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

# /T /I #

# / |/ | .-~/ #

# T\ Y I |/ / \_ #

# /T | \I | I Y.-~/ #

# I l /I T\ | | l | T / #

# T\ | \ Y l /T | \I l \ ` l Y If your going to copy #

# \_\_ | \l \l \I l \_\_l l \ ` \_. | this addon just #

# \ ~-l `\ `\ \ \ ~\ \ `. .-~ | give credit! #

# \ ~-. "-. ` \ ^.\_ ^. "-. / \ | #

#.--~-.\_ ~- ` \_ ~-\_.-"-." .\_ /.\_ ." ./ Stop Deleting the #

# >--. ~-. .\_ ~>-" "\ 7 7 ] credits file! #

#^.\_\_\_~"--.\_ ~-{ .-~ . `\ Y . / | #

# <\_\_ ~"-. ~ /\_/ \ \I Y : | #

# ^-.\_\_ ~(\_/ \ >.\_: | l\_\_\_\_\_\_ #

# ^--.,\_\_\_.-~" /\_/ ! `-.~"--l\_ / ~"-. #

# (\_/ . ~( /' "~"--,Y -=b-. \_) #

# (\_/ . \ : / l c"~o \ #

# \ / `. . .^ \\_.-~"~--. ) #

# (\_/ . ` / / ! )/ #

# / / \_. '. .': / ' #

# ~(\_/ . / \_ ` .-<\_ #

# /\_/ . ' .-~" `. / \ \ ,z=. Surfacingx #

# ~( / ' : | K "-.~-.\_\_\_\_\_\_// Original Author #

# "-,. l I/ \\_ \_\_{--->.\_(==. #

# //( \ < ~"~" // #

# /' /\ \ \ ,v=. (( Fire TV Guru #

# .^. / /\ " }\_\_ //===- ` PyXBMCt LaYOUt #

# / / ' ' "-.,\_\_ {---(==- #

# .^ ' : T ~" ll #

# / . . . : | :! \ #

# (\_/ / | | j-" ~^ #

# ~-<\_(\_.^-~" #

# #

# Copyright (C) One of those Years.... #

# #

# 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 3 of the License, 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. #

# #

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

import xbmc, xbmcaddon, xbmcgui, xbmcplugin, os, sys, xbmcvfs, HTMLParser, glob, json

import shutil

import errno

import string

import random

import urllib2,urllib

import re

import downloader

import extract

import uservar

import skinSwitch

import time

import pyqrcode

from datetime import date, datetime, timedelta

try: from sqlite3 import dbapi2 as database

except: from pysqlite2 import dbapi2 as database

from string import digits

try:

from urllib2 import urlopen, Request, HTTPError, URLError

except ImportError:

from urllib.request import urlopen, Request, HTTPError, URLError

try:

import xml.etree.cElementTree as ET

except ImportError:

try:

import xml.etree.ElementTree as ET

except ImportError:

from xml.dom import minidom as DOM

ET = None

try :

import xml . etree . cElementTree as ET

from xml . dom import minidom as DOM

except ImportError :

try :

import xml . etree . ElementTree as ET

except ImportError :

from xml . dom import minidom as DOM

ET = None

user\_agent = None

ADDON\_ID = uservar.ADDON\_ID

ADDONTITLE = uservar.ADDONTITLE

ADDON = xbmcaddon.Addon(ADDON\_ID)

VERSION = ADDON.getAddonInfo('version')

USER\_AGENT = 'Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36 SE 2.X MetaSr 1.0'

DIALOG = xbmcgui.Dialog()

DP = xbmcgui.DialogProgress()

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

XBMC = xbmc.translatePath('special://xbmc/')

LOG = xbmc.translatePath('special://logpath/')

PROFILE = xbmc.translatePath('special://profile/')

TEMPDIR = xbmc.translatePath('special://temp')

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

USERDATA = os.path.join(HOME, 'userdata')

PLUGIN = os.path.join(ADDONS, ADDON\_ID)

PACKAGES = os.path.join(ADDONS, 'packages')

ADDOND = os.path.join(USERDATA, 'addon\_data')

ADDONDATA = os.path.join(USERDATA, 'addon\_data', ADDON\_ID)

ADVANCED = os.path.join(USERDATA, 'advancedsettings.xml')

SOURCES = os.path.join(USERDATA, 'sources.xml')

GUISETTINGS = os.path.join(USERDATA, 'guisettings.xml')

FAVOURITES = os.path.join(USERDATA, 'favourites.xml')

FAVdest = os.path.join(ADDONDATA, 'favs')

FAVfile = os.path.join(FAVdest, 'favourites.xml')

PROFILES = os.path.join(USERDATA, 'profiles.xml')

THUMBS = os.path.join(USERDATA, 'Thumbnails')

DATABASE = os.path.join(USERDATA, 'Database')

FANART = os.path.join(PLUGIN, 'fanart.jpg')

ICON = os.path.join(PLUGIN, 'icon.png')

ART = os.path.join(PLUGIN, 'resources', 'art')

WIZLOG = os.path.join(ADDONDATA, 'wizard.log')

WHITELIST = os.path.join(ADDONDATA, 'whitelist.txt')

QRCODES = os.path.join(ADDONDATA, 'QRCodes')

TEXTCACHE = os.path.join(ADDONDATA, 'Cache')

ARCHIVE\_CACHE = os.path.join(TEMPDIR, 'archive\_cache')

SKIN = xbmc.getSkinDir()

TODAY = date.today()

TOMORROW = TODAY + timedelta(days=1)

TWODAYS = TODAY + timedelta(days=2)

THREEDAYS = TODAY + timedelta(days=3)

ONEWEEK = TODAY + timedelta(days=7)

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

if KODIV > 17:

from resources.libs import zfile as zipfile #FTG mod for Kodi 18

else:

import zipfile

EXCLUDES = uservar.EXCLUDES

CACHETEXT = uservar.CACHETEXT

CACHEAGE = uservar.CACHEAGE if str(uservar.CACHEAGE).isdigit() else 30

BUILDFILE = uservar.BUILDFILE

APKFILE = uservar.APKFILE

YOUTUBEFILE = uservar.YOUTUBEFILE

ADDONFILE = uservar.ADDONFILE

ADVANCEDFILE = uservar.ADVANCEDFILE

AUTOUPDATE = uservar.AUTOUPDATE

WIZARDFILE = uservar.WIZARDFILE

NOTIFICATION = uservar.NOTIFICATION

ENABLE = uservar.ENABLE

AUTOINSTALL = uservar.AUTOINSTALL

REPOADDONXML = uservar.REPOADDONXML

REPOZIPURL = uservar.REPOZIPURL

CONTACT = uservar.CONTACT

COLOR1 = uservar.COLOR1

COLOR2 = uservar.COLOR2

COLOR3 = uservar.COLOR3

COLOR4 = uservar.COLOR4

INCLUDEVIDEO = ADDON.getSetting('includevideo')

INCLUDEALL = ADDON.getSetting('includeall')

INCLUDEBOB = ADDON.getSetting('includebob')

INCLUDEZEN = ADDON.getSetting('includezen')

INCLUDEPHOENIX = ADDON.getSetting('includephoenix')

INCLUDESPECTO = ADDON.getSetting('includespecto')

INCLUDEGENESIS = ADDON.getSetting('includegenesis')

INCLUDEEXODUS = ADDON.getSetting('includeexodus')

INCLUDEONECHAN = ADDON.getSetting('includeonechan')

INCLUDESALTS = ADDON.getSetting('includesalts')

INCLUDESALTSHD = ADDON.getSetting('includesaltslite')

SHOWADULT = ADDON.getSetting('adult')

WIZDEBUGGING = ADDON.getSetting('addon\_debug')

DEBUGLEVEL = ADDON.getSetting('debuglevel')

ENABLEWIZLOG = ADDON.getSetting('wizardlog')

CLEANWIZLOG = ADDON.getSetting('autocleanwiz')

CLEANWIZLOGBY = ADDON.getSetting('wizlogcleanby')

CLEANDAYS = ADDON.getSetting('wizlogcleandays')

CLEANSIZE = ADDON.getSetting('wizlogcleansize')

CLEANLINES = ADDON.getSetting('wizlogcleanlines')

INSTALLMETHOD = ADDON.getSetting('installmethod')

DEVELOPER = ADDON.getSetting('developer')

THIRDPARTY = ADDON.getSetting('enable3rd')

THIRD1NAME = ADDON.getSetting('wizard1name')

THIRD1URL = ADDON.getSetting('wizard1url')

THIRD2NAME = ADDON.getSetting('wizard2name')

THIRD2URL = ADDON.getSetting('wizard2url')

THIRD3NAME = ADDON.getSetting('wizard3name')

THIRD3URL = ADDON.getSetting('wizard3url')

BACKUPLOCATION = ADDON.getSetting('path') if not ADDON.getSetting('path') == '' else 'special://home/'

MYBUILDS = os.path.join(BACKUPLOCATION, 'My\_Builds', '')

LOGFILES = ['log', 'xbmc.old.log', 'kodi.log', 'kodi.old.log', 'spmc.log', 'spmc.old.log', 'tvmc.log', 'tvmc.old.log', 'firemc.log','firemc.old.log']

DEFAULTPLUGINS = ['metadata.album.universal', 'metadata.artists.universal', 'metadata.common.fanart.tv', 'metadata.common.imdb.com', 'metadata.common.musicbrainz.org', 'metadata.themoviedb.org', 'metadata.tvdb.com', 'service.xbmc.versioncheck']

MAXWIZSIZE = [100, 200, 300, 400, 500, 1000]

MAXWIZLINES = [100, 200, 300, 400, 500]

MAXWIZDATES = [1, 2, 3, 7]

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

def SYSINFO():

infoLabel = ['System.FriendlyName',

'System.BuildVersion',

'System.CpuUsage',

'System.ScreenMode',

'Network.IPAddress',

'Network.MacAddress',

'System.Uptime',

'System.TotalUptime',

'System.FreeSpace',

'System.UsedSpace',

'System.TotalSpace',

'System.Memory(free)',

'System.Memory(used)',

'System.Memory(total)']

data = []; x = 0

for info in infoLabel:

temp = getInfo(info)

y = 0

while temp == "Busy" and y < 10:

temp = getInfo(info); y += 1; log("%s sleep %s" % (info, str(y))); xbmc.sleep(200)

data.append(temp)

x += 1

name = data[0]

if platform() == 'android':

free,size,used = extsize()

storage\_free = free

storage\_used = used

storage\_total = size

#elif platform() == 'linux' or 'osx' or 'ios':

# storage\_free = None

# storage\_used = None

# storage\_total = None

else:

storage\_free = data[8] if 'Una' in data[8] else convertSize(int(float(data[8][:-8]))\*1024\*1024)

storage\_used = data[9] if 'Una' in data[9] else convertSize(int(float(data[9][:-8]))\*1024\*1024)

storage\_total = data[10] if 'Una' in data[10] else convertSize(int(float(data[10][:-8]))\*1024\*1024)

ram\_free = convertSize(int(float(data[11][:-2]))\*1024\*1024)

ram\_used = convertSize(int(float(data[12][:-2]))\*1024\*1024)

ram\_total = convertSize(int(float(data[13][:-2]))\*1024\*1024)

xbmc\_version=xbmc.getInfoLabel("System.BuildVersion")

version=float(xbmc\_version[:4])

if version >= 11.0 and version <= 11.9:

codename = 'Eden'

elif version >= 12.0 and version <= 12.9:

codename = 'Frodo'

elif version >= 13.0 and version <= 13.9:

codename = 'Gotham'

elif version >= 14.0 and version <= 14.9:

codename = 'Helix'

elif version >= 15.0 and version <= 15.9:

codename = 'Isengard'

elif version >= 16.0 and version <= 16.9:

codename = 'Jarvis'

elif version >= 17.0 and version <= 17.9:

codename = 'Krypton'

elif version >= 18.0 and version <= 18.9:

codename = 'Leia'

else: codename = "Decline"

picture = []; music = []; video = []; programs = []; repos = []; scripts = []; skins = []

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

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

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

if foldername == 'packages': continue

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

if os.path.exists(xml):

f = open(xml)

a = f.read()

prov = re.compile("<provides>(.+?)</provides>").findall(a)

if len(prov) == 0:

if foldername.startswith('skin'): skins.append(foldername)

elif foldername.startswith('repo'): repos.append(foldername)

else: scripts.append(foldername)

elif not (prov[0]).find('executable') == -1: programs.append(foldername)

elif not (prov[0]).find('video') == -1: video.append(foldername)

elif not (prov[0]).find('audio') == -1: music.append(foldername)

elif not (prov[0]).find('image') == -1: picture.append(foldername)

return picture, music, video, programs, repos, scripts, skins, codename, version, name,storage\_free ,storage\_used, storage\_total, ram\_free, ram\_used, ram\_total

def extsize():

stat = os.statvfs('/storage/emulated/0/')

size = stat.f\_frsize \* stat.f\_blocks/1024/1024

free = stat.f\_frsize \* stat.f\_bfree/1024/1024

used = size - free

storage\_free = convertSize(int(float(free))\*1024\*1024)

storage\_total = convertSize(int(float(used))\*1024\*1024)

storage\_used = convertSize(int(float(size))\*1024\*1024)

return storage\_free,storage\_used,storage\_total

def net\_info():

import re

import json

from urllib2 import urlopen

infoLabel = ['Network.IPAddress',

'Network.MacAddress',]

data = []; x = 0

for info in infoLabel:

temp = getInfo(info)

y = 0

while temp == "Busy" and y < 10:

temp = getInfo(info); y += 1; log("%s sleep %s" % (info, str(y))); xbmc.sleep(200)

data.append(temp)

x += 1

try:

url = 'http://extreme-ip-lookup.com/json/'

req = urllib2.Request(url)

req.add\_header('User-Agent', 'Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.9.0.3) Gecko/2008092417 Firefox/3.0.3')

response = urllib2.urlopen(req)

geo = json.load(response)

except:

url = 'http://ip-api.com/json'

req = urllib2.Request(url)

req.add\_header('User-Agent', 'Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.9.0.3) Gecko/2008092417 Firefox/3.0.3')

response = urllib2.urlopen(req)

geo = json.load(response)

mac = data[1]

inter\_ip = data[0]

ip=geo['query']

isp=geo['org']

city = geo['city']

country=geo['country']

state=geo['region']

return mac,inter\_ip,ip,city,state,country,isp

import json

global json

global jsonfile

jsonfile = os.path.join(ADDONDATA, 'var.json')

def writejson(path, specs):

with open(jsonfile, 'w') as fj:

json.dump(specs, fj, indent=2)

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

###### Settings Items #####

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

def getS(name):

try: return ADDON.getSetting(name)

except: return False

def setS(name, value):

try: ADDON.setSetting(name, value)

except: return False

def openS(name=""):

ADDON.openSettings(name)

def clearS(type):

build = {'buildname':'', 'buildversion':'', 'buildtheme':'', 'latestversion':'', 'lastbuildcheck':'2018-01-01'}

install = {'installed':'false', 'extract':'', 'errors':''}

default = {'defaultskinignore':'false', 'defaultskin':'', 'defaultskinname':''}

lookfeel = ['default.enablerssfeeds', 'default.font', 'default.rssedit', 'default.skincolors', 'default.skintheme', 'default.skinzoom', 'default.soundskin', 'default.startupwindow', 'default.stereostrength']

if type == 'build':

for set in build:

setS(set, build[set])

for set in install:

setS(set, install[set])

for set in default:

setS(set, default[set])

for set in lookfeel:

setS(set, '')

elif type == 'default':

for set in default:

setS(set, default[set])

for set in lookfeel:

setS(set, '')

elif type == 'install':

for set in install:

setS(set, install[set])

elif type == 'lookfeel':

for set in lookfeel:

setS(set, '')

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

###### Display Items ######

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

# def TextBoxes(heading,announce):

# class TextBox():

# WINDOW=10147

# CONTROL\_LABEL=1

# CONTROL\_TEXTBOX=5

# def \_\_init\_\_(self,\*args,\*\*kwargs):

# ebi("ActivateWindow(%d)" % (self.WINDOW, )) # activate the text viewer window

# self.win=xbmcgui.Window(self.WINDOW) # get window

# xbmc.sleep(500) # give window time to initialize

# self.setControls()

# def setControls(self):

# self.win.getControl(self.CONTROL\_LABEL).setLabel(heading) # set heading

# try: f=open(announce); text=f.read()

# except: text=announce

# self.win.getControl(self.CONTROL\_TEXTBOX).setText(str(text))

# return

# TextBox()

# while xbmc.getCondVisibility('Window.IsVisible(10147)'):

# xbmc.sleep(500)

ACTION\_PREVIOUS\_MENU = 10 ## ESC action

ACTION\_NAV\_BACK = 92 ## Backspace action

ACTION\_MOVE\_LEFT = 1 ## Left arrow key

ACTION\_MOVE\_RIGHT = 2 ## Right arrow key

ACTION\_MOVE\_UP = 3 ## Up arrow key

ACTION\_MOVE\_DOWN = 4 ## Down arrow key

ACTION\_MOUSE\_WHEEL\_UP = 104 ## Mouse wheel up

ACTION\_MOUSE\_WHEEL\_DOWN = 105 ## Mouse wheel down

ACTION\_MOVE\_MOUSE = 107 ## Down arrow key

ACTION\_SELECT\_ITEM = 7 ## Number Pad Enter

ACTION\_BACKSPACE = 110 ## ?

ACTION\_MOUSE\_LEFT\_CLICK = 100

ACTION\_MOUSE\_LONG\_CLICK = 108

def TextBox(title, msg):

class TextBoxes(xbmcgui.WindowXMLDialog):

def onInit(self):

self.title = 101

self.msg = 102

self.scrollbar = 103

self.okbutton = 201

self.showdialog()

def showdialog(self):

self.getControl(self.title).setLabel(title)

self.getControl(self.msg).setText(msg)

self.setFocusId(self.scrollbar)

def onClick(self, controlId):

if (controlId == self.okbutton):

self.close()

def onAction(self, action):

if action == ACTION\_PREVIOUS\_MENU: self.close()

elif action == ACTION\_NAV\_BACK: self.close()

tb = TextBoxes( "Textbox.xml" , ADDON.getAddonInfo('path'), 'DefaultSkin', title=title, msg=msg)

tb.doModal()

del tb

def highlightText(msg):

msg = msg.replace('\n', '[NL]')

matches = re.compile("-->Python callback/script returned the following error<--(.+?)-->End of Python script error report<--").findall(msg)

for item in matches:

string = '-->Python callback/script returned the following error<--%s-->End of Python script error report<--' % item

msg = msg.replace(string, '[COLOR red]%s[/COLOR]' % string)

msg = msg.replace('WARNING', '[COLOR yellow]WARNING[/COLOR]').replace('ERROR', '[COLOR red]ERROR[/COLOR]').replace('[NL]', '\n').replace(': EXCEPTION Thrown (PythonToCppException) :', '[COLOR red]: EXCEPTION Thrown (PythonToCppException) :[/COLOR]')

msg = msg.replace('\\\\', '\\').replace(HOME, '')

return msg

def LogNotify(title, message, times=2000, icon=ICON,sound=False):

DIALOG.notification(title, message, icon, int(times), sound)

#ebi('XBMC.Notification(%s, %s, %s, %s)' % (title, message, times, icon))

def percentage(part, whole):

return 100 \* float(part)/float(whole)

def addonUpdates(do=None):

setting = '"general.addonupdates"'

if do == 'set':

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

response = xbmc.executeJSONRPC(query)

match = re.compile('{"value":(.+?)}').findall(response)

if len(match) > 0: default = match[0]

else: default = 0

setS('default.addonupdate', str(default))

query = '{"jsonrpc":"2.0", "method":"Settings.SetSettingValue","params":{"setting":%s,"value":%s}, "id":1}' % (setting, '2')

response = xbmc.executeJSONRPC(query)

elif do == 'reset':

try:

value = int(float(getS('default.addonupdate')))

except:

value = 0

if not value in [0, 1, 2]: value = 0

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

response = xbmc.executeJSONRPC(query)

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

###### Build Info #########

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

def stripcolortags(string):

colortag = "COLOR"

if colortag in string:

opentag = string.find(']')

firstpart = string[opentag+1:]

closetag = firstpart.find('[')

string = firstpart[:closetag]

return string

def checkBuild(name, ret, burl=None):

link = openURL(BUILDFILE).replace('\n','').replace('\r','').replace('\t','').replace('gui=""', 'gui="http://"').replace('theme=""', 'theme="http://"')

match = re.compile('name="%s".+?ersion="(.+?)".+?rl="(.+?)".+?ui="(.+?)".+?odi="(.+?)".+?heme="(.+?)".+?con="(.+?)".+?anart="(.+?)".+?review="(.+?)".+?dult="(.+?)".+?escription="(.+?)"' % name).findall(link)

if len(match) > 0:

for version, url, gui, kodi, theme, icon, fanart, preview, adult, description in match:

if ret == 'version': return version

elif ret == 'url': return url

elif ret == 'gui': return gui

elif ret == 'kodi': return kodi

elif ret == 'theme': return theme

elif ret == 'icon': return icon

elif ret == 'fanart': return fanart

elif ret == 'preview': return preview

elif ret == 'adult': return adult

elif ret == 'description': return description

elif ret == 'all': return name, version, url, gui, kodi, theme, icon, fanart, preview, adult, description

else: return False

def checkAddon(name, ret):

link = openURL(ADDONFILE).replace('\n','').replace('\r','').replace('\t','')

match = re.compile('name="%s".+?lugin="(.+?)".+?rl="(.+?)"' % name).findall(link)

if len(match) > 0:

for plugin, url in match:

if ret == 'url': return url

else: return False

def checkAPK(name, ret):

link = openURL(APKFILE).replace('\n','').replace('\r','').replace('\t','')

match = re.compile('name="%s".+?rl="(.+?)"' % name).findall(link)

if len(match) > 0:

for plugin, url in match:

if ret == 'url': return url

else: return False

def checkTheme(name, theme, ret):

themeurl = checkBuild(name, 'theme')

if not workingURL(themeurl) == True: return False

link = openURL(themeurl).replace('\n','').replace('\r','').replace('\t','')

match = re.compile('name="%s".+?rl="(.+?)".+?con="(.+?)".+?anart="(.+?)".+?dult=(.+?).+?escription="(.+?)"' % theme).findall(link)

if len(match) > 0:

for url, icon, fanart, adult, description in match:

if ret == 'url': return url

elif ret == 'icon': return icon

elif ret == 'fanart': return fanart

elif ret == 'adult': return adult

elif ret == 'description': return description

elif ret == 'all': return name, theme, url, icon, fanart, adult, description

else: return False

def checkWizard(ret):

if not workingURL(WIZARDFILE) == True: return False

link = openURL(WIZARDFILE).replace('\n','').replace('\r','').replace('\t','')

match = re.compile('id="%s".+?ersion="(.+?)".+?ip="(.+?)"' % ADDON\_ID).findall(link)

if len(match) > 0:

for version, zip in match:

if ret == 'version': return version

elif ret == 'zip': return zip

elif ret == 'all': return ADDON\_ID, version, zip

else: return False

def checkInfo(url):

if not workingURL(url) == True: return False

link = openURL(url).replace('\n','').replace('\r','').replace('\t','')

match = re.compile('name="(.+?)".+?xtracted="(.+?)".+?ipsize="(.+?)".+?kin="(.+?)".+?reated="(.+?)".+?rograms="(.+?)".+?ideo="(.+?)".+?usic="(.+?)".+?icture="(.+?)".+?epos="(.+?)".+?cripts="(.+?)"').findall(link)

if len(match) > 0:

for name, extracted, zipsize, skin, created, programs, video, music, picture, repos, scripts in match:

return name, extracted, zipsize, skin, created, programs, video, music, picture, repos, scripts

else: return False

def buildCount(ver=None, url=None):

if url == None: url = BUILDFILE

link = textCache(url).replace('\n','').replace('\r','').replace('\t','')

match = re.compile('name="(.+?)".+?odi="(.+?)".+?dult="(.+?)"').findall(link)

total = 0; count15 = 0; count16 = 0; count17 = 0; count18 = 0; hidden = 0; adultcount = 0

if len(match) > 0:

for name, kodi, adult in match:

if not SHOWADULT == 'true' and adult.lower() == 'yes': hidden += 1; adultcount +=1; continue

if not DEVELOPER == 'true' and strTest(name): hidden += 1; continue

kodi = int(float(kodi))

total += 1

if kodi == 18: count18 += 1

elif kodi == 17: count17 += 1

elif kodi == 16: count16 += 1

elif kodi <= 15: count15 += 1

return total, count15, count16, count17, count18, adultcount, hidden

def strTest(string):

a = (string.lower()).split(' ')

if 'test' in a: return True

else: return False

def themeCount(name, count=True):

themefile = checkBuild(name, 'theme')

if themefile == 'http://': return False

link = openURL(themefile).replace('\n','').replace('\r','').replace('\t','')

match = re.compile('name="(.+?)"').findall(link)

if len(match) == 0: return False

if count == True: return len(match)

else:

themes = []

for item in match:

themes.append(item)

return themes

def thirdParty(url=None):

if url == None: return

link = openURL(url).replace('\n','').replace('\r','').replace('\t','')

match = re.compile('name="(.+?)".+?ersion="(.+?)".+?rl="(.+?)".+?odi="(.+?)".+?con="(.+?)".+?anart="(.+?)".+?dult="(.+?)".+?escription="(.+?)"').findall(link)

match2 = re.compile('name="(.+?)".+?rl="(.+?)".+?mg="(.+?)".+?anart="(.+?)".+?escription="(.+?)"').findall(link)

if len(match) > 0:

return True, match

elif len(match2) > 0:

return False, match2

else:

return False, []

def basecode(text, encode=True):

import binascii

if encode == True:

msg = binascii.hexlify(text)

else:

msg = binascii.unhexlify(text)

return msg

def flushOldCache():

try: age = int(float(CACHEAGE))

except: age = 30

match = glob.glob(os.path.join(TEXTCACHE,'\*.txt'))

for file in match:

file\_modified = datetime.fromtimestamp(os.path.getmtime(file))

if datetime.now() - file\_modified > timedelta(minutes=age):

log("Found: %s" % file)

os.remove(file)

def textCache(url):

try: age = int(float(CACHEAGE))

except: age = 30

if CACHETEXT.lower() == 'yes':

spliturl = url.split('/')

if not os.path.exists(TEXTCACHE): os.makedirs(TEXTCACHE)

file = xbmc.makeLegalFilename(os.path.join(TEXTCACHE, spliturl[-1]+'\_'+spliturl[-2]+'.txt'))

if os.path.exists(file):

file\_modified = datetime.fromtimestamp(os.path.getmtime(file))

if datetime.now() - file\_modified > timedelta(minutes=age):

if workingURL(url):

os.remove(file)

if not os.path.exists(file):

if not workingURL(url): return False

f = open(file, 'w+')

textfile = openURL(url)

content = basecode(textfile, True)

f.write(content)

f.close()

f = open(file, 'r')

a = basecode(f.read(), False)

f.close()

return a

else:

textfile = openURL(url)

return textfile

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

###### URL Checks #########

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

def workingURL(url):

if url in ['http://', 'https://', '']: return False

check = 0; status = ''

while check < 3:

check += 1

try:

req = urllib2.Request(url)

req.add\_header('User-Agent', USER\_AGENT)

response = urllib2.urlopen(req)

response.close()

status = True

break

except Exception, e:

status = str(e)

log("Working Url Error: %s [%s]" % (e, url))

xbmc.sleep(500)

return status

def openURL(url):

req = urllib2.Request(url)

req.add\_header('User-Agent', USER\_AGENT)

response = urllib2.urlopen(req)

link=response.read()

response.close()

return link

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

###### Misc Functions #####

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

def getKeyboard( default="", heading="", hidden=False ):

keyboard = xbmc.Keyboard( default, heading, hidden )

keyboard.doModal()

if keyboard.isConfirmed():

return unicode( keyboard.getText(), "utf-8" )

return default

def getSize(path, total=0):

for dirpath, dirnames, filenames in os.walk(path):

for f in filenames:

fp = os.path.join(dirpath, f)

total += os.path.getsize(fp)

return total

def getTotal(path, total=0):

for root, dirs, files in os.walk(path):

total += len(files)

return total

#str(file\_count)

def convertSize(num, suffix='B'):

for unit in ['', 'K', 'M', 'G']:

if abs(num) < 1024.0:

return "%3.02f %s%s" % (num, unit, suffix)

num /= 1024.0

return "%.02f %s%s" % (num, 'G', suffix)

def getCacheSize():

PROFILEADDONDATA = os.path.join(PROFILE,'addon\_data')

dbfiles = [

(os.path.join(ADDOND, 'plugin.video.phstreams', 'cache.db')),

(os.path.join(ADDOND, 'plugin.video.zen', 'cache.db')),

(os.path.join(ADDOND, 'plugin.video.bob', 'cache.db')),

(os.path.join(ADDOND, 'plugin.video.specto', 'cache.db')),

(os.path.join(ADDOND, 'plugin.video.genesis', 'cache.db')),

(os.path.join(ADDOND, 'plugin.video.exodus', 'cache.db')),

(os.path.join(DATABASE, 'onechannelcache.db')),

(os.path.join(DATABASE, 'saltscache.db')),

(os.path.join(DATABASE, 'saltshd.lite.db'))]

cachelist = [

(ADDOND),

(os.path.join(HOME,'cache')),

(os.path.join(HOME,'temp')),

(os.path.join('/private/var/mobile/Library/Caches/AppleTV/Video/', 'Other')),

(os.path.join('/private/var/mobile/Library/Caches/AppleTV/Video/', 'LocalAndRental')),

(os.path.join(ADDOND,'script.module.simple.downloader')),

(os.path.join(ADDOND,'plugin.video.itv','Images'))]

if not PROFILEADDONDATA == ADDOND:

cachelist.append(os.path.join(PROFILEADDONDATA,'script.module.simple.downloader'))

cachelist.append(os.path.join(PROFILEADDONDATA,'plugin.video.itv','Images'))

cachelist.append(PROFILEADDONDATA)

totalsize = 0

for item in cachelist:

if not os.path.exists(item): continue

if not item in [ADDOND, PROFILEADDONDATA]:

totalsize = getSize(item, totalsize)

else:

for root, dirs, files in os.walk(item):

for d in dirs:

if 'cache' in d.lower() and not d.lower() in ['meta\_cache']:

totalsize = getSize(os.path.join(root, d), totalsize)

if INCLUDEVIDEO == 'true':

files = []

if INCLUDEALL == 'true': files = dbfiles

else:

if INCLUDEURANUS == 'true': files.append(os.path.join(ADDOND, 'plugin.video.uranus', 'cache.db'))

if INCLUDECOVEN == 'true': files.append(os.path.join(ADDOND, 'plugin.video.covenant', 'cache.db'))

if INCLUDEINCUR == 'true': files.append(os.path.join(ADDOND, 'plugin.video.incursion', 'cache.db'))

if INCLUDENEPTUNE == 'true': files.append(os.path.join(ADDOND, 'plugin.video.neptune', 'cache.db'))

if INCLUDESUBZERO == 'true': files.append(os.path.join(ADDOND, 'plugin.video.subzero', 'database.db'))

if INCLUDEPLACEN == 'true': files.append(os.path.join(ADDOND, 'plugin.video.placenta', 'cache.db'))

if INCLUDEINCUR == 'true': files.append(os.path.join(ADDOND, 'plugin.video.incursion', 'cache.db'))

if INCLUDESTREAMH == 'true': files.append(os.path.join(ADDOND, 'plugin.video.streamhub', 'cache.db'))

if INCLUDENOTSURE == 'true': files.append(os.path.join(ADDOND, 'plugin.video.sedundnes', 'cache.db'))

if INCLUDEATHEFL == 'true': files.append(os.path.join(ADDOND, 'plugin.video.AtTheFlix', 'database.db'))

if INCLUDEMANCAVE == 'true': files.append(os.path.join(ADDOND, 'plugin.video.mancaveflix', 'database.db'))

if INCLUDESTARTEC == 'true': files.append(os.path.join(ADDOND, 'plugin.video.StarTec', 'database.db'))

if INCLUDEDEATH == 'true': files.append(os.path.join(DATABASE, 'DEATHScache.db'))

if INCLUDEUKTURK == 'true': files.append(os.path.join(DATABASE, 'UKTurk.db'))

if len(files) > 0:

for item in files: totalsize = getSize(item, totalsize)

else: log("Clear Cache: Clear Video Cache Not Enabled", xbmc.LOGNOTICE)

return totalsize

def getInfo(label):

try: return xbmc.getInfoLabel(label)

except: return False

def removeFolder(path):

log("Deleting Folder: %s" % path, xbmc.LOGNOTICE)

try: shutil.rmtree(path,ignore\_errors=True, onerror=None)

except: return False

def removeFile(path):

log("Deleting File: %s" % path, xbmc.LOGNOTICE)

try: os.remove(path)

except: return False

def currSkin():

return xbmc.getSkinDir()

def cleanHouse(folder, ignore=False):

log(folder)

total\_files = 0; total\_folds = 0

for root, dirs, files in os.walk(folder):

if ignore == False: dirs[:] = [d for d in dirs if d not in EXCLUDES]

file\_count = 0

file\_count += len(files)

if file\_count >= 0:

for f in files:

try:

os.unlink(os.path.join(root, f))

total\_files += 1

except:

try:

shutil.rmtree(os.path.join(root, f))

except:

log("Error Deleting %s" % f, xbmc.LOGERROR)

for d in dirs:

total\_folds += 1

try:

shutil.rmtree(os.path.join(root, d))

total\_folds += 1

except:

log("Error Deleting %s" % d, xbmc.LOGERROR)

return total\_files, total\_folds

def emptyfolder(folder):

total = 0

for root, dirs, files in os.walk(folder, topdown=True):

dirs[:] = [d for d in dirs if d not in EXCLUDES]

file\_count = 0

file\_count += len(files) + len(dirs)

if file\_count == 0:

shutil.rmtree(os.path.join(root))

total += 1

log("Empty Folder: %s" % root, xbmc.LOGNOTICE)

return total

def log(msg, level=xbmc.LOGDEBUG):

if not os.path.exists(ADDONDATA): os.makedirs(ADDONDATA)

if not os.path.exists(WIZLOG): f = open(WIZLOG, 'w'); f.close()

if WIZDEBUGGING == 'false': return False

if DEBUGLEVEL == '0': return False

if DEBUGLEVEL == '1' and not level in [xbmc.LOGNOTICE, xbmc.LOGERROR, xbmc.LOGSEVERE, xbmc.LOGFATAL]: return False

if DEBUGLEVEL == '2': level = xbmc.LOGNOTICE

try:

if isinstance(msg, unicode):

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

xbmc.log('%s: %s' % (ADDONTITLE, msg), level)

except Exception as e:

try: xbmc.log('Logging Failure: %s' % (e), level)

except: pass

if ENABLEWIZLOG == 'true':

lastcheck = getS('nextcleandate') if not getS('nextcleandate') == '' else str(TODAY)

if CLEANWIZLOG == 'true' and lastcheck <= str(TODAY): checkLog()

with open(WIZLOG, 'a') as f:

line = "[%s %s] %s" % (datetime.now().date(), str(datetime.now().time())[:8], msg)

f.write(line.rstrip('\r\n')+'\n')

def FTGlog(msg, level=xbmc.LOGDEBUG):

if not os.path.exists(ADDONDATA): os.makedirs(ADDONDATA)

if not os.path.exists(WIZLOG): f = open(WIZLOG, 'w'); f.close()

if WIZDEBUGGING == 'false': return False

if DEBUGLEVEL == '0': return False

if DEBUGLEVEL == '1' and not level in [xbmc.LOGNOTICE, xbmc.LOGERROR, xbmc.LOGSEVERE, xbmc.LOGFATAL]: return False

if DEBUGLEVEL == '2': level = xbmc.LOGNOTICE

try:

if isinstance(msg, unicode):

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

xbmc.log('%s: %s' % ('<[ FTG GUI Log ]>', msg), level)

except Exception as e:

try: xbmc.log('Logging Failure: %s' % (e), level)

except: pass

def checkLog():

nextclean = getS('nextcleandate')

next = TOMORROW

if CLEANWIZLOGBY == '0':

keep = TODAY - timedelta(days=MAXWIZDATES[int(float(CLEANDAYS))])

x = 0

f = open(WIZLOG); a = f.read(); f.close(); lines = a.split('\n')

for line in lines:

if str(line[1:11]) >= str(keep):

break

x += 1

newfile = lines[x:]

writing = '\n'.join(newfile)

f = open(WIZLOG, 'w'); f.write(writing); f.close()

elif CLEANWIZLOGBY == '1':

maxsize = MAXWIZSIZE[int(float(CLEANSIZE))]\*1024

f = open(WIZLOG); a = f.read(); f.close(); lines = a.split('\n')

if os.path.getsize(WIZLOG) >= maxsize:

start = len(lines)/2

newfile = lines[start:]

writing = '\n'.join(newfile)

f = open(WIZLOG, 'w'); f.write(writing); f.close()

elif CLEANWIZLOGBY == '2':

f = open(WIZLOG); a = f.read(); f.close(); lines = a.split('\n')

maxlines = MAXWIZLINES[int(float(CLEANLINES))]

if len(lines) > maxlines:

start = len(lines) - int(maxlines/2)

newfile = lines[start:]

writing = '\n'.join(newfile)

f = open(WIZLOG, 'w'); f.write(writing); f.close()

setS('nextcleandate', str(next))

def latestDB(DB):

if DB in ['Addons', 'ADSP', 'Epg', 'MyMusic', 'MyVideos', 'Textures', 'TV', 'ViewModes']:

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

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

highest = 0

for file in match :

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

except: check = 0

if highest < check :

highest = check

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

else: return False

def viewFile(name, url):

return

def forceText():

cleanHouse(TEXTCACHE)

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), '[COLOR %s]Text Files Flushed![/COLOR]' % (COLOR2))

def addonId(add):

try:

return xbmcaddon.Addon(id=add)

except:

return False

def toggleDependency(name, DP=None):

dep=os.path.join(ADDONS, name, 'addon.xml')

if os.path.exists(dep):

source = open(dep,mode='r'); link=source.read(); source.close();

match = parseDOM(link, 'import', ret='addon')

for depends in match:

if not 'xbmc.python' in depends:

dependspath=os.path.join(ADDONS, depends)

if not DP == None:

DP.update("","Checking Dependency [COLOR yellow]%s[/COLOR] for [COLOR yellow]%s[/COLOR]" % (depends, name),"")

if os.path.exists(dependspath):

toggleAddon(name, 'true')

xbmc.sleep(100)

def toggleAdult():

do = DIALOG.yesno(ADDONTITLE, "[COLOR %s]Would you like to [COLOR %s]Enable[/COLOR] or [COLOR %s]Disable[/COLOR] all Adult addons?[/COLOR]" % (COLOR2, COLOR1, COLOR1), yeslabel="[B][COLOR green]Enable[/COLOR][/B]", nolabel="[B][COLOR red]Disable[/COLOR][/B]")

state = 'true' if do == 1 else 'false'

goto = 'Enabling' if do == 1 else 'Disabling'

link = openURL('http://noobsandnerds.com/TI/AddonPortal/adult.php').replace('\n','').replace('\r','').replace('\t','')

list = re.compile('i="(.+?)"').findall(link)

found = []

for item in list:

fold = os.path.join(ADDONS, item)

if os.path.exists(fold):

found.append(item)

toggleAddon(item, state, True)

log("[Toggle Adult] %s %s" % (goto, item), xbmc.LOGNOTICE)

if len(found) > 0:

if DIALOG.yesno(ADDONTITLE, "[COLOR %s]Would you like to view a list of the addons that where %s?[/COLOR]" % (COLOR2, goto.replace('ing', 'ed')), yeslabel="[B][COLOR green]View List[/COLOR][/B]", nolabel="[B][COLOR red]Cancel[/COLOR][/B]"):

editlist = '[CR]'.join(found)

TextBox(ADDONTITLE, "[COLOR %s]Here are a list of the addons that where %s for Adult Content:[/COLOR][CR][CR][COLOR %s]%s[/COLOR]" % (COLOR1, goto.replace('ing', 'ed'), COLOR2, editlist))

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s][COLOR %s]%d[/COLOR] Adult Addons %s[/COLOR]" % (COLOR2, COLOR1, count, goto.replace('ing', 'ed')))

forceUpdate(True)

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]No Adult Addons Found[/COLOR]" % COLOR2)

def createTemp(plugin):

temp = os.path.join(PLUGIN, 'resources', 'tempaddon.xml')

f = open(temp, 'r'); r = f.read(); f.close()

plugdir = os.path.join(ADDONS, plugin)

if not os.path.exists(plugdir): os.makedirs(plugdir)

a = open(os.path.join(plugdir, 'addon.xml'), 'w')

a.write(r.replace('testid', plugin).replace('testversion', '0.0.1'))

a.close()

log("%s: wrote addon.xml" % plugin)

def fixmetas():

idlist = ['plugin.video.metalliq', 'plugin.video.meta', 'script.renegadesmeta']

#temp = os.path.join(PLUGIN, 'resources', 'tempaddon.xml')

#f = open(temp, 'r'); r = f.read(); f.close()

for item in idlist:

fold = os.path.join(ADDOND, item)

if os.path.exists(fold):

storage = os.path.join(fold, '.storage')

if os.path.exists(storage):

cleanHouse(storage)

removeFolder(storage)

#if not os.path.exists(os.path.join(fold, 'addon.xml')): continue

#a = open(os.path.join(fold, 'addon.xml'), 'w')

#a.write(r.replace('testid', item).replace('testversion', '0.0.1'))

#a.close()

#log("%s: re-wrote addon.xml" % item)

def toggleAddon(id, value, over=None):

log("toggling %s" % id)

# if KODIV >= 17:

# log("kodi 17 way")

# goto = 0 if value == 'false' else 1

# addonDatabase(id, goto)

# if not over == None:

# forceUpdate(True)

# return

addonid = id

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

if os.path.exists(addonxml):

f = open(addonxml)

b = f.read()

tid = parseDOM(b, 'addon', ret='id')

tname = parseDOM(b, 'addon', ret='name')

tservice = parseDOM(b, 'extension', ret='library', attrs = {'point': 'xbmc.service'})

try:

if len(tid) > 0:

addonid = tid[0]

if len(tservice) > 0:

log("We got a live one, stopping script: %s" % match[0], xbmc.LOGDEBUG)

ebi('StopScript(%s)' % os.path.join(ADDONS, addonid))

ebi('StopScript(%s)' % addonid)

ebi('StopScript(%s)' % os.path.join(ADDONS, addonid, tservice[0]))

xbmc.sleep(500)

except:

pass

query = '{"jsonrpc":"2.0", "method":"Addons.SetAddonEnabled","params":{"addonid":"%s","enabled":%s}, "id":1}' % (addonid, value)

response = xbmc.executeJSONRPC(query)

if 'error' in response and over == None:

v = 'Enabling' if value == 'true' else 'Disabling'

DIALOG.ok(ADDONTITLE, "[COLOR %s]Error %s [COLOR %s]%s[/COLOR]" % (COLOR2, v, COLOR1 , id), "Check to make sure the addon list is upto date and try again.[/COLOR]")

forceUpdate()

def addonInfo(add, info):

addon = addonId(add)

if addon: return addon.getAddonInfo(info)

else: return False

def whileWindow(window, active=False, count=0, counter=15):

windowopen = getCond('Window.IsActive(%s)' % window)

log("%s is %s" % (window, windowopen), xbmc.LOGDEBUG)

while not windowopen and count < counter:

log("%s is %s(%s)" % (window, windowopen, count))

windowopen = getCond('Window.IsActive(%s)' % window)

count += 1

xbmc.sleep(500)

while windowopen:

active = True

log("%s is %s" % (window, windowopen), xbmc.LOGDEBUG)

windowopen = getCond('Window.IsActive(%s)' % window)

xbmc.sleep(250)

return active

def id\_generator(size=6, chars=string.ascii\_uppercase + string.digits):

return ''.join(random.choice(chars) for \_ in range(size))

def generateQR(url, filename):

if not os.path.exists(QRCODES): os.makedirs(QRCODES)

imagefile = os.path.join(QRCODES,'%s.png' % filename)

qrIMG = pyqrcode.create(url)

qrIMG.png(imagefile, scale=10)

return imagefile

def createQR():

url = getKeyboard('', "%s: Insert the URL for the QRCode." % ADDONTITLE)

if url == "": LogNotify("[COLOR %s]Create QR[/COLOR]" % COLOR1, '[COLOR %s]Create QR Code Cancelled![/COLOR]' % COLOR2); return

if not url.startswith('http://') and not url.startswith('https://'): LogNotify("[COLOR %s]Create QR[/COLOR]" % COLOR1, '[COLOR %s]Not a Valid URL![/COLOR]' % COLOR2); return

if url == 'http://' or url == 'https://': LogNotify("[COLOR %s]Create QR[/COLOR]" % COLOR1, '[COLOR %s]Not a Valid URL![/COLOR]' % COLOR2); return

working = workingURL(url)

if not working == True:

if not DIALOG.yesno(ADDONTITLE, "[COLOR %s]It seems the your enter isnt working, Would you like to create it anyways?[/COLOR]" % COLOR2, "[COLOR %s]%s[/COLOR]" % (COLOR1, working), yeslabel="[B][COLOR red]Yes Create[/COLOR][/B]", nolabel="[B][COLOR green]No Cancel[/COLOR][/B]"):

return

name = getKeyboard('', "%s: Insert the name for the QRCode." % ADDONTITLE)

name = "QrImage\_%s" % id\_generator(6) if name == "" else name

image = generateQR(url, name)

DIALOG.ok(ADDONTITLE, "[COLOR %s]The QRCode image has been created and is located in the addondata directory:[/COLOR]" % COLOR2, "[COLOR %s]%s[/COLOR]" % (COLOR1, image.replace(HOME, '')))

def cleanupBackup():

mybuilds = xbmc.translatePath(MYBUILDS)

folder = glob.glob(os.path.join(mybuilds, "\*"))

list = []; filelist = []

if len(folder) == 0:

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Backup Location: Empty[/COLOR]" % (COLOR2))

return

for item in sorted(folder, key=os.path.getmtime):

filelist.append(item)

base = item.replace(mybuilds, '')

if os.path.isdir(item):

list.append('/%s/' % base)

elif os.path.isfile(item):

list.append(base)

list = ['--- Remove All Items ---'] + list

selected = DIALOG.select("%s: Select the items to remove from 'MyBuilds'." % ADDONTITLE, list)

if selected == -1:

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Clean Up Cancelled![/COLOR]" % COLOR2)

elif selected == 0:

if DIALOG.yesno(ADDONTITLE, "[COLOR %s]Would you like to clean up all items in your 'My\_Builds' folder?[/COLOR]" % COLOR2, "[COLOR %s]%s[/COLOR]" % (COLOR1, MYBUILDS), yeslabel="[B][COLOR green]Clean Up[/COLOR][/B]", nolabel="[B][COLOR red]No Cancel[/COLOR][/B]"):

clearedfiles, clearedfolders = cleanHouse(xbmc.translatePath(MYBUILDS))

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Removed Files: [COLOR %s]%s[/COLOR] / Folders:[/COLOR] [COLOR %s]%s[/COLOR]" % (COLOR2, COLOR1, clearedfiles, COLOR1, clearedfolders))

else:

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Clean Up Cancelled![/COLOR]" % COLOR2)

else:

path = filelist[selected-1]; passed = False

if DIALOG.yesno(ADDONTITLE, "[COLOR %s]Would you like to remove [COLOR %s]%s[/COLOR] from 'My\_Builds' folder?[/COLOR]" % (COLOR2, COLOR1, list[selected]), "[COLOR %s]%s[/COLOR]" % (COLOR1, path), yeslabel="[B][COLOR green]Clean Up[/COLOR][/B]", nolabel="[B][COLOR red]No Cancel[/COLOR][/B]"):

if os.path.isfile(path):

try:

os.remove(path)

passed = True

except:

log("Unable to remove: %s" % path)

else:

cleanHouse(path)

try:

shutil.rmtree(path)

passed = True

except Exception ,e:

log("Error removing %s" % path, xbmc.LOGNOTICE)

if passed: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]%s Removed![/COLOR]" % (COLOR2, list[selected]))

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Error Removing %s![/COLOR]" % (COLOR2, list[selected]))

else:

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Clean Up Cancelled![/COLOR]" % COLOR2)

def getCond(type):

return xbmc.getCondVisibility(type)

def ebi(proc):

xbmc.executebuiltin(proc)

def refresh():

return##ebi('Container.Refresh()')

def splitNotify(notify):

link = openURL(notify).replace('\r','').replace('\t','').replace('\n', '[CR]')

if link.find('|||') == -1: return False, False

id, msg = link.split('|||')

if msg.startswith('[CR]'): msg = msg[4:]

return id.replace('[CR]', ''), msg

def forceUpdate(silent=False):

ebi('UpdateAddonRepos()')

ebi('UpdateLocalAddons()')

if silent == False: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), '[COLOR %s]Forcing Addon Updates[/COLOR]' % COLOR2)

def convertSpecial(url, over=False):

total = fileCount(url); start = 0

DP.create(ADDONTITLE, "[COLOR %s]Changing Physical Paths To Special" % COLOR2, "", "Please Wait[/COLOR]")

for root, dirs, files in os.walk(url):

for file in files:

start += 1

perc = int(percentage(start, total))

if file.endswith(".xml") or file.endswith(".hash") or file.endswith("properies"):

DP.update(perc, "[COLOR %s]Scanning: [COLOR %s]%s[/COLOR]" % (COLOR2, COLOR1, root.replace(HOME, '')), "[COLOR %s]%s[/COLOR]" % (COLOR1, file), "Please Wait[/COLOR]")

a = open(os.path.join(root, file)).read()

encodedpath = urllib.quote(HOME)

encodedpath2 = urllib.quote(HOME).replace('%3A','%3a').replace('%5C','%5c')

b = a.replace(HOME, 'special://home/').replace(encodedpath, 'special://home/').replace(encodedpath2, 'special://home/')

f = open((os.path.join(root, file)), mode='w')

f.write(str(b))

f.close()

if DP.iscanceled():

DP.close()

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Convert Path Cancelled[/COLOR]" % COLOR2)

sys.exit()

DP.close()

log("[Convert Paths to Special] Complete", xbmc.LOGNOTICE)

if over == False: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Convert Paths to Special: Complete![/COLOR]" % COLOR2)

def clearCrash():

files = []

for file in glob.glob(os.path.join(LOG, '\*crashlog\*.\*')):

files.append(file)

if len(files) > 0:

if DIALOG.yesno(ADDONTITLE, '[COLOR %s]Would you like to delete the Crash logs?' % COLOR2, '[COLOR %s]%s[/COLOR] Files Found[/COLOR]' % (COLOR1, len(files)), yeslabel="[B][COLOR green]Remove Logs[/COLOR][/B]", nolabel="[B][COLOR red]Keep Logs[/COLOR][/B]"):

for f in files:

os.remove(f)

LogNotify('[COLOR %s]Clear Crash Logs[/COLOR]' % COLOR1, '[COLOR %s]%s Crash Logs Removed[/COLOR]' % (COLOR2, len(files)))

else: LogNotify('[COLOR %s]%s[/COLOR]' % (COLOR1, ADDONTITLE), '[COLOR %s]Clear Crash Logs Cancelled[/COLOR]' % COLOR2)

else: LogNotify('[COLOR %s]Clear Crash Logs[/COLOR]' % COLOR1, '[COLOR %s]No Crash Logs Found[/COLOR]' % COLOR2)

def hidePassword():

if DIALOG.yesno(ADDONTITLE, "[COLOR %s]Would you like to [COLOR %s]hide[/COLOR] all passwords when typing in the add-on settings menus?[/COLOR]" % (COLOR2, COLOR1), yeslabel="[B][COLOR green]hide Passwords[/COLOR][/B]", nolabel="[B][COLOR red]No Cancel[/COLOR][/B]"):

count = 0

for folder in glob.glob(os.path.join(ADDONS, '\*/')):

sett = os.path.join(folder, 'resources', 'settings.xml')

if os.path.exists(sett):

f = open(sett).read()

match = parseDOM(f, 'addon', ret='id')

for line in match:

if 'pass' in line:

if not 'option="hidden"' in line:

try:

change = line.replace('/', 'option="hidden" /')

f.replace(line, change)

count += 1

log("[Hide Passwords] found in %s on %s" % (sett.replace(HOME, ''), line), xbmc.LOGDEBUG)

except:

pass

f2 = open(sett, mode='w'); f2.write(f); f2.close()

LogNotify("[COLOR %s]Hide Passwords[/COLOR]" % COLOR1, "[COLOR %s]%s items changed[/COLOR]" % (COLOR2, count))

log("[Hide Passwords] %s items changed" % count, xbmc.LOGNOTICE)

else: log("[Hide Passwords] Cancelled", xbmc.LOGNOTICE)

def unhidePassword():

if DIALOG.yesno(ADDONTITLE, "[COLOR %s]Would you like to [COLOR %s]unhide[/COLOR] all passwords when typing in the add-on settings menus?[/COLOR]" % (COLOR2, COLOR1), yeslabel="[B][COLOR green]Unhide Passwords[/COLOR][/B]", nolabel="[B][COLOR red]No Cancel[/COLOR][/B]"):

count = 0

for folder in glob.glob(os.path.join(ADDONS, '\*/')):

sett = os.path.join(folder, 'resources', 'settings.xml')

if os.path.exists(sett):

f = open(sett).read()

match = parseDOM(f, 'addon', ret='id')

for line in match:

if 'pass' in line:

if 'option="hidden"' in line:

try:

change = line.replace('option="hidden"', '')

f.replace(line, change)

count += 1

log("[Unhide Passwords] found in %s on %s" % (sett.replace(HOME, ''), line), xbmc.LOGDEBUG)

except:

pass

f2 = open(sett, mode='w'); f2.write(f); f2.close()

LogNotify("[COLOR %s]Unhide Passwords[/COLOR]" % COLOR1, "[COLOR %s]%s items changed[/COLOR]" % (COLOR2, count))

log("[Unhide Passwords] %s items changed" % count, xbmc.LOGNOTICE)

else: log("[Unhide Passwords] Cancelled", xbmc.LOGNOTICE)

def wizardUpdate(startup=None):

if workingURL(WIZARDFILE):

ver = checkWizard('version')

zip = checkWizard('zip')

if ver > VERSION:

yes = DIALOG.yesno(ADDONTITLE, '[COLOR %s]There is a new version of the [COLOR %s]%s[/COLOR]!' % (COLOR2, COLOR1, ADDONTITLE), 'Would you like to download [COLOR %s]v%s[/COLOR]?[/COLOR]' % (COLOR1, ver), nolabel='[B][COLOR red]Remind Me Later[/COLOR][/B]', yeslabel="[B][COLOR green]Update Wizard[/COLOR][/B]")

if yes:

log("[Auto Update Wizard] Installing wizard v%s" % ver, xbmc.LOGNOTICE)

DP.create(ADDONTITLE,'[COLOR %s]Downloading Update...' % COLOR2,'', 'Please Wait[/COLOR]')

lib=os.path.join(PACKAGES, '%s-%s.zip' % (ADDON\_ID, ver))

try: os.remove(lib)

except: pass

downloader.download(zip, lib, DP)

xbmc.sleep(2000)

DP.update(0,"", "Installing %s update" % ADDONTITLE)

percent, errors, error = extract.all(lib, ADDONS, DP, True)

DP.close()

xbmc.sleep(1000)

ebi('UpdateAddonRepos()')

ebi('UpdateLocalAddons()')

xbmc.sleep(1000)

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE),'[COLOR %s]Add-on updated[/COLOR]' % COLOR2)

log("[Auto Update Wizard] Wizard updated to v%s" % ver, xbmc.LOGNOTICE)

reloadProfile()

return

else: log("[Auto Update Wizard] Install New Wizard Ignored: %s" % ver, xbmc.LOGNOTICE)

else:

if not startup: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]No New Version of Wizard[/COLOR]" % COLOR2)

log("[Auto Update Wizard] No New Version v%s" % ver, xbmc.LOGNOTICE)

else: log("[Auto Update Wizard] Url for wizard file not valid: %s" % WIZARDFILE, xbmc.LOGNOTICE)

def convertText():

TEXTFILES = os.path.join(ADDONDATA, 'TextFiles')

if not os.path.exists(TEXTFILES): os.makedirs(TEXTFILES)

DP.create(ADDONTITLE,'[COLOR %s][B]Converting Text:[/B][/COLOR]' % (COLOR2),'', 'Please Wait')

if not BUILDFILE == 'http://':

filename = os.path.join(TEXTFILES, 'builds.txt')

writing = '';x = 0

a = openURL(BUILDFILE).replace('\n','').replace('\r','').replace('\t','')

DP.update(0,'[COLOR %s][B]Converting Text:[/B][/COLOR] [COLOR %s]Builds.txt[/COLOR]' % (COLOR2, COLOR1),'', 'Please Wait')

if WIZARDFILE == BUILDFILE:

try:

addonid, version, url = checkWizard('all')

writing = 'id="%s"\n' % addonid

writing += 'version="%s"\n' % version

writing += 'zip="%s"\n' % url

except:

pass

match = re.compile('name="(.+?)".+?ersion="(.+?)".+?rl="(.+?)".+?ui="(.+?)".+?odi="(.+?)".+?heme="(.+?)".+?con="(.+?)".+?anart="(.+?)"').findall(a)

match2 = re.compile('name="(.+?)".+?ersion="(.+?)".+?rl="(.+?)".+?ui="(.+?)".+?odi="(.+?)".+?heme="(.+?)".+?con="(.+?)".+?anart="(.+?)".+?review="(.+?)"+?dult="(.+?)".+?escription="(.+?)"').findall(a)

if len(match2) == 0:

for name, version, url, gui, kodi, theme, icon, fanart in match:

x += 1

DP.update(int(percentage(x, len(match2))), '', "[COLOR %s]%s[/COLOR]" % (COLOR1, name))

if not writing == '': writing += '\n'

writing += 'name="%s"\n' % name

writing += 'version="%s"\n' % version

writing += 'url="%s"\n' % url

writing += 'minor="http://"\n'

writing += 'gui="%s"\n' % gui

writing += 'kodi="%s"\n' % kodi

writing += 'theme="%s"\n' % theme

writing += 'icon="%s"\n' % icon

writing += 'fanart="%s"\n' % fanart

writing += 'preview="http://"\n'

writing += 'adult="no"\n'

writing += 'info="http://"\n'

writing += 'description="Download %s from %s"\n' % (name, ADDONTITLE)

if not theme == 'http://':

filename2 = os.path.join(TEXTFILES, '%s\_theme.txt' % name)

themewrite = ''; x2 = 0

a = openURL(theme).replace('\n','').replace('\r','').replace('\t','')

DP.update(0,'[COLOR %s][B]Converting Text:[/B][/COLOR] [COLOR %s]%s\_theme.txt[/COLOR]' % (COLOR2, COLOR1, name),'', 'Please Wait')

match3 = re.compile('name="(.+?)".+?rl="(.+?)".+?con="(.+?)".+?anart="(.+?)".+?escription="(.+?)"').findall(a)

for name, url, icon, fanart, description in match3:

x2 += 1

DP.update(int(percentage(x2, len(match2))), '', "[COLOR %s]%s[/COLOR]" % (COLOR1, name))

if not themewrite == '': themewrite += '\n'

themewrite += 'name="%s"\n' % name

themewrite += 'url="%s"\n' % url

themewrite += 'icon="%s"\n' % icon

themewrite += 'fanart="%s"\n' % fanart

themewrite += 'adult="no"\n'

themewrite += 'description="%s"\n' % description

f = open(filename2, 'w'); f.write(themewrite); f.close()

else:

for name, version, url, gui, kodi, theme, icon, fanart, preview, adult, description in match2:

x += 1

DP.update(int(percentage(x, len(match2))), '', "[COLOR %s]%s[/COLOR]" % (COLOR1, name))

if not writing == '': writing += '\n'

writing += 'name="%s"\n' % name

writing += 'version="%s"\n' % version

writing += 'url="%s"\n' % url

writing += 'minor="http://"\n'

writing += 'gui="%s"\n' % gui

writing += 'kodi="%s"\n' % kodi

writing += 'theme="%s"\n' % theme

writing += 'icon="%s"\n' % icon

writing += 'fanart="%s"\n' % fanart

writing += 'preview="%s"\n' % preview

writing += 'adult="%s"\n' % adult

writing += 'info="http://"\n'

writing += 'description="%s"\n' % description

if not theme == 'http://':

filename2 = os.path.join(TEXTFILES, '%s\_theme.txt' % name)

themewrite = ''; x2 = 0

a = openURL(theme).replace('\n','').replace('\r','').replace('\t','')

DP.update(0,'[COLOR %s][B]Converting Text:[/B][/COLOR] [COLOR %s]%s\_theme.txt[/COLOR]' % (COLOR2, COLOR1, name),'', 'Please Wait')

match3 = re.compile('name="(.+?)".+?rl="(.+?)".+?con="(.+?)".+?anart="(.+?)".+?escription="(.+?)"').findall(a)

for name, url, icon, fanart, description in match3:

x2 += 1

DP.update(int(percentage(x2, len(match2))), '', "[COLOR %s]%s[/COLOR]" % (COLOR1, name))

if not themewrite == '': themewrite += '\n'

themewrite += 'name="%s"\n' % name

themewrite += 'url="%s"\n' % url

themewrite += 'icon="%s"\n' % icon

themewrite += 'fanart="%s"\n' % fanart

themewrite += 'adult="no"\n'

themewrite += 'description="%s"\n' % description

f = open(filename2, 'w'); f.write(themewrite); f.close()

f = open(filename, 'w'); f.write(writing); f.close()

if not APKFILE == 'http://':

filename = os.path.join(TEXTFILES, 'apks.txt')

writing = ''; x = 0

a = openURL(APKFILE).replace('\n','').replace('\r','').replace('\t','')

DP.update(0,'[COLOR %s][B]Converting Text:[/B][/COLOR] [COLOR %s]Apks.txt[/COLOR]' % (COLOR2, COLOR1), '', 'Please Wait')

match = re.compile('name="(.+?)".+?rl="(.+?)".+?con="(.+?)".+?anart="(.+?)"').findall(a)

match2 = re.compile('name="(.+?)".+?rl="(.+?)".+?con="(.+?)".+?anart="(.+?)".+?dult="(.+?)".+?escription="(.+?)"').findall(a)

if len(match2) == 0:

for name, url, icon, fanart in match:

x += 1

DP.update(int(percentage(x, len(match))), '', "[COLOR %s]%s[/COLOR]" % (COLOR1, name))

if not writing == '': writing += '\n'

writing += 'name="%s"\n' % name

writing += 'section="no"'

writing += 'url="%s"\n' % url

writing += 'icon="%s"\n' % icon

writing += 'fanart="%s"\n' % fanart

writing += 'adult="no"\n'

writing += 'description="Download %s from %s"\n' % (name, ADDONTITLE)

else:

for name, url, icon, fanart, adult, description in match2:

x += 1

DP.update(int(percentage(x, len(match2))), '', "[COLOR %s]%s[/COLOR]" % (COLOR1, name))

if not writing == '': writing += '\n'

writing += 'name="%s"\n' % name

writing += 'section="no"'

writing += 'url="%s"\n' % url

writing += 'icon="%s"\n' % icon

writing += 'fanart="%s"\n' % fanart

writing += 'adult="%s"\n' % adult

writing += 'description="%s"\n' % description

f = open(filename, 'w'); f.write(writing); f.close()

if not YOUTUBEFILE == 'http://':

filename = os.path.join(TEXTFILES, 'youtube.txt')

writing = ''; x = 0

a = openURL(YOUTUBEFILE).replace('\n','').replace('\r','').replace('\t','')

DP.update(0,'[COLOR %s][B]Converting Text:[/B][/COLOR] [COLOR %s]YouTube.txt[/COLOR]' % (COLOR2, COLOR1), '', 'Please Wait')

match = re.compile('name="(.+?)".+?rl="(.+?)".+?con="(.+?)".+?anart="(.+?)".+?escription="(.+?)"').findall(a)

for name, url, icon, fanart, description in match:

x += 1

DP.update(int(percentage(x, len(match))), '', "[COLOR %s]%s[/COLOR]" % (COLOR1, name))

if not writing == '': writing += '\n'

writing += 'name="%s"\n' % name

writing += 'section="no"'

writing += 'url="%s"\n' % url

writing += 'icon="%s"\n' % icon

writing += 'fanart="%s"\n' % fanart

writing += 'description="%s"\n' % description

f = open(filename, 'w'); f.write(writing); f.close()

if not ADVANCEDFILE == 'http://':

filename = os.path.join(TEXTFILES, 'advancedsettings.txt')

writing = ''; x = 0

a = openURL(ADVANCEDFILE).replace('\n','').replace('\r','').replace('\t','')

DP.update(0,'[COLOR %s][B]Converting Text:[/B][/COLOR] [COLOR %s]AdvancedSettings.txt[/COLOR]' % (COLOR2, COLOR1), '', 'Please Wait')

match = re.compile('name="(.+?)".+?rl="(.+?)".+?con="(.+?)".+?anart="(.+?)".+?escription="(.+?)"').findall(a)

for name, url, icon, fanart, description in match:

x += 1

DP.update(int(percentage(x, len(match))), '', "[COLOR %s]%s[/COLOR]" % (COLOR1, name))

if not writing == '': writing += '\n'

writing += 'name="%s"\n' % name

writing += 'section="no"'

writing += 'url="%s"\n' % url

writing += 'icon="%s"\n' % icon

writing += 'fanart="%s"\n' % fanart

writing += 'description="%s"\n' % description

f = open(filename, 'w'); f.write(writing); f.close()

DP.close()

DIALOG.ok(ADDONTITLE, '[COLOR %s]Your text files have been converted to 0.1.7 and are location in the [COLOR %s]/addon\_data/%s/[/COLOR] folder[/COLOR]' % (COLOR2, COLOR1, ADDON\_ID))

def reloadProfile(profile=None):

if profile == None:

#if os.path.exists(PROFILES):

# profile = getInfo('System.ProfileName')

# log("Profile: %s" % profile)

# ebi('LoadProfile(%s)' % profile)

#else:

#ebi('Mastermode')

ebi('LoadProfile(Master user)')

else: ebi('LoadProfile(%s)' % profile)

def chunks(s, n):

for start in range(0, len(s), n):

yield s[start:start+n]

def asciiCheck(use=None, over=False):

if use == None:

source = DIALOG.browse(3, '[COLOR %s]Select the folder you want to scan[/COLOR]' % COLOR2, 'files', '', False, False, HOME)

if over == True:

yes = 1

else:

yes = DIALOG.yesno(ADDONTITLE,'[COLOR %s]Do you want to [COLOR %s]delete[/COLOR] all filenames with special characters or would you rather just [COLOR %s]scan and view[/COLOR] the results in the log?[/COLOR]' % (COLOR2, COLOR1, COLOR1), yeslabel='[B][COLOR green]Delete[/COLOR][/B]', nolabel='[B][COLOR red]Scan[/COLOR][/B]')

else:

source = use

yes = 1

if source == "":

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]ASCII Check: Cancelled[/COLOR]" % COLOR2)

return

files\_found = os.path.join(ADDONDATA, 'asciifiles.txt')

files\_fails = os.path.join(ADDONDATA, 'asciifails.txt')

afiles = open(files\_found, mode='w+')

afails = open(files\_fails, mode='w+')

f1 = 0; f2 = 0

items = fileCount(source)

msg = ''

prog = []

log("Source file: (%s)" % str(source), xbmc.LOGNOTICE)

DP.create(ADDONTITLE, 'Please wait...')

for base, dirs, files in os.walk(source):

dirs[:] = [d for d in dirs]

files[:] = [f for f in files]

for file in files:

prog.append(file)

prog2 = int(len(prog) / float(items) \* 100)

DP.update(prog2,"[COLOR %s]Checking for non ASCII files" % COLOR2,'[COLOR %s]%s[/COLOR]' % (COLOR1, d), 'Please Wait[/COLOR]')

try:

file.encode('ascii')

except UnicodeDecodeError:

badfile = os.path.join(base, file)

if yes:

try:

os.remove(badfile)

for chunk in chunks(badfile, 75):

afiles.write(chunk+'\n')

afiles.write('\n')

f1 += 1

log("[ASCII Check] File Removed: %s " % badfile, xbmc.LOGERROR)

except:

for chunk in chunks(badfile, 75):

afails.write(chunk+'\n')

afails.write('\n')

f2 += 1

log("[ASCII Check] File Failed: %s " % badfile, xbmc.LOGERROR)

else:

for chunk in chunks(badfile, 75):

afiles.write(chunk+'\n')

afiles.write('\n')

f1 += 1

log("[ASCII Check] File Found: %s " % badfile, xbmc.LOGERROR)

pass

if DP.iscanceled():

DP.close()

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Ascii Check Cancelled[/COLOR]" % COLOR2)

sys.exit()

DP.close(); afiles.close(); afails.close()

total = int(f1) + int(f2)

if total > 0:

if os.path.exists(files\_found): afiles = open(files\_found, mode='r'); msg = afiles.read(); afiles.close()

if os.path.exists(files\_fails): afails = open(files\_fails, mode='r'); msg2 = afails.read(); afails.close()

if yes:

if use:

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]ASCII Check: %s Removed / %s Failed.[/COLOR]" % (COLOR2, f1, f2))

else:

TextBox(ADDONTITLE, "[COLOR yellow][B]%s Files Removed:[/B][/COLOR]\n %s\n\n[COLOR yellow][B]%s Files Failed:[B][/COLOR]\n %s" % (f1, msg, f2, msg2))

else:

TextBox(ADDONTITLE, "[COLOR yellow][B]%s Files Found:[/B][/COLOR]\n %s" % (f1, msg))

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]ASCII Check: None Found.[/COLOR]" % COLOR2)

def fileCount(home, excludes=True):

exclude\_dirs = [ADDON\_ID, 'cache', 'system', 'packages', 'Thumbnails', 'peripheral\_data', 'temp', 'My\_Builds', 'library', 'keymaps']

exclude\_files = ['Textures13.db', '.DS\_Store', 'advancedsettings.xml', 'Thumbs.db', '.gitignore']

item = []

for base, dirs, files in os.walk(home):

if excludes:

dirs[:] = [d for d in dirs if d not in exclude\_dirs]

files[:] = [f for f in files if f not in exclude\_files]

for file in files:

item.append(file)

return len(item)

def defaultSkin():

log("[Default Skin Check]", xbmc.LOGNOTICE)

tempgui = os.path.join(USERDATA, 'guitemp.xml')

gui = tempgui if os.path.exists(tempgui) else GUISETTINGS

if not os.path.exists(gui): return False

log("Reading gui file: %s" % gui, xbmc.LOGNOTICE)

guif = open(gui, 'r+')

msg = guif.read().replace('\n','').replace('\r','').replace('\t','').replace(' ',''); guif.close()

log("Opening gui settings", xbmc.LOGNOTICE)

match = re.compile('<lookandfeel>.+?<ski.+?>(.+?)</skin>.+?</lookandfeel>').findall(msg)

log("Matches: %s" % str(match), xbmc.LOGNOTICE)

if len(match) > 0:

skinid = match[0]

addonxml = os.path.join(ADDONS, match[0], 'addon.xml')

if os.path.exists(addonxml):

addf = open(addonxml, 'r+')

msg2 = addf.read(); addf.close()

match2 = parseDOM(msg2, 'addon', ret='name')

if len(match2) > 0: skinname = match2[0]

else: skinname = 'no match'

else: skinname = 'no file'

log("[Default Skin Check] Skin name: %s" % skinname, xbmc.LOGNOTICE)

log("[Default Skin Check] Skin id: %s" % skinid, xbmc.LOGNOTICE)

setS('defaultskin', skinid)

setS('defaultskinname', skinname)

setS('defaultskinignore', 'false')

if os.path.exists(tempgui):

log("Deleting Temp Gui File.", xbmc.LOGNOTICE)

os.remove(tempgui)

log("[Default Skin Check] End", xbmc.LOGNOTICE)

def lookandFeelData(do='save'):

scan = ['lookandfeel.enablerssfeeds', 'lookandfeel.font', 'lookandfeel.rssedit', 'lookandfeel.skincolors', 'lookandfeel.skintheme', 'lookandfeel.skinzoom', 'lookandfeel.soundskin', 'lookandfeel.startupwindow', 'lookandfeel.stereostrength']

if do == 'save':

for item in scan:

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

response = xbmc.executeJSONRPC(query)

if not 'error' in response:

match = re.compile('{"value":(.+?)}').findall(str(response))

setS(item.replace('lookandfeel', 'default'), match[0])

log("%s saved to %s" % (item, match[0]), xbmc.LOGNOTICE)

else:

for item in scan:

value = getS(item.replace('lookandfeel', 'default'))

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

response = xbmc.executeJSONRPC(query)

log("%s restored to %s" % (item, value), xbmc.LOGNOTICE)

def sep(middle=''):

char = uservar.SPACER

ret = char \* 40

if not middle == '':

middle = '[ %s ]' % middle

fluff = int((40 - len(middle))/2)

ret = "%s%s%s" % (ret[:fluff], middle, ret[:fluff+2])

return ret[:40]

def convertAdvanced():

if os.path.exists(ADVANCED):

f = open(ADVANCED)

a = f.read()

if KODIV >= 17:

return

else:

return

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]AdvancedSettings.xml not found[/COLOR]")

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

###BACK UP/RESTORE #######

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

def backUpOptions(type, name=""):

exclude\_dirs = [ADDON\_ID, 'cache', 'system', 'Thumbnails', 'peripheral\_data', 'temp', 'My\_Builds', 'keymaps']

exclude\_files = ['Textures13.db', '.DS\_Store', 'advancedsettings.xml', 'Thumbs.db', '.gitignore']

bad\_files = [os.path.join(DATABASE, 'onechannelcache.db'),

os.path.join(DATABASE, 'saltscache.db'),

os.path.join(DATABASE, 'saltscache.db-shm'),

os.path.join(DATABASE, 'saltscache.db-wal'),

os.path.join(DATABASE, 'saltshd.lite.db'),

os.path.join(DATABASE, 'saltshd.lite.db-shm'),

os.path.join(DATABASE, 'saltshd.lite.db-wal'),

os.path.join(ADDOND, 'script.trakt', 'queue.db'),

os.path.join(HOME, 'cache', 'commoncache.db'),

os.path.join(ADDOND, 'script.module.dudehere.routines', 'access.log'),

os.path.join(ADDOND, 'script.module.dudehere.routines', 'trakt.db'),

os.path.join(ADDOND, 'script.module.metahandler', 'meta\_cache', 'video\_cache.db')]

backup = xbmc.translatePath(BACKUPLOCATION)

mybuilds = xbmc.translatePath(MYBUILDS)

try:

if not os.path.exists(backup): xbmcvfs.mkdirs(backup)

if not os.path.exists(mybuilds): xbmcvfs.mkdirs(mybuilds)

except Exception, e:

DIALOG.ok(ADDONTITLE, "[COLOR %s]Error making Back Up directories:[/COLOR]" % (COLOR2), "[COLOR %s]%s[/COLOR]" % (COLOR1, str(e)))

return

if type == "addon pack":

if DIALOG.yesno(ADDONTITLE, "[COLOR %s]Are you sure you wish to create an Addon Pack?[/COLOR]" % COLOR2, nolabel="[B][COLOR red]Cancel Backup[/COLOR][/B]", yeslabel="[B][COLOR green]Create Pack[/COLOR][/B]"):

if name == "":

name = getKeyboard("","Please enter a name for the %s zip" % type)

if not name: return False

name = urllib.quote\_plus(name)

name = '%s.zip' % name; tempzipname = ''

zipname = os.path.join(mybuilds, name)

try:

zipf = zipfile.ZipFile(xbmc.translatePath(zipname), mode='w')

except:

try:

tempzipname = os.path.join(PACKAGES, '%s.zip' % name)

zipf = zipfile.ZipFile(tempzipname, mode='w')

except:

log("Unable to create %s.zip" % name, xbmc.LOGERROR)

if DIALOG.yesno(ADDONTITLE, "[COLOR %s]We are unable to write to the current backup directory, would you like to change the location?[/COLOR]" % COLOR2, yeslabel="[B][COLOR green]Change Directory[/COLOR][/B]", nolabel="[B][COLOR red]Cancel[/COLOR][/B]"):

openS()

return

else:

return

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

addonnames = []; addonfolds = []

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

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

if foldername in EXCLUDES: continue

elif foldername in DEFAULTPLUGINS: continue

elif foldername == 'packages': continue

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

if os.path.exists(xml):

f = open(xml)

a = f.read()

match = parseDOM(a, 'addon', ret='name')

if len(match) > 0:

addonnames.append(match[0])

addonfolds.append(foldername)

else:

addonnames.append(foldername)

addonfolds.append(foldername)

if KODIV > 16:

selected = DIALOG.multiselect("%s: Select the addons you wish to add to the zip." % ADDONTITLE, addonnames)

if selected == None: selected = []

else:

selected = []; choice = 0

tempaddonnames = ["-- Click here to Continue --"] + addonnames

while not choice == -1:

choice = DIALOG.select("%s: Select the addons you wish to add to the zip." % ADDONTITLE, tempaddonnames)

if choice == -1: break

elif choice == 0: break

else:

choice2 = (choice-1)

if choice2 in selected:

selected.remove(choice2)

tempaddonnames[choice] = addonnames[choice2]

else:

selected.append(choice2)

tempaddonnames[choice] = "[B][COLOR %s]%s[/COLOR][/B]" % (COLOR1, addonnames[choice2])

log(selected)

DP.create(ADDONTITLE,'[COLOR %s][B]Creating Zip File:[/B][/COLOR]' % COLOR2,'', 'Please Wait')

if len(selected) > 0:

added = []

for item in selected:

added.append(addonfolds[item])

DP.update(0, "", "[COLOR %s]%s[/COLOR]" % (COLOR1, addonfolds[item]))

for base, dirs, files in os.walk(os.path.join(ADDONS,addonfolds[item])):

files[:] = [f for f in files if f not in exclude\_files]

for file in files:

if file.endswith('.pyo'): continue

DP.update(0, "", "[COLOR %s]%s[/COLOR]" % (COLOR1, addonfolds[item]), "[COLOR %s]%s[/COLOR]" % (COLOR1, file))

fn = os.path.join(base, file)

zipf.write(fn, fn[len(ADDONS):], zipfile.ZIP\_DEFLATED)

dep=os.path.join(ADDONS,addonfolds[item],'addon.xml')

if os.path.exists(dep):

source = open(dep,mode='r'); link = source.read(); source.close();

match = parseDOM(link, 'import', ret='addon')

for depends in match:

if 'xbmc.python' in depends: continue

if depends in added: continue

DP.update(0, "", "[COLOR %s]%s[/COLOR]" % (COLOR1, depends))

for base, dirs, files in os.walk(os.path.join(ADDONS,depends)):

files[:] = [f for f in files if f not in exclude\_files]

for file in files:

if file.endswith('.pyo'): continue

DP.update(0, "", "[COLOR %s]%s[/COLOR]" % (COLOR1, depends), "[COLOR %s]%s[/COLOR]" % (COLOR1, file))

fn = os.path.join(base, file)

zipf.write(fn, fn[len(ADDONS):], zipfile.ZIP\_DEFLATED)

added.append(depends)

DIALOG.ok(ADDONTITLE, "[COLOR %s]%s[/COLOR] [COLOR %s]backup successful:[/COLOR]" % (COLOR1, name, COLOR2), "[COLOR %s]%s[/COLOR]" % (COLOR1, zipname))

elif type == "build":

if DIALOG.yesno(ADDONTITLE, "[COLOR %s]Are you sure you wish to backup the current build?[/COLOR]" % COLOR2, nolabel="[B][COLOR red]Cancel Backup[/COLOR][/B]", yeslabel="[B][COLOR green]Backup Build[/COLOR][/B]"):

if name == "":

name = getKeyboard("","Please enter a name for the %s zip" % type)

if not name: return False

name = name.replace('\\', '').replace('/', '').replace(':', '').replace('\*', '').replace('?', '').replace('"', '').replace('<', '').replace('>', '').replace('|', '')

name = urllib.quote\_plus(name); tempzipname = ''

zipname = os.path.join(mybuilds, '%s.zip' % name)

for\_progress = 0

ITEM = []

if not DIALOG.yesno(ADDONTITLE, "[COLOR %s]Do you want to include your addon\_data folder?" % COLOR2, 'This contains [COLOR %s]ALL[/COLOR] addon settings including passwords but may also contain important information such as skin shortcuts. We recommend [COLOR %s]MANUALLY[/COLOR] removing the addon\_data folders that aren\'t required.' % (COLOR1, COLOR1), '[COLOR %s]%s[/COLOR] addon\_data is ignored[/COLOR]' % (COLOR1, ADDON\_ID), yeslabel='[B][COLOR green]Include data[/COLOR][/B]',nolabel='[B][COLOR red]Don\'t Include[/COLOR][/B]'):

exclude\_dirs.append('addon\_data')

convertSpecial(HOME, True)

asciiCheck(HOME, True)

extractsize = 0

try:

zipf = zipfile.ZipFile(xbmc.translatePath(zipname), mode='w')

except:

try:

tempzipname = os.path.join(PACKAGES, '%s.zip' % name)

zipf = zipfile.ZipFile(tempzipname, mode='w')

except:

log("Unable to create %s.zip" % name, xbmc.LOGERROR)

if DIALOG.yesno(ADDONTITLE, "[COLOR %s]We are unable to write to the current backup directory, would you like to change the location?[/COLOR]" % COLOR2, yeslabel="[B][COLOR green]Change Directory[/COLOR][/B]", nolabel="[B][COLOR red]Cancel[/COLOR][/B]"):

openS()

return

else:

return

DP.create("[COLOR %s]%s[/COLOR][COLOR %s]: Creating Zip[/COLOR]" % (COLOR1, ADDONTITLE,COLOR2), "[COLOR %s]Creating back up zip" % COLOR2, "", "Please Wait...[/COLOR]")

for base, dirs, files in os.walk(HOME):

dirs[:] = [d for d in dirs if d not in exclude\_dirs]

files[:] = [f for f in files if f not in exclude\_files]

for file in files:

ITEM.append(file)

N\_ITEM = len(ITEM)

picture = []; music = []; video = []; programs = []; repos = []; scripts = []; skins = []

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

idlist = []

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

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

if foldername == 'packages': continue

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

if os.path.exists(xml):

f = open(xml)

a = f.read()

prov = re.compile("<provides>(.+?)</provides>").findall(a)

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

addid = foldername if len(match) == 0 else match[0]

if addid in idlist:

continue

idlist.append(addid)

try:

add = xbmcaddon.Addon(id=addid)

aname = add.getAddonInfo('name')

aname = aname.replace('[', '<').replace(']', '>')

aname = str(re.sub('<[^<]+?>', '', aname)).lstrip()

except:

aname = foldername

if len(prov) == 0:

if foldername.startswith('skin'): skins.append(aname)

elif foldername.startswith('repo'): repos.append(aname)

else: scripts.append(aname)

continue

if not (prov[0]).find('executable') == -1: programs.append(aname)

if not (prov[0]).find('video') == -1: video.append(aname)

if not (prov[0]).find('audio') == -1: music.append(aname)

if not (prov[0]).find('image') == -1: picture.append(aname)

fixmetas()

for base, dirs, files in os.walk(HOME):

dirs[:] = [d for d in dirs if d not in exclude\_dirs]

files[:] = [f for f in files if f not in exclude\_files]

for file in files:

try:

for\_progress += 1

progress = percentage(for\_progress, N\_ITEM)

DP.update(int(progress), '[COLOR %s]Creating back up zip: [COLOR%s]%s[/COLOR] / [COLOR%s]%s[/COLOR]' % (COLOR2, COLOR1, for\_progress, COLOR1, N\_ITEM), '[COLOR %s]%s[/COLOR]' % (COLOR1, file), '')

fn = os.path.join(base, file)

if file in LOGFILES: log("[Back Up] Type = '%s': Ignore %s" % (type, file), xbmc.LOGNOTICE); continue

elif os.path.join(base, file) in bad\_files: log("[Back Up] Type = '%s': Ignore %s" % (type, file), xbmc.LOGNOTICE); continue

elif os.path.join('addons', 'packages') in fn: log("[Back Up] Type = '%s': Ignore %s" % (type, file), xbmc.LOGNOTICE); continue

elif file.endswith('.csv'): log("[Back Up] Type = '%s': Ignore %s" % (type, file), xbmc.LOGNOTICE); continue

elif file.endswith('.pyo'): continue

elif file.endswith('.db') and 'Database' in base:

temp = file.replace('.db', '')

temp = ''.join([i for i in temp if not i.isdigit()])

if temp in ['Addons', 'ADSP', 'Epg', 'MyMusic', 'MyVideos', 'Textures', 'TV', 'ViewModes']:

if not file == latestDB(temp): log("[Back Up] Type = '%s': Ignore %s" % (type, file), xbmc.LOGNOTICE); continue

try:

zipf.write(fn, fn[len(HOME):], zipfile.ZIP\_DEFLATED)

extractsize += os.path.getsize(fn)

except Exception, e:

log("[Back Up] Type = '%s': Unable to backup %s" % (type, file), xbmc.LOGNOTICE)

log("%s / %s" % (Exception, e))

if DP.iscanceled():

DP.close()

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Backup Cancelled[/COLOR]" % COLOR2)

sys.exit()

except Exception, e:

log("[Back Up] Type = '%s': Unable to backup %s" % (type, file), xbmc.LOGNOTICE)

log("Build Backup Error: %s" % str(e), xbmc.LOGNOTICE)

if 'addon\_data' in exclude\_dirs:

match = glob.glob(os.path.join(ADDOND,'skin.\*', ''))

for fold in match:

fd = os.path.split(fold[:-1])[1]

if not fd in ['skin.confluence', 'skin.re-touch', 'skin.estuary', 'skin.estouchy']:

for base, dirs, files in os.walk(os.path.join(ADDOND,fold)):

files[:] = [f for f in files if f not in exclude\_files]

for file in files:

fn = os.path.join(base, file)

zipf.write(fn, fn[len(HOME):], zipfile.ZIP\_DEFLATED)

extractsize += os.path.getsize(fn)

xml = os.path.join(ADDONS, fd, 'addon.xml')

if os.path.exists(xml):

source = open(xml,mode='r'); link = source.read(); source.close();

matchxml = parseDOM(link, 'import', ret='addon')

if 'script.skinshortcuts' in matchxml:

for base, dirs, files in os.walk(os.path.join(ADDOND,'script.skinshortcuts')):

files[:] = [f for f in files if f not in exclude\_files]

for file in files:

fn = os.path.join(base, file)

zipf.write(fn, fn[len(HOME):], zipfile.ZIP\_DEFLATED)

extractsize += os.path.getsize(fn)

zipf.close()

xbmc.sleep(500)

DP.close()

backUpOptions('guifix', name)

if not tempzipname == '':

success = xbmcvfs.rename(tempzipname, zipname)

if success == 0:

xbmcvfs.copy(tempzipname, zipname)

xbmcvfs.delete(tempzipname)

info = zipname.replace('.zip', '.txt')

f = open(info, 'w'); f.close()

with open(info, 'a') as f:

f.write('name="%s"\n' % name)

f.write('extracted="%s"\n' % extractsize)

f.write('zipsize="%s"\n' % os.path.getsize(xbmc.translatePath(zipname)))

f.write('skin="%s"\n' % currSkin())

f.write('created="%s"\n' % datetime.now().date())

f.write('programs="%s"\n' % ', '.join(programs) if len(programs) > 0 else 'programs="none"\n')

f.write('video="%s"\n' % ', '.join(video) if len(video) > 0 else 'video="none"\n')

f.write('music="%s"\n' % ', '.join(music) if len(music) > 0 else 'music="none"\n')

f.write('picture="%s"\n' % ', '.join(picture) if len(picture) > 0 else 'picture="none"\n')

f.write('repos="%s"\n' % ', '.join(repos) if len(repos) > 0 else 'repos="none"\n')

f.write('scripts="%s"\n' % ', '.join(scripts) if len(scripts) > 0 else 'scripts="none"\n')

DIALOG.ok(ADDONTITLE, "[COLOR %s]%s[/COLOR] [COLOR %s]backup successful:[/COLOR]" % (COLOR1, name, COLOR2), "[COLOR %s]%s[/COLOR]" % (COLOR1, zipname))

elif type == "guifix":

if name == "":

guiname = getKeyboard("","Please enter a name for the %s zip" % type)

if not guiname: return False

convertSpecial(USERDATA, True)

asciiCheck(USERDATA, True)

else: guiname = name

guiname = urllib.quote\_plus(guiname); tempguizipname = ''

guizipname = xbmc.translatePath(os.path.join(mybuilds, '%s\_guisettings.zip' % guiname))

if os.path.exists(GUISETTINGS):

try:

zipf = zipfile.ZipFile(guizipname, mode='w')

except:

try:

tempguizipname = os.path.join(PACKAGES, '%s\_guisettings.zip' % guiname)

zipf = zipfile.ZipFile(tempguizipname, mode='w')

except:

log("Unable to create %s\_guisettings.zip" % guiname, xbmc.LOGERROR)

if DIALOG.yesno(ADDONTITLE, "[COLOR %s]We are unable to write to the current backup directory, would you like to change the location?[/COLOR]" % COLOR2, yeslabel="[B][COLOR green]Change Directory[/COLOR][/B]", nolabel="[B][COLOR red]Cancel[/COLOR][/B]"):

openS()

return

else:

return

try:

zipf.write(GUISETTINGS, 'guisettings.xml', zipfile.ZIP\_DEFLATED)

zipf.write(PROFILES, 'profiles.xml', zipfile.ZIP\_DEFLATED)

match = glob.glob(os.path.join(ADDOND,'skin.\*', ''))

for fold in match:

fd = os.path.split(fold[:-1])[1]

if not fd in ['skin.confluence', 'skin.re-touch', 'skin.estuary', 'skin.estouchy']:

if DIALOG.yesno(ADDONTITLE, "[COLOR %s]Would you like to add the following skin folder to the GuiFix Zip File?[/COLOR]" % COLOR2, "[COLOR %s]%s[/COLOR]" % (COLOR1, fd), yeslabel="[B][COLOR green]Add Skin[/COLOR][/B]", nolabel="[B][COLOR red]Skip Skin[/COLOR][/B]"):

for base, dirs, files in os.walk(os.path.join(ADDOND,fold)):

files[:] = [f for f in files if f not in exclude\_files]

for file in files:

fn = os.path.join(base, file)

zipf.write(fn, fn[len(USERDATA):], zipfile.ZIP\_DEFLATED)

xml = os.path.join(ADDONS, fd, 'addon.xml')

if os.path.exists(xml):

source = open(xml,mode='r'); link = source.read(); source.close();

matchxml = parseDOM(link, 'import', ret='addon')

if 'script.skinshortcuts' in matchxml:

for base, dirs, files in os.walk(os.path.join(ADDOND,'script.skinshortcuts')):

files[:] = [f for f in files if f not in exclude\_files]

for file in files:

fn = os.path.join(base, file)

zipf.write(fn, fn[len(USERDATA):], zipfile.ZIP\_DEFLATED)

else: log("[Back Up] Type = '%s': %s ignored" % (type, fold), xbmc.LOGNOTICE)

except Exception, e:

log("[Back Up] Type = '%s': %s" % (type, e), xbmc.LOGNOTICE)

pass

zipf.close()

if not tempguizipname == '':

success = xbmcvfs.rename(tempguizipname, guizipname)

if success == 0:

xbmcvfs.copy(tempguizipname, guizipname)

xbmcvfs.delete(tempguizipname)

else: log("[Back Up] Type = '%s': guisettings.xml not found" % type, xbmc.LOGNOTICE)

if name == "":

DIALOG.ok(ADDONTITLE, "[COLOR %s]GuiFix backup successful:[/COLOR]" % (COLOR2), "[COLOR %s]%s[/COLOR]" % (COLOR1, guizipname))

elif type == "theme":

if not DIALOG.yesno('[COLOR %s]%s[/COLOR][COLOR %s]: Theme Backup[/COLOR]' % (COLOR1, ADDONTITLE, COLOR2), "[COLOR %s]Would you like to create a theme backup?[/COLOR]" % COLOR2, yeslabel="[B][COLOR green]Continue[/COLOR][/B]", nolabel="[B][COLOR red]No Cancel[/COLOR][/B]"): LogNotify("Theme Backup", "Cancelled!"); return False

if name == "":

themename = getKeyboard("","Please enter a name for the %s zip" % type)

if not themename: return False

else: themename = name

themename = urllib.quote\_plus(themename); tempzipname = ''

zipname = os.path.join(mybuilds, '%s.zip' % themename)

try:

zipf = zipfile.ZipFile(xbmc.translatePath(zipname), mode='w')

except:

try:

tempzipname = os.path.join(PACKAGES, '%s.zip' % themename)

zipf = zipfile.ZipFile(tempzipname, mode='w')

except:

log("Unable to create %s.zip" % themename, xbmc.LOGERROR)

if DIALOG.yesno(ADDONTITLE, "[COLOR %s]We are unable to write to the current backup directory, would you like to change the location?[/COLOR]" % COLOR2, yeslabel="[B][COLOR green]Change Directory[/COLOR][/B]", nolabel="[B][COLOR red]Cancel[/COLOR][/B]"):

openS()

return

else:

return

convertSpecial(USERDATA, True)

asciiCheck(USERDATA, True)

try:

if not SKIN == 'skin.confluence':

skinfold = os.path.join(ADDONS, SKIN, 'media')

match2 = glob.glob(os.path.join(skinfold,'\*.xbt'))

if len(match2) > 1:

if DIALOG.yesno('[COLOR %s]%s[/COLOR][COLOR %s]: Theme Backup[/COLOR]' % (COLOR1, ADDONTITLE, COLOR2), "[COLOR %s]Would you like to go through the Texture Files for?[/COLOR]" % COLOR2, "[COLOR %s]%s[/COLOR]" % (COLOR1, SKIN), yeslabel="[B][COLOR green]Add Textures[/COLOR][/B]", nolabel="[B][COLOR red]Skip Textures[/COLOR][/B]"):

skinfold = os.path.join(ADDONS, SKIN, 'media')

match2 = glob.glob(os.path.join(skinfold,'\*.xbt'))

for xbt in match2:

if DIALOG.yesno('[COLOR %s]%s[/COLOR][COLOR %s]: Theme Backup[/COLOR]' % (COLOR1, ADDONTITLE, COLOR2), "[COLOR %s]Would you like to add the Texture File [COLOR %s]%s[/COLOR]?" % (COLOR1, COLOR2, xbt.replace(skinfold, "")[1:]), "from [COLOR %s]%s[/COLOR][/COLOR]" % (COLOR1, SKIN), yeslabel="[B][COLOR green]Add Textures[/COLOR][/B]", nolabel="[B][COLOR red]Skip Textures[/COLOR][/B]"):

fn = xbt

fn2 = fn.replace(HOME, "")

zipf.write(fn, fn2, zipfile.ZIP\_DEFLATED)

else:

for xbt in match2:

if DIALOG.yesno('[COLOR %s]%s[/COLOR][COLOR %s]: Theme Backup[/COLOR]' % (COLOR1, ADDONTITLE, COLOR2), "[COLOR %s]Would you like to add the Texture File [COLOR %s]%s[/COLOR]?" % (COLOR2, COLOR1, xbt.replace(skinfold, "")[1:]), "from [COLOR %s]%s[/COLOR][/COLOR]" % (COLOR1, SKIN), yeslabel="[B][COLOR green]Add Textures[/COLOR][/B]", nolabel="[B][COLOR red]Skip Textures[/COLOR][/B]"):

fn = xbt

fn2 = fn.replace(HOME, "")

zipf.write(fn, fn2, zipfile.ZIP\_DEFLATED)

ad\_skin = os.path.join(ADDOND, SKIN, 'settings.xml')

if os.path.exists(ad\_skin):

if DIALOG.yesno('[COLOR %s]%s[/COLOR][COLOR %s]: Theme Backup[/COLOR]' % (COLOR1, ADDONTITLE, COLOR2), "[COLOR %s]Would you like to go add the [COLOR %s]settings.xml[/COLOR] in [COLOR %s]/addon\_data/[/COLOR] for?" % (COLOR2, COLOR1, COLOR1), "[COLOR %s]%s[/COLOR]" % (COLOR1, SKIN), yeslabel="[B][COLOR green]Add Settings[/COLOR][/B]", nolabel="[B][COLOR red]Skip Settings[/COLOR][/B]"):

skinfold = os.path.join(ADDOND, SKIN)

zipf.write(ad\_skin, ad\_skin.replace(HOME, ""), zipfile.ZIP\_DEFLATED)

f = open(os.path.join(ADDONS, SKIN, 'addon.xml')); r = f.read(); f.close()

match = parseDOM(r, 'import', ret='addon')

if 'script.skinshortcuts' in match:

if DIALOG.yesno('[COLOR %s]%s[/COLOR][COLOR %s]: Theme Backup[/COLOR]' % (COLOR1, ADDONTITLE, COLOR2), "[COLOR %s]Would you like to go add the [COLOR %s]settings.xml[/COLOR] for [COLOR %s]script.skinshortcuts[/COLOR]?" % (COLOR2, COLOR1, COLOR1), yeslabel="[B][COLOR green]Add Settings[/COLOR][/B]", nolabel="[B][COLOR red]Skip Settings[/COLOR][/B]"):

for base, dirs, files in os.walk(os.path.join(ADDOND,'script.skinshortcuts')):

files[:] = [f for f in files if f not in exclude\_files]

for file in files:

fn = os.path.join(base, file)

zipf.write(fn, fn[len(HOME):], zipfile.ZIP\_DEFLATED)

if DIALOG.yesno('[COLOR %s]%s[/COLOR][COLOR %s]: Theme Backup[/COLOR]' % (COLOR1, ADDONTITLE, COLOR2), "[COLOR %s]Would you like to include a [COLOR %s]Backgrounds[/COLOR] folder?[/COLOR]" % (COLOR2, COLOR1), yeslabel="[B][COLOR green]Yes Include[/COLOR][/B]", nolabel="[B][COLOR red]No Continue[/COLOR][/B]"):

fn = DIALOG.browse(0, 'Select location of backgrounds', 'files', '', True, False, HOME, False)

if not fn == HOME:

for base, dirs, files in os.walk(fn):

dirs[:] = [d for d in dirs if d not in exclude\_dirs]

files[:] = [f for f in files if f not in exclude\_files]

for file in files:

try:

fn2 = os.path.join(base, file)

zipf.write(fn2, fn2[len(HOME):], zipfile.ZIP\_DEFLATED)

except Exception, e:

log("[Back Up] Type = '%s': Unable to backup %s" % (type, file), xbmc.LOGNOTICE)

log("Backup Error: %s" % str(e), xbmc.LOGNOTICE)

text = latestDB('Textures')

if DIALOG.yesno('[COLOR %s]%s[/COLOR][COLOR %s]: Theme Backup[/COLOR]' % (COLOR1, ADDONTITLE, COLOR2), "[COLOR %s]Would you like to include the [COLOR %s]%s[/COLOR]?[/COLOR]" % (COLOR2, COLOR1, text), yeslabel="[B][COLOR green]Yes Include[/COLOR][/B]", nolabel="[B][COLOR red]No Continue[/COLOR][/B]"):

zipf.write(os.path.join(DATABASE, text), '/userdata/Database/%s' % text, zipfile.ZIP\_DEFLATED)

if DIALOG.yesno('[COLOR %s]%s[/COLOR][COLOR %s]: Theme Backup[/COLOR]' % (COLOR1, ADDONTITLE, COLOR2), "[COLOR %s]Would you like to include any addons?[/COLOR]" % (COLOR2), yeslabel="[B][COLOR green]Yes Include[/COLOR][/B]", nolabel="[B][COLOR red]No Continue[/COLOR][/B]"):

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

addonnames = []; addonfolds = []

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

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

if foldername in EXCLUDES: continue

elif foldername in DEFAULTPLUGINS: continue

elif foldername == 'packages': continue

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

if os.path.exists(xml):

f = open(xml)

a = f.read()

match = parseDOM(a, 'addon', ret='name')

if len(match) > 0:

addonnames.append(match[0])

addonfolds.append(foldername)

else:

addonnames.append(foldername)

addonfolds.append(foldername)

if KODIV > 16:

selected = DIALOG.multiselect("%s: Select the addons you wish to add to the zip." % ADDONTITLE, addonnames)

if selected == None: selected = []

else:

selected = []; choice = 0

tempaddonnames = ["-- Click here to Continue --"] + addonnames

while not choice == -1:

choice = DIALOG.select("%s: Select the addons you wish to add to the zip." % ADDONTITLE, tempaddonnames)

if choice == -1: break

elif choice == 0: break

else:

choice2 = (choice-1)

if choice2 in selected:

selected.remove(choice2)

tempaddonnames[choice] = addonnames[choice2]

else:

selected.append(choice2)

tempaddonnames[choice] = "[B][COLOR %s]%s[/COLOR][/B]" % (COLOR1, addonnames[choice2])

if len(selected) > 0:

added = []

for item in selected:

added.append(addonfolds[item])

for base, dirs, files in os.walk(os.path.join(ADDONS,addonfolds[item])):

files[:] = [f for f in files if f not in exclude\_files]

for file in files:

if file.endswith('.pyo'): continue

fn = os.path.join(base, file)

zipf.write(fn, fn[len(HOME):], zipfile.ZIP\_DEFLATED)

dep=os.path.join(ADDONS,addonfolds[item],'addon.xml')

if os.path.exists(dep):

source = open(dep,mode='r'); link = source.read(); source.close();

match = parseDOM(link, 'import', ret='addon')

for depends in match:

if 'xbmc.python' in depends: continue

if depends in added: continue

for base, dirs, files in os.walk(os.path.join(ADDONS,depends)):

files[:] = [f for f in files if f not in exclude\_files]

for file in files:

if file.endswith('.pyo'): continue

fn = os.path.join(base, file)

zipf.write(fn, fn[len(HOME):], zipfile.ZIP\_DEFLATED)

added.append(depends)

if DIALOG.yesno('[COLOR %s]%s[/COLOR][COLOR %s]: Theme Backup[/COLOR]' % (COLOR1, ADDONTITLE, COLOR2), "[COLOR %s]Would you like to include the [COLOR %s]guisettings.xml[/COLOR]?[/COLOR]" % (COLOR2, COLOR1), yeslabel="[B][COLOR green]Yes Include[/COLOR][/B]", nolabel="[B][COLOR red]No Continue[/COLOR][/B]"):

zipf.write(GUISETTINGS, '/userdata/guisettings.xml', zipfile.ZIP\_DEFLATED)

except Exception, e:

zipf.close()

log("[Back Up] Type = '%s': %s" % (type, str(e)), xbmc.LOGNOTICE)

DIALOG.ok(ADDONTITLE, "[COLOR %s]%s[/COLOR][COLOR %s] theme zip failed:[/COLOR]" % (COLOR1, themename, COLOR2), "[COLOR %s]%s[/COLOR]" % (COLOR1, str(e)))

if not tempzipname == '':

try: os.remove(xbmc.translatePath(tempzipname))

except Exception, e: log(str(e))

else:

try: os.remove(xbmc.translatePath(zipname))

except Exception, e: log(str(e))

return

zipf.close()

if not tempzipname == '':

success = xbmcvfs.rename(tempzipname, zipname)

if success == 0:

xbmcvfs.copy(tempzipname, zipname)

xbmcvfs.delete(tempzipname)

DIALOG.ok(ADDONTITLE, "[COLOR %s]%s[/COLOR][COLOR %s] theme zip successful:[/COLOR]" % (COLOR1, themename, COLOR2), "[COLOR %s]%s[/COLOR]" % (COLOR1, zipname))

elif type == "addondata":

if DIALOG.yesno(ADDONTITLE, "[COLOR %s]Are you sure you wish to backup the current addon\_data?[/COLOR]" % COLOR2, nolabel="[B][COLOR red]Cancel Backup[/COLOR][/B]", yeslabel="[B][COLOR green]Backup Addon\_Data[/COLOR][/B]"):

if name == "":

name = getKeyboard("","Please enter a name for the %s zip" % type)

if not name: return False

name = urllib.quote\_plus(name)

name = '%s\_addondata.zip' % name; tempzipname = ''

zipname = os.path.join(mybuilds, name)

try:

zipf = zipfile.ZipFile(xbmc.translatePath(zipname), mode='w')

except:

try:

tempzipname = os.path.join(PACKAGES, '%s.zip' % name)

zipf = zipfile.ZipFile(tempzipname, mode='w')

except:

log("Unable to create %s\_addondata.zip" % name, xbmc.LOGERROR)

if DIALOG.yesno(ADDONTITLE, "[COLOR %s]We are unable to write to the current backup directory, would you like to change the location?[/COLOR]" % COLOR2, yeslabel="[B][COLOR green]Change Directory[/COLOR][/B]", nolabel="[B][COLOR red]Cancel[/COLOR][/B]"):

openS()

return

else:

return

for\_progress = 0

ITEM = []

convertSpecial(ADDOND, True)

asciiCheck(ADDOND, True)

DP.create("[COLOR %s]%s[/COLOR][COLOR %s]: Creating Zip[/COLOR]" % (COLOR1, ADDONTITLE,COLOR2), "[COLOR %s]Creating back up zip" % COLOR2, "", "Please Wait...[/COLOR]")

for base, dirs, files in os.walk(ADDOND):

dirs[:] = [d for d in dirs if d not in exclude\_dirs]

files[:] = [f for f in files if f not in exclude\_files]

for file in files:

ITEM.append(file)

N\_ITEM = len(ITEM)

for base, dirs, files in os.walk(ADDOND):

dirs[:] = [d for d in dirs if d not in exclude\_dirs]

files[:] = [f for f in files if f not in exclude\_files]

for file in files:

try:

for\_progress += 1

progress = percentage(for\_progress, N\_ITEM)

DP.update(int(progress), '[COLOR %s]Creating back up zip: [COLOR%s]%s[/COLOR] / [COLOR%s]%s[/COLOR]' % (COLOR2, COLOR1, for\_progress, COLOR1, N\_ITEM), '[COLOR %s]%s[/COLOR]' % (COLOR1, file), '')

fn = os.path.join(base, file)

if file in LOGFILES: log("[Back Up] Type = '%s': Ignore %s" % (type, file), xbmc.LOGNOTICE); continue

elif os.path.join(base, file) in bad\_files: log("[Back Up] Type = '%s': Ignore %s" % (type, file), xbmc.LOGNOTICE); continue

elif os.path.join('addons', 'packages') in fn: log("[Back Up] Type = '%s': Ignore %s" % (type, file), xbmc.LOGNOTICE); continue

elif file.endswith('.csv'): log("[Back Up] Type = '%s': Ignore %s" % (type, file), xbmc.LOGNOTICE); continue

elif file.endswith('.db') and 'Database' in base:

temp = file.replace('.db', '')

temp = ''.join([i for i in temp if not i.isdigit()])

if temp in ['Addons', 'ADSP', 'Epg', 'MyMusic', 'MyVideos', 'Textures', 'TV', 'ViewModes']:

if not file == latestDB(temp): log("[Back Up] Type = '%s': Ignore %s" % (type, file), xbmc.LOGNOTICE); continue

try:

zipf.write(fn, fn[len(ADDOND):], zipfile.ZIP\_DEFLATED)

except Exception, e:

log("[Back Up] Type = '%s': Unable to backup %s" % (type, file), xbmc.LOGNOTICE)

log("Backup Error: %s" % str(e), xbmc.LOGNOTICE)

except Exception, e:

log("[Back Up] Type = '%s': Unable to backup %s" % (type, file), xbmc.LOGNOTICE)

log("Backup Error: %s" % str(e), xbmc.LOGNOTICE)

zipf.close()

if not tempzipname == '':

success = xbmcvfs.rename(tempzipname, zipname)

if success == 0:

xbmcvfs.copy(tempzipname, zipname)

xbmcvfs.delete(tempzipname)

DP.close()

DIALOG.ok(ADDONTITLE, "[COLOR %s]%s[/COLOR] [COLOR %s]backup successful:[/COLOR]" % (COLOR1, name, COLOR2), "[COLOR %s]%s[/COLOR]" % (COLOR1, zipname))

def restoreLocal(type):

backup = xbmc.translatePath(BACKUPLOCATION)

mybuilds = xbmc.translatePath(MYBUILDS)

try:

if not os.path.exists(backup): xbmcvfs.mkdirs(backup)

if not os.path.exists(mybuilds): xbmcvfs.mkdirs(mybuilds)

except Exception, e:

DIALOG.ok(ADDONTITLE, "[COLOR %s]Error making Back Up directories:[/COLOR]" % (COLOR2), "[COLOR %s]%s[/COLOR]" % (COLOR1, str(e)))

return

file = DIALOG.browse(1, '[COLOR %s]Select the backup file you want to restore[/COLOR]' % COLOR2, 'files', '.zip', False, False, mybuilds)

log("[RESTORE BACKUP %s] File: %s " % (type.upper(), file), xbmc.LOGNOTICE)

if file == "" or not file.endswith('.zip'):

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Local Restore: Cancelled[/COLOR]" % COLOR2)

return

DP.create(ADDONTITLE,'[COLOR %s]Installing Local Backup' % COLOR2,'', 'Please Wait[/COLOR]')

if not os.path.exists(USERDATA): os.makedirs(USERDATA)

if not os.path.exists(ADDOND): os.makedirs(ADDOND)

if not os.path.exists(PACKAGES): os.makedirs(PACKAGES)

if type == "gui": loc = USERDATA

elif type == "addondata":

loc = ADDOND

else : loc = HOME

log("Restoring to %s" % loc, xbmc.LOGNOTICE)

display = os.path.split(file)

fn = display[1]

try:

zipfile.ZipFile(file, 'r')

except:

DP.update(0, '[COLOR %s]Unable to read zipfile from current location.' % COLOR2, 'Copying file to packages')

pack = os.path.join('special://home', 'addons', 'packages', fn)

xbmcvfs.copy(file, pack)

file = xbmc.translatePath(pack)

DP.update(0, '', 'Copying file to packages: Complete')

zipfile.ZipFile(file, 'r')

percent, errors, error = extract.all(file,loc,DP)

fixmetas()

clearS('build')

DP.close()

defaultSkin()

lookandFeelData('save')

if not file.find('packages') == -1:

try: os.remove(file)

except: pass

if int(errors) >= 1:

yes=DIALOG.yesno(ADDONTITLE, '[COLOR %s][COLOR %s]%s[/COLOR]' % (COLOR2, COLOR1, fn), 'Completed: [COLOR %s]%s%s[/COLOR] [Errors:[COLOR %s]%s[/COLOR]]' % (COLOR1, percent, '%', COLOR1, errors), 'Would you like to view the errors?[/COLOR]', nolabel='[B][COLOR red]No Thanks[/COLOR][/B]',yeslabel='[B][COLOR green]View Errors[/COLOR][/B]')

if yes:

if isinstance(errors, unicode):

error = error.encode('utf-8')

TextBox(ADDONTITLE, error.replace('\t',''))

setS('installed', 'true')

setS('extract', str(percent))

setS('errors', str(errors))

if INSTALLMETHOD == 1: todo = 1

elif INSTALLMETHOD == 2: todo = 0

else: todo = DIALOG.yesno(ADDONTITLE, "[COLOR %s]Would you like to [COLOR %s]Force close[/COLOR] kodi or [COLOR %s]Reload Profile[/COLOR]?[/COLOR]" % (COLOR2, COLOR1, COLOR1), yeslabel="[B][COLOR red]Reload Profile[/COLOR][/B]", nolabel="[B][COLOR green]Force Close[/COLOR][/B]")

if todo == 1: reloadFix()

else: killxbmc(True)

def restoreExternal(type):

source = DIALOG.browse(1, '[COLOR %s]Select the backup file you want to restore[/COLOR]' % COLOR2, 'files', '.zip', False, False)

if source == "" or not source.endswith('.zip'):

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]External Restore: Cancelled[/COLOR]" % COLOR2)

return

if not source.startswith('http'):

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]External Restore: Invalid URL[/COLOR]" % COLOR2)

return

try:

work = workingURL(source)

except:

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]External Restore: Error Valid URL[/COLOR]" % COLOR2)

log("Not a working url, if source was local then use local restore option", xbmc.LOGNOTICE)

log("External Source: %s" % source, xbmc.LOGNOTICE)

return

log("[RESTORE EXT BACKUP %s] File: %s " % (type.upper(), source), xbmc.LOGNOTICE)

zipit = os.path.split(source); zname = zipit[1]

DP.create(ADDONTITLE,'[COLOR %s]Downloading Zip file' % COLOR2,'', 'Please Wait[/COLOR]')

if type == "gui": loc = USERDATA

elif type == "addondata": loc = ADDOND

else : loc = HOME

if not os.path.exists(USERDATA): os.makedirs(USERDATA)

if not os.path.exists(ADDOND): os.makedirs(ADDOND)

if not os.path.exists(PACKAGES): os.makedirs(PACKAGES)

file = os.path.join(PACKAGES, zname)

downloader.download(source, file, DP)

DP.update(0,'Installing External Backup','', 'Please Wait')

percent, errors, error = extract.all(file,loc,DP)

fixmetas()

clearS('build')

DP.close()

defaultSkin()

lookandFeelData('save')

if int(errors) >= 1:

yes=DIALOG.yesno(ADDONTITLE, '[COLOR %s][COLOR %s]%s[/COLOR]' % (COLOR2, COLOR1, zname), 'Completed: [COLOR %s]%s%s[/COLOR] [Errors:[COLOR %s]%s[/COLOR]]' % (COLOR1, percent, '%', COLOR1, errors), 'Would you like to view the errors?[/COLOR]', nolabel='[B][COLOR red]No Thanks[/COLOR][/B]',yeslabel='[B][COLOR green]View Errors[/COLOR][/B]')

if yes:

TextBox(ADDONTITLE, error.replace('\t',''))

setS('installed', 'true')

setS('extract', str(percent))

setS('errors', str(errors))

try: os.remove(file)

except: pass

if INSTALLMETHOD == 1: todo = 1

elif INSTALLMETHOD == 2: todo = 0

else: todo = DIALOG.yesno(ADDONTITLE, "[COLOR %s]Would you like to [COLOR %s]Force close[/COLOR] kodi or [COLOR %s]Reload Profile[/COLOR]?[/COLOR]" % (COLOR2, COLOR1, COLOR1), yeslabel="[B][COLOR red]Reload Profile[/COLOR][/B]", nolabel="[B][COLOR green]Force Close[/COLOR][/B]")

if todo == 1: reloadFix()

else: killxbmc(True)

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

###DETERMINE PLATFORM#####

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

def platform():

if xbmc.getCondVisibility('system.platform.android'): return 'android'

elif xbmc.getCondVisibility('system.platform.linux'): return 'linux'

elif xbmc.getCondVisibility('system.platform.linux.Raspberrypi'): return 'linux'

elif xbmc.getCondVisibility('system.platform.windows'): return 'windows'

elif xbmc.getCondVisibility('system.platform.osx'): return 'osx'

elif xbmc.getCondVisibility('system.platform.atv2'): return 'atv2'

elif xbmc.getCondVisibility('system.platform.ios'): return 'ios'

elif xbmc.getCondVisibility('system.platform.darwin'): return 'ios'

def Grab\_Log(file=False, old=False, wizard=False):

if wizard == True:

if not os.path.exists(WIZLOG): return False

else:

if file == True:

return WIZLOG

else:

filename = open(WIZLOG, 'r')

logtext = filename.read()

filename.close()

return logtext

finalfile = 0

logfilepath = os.listdir(LOG)

logsfound = []

for item in logfilepath:

if old == True and item.endswith('.old.log'): logsfound.append(os.path.join(LOG, item))

elif old == False and item.endswith('.log') and not item.endswith('.old.log'): logsfound.append(os.path.join(LOG, item))

if len(logsfound) > 0:

logsfound.sort(key=lambda f: os.path.getmtime(f))

if file == True: return logsfound[-1]

else:

filename = open(logsfound[-1], 'r')

logtext = filename.read()

filename.close()

return logtext

else:

return False

def whiteList(do):

backup = xbmc.translatePath(BACKUPLOCATION)

mybuilds = xbmc.translatePath(MYBUILDS)

if do == 'edit':

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

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

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

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

if foldername in EXCLUDES: continue

elif foldername in DEFAULTPLUGINS: continue

elif foldername == 'packages': continue

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

if os.path.exists(xml):

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)

fold2 = glob.glob(os.path.join(ADDOND, '\*/'))

for folder in sorted(fold2, key = lambda x: x):

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

if foldername in addonfolds: continue

if foldername in EXCLUDES: continue

xml = os.path.join(ADDONS, foldername, 'addon.xml')

xml2 = os.path.join(XBMC, 'addons', foldername, 'addon.xml')

if os.path.exists(xml):

f = open(xml)

elif os.path.exists(xml2):

f = open(xml2)

else: continue

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)

selected = []; choice = 0

tempaddonnames = ["-- Click here to Continue --"] + addonnames

currentWhite = whiteList('read')

for item in currentWhite:

log(str(item), xbmc.LOGDEBUG)

try: name, id, fold = item

except Exception, e: log(str(e))

if id in addonids:

pos = addonids.index(id)+1

selected.append(pos-1)

tempaddonnames[pos] = "[B][COLOR %s]%s[/COLOR][/B]" % (COLOR1, name)

else:

addonids.append(id)

addonnames.append(name)

tempaddonnames.append("[B][COLOR %s]%s[/COLOR][/B]" % (COLOR1, name))

choice = 1

while not choice in [-1, 0]:

choice = DIALOG.select("%s: Select the addons you wish to White List." % ADDONTITLE, tempaddonnames)

if choice == -1: break

elif choice == 0: break

else:

choice2 = (choice-1)

if choice2 in selected:

selected.remove(choice2)

tempaddonnames[choice] = addonnames[choice2]

else:

selected.append(choice2)

tempaddonnames[choice] = "[B][COLOR %s]%s[/COLOR][/B]" % (COLOR1, addonnames[choice2])

whitelist = []

if len(selected) > 0:

for addon in selected:

whitelist.append("['%s', '%s', '%s']" % (addonnames[addon], addonids[addon], addonfolds[addon]))

writing = '\n'.join(whitelist)

f = open(WHITELIST, 'w'); f.write(writing); f.close()

else:

try: os.remove(WHITELIST)

except: pass

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]%s Addons in White List[/COLOR]" % (COLOR2, len(selected)))

elif do == 'read' :

white = []

if os.path.exists(WHITELIST):

f = open(WHITELIST)

a = f.read()

f.close()

lines = a.split('\n')

for item in lines:

try:

name, id, fold = eval(item)

white.append(eval(item))

except:

pass

return white

elif do == 'view' :

list = whiteList('read')

if len(list) > 0:

msg = "Here is a list of your whitelist items, these items(along with dependencies) will not be removed when preforming a fresh start or the userdata overwritten in a build install.[CR][CR]"

for item in list:

try: name, id, fold = item

except Exception, e: log(str(e))

msg += "[COLOR %s]%s[/COLOR] [COLOR %s]\"%s\"[/COLOR][CR]" % (COLOR1, name, COLOR2, id)

TextBox("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), msg)

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]No items in White List[/COLOR]" % COLOR2)

elif do == 'import':

source = DIALOG.browse(1, '[COLOR %s]Select the whitelist file to import[/COLOR]' % COLOR2, 'files', '.txt', False, False, HOME)

log(str(source))

if not source.endswith('.txt'):

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Import Cancelled![/COLOR]" % COLOR2)

return

f = xbmcvfs.File(source)

a = f.read()

f.close()

current = whiteList('read'); idList = []; count = 0

for item in current:

name, id, fold = item

idList.append(id)

lines = a.split('\n')

with open(WHITELIST, 'a') as f:

for item in lines:

try:

name, id, folder = eval(item)

except Exception, e:

log("Error Adding: '%s' / %s" % (item, str(e)), xbmc.LOGERROR)

continue

log("%s / %s / %s" % (name, id, folder), xbmc.LOGDEBUG)

if not id in idList:

count += 1

writing = "['%s', '%s', '%s']" % (name, id, folder)

if len(idList) + count > 1: writing = "\n%s" % writing

f.write(writing)

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]%s Item(s) Added[/COLOR]" % (COLOR2, count))

elif do == 'export':

source = DIALOG.browse(3, '[COLOR %s]Select where you wish to export the whitelist file[/COLOR]' % COLOR2, 'files', '.txt', False, False, HOME)

log(str(source), xbmc.LOGDEBUG)

try:

xbmcvfs.copy(WHITELIST, os.path.join(source, 'whitelist.txt'))

DIALOG.ok(ADDONTITLE, "[COLOR %s]Whitelist has been exported to:[/COLOR]" % (COLOR2), "[COLOR %s]%s[/COLOR]" % (COLOR1, os.path.join(source, 'whitelist.txt')))

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Whitelist Exported[/COLOR]" % (COLOR2))

except Exception, e:

log("Export Error: %s" % str(e), xbmc.LOGERROR)

if not DIALOG.yesno(ADDONTITLE, "[COLOR %s]The location you selected isnt writable would you like to select another one?[/COLOR]" % COLOR2, yeslabel="[B][COLOR green]Change Location[/COLOR][/B]", nolabel="[B][COLOR red]No Cancel[/COLOR][/B]"):

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Whitelist Export Cancelled[/COLOR]" % (COLOR2, e))

else:

whitelist(export)

elif do == 'clear':

if not DIALOG.yesno(ADDONTITLE, "[COLOR %s]Are you sure you want to clear your whitelist?" % COLOR2, "This process can't be undone.[/COLOR]", yeslabel="[B][COLOR green]Yes Remove[/COLOR][/B]", nolabel="[B][COLOR red]No Cancel[/COLOR][/B]"):

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Clear Whitelist Cancelled[/COLOR]" % (COLOR2))

return

try:

os.remove(WHITELIST)

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Whitelist Cleared[/COLOR]" % (COLOR2))

except:

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Error Clearing Whitelist![/COLOR]" % (COLOR2))

###################################Added Startup Maint###########################################################

def clearThumb(type=None):

latest = latestDB('Textures')

size = getS('filesizethumb\_alert')

folder = convertSize(getSize(THUMBS))

if not type == None: choice = 1

else: choice = DIALOG.yesno("[COLOR=%s]%s[/COLOR]",'[COLOR %s] The thumbnail folder Has exceeded the size of [COLOR red]%s MB[/COLOR] ' % (COLOR1,ADDONTITLE,COLOR4, size), '[COLOR %s]Would you like to delete the [COLOR red]%s[/COLOR] of them?' % (COLOR4, folder), "They will repopulate on the next startup[/COLOR]", nolabel='[B]Don\'t Delete[/B]', yeslabel='[B]Delete Thumbs[/B]')

if choice == 1:

try: removeFile(os.join(DATABASE, latest))

except: log('Failed to delete, Purging DB.'); purgeDb(latest)

removeFolder(THUMBS)

if not type == 'total': killxbmc()

else: log('Clear thumbnames cancelled')

def clearPackagesStart(over=None):

filesize = getS('filesize\_alert')

if os.path.exists(PACKAGES):

try:

for root, dirs, files in os.walk(PACKAGES):

file\_count = 0

file\_count += len(files)

if file\_count > 0:

size = convertSize(getSize(PACKAGES))

if over: yes=1

else: yes=DIALOG.yesno("[COLOR=%s]%s[/COLOR",'[COLOR %s]The packages folder Has exceeded the size of [COLOR red]%s MB[/COLOR] ' % (COLOR1,ADDONTITLE,COLOR4, filesize), "[COLOR %s]%s[/COLOR] files found / [COLOR %s]%s[/COLOR] in size." % (COLOR1, str(file\_count), COLOR3, size), "Do you want to delete them?", nolabel='[B]Don\'t Clear[/B]',yeslabel='[B]Clear Packages[/B]')

if yes:

for f in files: os.unlink(os.path.join(root, f))

for d in dirs: shutil.rmtree(os.path.join(root, d))

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE),'[COLOR %s]Clear Packages: Success![/COLOR]' % COLOR2)

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE),'[COLOR %s]Clear Packages: None Found![/COLOR]' % COLOR2)

except Exception, e:

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE),'[COLOR %s]Clear Packages: Error![/COLOR]' % COLOR2)

log("Clear Packages Error: %s" % str(e), xbmc.LOGERROR)

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE),'[COLOR %s]Clear Packages: None Found![/COLOR]' % COLOR2)

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

def clearPackages(over=None):

if os.path.exists(PACKAGES):

try:

for root, dirs, files in os.walk(PACKAGES):

file\_count = 0

file\_count += len(files)

if file\_count > 0:

size = convertSize(getSize(PACKAGES))

if over: yes=1

else: yes=DIALOG.yesno("[COLOR %s]Delete Package Files" % COLOR2, "[COLOR %s]%s[/COLOR] files found / [COLOR %s]%s[/COLOR] in size." % (COLOR1, str(file\_count), COLOR1, size), "Do you want to delete them?[/COLOR]", nolabel='[B][COLOR red]Don\'t Clear[/COLOR][/B]',yeslabel='[B][COLOR green]Clear Packages[/COLOR][/B]')

if yes:

for f in files: os.unlink(os.path.join(root, f))

for d in dirs: shutil.rmtree(os.path.join(root, d))

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE),'[COLOR %s]Clear Packages: Success![/COLOR]' % COLOR2)

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE),'[COLOR %s]Clear Packages: None Found![/COLOR]' % COLOR2)

except Exception, e:

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE),'[COLOR %s]Clear Packages: Error![/COLOR]' % COLOR2)

log("Clear Packages Error: %s" % str(e), xbmc.LOGERROR)

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE),'[COLOR %s]Clear Packages: None Found![/COLOR]' % COLOR2)

def clearPackagesStartup():

start = datetime.utcnow() - timedelta(minutes=3)

file\_count = 0; cleanupsize = 0

if os.path.exists(PACKAGES):

pack = os.listdir(PACKAGES)

pack.sort(key=lambda f: os.path.getmtime(os.path.join(PACKAGES, f)))

try:

for item in pack:

file = os.path.join(PACKAGES, item)

lastedit = datetime.utcfromtimestamp(os.path.getmtime(file))

if lastedit <= start:

if os.path.isfile(file):

file\_count += 1

cleanupsize += os.path.getsize(file)

os.unlink(file)

elif os.path.isdir(file):

cleanupsize += getSize(file)

cleanfiles, cleanfold = cleanHouse(file)

file\_count += cleanfiles + cleanfold

try:

shutil.rmtree(file)

except Exception, e:

log("Failed to remove %s: %s" % (file, str(e), xbmc.LOGERROR))

if file\_count > 0: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), '[COLOR %s]Clear Packages: Success: %s[/COLOR]' % (COLOR2, convertSize(cleanupsize)))

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), '[COLOR %s]Clear Packages: None Found![/COLOR]' % COLOR2)

except Exception, e:

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), '[COLOR %s]Clear Packages: Error![/COLOR]' % COLOR2)

log("Clear Packages Error: %s" % str(e), xbmc.LOGERROR)

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), '[COLOR %s]Clear Packages: None Found![/COLOR]' % COLOR2)

def clearArchive():

if os.path.exists(ARCHIVE\_CACHE):

cleanHouse(ARCHIVE\_CACHE)

def clearCache(over=None):

PROFILEADDONDATA = os.path.join(PROFILE,'addon\_data')

dbfiles = [

(os.path.join(ADDOND, 'plugin.video.phstreams', 'cache.db')),

(os.path.join(ADDOND, 'plugin.video.bob', 'cache.db')),

(os.path.join(ADDOND, 'plugin.video.zen', 'cache.db')),

(os.path.join(ADDOND, 'plugin.video.specto', 'cache.db')),

(os.path.join(ADDOND, 'plugin.video.genesis', 'cache.db')),

(os.path.join(ADDOND, 'plugin.video.exodus', 'cache.db')),

(os.path.join(DATABASE, 'onechannelcache.db')),

(os.path.join(DATABASE, 'saltscache.db')),

(os.path.join(DATABASE, 'saltshd.lite.db'))]

cachelist = [

(PROFILEADDONDATA),

(ADDOND),

(os.path.join(HOME,'cache')),

(os.path.join(HOME,'temp')),

(os.path.join('/private/var/mobile/Library/Caches/AppleTV/Video/', 'Other')),

(os.path.join('/private/var/mobile/Library/Caches/AppleTV/Video/', 'LocalAndRental')),

(os.path.join(ADDOND,'script.module.simple.downloader')),

(os.path.join(ADDOND,'plugin.video.itv','Images')),

(os.path.join(PROFILEADDONDATA,'script.module.simple.downloader')),

(os.path.join(PROFILEADDONDATA,'plugin.video.itv','Images'))]

delfiles = 0

excludes = ['meta\_cache', 'archive\_cache']

for item in cachelist:

if not os.path.exists(item): continue

if not item in [ADDOND, PROFILEADDONDATA]:

for root, dirs, files in os.walk(item):

dirs[:] = [d for d in dirs if d not in excludes]

file\_count = 0

file\_count += len(files)

if file\_count > 0:

for f in files:

if not f in LOGFILES:

try:

os.unlink(os.path.join(root, f))

log("[Wiped] %s" % os.path.join(root, f), xbmc.LOGNOTICE)

delfiles += 1

except:

pass

else: log('Ignore Log File: %s' % f, xbmc.LOGNOTICE)

for d in dirs:

try:

shutil.rmtree(os.path.join(root, d))

delfiles += 1

log("[Success] cleared %s files from %s" % (str(file\_count), os.path.join(item,d)), xbmc.LOGNOTICE)

except:

log("[Failed] to wipe cache in: %s" % os.path.join(item,d), xbmc.LOGNOTICE)

else:

for root, dirs, files in os.walk(item):

dirs[:] = [d for d in dirs if d not in excludes]

for d in dirs:

if not str(d.lower()).find('cache') == -1:

try:

shutil.rmtree(os.path.join(root, d))

delfiles += 1

log("[Success] wiped %s " % os.path.join(root,d), xbmc.LOGNOTICE)

except:

log("[Failed] to wipe cache in: %s" % os.path.join(item,d), xbmc.LOGNOTICE)

if INCLUDEVIDEO == 'true' and over == None:

files = []

if INCLUDEALL == 'true': files = dbfiles

else:

if INCLUDEBOB == 'true': files.append(os.path.join(ADDOND, 'plugin.video.bob', 'cache.db'))

if INCLUDEPHOENIX == 'true': files.append(os.path.join(ADDOND, 'plugin.video.phstreams', 'cache.db'))

if INCLUDESPECTO == 'true': files.append(os.path.join(ADDOND, 'plugin.video.specto', 'cache.db'))

if INCLUDEGENESIS == 'true': files.append(os.path.join(ADDOND, 'plugin.video.genesis', 'cache.db'))

if INCLUDEZEN == 'true': files.append(os.path.join(ADDOND, 'plugin.video.zen', 'cache.db'))

if INCLUDEEXODUS == 'true': files.append(os.path.join(ADDOND, 'plugin.video.exodus', 'cache.db'))

if INCLUDEONECHAN == 'true': files.append(os.path.join(DATABASE, 'onechannelcache.db'))

if INCLUDESALTS == 'true': files.append(os.path.join(DATABASE, 'saltscache.db'))

if INCLUDESALTSHD == 'true': files.append(os.path.join(DATABASE, 'saltshd.lite.db'))

if len(files) > 0:

for item in files:

if os.path.exists(item):

delfiles += 1

try:

textdb = database.connect(item)

textexe = textdb.cursor()

except Exception, e:

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

continue

if 'Database' in item:

try:

textexe.execute("DELETE FROM url\_cache")

textexe.execute("VACUUM")

textdb.commit()

textexe.close()

log("[Success] wiped %s" % item, xbmc.LOGNOTICE)

except Exception, e:

log("[Failed] wiped %s: %s" % (item, str(e)), xbmc.LOGNOTICE)

else:

textexe.execute("SELECT name FROM sqlite\_master WHERE type = 'table'")

for table in textexe.fetchall():

try:

textexe.execute("DELETE FROM %s" % table[0])

textexe.execute("VACUUM")

textdb.commit()

log("[Success] wiped %s in %s" % (table[0], item), xbmc.LOGNOTICE)

except Exception, e:

try:

log("[Failed] wiped %s in %s: %s" % (table[0], item, str(e)), xbmc.LOGNOTICE)

except:

pass

textexe.close()

else: log("Clear Cache: Clear Video Cache Not Enabled", xbmc.LOGNOTICE)

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), '[COLOR %s]Clear Cache: Removed %s Files[/COLOR]' % (COLOR2, delfiles))

def regex(text, from\_string, to\_string, excluding=True):

if excluding:

try: r = re.search("(?i)" + from\_string + "([\S\s]+?)" + to\_string, text).group(1)

except: r = ''

else:

try: r = re.search("(?i)(" + from\_string + "[\S\s]+?" + to\_string + ")", text).group(1)

except: r = ''

return r

def checkSources():

if not os.path.exists(SOURCES):

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]No Sources.xml File Found![/COLOR]" % COLOR2)

return False

x = 0

bad = []

remove = []

f = open(SOURCES)

a = f.read()

temp = a.replace('\r','').replace('\n','').replace('\t','')

match = re.compile('<files>.+?</files>').findall(temp)

f.close()

if len(match) > 0:

match2 = re.compile('<source>.+?<name>(.+?)</name>.+?<path pathversion="1">(.+?)</path>.+?<allowsharing>(.+?)</allowsharing>.+?</source>').findall(match[0])

DP.create(ADDONTITLE, "[COLOR %s]Scanning Sources for Broken links[/COLOR]" % COLOR2)

for name, path, sharing in match2:

x += 1

perc = int(percentage(x, len(match2)))

DP.update(perc, '', "[COLOR %s]Checking [COLOR %s]%s[/COLOR]:[/COLOR]" % (COLOR2, COLOR1, name), "[COLOR %s]%s[/COLOR]" % (COLOR1, path))

if 'http' in path:

working = workingURL(path)

if not working == True:

bad.append([name, path, sharing, working])

log("Bad Sources: %s" % len(bad), xbmc.LOGNOTICE)

if len(bad) > 0:

choice = DIALOG.yesno(ADDONTITLE, "[COLOR %s]%s[/COLOR][COLOR %s] Source(s) have been found Broken" % (COLOR1, len(bad), COLOR2),"Would you like to Remove all or choose one by one?[/COLOR]", yeslabel="[B][COLOR green]Remove All[/COLOR][/B]", nolabel="[B][COLOR red]Choose to Delete[/COLOR][/B]")

if choice == 1:

remove = bad

else:

for name, path, sharing, working in bad:

log("%s sources: %s, %s" % (name, path, working), xbmc.LOGNOTICE)

if DIALOG.yesno(ADDONTITLE, "[COLOR %s]%s[/COLOR][COLOR %s] was reported as non working" % (COLOR1, name, COLOR2), "[COLOR %s]%s[/COLOR]" % (COLOR1, path), "[COLOR %s]%s[/COLOR]" % (COLOR1, working), yeslabel="[B][COLOR green]Remove Source[/COLOR][/B]", nolabel="[B][COLOR red]Keep Source[/COLOR][/B]"):

remove.append([name, path, sharing, working])

log("Removing Source %s" % name, xbmc.LOGNOTICE)

else: log("Source %s was not removed" % name, xbmc.LOGNOTICE)

if len(remove) > 0:

for name, path, sharing, working in remove:

a = a.replace('\n <source>\n <name>%s</name>\n <path pathversion="1">%s</path>\n <allowsharing>%s</allowsharing>\n </source>' % (name, path, sharing), '')

log("Removing Source %s" % name, xbmc.LOGNOTICE)

f = open(SOURCES, mode='w')

f.write(str(a))

f.close()

alive = len(match) - len(bad)

kept = len(bad) - len(remove)

removed = len(remove)

DIALOG.ok(ADDONTITLE, "[COLOR %s]Checking sources for broken paths has been completed" % COLOR2, "Working: [COLOR %s]%s[/COLOR] | Kept: [COLOR %s]%s[/COLOR] | Removed: [COLOR %s]%s[/COLOR][/COLOR]" % (COLOR2, COLOR1, alive, COLOR1, kept, COLOR1, removed))

else: log("No Bad Sources to be removed.", xbmc.LOGNOTICE)

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]All Sources Are Working[/COLOR]" % COLOR2)

else: log("No Sources Found", xbmc.LOGNOTICE)

def checkRepos():

DP.create(ADDONTITLE, '[COLOR %s]Checking Repositories...[/COLOR]' % COLOR2)

badrepos = []

ebi('UpdateAddonRepos')

repolist = glob.glob(os.path.join(ADDONS,'repo\*'))

if len(repolist) == 0:

DP.close()

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]No Repositories Found![/COLOR]" % COLOR2)

return

sleeptime = len(repolist); start = 0;

while start < sleeptime:

start += 1

if DP.iscanceled(): break

perc = int(percentage(start, sleeptime))

DP.update(perc, '', '[COLOR %s]Checking: [/COLOR][COLOR %s]%s[/COLOR]' % (COLOR2, COLOR1, repolist[start-1].replace(ADDONS, '')[1:]))

xbmc.sleep(1000)

if DP.iscanceled():

DP.close()

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Enabling Addons Cancelled[/COLOR]" % COLOR2)

sys.exit()

DP.close()

logfile = Grab\_Log(False)

fails = re.compile('CRepositoryUpdateJob(.+?)failed').findall(logfile)

for item in fails:

log("Bad Repository: %s " % item, xbmc.LOGNOTICE)

brokenrepo = item.replace('[','').replace(']','').replace(' ','').replace('/','').replace('\\','')

if not brokenrepo in badrepos:

badrepos.append(brokenrepo)

if len(badrepos) > 0:

msg = "[COLOR %s]Below is a list of Repositories that did not resolve. This does not mean that they are Depreciated, sometimes hosts go down for a short period of time. Please do serveral scans of your repository list before removing a repository just to make sure it is broken.[/COLOR][CR][CR][COLOR %s]" % (COLOR2, COLOR1)

msg += '[CR]'.join(badrepos)

msg += '[/COLOR]'

TextBox("%s: Bad Repositories" % ADDONTITLE, msg)

else:

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]All Repositories Working![/COLOR]" % COLOR2)

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

####KILL XBMC ###############

#####THANKS BRACKETS ########

def killxbmc(over=None):

if over: choice = 1

else: choice = DIALOG.yesno('Force Close Kodi', '[COLOR %s]You are about to close Kodi' % COLOR2, 'Would you like to continue?[/COLOR]', nolabel='[B][COLOR red] No Cancel[/COLOR][/B]',yeslabel='[B][COLOR green]Force Close Kodi[/COLOR][/B]')

if choice == 1:

log("Force Closing Kodi: Platform[%s]" % str(platform()), xbmc.LOGNOTICE)

os.\_exit(1)

def redoThumbs():

if not os.path.exists(THUMBS): os.makedirs(THUMBS)

thumbfolders = '0123456789abcdef'

videos = os.path.join(THUMBS, 'Video', 'Bookmarks')

for item in thumbfolders:

foldname = os.path.join(THUMBS, item)

if not os.path.exists(foldname): os.makedirs(foldname)

if not os.path.exists(videos): os.makedirs(videos)

def reloadFix(default=None):

DIALOG.ok(ADDONTITLE, "[COLOR %s]WARNING: Sometimes Reloading the Profile causes Kodi to crash. While Kodi is Reloading the Profile Please Do Not Press Any Buttons![/COLOR]" % COLOR2)

if not os.path.exists(PACKAGES): os.makedirs(PACKAGES)

if default == None:

lookandFeelData('save')

redoThumbs()

ebi('ActivateWindow(Home)')

reloadProfile()

xbmc.sleep(10000)

if KODIV >= 17: kodi17Fix()

if default == None:

log("Switching to: %s" % getS('defaultskin'))

gotoskin = getS('defaultskin')

swapSkins(gotoskin)

lookandFeelData('restore')

addonUpdates('reset')

forceUpdate()

ebi("ReloadSkin()")

def skinToDefault(title):

if not currSkin() in ['skin.confluence', 'skin.estuary']:

skin = 'skin.confluence' if KODIV < 17 else 'skin.estuary'

return swapSkins(skin, title)

def swapSkins(goto, title="Error"):

skinSwitch.swapSkins(goto)

x = 0

xbmc.sleep(1000)

while not xbmc.getCondVisibility("Window.isVisible(yesnodialog)") and x < 150:

x += 1

xbmc.sleep(100)

#ebi('SendAction(Select)')

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

ebi('SendClick(11)')

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), '[COLOR %s]%s: Skin Swap Timed Out![/COLOR]' % (COLOR2, title)); return False

return True

def mediaCenter():

if str(HOME).lower().find('kodi'):

return 'Kodi'

elif str(HOME).lower().find('spmc'):

return 'SPMC'

else:

return 'Unknown Fork'

def kodi17Fix():

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

disabledAddons = []

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

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

if os.path.exists(addonxml):

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

f = open(addonxml)

a = f.read()

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

f.close()

try:

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

else: addonid = fold

add = xbmcaddon.Addon(id=addonid)

except:

try:

log("%s was disabled" % aid[0], xbmc.LOGDEBUG)

disabledAddons.append(addonid)

except:

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

if len(disabledAddons) > 0:

addonDatabase(disabledAddons, 1, True)

LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), "[COLOR %s]Enabling Addons Complete![/COLOR]" % COLOR2)

forceUpdate()

ebi("ReloadSkin()")

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

dbfile = latestDB('Addons')

dbfile = os.path.join(DATABASE, dbfile)

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

if os.path.exists(dbfile):

try:

textdb = database.connect(dbfile)

textexe = textdb.cursor()

except Exception, e:

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

return False

else: return False

if state == 2:

try:

textexe.execute("DELETE FROM installed WHERE addonID = ?", (addon,))

textdb.commit()

textexe.close()

except Exception, e:

log("Error Removing %s from DB" % addon)

return True

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:

log("Erroring enabling addon: %s" % addon)

def data\_type(str):

datatype = type(str).\_\_name\_\_

return datatype

def RESET():

log("Reset Kodi: Platform[%s]" % str(platform()), xbmc.LOGNOTICE)

xbmc.executebuiltin('UpdateAddonRepos()')

xbmc.executebuiltin('UpdateLocalAddons()')

xbmc.executebuiltin('ActivateWindow(Home)')

xbmc.executebuiltin('Mastermode')

xbmc.executebuiltin('LoadProfile(Master user,[prompt])')

xbmc.executebuiltin('ActivateWindow(Home)')

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

### PURGE DATABASE #######

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

def purgeDb(name):

#dbfile = name.replace('.db','').translate(None, digits)

#if dbfile not in ['Addons', 'ADSP', 'Epg', 'MyMusic', 'MyVideos', 'Textures', 'TV', 'ViewModes']: return False

#textfile = os.path.join(DATABASE, name)

log('Purging DB %s.' % name, xbmc.LOGNOTICE)

if os.path.exists(name):

try:

textdb = database.connect(name)

textexe = textdb.cursor()

except Exception, e:

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

return False

else: log('%s not found.' % name, xbmc.LOGERROR); return False

textexe.execute("SELECT name FROM sqlite\_master WHERE type = 'table'")

for table in textexe.fetchall():

if table[0] == 'version':

log('Data from table `%s` skipped.' % table[0], xbmc.LOGDEBUG)

else:

try:

textexe.execute("DELETE FROM %s" % table[0])

textdb.commit()

log('Data from table `%s` cleared.' % table[0], xbmc.LOGDEBUG)

except Exception, e: log("DB Remove Table `%s` Error: %s" % (table[0], str(e)), xbmc.LOGERROR)

textexe.close()

log('%s DB Purging Complete.' % name, xbmc.LOGNOTICE)

show = name.replace('\\', '/').split('/')

LogNotify("[COLOR %s]Purge Database[/COLOR]" % COLOR1, "[COLOR %s]%s Complete[/COLOR]" % (COLOR2, show[len(show)-1]))

def oldThumbs():

dbfile = os.path.join(DATABASE, latestDB('Textures'))

use = 10

week = TODAY - timedelta(days=7)

ids = []

images = []

size = 0

if os.path.exists(dbfile):

try:

textdb = database.connect(dbfile)

textexe = textdb.cursor()

except Exception, e:

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

return False

else: log('%s not found.' % dbfile, xbmc.LOGERROR); return False

textexe.execute("SELECT idtexture FROM sizes WHERE usecount < ? AND lastusetime < ?", (use, str(week)))

found = textexe.fetchall()

for rows in found:

idfound = rows[0]

ids.append(idfound)

textexe.execute("SELECT cachedurl FROM texture WHERE id = ?", (idfound, ))

found2 = textexe.fetchall()

for rows2 in found2:

images.append(rows2[0])

log("%s total thumbs cleaned up." % str(len(images)), xbmc.LOGNOTICE)

for id in ids:

textexe.execute("DELETE FROM sizes WHERE idtexture = ?", (id, ))

textexe.execute("DELETE FROM texture WHERE id = ?", (id, ))

textexe.execute("VACUUM")

textdb.commit()

textexe.close()

for image in images:

path = os.path.join(THUMBS, image)

try:

imagesize = os.path.getsize(path)

os.remove(path)

size += imagesize

except:

pass

removed = convertSize(size)

if len(images) > 0: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), '[COLOR %s]Clear Thumbs: %s Files / %s MB[/COLOR]!' % (COLOR2, str(len(images)), removed))

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), '[COLOR %s]Clear Thumbs: None Found![/COLOR]' % COLOR2)

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

def replaceHTMLCodes(txt):

txt = re.sub("(&#[0-9]+)([^;^0-9]+)", "\\1;\\2", txt)

txt = HTMLParser.HTMLParser().unescape(txt)

txt = txt.replace("&quot;", "\"")

txt = txt.replace("&amp;", "&")

return txt

import os

from shutil import \*

def copytree(src, dst, symlinks=False, ignore=None):

names = os.listdir(src)

if ignore is not None:

ignored\_names = ignore(src, names)

else:

ignored\_names = set()

if not os.path.isdir(dst):

os.makedirs(dst)

errors = []

for name in names:

if name in ignored\_names:

continue

srcname = os.path.join(src, name)

dstname = os.path.join(dst, name)

try:

if symlinks and os.path.islink(srcname):

linkto = os.readlink(srcname)

os.symlink(linkto, dstname)

elif os.path.isdir(srcname):

copytree(srcname, dstname, symlinks, ignore)

else:

copy2(srcname, dstname)

except Error, err:

errors.extend(err.args[0])

except EnvironmentError, why:

errors.append((srcname, dstname, str(why)))

try:

copystat(src, dst)

except OSError, why:

errors.extend((src, dst, str(why)))

if errors:

raise Error, errors

def RESTOREFAV():

if os.path.exists(FAVfile):

choice = xbmcgui.Dialog().yesno(ADDONTITLE, 'Do you want to Restore your favorites?', '', '', yeslabel='[COLOR=red]Yes[/COLOR]',nolabel='[COLOR=green]No[/COLOR]')

if choice == 0:

return

elif choice == 1:

DP.create(ADDONTITLE,"Restoring",'', 'Please Wait')

shutil.copy(FAVfile,USERDATA)

xbmc.sleep(5)

DP.close()

DIALOG.ok(ADDONTITLE,'[COLOR=red]COMPLETE[/COLOR]', 'Your favorites are Restored.', '')

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), '[COLOR %s]No Backup found![/COLOR]' % COLOR2)

def BACKUPFAV():

if not os.path.exists(FAVdest): os.makedirs(FAVdest)

if os.path.exists(FAVOURITES):

choice = xbmcgui.Dialog().yesno(ADDONTITLE, 'Do you want to Back-up your favorites?', '', '', yeslabel='[COLOR=red]Yes[/COLOR]',nolabel='[COLOR=green]No[/COLOR]')

if choice == 0:

return

elif choice == 1:

DP.create(ADDONTITLE,"Backing Up Favourites",'', 'Please Wait')

shutil.copy(FAVOURITES, FAVdest)

xbmc.sleep(10)

DP.close()

setS('favouriteslastsave', str(TODAY))

DIALOG.ok(ADDONTITLE,'[COLOR=red]COMPLETE[/COLOR]', 'Your favorites are Backed up.', '')

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), '[COLOR %s]You have no Favourites![/COLOR]' % COLOR2)

def DELFAV():

if os.path.exists(FAVfile):

choice = xbmcgui.Dialog().yesno(ADDONTITLE, 'Are you sure you want to PERMANENTLY delete your backup?!?!', '', '', yeslabel='[COLOR=red]Yes[/COLOR]',nolabel='[COLOR=green]No[/COLOR]')

if choice == 0:

return

elif choice == 1:

shutil.rmtree(os.path.join(FAVdest))#(FAVdest)

DIALOG.ok(ADDONTITLE,'[COLOR=red]COMPLETE[/COLOR]', 'Backed up deleted.', '')

else: LogNotify("[COLOR %s]%s[/COLOR]" % (COLOR1, ADDONTITLE), '[COLOR %s]No Favourites to remove![/COLOR]' % COLOR2)

def getAttributesByTagName(dom, tagName):

elem = dom.getElementsByTagName(tagName)[0]

return dict(list(elem.attributes.items()))

def build\_request(url, data=None, headers={}):

if url[0] == ':':

schemed\_url = '%s%s' % (scheme, url)

else:

schemed\_url = url

headers['User-Agent'] = user\_agent

return Request(schemed\_url, data=data, headers=headers)

def catch\_request(request):

try:

uh = urlopen(request)

return uh

except (HTTPError, URLError, socket.error):

e = sys.exc\_info()[1]

return None, e

def getConfig():

request = build\_request('http://www.speedtest.net/speedtest-config.php')

uh = catch\_request(request)

if uh is False:

sys.exit(1)

configxml = []

while 1:

configxml.append(uh.read(10240))

if len(configxml[-1]) == 0:

break

if int(uh.code) != 200:

return None

uh.close()

try:

try:

root = ET.fromstring(''.encode().join(configxml))

config = {

'client': root.find('client').attrib,

'times': root.find('times').attrib}

except Exception:

root = DOM.parseString(''.join(configxml))

config = {

'client': getAttributesByTagName(root, 'client'),

'times': getAttributesByTagName(root, 'times')}

except SyntaxError:

sys.exit(1)

del root

del configxml

return config