

# Genie Programming tutorial for non programmers

v 0.2, 26 May 2008

## How to start making your own programs for Genie

This guide will explain how easy is to make your own programs that can be run on your Genie device. From the simple 'Hello World' to more complex applications - Genie is the perfect development device for learning mobile phone programming. Using the Pawn programming language combined with the Genie SDK will allow even non programmers to start experimenting with their devices and learn coding in few easy steps. By learning to make programs for your Genie you can have very powerful tool in your workshop for experimenting and debugging of hardware and also creating all types of hobby electronics projects.

## What do you need

1. Genie Device with all supplied accessories - SD card, Charger, Mini USB cable
2. PC running Windows XP
3. Any type of SD card reader that you can plug to your PC - external USB reader or built in on your PC/laptop.

## First Steps

1. Always make sure that your Genie device is updated to the latest firmware. You can check this thread: <http://www.genie-universal.com/bbs/showthread.php?t=5> for the latest release info and download. You will also find instructions how to update your device
2. Download the latest version of the Pawn compiler from the Pawn site: <http://www.compuphase.com/pawn/pawn.htm> Download the Windows installer, run it and when finished you will have all needed tools installed in /Program Files/Pawn. You can check the pdf documentation for detailed information about Pawn usage
3. Download the latest version of the Genie SDK. You can get access to it by registering in the official Genie Universal Forum and requesting access to the Beta Section. Here is the link: <http://www.genie-universal.com/bbs/index.php> Unpack it to any directory you like

## Compile your first program

The Genie SDK contains a directory called 'examples'. Open the hello directory and double click on the hello.p file. It will open the Quincy.exe - this is the default Pawn IDE (Integrated Development Environment). All .p files are associated with Quincy, and those files are the actual source code that will produce your final program (executable).

The executable file of a Pawn program is with the .amx extension. It is same as Windows programs have .exe at the end. Genie will recognize this format as executable script and will be able to run it.

So i assume you have your hello.p example opened, you can see the code inside. We will go into the code later, for now you can try to compile it - this is the process of producing an executable from a source code. Simply press F7 on your PC, you should see the progress of the compilation in the bottom window. The perfect case is where you don't see any errors or warnings. In the directory where your hello.p file is, a new hello.amx file should be created. It was already supplied with the SDK, but don't worry, just delete it and press F7, you will see that is created by the Pawn compiler.

That's it - you have made your first Genie program. Congratulations!

## Running Pawn scripts in Genie

To do this you will need to connect your SD card reader to your PC. Then remove the SD card from Genie and plug into the SD card reader. Open MyComputer, find the newly found disk, open it and locate the Programs directory. You will have to copy your newly created hello.amx program into this directory. Please make sure that you put it in Programs or any subdirectory (like Nokia for example) but not in Files or System, or in the card root directory because Genie will not be able to execute it.

Then put the SD card back in Genie, wait the card to be detected (you can see the SD symbol in the right top corner) and using the Menu button, go to Programs. You should see now in the list of the directories you hello.amx script. Scroll down and select it, then press Run.

The screen should become dark and 'Hello World' should be printed. Congratulations you just run your first program on your Genie device.

## Modifying the source code

Now when you know how to compile and execute programs, you can try to modify the source code, add new things or simply just have fun with Pawn language. It is a good idea to do a bit of reading about programming, but if you are really impatient, then you can maybe learn by trial and mistake.

hello.p source code:

```
// -----  
// Hello world application  
// -----  
  
#include "../genieapi.h"  
  
main()  
{  
    printf "Hello world\n"  
    os_sleep 500  
}
```

The top 3 lines that start with '//' characters are called comments. Those are ignored by the Pawn compiler, you can put any text after them. It is usually used to show some additional information about the program, functions or any remarks.

Then #include ... is called directive and tell the compiler to use the provided genieapi.h file that contains all available API calls. More info about them later.

The next line contains main() - this is called a function, and it contains number of operations that Genie will execute. Every operation between '{' and '}' brackets will be compiled and then executed.

In our first program we have only 2 operators - printf and os\_sleep. The first one prints to Genie screen, the second one just delays the time before the script is unloaded. The printf will print the text after it - "Hello world\n". The text is called an argument for the function printf. You can obviously change it to whatever you like, recompile, copy again to the SD card and run.

The os\_sleep function takes as argument the time in milliseconds (1000 mS = 1 s). This is the time that our script will sleep (do nothing).

So give it a go - try to change the text, try to change the delay, or copy the printf on a new line and print a second text. Lets try something like this:

```
// -----  
// My modified code ;)  
// -----  
  
#include "../genieapi.h"  
  
main()  
{  
    printf "This is a test\n"  
    os_sleep 100  
  
    printf "Oh, second line\n"  
    os_sleep 100  
  
    printf "Tired already\n"  
    os_sleep 100  
}
```

You can create new text file, name it test.p, insert this code, save, Then double click on it, when Quincy opens, press F7 and put the created test.amx on your card and run it.

## Playing with the Hardware

Now when you are really better coder, lets try to make something really cool - blinking LED application.

### What do you need

1. Old service cable with 10 pin RJ-45 connector
2. Bunch of old LEDs - red, yellow and green at least
3. Soldering Iron, solder, maybe some flux



### Preparation

First make sure the connector is really 10 pin and that you have all 10 wires present and crimped into the RJ-45 connector

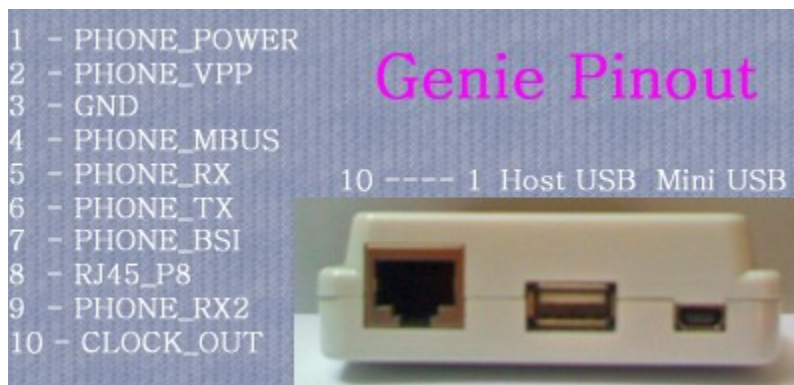


Ok, you have the correct cable, cut it while leaving enough length, then remove 10 cm from the rubber cover and using your soldering iron prepare all pins as shown in the next picture:



Let's make a LED blink once

Ok we can make small script to make a LED to blink. First solder the Green LED to your cable. Don't worry, its not impossible to locate it the correct pins, here the Genie pinouts:



So by checking the colors of the wires in your cable and looking at this diagram, you have to solder the negative terminal of the Green LED to the GND pin of Genie, and the positive terminal of the Green LED to the PHONE\_MBUS pin of Genie. On my cable GND is the violet wire and PHONE\_MBUS is the blue wire. On your cable it might be different colors, so be careful.

Here what the soldered LED look like:



About negative and positive terminal of a LED - those can be located by carefully checking the metal pads inside the LED, usually the negative is bigger, but on some LEDs (red ones) it is the opposite way. The best way is to locate them using a multimeter in semiconductor checking mode or simply use a 1.5 V battery, swap the pins till you get it to light on.

Now let's make the code

If all done, you can make a new file, rename it to blink.p, double click to open in it in Quincy and put this code:

```
// -----  
// Blink once  
// -----  
  
#include "..\genieapi.h"  
  
main()  
{  
    printf "Blink...\n"  
  
    set_pin(PHONE_MBUS,1)  
    os_sleep 1000  
  
    set_pin(PHONE_MBUS,0)  
    os_sleep 1000  
  
}
```

Compile by pressing F7, put the blink.amx in your SD card. Then put the card in Genie, make sure the newly soldered cable is plugged and run your blink.amx script.

The result should be something like this:



The `set_pin(PHONE_MBUS,1)` statement makes the voltage on the PHONE\_MBUS pin to go to 3.3V. This will cause the Green LED to light on, cause is connected to this pin.

Then we insert small delay with the already familiar function `os_sleep`. If we don't do this, the LED will blink for too short time and we won't be able to see it.

Then using `set_pin(PHONE_MBUS,0)` we make the output voltage on the same pin to go to 0 V. This will make the LED to switch off.

Now you can see the the function `set_pin` accepts 2 values - first one is the name of the pin we like to control, the second is the level of the voltage (1 and 0). Note that as we are going to use this further in the next example.

## Let's make it blink longer

Now we can make this even cooler, by simply editing the source code, we can make the LED to blink for whatever time we want. Here the code:

```
// -----  
// Blink 20 times  
// -----  
  
#include "..\genieapi.h"  
  
main()  
{  
    printf "Blinking...\n"  
  
    // Repeat the operation 20 times  
    for(new i = 0; i < 20; i++)  
    {  
        // Put high level on the pin  
        set_pin(PHONE_MBUS,1)  
        os_sleep 200  
  
        // Put low level on the pin  
        set_pin(PHONE_MBUS,0)  
        os_sleep 200  
    }  
}
```

The only new thing here is the `for(new i = 0; i < 20; i++)` statement. The for operator makes a operation to be repeated, in our case 20 times. The operations that will be repeated is between the '{' '}' brackets again.

Ok compile again, put in the card, run in Genie and watch your LED blink for 20 times. Feel free to experiment and maybe change the number of blinks, the delay between on and off states etc. The power is in your hands.

## Making more complex design

If you got tired of playing with just one LED, why not make more advance schematic, just solder 2 more LEDs. You will need to use your red and yellow LED. Connect them to the PHONE\_RX and PHONE\_TX pins (the positive terminals) and all negatives to GND.



So to make it even simple - all 3 LEDs have their negative terminal soldered together and to the GND of Genie. The positive terminal of the Green LED is connected to PHONE\_MBUS pin, the positive terminal of the Yellow LED is connected to PHONE\_RX pin and the positive terminal of the Red LED is connected to the PHONE\_TX pin of Genie. Leave all the unused wires unconnected.

You will need a new code to be able to control all 3 LEDs, why not start with this one:

```
// -----  
// Running light with 3 LEDs  
// - blue, yellow and red  
// -----  
  
#include "..\genieapi.h"  
  
main()  
{  
    printf "Blinking...\n"  
  
    // Repeat the operation 10 times  
    for(new i = 0; i < 10; i++)  
    {  
        // Put high level on the pin  
        set_pin(PHONE_MBUS,1)  
        os_sleep 50  
  
        // Put low level on the pin  
        set_pin(PHONE_MBUS,0)  
        os_sleep 50  
  
        // Put high level on the pin  
        set_pin(PHONE_RX,1)  
        os_sleep 50  
  
        // Put low level on the pin  
        set_pin(PHONE_RX,0)  
        os_sleep 50  
  
        // Put high level on the pin  
        set_pin(PHONE_TX,1)  
        os_sleep 50  
  
        // Put low level on the pin  
        set_pin(PHONE_TX,0)  
        os_sleep 50  
    }  
}
```

The code above will put on a LED, keep it on for a small amount of time, then put on the next one, wait, put it off, etc. Because all this is done very quickly the overall effect is that we have a running lights:



Enjoy