```
class BusinessObject:
  def __init__(self, objType, objName, objRevision):
    self.objType = objType
    self.objName = objName
    self.objRevision = objRevision

class MQLManager:
  def expand(self, businessObject, direction):
    name = re.sub("_", "/", businessObject.objName)
    name = re.sub("=", " ", name)

    #issue the command
    command = 'expand businessobject ' + businessObject.objType + ' "' + name
    command = command + '" ' + businessObject.objRevision + " " + direction + " recurse to 1;'
    data, errors = self.runCommand(command)

    #convert the list of lines into a list of Businessobjects
    objects = []
    for line in data:
      line = re.sub("/", "_", line)
      line = re.sub(" at ", " at ",line)
      line = re.split(' ', line)
      objects.append(BusinessObject(line[6], line[8], line[10]))

    return objects
```

```
def checkoutFiles(self, businessObject, directory):
    name = re.sub("_", "/", businessObject.objName)
    name = re.sub("=", " ", name)

    #determine the type(s) of the objects
    types = self.getProperties(businessObject, "select format.format")
    types = types[1:]
    for i in range(len(types)):
        line = types[i]
        line = re.split("format.format =", line)
        types[i] = line[1]

    #determine if attachments exist
    if len(types[0]) < 2:
        return [] #there are no files

    #extract the files
    for format in types:
        command = 'checkout businessobject ' + businessObject.objType + ' "' + name
        command += '" ' + businessObject.objRevision + ' format ' + format + ' file all ' + directory + ';'
        data, errors = self.runCommand(command)

    #use popen and ls to determine what files were added
    attachments = os.popen('ls').read()
    attachments = re.split('\n', attachments)

    return attachments
```

```
def getProperties(self, businessObject, attrs):
    name = re.sub("_", "/", businessObject.objName)
    name = re.sub("=", " ", name)

    #issue the command
    command = 'print businessobject ' + businessObject.objType + ' "' + name
    command = command + '" ' + businessObject.objRevision + ' ' + attrs + ';'
    data, errors = self.runCommand(command)

    return data

def exeQuery(self, template):
    #issue the commands
    command1 = "add query extract1 businessobject " + template.objType + ' "' + template.objName
    command1 = command1 + '" ' + template.objRevision + ";"
    command2 = "eval query extract1;"
    command3 = "delete query extract1;"
    data0, error1 = self.runCommand(command3)
    data1, error1 = self.runCommand(command1)
    data2, error2 = self.runCommand(command2)
    data3, error3 = self.runCommand(command3)

    #make a list of the BusisnessObjects
    objects = []
    for line in data2:
      line = re.sub("/", "_", line)
      line = re.sub(" at ", "=at=",line)
      line = re.split(' ', line)
      objects.append(BusinessObject(line[0], line[1], line[2]))

    return objects
```

```
def runCommand(self, MQLCommand):
    #put all of the required protocalls around the MQL request
    header = 'set context person SVXII_guest password "";\n'
    footer = 'quit;\n'

    #execute the script on the os
    datain, dataout, dataerr = os.popen3("mql")
    datain.write(header)
    datain.write(MQLCommand + "\n")
    datain.write(footer)
    datain.close()
    dlines = dataout.readlines()
    elines = dataerr.readlines()

    #strip out usless header information and '\n' chars
    dlines = dlines[4:]
    for index in range(len(dlines)):
        dlines[index] = dlines[index][0:-1]

    #return the results
    return dlines, elines
```

```
class Stack:
    def __init__(self):
        self.data = []

    def push(self, obj):
        self.data.append(obj)

    def peek(self):
        if self.data:
            return self.data[-1]
        else:
            return None

    def pop(self):
        obj = self.data[-1]
        self.data = self.data[0:-1]
        return obj

class Tag:
    def __init__(self, tagName):
        self.name = tagName
        self.attrs = []
        self.tags = []
        self.value = None

    def addAttr(self, attrName, attrValue):
        self.attrs.append((attrName, attrValue))

    def addNestedTag(self, tag):
        self.tags.append(tag)
```

```
def setValue(self, tagValue):
    self.value = tagValue

def __str__(self):
    #start the tag
    tagString = '<' + self.name
    #add attributes
    for attr in self.attrs:
        tagString += " " + attr[0] + '="' + attr[1] + '"'
    tagString += ">"
    #add the value
    if self.value:
        tagString += self.value
    else:
        tagString += "\n"
    #add nested tags
    for tag in self.tags:
        tagString += str(tag)
    #add close tag
    tagString += "</" + self.name + ">\n"
    return tagString
```

```
class XMLDocument:
    def __init__(self):
        self.heading = '<?xml version="1.0" encoding="ISO-8859-1"?>'
        self.root = None
        self.dtd = []
        self.tags = Stack()

    def openTag(self, tagName, attrs):
        newTag = Tag(tagName)
        for index in range(len(attrs)/2):
            newTag.addAttr(attrs[index*2], attrs[index*2 +1])
        if not self.root:
            self.root = newTag
        parentTag = self.tags.peek()
        if parentTag:
            parentTag.addNestedTag(newTag)
        self.tags.push(newTag)

    def closeTag(self):
        self.tags.pop()

    def addTag(self, tagName, attrs, value):
        newTag = Tag(tagName)
        newTag.setValue(value)
        for index in range(len(attrs)/2):
            newTag.addAttr(attr[index*2], attr[index*2 +1])
        parentTag = self.tags.peek()
        parentTag.addNestedTag(newTag)
```

```
    def __str__(self):
        if self.root:
            return self.heading + "\n\n" + str(self.root)
        else:
            return self.heading

    def write(self, filename):
        fout = open(filename, "w")
        fout.write(str(self))
        fout.close()

manager = MQLManager()

def getTypes():
    din = os.popen("ls", "r")
    types = din.readlines()
    for i in range(len(types)):
        types[i] = types[i][0:-1]
    return types

def getDatabaseObjects(Otype):
    template = BusinessObject(Otype, "*", "*")
    objs = manager.exeQuery(template)
    return objs
```

```
def buildHTMLPage(BO, children, parents, attachments):
    #create the info.html page
    page = SimpleDocument(title=BO.objType + " " + BO.objName, cgi=0)
    page.append(Heading(1, BO.objType + " " + BO.objName))
    page.append(BR(1))

    #add the children to the page
    page.append(Heading(2, "Children of this object"))
    page.append(BR(1))
    if len(children) == 0:
        page.append("None")
        page.append(BR(2))
    else:
        for child in children:
            oName = child.objName
            oType = child.objType
            page.append(Href(url="../../" + oType + "/" + oName + "/info.html", text=oType + " " + oName))
            page.append(BR(1))
        page.append(BR(1))
```

```
#add the parents to the page
page.append(Heading(2, "Parents of this object"))
page.append(BR(1))
if len(parents) == 0:
    page.append("None")
    page.append(BR(2))
else:
    for parent in parents:
        oName = parent.objName
        oType = parent.objType
        page.append(Href(url="../../" + oType + "/" + oName + "/info.html", text=oType + " " + oName))
        page.append(BR(1))
    page.append(BR(1))

#add the attachments to the page
page.append(Heading(2, "Attachments of this object"))
page.append(BR(1))
if len(attachments) == 0:
    page.append("None")
else:
    for attachment in attachments:
        page.append(Href(url=attachment, text=attachment))
        page.append(BR(1))
page.append(BR(1))

#add my name to the bottom ;)
page.append(Heading(5, "Page created by Phil Scherer and Dr. Ritchie"))

page.write(filename="info.html")
```

```
def main():
    types = getTypes()
    for typ in types:
        objs = getDatabaseObjects(typ)
        for obj in objs:
            extract(obj)
    print "Finished Extraction"

if __name__ == "__main__":
    main()
else:
    pass
```

```
def buildXMLPage(BO, links, propertyTuples):
    #create the data.xml page
    page = XMLDocument()
    page.openTag("businessobject", ())
    page.addTag("name", (), BO.objName)
    page.addTag("type", (), BO.objType)
    page.addTag("version", (), BO.objRevision)

    #add the links to the page
    page.openTag("links", ())
    for obj in links[0]:
        page.openTag("link", ("link", "to"))
        page.addTag("type", (), obj.objType)
        page.addTag("name", (), obj.objName)
        page.addTag("version", (), obj.objRevision)
        page.closeTag()
    for obj in links[1]:
        page.openTag("link", ("link", "from"))
        page.addTag("type", (), obj.objType)
        page.addTag("name", (), obj.objName)
        page.addTag("version", (), obj.objRevision)
        page.closeTag()
    page.closeTag()
```

```
#add the properties
page.openTag("properties", ())
for propertyTuple in propertyTuples:
    propName = propertyTuple[0]
    propValue = propertyTuple[1]
    page.addTag(propName, (), propValue)
page.closeTag()
page.closeTag()

page.write(filename="data.xml")
```

```
def extract(BO):
    #create directory for this object, or if it exists, then just return
    os.chdir(BO.objType)
    try:
        os.chdir(BO.objName)
        os.chdir("../..")
        print BO.objType + " " + BO.objName + " already exists. Skipping"
        return
    except:
        os.mkdir(BO.objName)
        os.chdir(BO.objName)
        print BO.objType + " " + BO.objName + " is new. Extracting"

    #retrive all the information for this object
    children = manager.expand(BO, "to")
    parents = manager.expand(BO, "from")
    attachments = manager.checkoutFiles(BO, os.getcwd())

    properties = manager.getProperties(BO, "select attribute.value")
    properties = properties[1:]
    propertyTuples = []
    for prop in properties:
        header = prop[0:14]
if header == "    attribute[":
            prop = re.split("].value = ", prop)
            propName = prop[0][14:]
            propValue = prop[1]
            if len(propValue) == 0:
                propValue = " "
            propertyTuples.append((propName, propValue))
        else: #it needs to be appended onto the previous element
```

```
        prevI = len(propertyTuples) - 1
        prevName = propertyTuples[prevI][0]
        prevValue = propertyTuples[prevI][1]
        propertyTuples[prevI] = (prevName, prevValue + prop)

    #make the HTML and XML pages
    buildHTMLPage(BO, children, parents, attachments)
    buildXMLPage(BO, (children, parents), propetryTuples)

    os.chdir("../..")


manager = MQLManager()

def getObjectTypes():
    din = os.popen("ls", "r")
    lines = din.readlines()
    types = []
    for line in lines:
        types.append(line[0:-1]) #remove \n chars

    return types

def getObjectNames(objType):
    din = os.popen("ls " + objType, "r")
    lines = din.readlines()
    names = []
    for line in lines:
        names.append(line[0:-1])

    return names
```

```
def verifyObject(BO):
    din = openings("ls " + BO.objType + "/" + BO.objName, "r")
    lines = din.readlines()
    hasHTML = 0
    hasXML = 0
    attachments = []
    for line in lines:
        line = line[0:-1] #remove \n chars
        if line == "info.html":
            hasHTML = 1
        elif line == "data.xml":
            hasXML = 1
        else:
            attachments.append(line)

    return (hasHTML, hasXML, attachments)
```

```
#simple existence verification
def phase1():
    failures = []
    types = getObjectTypes()
    for typ in types:
        names = getObjectNames(typ)
        for name in names:
            BO = BusinessObject(typ, name, "A")
            print "Phase 1 Error Detection:", typ, name,
            result = verifyObject(BO)
            if result[0] == 1 and result[1] == 1:
                print "Passed!"
            else:
                failures.append(BO)
                print "Failed!"

    return failures
```

```
#complex accuracy verification
def phase2():
    failures = []
    types = getObjectTypes()
    for typ in types:
        names = getObjectNames(typ)
        for name in names:
            BO = BusinessObject(typ, name, "A")
            print "Phase 2 Error Detection:", typ, name,
            #Check database
            datattachments = []
            lines = manager.getProperties(BO, "select format.file")
            lines = lines[1:] #remove the first line, it doesn't have info we need
            for line in lines:
                line = re.split("    format.file =", line)
                if len(line[1]) > 2: #is there an attachment
                    attachment = re.split("/", line[1])
                    attachment = attachment[-1]
                    datattachemts.append(attachment)
            #Check folder
            result = verifyObject(BO)
            outtachments = result[2]
            #compare the lists
            passed = 1
            if len(datattachments) == len(outtachments): #length
```

```
    #contents
    datattachments.sort()
    outtachments.sort()
    for i in range(len(datattachments)):
        if passed == 1: #no failure yet
            if datattachments[i] != outtachments[i]:
                passed = 0
else:
    passed = 0

if passed == 1:
    print "Passed!"
else:
    failures.append(BO)
    print "Failed!"
```

```
def main():
    failures1 = phase1()
    failures2 = phase2()
    if len(failures1) == 0:
        print "Phase1: No failures"
    else:
        print "Phase1 Failures:"
        for failure in failures1:
            print failure.objType, failure.objName
    if len(failures2) == 0:
        print "Phase2: No failures"
    else:
        print "Phase2 Failures:"
        for failure in failures2:
            print failure.objType, failure.objName

if __name__ == "__main__":
    main()
else:
    pass
```