Instruction Manual For
XBMC Emulator Scripts
Python Development Tool by Alexpoet

(Forgive me–I'm a tech writer by trade.)

Introduction and Purpose

The XBMC Emulator Scripts (which I'll just call "the emulator" from here on) are a pair of Python scripts named "xbmc.py" and "xbmcgui.py" that should be included in the same rar that contained this document. These files are intended to allow you to run XBMC Python scripts on your computer, for debugging purposes.

NOTE:

> The emulator is designed as a development tool, not (by any means) as an end-user type platform. The emulator will only successfully run scripts designed to work with the emulator (as described later). It's not safe to assume that any XBMC Python script you download will automatically work with the emulator. However, if you read and understand the contents of this document, it should be possible to *make* these scripts work with the emulator.

Let me say again: the purpose of the emulator is to allow you to run *XBMC scripts* on the PC. The emulator will not in any way improve performance on your XBox, and should not be installed on the XBox.

Also, remember that the emulator is trying to recreate an environment using a system (Tkinter, specifically) which is similar in nature but often quite different in approach. So I'm taking my best shot at recreating (or at least simulating) the XBMC environment, but it won't (and can't) be a perfect replica. Thus this document.

Within this document, I'll explain what the emulator can and can't do–what it *is* supposed to do, and what it's not supposed to do. Simply stated, it's designed to let you test your XBMC Python scripts on your PC. To be a little more precise, it's designed to let you test your *scripting*.

That is, it's meant to allow you to see how well you've written the program. It won't always allow you to see how well the program works. I hope you can see the difference. There are a lot of things that XBMC does–things built directly into XBMC that Python scripts can call, such as playing videos or playlists–that would take a significantly more complicated command to emulate, and wouldn't really reveal much.

Anyway, I think I might be getting into way too much detail for an introduction. Mainly I want you to be able to script out your program, run it as you would any other Python script you were developing, and get useful error output (it's WAY too easy to lock up the XBox and never have access to the error output). And, also, to see *basically* what your GUI is going to look like.

That's the purpose of the emulator. Now I'll start on the use.

Installation

Installing the emulator is fairly simple. I'm assuming you have Python installed on your PC. If you don't, visit www.python.org for further information.

If you have Python installed and working, find the folder containing your Python installation. This is typically either C:\Python\ or C:\Python23\ or similarly numbered to indicate the version (2.3, in this example). Within that folder should be a subfolder called "Lib" (or "lib").

To install the emulator, simply copy the files "xbmc.py" and "xbmcgui.py" into the "Lib" subfolder of your main Python install folder. That's all it takes.

Most of you probably already know why and how that works. Some of you don't care. Either way, you can skip ahead to the next section "Additional Setup." But for those who do care and don't already know, let me briefly explain.

Most scripts written for XBMC contain, as the first line:

```
import xbmc, xbmcgui
```

or something to that effect. This tells the script that it is allowed to use functions and classes contained within the external libraries named "xbmc" and "xbmcgui." All of the Controls, many of the basic functions, and the GUI

mainloop itself are defined in these external libraries–and they're NOT Python scripts. If they were, we could just open them up, see what they do, and replicate that simply.

Instead, they're resources (probably written in C) built into the XBMC install and pretty much entirely unavailable to us. We can get a copy of a makedoc showing what the libraries *do*, but not the code that shows how they do it.

What this means is, if you write a script to work on the XBox, and tell it to look for those two libraries, and then try to run it on your PC, it'll tell you that those two libraries can't be found. And then it'll stop working, which makes debugging hard.

A lot of people have gotten around this by writing "stubbed out" scripts–Python scripts by the same names ("xbmc.py" and "xbmcgui.py") that just pass off all the function calls, or return text. These stubbed out scripts will keep the import command from failing, and let the script run on, but they won't try to recreate any of the function calls.

Now, I'm sure many people have adapted their stubbed out scripts beyond just passing off functions (or returning defaults where required). Probably somebody's started with a stubbed out script that, by now, does a better job emulating than my emulator does. But I couldn't *find* any of those...so my emulator wins.

To make any of these stubbed out scripts work, including the emulator, you just have to put them somewhere that the Python installation on your PC can find them. The easiest (and probably most appropriate) place to do this is in the Python folder's "Lib" subfolder, which is where Python automatically looks for library scripts exactly like these.

As soon as the emulator files are in place, when you run an XBMC Python script it should start without complaining about the missing imports. It probably WILL complain about something...we're just past the import stage.

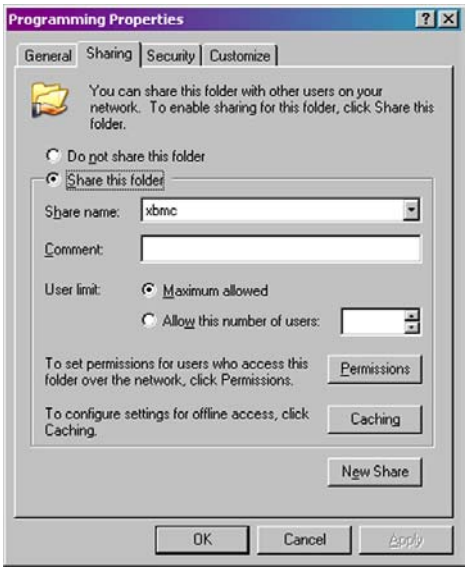
Additional Setup

There's one more step (and the credit and my thanks go entirely to Bitplane from the XBMC forums for this suggestion) that will make things run smoothly. Setting up a "Q:" drive.

That's not something I can easily emulate. I was thinking through all kinds of complicated code catching every os call and parsing the string and...just, crazy. Bitplane suggested something much easier, but you have to do it yourself.

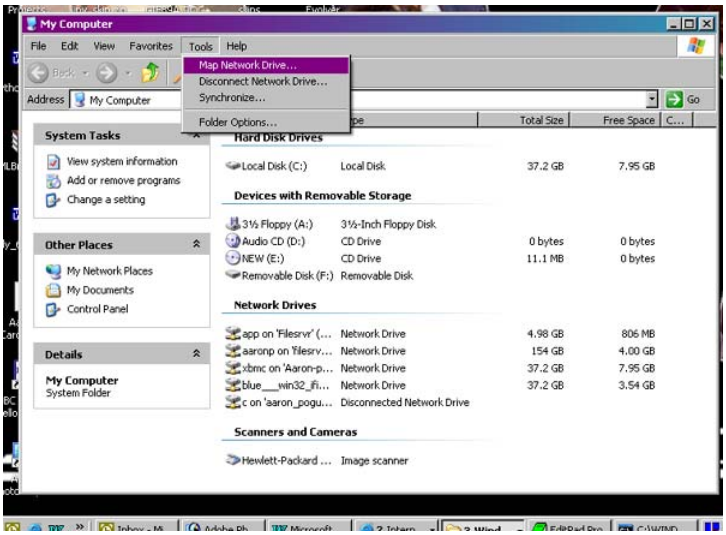
If you're using a Windows machine (sorry, it's the only OS I know--if you want to email me with details on how to do this on another system, I'll include your suggestions), simply set up a Network Share on your PC, and arrange the subfolders within it as necessary. I'll walk you through the process, but for some of you that sentence was enough information to get you going. If you're not interested in setting up a "Q" drive (or if you're unable), I'll have a note in the next section describing another way to handle it. You can skip ahead to the next section "Operation."

But setting up a "Q:" drive can be fairly simple. First, you have to create a folder on your computer (anywhere on your computer) that will simulate the "xbmc" folder on your XBox. For me, it was my "Programming" folder, where I already kept all of my programming projects. The folder name and location don't really matter. But find a folder, and share it out. I recommend using the share name "xbmc," but even this probably isn't necessary.



Create a network share of the folder containing your XBMC scripts.

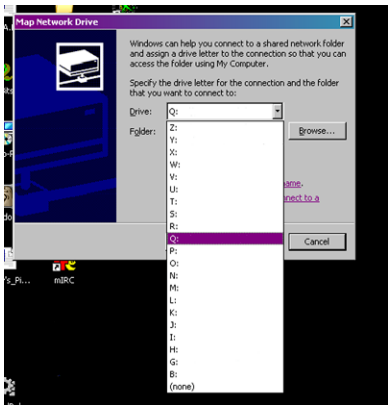
Once it is shared, create a "scripts" subfolder (just like the XBMC installation on your XBox has a "Q:\scripts" folder). And you're ready to turn the share into "Q." Now open up "My Computer, and click on the Tools menu, then click on "Map Network Drive..."



From My Computer, select "Map Network Drive" in the Tools menu.

This will popup another window, asking you which drive letter you want to assign. Obviously, the one you need is "Q." Click on the dropdown arrow next to the drive letter (it'll probably be "Z" by default), and then look through the list.

There is, of course, a chance that you'll already have "Q" mapped to another shared folder. If it's not too inconvenient, you might consider replacing that one with a different drive letter. If you just can't use "Q," though, then this process isn't for you. Skip ahead to the next section, "Operation."



Select "Q" from the available drive letters, then Browse to your share.

If your "Q" slot is free, though, simply click on it, and the dropdown box goes away. Now you have to pick which folder to associate with it. Click on the Browse button on the right to open up the folder selection dialog box.

In this list, search out your computer's network name, and under the shared folders it should include an "xbmc." Click on this folder, and then on "OK." Next, "OK" out of the Map Network Drive window, and you should now have a "Q:" drive properly setup.

Operation

Obviously, there's not much to do in terms of operating this sort of an emulator. Most of the operation will be of scripts *using* the emulator. There are a few necessary rules for setting up your scripts to work with it, though, and a couple of comments I want to include about onAction and controller emulation in general.

The Global Variable 'Emulating' and the Explicit Init

First of all, I need to state that this program cannot be entirely an emulator, right out of the box. I tried and tried, but I had to add a few lines to all of my XBox scripts that will run when on the PC, but that won't run when on the XBox.

The easiest way to make that work, I've found, is to add this line right after your import statements (or, in other words, at the top of your script):

```
Emulating = "Emulating" in dir(xbmcgui)
```

Make it full left, at the top of the script, so that "Emulating" is a global variable available throughout the script.

This works because there's a function called "Emulating" in the xbmcgui.py file, that (of course) doesn't exist in the C library. So if you're running this on the XBox, dir(xbmcgui) *won't* contain an "Emulating," and the global variable will be False. If you're running it on the PC, the variable will be True.

Once you have that variable to check, you can add the necessary lines (and I'm doing everything I can to keep them as few as possible) to your scripts, to make them run with the emulator.

The most important extra line (probably), is this:

```
if Emulating: xbmcgui.Window.__init__(self)
```

This line will appear within your main class's __init__ function. If you get the Runtime Error "Too early to create image," it's because you're missing this line in your init.

I won't try to explain (at least for now) why that's necessary. Not because it's too complicated for you to understand (most of you probably get it right away--it has to do with overriding the init function in my subclass), but *I* don't entirely get it, and if I tried to explain, I'd probably be way wrong.

Anyway, just trust me that that line's necessary. The following example show's a sample of what your new init would look like:

```
import os, re, string, urllib, xbmc, xbmcgui

Emulating = "Emulating" in dir(xbmcgui)

ACTION_MOVE_UP = 3
ACTION_MOVE_DOWN = 4
ACTION_SELECT = 7
ACTION_MENU = 10

class MovieGuide(xbmcgui.Window):
    def __init__(self):
            if Emulating: xbmcgui.Window.__init__(self)
            self.ZipCode = XMGConfig.Primary_Zip_Code
            self.AltZipCodes = XMGConfig.Alt_Zip_Codes
            self.SearchRadius = XMGConfig.Search_Radius
            ...
```

and all the other kinds of things you do at the beginning of a script. I tried to include enough of an actual script there (that's stripped out of my XMovieGuide.py, as it currently stands) that you could see how it works. The main thing to remember, though, is to add this line:

```
if Emulating: xbmcgui.Window.__init__(self)
```

at the very beginning of the __init__ function for whichever class (or classes? I'll have to think about that) derives from the xbmcgui.Window class. That is, any time you have

```
class ***(xbmcgui.Window)
```

(where *** is the name of your class), you should include the "if Emulating" line at the top of the class's init function.

Note that you *should* be able to add that line (as well as the 'Emulating = "Emulating" in dir(xbmcgui)' line) to most third-party scripts that you download and make them work with the emulator, but I can't make any guarantees there.

Working with the "Q:" Drive

As you get to this point in the manual, there are two possibilities (well, okay, *thousands* of possibilities, but I'm only going to handle two). One is that you were able to follow my instructions above and set up a network share on your

PC creating an actual simulation of the "Q:" drive situation you get with the XBox. If so, you're in pretty good shape. All you have to do is make sure that, as you set up scripts on your PC, put them all in the "scripts" subfolder of your network share. Also, keep all the file locations and folder structures on your PC parallel to the ones on your XBox. If you take care of this, you should be in good shape.

The other possibility is that, for one reason or another, you weren't able to make a "Q:" drive. If that's the case, there's not much you can do about running other people's scripts. If you download a webradio script or some such, written without the help of the emulator, then there's no reason to assume it would have been written to accommodate people who don't have a "Q:" drive.

But you can still build your own scripts to work with the emulator. It adds a couple ugly lines, but this is what I did before Bitplane showed up with the clever suggestion.

Early in your script, when you're still setting up globals, create a RootDir variable containing the path. I usually do that *anyway*, so that it looks like:

```
RootDir = "Q:\\scripts\\XMovieGuide\\"
```

and then later in the script, I just call:

```
fFile = file(RootDir + filename, "wb")
```

or whatever.

So, if you *don't* have the benefit of the "Q:" drive, be sure to use the global variable, and set it up like this:

```
if Emulating: RootDir = os.getcwd() + "\\"
else: RootDir = "Q:\\scripts\\XMovieGuide\\"
```

Not quite as pleasant, but it gives you something to work with. Of course, this is checking the "Emulating" global variable that we established earlier (see "The Global Variable 'Emulating' and the Explicit Init" if you don't remember this). You've got to have that set up, obviously, for this check to do any good.

That's all it takes. Those should be the big obstacles to getting scripts running with the emulator. Oh, wait! Except for:

Emulating the Controller/Remote

Here's the biggie. Right now, I've barely got any onAction controls built into the emulator.

"But why?" You ask. "onAction defines most of what we *do* with the XBox!"

And I can only shake my head sadly. It's...err...tough. I'm doing some kinda tricky stuff, and while emulating the controller isn't climbing Everest, it's not just simply falling into place, either.

But I'm working on it. For now, you can click on buttons, and it'll send the proper onControl signal. You can also double-click list items (from ControlLists) and properly trigger *that* kind of onControl. For all I know, that might cover everything onControl does. So that's still helpful. But the onAction commands are tougher.

That's not to say it won't work. I mean, it might even work by *tonight*. Just depends how many tricks I try, and which one is the right one. But in the meantime, just know that the controller isn't yet emulated. What I'll probably end up doing is either building a GUI frame below the main output window with buttons representing controller buttons (that idea just struck me, and should be easy), or else have predefined keyboard keystrokes to represent all the possible onAction events.

Anyway, whatever happens, it'll take a while to develop. But once I've got that working, I'll update this section of the manual with complete instructions. In the meantime, if you have a really great suggestion, please feel free to make it. Maybe I'll be able to make some progress that way.

Concluding Remarks

I conclude!

Actually...if you want to reach me at all, I recommend visiting the XBMC forums and either posting in my Emulator Scripts thread (it's stickied at the top of page 1), or send me a private message through the board.

Once I get my webpage up, you'll be able to leave emails/comments there, too.

I'm really trying to provide some useful material to the community. So if you can see something that needs fixing, let me know. If you appreciate something I've done...mention it. It's the good reviews that keep people like me working. And the bad ones, I must admit, that keep us getting better. So...whatever you've got to say, say it. I'll do my best to make something of it.

Alexpoet.