

**University of Pennsylvania**  
**Department of Electrical and Systems Engineering**  
**Digital Audio Basics**

ESE250 Spring 2012

Lab 10: Application-Specific File Systemw

Tuesday, March 27, 2012

---

**For Lab Session:** Tuesday, March 27, 2012 in Detkin Lab.

**Due:** Monday, April 2, 2012 by 12:00pm (noon).

**Collaboration:** Work in lab in teams of 2. Perform individual writeups. See course collaboration policy in the [Administrative handout](#).

**Objective:** Design a file system to support two different views of a set of media files.

**Prelab Requirements:** download and read this entire lab assignment. *Optional:* Bring your headphones. Bring two .wav file at least two minutes long for use in encode benchmarking.

### Deliverables

- Describe your solution approach.
- For the example set of songs in Section 1.1, provide a diagram similar to the one we provide in Section 1.2.2 showing how you represent the directory structure for the songs.
- Show pseudocode for your implementation of:
  - `songBNode ← Selector()`
  - `AddSong(artist, album, title, songBytes[])`

**handin:** All labs will be turned in electronically through the [Penn Blackboard](#) website. Go to the assignment submission link and follow the instructions. Your writeup should be a **PDF file**.

**Exit Ticket:** Explain your implementation to the TA.

## Motivation

A modern MP3 player allows you to browse your songs in several different ways. Common examples include:

- alphabetically by title
- alphabetically by album title
- alphabetically by artist

This requires that a the file system be organized in a way that allows easy browsing using any of these ways. In this lab you will design a file system that allows browsing by artist and by album.

## Lab Procedure Guide

### 1.1 Design Problem

We would like to be able to browse and select music by:

1. browse artists, then browse albums by that artists, then by title within the album (for the selected artists)
2. browse all albums, then all titles on the selected album (for multi-artist albums, the title list for case one will be different from the list in case two)

You may assume:

- album names are unique
- songs are assigned to a single artist

To help you out, we have provided an example song list that you should design for.

Figure 1 and 2 show one possible set of interactions for case one and both cases respectively. The full interaction graphs are shown in Figure 3 and 4 at the end of this handout.

### 1.2 Case One Solution

As a starter and example, we detail and implementation for the first case above.

Artist	Album	Title
Queen	Queen's Greatest hits	Bohemian Rhapsody
Queen	Queen's Greatest hits	We Will Rock You
Queen	Queen's Greatest hits	We Are The Champions
Queen	Queen's Greatest hits	A Kind of Magic
The Cyrkle	Superhits 1966	Red Rubber Ball
The Monkees	Superhits 1966	Last Train to Clarksville
Beach Boys	Superhits 1966	Sloop John B
Daydream	Superhits 1966	Lovin' Spoonful
The Monkees	The Monkees Greatest hits	Monkees Theme
The Monkees	The Monkees Greatest hits	I'm A Believer
The Monkees	The Monkees Greatest hits	Daydream Believer
The Monkees	The Monkees Greatest hits	Pleasant Valley Sunday
Beach Boys	Pet Sounds	Wouldn't It Be Nice
Beach Boys	Pet Sounds	God Only Knows
Beach Boys	Pet Sounds	I Know There's An Answer
Beach Boys	Pet Sounds	Pet Sounds

Table 1: Example song list

### 1.2.1 Description

Our solution is simply to represent the necessary structure directly as a directory hierarchy. At the top level, we have a directory of artists. In each artist directory, we place a directory of albums by that artist. In each of these album directories, we finally have a directory of the actual songs that the specified artist has on the album. When a new song is added, we create directory nodes as needed for artist and album, then link the song from the album directory.

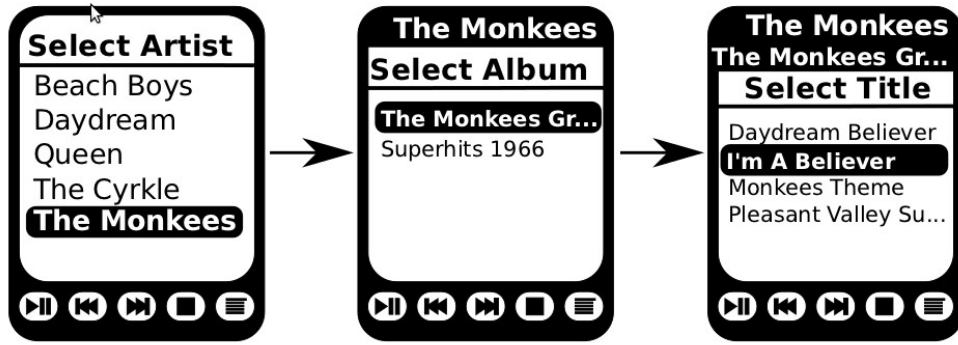


Figure 1: One possible interaction path for case one operating on the songs above.



Figure 2: One possible interaction path for both cases operating on the songs above.

### 1.2.2 Representation

Representation of BNodes used to solve case one.

Root Type:Dir									
Name	Node								
Beach Boys	2								
Daydream	3								
Queen	4								
The Cyrkle	5								
The Monkees	6								

2 Type:Dir		7 Type:Dir		11 Type:Dir		14 Type:MP3		22 Type:MP3			
Name	Node	Name	Node	Name	Node	Data Blocks		Data Blocks			
Pet Sounds	7	God Only Knows	14	Red Rubber Ball	24	30 31	...	37	93 94	...	99
Superhits 1966	8	I Know There's An Answer	15								
		Pet Sounds	16								
		Wouldn't It Be Nice	17								

3 Type:Dir		8 Type:Dir		12 Type:Dir		15 Type:MP3		23 Type:MP3			
Name	Node	Name	Node	Name	Node	Data Blocks		Data Blocks			
Superhits 1966	9	Sloop John B	18	Last Train to Clarksville	25	38 39	...	45	100 101	...	106

4 Type:Dir		9 Type:Dir		13 Type:Dir		16 Type:MP3		24 Type:MP3			
Name	Node	Name	Node	Name	Node	Data Blocks		Data Blocks			
Queen's Greatest Hits	10	Lovin' Spoonful	19	Daydream Believer	26	46 47	...	56	107 108	...	112
				I'm A Believer	27						
				Monkees Theme	28						
				Pleasant Valley Sunday	29						

5 Type:Dir		10 Type:Dir		17 Type:MP3		18 Type:MP3		25 Type:MP3				
Name	Node	Name	Node	Data Blocks		Data Blocks		Data Blocks				
Superhits 1966	11	A Kind of Magic	20	57 58	...	69	70 71	...	77	113 114	...	117

6 Type:Dir		20 Type:MP3		19 Type:MP3		26 Type:MP3		27 Type:MP3			
Name	Node	Data Blocks		Data Blocks		Data Blocks		Data Blocks			
Superhits 1966	12	83 84	...	86	78 79	...	82	124 125	...	130	
The Monkees	13										
Greatest Hits	13										

7 Type:Dir		21 Type:MP3		28 Type:MP3		29 Type:MP3				
Name	Node	Data Blocks		Data Blocks		Data Blocks				
We Will Rock You	23	87 88	...	92	131 132	...	142	143 144	...	158

### 1.2.3 Assumed Lower-Level Routines

Assume the following low-level routines exist and can be used in your design. Talk with your TA if you find there is a function you need that is missing.

- $stringList \leftarrow GetKeyStringList(directoryBNode)$  Given a  $directoryBNode$ , returns a list of items in the directory.
- $string \leftarrow DisplayChooseString(stringList)$  Displays a  $stringList$  and returns the chosen item.
- $bNode \leftarrow GetValueForKey(directoryBNode, string)$  Given a  $directoryBNode$  and a  $string$  corresponding to an item in the directory, returns the  $BNode$  for that item in that directory. If the string does not exist, returns a designated  $bNode$  token indicating the non-presence of the string.
- $directoryBNode \leftarrow NewDirectoryBNode()$  Creates a new  $BNode$  of type directory and returns the created  $BNode$ .
- $InsertInOrder(directoryBNode, string, bNode)$  Inserts into  $directoryBNode$  the item called  $string$  with corresponding  $bNode$ . Maintains alphabetical order within  $directoryBNode$ .
- $bNode \leftarrow NewDataBNode(length, type)$  Creates a new data  $BNode$  of  $length$  bytes and of type  $type$ .
- $StoreIntoBNode(bNode, bytes[])$  Stores the array bytes  $bytes[]$  representing data into  $bNode$ .

You may assume that functions `InsertInOrder` and `StoreIntoBNode` internally handle creating or expanding trees of `BNodes` to accommodate the data.

### 1.2.4 Implementation

- songBNode* ← `Selector()`

```
//Select a song by choosing artist then album and then title
//Choose artist
artistList ← GetKeyStringList(rootArtistBNode)
selectedArtist ← DisplayChooseString(artistList)
artistBNode ← GetValueForKey(rootArtistBNode, selectedArtist)

//Choose album for chosen artist
albumList ← GetKeyStringList(artistBNode)
selectedAlbum ← DisplayChooseString(albumList)
albumBNode ← GetValueForKey(artistBNode, selectedAlbum)

//Choose song for chosen album
titleList ← GetKeyStringList(albumBNode)
selectedTitle ← DisplayChooseString(titleList)
songBNode ← GetValueForKey(albumBNode, selectedTitle)
```
- `AddSong(artist, album, title, songBytes[])`

```
//Create BNodes for artis, album and title
//if necessary and store song in title BNode
artistBNode ← GetValueForKey(rootArtistBNode, artist)//get artist BNode
if (NotPresent(artistBNode))//check if artistBNode exists
    //artistBNode does not exist,
    //create it and add it to rootArtistBNode
    artistBNode ← NewDirectoryBnode()
    InsertInOrder(rootArtistBNode, artist, artistBNode)

albumBNode ← GetValueForKey(artistBNode, album)//get album BNode
if (NotPresent(albumBNode))//check if albumBNode exists
    //albumBNode does not exist,
    //create it and add it to artistBNode
    albumBNode ← NewDirectoryBnode()
    InsertInOrder(artistBNode, album, albumBNode)

titleBNode ← GetValueForKey(albumBNode, title)//get title BNode
if (NotPresent(titleBNode))//check if titleBNode exists
```

```
//titleBNode does not exist, create it
titleBNode ← NewDataBnode(songBytes.length, mp3)
InsertInOrder(albumBNode, title, titleBNode)
StoreIntoBNode(titleBNode, songBytes)//store songBytes in titleBNode
```

## Lab Writeup Guidelines

### Theory: Pre-lab

Assume:

- an 8GB solid-state drive with a read bandwidth of 100MB/s.
  - the drive primarily holds MP3 files for songs; a typical song is 3 minutes long.
  - the MP3 files are encoded at 128Kbits/s.
1. how many songs can the drive hold?
  2. how many hours of unique playtime?
  3. how long does it take to read through the entire contents of the solid-state drive?
  4. Would it be reasonable to perform this operation to list the contents of the drive?
  5. About how many bytes would it take to store the title, artist, and album name for a song? [State necessary assumptions.] how does this compare to the length of the encoded song?
  6. Assuming you stored this information (title, artist, song) along with an 8 Byte address pointer for every song in one large index file, how much data would this require? how long would it take to read this data from the disk?
  7. Assuming the consumer keeps the device for 3 years and listens to music on average 2 hours per day every day of the year, how many times (on average) is each song played?
  8. If you (the engineer designing the file system for this consumer component) have a choice of performing: (a) more work when the song is initially loaded onto the machine to accelerate consumer selection or (b) minimal work on load but more work when the consumer tries to find the song to play it, what does your answer to 7 suggest about making this choice?

### Design

**Your task is to create an implementation that supports both of the cases above.** You can modify our example and extend it to implement both cases, or you can create a completely new implementation that incorporates both cases.

## Further Thought

*This section is **not** part of your required assignment. Along with each week, we will offer directions and questions for further thought. Due to the nature of this course, we can only begin to glimpse the depth and richness of each of the topic areas. These questions offer some headings to contemplate further depth. These will often be open-ended questions.* Consider a file system that allows files to be deleted, edited, and overwritten. how would you provide the ability to recover deleted, overwritten, or older versions of files that were less than a user configurable amount of time old (*e.g.* 1 week, 1 month, 1 year)? how can this be done efficiently in the case where a series of small edits (*e.g.* <1KB) are performed on a large file (*e.g.* >1 MB)?



Figure 3: Full interaction graph for case one operating on the example song list.



Figure 4: Full interaction graph for both cases operating on the example song list.