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
datetimestamp = FMPasteBoxTools.datetimestamp

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')
        userdefaults.setObject_forKey_(u"", u'txtExportsPath')
        userdefaults.setObject_forKey_(False, u'cbDoExports')
        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
        defaults = NSUserDefaults.standardUserDefaults()
        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 )
                sessionFolder = os.path.join( dayFolder, t)
                try:
                    exportItems = pasteboardContents.additionals[:]
                    exportItems.append( pasteboardContents )
                    for item in exportItems:
                        if not os.path.exists( sessionFolder ):
                            os.makedirs( sessionFolder )
                        name = item.typ.name
                        ext = item.typ.fileExt
                        data = item.data
                        path = os.path.join( sessionFolder, name + ext )
                        if ext == ".xml":
                            data = makeunicode( data )
                            data = data.encode( "utf-8" )
                        f = open(path, ’w’)
                        f.write( data )
                        f.close()
                except Exception, err:
                    print
                    print "ADDITIONALS FAILED"
                    print err
                    print
        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') )
        self.txtExportsPath.setStringValue_( defaults.objectForKey_( u'txtExportsPath') )
        self.cbDoExports.setState_( defaults.objectForKey_( u'cbDoExports') )
        return self

    def windowWillClose_(self, notification):
        defaults = NSUserDefaults.standardUserDefaults()
        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
    def chooseFolder_(self, sender):
        if sender == self.butSetFileMakerAppPath:
            folders = FMPasteBoxTools.getApplicationDialog()
            if folders:
                self.txtFileMakerAppPath.setStringValue_( folders )
        elif sender == self.butSetExportsPath:
            folders = FMPasteBoxTools.getFolderDialog()
            if folders:
                self.txtExportsPath.setStringValue_( folders[0] )
```

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
```

```python
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):
    """
    """
    folder = os.path.abspath( folder )

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

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

        filename = filenamebase + sep + serialstring + ext

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

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

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

        n += 1
```

```
def gethashval( s ):
    m = hashlib.sha1()
    size = len(s)

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

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

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

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

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

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

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

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

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

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

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

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

def datetimestamp( dt=None ):
    # '2018-02-17 19:41:02'
    if not dt:
        dt = datetime.datetime.now()
    now = str(dt)
    now = now[:19]
    d, t = now.split()
    t = t.replace(':', '')
    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)
    s = struct.pack(">I", h)
    s = unicode(s, 'macroman')
    return s

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

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
# XMLO - 0x584D4C4F - layout objects

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

class PasteboardType(object):
    canonicalTypes = {
        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",

    }

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    additionals = []
    pasteboard = NSPasteboard.generalPasteboard()
    pbTypeNames = pasteboard.types()

    # additionalFMPPasteboardTypes

    for pbTypeName in pbTypeNames:

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

        if pbType == None:
            continue

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

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

            if mainType:
                data = makeunicode(data)

            pbTypeName = pbType.canonicalType

            pbEntry = PasteboardEntry(pbTypeName, data, pbType)

            if mainType:
                result = pbEntry
            else:
                additionals.append( pbEntry )

        except Exception, v:
            print v
            pdb.set_trace()
            pp(locals())
            print
    if result:
        result.additionals = additionals

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

FMPasteBoxVersion.py

```
import os

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

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

version = "0.2.0"
creator = 'KWFP'
bundleID = "org.kw.FMPasteBox"

description = (u"Filemaker Pasteboard interface and editor")
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:
    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'],
        }
    } )
```