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

# Copyright (C) 2015 OpenELEQ #

# #

# 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 #

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

import os, re,glob, shutil, time, xbmc, xbmcaddon, thread, wizard as wiz, uservar

try:

import json as simplejson

except:

import simplejson

KODIV = float(xbmc.getInfoLabel("System.BuildVersion")[:4])

COLOR1 = uservar.COLOR1

COLOR2 = uservar.COLOR2

HOME = xbmc.translatePath('special://home/')

ADDONS = os.path.join(HOME, 'addons')

#DIALOG = xbmcgui.Dialog()

def getOld(old):

try:

old = '"%s"' % old

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

response = xbmc.executeJSONRPC(query)

response = simplejson.loads(response)

if response.has\_key('result'):

if response['result'].has\_key('value'):

return response ['result']['value']

except:

pass

return None

def setNew(new, value):

try:

new = '"%s"' % new

value = '"%s"' % value

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

response = xbmc.executeJSONRPC(query)

except:

pass

return None

def swapSkins(skin):

if skin == 'skin.confluence':

HOME = xbmc.translatePath('special://home/')

skinfold = os.path.join(HOME, 'userdata', 'addon\_data', 'skin.confluence')

settings = os.path.join(skinfold, 'settings.xml')

if not os.path.exists(settings):

string = '<settings>\n <setting id="FirstTimeRun" type="bool">true</setting>\n</settings>'

os.makedirs(skinfold)

f = open(settings, 'w'); f.write(string); f.close()

else: xbmcaddon.Addon(id='skin.confluence').setSetting('FirstTimeRun', 'true')

old = 'lookandfeel.skin'

value = skin

current = getOld(old)

new = old

setNew(new, value)

# if not xbmc.getCondVisibility(Skin.HasSetting(FirstTimeRun)):

# while xbmc.getCondVisibility('Window.IsVisible(1112)'):

# xbmc.executebuiltin('SendClick(100)')

def swapUS():

new = '"addons.unknownsources"'

value = 'true'

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

response = xbmc.executeJSONRPC(query)

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

if 'false' in response:

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

xbmc.sleep(200)

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

response = xbmc.executeJSONRPC(query)

wiz.LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), '[COLOR %s]Unknown Sources:[/COLOR] [COLOR %s]Enabled[/COLOR]' % (COLOR1, COLOR2))

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

def dialogWatch():

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)')

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

#######################################Still Needs Work#########################################

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

#def popUPmenu():

# fold = glob.glob(os.path.join(ADDONS, 'skin\*'))

# addonnames = []; addonids = []; addonfolds = []

# for folder in sorted(fold, key = lambda x: x):

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

# if os.path.exists(xml):

# foldername = os.path.split(folder[:-1])[1]

# f = open(xml)

# a = f.read()

# f.close()

# getid = parseDOM(a, 'addon', ret='id')

# getname = parseDOM(a, 'addon', ret='name')

# addid = foldername if len(getid) == 0 else getid[0]

# title = foldername if len(getname) == 0 else getname[0]

# temp = title.replace('[', '<').replace(']', '>')

# temp = re.sub('<[^<]+?>', '', temp)

# addonnames.append(temp)

# addonids.append(addid)

# addonfolds.append(foldername)

# #currskin = ["Current Skin -- %s" % currSkin()] + addonids

# select = DIALOG.select("Select the Skin you want to swap with.", addonids#currskin )

# if select == -1: return

# elif select == 1: addonids[select]

# swapSkins(addonids)

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