# The Manor

*Python Extensions Reference*

Sep 12, 2004

MadWolf Software
support@madwolfsw.com

This manual assumes you are already familiar with Python (www.python.org). If you aren't already familiar with Python, no fear. If you have experience with other programming languages, it's fairly easy to pick up the basics of Python programming from looking at code samples.

Each script that is loaded is placed in a separate Python module. The cyborg script is placed in a module named "cyborg". Spot scripts are placed in modules named "spot_##" where ## is the ID of the spot. Spot scripts are also loaded in order, so later spots can import spot modules that have already been loaded.

In addition, all scripts that wish to make calls to The Manor client will need to import the manor module.

# Entry Points

If these routines are present in your script, they will be called as detailed. If your script does not contain at least one of these entry points, it won't be called by The Manor client software.

**mnr_signon**
cyborg only
Called when a connection is established to a Manor server.

**mnr_enter**(userID)
Called when a user enters the room.

**mnr_leave**(userID)
Called when a user leaves the room.

**mnr_spothit**(spotID)
spot script only
This is only called for the spot script of the spot the user has clicked on.

**mnr_spothitex**(spotID)
spot script only
This is called when the user clicks on any spot in the room.

chatText **mnr_inchat**(userID, chatText)
Called when chat text is received by the client, before it is displayed to the user.
UserID Which user initiated the message.
chatText: The text of the message

Your script should do any processing desired on the text, and then return the chatText.
Your script may modify the chatText if desired, or clear the text.

chatText **mnr_outchat**(chatText)
Called when chat text is sent by the user, before the text is actually sent to the server.

**mnr_statechange**(spotID, stateID)
Called when the state of a spot changes.

handled **mnr_render**(gworld, layer)
spot script only
Called during room rendering operations.
gworld The room display gworld. This is an opaque gworld.
layer What layer of the room is being rendered
0 Nothing has been rendered
1 Room background has been rendered
2 Spots have been rendered
3 All rendering completed
handled returning non zero stops any further rendering

handled **mnr_keydown**(char, key)
Called when the user presses a key
char Character code of the key pressed
key Key code of the key pressed.
*NOTE: Key codes differ between Macintosh and Windows systems*
handled returning non zero stops any further processing for the key down

handled **mnr_mousedown**(x, y)
spot script only
Called when the user presses a key
x, y Mouse position within the room
handled returning non zero stops any further processing for the key down

# Manor Calls

## *Output functions*

**say**(chatText)
Send chatText

**whisper**([userID, (userID...)], chatText)
Send chatText to the users indicated by userID list as a whisper

**logstr**(text)
Places text in the log

**logErrStr**(text)
Places text in the log as an error (red text)

**localMsg**(msg)
Displays a system message balloon. Only the user sees this message, it is not transmitted to the server.

**statusMsg** (msg, logit)
Set a status message

**dirtyRect**(rect)
Marks a section of the room as needing to be rendered.

## *User functions*

**isRegistered**()
Returns 1 if user is registered

userID **myID**
Returns this users ID

name **getUserName**(userID)
Returns the name of the user indicated by userID

count **getRoomUserCount**
Returns the number of users in the room

x,y **getUserPos**(userID)
Returns the position of the user indicated by userID

userID **getIdxRoomUser**(userIdx)
Returns the id for an indexed user. UserIdx is 0 <= userIdx < getRoomUserCount

## *Spot functions*

**hitSpot**(spotID)
Simulates user clicking a spot. Any smart spot actions will be taken after calling any spothit scripts

**setSpotState**(spotID, stateID, global)
Sets the current state of a spot
spotID The spot whose state will be set.
stateID state the spot is to be set to
global if 0, spot is set locally, if 1 it is set globally

*NOTE: If the spot contains state images, the new state image will not be displayed until the next time the client cycles.*

stateID **getSpotState**(spotID)
Returns the current state of a spot

title **getSpotTitle**(spotID)
Returns the title of a spot

count **getSpotCount**
Returns the number of spots in the room

spotID **getIdxSpot**(spotIdx)
Return the ID of the indexed spot. spotIdx is 0 <= spotIdx < getSpotCount

roomID **getSpotDest**(spotID)
Returns spots destination room ID

rect **getSpotBounds**(spotID)
Returns spots bounding rectangle

**inSpot**(spotID)
Return true if user is within the bounds of spotID.

**isLocked**(spotID)
Returns true if spotID is locked. If spotID does not exist or is not closable will return false.

**lock**(spotID)
Locks a spot. If spotID does not exist or is not closable an exception will be set.

**unlock**(spotID)
UnLocks a spot. If spotID does not exist or is not closable an exception will be set.

## *Prop functions*

**setProp**(propList)
Sets the current worn props to those in proplist.

propID **getTopPropID**()
Returns the id of the top most prop

**putProp**(propID, x, y)
Place a prop in the room

| | |
|---|---|
| propI | The prop to place in the room. |
| x | Horizontal position for the prop |
| y | Vertical position for the prop |

**clearProps**()
Remove all the worn props, i.e. naked

**dropProp**(x, y)
Drop the top prop in the the room at the location indicated by x and y

**donProp**(propID)
Put the prop on as the top most prop

**doffProp**()
Remove the top most prop

**numProps**()
Returns the number of currently worn props

**removeProp**(propID)
Removes propID from currently worn props

**faceColor**(r, g, b)
Change the face color to the rgb value in r, g, and b

**face**(faceNum)
Change the face to faceNum. FaceNum must be 0 – 15.

## *Navigation functions*

**gotoRoom**(roomNum)
Change the current room to roomNum

roomName **getRoomName**()
Returns the name of the current room

roomNum **getRoomNumber**()
Returns the number of the current room

siteName **getSiteName**()
Returns the name of the current Manor

**openConnection**(url)
Open a url. If the url is a manor url, a connection will be opened to the manor server. Otherwise the appropriate application will be launched for the URL.

**setPos**(x, y)
Set the users position

x,y **getMouseRmPos**()
Returns the current mouse positions clipped to the bounds of the room.

isDown **stillDown**()
Returns non-zero if the mouse button is still down.

## *Misc functions*

boolean **isKeyDown**(keyCode)
Returns 1 is the key is down. Note keycodes are different on Mac vs Windows systems.

clientType **getClientType**()
Returns the client type. 0 = Mac OS9, 1 = Windows, 2 = Mac OSX

mills **milliseconds**()
Returns the number of milliseconds (1000ths of a second) since system start

ticks **tickcount**()
Returns the number of ticks (60ths of a second) since system start

**setTimer**(ticks, function, params)
Sets a timer function.

| | |
|---|---|
| ticks | The tick time the timer should be triggered. |
| function | The function to be called when the timer triggers The function is of the form: func(param) |
| params | Parameters to pass to the trigger function. If you are calling this from a class, this must be “self”. |

**playSound**(sound)
Plays a sound file from the cache. If the sound file is not already in the cache an exception is raised and the call returns. "sound" can either be just the name of the file to play (i.e. "sound.wav") or it can be a full URL (i.e. "http://www.madwolfsw.com/sound.wav). In most cases you will just want to use the name of the file to use your sites sounds.

**fetchSound**(sound, function, params)
Checks a sound is in the cache. If the sound is not in the cache, or is out of date, downloads the sound to the cache. "sound" can either be just the name of the file to play (i.e. "sound.wav") or it can be a full URL (i.e. "http://www.madwolfsw.com/sound.wav). In most cases you will just want to use the name of the file to use your sites sounds.

| | |
|---|---|
| function | The function to be called when the sound is downloaded. The function is of the form: func(param) |
| params | Parameters to pass to the completion function. If you are calling this from a class, this must be “self”. |

*NOTE: If this is the first time the sound file has been used in a session it can take several seconds for the download to complete even if the sound is already cached. This is normal as it is the time it takes to verify the sound file is up to date.*

**fetchSoundObject**(sound, function, params)

Checks a sound is in the cache. If the sound is not in the cache, or is out of date, downloads the sound to the cache. "sound" can either be just the name of the file to load (i.e. "sound.wav") or it can be a full URL (i.e. "http://www.madwolfsw.com/sound.wav). In most cases you will just want to use the name of the file to use your sites sounds. Once the sound is available it will be passed as an object to your completion function.

| | |
|---|---|
| function | The function to be called when the sound is downloaded. The function is of the form: func(sound, params) |
| params | Parameters to pass to the completion function. If you are calling this from a class, this must be “self”. |

*NOTE: If this is the first time the sound file has been used in a session it can take several seconds for the download to complete even if the sound is already cached. This is normal as it is the time it takes to verify the sound file is up to date.*

**playSoundObject**(soundObject, interupt, loop)

Plays a sound object. Sound objects should be kept small as they are kept in memory. They are intended for use of sound effects in games.

| | |
|---|---|
| soundObject | The sound object returned by fetchSoundObject |
| interupt | if not zero any other sound currently playing will be stopped |
| loop | if not zero this sound will be played until stopped |

**stopSounds**()

Stops any sounds that may be currently playing

**broadcastData**(msgID, isGlobal, data)

Broadcasts data to all the users. Global broadcasting requires being authenticated to a group with global message permissions.

| | |
|---|---|
| msgID | 15 character string identifying the type of message |
| isGlobal | if not zero, message is to be broadcast server wide |
| data | any object to be sent |

**sendData**(msgID, [userID, (userID...)], data)

Sends data to specific users..

| | |
|---|---|
| msgID | 15 character string identifying the type of message |
| userID | list of users to send data to |
| data | any object to be sent |

**regDataReceive**(msgID, func, param)

Registers a function to receive message types

| | |
|---|---|
| msgID | 15 character string identifying the type of message |
| func | Function to be called when msgID is received. The function is of the form: func(userID, data, param) |
| param | parameter to pass to receive function |

# Object Modules

## *rect*

The rect object defines rectangles used by other graphics objects.

myRect = rect(top, left, bottom, right)

| Function | Behavior |
|---|---|
| offset(x, y) | Offset the rectangle by x, y pixels |
| inset(x,y) | Inset the rectangle by x, y pixels |
| t, l, b, r = bounds | returns the bounds of the rectangle |

| Unary | Behavior |
|---|---|
| \| | Union |
| & | Intersection |
| = | Copy |
| == | Equal to |
| != | Not equal |
| > | Area greater than |
| < | Area less than |

## *shape*

shapes are predefined line drawings that can be manipulated as a single object. They also draw slightly faster than drawing a comparable image with the gworld move and line functions.

myShape = shape()

| Function | Behavior |
|---|---|
| move(x, y) | Add a move point to the shape. x and y are relative to shape center. |
| line(x, y) | Add a line point to the shape. x and y are relative to shape center. |
| draw(gworld, x, y) | Draw the shape |
| rotate(a) | Rotate the image to angle in radians. Zero is straight up, angles proceeding clockwise. Rotation is always from the original shape definition. |
| rect = bounds() | Return the bounding rectangle |

## *gworld*

gworlds are offscreen graphics worlds. This is the workhorse object for graphics operations.

| Function | Behavior |
|---|---|
| foreColor(a, r, g, b) | Set the foreground color to alpha, red, green, blue. Alpha values are ignored when working with opaque gworlds.drawString(string) Draw a string at the current pen position |
| stringWidth(string) | Returns the pixel width of the string |
| fontHeight() | Returns the pixel height of the font |
| fontAscent() | Returns the ascent height of the font |
| penSize(h, v) | Set the horizontal and vertical size of the pen |
| moveTo(x, y) | Change the pen position |
| lineTo(x, y) | Draw a line from the current pen position |
| frameRect(rect) | Draw a frame around a rectangle |
| fillRect(rect) | Fill a rectangle with the foreground color |
| frameOval(rect) | Draw an oval within the bounds of a rectangle |
| fillOval(rect) | Fill an oval within the bounds of a rectangle |

# SAMPLE CODE

## Bouncing Text

*This was the initial test script for very early gworld functionality*

```
import manor

global xpos, ypos, xdir, ydir

xpos = 105
ypos = 112
xdir = 3
ydir = 3

def mover(param):
    global xpos, ypos, xdir, ydir

    manor.dirtyRect(rect(xpos - 50, ypos + 3, xpos + 50, ypos - 13))
    xpos += xdir
    ypos += ydir
    if (xpos > 510):
        xdir = -3
    if (xpos < 4):
        xdir = 3
    if (ypos > 380):
        ydir = -3
    if (ypos < 4):
        ydir = 3
    manor.dirtyRect(rect(xpos - 50, ypos + 3, xpos + 50, ypos - 13))
    manor.setTimer(manor.tickcount(), mover, 0)

def mnr_render(gworld, layer):
    global xpos, ypos, xdir, ydir

    handled = 0
    if layer == 1:
        gworld.moveTo(xpos, ypos)
        gworld.foreColor(0xFF, 0xFF, 0xFF, 0xFF)
        gworld.drawString("test")
        handled = 1
    return handled

def mnr_enter(userID):
    if userID == manor.myID() or manor.myID() == 0:
        manor.setTimer(manor.tickcount() + 1, mover, 0)
```

## Bouncing Ball

*A variation of the bouncing text to display a bouncing ball in response to the user saying "ball" or "no ball" to stop.  Assumes a room of 750x384*

```
import manor

global xpos, ypos, xdir, ydir
global ball_on
```

```
xpos = 105
ypos = 112
xdir = 1
ydir = 1
ball_on = 0

def mover(param):
    global xpos, ypos, xdir, ydir, ball_on

    manor.dirtyRect(rect(xpos - 5, ypos - 5, xpos + 6, ypos + 6))
    xpos += xdir
    ypos += ydir
    if (xpos > 740):
        xdir = -1
    if (xpos < 5):
        xdir = 1
    if (ypos > 379):
        ydir = -1
    if (ypos < 5):
        ydir = 1
    manor.dirtyRect(rect(xpos - 5, ypos - 5, xpos + 6, ypos + 6))

    if ball_on != 0:
        manor.setTimer(0, mover, 0)

def mnr_render(gworld, layer):
    global ball_on

    if (layer == 1) and (ball_on == 1):
        gworld.foreColor(0xFF, 0xFF, 0, 0)
        gworld.fillOval(rect(xpos - 5, ypos - 5, xpos + 5, ypos + 5))
    return 0

def mnr_outchat(chatText):
    global ball_on

    if chatText == "ball":
        ball_on = 1
        manor.setTimer(0, mover, 0)
        chatText = ""

    if chatText == "no ball":
        ball_on = 0
        chatText = ""

    return chatText
```