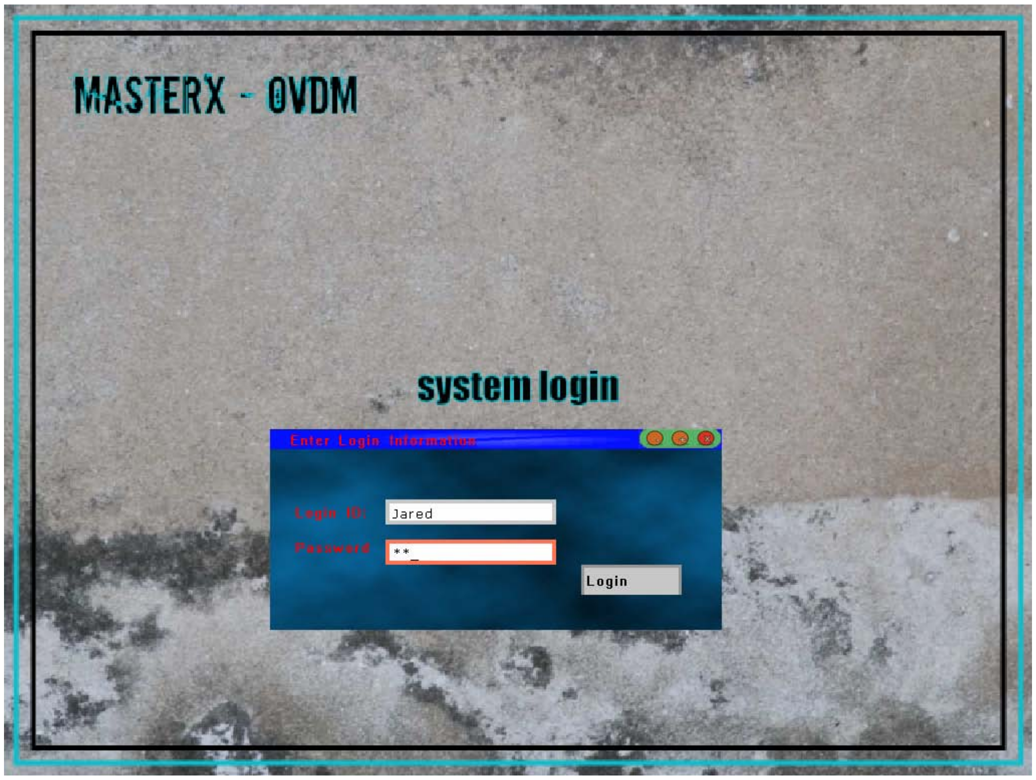
MasterX Online Virtual Dimension Manager (MAsterX-ovDM) (a OS Simulator)

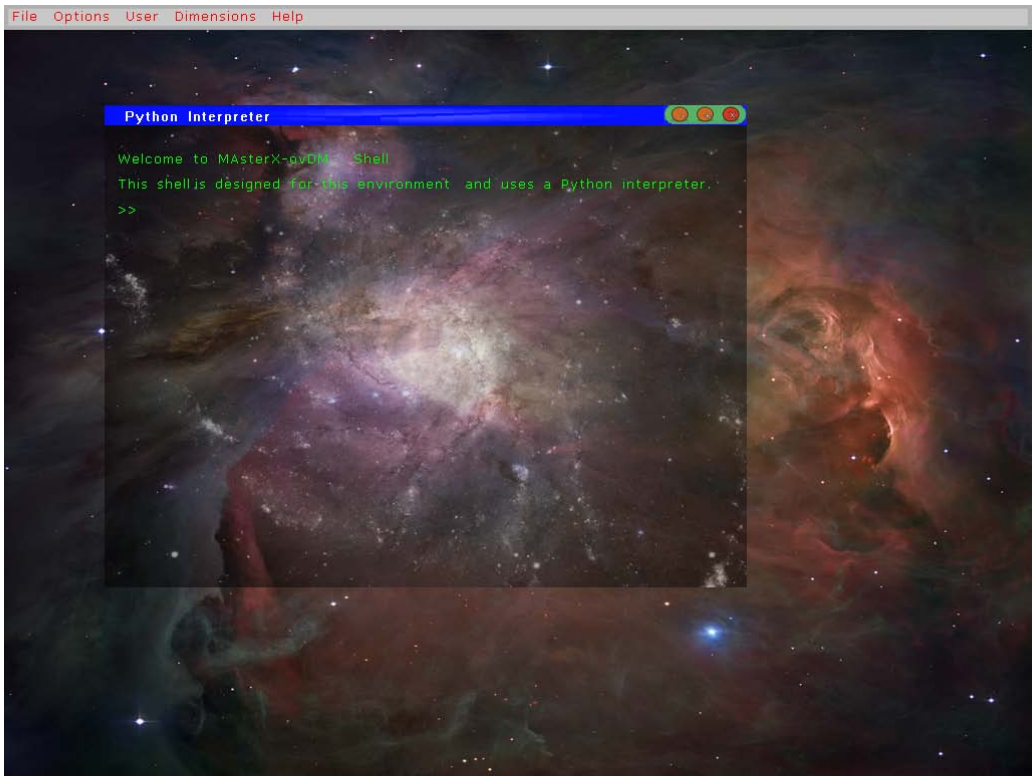
web page: http://lostsidedead.com/project.php

Download it from: Mac OS X 10.5.5 (Intel): http://lostsidedead.biz/osx/masterx/masterx.dmg
Ubuntu 8.10: http://lostsidedead.biz/deb/MasterX-ovDM-Ubuntu-8.10-r2.deb What is MasterX ? MasterX is a Simulator for a interactive environment for Python. It contains a embedded Python Interpreter and Multi-Dimensional user interface for creating Python Applications. What is in Version 1.0 ? This is a Demo Release, so not all the features are complete, but the basic functionality is there. You can create dimensions, windows, assign skins, draw and process input with Python. Also you can Chat with other users online. When you first open the system, log in with



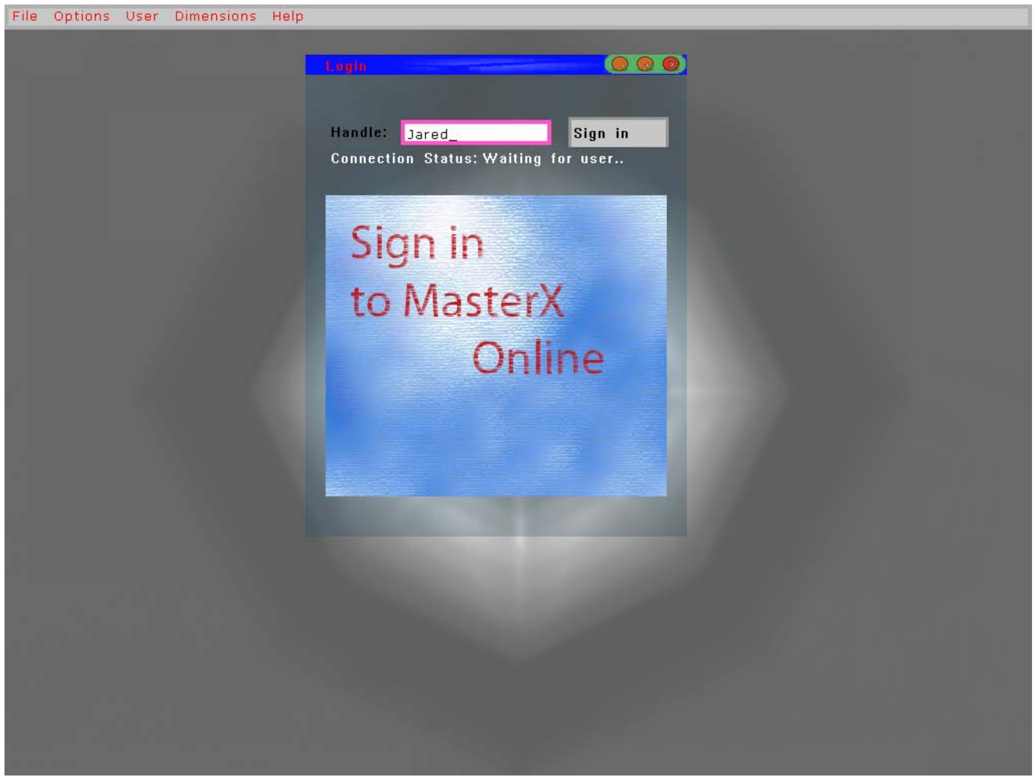
user: Jared
pass: masterx
it is case sensitive

then go to the menu, User->change user/pass and create your own user name and password.



```
Welcome to MAsterX-ovDM  Shell
This shell is designed for this environment and uses a Python interpreter.
>>
```

The above screen shot is the default System console, where there is a interactive Python session were you can change just about anything in the System.
Place your Python scripts in the scripts directory and they will appear in the home dimension's Python menu where they can be executed.
The demo comes with a example Python script for a game of 3 card monte. The application still needs to have all the window widgets and controls
be accessible from Python which is still a work in progress.



Handle: Jared_

Connection Status: Waiting for user..

Some more informatin:
The Original Idea:

"The idea is that for every 'space' in the operating systems graphical user interface, there will be multiple windows, and other spaces. Thinking of a desktop as a virtual space, the application would create a new 'dimension' for your applications on on the fly. Switching between dimensions is simple using a combination of key presses or the scroll on your mouse. Each dimension would have its own icons, wallpaper, windows, and other features. Very similar to KDE and GNOME's window pagers. The difference between the two is that my application will not be a window manager. It will function on its own, and be able to compile for a multitude of platforms. Like a window that has many windows inside of it. This is because I do not have the ability yet to code a window manager for X. However I am studying on the Xlib api. Dreams also include a built in scripting language shell. A lot like bash but geared more towards triggering change within the windowing environment.. A example of this would be Window A in side of Dimension B would create Dimension C with Window D. The shell could expand or become a single desktop. Dependent on the applications currently running and the users preference. This idea I orignaly wrote in the year 2000, and I believe I finally have learned enough about computer programming to properly implement it. The concept created in this project will just be a prototype for more time down the road when I can actually implement it for say X, or as a real Operating System itself.."

```
Supported Platforms:

Mac OS X Leopard ( Intel )
Linux
FreeBSD


Program  Controls

Controls:

Shift + Escape                    (Quick Quit)
LEFT ALT  + CTRL + UP             (Restore all Window)
LEFTALT + CTRL+ DOWN              (minimize all Windows)
LEFT SHIFT + LEFT                 (shift left one dimension)
LEFTSHIFT + RIGHT                 (shift right  one dimension)
Left Alt  + Left Shift            (shift focus order of widgets)
F1                                ( jump to Console Dimension )
F2                                (take a screen shot)
CTRL+SHHIFT+UP                    (Restore all Windows )
CTRL+SHIFT+DOWN                   (Minimize all windows)


For Python Programming.

import masterx

in a  interactive session or python script  to be run in the application ( Dimensions -> Console ) to gain access to
versions of libmxui classes
exported in python. ( Still in progress ).

program arguments:

When executing the application from the command line you can pass the following arguments:

--width=W
--width=H

W is width in pixels
H is height in pixels

example

./masterx --width=1280 --height=1024

or  if you wish to automatically export the PYTHONPATH

./RunMasterX --width=1280 --height=1024

when no arguments are given, defaults to 1024x768

--log=true

will send stdout log information to a file called system.log
versus printing to the standard output stream.


Python Feature Log:
in the shell:

masterx moudle classes

Explantation of masterx module api
```

import masterx	**this imports the module that contains the functions and classes used for manipulating the masterX interface**

win = masterx.mxWin("Title", x_cord, y_cord, width, height)	**This is how you create a window using masterx module to later be added to a dimension.**
win.useskin("img/skins/mx/black")	**This is how you assign a skin to a window created by class mxWin skins are located in the img/skin directory**
win.settitle(" title ")	**This is how you set the title of the window.**
win.setpos(x,y)	**this his how you set the position of the window within the dimension.**
win.setsize(width,height)	**This is how you set the windows width and height, in pixels.**
win.setcallback("win")	**This is how you set the callback name for the interpreter to call when a event occours. render callbacks, keypress callbacks, mousemove, mouseclick, and mousedown events.**
`def render_callback():` `        # whatever you want here` `        win.fillrect(0,0,640,480,255,255,255)` `win.render = render_callback` `#assign input processing callbacks` `def keydown(key):` `      print "key pressed: " + str(key) + " !"` `win.keydown = keydown`	**This is how you assign callbacks to the win object, created with mxWin (could be named anything you just must create a instance of the object with a call to mxWin) Setting the render callback, will cause it to be called ever frame update, keydown callback will be called when a key is pressed. Some other callbacks that you can set are:** **win.keyup** **win.mousemove** **win.mousedown** **win.mouseup**
win.fillrect(x,y,width,height, red,green,blue)	**You may have noticed the fillrect call in the render_callback previously this is one of the drawing commands that the window can use to update its state and draw shapes images, and other objects. This call takes x,y position width and height and rgb values as red green and blue.**
win.printtext(x,y,r,g,,b,"hello")	**This will print text on the window's surface, x is the x-coordinate, y is the y-coordinate, r is the red component , g is the green component, and b i sthe blue component of the color, followed by the string to print**
`setting a pixel:` `first you must lock the surface:` `win.lock()` `then set  the pixels:` `win.setpixel(x,y,red,green,blue)` `then when finished unlock the surface:` `win.unlock()`	**Setting a pixel, or pixels is a 3 step process. First you must lock the window's surface with a call to win.lock(), once it is locked you can set as many pixels as you want. Once your done you must unlock the surface with win.unlock()**
win.bail()	**This will exit your python script successfully releasing all context's and resources.**

surf = masterx.mxSurf()	this will create a offscreen surface to hold pixel data like a image or offscreen drawing surface.

surf.loadpng("surf.png")
surf.load("surf.bmp")
	loadpng will load a .png graphic while load will load a Bitmap (BMP) file.

surf.release()	This will allow you to prematurely release the surface, if you do not call this the surface is automatically released on your programs destruction.

copying to a window:
 	
win.copysurf(surf, 0, 0, 1024, 768)
 	
resize copy
 	
win.copyresizesurf(surf, 0, 0, 1280, 1024)
	These functions allow you to copy your surfaces to a active window, normally called during a render callback function.

dim = masterx.mxDim()	Creating a Dimension for your Windows to Live in this creates a dimension called dim
dim.add()	This add's your dimension to the dimension manager so it is viewable and useable.
dim.addwindow(win)	This allows your to add your windows to a created dimension.
dim.setbgcolor(r,g,b)	Sets the background color for the dimension
dim.setbackground("image.png")	Set the background image for the dimension, must be a png graphic.
dim.width()	returns the dimensions width in pixels
dim.height()	returns the dimensions height in pixels
dim.settitle("title")	sets the dimensions title displayed on the top of dimension manager
dim.setfocus()	Sets the current dimension focus to the current dimension pointed to by the caller

Working with the Console	
sys.stdout.setfont("arial")	**Sets the font for the System Console**
sys.stdout.setcolor(r,g,b)	**Sets the color for the System console**

PYTHONPATH

PYTHONPATH must point to the scripts directory
I have included a handy shell script called RunMasterX
which you can use to start the program and it will set the variables for you

so instead of running

./masterx

run

./RunMasterX

The program comes with a example Python Script of a game of 3 Card Monte:
Here is the example code:

```
#example

import masterx
import random

class card(object):
    def __init__(self, x,y, select,  speed):
        self.setpos(x,y)
        self.value = select
        self.speed = speed
    def setpos(self, x,y):
        self.x = x
        self.y = y
    def setvalue(self, value):
        self.value = value
    def draw(self):
        if self.value == 1:
            card_window.copysurf(card_window.card, self.x, self.y, card_window.card.w(), card_window.card.h())
        elif self.value == 2:
            card_window.copysurf(card_window.card_select, self.x, self.y, card_window.card_select.w(),
card_window.card_select.h())
    def drawBack(self):
        card_window.copysurf(card_window.card, self.x, self.y, card_window.card.w(), card_window.card.h())

def drawSuccess():
    for c in card_window.card_list:
        c.draw()

    card_window.printtext(50,50, 0x0, 0x0, 0x0, "Success ! Press Escape to quit");

def  drawFailure():
    for c in card_window.card_list:
        c.draw();
    card_window.printtext(50,50, 0xff, 0,    0, "Failure you lose, press  Escape to quit")

def draw_game():
    for c in  card_window.card_list:
        c.draw()
    card_window.printtext(250, 250+card_window.card_select.h(), 0xff, 0, 0, "Press Enter to Shuffle");

def draw_in_place():
    for c in card_window.card_list:
        c.drawBack()

def  draw_shuffle():
    for  c in card_window.card_list:
            c.x -=c.speed
            c.setpos(c.x, c.y)
            if(c.x < 75 ):
                c.setpos(700, 200)
                #c.setpos(600+card_wiidow.card_select.w(),200)
            c.drawBack()
    card_window.printtext(250,  250+card_window.card_select.h(),  0xff, 0xff, 0xff, "Press Space to stop shuffle")
```

```
def render():
    #card_window.fillrect(0, 0, masterx_dim.width(), masterx_dim.height(), 0, 0, 0)
    #card_window.copysurf(background.obj,   0, 0, background.obj.w(),  background.obj.h())
    card_window.copysurf(card_window.bg, 0,   0, card_window.bg.w(), card_window.bg.h())

    if card_window.render_mode == 1:
       card_window.printtext(50, 50, 0xff,0x0,0x0,"Press Space to Play")
       card_window.printtext(50, 75,0x0,0x0,0xff, "Press Escape to  Quit")
       card_window.copysurf(card_window.card_select,250,250,card_window.card.w(), card_window.card.h())
    elif card_window.render_mode == 2:
       draw_game()
    elif card_window.render_mode == 3:
      draw_shuffle()
    elif card_window.render_mode == 4:
        card_window.printtext(50,50,0xff,0xff,0xff, "Choose the card you think contains the pattern, ethier  1 2 or 3")
        draw_in_place()
        card_window.printtext(200+card_window.card.w()/2-25,200+card_window.card.h()+10,  0xff, 0x0, 0x0, "1")
        card_window.printtext(400+card_window.card.w()/2-25,200+card_window.card.h()+10,  0x0, 0xff , 0x0, "2")
        card_window.printtext(600+card_window.card.w()/2-25,200+card_window.card.h()+10,  0x0, 0x0,  0xff, "3");
    elif card_window.render_mode  == 5:
            drawSuccess()
    elif card_window.render_mode == 6:
            drawFailure()


def keydown(key):

    if key == 27:
        masterx_dim.exit()

    if card_window.render_mode == 1:
        if key < 256 and chr(key) == ' ':
            card_window.render_mode = 2
        elif key == 27:
            masterx_dim.exit()
    elif card_window.render_mode == 2:
        if(key == 13):
            card_window.render_mode = 3;
    elif card_window.render_mode == 3:
        if key < 256 and chr(key) ==  ' ':
            card_window.render_mode = 4

            if(card_window.card_list[0].x < card_window.card_list[1].x and card_window.card_list[0].x <
card_window.card_list[2].x):
                card_window.card_list[0].x = 200;
            elif(card_window.card_list[0].x > card_window.card_list[1].x and card_window.card_list[0].x >
card_window.card_list[2].x):
                card_window.card_list[0].x = 600
            else:
                card_window.card_list[0].x = 400


            if(card_window.card_list[1].x < card_window.card_list[0].x and card_window.card_list[1].x <
card_window.card_list[2].x):
                card_window.card_list[1].x = 200;
                card_window.winner = 1
            elif(card_window.card_list[1].x > card_window.card_list[0].x and card_window.card_list[1].x
>card_window.card_list[2].x):
                card_window.card_list[1].x = 600
                card_window.winner = 3
            else:
                card_window.card_list[1].x = 400
                card_window.winner = 2

            if(card_window.card_list[2].x < card_window.card_list[1].x and card_window.card_list[2].x <
card_window.card_list[0].x):
                card_window.card_list[2].x = 200;
            elif(card_window.card_list[2].x > card_window.card_list[1].x and card_window.card_list[2].x >
card_window.card_list[0].x):
                card_window.card_list[2].x = 600
            else:
                card_window.card_list[2].x = 400

    elif card_window.render_mode == 4:

        if(key < 256 and  chr(key) ==  '1' and  card_window.winner == 1 and chr(key)  != ' '):
            card_window.render_mode = 5
        elif (key <256  and chr(key) == '2' and card_window.winner == 2 and chr(key) !=  ' '):
            card_window.render_mode = 5
        elif (key <256 and chr(key) == '3'  and  card_window.winner == 3 and   chr(key) != ' '):
            card_window.render_mode = 5;
        else:
            card_window.render_mode = 6;

masterx_dim=masterx.mxDim()
masterx_dim.settitle("3card - Python Demo")
card_window=masterx.mxWin("", 0, 0, masterx_dim.width(), masterx_dim.height())
card_window.render_mode = 1
card_window.render = render
card_window.keydown = keydown
card_window.setcallback("card_window")
card_window.card=masterx.mxSurf()
card_window.card.loadpng("img/pydata/card.png")
card_window.card_select = masterx.mxSurf()
card_window.card_select.loadpng("img/pydata/cardselect.png")

background = masterx.mxSurf()
background.loadpng("img/pydata/cardbg.png")
card_window.bg = background
```

```
masterx_dim.addwindow(card_window)
masterx_dim.add()
card_window.card_list = [ ]
card_window.card_list.append( card(200, 200, 1,10))#random.randint(1,3)))
card_window.card_list.append( card(400, 200, 2,15)),#random.randint(1,3)))
card_window.card_list.append( card(600, 200, 1,12))#radom.randint(1,3)))
masterx_dim.setfocus()
```