nodebox/Config.py

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

import sys
import os

class Config(object):
    """Just a class to carry all the options.

    Defaults to command line usage.
    """
    def __init__(self):
        self.accounts = True
        self.assets = True
        self.basetables = True
        self.customfunctions = True
        self.custommenus = True
        self.custommenusets = True
        self.privileges = True
        self.extendedprivileges = True
        self.filereferences = True

        self.layouts = True
        self.layoutGroups = True
        self.layoutOrder = True

        self.authfile = True
        self.externaldatasources = True
        self.themecatalog = True
        self.basedirectory = True

        self.relationships = True

        self.scripts = True
        self.scriptGroups = True
        self.scriptOrder = True

        self.valueLists = True

        self.summaryfile = ""
        self.exportfolder = ""

        self.ignoreFilenameIDs = False
        self.logfunction = None

        self.OPMLExport = False
        self.OPMLSplit = False
        self.OPMLDetails = False

    def pp(self):
        print "accounts", repr(self.accounts)
        print "assets", repr(self.assets)
        print "basetables", repr(self.basetables)
        print "customfunctions", repr(self.customfunctions)
        print "custommenus", repr(self.custommenus)
        print "custommenusets", repr(self.custommenusets)
        print "privileges", repr(self.privileges)
        print "extendedprivileges", repr(self.extendedprivileges)
        print "filereferences", repr(self.filereferences)
        print "layouts", repr(self.layouts)
        print "layoutGroups", repr(self.layoutGroups)
        print "layoutOrder", repr(self.layoutOrder)
```

```
        print "relationships", repr(self.relationships)
65
        print "scripts", repr(self.scripts)
        print "scriptGroups", repr(self.scriptGroups)
        print "scriptOrder", repr(self.scriptOrder)

70      print "valueLists", repr(self.valueLists)

        print "authfile", repr(self.authfile)
        print "externaldatasources", repr(self.externaldatasources)
        print "themecatalog", repr(self.themecatalog)
75      print "basedirectory", repr(self.basedirectory)

        print "summaryfile", repr(self.summaryfile)
        print "exportfolder", repr(self.exportfolder)
        print "ignoreFilenameIDs", repr(self.ignoreFilenameIDs)
80      print "logfunction", repr(self.logfunction)
```

nodebox/ddrsplit.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
 5 import os

import unicodedata
import time
import binascii
10 import base64
import hashlib

import pprint
pp = pprint.pprint
15
import pdb

import xml.etree.cElementTree
ElementTree = xml.etree.cElementTree
20
import xml.parsers.expat

import Config

25 import ReferenceCollector

#
# globals
#
30 gCancel = False
gREF = ReferenceCollector.ReferenceCollector()

#
# tools
35 #
def makeunicode(s, srcencoding="utf-8", normalizer="NFC"):
    if type(s) != unicode:
        s = unicode(s, srcencoding)
    s = unicodedata.normalize(normalizer, s)
40  return s

def stringhash( s ):
```

```
        m = hashlib.sha1()
        m.update(s)
        return m.hexdigest().upper()

    def logfunction(s):
        s = s + u"\n"
        sys.stdout.write(s.encode("utf-8"))

    #
    # parsers
    #
    def xmlexportfolder(basefolder, dbname, category, obname, obid="", ext=".xml"):
        # create or get folder where to put layout, script or basetable xml
        path = os.path.abspath(basefolder)

        catfolder = os.path.join( path, dbname, category)

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

        if obid:
            obid = str(obid).rjust(7,"0") + " "
        filename = obid + obname + ext
        filename = filename.replace('/', '_')
        filename = filename.replace(':', '_')
        filename = filename.replace('\\', '_')

        fullpath = os.path.join( catfolder, filename)
        fullpath = makeunicode( fullpath, normalizer="NFD" )
        return fullpath.encode("utf-8")

    def get_text_object(cfg, cur_fmpxml, cur_db, cur_fmpbasename, cur_node, cur_object):
        # check for global variables and merge fields
        pass

    def get_script_step(cfg, cur_fmpxml, cur_db, cur_fmpbasename, cur_node, cur_object):

        # BAUSTELLE

        # check for scripstep and field parameters
        # catch
        #       perform scrip
        #       exit script (parameter)
        #       set variable
        #       install on  timer script
        #       go to layout
        #       go to related record
        #       go to object
        #       go to field
        #       enter find mode
        step_id = cur_node.attrib.get("id", -1)
        step_name = cur_node.attrib.get("name", "NO SCRIPTSTEP NAME")

        fref_id = fref_name = step_calc_text = None

        for subnode in cur_node.iter():
            if subnode.tag == "FileReference":
                fref_id = subnode.attrib.get("id", -1)
                fref_name = subnode.attrib.get("name", "NO FILEREFERENCE NAME")

            elif subnode.tag == "DisplayCalculation":
                get_displaycalculation(cfg, cur_fmpxml, cur_db, cur_fmpbasename, subnode)
            elif subnode.tag == "Calculation":
                step_calc_text = subnode.text
```

```
        external = fref_id and fref_name

110 def get_displaycalculation(cfg, cur_fmpxml, cur_db, cur_fmpbasename, cur_node):
        clc_text = clc_noref = clc_fnctref = clc_fieldref = clc_cf = ""

        for node in cur_node.iter():
            dpc_tag = node.tag
115         dpc_typ = node.attrib.get( "type", "")
            if dpc_typ == "NoRef":
                clc_noref = node.text
            elif dpc_typ == "FunctionRef":
                pass
120         elif dpc_typ == "FieldRef":
                # <Field id="1" name="F1" table="to_Test1" />
                pass
            elif dpc_typ == "CustomFunctionRef":
                pass
125
    def get_authfilecatalog(cfg, cur_fmpxml, cur_db, cur_fmpbasename, authfiles,
                            groups, exportfolder, idx):

        for authfile in authfiles:
130         authfile_attr = authfile.attrib
            authfile_tag = authfile.tag
            authfile_name = authfile_attr.get("name", "NONAME")

            cur_object = (cur_fmpxml, 'AuthFile', authfile_name)
135
            path = "AuthFiles"

            s = ElementTree.tostring(authfile, encoding="utf-8", method="xml")

140         sortid = authfile_attr.get("id", "0").rjust(7,"0")

            objectID = sortid
            if cfg.ignoreFilenameIDs:
                objectID = ""
145         path = xmlexportfolder(exportfolder,
                                   cur_fmpbasename,
                                   path,
                                   authfile_name,
                                   objectID)
150         f = open(path, "wb")
            f.write( s )
            f.close()
            idx += 1
        return idx
155
    def get_externaldatasources(cfg, cur_fmpxml, cur_db, cur_fmpbasename, externaldatasources,
                                groups, exportfolder, idx):

        for externaldatasource in externaldatasources:
160         externaldatasource_attr = externaldatasource.attrib
            externaldatasource_tag = externaldatasource.tag
            externaldatasource_name = externaldatasource_attr.get("name", "NONAME")

            cur_object = (cur_fmpxml, 'ExternalDataSource', externaldatasource_name)
165
            path = "ExternalDataSources"

            s = ElementTree.tostring(externaldatasource, encoding="utf-8", method="xml")

170         sortid = externaldatasource_attr.get("id", "0").rjust(7,"0")
```

```
            objectID = sortid
            if cfg.ignoreFilenameIDs:
                objectID = ""
        path = xmlexportfolder(exportfolder,
                               cur_fmpbasename,
                               path,
                               externaldatasource_name,
                               objectID)
        f = open(path, "wb")
        f.write( s )
        f.close()
        idx += 1
    return idx

def get_themecatalog(cfg, cur_fmpxml, cur_db, cur_fmpbasename, themes,
                         groups, exportfolder, idx):

    for theme in themes:
        theme_attr = theme.attrib
        theme_tag = theme.tag
        theme_name = theme_attr.get("name", "NONAME")

        cur_object = (cur_fmpxml, 'ThemeCatalog', theme_name)

        path = "Themes"

        s = ElementTree.tostring(theme, encoding="utf-8", method="xml")

        sortid = theme_attr.get("id", "0").rjust(7,"0")

        objectID = sortid
        if cfg.ignoreFilenameIDs:
            objectID = ""
        path = xmlexportfolder(exportfolder,
                               cur_fmpbasename,
                               path,
                               theme_name,
                               objectID)
        f = open(path, "wb")
        f.write( s )
        f.close()
        idx += 1
    return idx

def get_basedirectories(cfg, cur_fmpxml, cur_db, cur_fmpbasename, basedirectories,
                         groups, exportfolder, idx):

    for basedirectory in basedirectories:
        basedirectory_attr = basedirectory.attrib
        basedirectory_tag = basedirectory.tag
        basedirectory_name = basedirectory_attr.get("name", "NONAME")

        cur_object = (cur_fmpxml, 'BaseDirectoryCatalog', basedirectory_name)

        path = "BaseDirectoryCatalog"

        s = ElementTree.tostring(basedirectory, encoding="utf-8", method="xml")

        sortid = basedirectory_attr.get("id", "0").rjust(7,"0")

        objectID = sortid
        if cfg.ignoreFilenameIDs:
            objectID = ""
```

```
        path = xmlexportfolder(exportfolder,
                               cur_fmpbasename,
                               path,
                               basedirectory_name,
                               objectID)
        f = open(path, "wb")
        f.write( s )
        f.close()
        idx += 1
    return idx

def get_layouts_and_groups(cfg, cur_fmpxml, cur_db, cur_fmpbasename, laynode,
                           groups, exportfolder, idx):

    for layout in laynode:
        layout_attr = layout.attrib
        layout_tag = layout.tag
        layout_name = layout_attr.get("name", "NONAME")

        cur_object = (cur_fmpxml, 'Layout', layout_name)

        if layout_tag == "Group":
            grp_attrib = layout_attr
            groupid = layout_attr.get("id", "0")

            # get layout folder name
            groupname = (  groupid.rjust(7,"0")
                         + ' '
                         + layout_name )

            if cfg.layoutOrder:
                groupname = (  str(idx).rjust(5,"0")
                             + ' '
                             + groupid.rjust(7,"0")
                             + ' '
                             + layout_name )

            if cfg.ignoreFilenameIDs:
                groupname = layout_name

            groups.append( groupname )

            idx += 1
            idx = get_layouts_and_groups(cfg, cur_fmpxml, cur_db, cur_fmpbasename, layout,
                                         groups, exportfolder, idx)
            groups.pop()
        else:
            path = "Layouts"

            if groups and cfg.layoutGroups:
                path = os.path.join("Layouts", *groups)
            s = ElementTree.tostring(layout, encoding="utf-8", method="xml")

            sortid = layout_attr.get("id", "0").rjust(7,"0")
            if cfg.layoutOrder:
                sortid = (str(idx).rjust(5,"0")
                          + ' '
                          + layout_attr.get("id", "0").rjust(7,"0") )

            objectID = sortid
            if cfg.ignoreFilenameIDs:
                objectID = ""
            path = xmlexportfolder(exportfolder,
                                   cur_fmpbasename,
```

```
                                    path,
                                    layout_name,
                                    objectID)
            f = open(path, "wb")
            f.write( s )
            f.close()
            idx += 1

            if not cfg.assets:
                continue

            for l in layout.getchildren():
                t = l.tag
                if t == u'Object':
                    get_layout_object(cfg, cur_fmpxml, cur_db, cur_fmpbasename,
                                      l, cur_object, exportfolder)
    return idx

def get_layout_object(cfg, cur_fmpxml, cur_db, cur_fmpbasename, laynode,
                      cur_object, exportfolder):
    nodes = list(laynode)
    extensions = dict(zip( ("JPEG","PDF ", "PNGf", "PICT",
                            "GIFf", "8BPS", "BMPf"),
                           (".jpg",".pdf", ".png", ".pict",
                            ".gif", ".psd", ".bmp")))
    exttypelist = extensions.keys()

    cur_tableOccurrenceName = cur_tableOccurrenceID = ""

    for node in nodes:
        cur_tag = node.tag

        if cur_tag == u'Object':
            # get layout object
            get_layout_object(cfg, cur_fmpxml, cur_db, cur_fmpbasename, node,
                              cur_object, exportfolder)

        elif cur_tag == u'ObjectStyle':
            continue

        elif cur_tag == u'Table':
            # <Table id="13631489" name="to_Bildarchiv" />
            cur_tableOccurrenceID = node.get("id", -1)
            cur_tableOccurrenceName = node.get("name",
                                       "NO TABLE OCCURRENCE NAME FOR LAYOUT")
            cur_objectID = gREF.addObject( cur_object )
            gREF.addFilemakerAttribute( cur_objectID, "tableOccurrenceID",
                                                      cur_tableOccurrenceID)
            gREF.addFilemakerAttribute( cur_objectID, "tableOccurrenceName",
                                                      cur_tableOccurrenceName)

        elif cur_tag == u'GraphicObj':
            for grobnode in node:
                if grobnode.tag == "Stream":
                    stype = []
                    sdata = ""
                    for streamnode in grobnode:
                        streamtag = streamnode.tag
                        streamtext = streamnode.text
                        if streamtag == "Type":
                            if streamtext not in exttypelist:
                                stype.append( '.' + streamtext )
                            else:
                                stype.append( streamtext )
```

```
                    elif streamtag in ("Data", "HexData"):
                        if not stype:
                            continue
                        curtype = stype[-1]
                        ext = extensions.get( curtype, False )
                        if not ext:
                            ext = curtype
                        data = None
                        if streamtag == "HexData":
                            try:
                                data = binascii.unhexlify ( streamtext )
                            except TypeError, err:
                                pass
                        elif streamtag == "Data":
                            try:
                                data = base64.b64decode( streamtext )
                            except TypeError, err:
                                pass
                        if not data:
                            continue

                        fn = stringhash( data )
                        path = xmlexportfolder(exportfolder,
                                               cur_fmpbasename,
                                               "Assets",
                                               fn,
                                               "",
                                               ext)
                        # write Asset file
                        if not os.path.exists( path ):
                            f = open(path, "wb")
                            f.write( data )
                            f.close()

    # the following tags are for reference collection only

    # laynode is current node
    # cur_object is ref1
    elif cur_tag == u'GroupButtonObj':
        # recurse
        get_layout_object(cfg, cur_fmpxml, cur_db, cur_fmpbasename,
                          node, cur_object, exportfolder)

    elif cur_tag == u'FieldObj':
        # check for scripstep and field parameters
        for subnode in node.iter():
            if subnode.tag == "Field":
                # <Field id="2" maxRepetition="1" name="F2"
                # repetition="1" table="Test1" />
                fld_id = int(subnode.attrib.get("id", -1))
                fld_name = subnode.attrib.get("name",
                                              "NO FIELD NAME")
                fld_to = subnode.attrib.get("table",
                                              "NO TABLE OCCURRENCE")

                fld_obj = (cur_object[0], "Field", fld_name, fld_to)
                fld_obj_id = gREF.addObject( fld_obj )
                gREF.addFilemakerAttribute(fld_obj_id, "id", fld_id)

                gREF.addReference(cur_object, fld_obj)

    elif cur_tag == u'Step':
        get_script_step(cfg, cur_fmpxml, cur_db, cur_fmpbasename,
                        node, cur_object)
```

```
            elif cur_tag == u'TextObj':
                get_text_object(cfg, cur_fmpxml, cur_db, cur_fmpbasename,
                                node, cur_object)

    def get_scripts_and_groups(cfg, cur_fmpxml, cur_db, cur_fmpbasename, scriptnode,
                               exportfolder, groups, namecache, idx):

        for scpt in scriptnode:
            if scpt.tag == "Script":
                path = "Scripts"
                if groups and cfg.scriptGroups:
                    path = os.path.join("Scripts", *groups)

                sortid = scpt.get("id", "0").rjust(7,"0")
                if cfg.scriptOrder:
                    sortid = (str(idx).rjust(5,"0")
                              + ' '
                              + scpt.get("id", "0").rjust(7,"0") )
                s = ElementTree.tostring(scpt, encoding="utf-8", method="xml")

                objectID = sortid
                if cfg.ignoreFilenameIDs:
                    objectID = ""
                path = xmlexportfolder(exportfolder, cur_fmpbasename, path,
                                       scpt.get("name", "NONAME"),
                                       objectID)
                idx += 1
                f = open(path, "wb")
                f.write( s )
                f.close()

            elif scpt.tag == "Group":
                grp_attrib = scpt.attrib
                groupid = grp_attrib.get("id", "0")

                # script folder name (if any)
                groupname = (groupid.rjust(7,"0")
                             + ' '
                             + grp_attrib.get("name", "No folder name") )

                if cfg.scriptOrder:
                    groupname = (str(idx).rjust(5,"0")
                                 + ' ' + groupid.rjust(7,"0")
                                 + ' ' + grp_attrib.get("name", "No folder name"))

                if cfg.ignoreFilenameIDs:
                    groupname = grp_attrib.get("name", "No folder name")
                groups.append( groupname )

                idx += 1
                idx = get_scripts_and_groups(cfg, cur_fmpxml, cur_db, cur_fmpbasename,
                                             scpt, exportfolder, groups, namecache, idx)
                groups.pop()
        return idx

    def get_relationshipgraph_catalog(cfg, cur_fmpxml, cur_db, cur_fmpbasename,
                                      rg_cat, exportfolder):
        for tablst in rg_cat:
            if tablst.tag == u'TableList':
                for tab in tablst:
                    if tab.tag == u'Table':

                        to_attr = tab.attrib
```

```
                    to_name = tab.attrib.get("name", "NO TABLE OCCURRENCE NAME")
                    to_id = tab.attrib.get("id", -1)
                    to_btid = tab.attrib.get("baseTableId", -1)
                    to_bt = tab.attrib.get("baseTable", "NO BASETABLE FOR TABLE OCCURRENCE")

                    s = ElementTree.tostring(tab, encoding="utf-8", method="xml")

                    objectID = to_id
                    if cfg.ignoreFilenameIDs:
                        objectID = ""
                    path = xmlexportfolder(exportfolder,
                                           cur_fmpbasename,
                                           "Relationships/TableList",
                                           to_name,
                                           objectID)
                    f = open(path, "wb")
                    f.write( s )
                    f.close()

                    external = eto_id = eto_name = False
                    for node in tab.iter():
                        if node.tag == "FileReference":
                            external = True
                            eto_id = node.get("id", -1)
                            eto_name = node.get("name",
                                        "NO EXTERNAL FILEREF NAME FOR TABLE OCCURRENCE")

                    toObject = (cur_fmpxml, "TableOccurrence", to_name)
                    if external:
                        toObject = (cur_fmpxml, "ExternalTableOccurrence", to_name)

                    toObjectId = gREF.addObject( toObject )
                    gREF.addFilemakerAttribute( toObjectId, 'baseTableId', to_btid)
                    gREF.addFilemakerAttribute( toObjectId, 'baseTable', to_bt)
                    if external:
                        gREF.addFilemakerAttribute( toObjectId, 'fileReferenceID', eto_id)
                        gREF.addFilemakerAttribute( toObjectId, 'fileReferenceName', eto_name)

        # <Table baseTable="bt_Bildarchiv" baseTableId="32769"
        #  color="#777777" id="13631489" name="to_Bildarchiv" />

        # <Table baseTable="bt_Text" baseTableId="32769" color="#777777"
        # id="13631498" name="eto_TEX_artnum">
        # <FileReference id="1" name="Text" />
        # </Table>

            elif tablst.tag == u'RelationshipList':
                for rel in tablst:
                    if rel.tag == u'Relationship':
                        rel_cat = {}
                        re_attr = rel.attrib
                        relid = re_attr.get("id", "0")
                        rel_cat['id'] = re_attr.get("id", "0")

                        for rel_component in rel.getchildren():
                            if rel_component.tag == "LeftTable":
                                rel_cat['lefttable'] = rel_component.attrib.get("name",
                                                                "NO-LEFTTABLENAME")
                            elif rel_component.tag == "RightTable":
                                rel_cat['righttable'] = rel_component.attrib.get("name",
                                                                "NO-LEFTTABLENAME")

                        s = ElementTree.tostring(rel, encoding="utf-8", method="xml")
                        filename = (rel_cat['lefttable']
```

```
                                + "---"
                                + rel_cat['righttable'])

                    objectID = rel_cat['id']
                    if cfg.ignoreFilenameIDs:
                        objectID = ""
                    path = xmlexportfolder(exportfolder,
                                           cur_fmpbasename,
                                           "Relationships/Relationship",
                                           filename,
                                           objectID)
                    f = open(path, "wb")
                    f.write( s )
                    f.close()

def main(cfg):

    xmlfile = cfg.summaryfile
    xml_folder, xmlfilename = os.path.split( xmlfile )

    ddr = ElementTree.parse( xmlfile )

    summary = ddr.getroot()

    files = summary.findall( "File" )
    nooffiles = len( files )

    filelist = {}

    starttime = time.time()

    log = logfunction
    if cfg.logfunction:
        log = cfg.logfunction

    for fmpreport in summary.getiterator("FMPReport"):
        for xmlfile in fmpreport.getiterator("File"):
            xml_fmpfilename = xmlfile.get("name", "NO FILE NAME")
            xml_xmllink = xmlfile.get("link", "")
            xml_fmppath = xmlfile.get("path", "")

            if not xml_xmllink:
                s = u"\nERROR: Could not find XML file '%s'\nContinue.\n"
                log( s %  xml_xmllink)
                continue

            # xml_xmllink
            # cleanup filename
            while xml_xmllink.startswith( './/' ): xml_xmllink = xml_xmllink[ 3: ]
            while xml_xmllink.startswith( './' ):  xml_xmllink = xml_xmllink[ 2: ]

            xmlbasename, ext  = os.path.splitext( xml_xmllink )

            filelist[ xml_xmllink ] = (xml_fmpfilename, xml_fmppath, xmlbasename)

    for cur_xml_file_name in filelist.keys():

        # path to DDR-XML file
        next_xml_file_path = os.path.join( xml_folder, cur_xml_file_name )

        # some UI glitz
        line = '-' * 100
        log( u"\n\n%s\n\nXMLFILE: %s" % (line, cur_xml_file_name) )
        print "filelist[ xml_xmllink ]:", repr(filelist[ cur_xml_file_name ])
```

```
        # parse xml file
        try:
            basenode = ElementTree.parse( next_xml_file_path )
        except  (xml.parsers.expat.ExpatError, SyntaxError), v:
            xml.parsers.expat.error()
            log( u"EXCEPTION: '%s'" % v )
            log( u"Failed parsing '%s'\n" % next_xml_file_path )
            continue

        # more often the xml filename is required for identification
        cur_db = filelist[ cur_xml_file_name ][0]
        cur_fmpbasename = filelist[ cur_xml_file_name ][2]
        cur_fmpxml = cur_xml_file_name

        cur_fileRef = (cur_fmpxml, "DatabaseFile", cur_db)

        exportfolder = cfg.exportfolder

        # relationships need to be analyzed first for the baseTable -> TO graph
        # for that to happen, filereferences must go before that

        print
        print

        #
        # FileReferenceCatalog
        #
        # todo check if refs && cfg.filereferences
        if 1: #cfg.filereferences:
            log( u'File References "%s"' % cur_fmpxml )
            for fr_cat in basenode.getiterator ( "FileReferenceCatalog" ):
                for fileref in fr_cat.getchildren():
                    fileref_attrib = fileref.attrib
                    prefix = ""
                    if fileref.tag == "OdbcDataSource":
                        prefix = "ODBC-"
                    elif fileref.tag == "FileReference":
                        prefix = "FREF-"
                        #
                        # <FileReference id="2" link="Menu_fp7.xml" name="Menu"
                        #                pathList="file:Menu.fp7" />
                        #
                        frf_id = fileref.attrib.get("id", -1)
                        frf_link = fileref.attrib.get("link", "NO DDR.XML FILE")
                        frf_name = fileref.attrib.get("name", "NO FILEREF NAME")
                        frf_pathList = fileref.attrib.get("pathList",
                                                          "NO FILEREF PATHLIST")
                        gREF.addFileReference(cur_xml_file_name, frf_link, frf_name,
                                                          frf_id, frf_pathList)

                        frf_object = (cur_xml_file_name, 'FileReference', frf_name)
                        gREF.addObject(frf_object)
                        gREF.addReference(cur_fileRef, frf_object)

                    else:
                        prefix = "UNKN-"
                    name = prefix + fileref_attrib.get("name", "NONAME")

                    s = ElementTree.tostring(fileref,
                                             encoding="utf-8",
                                             method="xml")

                    objectID = fileref_attrib.get("id", "0")
```

```
                    if cfg.ignoreFilenameIDs:
                        objectID = ""
                    path = xmlexportfolder(exportfolder,
                                           cur_fmpbasename,
                                           "Filereferences",
                                           name,
                                           objectID)
                    f = open(path, "wb")
                    f.write( s )
                    f.close()
                # collect references to fields, CFs, value lists,
                # merge fields, scripts, TOs, FileReferences

            #
            # relationship graph
            #
            if cfg.relationships:
                log( u'Relationship Graph "%s"' % cur_fmpxml )
                for rg_cat in basenode.getiterator ( "RelationshipGraph" ):
                    get_relationshipgraph_catalog(cfg, cur_fmpxml, cur_db,
                                                  cur_fmpbasename, rg_cat, exportfolder)
                # collect references from FRF to FRF

            #
            # base table catalog
            #
            if cfg.basetables:
                log( u'Base Tables "%s"' % cur_fmpxml )
                for base_table_catalog in basenode.getiterator( u'BaseTableCatalog' ):
                    for base_table in base_table_catalog.getiterator( u'BaseTable' ):
                        bt_name = base_table.get("name", "NONAME")
                        bt_id = base_table.get("id", "0")
                        s = ElementTree.tostring(base_table,
                                                 encoding="utf-8",
                                                 method="xml")

                        objectID = bt_id
                        if cfg.ignoreFilenameIDs:
                            objectID = ""
                        path = xmlexportfolder(exportfolder,
                                               cur_fmpbasename,
                                               "Basetables",
                                               bt_name,
                                               objectID)
                        f = open(path, "wb")
                        f.write( s )
                        f.close()

                        cur_btRef = (cur_fmpxml, "BaseTable", bt_name)
                        bt_objID = gREF.addObject( cur_btRef )

                        # make the basetable id known without using it for references
                        gREF.addFilemakerAttribute(bt_objID, "id", bt_id)

                        gREF.addReference( cur_fileRef, cur_btRef)

                        # TODO
                        #
                        # FIELDS
                        #
                        # cur_db, cur_btRef, bt_name, bt_id, bt_objID
                        for field_catalog in base_table.getiterator( u'FieldCatalog' ):
                            for field in field_catalog.getiterator( u'Field' ):
                                # dataType="Date"
```

```
                            # fieldType="Normal"
                            # id="9"
                            # name="dat_BAR_created"
                            fld_name = field.get("name", "NONAME")
                            fld_id = field.get("id", "0")
                            fld_type = field.get("fieldType", "NO FIELD TYPE")
                            fld_dataType = field.get("dataType", "NO DATA TYPE")

                            cur_fldRef = (cur_fmpxml, "Field", fld_name, bt_name)
                            fld_objID = gREF.addObject( cur_fldRef )

                            gREF.addFilemakerAttribute(fld_objID, "id", fld_id)
                            gREF.addFilemakerAttribute(fld_objID, "dataType",
                                                                  fld_dataType)
                            gREF.addFilemakerAttribute(fld_objID, "fieldType",
                                                                  fld_id)

                            gREF.addReference( cur_btRef, cur_fldRef)
                            # TODO
                            #
                            # add ref to TO (needs Calculations)

                # collect references to fields, CFs, value lists, TOs, FileReferences

            #
            # LayoutCatalog
            #
            if cfg.layouts:
                log( u'Layout Catalog "%s"' % cur_fmpxml )
                for layout_catalog in basenode.getiterator ( "LayoutCatalog" ):
                    groups = []
                    get_layouts_and_groups(cfg,
                                           cur_fmpxml,
                                           cur_db,
                                           cur_fmpbasename,
                                           layout_catalog,
                                           groups,
                                           exportfolder,
                                           1)
                    # collect references to fields, CFs, value lists, merge fields,
                    # scripts, TOs, FileReferences

            #
            # account catalog
            #
            if cfg.accounts:
                log( u'Accounts for "%s"' % cur_fmpxml )
                for acc_cat in basenode.getiterator ( "AccountCatalog" ):
                    for acc in acc_cat.getchildren():
                        acc_attrib = acc.attrib
                        s = ElementTree.tostring(acc, encoding="utf-8", method="xml")

                        objectID = acc_attrib.get("id", "0")
                        if cfg.ignoreFilenameIDs:
                            objectID = ""
                        path = xmlexportfolder(exportfolder,
                                               cur_fmpbasename,
                                               "Accounts",
                                               acc_attrib.get("name", "NONAME"),
                                               objectID)
                        f = open(path, "wb")
                        f.write( s )
                        f.close()
                    # collect references to fields, CFs, value lists, TOs, FileReferences
```

```
        #
        # script catalog
        #
        if cfg.scripts:
            log( u'Scripts for "%s"' % cur_fmpxml )
            for scpt_cat in basenode.getiterator ( "ScriptCatalog" ):
                groups = []
                namecache = [{},{}]
                get_scripts_and_groups(cfg,
                                       cur_fmpxml,
                                       cur_db,
                                       cur_fmpbasename,
                                       scpt_cat,
                                       exportfolder,
                                       groups,
                                       namecache,
                                       1)
            # collect references to fields, CFs, value lists, scripts,
            # TOs, FileReferences

        #
        # custom function catalog
        #
        #
        if cfg.customfunctions:
            log( u'Custom Functions for "%s"' % cur_fmpxml )
            for cf_cat in basenode.getiterator ( "CustomFunctionCatalog" ):
                groups = []
                for cf in cf_cat.getchildren():
                    cf_attrib = cf.attrib
                    s = ElementTree.tostring(cf, encoding="utf-8", method="xml")

                    objectID = cf_attrib.get("id", "0")
                    if cfg.ignoreFilenameIDs:
                        objectID = ""
                    path = xmlexportfolder(exportfolder,
                                           cur_fmpbasename,
                                           "CustomFunctions",
                                           cf_attrib.get("name", "NONAME"),
                                           objectID)
                    f = open(path, "wb")
                    f.write( s )
                    f.close()
            # collect references to fields, CFs, value lists,TOs, FileReferences

        #
        # PrivilegesCatalog
        #
        if cfg.privileges:
            log( u'Privileges for "%s"' % cur_fmpxml )
            for pv_cat in basenode.getiterator( "PrivilegesCatalog" ):
                for pv in pv_cat.getchildren():
                    pv_attrib = pv.attrib
                    s = ElementTree.tostring(pv, encoding="utf-8", method="xml")

                    objectID = pv_attrib.get("id", "0")
                    if cfg.ignoreFilenameIDs:
                        objectID = ""
                    path = xmlexportfolder(exportfolder,
                                           cur_fmpbasename,
                                           "Privileges",
                                           pv_attrib.get("name", "NONAME"),
                                           objectID)
```

```
                    f = open(path, "wb")
                    f.write( s )
                    f.close()
                # collect references to fields, CFs, value lists, TOs, FileReferences

            #
            # ExtendedPrivilegeCatalog
            #
            if cfg.extendedprivileges:
                log( u'Extended Privileges for "%s"' % cur_fmpxml )
                for epv_cat in basenode.getiterator( "ExtendedPrivilegeCatalog" ):
                    for epv in epv_cat.getchildren():
                        epv_attrib = epv.attrib
                        s = ElementTree.tostring(epv, encoding="utf-8", method="xml")

                        objectID = epv_attrib.get("id", "0")
                        if cfg.ignoreFilenameIDs:
                            objectID = ""
                        path = xmlexportfolder(exportfolder,
                                               cur_fmpbasename,
                                               "ExtendedPrivileges",
                                               epv_attrib.get("name", "NONAME"),
                                               objectID)
                        f = open(path, "wb")
                        f.write( s )
                        f.close()
                # collect references to fields, CFs, value lists, TOs, FileReferences

            #
            # AuthFileCatalog
            #
            if cfg.authfile:
                log( u'AuthFile Catalog "%s"' % cur_fmpxml )
                for authfile_catalog in basenode.getiterator ( "AuthFileCatalog" ):
                    groups = []
                    get_authfilecatalog(cfg,
                                        cur_fmpxml,
                                        cur_db,
                                        cur_fmpbasename,
                                        authfile_catalog,
                                        groups,
                                        exportfolder,
                                        1)

            #
            # ExternalDataSourcesCatalog
            #
            if cfg.externaldatasources:
                log( u'ExternalDataSources Catalog "%s"' % cur_fmpxml )
                for externaldatasource in basenode.getiterator ( "ExternalDataSourcesCatalog" ):
                    groups = []
                    get_externaldatasources(cfg,
                                        cur_fmpxml,
                                        cur_db,
                                        cur_fmpbasename,
                                        externaldatasource,
                                        groups,
                                        exportfolder,
                                        1)

            #
            # ThemeCatalog
            #
            if cfg.themecatalog:
```

```
            log( u'Theme Catalog "%s"' % cur_fmpxml )
            for theme in basenode.getiterator ( "ThemeCatalog" ):
                groups = []
                get_themecatalog(cfg,
                                 cur_fmpxml,
                                 cur_db,
                                 cur_fmpbasename,
                                 theme,
                                 groups,
                                 exportfolder,
                                 1)

        #
        # BaseDirectoryCatalog
        #
        if cfg.basedirectory:
            log( u'BaseDirectory Catalog "%s"' % cur_fmpxml )
            for basedir in basenode.getiterator ( "BaseDirectoryList" ):
                groups = []
                get_basedirectories(cfg,
                                 cur_fmpxml,
                                 cur_db,
                                 cur_fmpbasename,
                                 basedir,
                                 groups,
                                 exportfolder,
                                 1)

        #
        # CustomMenuCatalog
        #
        if cfg.custommenus:
            log( u'Custom Menus for "%s"' % cur_fmpxml )
            for cm_cat in basenode.getiterator( "CustomMenuCatalog" ):
                for cm in cm_cat.getchildren():
                    cm_attrib = cm.attrib
                    s = ElementTree.tostring(cm, encoding="utf-8", method="xml")

                    objectID = cm_attrib.get("id", "0")
                    if cfg.ignoreFilenameIDs:
                        objectID = ""
                    path = xmlexportfolder(exportfolder,
                                           cur_fmpbasename,
                                           "CustomMenus",
                                           cm_attrib.get("name", "NONAME"),
                                           objectID)
                    f = open(path, "wb")
                    f.write( s )
                    f.close()
            # collect references to fields, CFs, value lists, TOs, FileReferences

        #
        # CustomMenuSetCatalog
        #
        if cfg.custommenusets:
            log( u'Custom Menu Sets for "%s"' % cur_fmpxml )
            for cms_cat in basenode.getiterator( "CustomMenuSetCatalog" ):
                for cms in cms_cat.getchildren():
                    cms_attrib = cms.attrib
                    s = ElementTree.tostring(cms, encoding="utf-8", method="xml")

                    objectID = cms_attrib.get("id", "0")
                    if cfg.ignoreFilenameIDs:
                        objectID = ""
```

```
                    path = xmlexportfolder(exportfolder,
                                           cur_fmpbasename,
                                           "CustomMenuSets",
                                           cms_attrib.get("name", "NONAME"),
                                           objectID)
                    f = open(path, "wb")
                    f.write( s )
                    f.close()
                # collect references to fields, CFs, value lists, TOs, FileReferences

            #
            # ValueListCatalog
            #
            if cfg.valueLists:
                log('Value Lists for "%s"' % cur_fmpxml )
                for vl_cat in basenode.getiterator( "ValueListCatalog" ):
                    for vl in vl_cat.getchildren():
                        vl_attrib = vl.attrib
                        s = ElementTree.tostring(vl, encoding="utf-8", method="xml")

                        objectID = vl_attrib.get("id", "0")
                        if cfg.ignoreFilenameIDs:
                            objectID = ""
                        path = xmlexportfolder(exportfolder,
                                               cur_fmpbasename,
                                               "ValueLists",
                                               vl_attrib.get("name", "NONAME"),
                                               objectID)
                        f = open(path, "wb")
                        f.write( s )
                        f.close()
                # collect references to fields, CFs, value lists, TOs, FileReferences

            if gCancel:
                time.sleep(0.3)
                log("\n\n####  CANCELLED.  ####")
                return
            else:
                print
                print

        #
        # References
        #

        # objects

        path = xmlexportfolder(exportfolder,
                               "",
                               "References",
                               "id_object",
                               ext=".tab")
        f = open(path, "wb")
        s = u"%i\t%s\n"
        s = u"%i\t%s\t%s\t%s\t%s\t%s\n"
        keys = gREF.objectsReverse.keys()
        keys.sort()
        for key in keys:
            v = gREF.objectsReverse[ key ]
            s4 = s5 = u""
            s1, s2, s3 = v[:3]
            if len(v) == 4:
                s4 = v[3]
            elif len(v) == 5:
```

```
                s4 = v[3]
                s5 = v[4]

            t = s % (key, s1, s2, s3, s4, s5)
            f.write( t.encode("utf-8") )
        f.close()

        # filemakerAttributes
        path = xmlexportfolder(exportfolder,
                               "",
                               "References",
                               "objid_name_fmpattribute",
                               ext=".tab")
        f = open(path, "wb")
        s = u"%s\t%s\t%s\n"
        for objid in gREF.filemakerAttributes:
            obj = gREF.filemakerAttributes[objid]
            for name in obj:
                v = obj[name]
                t = s % (objid, name, v)
                f.write( t.encode("utf-8") )
        f.close()

        # references
        path = xmlexportfolder(exportfolder,
                               "",
                               "References",
                               "objid_objid_reference",
                               ext=".tab")
        f = open(path, "wb")
        s = u"%i\t%i\n"
        keys = gREF.references.keys()
        #keys.sort()
        for r1 in keys:
            referrers = gREF.references[r1]
            #referrers.sort()
            for r2 in referrers:
                t = s % (r1, r2)
                f.write( t.encode("utf-8") )
        f.close()

        time.sleep(0.3)
        stoptime = time.time()

        t = "\nRuntime %.4f\n\n####  FINISHED.  ####\n\n"
        log(t % ( round(stoptime - starttime, 4), ))

        # pdb.set_trace()
        print "FileReferences"
        pp( gREF.fileReferences )
        print

    if __name__ == '__main__':

        infiles = sys.argv[1:]
        for f in infiles:
            f = os.path.abspath( os.path.expanduser(f) )
            folder, filename = os.path.split( f )

            cfg = Config.Config()

            # if run from terminal, customize here
            cfg.accounts = True
            cfg.assets = True
```

```
            cfg.basetables = True
            cfg.customfunctions = True
            cfg.custommenus = True
            cfg.custommenusets = True
1135        cfg.privileges = True
            cfg.extendedprivileges = True
            cfg.filereferences = True
            cfg.layouts = True
            cfg.layoutGroups = False
1140        cfg.layoutOrder = False
            cfg.relationships = True
            cfg.scripts = True
            cfg.valueLists = True

1145        cfg.scriptGroups = False
            cfg.scriptOrder = False

            cfg.ignoreFilenameIDs = False

1150        # do not customize these
            cfg.summaryfile = f
            cfg.exportfolder = os.path.join( folder, "Exports")
            cfg.logfunction = logfunction

1155        main( cfg )
```

nodebox/FMPDDRSplitter.py

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

    import sys
  5 import os

    import traceback

    import thread
 10
    import time
    import re

    import pdb
 15 kwdbg = False

    import pprint
    pp = pprint.pprint

 20 import thread

    import objc

    import Foundation
 25 NSObject = Foundation.NSObject
    NSUserDefaults = Foundation.NSUserDefaults
    NSMutableDictionary = Foundation.NSMutableDictionary
    NSMakeRange = Foundation.NSMakeRange
    NSAttributedString = Foundation.NSAttributedString
 30 NSAutoreleasePool = Foundation.NSAutoreleasePool

    import AppKit
    NSWindowController = AppKit.NSWindowController
    NSApplication = AppKit.NSApplication
 35
```

```
   import PyObjCTools
   import PyObjCTools.AppHelper

   import Config
40 import ddrsplit

   gIsRunning = False
   gLastUpdate = 0.0

45 class FMPDDRSWindowController (NSWindowController):

       cbAssets = objc.IBOutlet()
       rbAssets = objc.IBOutlet()

50     cbBaseTables = objc.IBOutlet()
       cbAccounts = objc.IBOutlet()
       cbCustomFunctions = objc.IBOutlet()
       cbCustomMenus = objc.IBOutlet()
       cbCustomMenuSets = objc.IBOutlet()
55     cbPrivileges = objc.IBOutlet()
       cbExtendedPrivileges = objc.IBOutlet()
       cbFileReferences = objc.IBOutlet()
       cbRelationships = objc.IBOutlet()
       cbValueLists = objc.IBOutlet()
60
       cbAuthFiles = objc.IBOutlet()
       cbExternalDatasources = objc.IBOutlet()
       cbThemeCatalog = objc.IBOutlet()
       cbBaseDirectories = objc.IBOutlet()
65
       cbLayouts = objc.IBOutlet()
       cbLayoutFolders = objc.IBOutlet()
       cbLayoutOrder = objc.IBOutlet()

70     cbReferenceCollection = objc.IBOutlet()

       cbScripts = objc.IBOutlet()
       cbScriptFolders = objc.IBOutlet()
       cbScriptOrder = objc.IBOutlet()
75
       tbSummaryFile = objc.IBOutlet()
       tbExportFolder = objc.IBOutlet()

       tfStatusText = objc.IBOutlet()
80     rbAssets = objc.IBOutlet()
       cbIgnoreFilenameIDs = objc.IBOutlet()

       btOpenSummary = objc.IBOutlet()
       btOpenExport = objc.IBOutlet()
85     btCancel = objc.IBOutlet()
       btExport = objc.IBOutlet()

       summaryFile = None
       saveFolder = None
90
       def awakeFromNib(self):
           defaults = NSUserDefaults.standardUserDefaults()

           self.cbAssets.setState_(defaults.boolForKey_( u"assets" ))
95
           self.cbBaseTables.setState_(defaults.boolForKey_( u"basetables" ))

           self.cbAccounts.setState_(defaults.boolForKey_( u"accounts" ))
```

```
        self.cbCustomFunctions.setState_(defaults.boolForKey_( u"customfunctions" ))

        self.cbCustomMenus.setState_(defaults.boolForKey_( u"custommenus" ))
        self.cbCustomMenuSets.setState_(defaults.boolForKey_( u"custommenusets" ))

        self.cbPrivileges.setState_(defaults.boolForKey_( u"privileges" ))
        self.cbExtendedPrivileges.setState_(defaults.boolForKey_( u"extendedprivileges" ))

        self.cbFileReferences.setState_(defaults.boolForKey_( u"filereferences" ))

        self.cbIgnoreFilenameIDs.setState_(defaults.boolForKey_( u"ignoreFilenameIDs" ))

        self.cbRelationships.setState_(defaults.boolForKey_( u"relationships" ))

        self.cbValueLists.setState_(defaults.boolForKey_( u"valueLists" ))

        self.cbAuthFiles.setState_(defaults.boolForKey_( u"authfiles" ))
        self.cbExternalDatasources.setState_(defaults.boolForKey_( u"externaldatasources" ))
        self.cbThemeCatalog.setState_(defaults.boolForKey_( u"themes" ))
        self.cbBaseDirectories.setState_(defaults.boolForKey_( u"basedirectories" ))

        self.cbLayouts.setState_(defaults.boolForKey_( u"layouts" ))
        self.cbLayoutFolders.setState_(defaults.boolForKey_( u"layoutGroups" ))
        self.cbLayoutOrder.setState_(defaults.boolForKey_( u"layoutOrder" ))

        self.cbReferenceCollection.setState_(defaults.boolForKey_( u"referenceCollection" ))

        self.cbScripts.setState_(defaults.boolForKey_( u"scripts" ))

        self.cbScriptFolders.setState_(defaults.boolForKey_( u"scriptGroups" ))
        self.cbScriptOrder.setState_(defaults.boolForKey_( u"scriptOrder" ))

        self.tbSummaryFile.setStringValue_(defaults.stringForKey_( u"summaryfile" ))
        self.tbExportFolder.setStringValue_(defaults.stringForKey_( u"exportfolder" ))

    @objc.IBAction
    def openSummary_(self, sender):
        summary = getSummaryFileDialog()
        if summary and os.path.exists( summary ):
            self.summaryFile = unicode(summary)
            self.tbSummaryFile.setStringValue_( self.summaryFile )

    @objc.IBAction
    def openSaveFolder_(self, sender):
        folder = getFolderDialog()
        if folder and os.path.exists( folder ):
            self.saveFolder = unicode(folder)
            self.tbExportFolder.setStringValue_( self.saveFolder )

    @objc.IBAction
    def doCancel_(self, sender):
        ddrsplit.gCancel = True

    @objc.IBAction
    def doExport_(self, sender):
        # pdb.set_trace()
        cfg = Config.Config()
        defaults = NSUserDefaults.standardUserDefaults()

        cfg.accounts = self.cbAccounts.state()
        defaults.setObject_forKey_(cfg.accounts, u'accounts')

        cfg.assets = self.cbAssets.state()
        defaults.setObject_forKey_(cfg.assets, u'assets')
```

```
        cfg.basetables = self.cbBaseTables.state()
        defaults.setObject_forKey_(cfg.basetables, u'basetables')

        cfg.customfunctions = self.cbCustomFunctions.state()
        defaults.setObject_forKey_(cfg.customfunctions, u'customfunctions')

        cfg.custommenus = self.cbCustomMenus.state()
        defaults.setObject_forKey_(cfg.custommenus, u'custommenus')

        cfg.custommenusets = self.cbCustomMenuSets.state()
        defaults.setObject_forKey_(cfg.custommenusets, u'custommenusets')

        cfg.privileges = self.cbPrivileges.state()
        defaults.setObject_forKey_(cfg.privileges, u'privileges')

        cfg.extendedprivileges = self.cbExtendedPrivileges.state()
        defaults.setObject_forKey_(cfg.extendedprivileges, u'extendedprivileges')

        cfg.filereferences = self.cbFileReferences.state()
        defaults.setObject_forKey_(cfg.filereferences, u'filereferences')

        cfg.layouts = self.cbLayouts.state()
        defaults.setObject_forKey_(cfg.layouts, u'layouts')

        cfg.layoutGroups = self.cbLayoutFolders.state()
        defaults.setObject_forKey_(cfg.layoutGroups, u'layoutGroups')

        cfg.layoutOrder = self.cbLayoutOrder.state()
        defaults.setObject_forKey_(cfg.layoutOrder, u'layoutOrder')

        cfg.relationships = self.cbRelationships.state()
        defaults.setObject_forKey_(cfg.relationships, u'relationships')

        # cbReferenceCollection
        cfg.referenceCollection = self.cbReferenceCollection.state()
        defaults.setObject_forKey_(cfg.referenceCollection, u'referenceCollection')

        cfg.scripts = self.cbScripts.state()
        defaults.setObject_forKey_(cfg.scripts, u'scripts')

        cfg.ignoreFilenameIDs = self.cbIgnoreFilenameIDs.state()
        defaults.setObject_forKey_(cfg.ignoreFilenameIDs, u'ignoreFilenameIDs')

        cfg.scriptGroups = self.cbScriptFolders.state()
        defaults.setObject_forKey_(cfg.scriptGroups, u'scriptGroups')

        cfg.scriptOrder = self.cbScriptOrder.state()
        defaults.setObject_forKey_(cfg.scriptOrder, u'scriptOrder')

        cfg.valueLists = self.cbValueLists.state()
        defaults.setObject_forKey_(cfg.valueLists, u'valueLists')

        cfg.authfile = self.cbAuthFiles.state()
        defaults.setObject_forKey_(cfg.authfile, u'authfiles')

        cfg.externaldatasources = self.cbExternalDatasources.state()
        defaults.setObject_forKey_(cfg.externaldatasources, u'externaldatasources')

        cfg.themecatalog = self.cbThemeCatalog.state()
        defaults.setObject_forKey_(cfg.themecatalog, u'themes')

        cfg.basedirectory = self.cbBaseDirectories.state()
        defaults.setObject_forKey_(cfg.basedirectory, u'basedirectories')
```

```
            cfg.summaryfile = self.tbSummaryFile.stringValue()
230         defaults.setObject_forKey_(cfg.summaryfile, u'summaryfile')

            cfg.exportfolder = self.tbExportFolder.stringValue()
            defaults.setObject_forKey_(cfg.exportfolder, u'exportfolder')

235         cfg.logfunction = callFromWorkerMsg_

            if 1:
                thread.start_new_thread(threadwrapper, (cfg, ) )
            else:
240             threadwrapper(cfg)

        def setButtons(self):
            if gIsRunning:
                self.btOpenSummary.setEnabled_( False )
245             self.btOpenExport.setEnabled_( False )
                self.btCancel.setEnabled_( True )
                self.btExport.setEnabled_( False )
            else:
                self.btOpenSummary.setEnabled_( True )
250             self.btOpenExport.setEnabled_( True )
                self.btCancel.setEnabled_( False )
                self.btExport.setEnabled_( True )

    class FMPDDRSAppDelegate(NSObject):
255
        def applicationDidFinishLaunching_(self, notification):
            global gIsRunning
            gIsRunning = False
            app = NSApplication.sharedApplication()
260         app.activateIgnoringOtherApps_(True)

            # ugly hack
            wins = app.windows()
            if not wins:
265             return
            win = wins[0]
            controller = win.windowController()
            controller.setButtons()

270     @objc.IBAction
        def terminate_(self, sender):
            pass

        @objc.IBAction
275     def orderFrontStandardAboutPanel_(self, sender):
            pass

        def initialize(self):
            # default settings for preferences
280         userdefaults = NSMutableDictionary.dictionary()

            userdefaults.setObject_forKey_(True,       u'accounts')
            userdefaults.setObject_forKey_(True,       u'assets')
            userdefaults.setObject_forKey_(True,       u'basetables')
285         userdefaults.setObject_forKey_(True,       u'customfunctions')
            userdefaults.setObject_forKey_(True,       u'custommenus')
            userdefaults.setObject_forKey_(True,       u'custommenusets')
            userdefaults.setObject_forKey_(True,       u'privileges')
            userdefaults.setObject_forKey_(True,       u'extendedprivileges')
290         userdefaults.setObject_forKey_(True,       u'filereferences')
            userdefaults.setObject_forKey_(False,      u'ignoreFilenameIDs')
```

```
        userdefaults.setObject_forKey_(True,       u'layouts')
        userdefaults.setObject_forKey_(True,       u'layoutGroups')
        userdefaults.setObject_forKey_(True,       u'layoutOrder')
        userdefaults.setObject_forKey_(True,       u'relationships')
        userdefaults.setObject_forKey_(True,       u'referenceCollection')
        userdefaults.setObject_forKey_(True,       u'scripts')
        userdefaults.setObject_forKey_(True,       u'scriptGroups')
        userdefaults.setObject_forKey_(True,       u'scriptOrder')
        userdefaults.setObject_forKey_(True,       u'valueLists')
        userdefaults.setObject_forKey_(u"",        u'summaryfile')
        userdefaults.setObject_forKey_(u"",        u'exportfolder')
        NSUserDefaults.standardUserDefaults().registerDefaults_(userdefaults)

    def applicationShouldTerminate_(self, aNotification):
        """Store preferences before quitting."""
        userdefaults = NSUserDefaults.standardUserDefaults()
        app = NSApplication.sharedApplication()
        app.activateIgnoringOtherApps_(True)
        win = app.keyWindow()
        if not win:
            return True
        c = win.windowController()
        if not c:
            return True
        userdefaults.setObject_forKey_(c.cbAccounts.state(),            u'accounts')
        userdefaults.setObject_forKey_(c.cbAssets.state(),              u'assets')
        userdefaults.setObject_forKey_(c.cbBaseTables.state(),          u'basetables')
        userdefaults.setObject_forKey_(c.cbCustomFunctions.state(),     u'customfunctions')
        userdefaults.setObject_forKey_(c.cbCustomMenus.state(),         u'custommenus')
        userdefaults.setObject_forKey_(c.cbCustomMenuSets.state(),      u'custommenusets')
        userdefaults.setObject_forKey_(c.cbPrivileges.state(),          u'privileges')
        userdefaults.setObject_forKey_(c.cbExtendedPrivileges.state(),  u'extendedprivileges')
        userdefaults.setObject_forKey_(c.cbFileReferences.state(),      u'filereferences')
        userdefaults.setObject_forKey_(c.cbIgnoreFilenameIDs.state(),   u'ignoreFilenameIDs')
        userdefaults.setObject_forKey_(c.cbLayouts.state(),             u'layouts')
        userdefaults.setObject_forKey_(c.cbLayoutFolders.state(),       u'layoutGroups')
        userdefaults.setObject_forKey_(c.cbLayoutOrder.state(),         u'layoutOrder')
        userdefaults.setObject_forKey_(c.cbRelationships.state(),       u'relationships')
        userdefaults.setObject_forKey_(c.cbReferenceCollection.state(), u'referenceCollection')
        userdefaults.setObject_forKey_(c.cbScripts.state(),             u'scripts')
        userdefaults.setObject_forKey_(c.cbScriptFolders.state(),       u'scriptGroups')
        userdefaults.setObject_forKey_(c.cbScriptOrder.state(),         u'scriptOrder')
        userdefaults.setObject_forKey_(c.cbValueLists.state(),          u'valueLists')

        userdefaults.setObject_forKey_(c.cbAuthFiles.state(),           u'authfiles')
        userdefaults.setObject_forKey_(c.cbExternalDatasources.state(), u'externaldatasources')
        userdefaults.setObject_forKey_(c.cbThemeCatalog.state(),        u'themes')
        userdefaults.setObject_forKey_(c.cbBaseDirectories.state(),     u'basedirectories')

        userdefaults.setObject_forKey_(c.tbSummaryFile.stringValue(),   u'summaryfile')
        userdefaults.setObject_forKey_(c.tbExportFolder.stringValue(),  u'exportfolder')
        return True

    def setstatus_show_(self, appendage, showit=False):
        global gLastUpdate
        s = appendage + u"\n"
        if type(s) != unicode:
            s = unicode(appendage, "utf-8")
        sys.stdout.write(s.encode("utf-8"))
        # view
        app = NSApplication.sharedApplication()
        wins = app.windows()
        if not wins:
            return
```

```
            win = wins[0]
            controller = win.windowController()

            myView = controller.tfStatusText
360
            #
            # model
            storage = myView.textStorage()

365         # where in the string
            l = storage.length()
            endRange = NSMakeRange(l, 0)

            # merge it
370         try:
                t = NSAttributedString.alloc().initWithString_(s)
                storage.appendAttributedString_(t)
            except Exception, v:
                print
375             print "ERROR status inserting:", v
                print

            tm = time.time()
            if showit and tm >= gLastUpdate + 0.3:
380             gLastUpdate = tm
                try:
                    myView.scrollRangeToVisible_(endRange)
                    myView.setNeedsDisplay_(True)
                except Exception, v:
385                 print
                    print "ERROR status scrolling:", v
                    print

    def threadwrapper(cfg):
390     global gIsRunning
        pool = NSAutoreleasePool.alloc().init()
        if not pool:
            return
        if gIsRunning:
395         return
        gIsRunning = True
        ddrsplit.gCancel = False

        # view
400     app = NSApplication.sharedApplication()
        win = app.keyWindow()
        controller = win.windowController()
        controller.setButtons()

405     try:
            ddrsplit.main(cfg)

        except (Exception,), v:
            tb = traceback.format_exc()
410         tb = unicode( tb )
            v = unicode( repr(v) )
            tb = tb + u"\n\n" + v
            sys.stdout.write( tb )

415     finally:
            gIsRunning = False
            ddrsplit.gCancel = False
            controller.setButtons()
        del pool
```

```
420
    def getFolderDialog():
        panel = AppKit.NSOpenPanel.openPanel()
        panel.setCanChooseFiles_(False)
        panel.setCanChooseDirectories_(True)
425     panel.setAllowsMultipleSelection_(False)
        rval = panel.runModalForTypes_([])
        if rval != 0:
            f = [unicode(t) for t in panel.filenames()]
            return f[0]
430     else:
            return False

    def getSummaryFileDialog():
        panel = AppKit.NSOpenPanel.openPanel()
435     panel.setCanChooseFiles_(True)
        panel.setCanChooseDirectories_(False)
        panel.setAllowsMultipleSelection_(False)
        rval = panel.runModalForTypes_( ['xml'] )
        if rval != 0:
440         f = [unicode(t) for t in panel.filenames()]
            return f[0]
        else:
            return False

445 def callFromWorkerMsg_( message ):
        PyObjCTools.AppHelper.callAfter( statwrap_, message=message)

    def statwrap_(message):
        appl = NSApplication.sharedApplication()
450     delg = appl.delegate()
        try:
            delg.setstatus_show_(message, 1)
        except Exception, v:
            print
455         print "STATUS WRAPPER ERROR:", v
            print

    if __name__ == "__main__":
        PyObjCTools.AppHelper.runEventLoop()
```

nodebox/ReferenceCollector.py

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

    import sys
  5 import os

    import pdb

    class ReferenceCollector(object):
 10     """A 2-pass data collector.

        1. pass
            collect all references by value (file, type, name, [to], [id], [scriptline])
                -> (file, type, name, scriptline)
 15
        2. pass
            convert all types to ID
            convert all files to ID
            convert all names to ID
 20         convert all tuples to ID
```

```
        create referencetable ID -> ID
        create filetable ID,name
        create BTtable ID,name
        create TOtable ID,name,btID

    """
    def __init__(self):

        self.references = {}

        # (file, type, name, scriptline) -> ID
        self.objects = {}
        self.objectsReverse = {}
        self.objectID = 1

        self.filemakerAttributes = {}

        self.fileReferences = {}

        #
        # for now this is only chaotic sugar coating
        #

        # (fileID,
        self.tables = {}
        self.tablesID = 1

        # (name,
        self.files = {}
        self.filesID = 1

        # (name, bt, to) -> id
        self.layouts = {}
        self.layoutsID = 1

        self.variables = {}

    def pp(self):
        pass

    def addObject(self, obj):
        """retrieve ID for object. Add obj to collection if necessary."""

        if 0: #obj == "":
            pdb.set_trace()

        # xmlref, typ, name, *rest = obj

        if obj in self.objects:
            return self.objects[ obj ]
        else:
            i = self.objectID
            self.objects[ obj ] = i
            self.objectsReverse[ i ] = obj
            self.objectID += 1
            return i

    def addFileReference(self, xmlsource, xmldest, name, id_, pathList):
        if xmlsource not in self.fileReferences:
            self.fileReferences[xmlsource] = {
                                                name: {
                                                    'id': id_,
                                                    'pathList': pathList,
                                                    'destination': xmldest}}
```

```
        else:
            d = self.fileReferences[xmlsource]
            if name not in d:
                d[name] = {'id': id_,'pathList': pathList,'destination': xmldest}
            else:
                # checking for double entries here?
                d[name] = {'id': id_,'pathList': pathList,'destination': xmldest}

    def addReference(self, ref1, ref2):
        """ref1 object refers to ref2 object

        objects are ALWAYS passed by value. E.g. (file, type, name) tuples."""

        idref1 = self.addObject( ref1 )
        idref2 = self.addObject( ref2 )

        if idref1 not in self.references:
            self.references[ idref1 ] = [ idref2, ]
        else:
            if idref2 not in self.references[ idref1 ]:
                self.references[ idref1 ].append( idref2 )

    def addFilemakerAttribute(self, idobj, name, value):
        # pdb.set_trace()
        if idobj not in self.filemakerAttributes:
            self.filemakerAttributes[ idobj ] = {name:value}
        else:
            d = self.filemakerAttributes[ idobj ]

            if name not in d:
                d[name] = value
            else:
                # should i check unequal values here?
                d[name] = value
```

nodebox/setup.py

```
"""
Script for building the FileMaker-DDR-Splitter application

Usage:
    python setup.py py2app -gx -O2
"""
import os

from distutils.core import setup
import py2app

appname = "FileMaker DDR Splitter"
appnameshort = "FMPSplit"
version = "V0.4.1"

copyright = u"Copyright 2015 Karsten Wolf"

infostr = appname + u' ' + version + u' ' + copyright

setup(
    app=[{
        'script': "FMPDDRSplitter.py",

        'plist':{
            'CFBundleGetInfoString': infostr,
            'CFBundleIdentifier': 'org.kw.ddrsplitter',
```

```
            'CFBundleShortVersionString': version,
            'CFBundleDisplayName': appnameshort,
            'CFBundleName': appnameshort,
            'CFBundleSignature': 'KWDs',
            'LSHasLocalizedDisplayName': False,
            'NSAppleScriptEnabled': False,
            'NSHumanReadableCopyright': copyright}},

    data_files=["English.lproj/MainMenu.xib",
                "Icon.icns"],
    options={
        'py2app':{
            'iconfile': "Icon.icns",
            'excludes':['Tkinter',],
        },
    },
)
```