FJL:
THE FILE JOURNALING LANGUAGE

John Petrella
jp@cs.columbia.edu
UNI: jep2124
June 11, 2007

Introduction:

In today's day and age the cost of hard drives are nominal and has consequently created the notion of unlimited file storage. Due to this trend remembering a file's location in a directory structure or keeping track of which files are living on a system has become a chore. The use of **updatedb** and **locate**, has become cumbersome returning a long listing of files, most of which are typically useless. FJL was invented as a way to keep track of the files a user owns. It is eloquent and useful, providing tailored file views consisting of file type groupings and sorting mechanisms to effortlessly customize a hierarchical view. Organizing a list of your own files has never been so easy and simply presented.

Problem Statement:

With file storage becoming seemingly endless, the necessity to clean up and delete less frequently used files has diminished. Files are now kept forever and their location easily forgotten. Currently, there is no programming language designed to help ease the burden of remembering file location or organizing files into a personalized representation. This has resulted in enormous amounts of wasted time fiddling with specialized **ls**, **grep**, and **find** commands.

FJL Solution:

A simple, useful organization tool for users concerned with keeping track of files living throughout their filesystem hierarchy. The FJL programming language will provide mechanisms to view you filesystem hierarchy in a personalized manner. Grouping of files and sorting through files by size or files between a particular threshold has never been easier.

Features:

Building FJL meant providing data types and mechanisms to make filesystem information readily and simply available to a programmer.

FJL developers envisioned that a mere function call should read in a permissible directory or recursed set of directories. The FJL language supplies this mechanism in the standard library. The standard library provides extremely useful directed functions.

The data types in the programming language were carefully crafted to supply the programmer with the all the file information he/she could ever have use for.

Portability:

FJL is an interpreted language, that was developed to be portable to all versions of Linux and Unix. Develop in Linux, and execute it on your old Solaris machine, without any issues.

Details:

FJL provides several functions to enable easy access to the filesystem. A call to **readdir** given a directory will read in the directory specified and assign it to our **dir** type. A call to **rreaddir** given a directory would recursively read in the specified directory and assign it to a special **rdir** type. Stack mechanisms, including but not limited to, **push** and **pop** have been provided to manipulate **dir** types.

FJL has also crafted several language specific data types. Some of the most useful being the **file** and **dir** data types. The **file** data type has attributes "name", "time" and "size" that provide an enormously simple way of accessing details of files. While the **dir** data type serves as a container for **file** types, it has it's own set attributes "nof" (number of files), "lt" (last touched), and "name" to give directory specific information.

FJL also provides flow control with the usage of **while** loops and **if** statements, which provide the looping feature needed to access individual file attributes. Comparison operators are also made available.

Examples:

A brief example of the simplicity of the code is shown below. The example below shows how easy it is read in the current directory. The example also groups by a particular file type that has size over a certain threshold.

```
/***Start***/

int FILESIZE_THRESHOLD = 10240
dirname homeworkdir = "/home/user/homework"
dir dir_contents = readir(homeworkdir)
dir fs_dir

function dir FileNameGroup(dir dir_contents){
  fs_dir.name = “Text Files”
```

```
  fileext text = "txt"
  while ( dir_contents.hasmore ){
    file x = pop(dir_contents)
    if ( x.name.hasExt(fileext) && x.size >
     FILESIZE_THRESHOLD )
     push(fs_dir,x)
    }
  }
  return fs_dir
}

dir retval = FileNameGroup(dir_contents)
print retval.name
print retval

/***End***/
```

Conclusion:

The benefits of FJL are easily discussed. FJL provides a elegant solution to a specific problem that will continue to bug and bother users with growing storage and an unwillingness to clean up their user space.