FMPasteBox.py

```
#
#  FMPasteBox.py
#  FMPasteBox
#

import objc
import Foundation
import AppKit

from PyObjCTools import AppHelper

import FMPasteBoxAppDelegate

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

FMPasteBoxAppDelegate.py

```
#
#  FMPasteBoxAppDelegate.py
#  FMPasteBox
#

import sys
import os

import pprint
pp = pprint.pprint

import pdb
kwlog = True

import objc

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

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

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

import FMPasteBoxVersion

import FMPasteBoxPrefController
PrefController = FMPasteBoxPrefController.FMPasteBoxPreferenceController

class FMPasteBoxAppDelegate(NSObject):
```

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

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

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

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

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

    @objc.IBAction
    def getClipboard_(self, sender):
        pasteboardContents = read_pb()
        if not pasteboardContents:
            # abort - nothing on pasteboard
            NSBeep()
            # we must return implicit None! Crashing otherwise.
            return
        pbType = pasteboardContents.typ
        pbTypeName = pbType.name
        self.menClipboardtype.setTitle_( pbTypeName )
        self.tfXMLEditor.setString_( makeunicode( pasteboardContents.data ) )
        window = self.tfXMLEditor.window()
        window.makeFirstResponder_(self.tfXMLEditor)

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

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

```
            # get pasteboard type
            pasteboardType = displaynameTypes.get( self.menClipboardtype.title(), u"" )
            if not pasteboardType:
                NSBeep()
                # we must return implicit None! Crashing otherwise.
                return
            # write to pasteboard
            pasteboard = NSPasteboard.generalPasteboard()
            pasteboard.clearContents()
            pasteboardTypeName = pasteboardType.pbname
            pasteboard.setData_forType_( nsdata, pasteboardTypeName)

    @objc.IBAction
    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.
#   Copyright 2018 Karsten Wolf. All rights reserved.
#

import objc

import Foundation
NSUserDefaults = Foundation.NSUserDefaults

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

import FMPasteBoxTools

class FMPasteBoxPreferenceController (NSWindowController):

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

    cbDoExports = objc.IBOutlet()

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

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

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

        defaults = NSUserDefaults.standardUserDefaults()
        self.txtFileMakerAppPath.setStringValue_( defaults.objectForKey_( u'txtFileMakerAppPath') )
        return self

    def windowWillClose_(self, notification):
        defaults = NSUserDefaults.standardUserDefaults()
        defaults.setObject_forKey_(self.txtFileMakerAppPath.stringValue(),   u'txtFileMakerAppPath')
```

```
    @objc.IBAction
    def chooseFolder_(self, sender):
        if sender == self.butSetFileMakerAppPath:
            folders = FMPasteBoxTools.getApplicationDialog()
            if folders:
                self.txtFileMakerAppPath.setStringValue_( folders )
```

FMPasteBoxTools.py

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

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

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

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

import mactypes
import appscript
asc = appscript

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

import pprint
pp = pprint.pprint

import urllib
import urlparse

import objc

import Foundation
NSURL = Foundation.NSURL
NSFileManager = Foundation.NSFileManager
NSUserDefaults = Foundation.NSUserDefaults
NSString = Foundation.NSString

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

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

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

def makeunicode(s, srcencoding="utf-8", normalizer="NFC"):
    try:
        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"
    s = unicodedata.normalize(normalizer, s)
    return s

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

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

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

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

def setFileModificationDate( filepath, modfdt ):
    l = getFileProperties( filepath )
    date = Foundation.NSDate.dateWithString_( datestring_nsdate( modfdt ) )
    l['NSFileModificationDate'] = date
    setFileProperties( filepath, l)
    folder, filename = os.path.split( filepath )
    print "Setting file(%s) 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.sha1()
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 get_type_from_hexstring( hexstring ):
230     """Extract the 4-char macroman type code from the pasteboard type name.

        """
        h = int(hexstring, 16)
        s = struct.pack(">I", h)
235     s = unicode(s, 'macroman')
        return s

    def get_hexstring_for_type( typ_ ):
        """
240     """
        s = struct.pack( "BBBB", typ_ )
        i = struct.unpack( ">I", s)
        return hex(i)

245 def get_type_from_intstring( intstring ):
        h = int(intstring)
```

```
        s = struct.pack(">I", h)
        s = unicode(s, 'macroman')
        return s

    def get_flavor(s):
        """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})")

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

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

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

    # FMPA 11
    # XMFN - 0x584D464E - Custom Functions

    # FileMaker Advanced Pasteboard types
    # XMFD - 0x584D4644 - fields
    # XMTB - 0x584D5442 - basetables
    # XMSC - 0x584D5343 - scripts
    # XMSS - 0x584D5353 - script step
    # XML0 - 0x584D4C4F - layout objects

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

    class PasteboardType(object):
        def __init__(self, pbname, typ, dataType, name, fileExt):
            self.pbname = pbname
            self.typ = typ
            self.dataType = dataType
            self.name = name
            self.fileExt = fileExt
            self.alternates = []

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

    class PasteboardEntry(object):
        def __init__(self, name, data, typ):
            self.name = name
            self.data = data
```

```
            self.typ = typ

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

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

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

        u"CorePasteboardFlavorType 0x584D4644":
            PasteboardType(u"CorePasteboardFlavorType 0x584D4644",
330                        'XMFD', 'snippetXML', "Fields", '.xml'),

        u"CorePasteboardFlavorType 0x584D5343":
            PasteboardType(u"CorePasteboardFlavorType 0x584D5343",
                           'XMSC', 'snippetXML', "Scripts", '.xml'),
335
        u"CorePasteboardFlavorType 0x584D5353":
            PasteboardType(u"CorePasteboardFlavorType 0x584D5353",
                           'XMSS', 'snippetXML', "Script Steps", '.xml'),

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

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

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

355 additionalFMPPasteboardTypes = {
        u"CorePasteboardFlavorType 0x4A504547":
            PasteboardType(u"CorePasteboardFlavorType 0x4A504547",
                           'JPEG', 'binaryData',
                           "Layout Objects JPEG Image", '.jpg'),
360
        u'Apple PDF pasteboard type':
            PasteboardType(u'Apple PDF pasteboard type',
                           'PDF', 'binaryData',
                           "Layout Objects PDF Image", '.pdf'),
365
        u'com.adobe.pdf':
            PasteboardType(u'com.adobe.pdf',
                           'PDF', 'binaryData',
                           "Layout Objects PDF Image", '.pdf'),
370
        u'Apple PICT pasteboard type':
            PasteboardType(u'Apple PICT pasteboard type',
                           'PICT', 'binaryData',
                           "Layout Objects PICT Image (obsolete)", '.pic'),
```

```
375
        u’NeXT TIFF v4.0 pasteboard type’:
            PasteboardType(u’NeXT TIFF v4.0 pasteboard type’,
                           ’TIFF’, ’binaryData’,
                           "Layout Objects TIFF Image", ’.tif’),
380
        u’public.jpeg’:
            PasteboardType(u’public.jpeg’,
                           ’JPEG’, ’binaryData’,
                           "Layout Objects JPEG Image", ’.jpg’),
385
        u’public.tiff’:
            PasteboardType(u’public.tiff’,
                           ’TIFF’, ’binaryData’,
                           "Layout Objects TIFF Image", ’.tif’),
390 }

    def read_pb():
        result = None
        hashes = set()
395
        pasteboard = NSPasteboard.generalPasteboard()
        pbTypeNames = pasteboard.types()

        # additionalFMPPasteboardTypes
400
        for pbTypeName in pbTypeNames:

            pbType = None
            if pbTypeName in fmpPasteboardTypes:
405             pbType = fmpPasteboardTypes.get( pbTypeName, pbTypeName )
                maintype = True
            else:
                continue
                # NOT NOW
410             #if pbTypeName in additionalFMPPasteboardTypes:
                #    pbType = additionalFMPPasteboardTypes.get( pbTypeName )
                #    maintype = False

            if pbTypeName == None:
415             continue

            try:
                s = pasteboard.dataForType_( pbTypeName )
                data = s.bytes().tobytes()
420
                # dont load duplicate data
                hashval, _ = gethashval( data )
                if hashval in hashes:
                    continue
425             hashes.add( hashval )

                data = makeunicode(data)

                result = PasteboardEntry(pbTypeName, data, pbType)
430             return result

            except Exception, v:
                print v
                # pdb.set_trace()
435             pp(locals())
                print
        return result
```

FMPasteBoxVersion.py

```
import os

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

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

version = "0.1.1"
creator = 'KWFP'
bundleID = "org.kw.FMPasteBox"

description = (u"Filemaker Pasteboard interface and editor")
longdescription = u"""FMPasteBox is a Mac OS X application..."""

#
#user_agent = "%s/%s +https://github.com/karstenw/FMPasteBox" % (appname, version)

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

#cachefolder = os.path.expanduser("~/Library/Application Support/%s" % appname )

developmentversion = False
```

setup.py

```
"""
Script for building FMPasteBox

Usage:
    python setup.py py2app
"""
from distutils.core import setup
from setuptools.extension import Extension

import py2app

import FMPasteBoxVersion

setup(
    name = FMPasteBoxVersion.appname,
    version = FMPasteBoxVersion.version,
    description = FMPasteBoxVersion.description,
    long_description = FMPasteBoxVersion.longdescription,
    author = FMPasteBoxVersion.author,
    app=[{
        'script': "FMPasteBox.py",

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

    data_files=[
```

```
        "English.lproj/MainMenu.nib",
        "English.lproj/Preferences.nib",
        #"English.lproj/FMPasteBoxDocument.nib",
        "+icon/FMPasteBox.icns",
        #"+icon/FMPasteBoxFile.icns",
        ],

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