File Coordination for OS X and iCloud

Jim Dovey
http://alanquatermain.net

Files Are Hard

Files Are Hard

- Anything can write to a file that's in use.

Files Are Hard

- Anything can write to a file that's in use.
- Reads and writes can happen at any time.

Files Are Hard

- Anything can write to a file that's in use.
- Reads and writes can happen at any time.
- Files can be moved or deleted while in use.

Files Are Hard

- Anything can write to a file that's in use.
- Reads and writes can happen at any time.
- Files can be moved or deleted while in use.
- A file's ancestors could be moved or renamed.

Wednesday, 12 September, 12

App A: Read File

“Hello”

App A: Read File

App B: Read File

“Hello”

“Hello”

App A: Read File

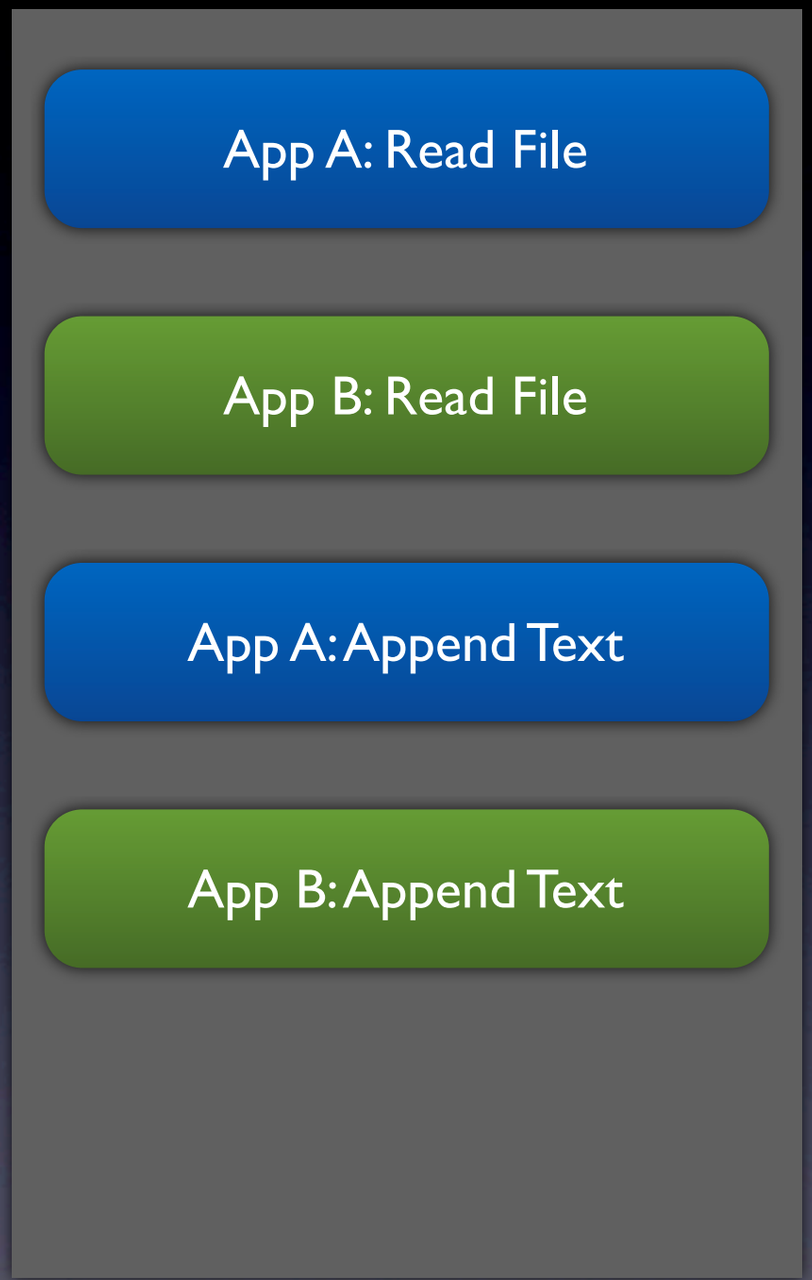
App B: Read File

App A: Append Text

“Hello”

“Hello”

“Hello everyone”



"Hello"

"Hello"

"Hello everyone"

"Hello, World!"

App A: Read File

App B: Read File

App A: Append Text

App B: Append Text

“Hello”

“Hello”

“Hello everyone”

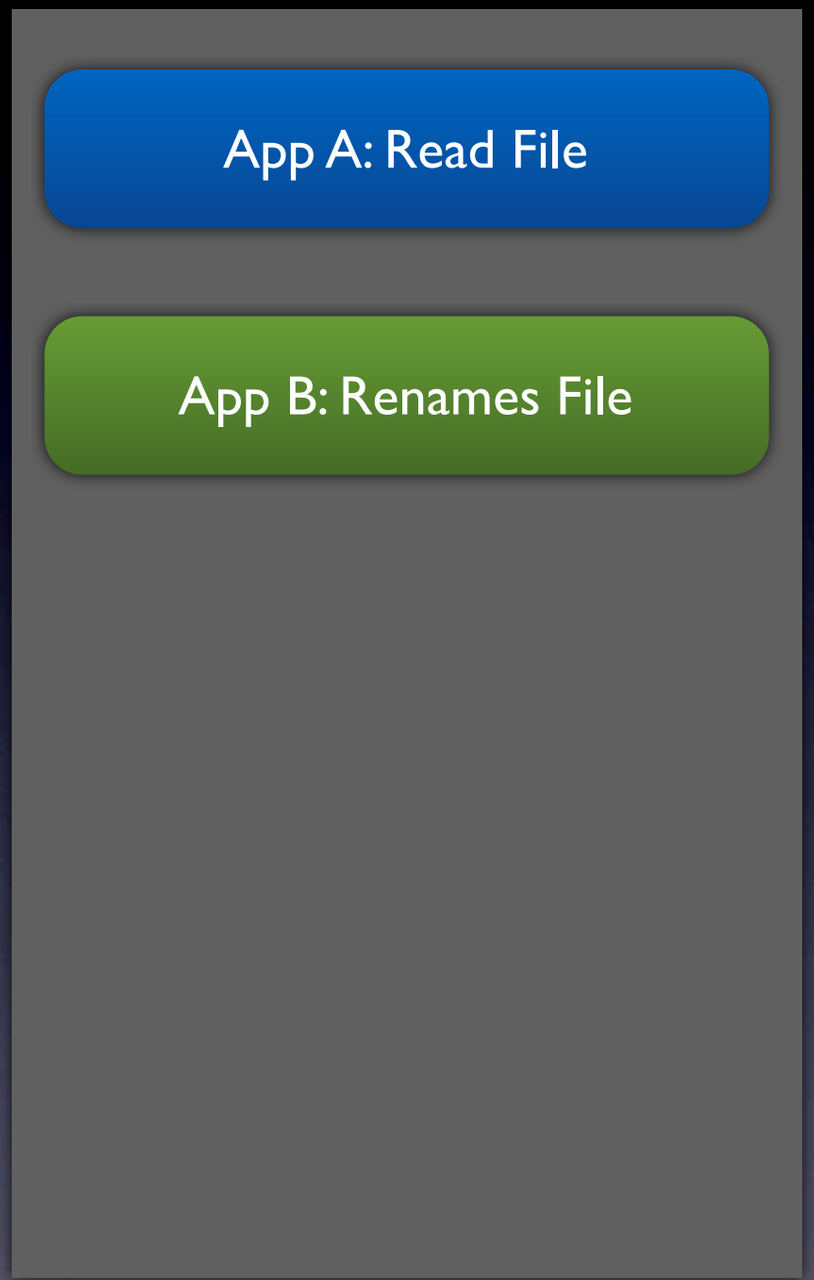
OVERWRITTEN!

“Hello, World!”

Wednesday, 12 September, 12

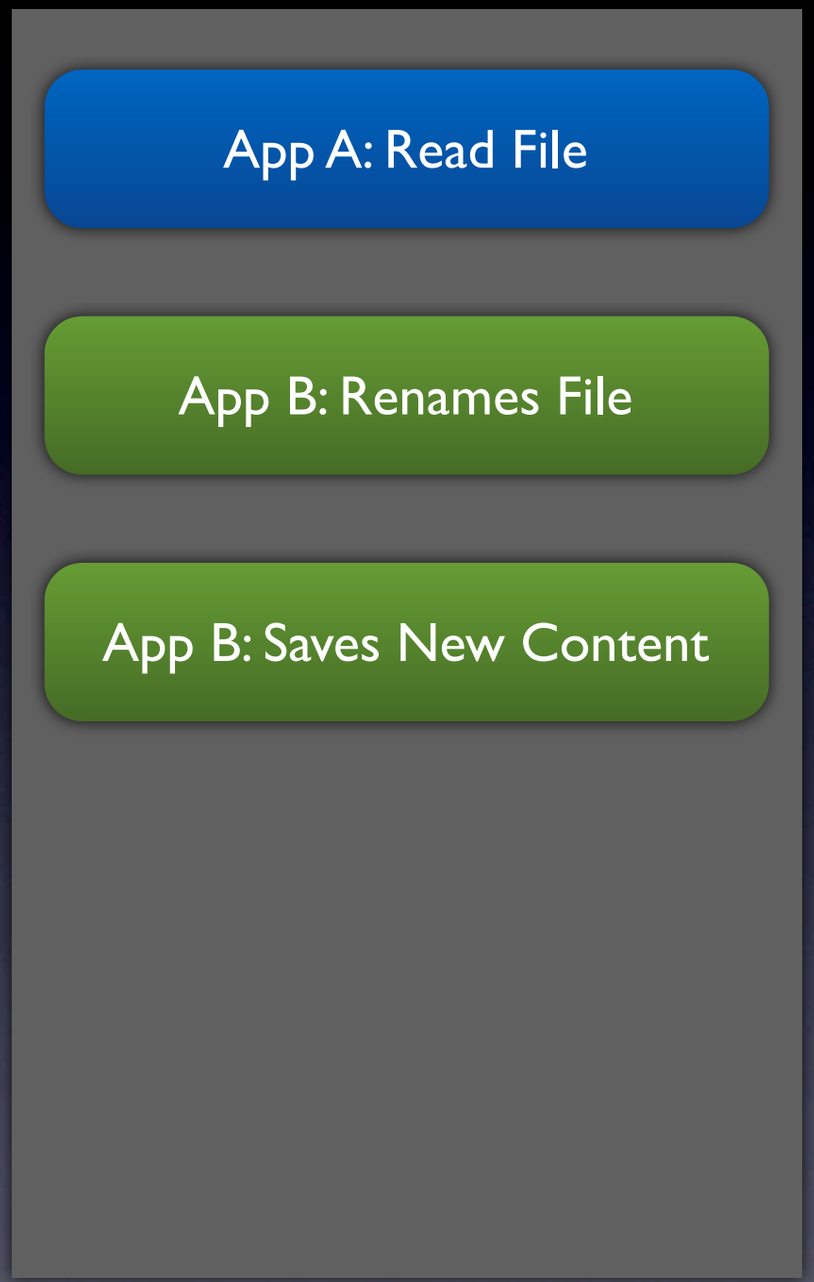
App A: Read File

~/Documents/Project A/Chapter 1



~/Documents/Project A/Chapter 1

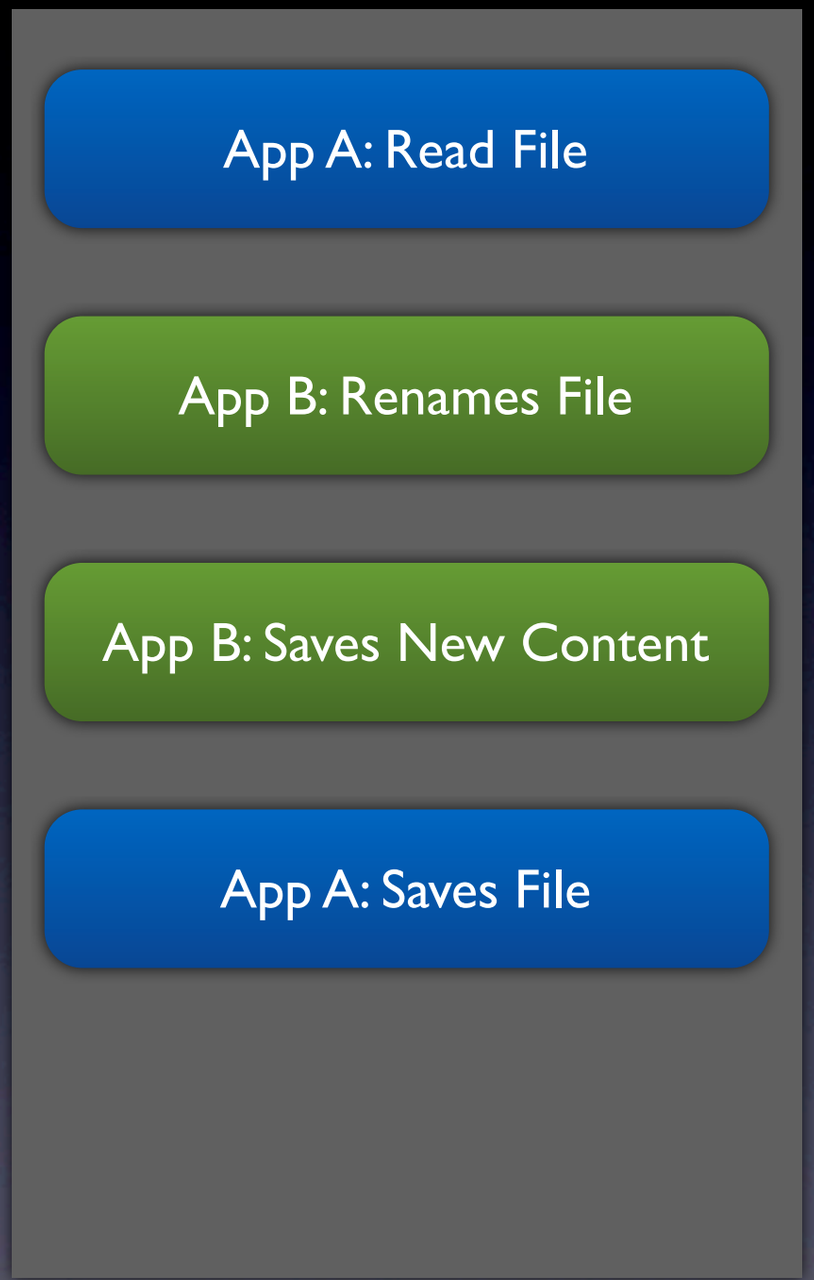
~/Documents/Project A/Introduction



~/Documents/Project A/Chapter 1

~/Documents/Project A/Introduction

~/Documents/Project A/Chapter 1



~/Documents/Project A/Chapter 1

~/Documents/Project A/Introduction

~/Documents/Project A/Chapter 1

App A: Read File

App B: Renames File

App B: Saves New Content

App A: Saves File

~/Documents/Project A/Chapter 1

~/Documents/Project A/Introduction

~/Documents/Project A/Chapter 1

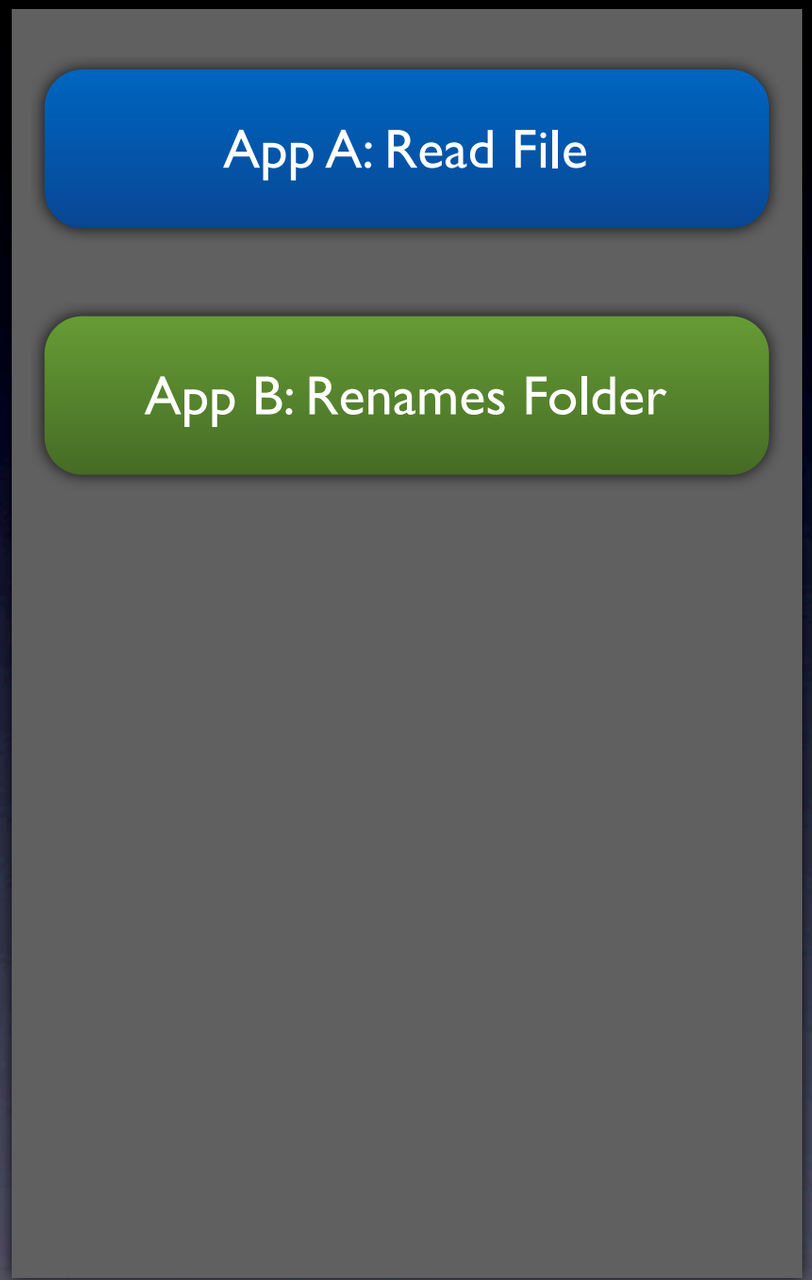
OVERWRITTEN!

~/Documents/Project A/Chapter 1

Wednesday, 12 September, 12

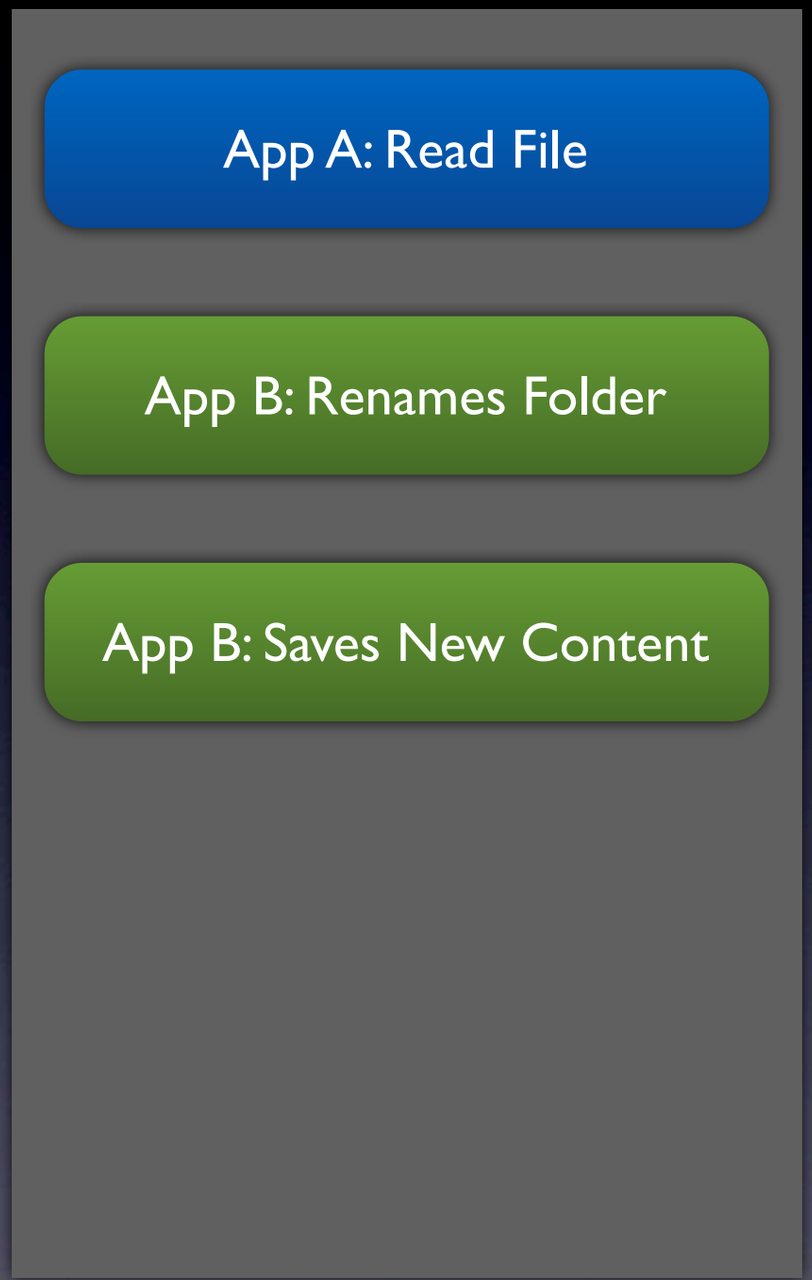
App A: Read File

~/Documents/Project A/Chapter 1



~/Documents/Project A/Chapter 1

~/Documents/Project B/Chapter 1



~/Documents/Project A/Chapter 1

~/Documents/Project B/Chapter 1

~/Documents/Project B/Chapter 1

App A: Read File

~/Documents/Project A/Chapter 1

App B: Renames Folder

~/Documents/Project B/Chapter 1

App B: Saves New Content

~/Documents/Project B/Chapter 1

App A: Saves File

~/Documents/Project A/Chapter 1

App A: Read File

App B: Renames Folder

App B: Saves New Content

App A: Saves File

~/Documents/Project A/Chapter 1

~/Documents/Project B/Chapter 1

~/Documents/Project B/Chapter 1

~/Documents/Project A/Chapter 1

ERROR!

Mitigation: NSURL

Mitigation: NSURL

- Can provide file-ID access to files, avoiding path-centric problems.

Mitigation: NSURL

- Can provide file-ID access to files, avoiding path-centric problems.
- BUT: requires explicit setup; URLs are still path-based by default, and a file must exist on the filesystem to get a file-ID URL to it.

Mitigation: NSURL

```
- (void) setFileURL: (NSURL *) url
{
        _url = [url fileReferenceURL];
}
```

Files Are Hard

- Anything can write to a file that's in use.
- Reads and writes can happen at any time.
- ~~Files can be moved or deleted while in use~~.
- ~~A file's ancestors could be moved or renamed~~.

The Real Solution:

Intentions

File Coordinator

App 1

Read Docs/File.txt

Write Docs/File.txt

App 2

Read Docs/File.txt

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

App 1

Read Docs/File.txt

Write Docs/File.txt

App 2

Read Docs/File.txt

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

Read Docs/File.txt

App 1

Write Docs/File.txt

App 2

Read Docs/File.txt

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

Read Docs/File.txt

Read Docs/File.txt

App 1

Write Docs/File.txt

App 2

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

App 1

Write Docs/File.txt

App 2

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

Read Docs/File.txt

Write Docs/File.txt

App 2

Read Docs/File.txt

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

App 1

Read Docs/File.txt

Write Docs/File.txt

App 2

Read Docs/File.txt

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

Read Docs/File.txt

App 1

Read Docs/File.txt

Write Docs/File.txt

App 2

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

App 1

Read Docs/File.txt

Write Docs/File.txt

App 2

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

App 1

Read Docs/File.txt

Write Docs/File.txt

App 2

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

App 1

Read Docs/File.txt

Write Docs/File.txt

App 2

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

Read Docs/File.txt

App 1

Write Docs/File.txt

App 2

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

Read Docs/File.txt

App 1

Write Docs/File.txt

App 2

Save Changes

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

Read Docs/File.txt

App 1

Write Docs/File.txt

App 2

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

App 1

Write Docs/File.txt

App 2

App 3

Move Docs to MyDocs

File Coordinator

App 1

Read Docs/File.txt

Write Docs/File.txt

App 2

Read Docs/File.txt

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

App 1

Write Docs/File.txt

App 2

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

Write Docs/File.txt

App 1

App 2

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

Write Docs/File.txt

App 1

App 2

Relinquish For Write

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

Write Docs/File.txt

Retrieve Access

App 1

App 2

Relinquish For Write

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

Write Docs/File.txt

Retrieve Access

App 1

App 2

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

Retrieve Access

App 1

App 2

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

App 1

App 2

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

Read Docs/File.txt

Write Docs/File.txt

App 2

Read Docs/File.txt

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

App 1

Write Docs/File.txt

App 2

Update Contents

App 3

Move Docs to MyDocs

File Coordinator

App 1

App 2

Move Docs to MyDocs

App 1

Write Docs/File.txt

App 2

Update Contents

App 3

File Coordinator

App 1

App 2

Move Docs to MyDocs

App 1

Relinquish For Write

Write Docs/File.txt

App 2

Relinquish For Write

Update Contents

App 3

File Coordinator

App 1

App 2

Move Docs to MyDocs

Retrieve Access

Retrieve Access

App 1

Relinquish For Write

Write Docs/File.txt

App 2

Relinquish For Write

Update Contents

App 3

File Coordinator

App 1

App 2

Move Docs to MyDocs

Retrieve Access

Retrieve Access

App 1

Write Docs/File.txt

App 2

Update Contents

App 3

File Coordinator

App 1

App 2

Retrieve Access

Retrieve Access

App 1

Write Docs/File.txt

App 2

Update Contents

App 3

File Coordinator

App 1

App 2

Retrieve Access

Retrieve Access

App 1

Item Moved

Write Docs/File.txt

App 2

Item Moved

Update Contents

App 3

File Coordinator

App 1

App 2

Retrieve Access

Retrieve Access

App 1

Write Docs/File.txt

App 2

Update Contents

App 3

File Coordinator

App 1

App 2

App 1

Write Docs/File.txt

App 2

Update Contents

App 3

File Coordination

File Coordination

- Used to perform common filesystem operations.

File Coordination

- Used to perform common filesystem operations.
- Call the API passing a block which does the work.

File Coordination

- Used to perform common filesystem operations.
- Call the API passing a block which does the work.
- When appropriate, your block is invoked to perform its task.

File Coordination

- APIs available for files and folders.
- Specific routines for reading, writing, updating, replacing, and moving.
- Coordinates these actions *system-wide*.

File Coordination

- Coalesces operations on the same object into a single group to avoid disk thrashing.

File Coordination

- Coalesces operations on the same object into a single group to avoid disk thrashing.
- Discrete random-access calls may be coalesced with respect to other apps working on other files.

File Presentation

File Presentation

File Presentation

- When you're keeping track of a file's contents, you register a *file presenter* with the system.

File Presentation

- When you're keeping track of a file's contents, you register a *file presenter* with the system.
- You're advised whenever something else wants to access the file, and given a chance to clean up.

File Presentation

- When you're keeping track of a file's contents, you register a *file presenter* with the system.
- You're advised whenever something else wants to access the file, and given a chance to clean up.
- You're passed a block to run when you're done, and you can hand back a block to be notified when the operation is complete.

File Presentation

- Commonly you'll see the following sequence:

File Presentation

- Commonly you'll see the following sequence:
- Relinquish control

File Presentation

- Commonly you'll see the following sequence:
  - Relinquish control
  - Write/move/change occurred

File Presentation

- Commonly you'll see the following sequence:
  - Relinquish control
  - Write/move/change occurred
  - Retrieve Control

Le Code

Adopting NSFilePresenter

```
@interface MyViewer : NSWindowController <NSFilePresenter>
...
@property (readonly) NSURL *presentedItemURL;
@property (readonly) NSOperationQueue
                              *presentedItemOperationQueue;
...
@end
```

Implementation

```
- (void)applicationDidFinishLaunching:(NSNotification *)note
{
    _presentationQueue = [NSOperationQueue new];
    [_presentationQueue setMaxConcurrentOperationCount: 1];

    [NSFileCoordinator addFilePresenter: self];
}
- (void) applicationWillTerminate: (NSNotification *)
notification
{
    [NSFileCoordinator removeFilePresenter: self];
}
- (NSURL *) presentedItemURL
{
    return ( _fileURL );
}
- (NSOperationQueue *) presentedItemOperationQueue
{
    return ( _presentationQueue );
}
```

Implementation

```
- (void) relinquishPresentedItemToReader:
                (void (^)(void (^reacquirer)(void))) reader
{
    _relinquished = YES;
    reader(^{
        _relinquished = NO;
        if ( _hasDeferredWrites )
            [self saveChangesWithCompletion: nil];
        else if ( _hasDeferredReads )
            [self updateViewer];
    });
}
```

Implementation

```
- (void) relinquishPresentedItemToWriter:
                (void (^)(void (^reacquirer)(void))) writer
{
    _relinquished = YES;
    writer(^{
        _relinquished = NO;
        if ( _hasDeferredWrites )
            [self saveChangesWithCompletion: nil];
        else if ( _hasDeferredReads )
            [self updateViewer];
    });
}
```

Please Note…

Please Note…

- You do not save changes or read from the file *at any point* as a result of relinquishing or reclaiming a presented file.

Please Note…

- You do not save changes or read from the file *at any point* as a result of relinquishing or reclaiming a presented file.
- If saving your changes is interesting to the reader, *you will be told to do so.*

Please Note…

- You do not save changes or read from the file *at any point* as a result of relinquishing or reclaiming a presented file.
- If saving your changes is interesting to the reader, *you will be told to do so*.
- If the other app makes changes *you will be told about them*.

Saving Your Changes

```
- (void) savePresentedItemChangesWithCompletionHandler: (void
(^)(NSError *)) completionHandler
{
    NSError * error = nil;
    if ( [self hasChanges] )
    {
        [self _writeChangesInternal: self.fileURL
                              error: &error];
    }

    completionHandler(error);
}
```

Saving Your Changes

```
- (void) savePresentedItemChangesWithCompletionHandler: (void
(^)(NSError *)) completionHandler
{
    NSError * error = nil;
    if ( [self hasChanges] )
    {
        [self _writeChangesInternal: self.fileURL
                              error: &error];
    }

    completionHandler(error);
}
```

Saving Your Changes

```
- (void) savePresentedItemChangesWithCompletionHandler: (void
(^)(NSError *)) completionHandler
{
    NSError * error = nil;
    if ( [self hasChanges] )
    {
        [self _writeChangesInternal: self.fileURL
                              error: &error];
    }

    completionHandler(error);
}
```

Reading Other Changes

```
- (void) presentedItemDidChange
{
    NSFileCoordinator * coordinator = [[NSFileCoordinator
alloc] initWithFilePresenter: self];
    [coordinator coordinateReadingItemAtURL: self.fileURL
options: NSFileCoordinatorReadingWithoutChanges error: NULL
byAccessor: ^(NSURL *newURL) {
        NSDate * modDate = nil;
        if ( [newURL getResourceValue: &modDate forKey:
NSURLContentModificationDateKey error: NULL] == NO )
            return;
        if ( modDate == nil )
            return;
        if ( _lastModDate == nil ||
            [_lastModDate laterDate: modDate] == modDate )
        {
            // read the file
        }
    }];
}
```

Reading Other Changes

```
- (void) presentedItemDidChange
{
    NSFileCoordinator * coordinator = [[NSFileCoordinator
alloc] initWithFilePresenter: self];
    [coordinator coordinateReadingItemAtURL: self.fileURL
options: NSFileCoordinatorReadingWithoutChanges error: NULL
byAccessor: ^(NSURL *newURL) {
        NSDate * modDate = nil;
        if ( [newURL getResourceValue: &modDate forKey:
NSURLContentModificationDateKey error: NULL] == NO )
            return;
        if ( modDate == nil )
            return;
        if ( _lastModDate == nil ||
            [_lastModDate laterDate: modDate] == modDate )
        {
            // read the file
        }
    }];
}
```

Reading Other Changes

```
- (void) presentedItemDidChange
{
    NSFileCoordinator * coordinator = [[NSFileCoordinator
alloc] initWithFilePresenter: self];
    [coordinator coordinateReadingItemAtURL: self.fileURL
options: NSFileCoordinatorReadingWithoutChanges error: NULL
byAccessor: ^(NSURL *newURL) {
        NSDate * modDate = nil;
        if ( [newURL getResourceValue: &modDate forKey:
NSURLContentModificationDateKey error: NULL] == NO )
            return;
        if ( modDate == nil )
            return;
        if ( _lastModDate == nil ||
            [_lastModDate laterDate: modDate] == modDate )
        {
            // read the file
        }
    }];
}
```

Reading Other Changes

```
- (void) presentedItemDidChange
{
    NSFileCoordinator * coordinator = [[NSFileCoordinator
alloc] initWithFilePresenter: self];
    [coordinator coordinateReadingItemAtURL: self.fileURL
options: NSFileCoordinatorReadingWithoutChanges error: NULL
byAccessor: ^(NSURL *newURL) {
        NSDate * modDate = nil;
        if ( [newURL getResourceValue: &modDate forKey:
NSURLContentModificationDateKey error: NULL] == NO )
            return;
        if ( modDate == nil )
            return;
        if ( _lastModDate == nil ||
            [_lastModDate laterDate: modDate] == modDate )
        {
            // read the file
        }
    }];
}
```

L'Example

What about iCloud?

iCloud Support

- Place your files inside your application's ubiquity container.

Move To iCloud

```
- (NSURL *) putFileInTheCloud: (NSURL *) url
                        error: (NSError **) error
{
    NSURL * containerURL = [[NSFileManager defaultManager]
                    URLForUbiquityContainerIdentifier: nil];
    NSURL * dest = [containerURL URLByAppendingPathComponent:
                                        [url lastPathComponent]];
    if ( [[NSFileManager defaultManager] setUbiquitous: YES
                                             itemAtURL: url
                                        destinationURL: dest
                                                 error: error] == NO )
    {
        return ( nil );
    }

    return ( dest );
}
```

Move To iCloud

```
- (NSURL *) putFileInTheCloud: (NSURL *) url
                        error: (NSError **) error
{
    NSURL * containerURL = [[NSFileManager defaultManager]
                        URLForUbiquityContainerIdentifier: nil];
    NSURL * dest = [containerURL URLByAppendingPathComponent:
                                        [url lastPathComponent]];
    if ( [[NSFileManager defaultManager] setUbiquitous: YES
                                 itemAtURL: url
                            destinationURL: dest
                                     error: error] == NO )
    {
        return ( nil );
    }

    return ( dest );
}
```

Move To iCloud

```
- (NSURL *) putFileInTheCloud: (NSURL *) url
                        error: (NSError **) error
{
    NSURL * containerURL = [[NSFileManager defaultManager]
                        URLForUbiquityContainerIdentifier: nil];
    NSURL * dest = [containerURL URLByAppendingPathComponent:
                                        [url lastPathComponent]];
    if ( [[NSFileManager defaultManager] setUbiquitous: YES
                                             itemAtURL: url
                                        destinationURL: dest
                                                 error: error] == NO )
    {
        return ( nil );
    }

    return ( dest );
}
```

iCloud Support

- Place your files inside your application's ubiquity container.
- Get the new URL and make it your `presentedItemURL`.

Move To iCloud

```
- (BOOL) goToTheCloud: (NSError **) error
{
    NSURL * newURL = [self putFileInTheCloud: _url
                                       error: error];
    if ( newURL == nil )
        return ( NO );

    [self willChangeValueForKey: @"presentedItemURL"];
    _url = [newURL copy];
    [self didChangeValueForKey: @"presentedItemURL"];
}
```

Move To iCloud

```
- (BOOL) goToTheCloud: (NSError **) error
{
    NSURL * newURL = [self putFileInTheCloud: _url
                                       error: error];
    if ( newURL == nil )
        return ( NO );

    [self willChangeValueForKey: @"presentedItemURL"];
    _url = [newURL copy];
    [self didChangeValueForKey: @"presentedItemURL"];
}
```

iCloud Support

- Place your files inside your application's ubiquity container.
- Get the new URL and make it your `presentedItemURL`.
- Obey the file coordinator.

iCloud Support

- Place your files inside your application's ubiquity container.
- Get the new URL and make it your `presentedItemURL`.
- Obey the file coordinator.
  - It will ask you to save before it merges cloud datae.

iCloud Support

- Place your files inside your application's ubiquity container.
- Get the new URL and make it your `presentedItemURL`.
- Obey the file coordinator.
  - It will ask you to save before it merges cloud datae.
  - It will tell you when the content changes.

File Coordination for OS X and iCloud

Sample code is available on github:
http://github.com/alanQuatermain/filesync-talk/

Jim Dovey
http://alanquatermain.net