################################################################################

# Copyright (C) 2015 Surfacingx/NaN #

# #

# This Program is free software; you can redistribute it and/or modify #

# it under the terms of the GNU General Public License as published by #

# the Free Software Foundation; either version 2, or (at your option) #

# any later version. #

# #

# This Program is distributed in the hope that it will be useful, #

# but WITHOUT ANY WARRANTY; without even the implied warranty of #

# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the #

# GNU General Public License for more details. #

# #

# You should have received a copy of the GNU General Public License #

# along with XBMC; see the file COPYING. If not, write to #

# the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA. #

# http://www.gnu.org/copyleft/gpl.html #

################################################################################

# Credits

# ----------

# Tobias Ussing And Henrik Mosgaard Jensen for parseDOM

# WhiteCream thread for clicking yes on dialog for unknown sources

import xbmc, xbmcvfs, xbmcaddon, xbmcgui,re, os, glob, thread

from datetime import datetime

try: from sqlite3 import dbapi2 as database

except: from pysqlite2 import dbapi2 as database

def main():

class enableAll():

def \_\_init\_\_(self):

self.databasepath = xbmc.translatePath('special://database/')

self.addons = xbmc.translatePath('special://home/addons/')

self.tempauto = xbmc.translatePath('special://home/userdata/autoexec\_temp.py')

self.dbfilename = self.latestDB()

self.dbfilename = os.path.join(self.databasepath, self.dbfilename)

self.swapUS()

if not os.path.exists(os.path.join(self.databasepath, self.dbfilename)):

xbmcgui.Dialog().notification("AutoExec.py", "No Addons27.db file")

self.log("DB File not found.")

return False

self.addonlist = glob.glob(os.path.join(self.addons, '\*/'))

self.disabledAddons = []

for folder in sorted(self.addonlist, key = lambda x: x):

addonxml = os.path.join(folder, 'addon.xml')

if os.path.exists(addonxml):

fold = folder.replace(self.addons, '')[1:-1]

f = open(addonxml)

a = f.read()

aid = parseDOM(a, 'addon', ret='id')

f.close()

try:

if len(aid) > 0: add = aid[0]

else: add = fold

xadd = xbmcaddon.Addon(id=add)

except:

try:

self.disabledAddons.append(add)

except:

self.log("Unabled to enable: %s" % folder, xbmc.LOGERROR)

if len(self.disabledAddons) > 0:

self.addonDatabase(self.disabledAddons, 1, True)

xbmc.executebuiltin('UpdateAddonRepos()')

xbmc.executebuiltin('UpdateLocalAddons()')

xbmc.executebuiltin("ReloadSkin()")

def log(self, msg, level=xbmc.LOGNOTICE):

try:

if isinstance(msg, unicode):

msg = '%s' % (msg.encode('utf-8'))

xbmc.log('[AutoExec.py]: %s' % msg, level)

except Exception as e:

try: xbmc.log('[AutoExec.py] Logging Failure: %s' % (e), xbmc.LOGERROR)

except: pass

def latestDB(self, DB="Addons"):

match = glob.glob(os.path.join(self.databasepath,'%s\*.db' % DB))

comp = '%s(.+?).db' % DB[1:]

highest = 0

for file in match:

try: check = int(re.compile(comp).findall(file)[0])

except Exception, e: check = 0; self.log(str(e))

if highest < check:

highest = check

return '%s%s.db' % (DB, highest)

def swapUS(self):

new = '"addons.unknownsources"'

value = 'true'

query = '{"jsonrpc":"2.0", "method":"Settings.GetSettingValue","params":{"setting":%s}, "id":1}' % (new)

response = xbmc.executeJSONRPC(query)

self.log("Unknown Sources Get Settings: %s" % str(response), xbmc.LOGDEBUG)

if 'false' in response:

thread.start\_new\_thread(self.dialogWatch, ())

xbmc.sleep(200)

query = '{"jsonrpc":"2.0", "method":"Settings.SetSettingValue","params":{"setting":%s,"value":%s}, "id":1}' % (new, value)

response = xbmc.executeJSONRPC(query)

xbmcgui.Dialog().notification("AutoExec.py", "Unknown Sources: Enabled")

self.log("Unknown Sources Set Settings: %s" % str(response), xbmc.LOGDEBUG)

def dialogWatch(self):

x = 0

while not xbmc.getCondVisibility("Window.isVisible(yesnodialog)") and x < 100:

x += 1

xbmc.sleep(100)

if xbmc.getCondVisibility("Window.isVisible(yesnodialog)"):

xbmc.executebuiltin('SendClick(11)')

def addonDatabase(self, addon=None, state=1, array=False):

installedtime = str(datetime.now())[:-7]

if os.path.exists(self.dbfilename):

try:

textdb = database.connect(self.dbfilename)

textexe = textdb.cursor()

except Exception, e:

self.log("DB Connection Error: %s" % str(e), xbmc.LOGERROR)

return False

else: return False

try:

if array == False:

textexe.execute('INSERT or IGNORE into installed (addonID , enabled, installDate) VALUES (?,?,?)', (addon, state, installedtime,))

textexe.execute('UPDATE installed SET enabled = ? WHERE addonID = ? ', (state, addon,))

else:

for item in addon:

textexe.execute('INSERT or IGNORE into installed (addonID , enabled, installDate) VALUES (?,?,?)', (item, state, installedtime,))

textexe.execute('UPDATE installed SET enabled = ? WHERE addonID = ? ', (state, item,))

textdb.commit()

textexe.close()

except Exception, e:

self.log("Erroring enabling addon: %s" % addon, xbmc.LOGERROR)

try:

xbmcgui.Dialog().notification("AutoExec.py", "Starting Script...")

firstRun = enableAll()

xbmcgui.Dialog().notification("AutoExec.py", "All Addons Enabled")

xbmcvfs.delete('special://userdata/autoexec.py')

xbmcvfs.copy('special://home/userdata/autoexec\_temp.py', 'special://userdata/autoexec.py')

xbmcvfs.delete('special://userdata/autoexec\_temp.py')

except Exception, e:

xbmcgui.Dialog().notification("AutoExec.py", "Error Check LogFile")

xbmc.log(str(e), xbmc.LOGERROR)

xbmcvfs.delete('special://userdata/autoexec.py')

xbmcvfs.copy('special://home/userdata/autoexec\_temp.py', 'special://userdata/autoexec.py')

xbmcvfs.delete('special://userdata/autoexec\_temp.py')

def parseDOM(html, name=u"", attrs={}, ret=False):

# Copyright (C) 2010-2011 Tobias Ussing And Henrik Mosgaard Jensen

if isinstance(html, str):

try:

html = [html.decode("utf-8")]

except:

html = [html]

elif isinstance(html, unicode):

html = [html]

elif not isinstance(html, list):

return u""

if not name.strip():

return u""

ret\_lst = []

for item in html:

temp\_item = re.compile('(<[^>]\*?\n[^>]\*?>)').findall(item)

for match in temp\_item:

item = item.replace(match, match.replace("\n", " "))

lst = []

for key in attrs:

lst2 = re.compile('(<' + name + '[^>]\*?(?:' + key + '=[\'"]' + attrs[key] + '[\'"].\*?>))', re.M | re.S).findall(item)

if len(lst2) == 0 and attrs[key].find(" ") == -1:

lst2 = re.compile('(<' + name + '[^>]\*?(?:' + key + '=' + attrs[key] + '.\*?>))', re.M | re.S).findall(item)

if len(lst) == 0:

lst = lst2

lst2 = []

else:

test = range(len(lst))

test.reverse()

for i in test:

if not lst[i] in lst2:

del(lst[i])

if len(lst) == 0 and attrs == {}:

lst = re.compile('(<' + name + '>)', re.M | re.S).findall(item)

if len(lst) == 0:

lst = re.compile('(<' + name + ' .\*?>)', re.M | re.S).findall(item)

if isinstance(ret, str):

lst2 = []

for match in lst:

attr\_lst = re.compile('<' + name + '.\*?' + ret + '=([\'"].[^>]\*?[\'"])>', re.M | re.S).findall(match)

if len(attr\_lst) == 0:

attr\_lst = re.compile('<' + name + '.\*?' + ret + '=(.[^>]\*?)>', re.M | re.S).findall(match)

for tmp in attr\_lst:

cont\_char = tmp[0]

if cont\_char in "'\"":

if tmp.find('=' + cont\_char, tmp.find(cont\_char, 1)) > -1:

tmp = tmp[:tmp.find('=' + cont\_char, tmp.find(cont\_char, 1))]

if tmp.rfind(cont\_char, 1) > -1:

tmp = tmp[1:tmp.rfind(cont\_char)]

else:

if tmp.find(" ") > 0:

tmp = tmp[:tmp.find(" ")]

elif tmp.find("/") > 0:

tmp = tmp[:tmp.find("/")]

elif tmp.find(">") > 0:

tmp = tmp[:tmp.find(">")]

lst2.append(tmp.strip())

lst = lst2

else:

lst2 = []

for match in lst:

endstr = u"</" + name

start = item.find(match)

end = item.find(endstr, start)

pos = item.find("<" + name, start + 1 )

while pos < end and pos != -1:

tend = item.find(endstr, end + len(endstr))

if tend != -1:

end = tend

pos = item.find("<" + name, pos + 1)

if start == -1 and end == -1:

temp = u""

elif start > -1 and end > -1:

temp = item[start + len(match):end]

elif end > -1:

temp = item[:end]

elif start > -1:

temp = item[start + len(match):]

if ret:

endstr = item[end:item.find(">", item.find(endstr)) + 1]

temp = match + temp + endstr

item = item[item.find(temp, item.find(match)) + len(temp):]

lst2.append(temp)

lst = lst2

ret\_lst += lst

return ret\_lst

if \_\_name\_\_ == '\_\_main\_\_':

main()