Columbia University
CS4115 Programming Languages and Translators
Professor Stephen A. Edwards
Summer 2008

Proposal for CATALOG programming language

by Leonid Velikoselskiy (lvelik@gmail.com)

INTODUCTION

In our days of increasing dependency on digital information, while our old photo albums and music records are collecting dust on shelves, the great volumes of MP3s, AVIs, JPEGs and other file formats are growing on our hard drives, filling up writable media and creating a need for more removable storage and backup hardware. We think it is time to create something that will help us organize this chaos, remove a great deal of redundancy and hopefully free up some space. Unfortunately, it is not easy to build a general tool to help people catalog their data, therefore, we propose to build a whole new language for this purpose.

IDEA

All files are of certain type, which is defined by their extention and format. Files also have other properties depending on their type. For example, an image file can have *Camera Model* property and a music file can have *Artist* and *Album* properties. Properties are usually used to organize the files. Music can be in a folder named after its genre and photos can be in a folder that has a name of the event that photos were taken at, but may also contain the date of this event within the folder name.

Unfortunately, there is no simple way to translate these properties into directory structures or to easily move files from folder to folder depending on their types without manual intervention. Operating systems provide some tools, but most of the time users have to repeat same steps over and over again.

We propose to write a compiler for an object-oriented programming language that helps deal with digital media files and other files in an organized manner. This language, called CATALOG, will be simple enough for all users to be able to create custom programs for themselves quickly in order to catalog their data.

CLASS LIBRARY

All classes in CATALOG will derive from File class, just like in Java, where all classes derive from Object classs. Each type of file will be its own class. FileMusic, FileImage, FileVideo will all be subclasses of the File class. They will inherit main properties of File class such as *Size*, *Name*, *Location* and have their own properties such as *DatePictureTaken*, *Title*, etc.. When a general collection of files will be read from a certain folder, programmer would be able to check each file to see what type it is, in order to decide what to do with it.

There will also be a class named Folder that will not derive from File and no classes will derive from it either. Since all folders are created equal, this class will have only one implementation.

SYNTAX

Since the goal of the CATALOG programming language is to help catalog digital media, it must have a syntax that will allow programmers to extract file information and do file management tasks easily and effortlessly. Following are two real-life examples where this language can be used.

1) A number of unorganized music files is copied to «C:\Documents\Music» folder. User needs to organize these files into folders and subfolders with Genre names followed by Artist names followed by Album names and, finally, with the actual files inside. So, if the file were named «A Love Song.mp3» by «Famous Artist» from the album «The Best Of», it would be copied to C:\Documents\Music\Pop\Famous Artist\The Best Of\A Love Song.mp3. This needs to be done for all files.

The program to do this would look like this:

```
forall files in «C:\Documents\Music» {
     Folder folder = file.Genre\file.Artist\file.Album
     move file to folder
}
```

forall keyword will read all physical files from «C:\Documents\Music» into **files** collection. **files** collection will consist of objects of type File. Variable **file** will point to the next file in the collection on each iteration. Compiler will automatically add code to check if the folders exist. If one of the folders does not exist, there will be code to create it. In any case, a pointer to an existing folder will be assigned to **file** variable.

2) Another example will be a program that renames all image files in a folder adding an index at the end of the file name (before file's extension). Since files being renamed will be sorted by *Date Picture Taken*, the indexing will be correct chronologically. This is useful in case there are pictures from two different cameras taken at the same time at the same event, but need to compile one complete album.

```
forall files in «C:\Documents\Pics» sorted by file.DatePictureTaken {
     file.Name = «BirthdayParty» + file.CurrentIndex
}
```

file.*CurrentIndex* is a dynamic property of type String that represents **file**'s place in **files** collection. If the collection were sorted by a different static property (not by *DatePictureTaken*), some files would have different indices during program execution. Since number of files in the collection is known at compile time, it is also known how many 0's to add before *CurrentIndex*. For purposes of cataloging to allow for correct sorting by file names, index will start with 001, in case there are less than 1000 files (this is just one scenario).

These examples show just two out of many scenarios where CATALOG language will simplify such tasks as creating folders from file properties and moving and renaming files. Other use cases may include removing duplicate image files from multiple folders, adding ID3 tags to music files using file locations and names, shifting a «Date Created» property by some amount of time, and finding and deleting all empty folders in a directory structure.

SUMMARY

CATALOG will be a programming language with a wide scope of practical application. People will be able to create programs that are custom to their own file keeping habits and directory structures. This is a flexibility that file manipulation and cataloging tools cannot provide and operating systems simply lack. The language will be content oriented and very expandable, which is essential due to increasing number of digital formats and file properties that will be created in the future.