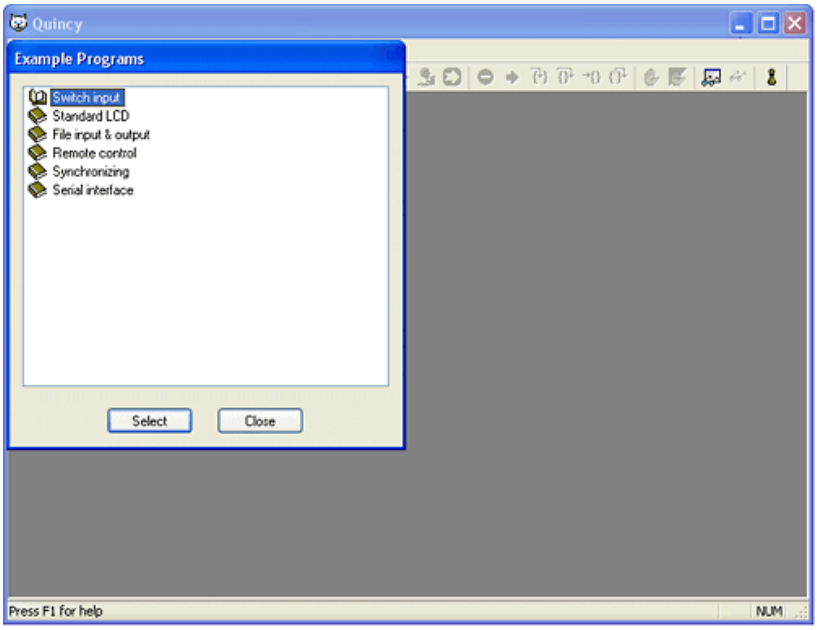
PROGRAMMING THE MP3 CONTROLLER/PLAYER: GETTING STARTED

This is a step-by-step guide for programming the H0420 MP3 controller/player. It uses a few very simple scripts to demonstrate the software tools that come with the H0420 device. On many of the subjects touched upon in this article, you can find more information in the various guides that are also included with the H0420 software package.

Install the software

The H0420 comes with a CD-ROM that holds, among others, a setup program for Microsoft Windows. The first step is to run this setup program and follow its directions.

After the installation completed, you will have a new group below "Program Files", called "H0420 MP3 Controller". From this group, choose the item "Pawn IDE (Quincy)". If all is well, the following screen pops up.



You can close the "Example Programs" dialog. In this tutorial, we will write our programs from scratch rather then basing them on an example.

The very first program

Assuming that you have Quincy (the PAWN IDE) in front of you, follow the list of steps below to create a script that plays a track in a loop.

1. Select the menu **File**, then item **New**. From the dialog that appears then, choose **Pawn Source File** and click **OK**.
2. Type the following text into the new file, exactly as below.

```
@reset()
    {
    play "track1.mp3", 255
  }
```

I will explain the contents of this program later on. First we will try to get it running on the H0420.

6. Save the file. You may choose yourself under what filename and path. It is advised, however, that you keep de default extension ".p".
7. Compile the program into an executable version. To compile, select the menu **Project**, then item **Compile**. This action will pop up a dialog for the build process, and it should look similar to the image below.

If the Build dialog lists any warnings or errors, there must be a difference between the code that you typed in Quincy and the example code listed above. Programming languages are quite strict, especially regarding white space and punctuation. Please also observe that character case is important and that the keywords "@reset()" and "play" are lower case.

If there are errors, close the Build dialog, correct the errors and compile again.

8. Now that the program is built, you can transfer it to a CompactFlash card, along with the file "track1.mp3". The PAWN compiler stores the compiled program in the same folder (directory) as the source script, but with the name "autorun.amx". The name autorun.amx is required by the H0420 device.

   Launch Windows Explorer, go to the path where you saved the test script and copy the autorun.amx that you will find there onto the CompactFlash drive —you need a "card reader/writer" for this.

   Example tracks are on the CD-ROM that comes with the H0420, in the folder "tracks". Copy the file track1.mp3 onto the CompactFlash card. You may also use an arbitrary MP3 track of your choice, but you must rename it to the filename "track1.mp3", because that is what the script refers to.

9. Remove the CompactFlash card from the card reader/write and insert it into the H0420 MP3 controller. Then connect an amplifier or amplified speakers (or a headphone) to the audio outputs and connect the H0420 to power (5V). After two or three seconds, the MP3 controller starts playing the track. When it reaches the end, the track re-starts playing back at the start.

How it works

Unlike retail MP3 players, the H0420 MP3 controller has no hard-wired functionality for playing MP3 files; it needs to be "told" what to play and when. You tell the H0420 how to behave via a programming language. In the above example, we have made a simple, a *very* simple, program.

Actually, saying that there is no hard-wired functionality in the H0420 is not quite accurate: in absense of a script, the H0420 will play all MP3 tracks that it finds in the root directory of the CompactFlash card in random order. That is all that it does. To make the H0420 do anything else, you must add a script.

Before the H0420 MP3 controller can run this program, the program must be compiled. This is because computers and devices like the H0420 only understand "machine language". You, however, would not enjoy writing the programs in machine language —trust me on this.

Therefore, a compiler translates from "human readable" source files, with instructions that sound like English words, to machine language.

PAWN is an event-driven language, which means that it runs a program (or part of a program) when "something specific" happens. That "something specific" is called an *event*. The H0420 understands events like:

- a switch press or release
- an incoming digital signal
- a time-out, or a timer "alarm"
- a change in audio status: start or stop playing, end of track reached
- and a few more...
- ... and "power on", of course.

When you switch on the power, or insert a CompactFlash card into the H0420, the MP3 controller resets and starts running. The reset starts with a self-test, which takes two or three seconds, but after that, the H0420 executes the "reset" event in the script. For reasons internal to the PAWN language, all event names start with an "@" symbol, so the reset event is actually called "@reset". Also, the event specification must say what its parameters are, between parentheses behind the event name.

The reset event has no parameters, but the parentheses must still be preset. This is how we get to the full event name specification @reset(), which is the top line of our script.

Following the event name specification must be a list of instructions between curly brackets.

The curly brackets are also called "braces"; instructions are also called "statements". You may choose where you position the curly brackets, but out of tidyness, it is advised to align them to the same column.

In our case, we have only one instruction: play. We give two parameters to this instruction. The first parameter is the filename of the track to play. This parameter is required. The second parameter is optional and it gives the number of times that the track must be repeated. The repeat number can be a value between 0 and 255. A repeat value of 1 means that the track plays and then repeats once, so that it plays two times. When you set the value 255, you might expect that the track will play 256 (255 + 1) times, but the value 255 is a special case: it sets endless repeating.

A more realistic script

The above script is indeed simple. Most users, though, want a track to be activated on a sensor or a switch. For the next switch, we will implement the following functionality:

- There are two tracks, called "track1.mp3" and track2.mp3".
- Each track plays when its respective switch is activated. There are, hence, two switches. The script uses switch entries 0 and 1 for the two switches.
- No track plays on start-up (or reset). You must press a switch before audio starts.

See below for the script for this functionality. In Quincy, you can create a new file and copy the script from this document into that file. Then, you can proceed as for the first program. When testing the script, you need to connect two switches, though. But before that, I will explain how the script works.

```
@button(pin, status)
  {
  if (status == 1)
    {
    switch (pin)
      {
      case 0:
        play "track1.mp3"
      case 1:
        play "track2.mp3"
      }
    }
  }
```

We need no activity on reset (or start-up), so we do not need to implement the @reset() at all. Instead, there is a new event function: @button() that, as you may guess, fires at each switch press and switch release.

If you are asking yourself why the "switch" event function is called @button() rather than @switch(), it is because switch is a keyword in the PAWN programming language.

Function @button() takes two parameters: pin is the sequence number of the switch, from 0 to 15; status is 1 (true) if the switch is pressed and 0 (false) if it is released. If you are new to programming, it may look strange at first that computers (and programming languages) start counting at zero instead of at 1, but you will get used to it quickly.

As before, all instructions for the @button() function are between curly brackets. In fact, there is only one instruction attached to @button(): an "if" statement. The other lines are sub-statements of this ifstatement. It is good practice to show the hierarchy of statements with the level of indenting (from the left margin).

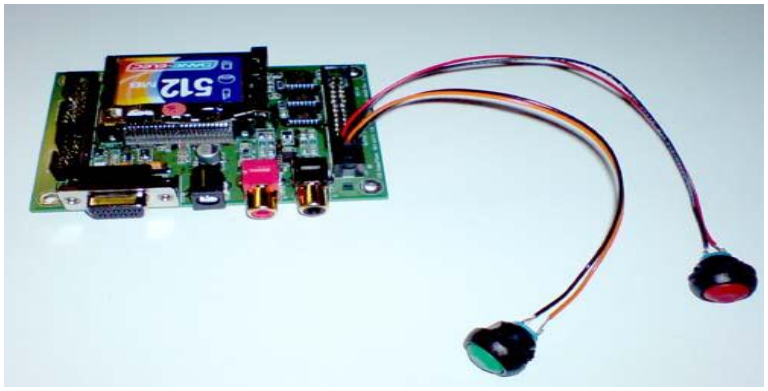
The if statement executes the sub-statements below it only if its argument evaluates to a *true* expression. The test expression compares parameter status to 1. So, only if status is equal to 1, the statements below the if statement will run. If status has any other value, the complete block of instructions below the if is skipped.

I could have used another if statement to test whether it was switch 0 or switch 1 that got pressed, but for variation, I used the switch statement here.

The switch statement is more easily extendible when you add more switches, because you may add an (almost) infinite number of "cases".

Testing the script

To test this second script, you need to connect a pair of switches to the connector block that the "User Guide" and the datasheet refer to as "J5". J5 is a 34-pin IDC connector, suitable for ribbon-cable and for crimp-sockets. When connecting just a few switches, a set of 2-way connector cables like in the picture at the right is often convenient. In the picture below, two of these connector cables are plugged into J5 at the positions for the first and second switches. Each cable pair has a simple push-button switch at the other end.



Once the switches are connected, and the CompactFlash card holds the script and the MP3 tracks, you can power up the H0420 MP3 player and push either switch to start one of the tracks.

Where to go from here...

The very first program developed in this article is so simple that you may take the syntax rules at face value and understand what the program does. The second program already requires that you understand the basics of programming and of the PAWN language.

The PAWN programming language is a very flexible and powerful language. With it, you can control almost every aspect of the H0420. If you are new to the PAWN language, please read the tutorial in the PAWN"Language Guide". If you are new to programming, you may want to read the paper: "A Gentle introduction to Programming" first.

The tutorial in the Language Guide does not contain any specific information related to the H0420 MP3 controller/player. However, the H0420 Reference Guide has its own tutorial introduction. More detailed information on specific topics is assembled in "application notes" for the H0420. Most of these notes are fairly advanced, though.

And finally, the Quincy IDE comes with various example programs. It is likely that one of these comes close to what you want to do with the H0420.

Source: http://www.compuphase.com/mp3/tutorial.htm