[CPython Media]

Software Design Specification

Draft 2 (final)

November 14, 2005

[Knights Who Say Ni!]

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
Final Version 1	Andrew Nagel, Dustin Roberts, John Burton, Sam Gawthrop	This is the first version of the design document.	9/22/05
Final Version 2	Andrew Nagel, Dustin Roberts, John Burton, Sam Gawthrop	This is the second and final version of the design document.	11/14/05

Contents

1 INTRODUCTION....1

1.1 SYSTEM OVERVIEW....1
1.2 DESIGN MAP....1
1.3 SUPPORTING MATERIALS....1
1.4 DEFINITIONS AND ACRONYMS....1

2 DESIGN CONSIDERATIONS....3

2.1 ASSUMPTIONS....3
2.2 CONSTRAINTS....3
2.3 SYSTEM ENVIRONMENT....3
2.4 DESIGN METHODOLOGY....4
2.5 RISKS AND VOLATILE AREAS....4

3 ARCHITECTURE....5

3.1 OVERVIEW....5
3.2 RATIONALE....5

4 HIGH LEVEL DESIGN....6

4.1 CONCEPTUAL VIEW....6

5 LOW LEVEL DESIGN....8

5.1 MODULE....8

6 USER INTERFACE DESIGN....12

6.1 APPLICATION CONTROL....12
6.2 SCREEN....12

1 Introduction

1.1 System Overview

JES is a development environment that is used to give students of CS1315 a nice interface in which to program using Python. Many of the projects that students are assigned involve manipulation of images and audio files. This functionality is not supported natively in Python, instead Java libraries are used to perform the actual manipulation of the image and audio files. This solution works well but it relatively slow. The objective of this project is to replace the media components of JES with libraries that use CPython, rather than Java. Once these libraries are created, students can use any IDE they wish and will no longer be required to use JES. This should dramatically increase the speed with which manipulation of the image and sound files can be performed

1.2 Design Map

This document contains the detail of how the Media.py system will be designed. There are 5 major sections of the document: design considerations, architecture, high level design, low level design, and interface design.

Design Considerations will explain the different constraints that will be placed on the design and the assumptions we have made in order to design the system. It also details the design methodology we are using, and how we will work around risks, if they arise.

The architecture explains the different software and hardware systems that must be in place in order for our system to operate correctly.

The high level design is fairly short because the system we are designing is small. It will include a component diagram of the system.

The low level design will detail the classes in UML class diagrams of the system we are designing.

The interface design will show an example of how the interface will look on a user's computer.

1.3 Supporting Materials

PyGame: *http://www.pygame.org/docs/*

wxPython: *http://www.wxpython.org/*

1.4 Definitions and Acronyms

BMP – Bitmap

CPython – Python implementation written in C

GIF – Graphics Interchange Format

GUI – Graphical User Interface

IDE – Integrated Developer's Envrionment

Java – Object oriented programming language developed by Sun Microsystem

JES – Jython Environment for Students

Jython – Java implementation of Python

Khz – Kilohertz

MPEG – Sound format for computers

PCX – A bitmapped graphics file format

PNG – Portable Network Graphics

PyGame – Media library written in C for the CPython programming language

SDL – Simple DirectMedia Layer

TGA – Targa Bitmap

UML – Unified Modeling Language

WAV – Sound format for computers

WxPython – Windows manager that is cross-platform

2 Design Considerations

2.1 Assumptions

It will be assumed that users have downloaded and installed Python version 2.3 from www.python.org. Also, user's will provide their own IDE, as JES will no longer be available for development, since we will no longer use Jython.

The image part of the media system we are building requires the presence of the pygames Python library installed on both the developer's and user's computer, which gives the ability to load and modify image data. The image file formats supported by the pygames library include BMP, GIF, PNG, JPG, PCX, and TGA at the very least and possibly more depending on what other formats are supported by the user's system. Finally for saving image data only the JPG format is supported by the library at this time.

The sound part of the system will also require PyGame to be installed on both the user and the developer's system. It supports MPEG and WAV playback, and possibly more that are supported by the user's system. It will also save in these formats. For optimal sound performance, all files should be 22khz, signed, 16bit stereo wav files. Most other wav formats can and will be converted to this format when they are loaded, at a possible loss of quality. The system will only save files to the wav format.

The GUI will use wxPython, which is a cross platform windows manager. The GUI is only required to support the display of images. The assumption is that wxPython will be available for the user and developer on whichever platform they choose to use. wxPython is available on Mac and Windows.

2.2 Constraints

The system must work with CPython, which is a version of python that is written in the C programming language. There are no specific performance requirements, because the point of this project is to see if there are significant performance increases from using CPython over Jython. Also, all functions will be written within Media.py, which will allow transparency from a current JES user.

The image system and the audio systems must use a C library system as the underlying implementation that allows the system to make actual modifications to the image files. The reason for this requirement is the speed boost, which is the reason that the implementation is being moved away from the JAVA implementation in the first place.

2.3 System Environment

The hardware required is the minimum required hardware for Windows XP or Mac OS X. Although that is a minimum requirement, the faster the processor and the more memory a user has, the better the computer will perform. The system is being designed to work on x86 and PowerPC platforms, but CPython is platform independent, therefore it should work on any system that is running Python.

Software required is CPython version 2.3 and PyGame version 1.7.1. Any desirable IDE can be used to program in the python programming language.

The software requires a windowing environment, and wxPython is the library used to implement the windowing operations. (This library will call local system calls, and interacts with the operating system and windowing environment that way)

The system should have a sound device and the appropriate drivers that PyGame can hook into. Most computers will meet this requirement with no changes necessary.

2.4 Design Methodology

Python is an object oriented programming language. Therefore we will use object-oriented ideas to do the design process. One of the requirements, though, is that all parts of the system be contained within Media.py, therefore there will not be separate classes. UML will still be used to do the design part of the system, but class diagrams are not as important as interaction diagrams.

The most important factor of this particular implementation for images is speed; as such all design considerations must take the speed of the operation into account. In the image implementation there are several different methods that will work to do the modifications to the image's data. The important point in developing this implementation is the speed of these several different implementation choices. The choices for implementation will be based on the documentation provided with PyGame and what options result in the best performance.

2.5 Risks and Volatile Areas

The largest risk we really face is that PyGame will not have the needed functionality to implement all methods. If this is the case, we will extend the PyGame library using C to add the functionality that we desire. This may require us to scale back our plans for early beta testing and debugging of the program.

A volatile area is that the SDL library does not support saving sound files. If this is the case, we will be required to write the files to disk using Python. Fortunately, this is not a large risk to our group because John Burton has experience in writing WAV files to disk, including manipulation of headers.

The only volatile area in the image implementation is the lack of any functions pre-existent in PyGame for the creation of shapes in the image. These functions are provided by the Java graphics package and since they are no longer going to be provided by the underlying system, because there is the potential for speed pitfalls in this area. There are functions for the manipulation of multiple pixel values simultaneously so while moving this functional capability to the built-in functionality is a risk that is not considered a major one. Nonetheless, it could require us to scale back the beta release date.

3 Architecture

3.1 Overview

The image system resident in PyGame utilizes several layers through which image data can progress. First there is the basic image class, which is used to load and save image data from a memory resident form and into a savable image format. The save and load functionality utilizes Surface objects which are essentially memory representations of an image. It is possible for us to directly manipulate this image data, but such a task is not recommended, as it is relatively slow. Finally there is a "surfarray" class that is an array-based representation of the image. There are two different implementations of this image array. One is a simple 2D array that stores each pixel value as a single number. This representation would be useful when modifying a black and white image. The second array implementation is a 3D array that stores pixel data as a collection of color value for each of the color channels.

For image modification the data stored in an unmapped "surfarray" will be modified. When the image is to be saved back to a file or requested for view it should be written back to a Surface object. This write back will only be done when the complete image structure is requested for some reason. PyGame does support the direct modification of the Surface objects as a linked back-end for a "surfarray", but this method is considerably slower and we feel that only writing back to the Surface object when necessary is the best option.

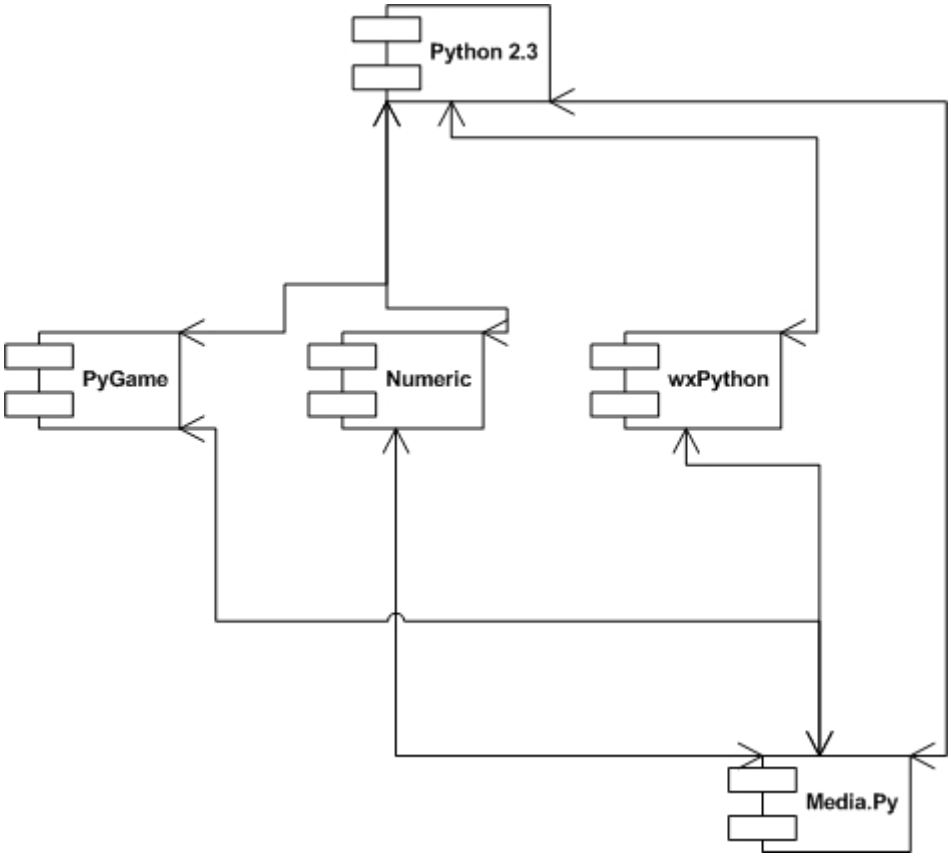
The Sound object will be broken down into sample objects, which will have an index. The index will point to its location within the sound object.

3.2 Rationale

Speed is the primary rationale for the design consideration decisions we have made in the image and sound implementations. The speed considerations are based on the documentation provided with PyGame. We will experiment with different implementations that will have the same end result, but this fastest method will be tried first as it is suggested by current documentation of PyGame.

4 High Level Design

4.1 Logical View



Developers should notice that Media.Py will use the Python language for all programming, but it will have to implement third party libraries (written in C) in order to provide all needed functionality. PyGame will provide all the image and audio manipulation functionality. Numeric will provide advanced math functions that aren't built into Python (such as square root). wxPython is included because it is a cross-platform windowing system. Although PyGame will allow users to display images, the customer has expressed interest in wxPython for future development with GUIs.

4.2 Process View

N/A

4.3 Physical View

N/A

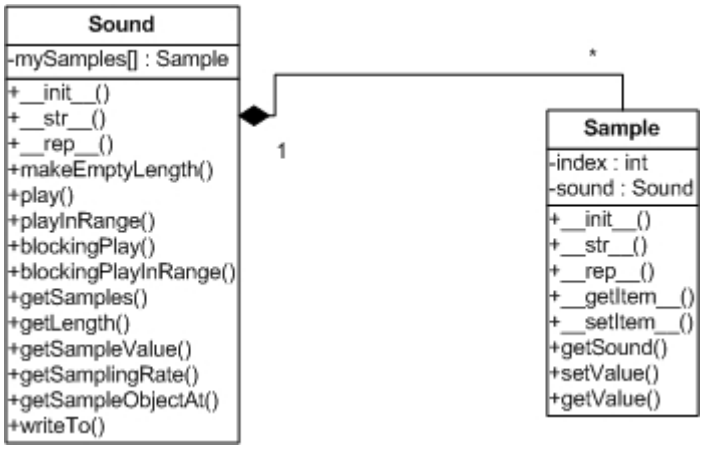
4.4 Module View

N/A

5 Low Level Design

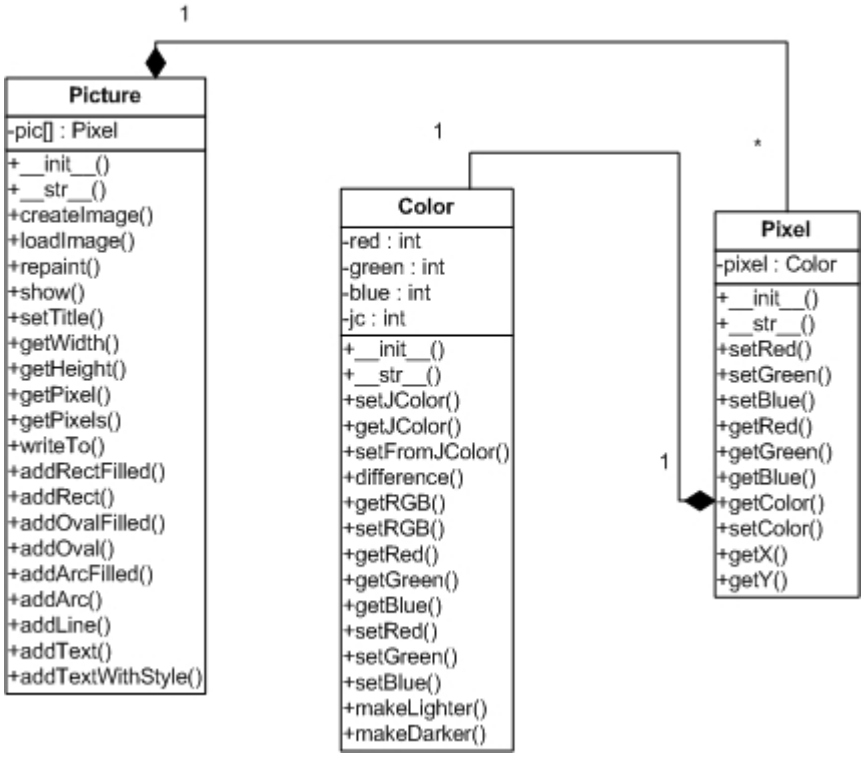
5.1 Modules

Sound:



Developers should notice that when any sound is opened, it will be converted into a collection of samples. Each sample has its own set of functionality, and knows its index number (where it is located in the sound). Upon opening, all sounds will be converted into 16 bit, 22000 Hz, signed, mono sounds.

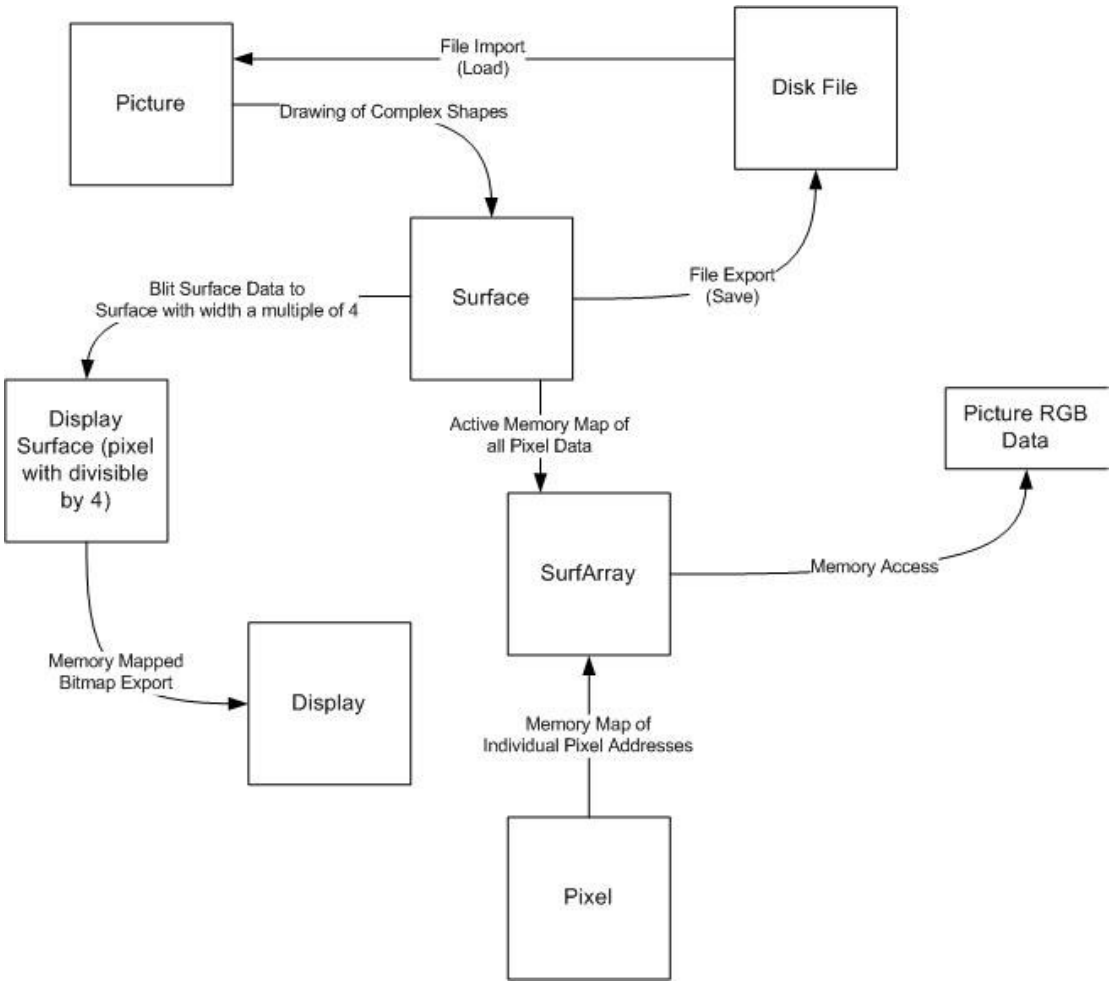
Images:



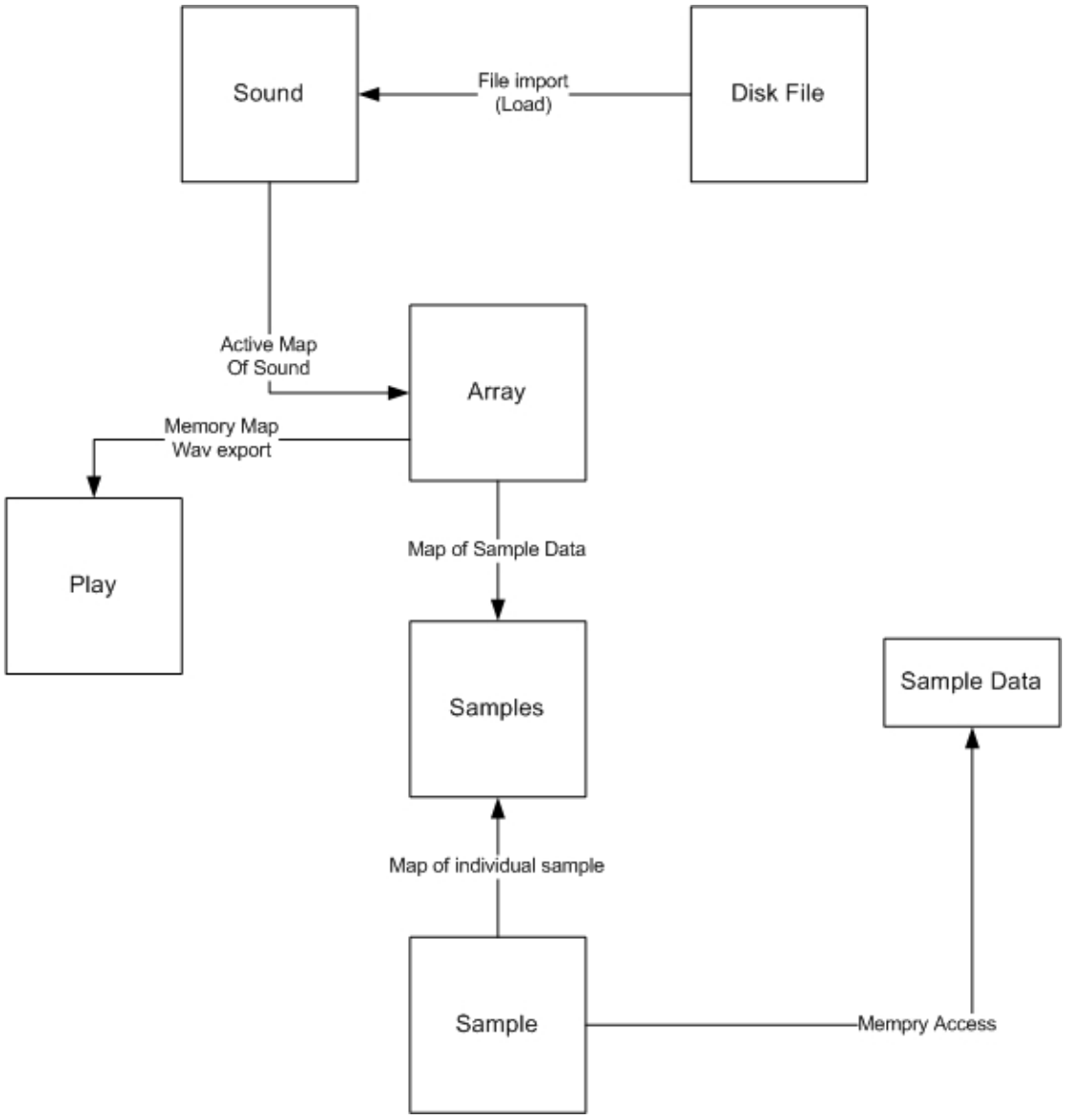
Developers should notice that when an image is opened it is stored as a collection of pixels. Each pixel knows where it is located and has a set of functions associated with it for manipulation. Every pixel is a color; that is why pixels have the get and set methods for the colors. Additionally, a user can create a color object that is independent of the image. These colors can be applied to the pixel or multiple pixels.

Data Flow:

Images:


Picture
File Import
(Load)
Disk File
Drawing of Complex Shapes
Surface
File Export
(Save)
Blit Surface Data to
Surface with width a multiple of 4
Display
Surface (pixel
with divisible
by 4)
Active Memory Map of
all Pixel Data
Picture RGB
Data
SurfArray
Memory Access
Memory Mapped
Bitmap Export
Display
Memory Map of
Individual Pixel Addresses
Pixel

Sound:


Sound
File import
(Load)
Disk File
Active Map
Of Sound
Array
Memory Map
Wav export
Play
Map of Sample Data
Samples
Sample Data
Map of individual sample
Sample
Mempry Access

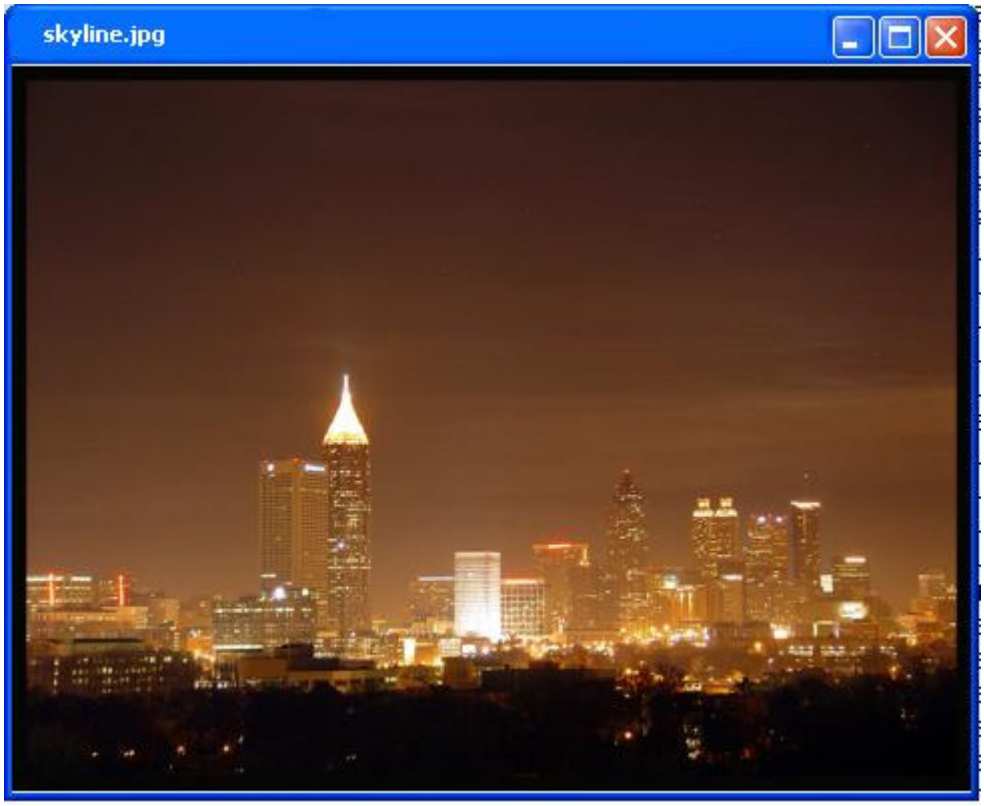
6 User Interface Design

6.1 Application Control

The window will be a display of an image. It will use the default operating system wrapper, and be interacting with the wrapper through the operating system.

6.2 Screen

Approximation of image display in Windows:



This is the GUI displaying a JPEG called skyline.jpg.

Developers do not need to manipulate the GUI in anyway. The GUI should open just as any windows would open using wxPython. The colors and layout of all GUIs should be the default layout and color that wxPython provides.