Simple File System (SFS) Format

Version SFS-V00.01

The SFS is a simple file system format optimized for writing and for low overhead.. The advantages of this format are:

- Event navigation is possible using simple content-independent file system like functions.
- Very low overhead. No loss due to block size granularity
- Entire valid file system can be created by appending content

On the other hand, random access directory navigation is rather slow because there is no built-in indexing or directory hierarchy. For a 500MB file system containing files with 5k bytes this represents an initial search overhead of ~1-2 sec (~100,000 seeks).

SFS Structure

The structure of a SFS file is as follows

VolumeSpec	
Head	
File1	File1 Binary Data
File2	File2 Binary Data
...	...
Tail	

VolumeSpec: This is simply a 12 byte character string representing filesystem version. For example: “SFS V00.01”

Head: This is a short header record. The byte order only applies to the time field of this record.

type = “HEAD”
byte_order = 0x04030201
time

File: The File records are a variable length record containing information about a file.

type = “FILE”
byte_order = 0x04030201
Sz

head_sz	attr	reserved
name....		
name (continued)....		

"byte_order" corresponds only to this header. The endiness of the data is undefined by SFS

"sz" corresponds to the datafile size. This may be any number, but the file itself will be padded to take up a multiple of 4 bytes

"head_sz" this must be a multiple of 4

"attr"

SFS_ATTR_INVALID: file deleted
SFS_ATTR_PUSHDIR: push current path to path stack
SFS_ATTR_POPDIR: pop current path from path stack
SFS_ATTR_NOCD: this record doesn't reset the basedir

"name" the name of the file.

Tail: The tail record has the same format as the "HEAD" record, but type="TAIL". This record is not needed, but can be present to represent that the file was properly closed.

SFS Filename Convention

Because the SFS file simply contains a list of file descriptors and binary data, paths may be thought of as adhoc creations of the SFS reader code. However the following conventions are applied so that one can reconstruct a normal directory structure from a SFS file.

/xxx/xxx/yyy	/xxx/xxx/ is the "directory part" yyy is the "file" part
/xxx	absolute path
xxx or xxx/xxx	path is relative to the directory part of the previous entry
xxx/	Directory
0 length	Zero length files are used to push/pop the current base path from a stack. The files attr field should be set to SFS_ATTR_PUSHDIR or SFS_ATTR_POPDIR

Because there need be no explicit directory entries, a directory is defined to exist if it is part of any existing file's path.

It is perfectly legal to have a "directory" which also contains data.