

FMPasteBox.py

```
#
# FMPasteBox.py
# FMPasteBox
#
5
import objc
import Foundation
import AppKit

10 from PyObjCTools import AppHelper

import FMPasteBoxAppDelegate

if __name__ == '__main__':
15     AppHelper.runEventLoop()
```

FMPasteBoxAppDelegate.py

```
#
# FMPasteBoxAppDelegate.py
# FMPasteBox
#
5
import sys
import os

import pprint
10 pp = pprint.pprint

import pdb
kwlog = True

15 import objc

import Foundation
NSObject = Foundation.NSObject
NSMutableDictionary = Foundation.NSMutableDictionary
20 NSData = Foundation.NSData

import AppKit
NSWindowController = AppKit.NSWindowController
NSApplication = AppKit.NSApplication
25 UserDefaults = AppKit.NSUserDefaults
NSMutableAttributedString = AppKit.NSMutableAttributedString
NSBeep = AppKit.NSBeep
NSPasteboard = AppKit.NSPasteboard

30 import FMPasteBoxLayoutObjects

import FMPasteBoxTools
read_pb = FMPasteBoxTools.read_pb
makeunicode = FMPasteBoxTools.makeunicode
35 fmpPasteboardTypes = FMPasteBoxTools.fmpPasteboardTypes
additionalFMPPasteboardTypes = FMPasteBoxTools.additionalFMPPasteboardTypes
displaynameTypes = FMPasteBoxTools.displaynameTypes
timestamp = FMPasteBoxTools.timestamp

40 import FMPasteBoxVersion

import FMPasteBoxPrefController
PrefController = FMPasteBoxPrefController.FMPasteBoxPreferenceController
```

```

45 class FMPasteBoxAppDelegate(NSObject):

    menClipboardtype = objc.IBOutlet()
    butGetClipboard = objc.IBOutlet()
    butPushClipboard = objc.IBOutlet()
50 tfXMLEditor = objc.IBOutlet()
    appWindow = objc.IBOutlet()

    def initialize(self):
        if kwlog:
55         print "FMPasteBoxAppDelegate.initialize()"
        userdefaults = NSMutableDictionary.dictionary()
        userdefaults.setObject_forKey_(u"", u'txtFileMakerAppPath')
        userdefaults.setObject_forKey_(u"", u'txtExportsPath')
        userdefaults.setObject_forKey_(False, u'cbDoExports')
60         NSUserDefaults.standardUserDefaults().registerDefaults_(userdefaults)
        self.preferenceController = None

    def awakeFromNib(self):
        # for later
65         defaults = NSUserDefaults.standardUserDefaults()

        # set up type menu
        self.menClipboardtype.removeAllItems()
        menuItems = [ u"" ]
70         menuItems.extend( displaynameTypes.keys() )
        menuItems.sort()
        for menuItem in menuItems:
            self.menClipboardtype.addItemWithTitle_( menuItem )
        self.menClipboardtype.setTitle_( u"" )
75         # set up text view
        self.tfXMLEditor.setUsesFindPanel_(True)
        window = self.tfXMLEditor.window()
        window.makeFirstResponder_(self.tfXMLEditor)

80     def applicationDidFinishLaunching_(self, notification):
        app = NSApplication.sharedApplication()
        app.activateIgnoringOtherApps_(True)
        window = self.tfXMLEditor.window()
        window.makeFirstResponder_(self.tfXMLEditor)
85

    @objc.IBAction
    def getClipboard_(self, sender):
        pasteboardContents = read_pb()
        if not pasteboardContents:
90         # abort - nothing on pasteboard
            NSBeep()
            # we must return implicit None! Crashing otherwise.
            return
        defaults = NSUserDefaults.standardUserDefaults()
95         exportClipboards = defaults.boolForKey_( u'cbDoExports' )
        if exportClipboards:
            exportFolder = makeunicode(defaults.objectForKey_( u'txtExportsPath' ))
            if os.path.exists( exportFolder ):
                d,t = FMPasteBoxTools.datetimestamp()
                dayFolder = os.path.join( exportFolder, d )
100                mainType = "-"
                try:
                    mainType = mainType + pasteboardContents.typ.name
                except:
                    pass
105                sessionFolder = os.path.join( dayFolder, t + mainType)
                try:

```

```

        exportItems = pasteboardContents.additional[:]
        exportItems.append( pasteboardContents )
110     for item in exportItems:
            name = item.typ.name
            ext = item.typ.fileExt
            data = item.data
            path = os.path.join( sessionFolder, name + ext )
115     if ext == ".xml":
            data = makeunicode( data )
            data = data.encode( "utf-8" )

            if not os.path.exists( sessionFolder ):
120                os.makedirs( sessionFolder )
            f = open(path, 'w')
            f.write( data )
            f.close()

            if ext == ".xml":
125                FMPasteBoxLayoutObjects.exportAssets( path, sessionFolder )
    except Exception, err:
        print
        print "ADDITIONALS FAILED"
130        print err
        print
        pbType = pasteboardContents.typ
        pbTypeName = pbType.name
        self.menClipboardtype.setTitle_( pbTypeName )
135        self.tfXMLEditor.setString_( makeunicode( pasteboardContents.data ) )
        window = self.tfXMLEditor.window()
        window.makeFirstResponder_(self.tfXMLEditor)

def textView(self):
140    # model
    return makeunicode( self.tfXMLEditor.string() )

@objc.IBAction
def pushClipboard_(self, sender):
145    # get text view data
    data = makeunicode(self.textView())
    data = data.encode("utf-8")
    l = len(data)
    nsdata = NSData.dataWithBytes_length_(data, l)

150    # get pasteboard type
    pasteboardType = displayNameTypes.get( self.menClipboardtype.title(), u"" )
    if not pasteboardType:
        NSBeep()
155    # we must return implicit None! Crashing otherwise.
    return

    # write to pasteboard
    pasteboard = NSPasteboard.generalPasteboard()
    pasteboard.clearContents()
160    pasteboardTypeName = pasteboardType.pbname
    pasteboard.setData_forType_( nsdata, pasteboardTypeName)

@objc.IBAction
def showPreferencePanel_(self, sender):
165    if self.preferenceController == None:
        self.preferenceController = PrefController.alloc().init()
        self.preferenceController.showWindow_( self.preferenceController )

```

FMPasteBoxPrefController.py

```

#
# FMPasteBoxPreferenceController.py
#
# Created by Karsten Wolf on 07.02.18.
5 # Copyright 2018 Karsten Wolf. All rights reserved.
#

import objc

10 import Foundation
    NSUserDefaults = Foundation.NSUserDefaults

    import AppKit
    NSApplication = AppKit.NSApplication
15 NSWindowController = AppKit.NSWindowController

    import FMPasteBoxTools

class FMPasteBoxPreferenceController (NSWindowController):
20
    butSetFileMakerAppPath = objc.IBOutlet()
    butSetExportsPath = objc.IBOutlet()

    cbDoExports = objc.IBOutlet()
25
    txtFileMakerAppPath = objc.IBOutlet()
    txtExportsPath = objc.IBOutlet()

    def init(self):
30
        self = self.initWithWindowNibName_("Preferences")

        wnd = self.window()
        wnd.setTitle_( u"FMPasteBox Preferences" )
        wnd.setDelegate_( self )
35

        defaults = NSUserDefaults.standardUserDefaults()
        self.txtFileMakerAppPath.setStringValue_( defaults.objectForKey_( u'txtFileMakerAppPath' ) )
        self.txtExportsPath.setStringValue_( defaults.objectForKey_( u'txtExportsPath' ) )
        self.cbDoExports.setState_( defaults.objectForKey_( u'cbDoExports' ) )
40
        return self

    def windowWillClose_(self, notification):
        defaults = NSUserDefaults.standardUserDefaults()
        defaults.setObject_forKey_(self.txtFileMakerAppPath.stringValue(), u'txtFileMakerAppPath')
45
        defaults.setObject_forKey_(self.txtExportsPath.stringValue(), u'txtExportsPath')
        defaults.setObject_forKey_(self.cbDoExports.state(), u'cbDoExports')

    @objc.IBAction
    def chooseFolder_(self, sender):
50
        if sender == self.butSetFileMakerAppPath:
            folders = FMPasteBoxTools.getApplicationDialog()
            if folders:
                self.txtFileMakerAppPath.setStringValue_( folders )
        elif sender == self.butSetExportsPath:
55
            folders = FMPasteBoxTools.getFolderDialog()
            if folders:
                self.txtExportsPath.setStringValue_( folders[0] )

```

FMPasteBoxTools.py

```

# -*- coding: utf-8 -*-

```

```

"""Some tools which are needed by most files.

```

```

"""
5
import sys
import os
import re
import struct
10 import traceback
import datetime
import unicodedata
import hashlib

15 import xml.etree.cElementTree
    ElementTree = xml.etree.cElementTree

import mactypes
import appscript
20 asc = appscript

import pdb
import FMPasteBoxVersion
kwdbg = FMPasteBoxVersion.developmentversion
25 kwlog = FMPasteBoxVersion.developmentversion

import pprint
pp = pprint.pprint

30 import urllib
import urlparse

import objc

35 import Foundation
NSURL = Foundation.NSURL
NSFileManager = Foundation.NSFileManager
NSUserDefaults = Foundation.NSUserDefaults
NSString = Foundation.NSString
40
import AppKit
NSOpenPanel = AppKit.NSOpenPanel
NSAlert = AppKit.NSAlert
NSSavePanel = AppKit.NSSavePanel
45 NSFileHandlingPanelOKButton = AppKit.NSFileHandlingPanelOKButton
NSPasteboard = AppKit.NSPasteboard
NSPasteboardCommunicationException = AppKit.NSPasteboardCommunicationException

def num2ostype( num ):
50     if num == 0:
        return '????'
    s = struct.pack(">I", num)
    return makeunicode(s, "macroman")

55 def ostype2num( ostype ):
    return struct.pack('BBBB', list(ostype))

def makeunicode(s, srcencoding="utf-8", normalizer="NFC"):
    try:
60         if type(s) not in (unicode, objc.pyobjc_unicode):
            s = unicode(s, srcencoding)
        except TypeError:
            print "makeunicode type conversion error"
            print "FAILED converting", type(s), "to unicode"
65     s = unicodedata.normalize(normalizer, s)
    return s

```

```

def NSURL2str( nsurl ):
    if isinstance(nsurl, NSURL):
70         return str(nsurl.absoluteString())
    return nsurl

def getFileProperties( theFile ):
    """
75     """
    sfm = NSFileManager defaultManager()
    props = sfm.fileAttributesAtPath_traverseLink_( theFile, True )
    if not props:
        return {}
80    mtprops = props.mutableCopy()
    mtprops.removeObjectsForKeys_( [
        u"NSFileExtensionHidden",
        u"NSFileGroupOwnerAccountID",
        u"NSFileGroupOwnerAccountName",
85        u"NSFileOwnerAccountID",
        u"NSFileOwnerAccountName",
        #u"NSFilePosixPermissions",
        #u"NSFileReferenceCount",
        # u"NSFileSize",
90        #u"NSFileSystemFileNumber",
        u"NSFileSystemNumber",
        u"NSFileType",
        # u"NSFileHFSCreatorCode",
        # u"NSFileHFSTypeCode",
95        #u"NSFileCreationDate"
    ] )
    return mtprops

def setFileProperties( theFile, props ):
100    sfm = NSFileManager defaultManager()
    return sfm.changeFileAttributes_atPath_( props, theFile )

def datestring_nsdate( dt=datetime.datetime.now() ):
    now = str(dt)
105    now = now[:19]
    now = now + " +0000"
    return now

def setFileModificationDate( filepath, modfdt ):
110    l = getFileProperties( filepath )
    date = Foundation.NSDate.dateWithString_( datestring_nsdate( modfdt ) )
    l['NSFileModificationDate'] = date
    setFileProperties( filepath, l )
    folder, filename = os.path.split( filepath )
115    print "Setting file(%) modification date to %s" % (filename, repr(modfdt))

def uniquepath(folder, filenamebase, ext, nfill=3, startindex=1, sep="_", always=True):
    """
    """
120    folder = os.path.abspath( folder )

    if not always:
        path = os.path.join(folder, filename + ext )
        if not os.path.exists( path ):
125            return path

    n = startindex
    while True:
        serialstring = str(n).rjust(nfill, "0")
130        filename = filenamebase + sep + serialstring + ext

```

```

        fullpath = os.path.join(folder, filename)

135     if n >= 10*nfill:
        nfill = nfill + 1

        if not os.path.exists(fullpath):
            return fullpath

140     n += 1

def gethashval( s ):
    m = hashlib.shal()
145     size = len(s)

    t = "blob %i\0%s" % (size, s)
    m.update(t)
    return (m.hexdigest(), size)

150 def cancelContinueAlert(title, message, butt1="OK", butt2=False):
    """Run a generic Alert with buttons "Weiter" & "Abbrechen".

    Returns True if "Weiter"; False otherwise
155 """
    alert = NSAlert.alloc().init()
    alert.setAlertStyle_( 0 )
    alert.setInformativeText_( title )
    alert.setMessageText_( message )
160 alert.setShowsHelp_( False )
    alert.addButtonWithTitle_( butt1 )

    if butt2:
        # button 2 has keyboard equivalent "Escape"
165     button2 = alert.addButtonWithTitle_( butt2 )
        button2.setKeyEquivalent_( unichr(27) )

    f = alert.runModal()
    return f == AppKit.NSAlertFirstButtonReturn

170 def errorDialog( message="Error", title="Some error occured..."):
    return cancelContinueAlert(title, message)

def getFileDialog(multiple=False):
175     panel = NSOpenPanel.openPanel()
    panel.setCanChooseFiles_(True)
    panel.setCanChooseDirectories_(False)
    panel.setAllowsMultipleSelection_(multiple)
    rval = panel.runModalForTypes_( None )
180     if rval:
        return [t for t in panel_filenames()]
    return []

def getApplicationDialog():
185     panel = NSOpenPanel.openPanel()
    panel.setCanChooseFiles_(True)
    panel.setCanChooseDirectories_(False)
    panel.setAllowsMultipleSelection_(False)
    rval = panel.runModalForTypes_( ['app'] )
190     if rval:
        l = [makeunicode(t.path()) for t in panel.URLs()]
        return l[0]
    return ""

195 def getFolderDialog(multiple=False):

```

```

    panel = NSOpenPanel.openPanel()
    panel.setCanChooseFiles_(False)
    panel.setCanChooseDirectories_(True)
    panel.setAllowsMultipleSelection_(multiple)
200   rval = panel.runModalForTypes_([])
    if rval:
        return [t for t in panel_filenames()]
    return []

205 def saveAsDialog(path):
    panel = NSSavePanel.savePanel()

    if path:
        panel.setDirectory_( path )
210
    panel.setMessage_( u"Save as OPML" )
    panel.setExtensionHidden_( False )
    panel.setCanSelectHiddenExtension_(True)
    panel.setRequiredFileType_( u"opml" )
215   if path:
        if not os.path.isdir( path ):
            folder, fle = os.path.split(path)
        else:
            folder = path
220            fle = "Untitled.opml"
        rval = panel.runModalForDirectory_file_(folder, fle)
    else:
        rval = panel.runModal()

225   if rval == NSFileHandlingPanelOKButton:
        return panel.filename()
    return False

def timestamp( dt=None ):
230   # '2018-02-17 19:41:02'
    if not dt:
        dt = datetime.datetime.now()
    now = str(dt)
    now = now[:19]
235   d, t = now.split()
    t = t.replace(':', '')
    return (d,t)

def get_type_from_hexstring( hexstring ):
240   """Extract the 4-char macroman type code from the pasteboard type name."""
    h = int(hexstring, 16)
    s = struct.pack(">I", h)
    s = unicode(s, 'macroman')
    return s
245

def get_hexstring_for_type( typ_ ):
    """
    """
    s = struct.pack( "BBBB", typ_ )
250   i = struct.unpack( ">I", s)
    return hex(i)

def get_type_from_intstring( intstring ):
    h = int(intstring)
255   s = struct.pack(">I", h)
    s = unicode(s, 'macroman')
    return s

def get_flavor(s):

```



```

260     """Return the 4-char type from a pasteboard name
        """

        # seems like the standart naming scheme for the pasteboard server
        re_pbtype = re.compile( u"CorePasteboardFlavorType 0x([A-F0-9]{,8})")

265     m = re_pbtype.match(s)
        result = ""
        if m:
            t = m.groups()[0]
270         result = get_type_from_hexstring(t)
        return result

    def writePasteboardFlavour( folder, basename, ext, data ):
        p = uniquepath(folder, basename, ext)
275         if data:
            f = open ( p, 'wb')
            f.write( data )
            f.close()

280 # fmpa 15
        # XML2 - 0x584D4C32 - generic xml for layout objects

        # FMPA 11
        # XMFN - 0x584D464E - Custom Functions
285     # FileMaker Advanced Pasteboard types
        # XMFD - 0x584D4644 - fields
        # XMTB - 0x584D5442 - basetables
        # XMSC - 0x584D5343 - scripts
290 # XMSS - 0x584D5353 - script step
        # XML0 - 0x584D4C4F - layout objects

        # FileMaker Developer Pasteboard types
        # beides binaerformate
295 # FTR5 - 0x46545235 -
        # FMP5

    class PasteboardType(object):
        canonicalTypes = {
300         u'com.adobe.pdf': u'Apple PDF pasteboard type',
            u'public.jpeg': u"CorePasteboardFlavorType 0x4A504547",
            u'NeXT TIFF v4.0 pasteboard type': u'public.tiff',
            # XML2
            u'dyn.ah62d4rv4gk8zuxnqgk': u"CorePasteboardFlavorType 0x584D4C32",
305         }

        def __init__(self, pbname, typ, dataType, name, fileExt):
            self.pbname = pbname
310         self.typ = typ
            self.dataType = dataType
            self.name = name
            self.fileExt = fileExt
            self.canonicalType = self.canonicalTypes.get( pbname, pbname )

315         def __repr__(self):
            return u"PasteboardType(%s, %s, %s, %s, %s, %s)" % (
                repr(self.pbname),
                repr(self.typ),
320                 repr(self.dataType),
                repr(self.name),
                repr(self.fileExt),
                repr(self.canonicalType),)

```

```

325 class PasteboardEntry(object):
    def __init__(self, name, data, typ):
        self.name = name
        self.data = data
        self.typ = typ
330     self.additional = []

    def __repr__(self):
        return u"PasteboardEntry(%s, data[%i], %s, %s)" % (
            repr(self.name),
335             len(self.data),
            repr(self.typ),
            repr(self.additional))

fmpPasteboardTypes = {
340     u"CorePasteboardFlavorType 0x584D4C32":
        PasteboardType(u"CorePasteboardFlavorType 0x584D4C32",
            'XML2', 'fullXML', "Layout Objects", '.xml'),

    u"CorePasteboardFlavorType 0x584D5442":
345     PasteboardType(u"CorePasteboardFlavorType 0x584D5442",
        'XMTB', 'snippetXML', "Base Tables", '.xml'),

    u"CorePasteboardFlavorType 0x584D4644":
        PasteboardType(u"CorePasteboardFlavorType 0x584D4644",
350         'XMGD', 'snippetXML', "Fields", '.xml'),

    u"CorePasteboardFlavorType 0x584D5343":
        PasteboardType(u"CorePasteboardFlavorType 0x584D5343",
            'XMSC', 'snippetXML', "Scripts", '.xml'),
355     u"CorePasteboardFlavorType 0x584D5353":
        PasteboardType(u"CorePasteboardFlavorType 0x584D5353",
            'XMSS', 'snippetXML', "Script Steps", '.xml'),

    u"CorePasteboardFlavorType 0x584D464E":
360     PasteboardType(u"CorePasteboardFlavorType 0x584D464E",
        'XMFN', 'snippetXML', "Custom Functions", '.xml'),

    u"CorePasteboardFlavorType 0x584D4C4F":
365     PasteboardType(u"CorePasteboardFlavorType 0x584D4C4F",
        'XML0', 'snippetXML', "Layout Objects (obsolete)", '.xml'),
}

displaynameTypes = {}
370 # "Custom Functions" -> PasteboardType(u"CorePasteboardFlavorType 0x584D464E",...
    for typeName in fmpPasteboardTypes:
        typ = fmpPasteboardTypes[typeName]
        displaynameTypes[typ.name] = typ

375 additionalFMPPasteboardTypes = {
    u"CorePasteboardFlavorType 0x4A504547":
        PasteboardType(u"CorePasteboardFlavorType 0x4A504547",
            'JPEG', 'binaryData',
            "Layout Objects JPEG Image", '.jpg'),
380     u'Apple PDF pasteboard type':
        PasteboardType(u'Apple PDF pasteboard type',
            'PDF', 'binaryData',
            "Layout Objects PDF Image", '.pdf'),
385     u'com.adobe.pdf':
        PasteboardType(u'com.adobe.pdf',

```

```

        'PDF', 'binaryData',
        "Layout Objects PDF Image", '.pdf'),
390
    u'Apple PICT pasteboard type':
        PasteboardType(u'Apple PICT pasteboard type',
            'PICT', 'binaryData',
            "Layout Objects PICT Image (obsolete)", '.pict'),
395
    u'NeXT TIFF v4.0 pasteboard type':
        PasteboardType(u'NeXT TIFF v4.0 pasteboard type',
            'TIFF', 'binaryData',
            "Layout Objects TIFF Image", '.tif'),
400
    u'public.jpeg':
        PasteboardType(u'public.jpeg',
            'JPEG', 'binaryData',
            "Layout Objects JPEG Image", '.jpg'),
405
    u'public.tiff':
        PasteboardType(u'public.tiff',
            'TIFF', 'binaryData',
            "Layout Objects TIFF Image", '.tif'),
410 }

def read_pb():
    result = None
    hashes = set()
415
    additional = []
    pasteboard = NSPasteboard.generalPasteboard()
    pbTypeNames = pasteboard.types()

420     # additionalFMPPasteboardTypes

    for pbTypeName in pbTypeNames:

        pbType = mainType = None
425         if pbTypeName in fmpPasteboardTypes:
            pbType = fmpPasteboardTypes.get( pbTypeName )
            mainType = True
        elif pbTypeName in additionalFMPPasteboardTypes:
            pbType = additionalFMPPasteboardTypes.get( pbTypeName )
430             mainType = False

        if pbType == None:
            continue

435         try:
            s = pasteboard.dataForType_( pbTypeName )
            data = s.bytes().tobytes()

            # dont load duplicate data
440             hashval, _ = gethashval( data )
            if hashval in hashes:
                continue
            hashes.add( hashval )

445             if mainType:
                data = makeunicode(data)

            pbTypeName = pbType.canonicalType

450             pbEntry = PasteboardEntry(pbTypeName, data, pbType)

```

```

        if mainType:
            result = pbEntry
        else:
455             additional.append( pbEntry )

    except Exception, v:
        print v
        pdb.set_trace()
460         pp(locals())
        print

    if result:
        result.additional = additional
465

    if kwlog:
        print
        print "result = "
        pp(result)
470         print
    return result

```

FMPasteBoxVersion.py

```

import os

appname = "FMPasteBox"
appnameshort = "FMPasteBox"
5 author = "Karsten Wolf"

years = "2018"
copyright = 'Copyright %s %s' % (years, author)

10 version = "0.3.1"
creator = 'KWFP'
bundleID = "org.kw.FMPasteBox"

description = (u"Filemaker Pasteboard interface and editor")
15 longdescription = u""FMPasteBox is a Mac OS X application for translating the FileMaker clipboard.""

#document_creator = "Created by %s %s" % (appname, version)

developmentversion = False

setup.py

"""
Script for building FMPasteBox

Usage:
5     python setup.py py2app
"""

from distutils.core import setup
from setuptools.extension import Extension

10 import py2app

import FMPasteBoxVersion

setup(
15     name = FMPasteBoxVersion.appname,
    version = FMPasteBoxVersion.version,
    description = FMPasteBoxVersion.description,

```

```

long_description = FMPasteBoxVersion.longdescription,
author = FMPasteBoxVersion.author,
20 app=[{
    'script': "FMPasteBox.py",

    "plist": {
        "NSPrincipalClass": 'NSApplication',
25         "CFBundleIdentifier": FMPasteBoxVersion.bundleID,
        "CFBundleName": FMPasteBoxVersion.appnameshort,
        "CFBundleSignature": FMPasteBoxVersion.creator,
        "CFBundleShortVersionString": FMPasteBoxVersion.version,
        "CFBundleGetInfoString": FMPasteBoxVersion.description,
30         "NSHumanReadableCopyright": FMPasteBoxVersion.copyright,
    }
}],

data_files=[
35     "English.lproj/MainMenu.nib",
    "English.lproj/Preferences.nib",
    #"English.lproj/FMPasteBoxDocument.nib",
    "+icon/FMPasteBox.icns",
    #"+icon/FMPasteBoxFile.icns",
40 ],

options={
    "py2app": {
        "iconfile": "+icon/FMPasteBox.icns",
45         # "packages": [],
        "excludes": ["TkInter", 'Tcl', 'Tk'],
    }
} )

```