Generating texture list by python:

texture\_types = set(['tif', 'tga', 'png'])

used\_files = set()

for file\_node in [cmds.ls](http://cmds.ls)(type='file') :

   file\_path = cmds.getAttr("%s.fileTextureName" % file\_node)

   if file\_path.lower().rsplit('.', 1)[-1] in texture\_types:

       used\_files.add(file\_path)

print used\_files

logging and maya script editor:

import logging  
import maya.utils  
  
myLogger = logging.getLogger("MyLogger")  
myLogger.propagate = False  
handler = maya.utils.MayaGuiLogHandler()  
handler.setLevel(logging.INFO)  
formatter = logging.Formatter("%(asctime)s %(message)s")  
handler.setFormatter(formatter)  
myLogger.addHandler(handler)  
  
myLogger.warning('is when this event was logged.')

maya debug:

setenv MAYA\_DEBUG\_ENABLE\_CRASH\_REPORTING 1

It enables a dump of the last actions, saved into a file with a .crash suffix.

It a Windows only thing. You can attach Visual Studio to Maya then use  
the following command to print stuff for python into the debug stream.

import win32api;  
win32api.OutputDebugString ('some message I want to see')

MAYA\_CMD\_FILE\_OUTPUT

This variable enables the cmdFileOutput command during startup so the content in the script editor automatically outputs to a designated file.

If no file name is specified, then errors are sent to the standard error output, for example the Terminal window.

Example (for windows):   
maya.exe -log c:\debug.log   
  
There are a couple of other debug options that can be added to   
your .env file.   
  
MAYA\_DEBUG\_ENABLE\_CRASH\_REPORTING is an environmental variable that   
when set to 1 will cause Maya to try dumping debug files to your %temp   
% directory. It dumps stack traces and other things that maybe helpful   
for debugging plugin crashes. As far as I know you can not change the   
path that these files get written to.   
  
MAYA\_CMD\_FILE\_OUTPUT is also an environmental variable. When set to   
the path of a log file it will output everything from the script   
editor window into the log.

[**Embedding a Maya widget into a PyQt UI**](http://nathanhorne.com/?p=381)

import maya.OpenMayaUI as apiUI

from PyQt4 import QtGui, QtCore

import sip

def getMayaWindow():

ptr = apiUI.MQtUtil.mainWindow()

return sip.wrapinstance(long(ptr), QtCore.QObject)

def toQtObject(mayaName):

'''

Given the name of a Maya UI element of any type,

return the corresponding QWidget or QAction.

If the object does not exist, returns None

'''

ptr = apiUI.MQtUtil.findControl(mayaName)

if ptr is None:

ptr = apiUI.MQtUtil.findLayout(mayaName)

if ptr is None:

ptr = apiUI.MQtUtil.findMenuItem(mayaName)

if ptr is not None:

return sip.wrapinstance(long(ptr), QtCore.QObject)

class MayaSubWindow(QtGui.QMainWindow):

def \_\_init\_\_(self, parent=getMayaWindow()):

super(MayaSubWindow, self).\_\_init\_\_(parent)

self.executer = cmds.cmdScrollFieldExecuter(sourceType="python")

qtObj = toQtObject(self.executer)

#Fill the window, could use qtObj.setParent

#and then add it to a layout.

self.setCentralWidget(qtObj)

myWindow = MayaSubWindow()

myWindow.show()

## [Assigning to \_\_main\_\_ in Maya](http://nathanhorne.com/?p=71)

you can add objects to the \_\_main\_\_ namespace (The script editor’s interactive prompt), this is also where python will search when you use the MEL command python();

#import the interactive main module

import \_\_main\_\_

def someCommand():

'''This is a pretty basic function...'''

print 'someCommand called!'

#now assign it to an attribute in \_\_main\_\_

\_\_main\_\_.someCommand = someCommand

python("someCommand()");

// someCommand called!

## [Python API grab frame buffer to image](http://nathanhorne.com/?p=261)

This code will grab the frame buffer from the active viewport (You could also change it to be a specified viewport with little work) and write it to any format that MImage supports:

#Import api modules

import maya.OpenMaya as api

import maya.OpenMayaUI as apiUI

#Grab the last active 3d viewport

view = apiUI.M3dView.active3dView()

#read the color buffer from the view, and save the MImage to disk

image = api.MImage()

view.readColorBuffer(image, True)

image.writeToFile('C:/test.jpg', 'jpg')