

FMPasteBox.py

```
#
# FMPasteBox.py
# FMPasteBox
#
5 from __future__ import print_function

import objc
import Foundation
10 import AppKit

from PyObjCTools import AppHelper

import FMPasteBoxAppDelegate
15 # py3 stuff

py3 = False
try:
20     unicode('')
    punicode = unicode
    pstr = str
    punichr = unichr
except NameError:
25     punicode = str
    pstr = bytes
    py3 = True
    punichr = chr

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

FMPasteBoxAppDelegate.py

```
#
# FMPasteBoxAppDelegate.py
# FMPasteBox
#
5 from __future__ import print_function

import sys
import os
10 import io

import pprint
pp = pprint.pprint

15 import pdb
kwlog = True

import objc

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

25 import AppKit
NSWindowController = AppKit.NSWindowController
NSApplication = AppKit.NSApplication
```

```

NSUserDefaults = AppKit.NSUserDefaults
NSMutableAttributedString = AppKit.NSMutableAttributedString
30 NSBeep = AppKit.NSBeep
NSPasteboard = AppKit.NSPasteboard

import FMPasteBoxLayoutObjects

35 import FMPasteBoxTools
read_pb = FMPasteBoxTools.read_pb
makeunicode = FMPasteBoxTools.makeunicode
fmpPasteboardTypes = FMPasteBoxTools.fmpPasteboardTypes
additionalFMPPasteboardTypes = FMPasteBoxTools.additionalFMPPasteboardTypes
40 displaynameTypes = FMPasteBoxTools.displaynameTypes
datetimestamp = FMPasteBoxTools.datetimestamp

import FMPasteBoxVersion

45 import FMPasteBoxPrefController
PrefController = FMPasteBoxPrefController.FMPasteBoxPreferenceController

# py3 stuff

50 py3 = False
try:
    unicode('')
    punicode = unicode
    pstr = str
55    punichr = unichr
except NameError:
    punicode = str
    pstr = bytes
    py3 = True
60    punichr = chr

class FMPasteBoxAppDelegate(NSObject):

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

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

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

        # set up type menu
85        self.menClipboardtype.removeAllItems()
        menuItems = [ u"" ]
        menuItems.extend( displaynameTypes.keys() )
        menuItems.sort()
        for menuItem in menuItems:
90            self.menClipboardtype.addItemWithTitle_( menuItem )
        self.menClipboardtype.setTitle_( u"" )

```

```

    # set up text view
    self.tfXMLEditor.setUsesFindPanel_(True)
    window = self.tfXMLEditor.window()
95    window.makeFirstResponder_(self.tfXMLEditor)

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

@objc.IBAction
    def getClipboard_(self, sender):
105    pasteboardContents = read_pb()
        if not pasteboardContents:
            # abort - nothing on pasteboard
            NSBeep()
            # we must return implicit None! Crashing otherwise.
110        return
        defaults = NSUserDefaults.standardUserDefaults()
        exportClipboards = defaults.boolForKey_( u'cbDoExports' )
        if exportClipboards:
            exportFolder = makeunicode(defaults.objectForKey_( u'txtExportsPath' ))
115
            if os.path.exists( exportFolder ):
                d,t = FMPasteBoxTools.datetimestamp()
                dayFolder = os.path.join( exportFolder, d )
                mainType = "-"
120                try:
                    mainType = mainType + pasteboardContents.typ.name
                except:
                    pass
                sessionFolder = os.path.join( dayFolder, t + mainType)
125                try:
                    # pdb.set_trace()
                    exportItems = pasteboardContents.additionalItems[:]
                    exportItems.append( pasteboardContents )
                    for item in exportItems:
130                        name = item.typ.name
                        ext = item.typ.fileExt
                        data = item.data
                        path = os.path.join( sessionFolder, name + ext )
                        if ext == ".xml":
135                            data = makeunicode( data )
                            data = data.encode("utf-8")

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

                            if ext == ".xml":
145                                # pdb.set_trace()
                                FMPasteBoxLayoutObjects.exportAssets( path, sessionFolder )
                except Exception as err:
                    print()
                    print( "ADDITIONALS FAILED" )
150                    print( err )
                    print()

        pbType = pasteboardContents.typ
        pbTypeName = pbType.name
155    self.menClipboardtype.setTitle_( pbTypeName )

```

```

        self.tfXMLEditor.setString_( makeunicode( pasteboardContents.data ) )
        window = self.tfXMLEditor.window()
        window.makeFirstResponder_(self.tfXMLEditor)

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

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

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

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

    @objc.IBAction
185     def showPreferencePanel_(self, sender):
        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.
#

from __future__ import print_function

10 import objc

import Foundation
NSUserDefaults = Foundation.NSUserDefaults

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

import FMPasteBoxTools
20
# py3 stuff

py3 = False
try:
25     unicode('')
    punicode = unicode
    pstr = str

```

```

        punichr = unichr
except NameError:
30     punicode = str
        pstr = bytes
        py3 = True
        punichr = chr

35 class FMPasteBoxPreferenceController (NSWindowController):

        butSetFileMakerAppPath = objc.IBOutlet()
        butSetExportsPath = objc.IBOutlet()

40     cbDoExports = objc.IBOutlet()

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

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

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

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

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

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

```

FMPasteBoxTools.py

```

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

from __future__ import print_function

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

import sys
import os
10 import re
import struct
import traceback
import datetime
import unicodedata

```

```

15 import hashlib

    import xml.etree.ElementTree
    ElementTree = xml.etree.ElementTree

20 import mactypes
    import appscript
    asc = appscript

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

    import pprint
30 pp = pprint.pprint

    import urllib

    import objc
35
    import Foundation
    NSURL = Foundation.NSURL
    NSFileManager = Foundation.NSFileManager
    NSUserDefaults = Foundation.NSUserDefaults
40 NSString = Foundation.NSString

    import AppKit
    NSOpenPanel = AppKit.NSOpenPanel
    NSAlert = AppKit.NSAlert
45 NSSavePanel = AppKit.NSSavePanel
    NSFileHandlingPanelOKButton = AppKit.NSFileHandlingPanelOKButton
    NSPasteboard = AppKit.NSPasteboard
    NSPasteboardCommunicationException = AppKit.NSPasteboardCommunicationException

50 # py3 stuff

    py3 = False
    try:
        unicode('')
55    punicode = unicode
        pstr = str
        punichr = unichr
    except NameError:
        punicode = str
60    pstr = bytes
        py3 = True
        punichr = chr

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

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

    def makeunicode(s, srcencoding="utf-8", normalizer="NFC"):
    if type(s) not in (punicode, pstr):
75        s = str( s )
        if type(s) != punicode:
            s = punicode(s, srcencoding)
            s = unicodedata.normalize(normalizer, s)

```

```

    return s
80
def NSURL2str( nsurl ):
    if isinstance(nsurl, NSURL):
        return str(nsurl.absoluteString())
    return nsurl
85
def getFileProperties( theFile ):
    """
    """
    sfm = NSFileManager defaultManager()
    props = sfm.fileAttributesAtPath_traverseLink_( theFile, True )
    if not props:
        return {}
    mtprops = props.mutableCopy()
    mtprops.removeObjectsForKeys_( [
95        u"NSFileExtensionHidden",
        u"NSFileGroupOwnerAccountID",
        u"NSFileGroupOwnerAccountName",
        u"NSFileOwnerAccountID",
        u"NSFileOwnerAccountName",
100        #u"NSFilePosixPermissions",
        #u"NSFileReferenceCount",
        # u"NSFileSize",
        #u"NSFileSystemFileNumber",
        u"NSFileSystemNumber",
105        u"NSFileType",
        # u"NSFileHFSCreatorCode",
        # u"NSFileHFSTypeCode",
        #u"NSFileCreationDate"
    ] )
110    return mtprops

def setFileProperties( theFile, props ):
    sfm = NSFileManager defaultManager()
    return sfm.changeFileAttributes_atPath_( props, theFile )
115
def datestring_nsdate( dt=datetime.datetime.now() ):
    now = str(dt)
    now = now[:19]
    now = now + " +0000"
120    return now

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

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

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

140    n = startindex
    while True:
        serialstring = str(n).rjust(nfill, "0")

```

```

        filename = filenamebase + sep + serialstring + ext
145
        fullpath = os.path.join(folder, filename)

        if n >= 10**nfill:
            nfill = nfill + 1
150

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

        n += 1
155
def gethashval( s ):
    m = hashlib.shal()
    size = len(s)

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

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

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

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

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

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

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

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

```



```

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

def saveAsDialog(path):
    panel = NSSavePanel.savePanel()
220
    if path:
        panel.setDirectory_( path )

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

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

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

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

def get_hexstring_for_type( typ_ ):
    """
    """
    s = struct.pack( "BBBB", typ_ )
    i = struct.unpack( ">I", s )
    return hex(i)
265

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

```

```

def get_flavor(s):
    """Return the 4-char type from a pasteboard name
    """
275
    # seems like the standart naming scheme for the pasteboard server
    re_pbtype = re.compile( u"CorePasteboardFlavorType 0x([A-F0-9]{,8})" )

    m = re_pbtype.match(s)
280
    result = ""
    if m:
        t = m.groups()[0]
        result = get_type_from_hexstring(t)
    return result
285
def writePasteboardFlavour( folder, basename, ext, data ):
    p = uniquepath(folder, basename, ext)
    if data:
        f = open ( p, 'wb' )
290
        f.write( data )
        f.close()

    # fmpa 18
    # XMVL - 0x584D564C - Value Lists
295 # public.utf16-plain-text - Custom Menu Set Catalogue
    # public.utf16-plain-text - Custom Menu Catalogue

    # fmpa 15
    # XML2 - 0x584D4C32 - generic xml for layout objects
300
    # FMPA 11
    # XMFN - 0x584D464E - Custom Functions

    # FileMaker Advanced Pasteboard types
305 # XMFD - 0x584D4644 - fields
    # XMTB - 0x584D5442 - basetables
    # XMSC - 0x584D5343 - scripts
    # XMSS - 0x584D5353 - script step
    # XML0 - 0x584D4C4F - layout objects
310
    # FileMaker Developer Pasteboard types
    # beides binaerformate
    # FTR5 - 0x46545235 -
    # FMP5
315
    class PasteboardType(object):
        canonicalTypes = {
            u'com.adobe.pdf': u'Apple PDF pasteboard type',
            u'public.jpeg': u"CorePasteboardFlavorType 0x4A504547",
320
            u'NeXT TIFF v4.0 pasteboard type': u'public.tiff',
            # XML2
            u'dyn.ah62d4rv4gk8zuxnqgk': u"CorePasteboardFlavorType 0x584D4C32",

        }
325

    def __init__(self, pbname, typ, dataType, name, fileExt):
        self.pbname = pbname
        self.typ = typ
        self.dataType = dataType
330
        self.name = name
        self.fileExt = fileExt
        self.canonicalType = self.canonicalTypes.get( pbname, pbname )

    def __repr__(self):

```

```

335         return u"PasteboardType(%s, %s, %s, %s, %s, %s)" % (
            repr(self.pbname),
            repr(self.typ),
            repr(self.dataType),
            repr(self.name),
340            repr(self.fileExt),
            repr(self.canonicalType),)

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

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

fmpPasteboardTypes = {

    u"CorePasteboardFlavorType 0x584D4C32":
360        PasteboardType(u"CorePasteboardFlavorType 0x584D4C32",
            'XML2', 'fullXML', "Layout Objects", '.xml'),

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

    u"CorePasteboardFlavorType 0x584D4644":
        PasteboardType(u"CorePasteboardFlavorType 0x584D4644",
            'XMGD', 'snippetXML', "Fields", '.xml'),
370    u"CorePasteboardFlavorType 0x584D5343":
        PasteboardType(u"CorePasteboardFlavorType 0x584D5343",
            'XMSC', 'snippetXML', "Scripts", '.xml'),

375    u"CorePasteboardFlavorType 0x584D5353":
        PasteboardType(u"CorePasteboardFlavorType 0x584D5353",
            'XMSS', 'snippetXML', "Script Steps", '.xml'),

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

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

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

additionalFMPPasteboardTypes = {
395    u"CorePasteboardFlavorType 0x4A504547":
        PasteboardType(u"CorePasteboardFlavorType 0x4A504547",
            'JPEG', 'binaryData',
            "Layout Objects JPEG Image", '.jpg'),

```

```

400     u'Apple PDF pasteboard type':
        PasteboardType(u'Apple PDF pasteboard type',
                        'PDF', 'binaryData',
                        "Layout Objects PDF Image", '.pdf'),

405     u'com.adobe.pdf':
        PasteboardType(u'com.adobe.pdf',
                        'PDF', 'binaryData',
                        "Layout Objects PDF Image", '.pdf'),

410     u'Apple PICT pasteboard type':
        PasteboardType(u'Apple PICT pasteboard type',
                        'PICT', 'binaryData',
                        "Layout Objects PICT Image (obsolete)", '.pict'),

415     u'NeXT TIFF v4.0 pasteboard type':
        PasteboardType(u'NeXT TIFF v4.0 pasteboard type',
                        'TIFF', 'binaryData',
                        "Layout Objects TIFF Image", '.tif'),

420     u'public.jpeg':
        PasteboardType(u'public.jpeg',
                        'JPEG', 'binaryData',
                        "Layout Objects JPEG Image", '.jpg'),

425     u'public.tiff':
        PasteboardType(u'public.tiff',
                        'TIFF', 'binaryData',
                        "Layout Objects TIFF Image", '.tif'),
}

430 def read_pb():
    result = None
    hashes = set()

435     additional = []
    pasteboard = NSPasteboard.generalPasteboard()
    pbTypeNames = pasteboard.types()

    # additionalFMPPasteboardTypes

440     for pbTypeName in pbTypeNames:
        if 1:
            print( "pbTypeName:", pbTypeName )

445         pbType = mainType = None

        if pbTypeName in fmpPasteboardTypes:
            pbType = fmpPasteboardTypes.get( pbTypeName )
            mainType = True

450         elif pbTypeName in additionalFMPPasteboardTypes:
            pbType = additionalFMPPasteboardTypes.get( pbTypeName )
            mainType = False

        if pbType == None:
455             continue

        try:

            # pdb.set_trace()

460             s = pasteboard.dataForType_( pbTypeName )
            data = s.bytes().tobytes()

```

```

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

470         if mainType:
            data = makeunicode(data)

        pbTypeName = pbType.canonicalType

475         pbEntry = PasteboardEntry(pbTypeName, data, pbType)

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

        except Exception as v:
            print( v )
            # pdb.set_trace()
            pp(locals())
            print()

485         if result:
            result.additional = additional

490         if 1:
            print()
            print( "result = " )
            pp(result)
            print()
495         return result

```

FMPasteBoxVersion.py

```

import os

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

years = "2018-2022"
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'],
        }
    } )

```