DOMS object model creation from tree

Description of the general method of creating the initial object hierarchy in DOMS based on a tree structure in a file system

General description

The goal of this document is to describe the work of transforming data files and accompanying metadata files in a tree structure on disk to a Content-Model-less object tree in DOMS.

For this to be generic some assumptions needs to be taken:

- Data files and metadata files that belongs together have the same prefix
- Data files can be recognized by their file suffix.
- Checksum files is not supposed to be represented directly, but rather as a property of the data the are associated with. They will thus be skipped as objects in the tree, but used in the ingest of the data they belong to.

General rules:

1. Fedora Objects correspond to file system directories
2. Sub-directories are represented by a "hasPart" relation to the subdirectory object.
3. Each object will, as an identifier, have the file system path to the directory
4. A data file is a file containing data. The actual data is stored outside DOMS. The data file is represented as a file object in doms.
5. A metadata file is a file containing metadata. The file is stored inside DOMS.
6. A grouping of files (files having a common prefix) is represented as a object with:
    a. Datastreams for each metadata file
    b. "hasFile" relations to data files (hasFile is a specialization of hasPart)

Pseudo code expressing the above rules

The following pseudo code is meant to express the above rules on a more formal basis.

In the codes, the methods:

- groupByPrefix(): returns a list of lists of files, grouped by their common prefix.
- isDataFile(): returns a boolean telling if the given file is a datafile.

```
void handleDir(myDir, domsParentObject) {
  thisDirObject = new Object(identifier = myDir.getPath());
  domsParentObject.addHasPart(object = thisDirObject);

  handleFiles(myDir.getFiles(), thisDirObject);

  for(dir in myDir.getDirectories()) {

    handleDir(dir, thisDirObject);
  }
}

void handleFiles(myFiles, dirParentObject) {
  groupedByPrefix = groupByPrefix(myFiles) //groupedByPrefix is a set of
groups. A group is a prefix and a set of files
  if (groupedByPrefix.size == 1){ //There is only one group, so add them as
datastreams to the current
      group = groupedByPrefix.get(0)
      for (file in group){
       handleFile(file, dirParentObject);
   }
  } else { // there is more than one group, so introduce sub directories
    for(group in groupedByPrefix) {
      addPart(group, dirParentObject);
    }
  }
}

void handleFile(file, parentObject) {
  if(isDataFile(file)) {
    addFile(file, parentObject);
  } else {
    addDataStream(file, parentObject);
  }
}

void addPart(fileGroup, dirParentObject) {
  thisPartObject = new Object(identifier = fileGroup.getPrefix());
  dirParentObject.addHasPart(object = thisPartObject);

  for(file in fileGroup) {
    handleFile(file, thisPartObject);
  }
}

void addFile(file, parentObject) {
   thisFileObject = new Object(identifier = file.getName);
   parentObject.addHasFile(object = thisFileObject);
}
```