MMSV2 Python Tutorial (version 0.4, 5 jul 2006)

Requirements

mmsv2-1.1.0-patch-158+

mmsv2 python-plugin

python 2.3+

Python calculates the module search path based upon the environment variable $PYTHONPATH. It's important that this variable isn't empty. $PYTHONPATH should contain one or more directories. Run command "echo $PYTHONPATH" to be certain. I will return to this subject when discussing installation.

Installation

Copy the python plugin to the directory plugins/feature/ in the mmsv2-1.1.0 source tree. Extracting the content should create a directory called python/. Enter this directory and open the file Makefile.common. Look for two variables:

PYTHON_VERSION and PYTHON_INCLUDE_PATH

These need to be changed according to your installation of python. The default values are:

PYTHON_VERSION = python2.4
PYTHON_INCLUDE_PATH=/usr/include/$(PYTHON_VERSION)

PYTHON_VERSION should be python2.3 or python2.4.
PYTHON_INCLUDE_PATH is the path to the python includes files.

After editing the Makefile.common run:

make

This will compile the entire source.

After compiling the source enter the directory called lib/. This directory should contain three files (libpython.so, mmsv2.so, mmsv2gui.so).

Copy the file libpython.so to the mmsv2 plugins directory (usually /usr/share/mms/plugins/ or /usr/local/share/mms/plugins/).

Copy the files mmsv2.so and mmsv2gui.so to one of the directories in the $PYTHONPATH environment variable.

Enter the directory called cfg/. Copy the file PythonConfig to the mmsv2 home directory (usually /etc/mms/).

Enter the directory called cfg/input/[input]/. Copy the file python to the mmsv2 [input]/ directory (usually /etc/mms/input/[input]/, [input] could be keyboard or lirc).

Enter the directory called theme/. Copy the directory python/ to the mmsv2 theme directory (usually /usr/share/mms/themes/[theme]/ or /usr/local/share/mms/themes/[theme]/).

One final copy left to do:

Copy the entire directory scripts/ (with content) to the mmsv2 home directory (usually /etc/mms/).

After the copy /etc/mms/ should contain a directory called scripts (with one subfolder, windowexample).

That's it, the python plugin is now installed.

Documentation

This document.
Python class documentation can be found in mmsv2.pdf and mmsv2gui.pdf.
Also look at the example script (windowexample.py).

Using

Place all python scripts you create in the scripts folder (or sub folder), which should be located in the mmsv2 home directory (problably /etc/mms/scripts).

Creating scripts

The goal of this document is not to teach Python. I recommend that you read the documentation available on www.python.org

There are two specific libraries available for script developers:

mmsv2 – This module contain the audio and movie player part for python mms.
mmsv2gui – This module contain the user interface part for python mms.

Window

The first thing to do is to import libraries

```
import mmsv2, mmsv2gui
```

A window can be displayed by calling the “do_modal” method:

```
win = mmsv2gui.Window()
win.do_modal()
del win
```

The code above will display the window until the user fire the “back” event. The del is used to delete the class instance.

If you put this code in a script (called display.py) you'll see an empty window. The user have to fire the “back” event to exit this script.

The code above is quite boring. The only way to get some action is to close the window. To make it more interesting we have to create our own window class. That can be achieved by inheritance.

```
import mmsv2, mmsv2gui

class MyWindow(mmsv2gui.Window):
     def __init__(self):
          pass

     def on_action(self, action):
          if action == 'back':
               self.close()
          elif action == 'action':
               print 'User pressed the action key'

win = MyWindow()
win.do_modal()
del win
```

The code above will exit if user fire the ”back” event, but will also print the text ”User pressed the action key” (to console) when user fire the “action” event. When we create a subclass to “mmsv2gui.Window”, we also have to implement the method “on_action”. If we don't, user can't close (“close” method above) the window. It doesn't matter on which action we decide to close the window.

LabelControl

Now it's time to display some text on the window we created. For that we will use the LabelControl class. LabelControl has some properties including position, color, font and size. To view the content of a control you first need to add it to a window.

```
import mmsv2, mmsv2gui

class MyWindow(mmsv2gui.Window):
     def __init__(self):
          self.label = mmsv2gui.LabelControl(10, 20, 200, 30, 'Some text', \
                                             'Vera', '0xffffff', 0, 'left')
          self.add_control(self.label)


     def on_action(self, action):
          if action == 'back':
               self.close()

win = MyWindow()
win.do_modal()
del win
```

The code above will display the text “Some text”

10 is the x position.
20 is the y position.
200 is the width.
30 is the height.
'Vera' is the text font.
'0xffffff' is the color of the text (0xRRGGBB, RR=Red, GG=Green, BB=Blue).
0 is the text offset counting from the x position.
'left' is the alignment of the text.

The “add_control” method adds the control to this window, without this call the control will not be visible.

If you add a control to a window, you can also remove it. This is done below when the user fire the “action” event.

```
import mmsv2, mmsv2gui

class MyWindow(mmsv2gui.Window):
     def __init__(self):
          self.label = mmsv2gui.LabelControl(10, 20, 200, 30, 'Some text', \
                                             'Vera', '0xffffff', 0, 'left')
          self.add_control(self.label)


     def on_action(self, action):
          if action == 'back':
               self.close()
          if action == 'action':
               self.remove_control(self.label)

win = MyWindow()
win.do_modal()
del win
```

ImageControl

The ImageControl is used to draw pictures on the screen.

```
import mmsv2, mmsv2gui

class MyWindow(mmsv2gui.Window):
     def __init__(self):
          self.bg = mmsv2gui.ImageControl(0, 0, self.get_width(), \
                                     self.get_height(), \
                                     'general/standard_bg.jpg')

          self.add_control(self.bg)


     def on_action(self, action):
          if action == 'back':
               self.close()

win = MyWindow()
win.do_modal()
del win
```

0 is the x position.
0 is the y position.
self.get_width() is the width.
self.get_height() is the height.
'general/standard_bg.jpg' is the path to the picture.

For width and height we use the window methods “get_width” and “get_height”. This is a easy way to create a custom background for the window, because “get_width” and “get_height” will return the screen resolution defined in mmsv2 config file.

ButtonControl

To make our scripts more interactive, we use the ButtonControl. If the script contain more then one ButtonControl, the user can navigate between them by pressing keys “next”, “prev”, “left” and “right”.

```
import mmsv2, mmsv2gui

class MyWindow(mmsv2gui.Window):
     def __init__(self):
          self.button1 = mmsv2gui.ButtonControl(0, 0, 100, 30, 'button1', \
                                 'Vera', '0xffffff', '0x000000', \
                                 'button-focus.png','button-nofocus.png', \
                                 0, 'left')

          self.button2 = mmsv2gui.ButtonControl(0, 40, 100, 30, 'button2', \
                                 'Vera', '0xffffff', '0x000000', \
                                 'button-focus.png','button-nofocus.png', \
                                 0, 'left')

          self.add_control(self.button1)
          self.add_control(self.button2)

          self.button1.control_next(self.button2)
          self.button1.control_prev(self.button2)
          self.button2.control_next(self.button1)
          self.button2.control_prev(self.button1)

          self.set_focus(self.button1)


     def on_action(self, action):
          if action == 'back':
               self.close()
          elif action == 'action':
               if self.get_focus() == self.button1:
                    print 'You pressed button 1'
               elif self.get_focus() == self.button2:
                    print 'You pressed button 2'

win = MyWindow()
win.do_modal()
del win
```

Arguments for the first ButtonControl:

0 is the x position.
0 is the y position.
100 is the width.
30 is the height.
'button1' is the text on the button.
Vera' is the button text font.
'0xffffff' is the color of the button text (0xRRGGBB, RR=Red, GG=Green, BB=Blue) when button is **not** in focus.
'0x000000' is the color of the button text (0xRRGGBB, RR=Red, GG=Green, BB=Blue) when button is in focus.

'button-focus.png' is the path to the picture that will be displayed if the button is in focus.
'button-nofocus.png' is the path to the picture that will be displayed if the button is not in focus.
0 is the button text offset counting from the x position.
'left' is the alignment of the button text.

The “control_next” and “control_prev” methods are used to decide which control that will receive focus when the user fire the “next” or “prev” event. Don't forget to set which control that initially should have focus. This is done with the “set_focus” method.

You can also check which control that currently have focus. This is done with the “get_focus” method. In the code above this is done when the user fire the “action” event. This emulate the event that the user press a button.

ListControl and ListItem

A ListControl is used when you want to display text in a list. The user can scroll up and down in this list. A ListControl contain any number of ListItems. The ListItem decides what will be displayed in the list. Start out by creating a custom ListItem:

```
import mmsv2, mmsv2gui

class MyListItem(mmsv2gui.ListItem):
     def __init__(self, text1, text2):
          self.set_label(text1)
          self.set_label2(text2)
```

The “set_label” method sets the main text that will be displayed on a row in the list. The “set_label2” method sets the text that will be displayed at the rightmost on a row in the list. It is not necessary to call this method. This could be useful if you want to create a list of files, and also display what type of files it is (mp3, py, avi, ...).

```
class MyWindow(mmsv2gui.Window):
     def __init__(self):
          self.list = mmsv2gui.ListControl(0, 0, 200, 300, 'Vera', \
                                 '0xffffff', '0x000000','marked-row.png', \
                                 'notmarked-row.png', 0, 'left', 30, 0)

          self.add_control(self.list)

          for i in range(10):
               self.list.add_item(MyListItem('Filename', 'suffix'))

          self.set_focus(self.list)

     def on_action(self, action):
          if action == 'back':
               self.close()
          elif action == 'action':
               if self.get_focus() == self.list:
                    print self.list.get_selected_item().get_label()

win = MyWindow()
win.do_modal()
del win
```

0 is the x position.

0 is the y position.
200 is the width.
300 is the height.
Vera' is the list text font.
'0xffffff' is the color of the list item text (0xRRGGBB, RR=Red, GG=Green, BB=Blue) when list item is **not** in focus.
0x000000' is the color of the list item text (0xRRGGBB, RR=Red, GG=Green, BB=Blue) when list item is in focus.
'marked-row.png' is the path to the picture that will be displayed if a list item is the current list item, and the list is in focus.
'notmarked-row.png' is the path to the picture that will be displayed if a list item is not the current list item.
0 is the list text offset counting from the x position (this will only affect the text that is set with the "set_label" method).
'left' is the alignment of the list text (this will only affect the text that is set with the "set_label" method).
30 is the height every list item.
0 is the space between list items.

You add items by calling the "add_item" method. In the code above this is done ten times in the for loop. You can get the current list item by calling the "get_selected_item" method.

TextBoxControl

TextBoxControl is used if you want to display a relatively large piece of text on the screen.

```
import mmsv2, mmsv2gui

class MyWindow(mmsv2gui.Window):
     def __init__(self):
          self.textbox = mmsv2gui.TextBoxControl(0, 0, 300, 100, 'Vera', \
                                   '0xffffff', 30, 0)

          self.textbox.set_text('A large piece of text, it sure is, I \
                      probably have to write something more')

          self.add_control(self.textbox)
          self.set_focus(self.textbox)

     def on_action(self, action):
          if action == 'back':
               self.close()

win = MyWindow()
win.do_modal()
del win
```

0 is the x position.
0 is the y position.
300 is the width.
100 is the height.
Vera' is the text font.
'0xffffff' is the color of the text (0xRRGGBB, RR=Red, GG=Green, BB=Blue).
30 is the height of the text.
0 is the space between text rows.

You assign text to the TextBoxControl by calling the “set_text” method. If the assigned text doesn't fit the dimensions of the textbox, user can scroll up and down to read the entire text.

RectangleControl

You also have the possibility to draw rectangles on the screen. This is done with the RectangleControl.

```
import mmsv2, mmsv2gui

class MyWindow(mmsv2gui.Window):
     def __init__(self):
          self.rec = mmsv2gui.RectangleControl(0, 0, 100, 200, 55, \
                                  '0xffffff')

          self.add_control(self.rec)

     def on_action(self, action):
          if action == 'back':
               self.close()

win = MyWindow()
win.do_modal()
del win
```

0 is the x position.
0 is the y position.
100 is the width.
200 is the height.
55 is the alpha value (transparency).
'0xffffff' is the color of the rectangle (0xRRGGBB, RR=Red, GG=Green, BB=Blue).

How to add a child window

It's always useful to add a child window when you don't have enough space on the main screen. It's really easy, just create another window :-).

```
import mmsv2, mmsv2gui

class MainWindow(mmsv2gui.Window):
     def __init__(self):
          self.button = mmsv2gui.ButtonControl(0, 0, 150, 30, 'Press me', \
                        'Vera', '0xffffff','0x000000','button-focus.png', \
                        'button-nofocus.png', 20, 'left')

          self.add_control(self.button)
          self.set_focus(self.button)

     def on_action(self, action):
          if action == 'back':
               self.close()
          elif action == 'action':
               if self.get_focus() == self.button:
                    childwin = ChildWindow()
                    childwin.do_modal()
                    del childwin
```

```
class ChildWindow(mmsv2gui.Window):
     def __init__(self):
          self.label = mmsv2gui.LabelControl(0, 0, 200, 30, 'Press back', \
                                             'Vera', '0xffffff', 0, 'left')
          self.add_control(self.label)


     def on_action(self, action):
          if action == 'back':
               self.close()

win = MainWindow()
win.do_modal()
del win
```

Further reading

The python class documentation can be found in mmsv2.pdf and mmv2gui.pdf.

Author

Fredrik