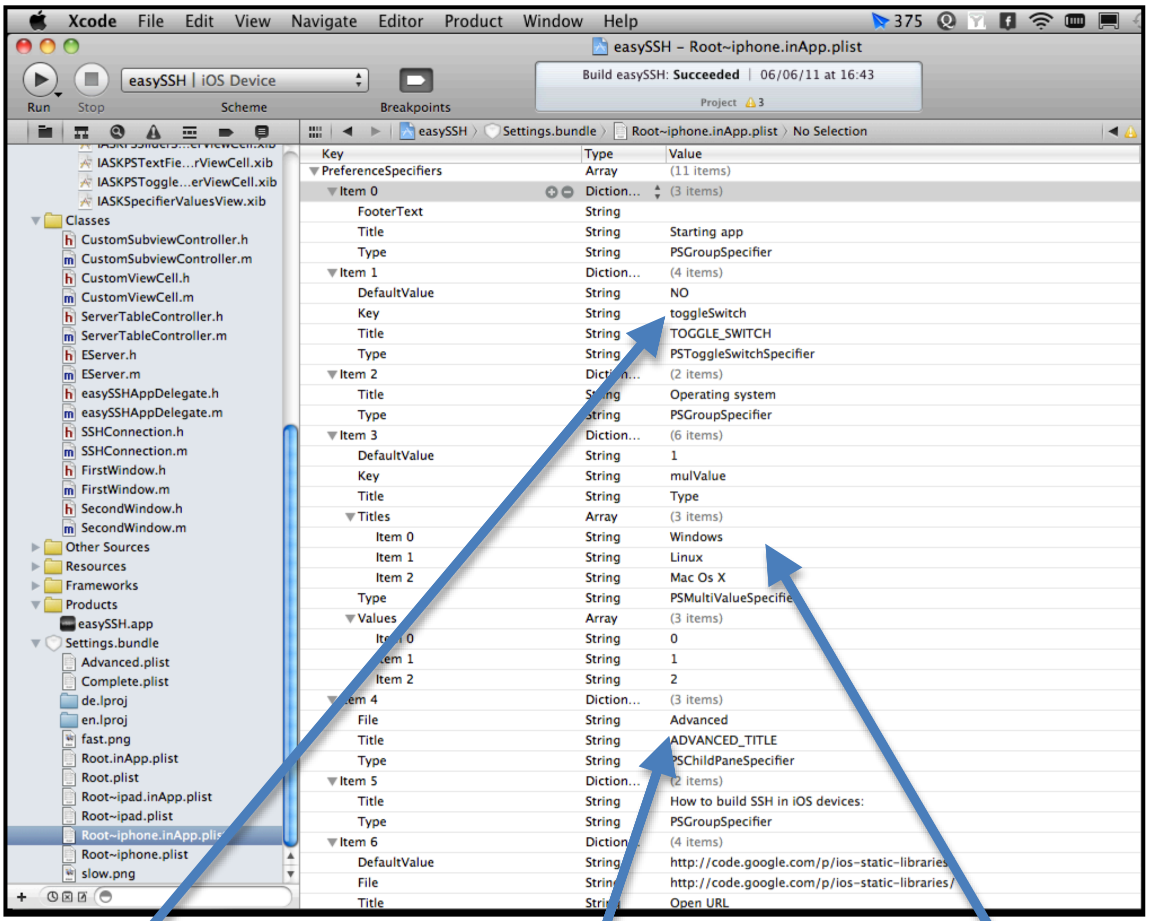
Properties decided:

- Kind of Operating System
- Parsing the answers
- Autoconnect at running

Where should we introduce them?

There are a bundle called Settings.bundle. So we must introduce them there, specifically we have introduced it at the file Root~iphone.inApp.plist.

In the next page we have showed a picture of this file and how we have edited it.



easySSH – Root~iphone.inApp.plist

Build easySSH: **Succeeded** | 06/06/11 at 16:43

Key	Type	Value
▼PreferenceSpecifiers	Array	(11 items)
▼Item 0	Diction…	(3 items)
FooterText	String	
Title	String	Starting app
Type	String	PSGroupSpecifier
▼Item 1	Diction…	(4 items)
DefaultValue	String	NO
Key	String	toggleSwitch
Title	String	TOGGLE_SWITCH
Type	String	PSToggleSwitchSpecifier
▼Item 2	Diction…	(2 items)
Title	String	Operating system
Type	String	PSGroupSpecifier
▼Item 3	Diction…	(6 items)
DefaultValue	String	1
Key	String	mulValue
Title	String	Type
▼Titles	Array	(3 items)
Item 0	String	Windows
Item 1	String	Linux
Item 2	String	Mac Os X
Type	String	PSMultiValueSpecifie
▼Values	Array	(3 items)
Item 0	String	0
Item 1	String	1
Item 2	String	2
▼Item 4	Diction…	(3 items)
File	String	Advanced
Title	String	ADVANCED_TITLE
Type	String	PSChildPaneSpecifier
▼Item 5	Diction…	(2 items)
Title	String	How to build SSH in iOS devices:
Type	String	PSGroupSpecifier
▼Item 6	Diction…	(4 items)
DefaultValue	String	http://code.google.com/p/ios-static-libraries
File	String	http://code.google.com/p/ios-static-libraries/
Title	String	Open URL

With the toggleSwitch object
we are able to manage the
Autoconnect propertie

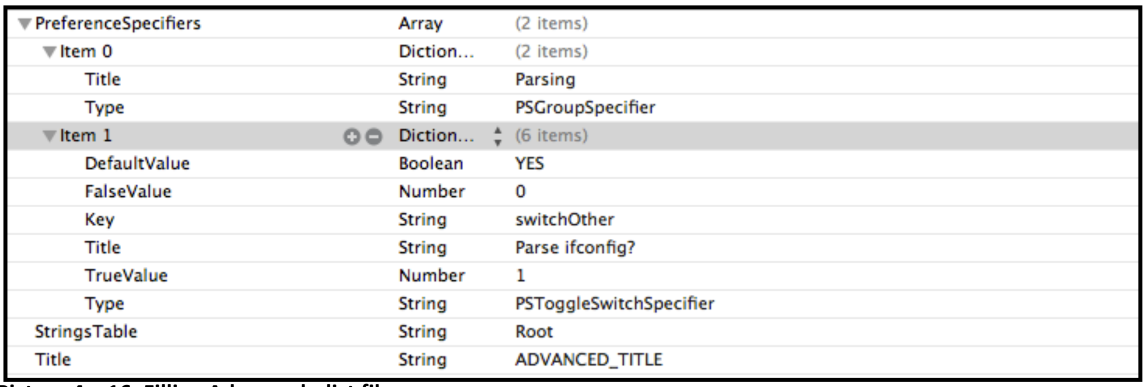
Her we manage the
kind of Operating System
Linux, MacOsX or Windows

With this Item we manage if the user want to
parse the answer of the server or not.

Picture 4 – 15. Filling Root-ipgone.inApp.plist file

How to access to this properties when we are connected and check the values?

For example if we want to know if the parsing actually is active we must to call to this functions:



▼ PreferenceSpecifiers	Array	(2 items)
▼ Item 0	Diction...	(2 items)
Title	String	Parsing
Type	String	PSGroupSpecifier
▼ Item 1	Diction...	(6 items)
DefaultValue	Boolean	YES
FalseValue	Number	0
Key	String	switchOther
Title	String	Parse ifconfig?
TrueValue	Number	1
Type	String	PSToggleSwitchSpecifier
StringsTable	String	Root
Title	String	ADVANCED_TITLE

Picture 4 – 16. Filling Advanced.plist file

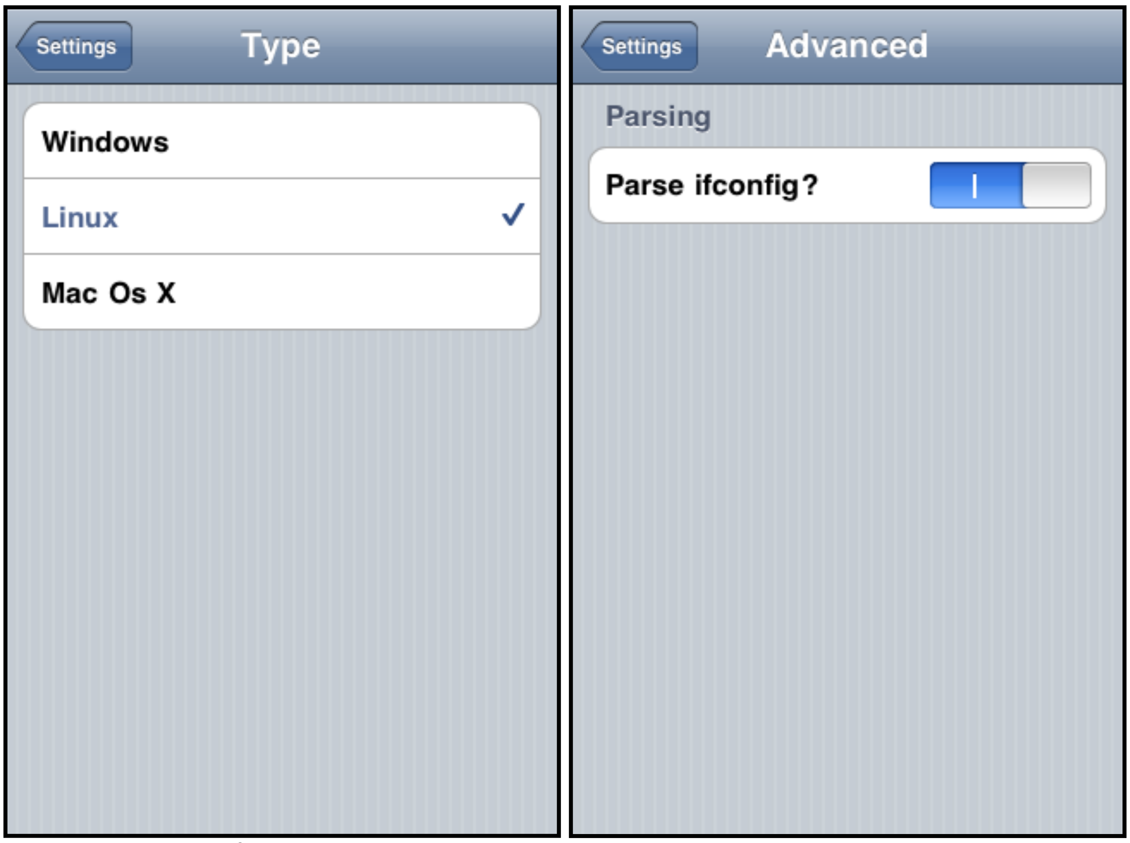
We only must to do a casting of the name of this item. In this case called "switchOther":

```
NSString *toogleValue = [[NSUserDefaults standardUserDefaults]
stringForKey:@"switchOther"];

if([toogleValue isEqualToString:@"1"])
{
…
      //THE PARSING IS ACTIVE!!
…
}
else
{
…
      //THE PARSING ISN'T ACTIVE!!
…

}
```

In the next page we can see some pictures showing the results.



Windows
Linux ✓
Mac Os X

Parsing
Parse ifconfig? ON

Picture 4 – 17. Display of some settings.

To finish this part, how can we call the view settings based on the source code InAppSettingsKit?

It's quite simple, we must only copy this function on FirstWindow.m

```
(IBAction)showSettingsModal:(id)sender {

    UINavigationController *aNavController =
[[UINavigationController alloc]
initWithRootViewController:self.appSettingsViewController];

    //[viewController setShowCreditsFooter:NO];   // Uncomment
to not display InAppSettingsKit credits for creators.
    // But we encourage you not to uncomment. Thank you!
    self.appSettingsViewController.showDoneButton = YES;

    [self presentModalViewController:aNavController
animated:YES];

    [aNavController release];
}
```

4.5) MANAGING THE ANSWERS

Once we can see the answer in the screen, printed in the UITextView, we have three options.:

- We can copy the answer pressing two times on the text, for next uses in the phone.

- We can send the answer via email to any computer

- The same but the sending is via SMS to the phone.

We are going to try to explain the second and third option which seem more complicated.

SENDING THE ANSWER BY EMAIL:

To make it working good, we must implement this both functions:

```
-(IBAction)sendMail {

    if ([MFMailComposeViewController canSendMail]) {
     MFMailComposeViewController *mfViewController =
[[MFMailComposeViewController alloc] init];
     mfViewController.mailComposeDelegate = self;
        [mfViewController setSubject:@"Command exit"];
        [mfViewController setMessageBody:response.text
isHTML:FALSE];

      [self presentModalViewController:mfViewController
animated:YES];
      [mfViewController release];
    }else {
     UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"Status:" message:@"Your phone is not currently
configured to send mail." delegate:nil cancelButtonTitle:@"ok"
otherButtonTitles:nil];

      [alert show];
      [alert release];
    }
}
```

```
-(void)mailComposeController:(MFMailComposeViewController*)
controller didFinishWithResult:(MFMailComposeResult)result
error:(NSError*)error {

    UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"Status:" message:@"" delegate:nil
cancelButtonTitle:@"ok" otherButtonTitles:nil];

    switch (result) {
     case MFMailComposeResultCancelled:
          alert.message = @"Message Canceled";
          break;
     case MFMailComposeResultSaved:
          alert.message = @"Message Saved";
          break;
     case MFMailComposeResultSent:
          alert.message = @"Message Sent";
          break;
     case MFMailComposeResultFailed:
          alert.message = @"Message Failed";
          break;
     default:
          alert.message = @"Message Not Sent";
        break;  }
    [self dismissModalViewControllerAnimated:YES];

    [alert show];
    [alert release];
}
```

The first one is used to send the email, we can take a look at the code with yelow background color , which copy the answer of the server into the body of the email.

The second one is used to control the different status of the email (canceled,saved…) and to come back to the previous view.

<u>SENDING THE ANSWER BY SMS:</u>

It's also the same like the last one, we must implement two functions and that is all.

This functions are defined in the next page:

```
-(IBAction) sendInAppSMS:(id) sender
{
    MFMessageComposeViewController *controller =
[[[MFMessageComposeViewController alloc] init] autorelease];
    if([MFMessageComposeViewController canSendText])
    {
     controller.body = response.text;
     controller.recipients = [NSArray arrayWithObjects:@"",
@"", nil];
     controller.messageComposeDelegate = self;
     [self presentModalViewController:controller animated:YES];
    }
}
```

```
-(void)
messageComposeViewController:(MFMessageComposeViewController
*)controller didFinishWithResult:(MessageComposeResult)result
{

    UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"Status:" message:@"" delegate:nil
cancelButtonTitle:@"ok" otherButtonTitles:nil];
    switch (result) {
     case MessageComposeResultCancelled:
          NSLog(@"Cancelled");
          break;
     case MessageComposeResultFailed:
          alert.message =@"Unknown Error";


          [alert show];
          [alert release];
          break;
     case MessageComposeResultSent:

          break;
     default:
          break;
    }

    [self dismissModalViewControllerAnimated:YES];
}
```

Mention also that the yellow background code in the first functions is used to copy the answer in the body of SMS and the second function is also used to control the status of the SMS.

6) Planning : GANNT DIAGRAM

This is the Gantt diagram after finish the project:

Phases/Month	January	February	March	April	May
Analysis	🟥	🟥			
Get information		🟨	🟨	🟨	🟨
GUI		▨	▨	▨	▨
Implementation			🟩	🟩	🟩
Testing					🟪
Documentation					

Phases/Month	June	July
Analysis		
Get information	🟨	
GUI		
Implementation	🟩	
Testing	🟪	
Documentation	🟧	🟧

As we can see, we started in January without any idea of how to do it, and we nedeed two months learning objective-c, checking how to work some simple app's downloaded from Internet and exploring all the possibilities that Objective-C allow us to do.

Furthermore, we had to prepare the environment with xCode, the frameworks, libraries, add the certificates for iphone, or even install photoshop to build the icon of the application have taken some time of this two months.

After that, collecting all the information have been the most time-consuming part of the project, which have notoriously taken more time than the others due to some problems arised along the way, such as problems finding ways to make the connection working, searching how to manage a database in iPhone, finding how to solve problems,...

Later on , GUI and the Implementation time have been the tasks, after getting information, that have required longer fragments of time. Nervertheless, these tasks have been more fun and entertaining than getting information.This is because we were able to check the results working on our device and doble-check if the information obtained was working correctly and matched the information gathered.

And later, the two last modules started more or less at the same time. The application has got some bugs and some errors like the aplication crashes if the user tries to connect without filling out the ip,user and pass…So we spent about 2 months fixing these errors.

The documentation has been the last one and we expect it will be helpful for future readers.

7) Conclusions

After 15 days testing the application on some iPhones and in different servers of the E305 department, the experience was so good, we solve all the bugs we get and also we experimented great feeling about the application. It works fast, showing quickly the answers of the servers, but it depends on the Internet connection.
In conclusion the application is working correctly, and usually in fast times.

Apart of this, we have to say is not so easy the way of a developer who wants to carry out applications for iPhone has to work. It is quite difficult to find a sustainable concept, then develop it and finish it completely without bugs, based on best practices, documented and solid foundations for further improvements and versioning.

We have learned to be patient when some code did not work in a week and it was keeping us away from moving forward on the project.

In addition, we have learned that we are the own developers of our application and we can choose what to do in each moment, but the best important thing is to organize the time and try to keep the schedule. Thus , organizing the first phases of the project well becomes such an important factor.

Summarizing, this project have been useful to know the world in which the iPhone is involved in. Learn the language of iPhone (Objective-C), deploy the code of the application, test it in the device,… All of this have been trully helpful to extend the knowledge about this topic.

8) Future work

We have raised several ideas to improve implementation in half term, which aim to bring out new features and enhancements adding to the application.

- Improve the application to make it faster: Delete the code which slow down the application

- Improve the UI adding images in background, adding more views to show the information and buttons more organized.

- Improve the settings menu deleting the S.O of each server and adding this property to the database. In this way we only must to specificy the S.O at the first time.

- Try commands to get files to iphone and save it in iPhone, for example like images and adding them in the iPhoto app installed by default.

- Adapt eSSH to the newest IOS 5.

9) Acknowledgment

First of all I want to thank my tutor of the project MIHNEA ALEXANDRU MOUCHA for offering me this project and supporting me from the beginning to the end when I had problems and I did not know how to solve them.

Thanks to the university and the entire department for letting me make this project and use the facilities

Finally, I also want to thank all my international friends and family who always helped me during the tough times. I could not have done it with all of you.

Thank you all very much!

10) Bibliography

http://apptrackr.org/

http://www.estudioiphone.com/

http://developer.apple.com/technologies/ios/networking.html

http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSFrameworks/iPhoneOSFrameworks.html

http://www.theiphonewiki.com/wiki/index.php?title=Main_Page

http://oreilly.com/iphone/excerpts/iphone-sdk/network-programming.html

http://stadium.weblogsinc.com/engadget/files/app-store-guidelines.pdf

http://www.iphoneexamples.com/

http://www.iphonesoftware.es/2008/06/24/programa-para-iphone-desde-dos-uitextfield-nivel-barragan/

http://www.libssh2.org/examples/ssh2_exec.html

http://sites.google.com/site/olipion/cross-compilation/libssh2

http://code.google.com/p/ios-static-libraries/

http://www.youtube.com/watch?v=rK4eK-z8-O0

http://www.iostipsandtricks.com/using-apples-mail-composer/

http://www.inappsettingskit.com/

http://www.aboutobjects.com/tutorials.html

http://www.x2on.de/2011/02/02/libssh2-for-ios-iphone-and-ipad-example-app-with-ssh-connection/