Proposal: File and Directory Manipulation Language (FDL)

Cara J. Borenstein
Columbia University
cjb2182@columbia.edu

Daniel Garzon
Columbia University
dg2796@columbia.edu

Daniel L. Newman
Columbia University
dln2111@columbia.edu

Pranav Bhalla
Columbia University
pb2538@columbia.edu

Rupayan Basu
Columbia University
rb3034@columbia.edu

September 25, 2013

1 Motivation

With the proliferation of storage devices, and the rise of mobile and cloud computing, users must now manage a large number of files scattered across several locations. Furthermore, with the availability of inexpensive storage options, users do not feel the need to delete files, often leading to an unmanageable accumulation of files. Thus the problem of accessing and organizing multiple files quickly and easily across diverse storage media is becoming increasingly important.

While the GUI offered by various operating systems is inefficient for handling large number of files and directories at the same time, the command line interface on the other hand requires users to learn complex Swiss-knife like commands and their innumerable options, even to perform basic operations. What is required is a programming language, that allows users to write simple programs that perform both specialized as well as routine tasks to efficiently and easily organize their files and directories.

2 Description

File and Directory Manipulation Language (FDL, pronounced fiddle) solves this problem by providing a simple and intuitive syntax for managing files and directories. By providing the user with new data types, and an extensive list of mathematical and logical operators, what used to be tedious and time consuming will now be easy and fast.

Users can write programs that organize their file systems by conveniently copying files and directories to different locations, and removing files and directories from specific file paths, through the use of mathematical operators. Users can loop through subdirectories and files contained within a chosen directory, with a template to browse the file/directory tree stemming from that directory by specifying different levels. One example is the ability to perform a function on all nodes of the tree at a certain level away from the root directory.

Files can be organized in this manner by the attributes spanning from last modified date to size, and additional, customized tags can be added to files for organizational purposes. Customized tags can be serialized and stored on the machine in XML format, to be loaded when users are navigating the file system.

3 Syntax

3.1 Basic Data Types

primitive	**Description**
int	The set of all positive natural numbers: $\mathbb{N}^0 = \{0, 1, 2, 3, \ldots, k\}$
bool	Used to compare two files or directories for equality. Returns 1 for *true* and 0 for *false*.
string	A sequence of *characters* surrounded by quotes.
dir	Object that holds the path to a *collection* of 0 or more *files* in memory. Directories can contain any number of *files* and any number of *sub-directories*.
file	Object that has a *file_type*, *modified_date*, *created_date*, and 0 or more customized *tags*.

3.2 File and Folder Attributes

attribute	**Description**
created_date	Field that holds the date when a *file* or *directory* was created.
modified_date	Field that holds the date of the last time a *file* or *directory* was modified.
file_type	Field that holds the type of a *file*. (ex. 'txt', 'jpeg').
tag	Field that holds a *customized association* of a *file*.
path	Field that holds the path of the *file* or *directory*.
name	Field that holds the name of the *file* or *directory*.

3.3 Mathematical Operators

operator	**Description**
$+$	Used to add *files* to *directories* and also to append *strings*.
$-$	Used to remove 1 or more *files* from a *directory*.
,	Used to specify multiple objects that should be evaluated separately by the previous operator.
$=$	Assignment operator.
$+=$	For a *directory* it is used to add a *file* or *sub-directory* to the *directory*. For integers, it is the *addition and assignment* operator.
$-=$	For a *directory* it is used to remove a *file* or *sub-directory* to from *directory*. For integers, it is the *substraction and assignment* operator.

3.4 Logical Operators

operator	**Description**
$==$	Equality operator.
$!=$	Inequality operator.
$>$	Used for checking the level of a *sub-directory* (select *files* at a level *greater than* the current *directory*), and for comparing *integers*.
$>=$	Used for checking the level of a *sub-directory* (select *files* at a level *greater than or equal to* the current *directory*), and for comparing *integers*.
$<$	Used for checking the level of a *sub-directory* (select *files* at a level *less than* the current *directory*), and for comparing *integers*.
$<=$	Used for checking the level of a *sub-directory* (select *files* at a level *less than or equal to* the current *directory*), and for comparing *integers*.

3.5 Control Statements

3.5.1 *if-then-else*

```
if <condition> then
  <expression>
else
  <expression>
end
```

3.5.2 *while*

```
while <condition> then
  <expression>
end
```

3.5.3 *for*

```
for <identifier> in <set of files> do
  <expression>
end
```

3.6 Function Definition

```
def <identifier> (<parameter list>)
  <expression>
end
```

4 Example Programs

4.1 Case 1:

Write a program that can pickup all .jpg files in a directory, or sub-directory, and create new folders by date and save copies in the respective folder.

```
    def main()
      dir D1 = "/SAMPLE_PATH" //path to the source directory
      string str = ""  //path to the destination folder

      // we expect file_temp will loop over all files in "D1" including subfolders
      for file_temp in D1 do
        if file_temp.type == "jpeg" do
          // we wish to name the folders with date on which images were created
          // the below stmt creates(in case it didnt exist) or points dtemp to the folder.
          dir dir_temp = str + file_temp.Date
          dir_temp += file_temp
        end
      end
    end
```

4.2 Case 2:

A user has downloaded several project folders from a course website and would like to separate the code and document files in these folders and organize them into two folders.

```
    def main()
      //Assuming project folders were unzipped in directory W4115
      dir desktop = |~/Desktop|
      dir projects = desktop.path + |/W4115|

      //Create new directories in the desktop
      dir project_code = desktop.path + |/projectCode|
      dir project_docs = desktop.path + |/projectDocs|

      for dir_temp in projects do
        for file_temp in dir_temp level <= 3 do
          if file_temp.type == "ml" do
            projectCode += file_temp
          else if file_temp.type == "pdf"
            projectDocs += file_temp
          end
        end
      end
    end
```

4.3 Case 3:

Suppose there is a group of peers who want to share pictures, or any other file, amongst themselves. One of them should be able to take the shared files, and copy them, but some duplicates may exist. That individual should be able to write a program that deletes the duplicates and copies all the disting files to a new directory.

```
def main()
  dir D1 = "" //path to the first source directory
  dir D2 = "" //path to the second source directory
  dir D3 = "" //path to the destination directory with no duplicates
  string duplicate_file_path  //list of comma separated duplicate files paths

  //We wish to compare files in the two folder(and subfolders)
  for file_temp1 in D1 do
    bool flag = true
    for file_dest in D3 do
      if file_temp.type == "jpeg":
        if file_temp1.name == file_dest.Name:
          // duplicate file found
          flag = false
          duplicate_file_path += " , " + file_temp1.path
          D1 -= file_temp1   // delete duplicate from original
          break
        end
      end
    end
  end
  if flag == true do
    D3 += file_temp1
  end

  for file_temp2 in D2 do
    bool flag = true
    for file_dest in D3 do
      if file_temp.type == "jpeg"
        if file_temp2.name == file_dest.name do
          flag = false
          duplicate_file_path += ", " + file_temp1.Path
          D1 -= file_temp2
          break
        end
      end
    if flag == true do
      D3 += file_temp2
    end
  end
end
```

4.4 Case 4:

User has copied 500 image files from his camera to a folder Canon, and would like to rename all of them to something meaningful.

```
def main()
  dir camera = input(|Enter device path: |)
  string name_prefix = input(|Enter name prefix: |)
  dir myPictures = |~/Desktop/MyPictures|

  count = 1
  for file in camera do
    if file.type == |jpeg| do
      file.name = name_prefix + string(count) //Convert int to string
      count = count + 1
      myPictures += file
    end
  end
end
```

4.5 Case 5:

Using custom tags to list all ebooks that have been read from a folder containing ebooks organized into subfolders A-Z, and add the wishlist tag to all other ebooks.

```
def main()
  dir library = "~/Desktop/Ebooks"
  print |List of books read: \n|
  for file in library level=|all| do
    if file.tag == |read|:
      print file.name + |\n|
    else
      file.tag = |wishlist|
    end
  end
end
```