iPhone & iPad

プログラミング Bible［下］

河西 朝雄著

Xcodeという統合開発環境を使用し、
Objective-Cというプログラミング言語を使って
iPhone/iPad向けアプリケーションを開発します。

KASAI.SOFTWARELAB

定価1,620円（税込）

iPhone&iPadプログラミングBible[下]

河西　朝雄著

iPhone/iPad向けアプリケーションを開発するにはAppleが提供するXcodeという統合開発環境を使用し、Objective-Cというプログラミング言語を使ってプログラムコードを記述します。

本シリーズは、iPhone/iPadアプリを開発するためのテクニックをすべて網羅するように22の章（カテゴリ）に分類して「上」、「下」の2分冊で構成しています。本書はその中の「下」です。

KASAI.SOFTWARELAB

定価 1,620 円（税込）

はじめに

iPhoneなどのスマートフォンやiPadなどのタブレット端末のユーザーインターフェースは指のタッチを基本とし、カメラやセンサーを内蔵し、音声認識・音声合成などが簡単に利用できる画期的なコンピュータです。マウス、キーボード、デイスプレイを主なユーザーインタフェースとするパソコンとは大きく異なります。「コンピュータ＝パソコン」の時代から「コンピュータ＝スマートフォン、タブレット端末」の時代に急速にパラダイムシフトしようとしています。スマートフォンは子供から女性、シニアまでの広い層に渡って、今までのパソコンユーザとは比べ物にならない数のユーザが見込まれます。

Appleが運営するアプリケーションマーケットとしてApp Store(アップストア）があります。2013年6月時点で有料、無料含め90万を超えるアプリケーションが提供されています、App Storeを通して企業だけでなく、一般ユーザが自作のアプリケーションを販売することができる点も今までにない利点です。つまり、ソフト会社の技術者以外にも、学生を中心に一般の人でもiPhone/iPadアプリで商売ができるようになる可能性があり、iPhone/iPadアプリ市場は今後急速に普及すると思います。

iPhone/iPad向けアプリケーションを開発するにはAppleが提供するXcodeという統合開発環境を使用し、Objective-Cというプログラミング言語を使ってプログラムコードを記述します。

本シリーズは、iPhone/iPadアプリを開発するためのテクニックをすべて網羅するように22の章（カテゴリ）に分類して「上」、「下」の2分冊で構成しています。本書はその中の「下」です。22の章というのはかなり多い章分けですが、細かく章分けをすることでカテゴリが分かり易く、各章のサイズは小さくなり初心者には、ひとつのまとまった単位がボリュームが少ないので、取りかかり易くなります。また、章の順序ではなく、知りたい章を先に学習することもできます。

既存の書籍やネット上の情報は重要な内容とそうでない情報がまぜこぜになっていたり、このプログラムをどこに書けばいいのかが曖昧だったり、サンプルが長すぎたりなど、初心者には理解しにくい内容が多いです。本シリーズではiPhone/iPadアプリを作る上で必要な技術的要素やテクニックを切り出し短いサンプルを付けて簡潔に提示します。

「下」は「上」で説明したiPhone/iPadプログラミングに関する基礎的内容をベースにして、iPhone/iPadでできる各種処理について10章～19章で説明します。最初に10章で、自身のアプリケーションに割り当てられるホームデイレクトリに対するファイル処理に関して説明します。iOSには、「Core Gracphics」ライブラリというものが標準で組み込まれています。「2章　グラフィックスを用いたObjective-C入門」でグラフィックスについては既に扱っていますが、11章で「Core Gracphics」ライブラリについて系統的に説明します。12章で、2次元・3次元コンピュータグラフィックス両方が扱えるOpenGLについて説明します。13章で、MKMapViewクラスを使用したマップ処理の方法を説明します。14章で、AVAudioPlayerクラスを使ったオーディオファイルのを録音、再生方法を説明します。15章で、MPMoviePlayerControllerクラスまたはMPMoviePlayerViewControllerクラスを使った動画ファイルの再生方法を説明します。16章で、加速度センサー、近接センサー、環境光（輝度）センサー、ジャイロセンサー、磁気センサーなどのセンサーの使い方を説明します。17章で、カメラによる静止画と動画の撮影方法について説明します。iPhoneのメールには、SMS、MMS、Eメール(i)、一般のEメールの4種類があります。18章で、Eメール(i)/一般のEメールを使ってメール送信する方法を説明します。19章で、iPhoneを使って通信を行う方法を説明します。

20章～22章はiPhone/iPadに関する技術的な内容ということでなく、iPhone/iPad用アプリを作る上でのヒントになるプログラミングテクニックについて説明します。最初は20章でリバーシーゲームを段階を追って作ります。21章では再帰を用いたゲームとして、ハノイの塔、迷路、マインスイーパーを作ります。22章では実用的なアプリを作る上でのヒントになる例題として電卓、My図鑑、電子書籍、お絵かきツールを作ります。

本書は次のような章の構成となります。

10章　ファイル処理
11章　グラフィックス
12章　OpenGL
13章　マップ
14章　オーディオ
15章　ムービー（動画）
16章　センサー
17章　カメラ
18章　メール
19章　通信
20章　リバーシー
21章　再帰を用いたゲーム
22章　実用アプリ

本書のプログラムはiPhoneを想定したものですが、基本的にはiPadでも使用できます。iPhone/iPad用のプログラムは基本的に同じ方法で記述できますが、両者は画面サイズが異なるため、画面サイズに依存するプログラムは変更が必要になります。また用意するアイコンのサイズも異なります。
これからiPhone/iPadアプリの開発を志す方々にとって、本書が少しでもお役に立てば幸いです。

2014年5月　河西朝雄

別冊のiPhone/iPadプログラミングBible[上]の内容は以下です。

目次

10章　ファイル処理

この章では以下のようなファイル処理に関する内容を説明します。

自身のアプリケーションに割り当てられるホームデイレクトリの下のDocumentsデイレクトリにファイルを保存し、読み出すことができます。

NSxxx型のデータはwriteToFileメソッドでファイルへのライトが、xxxWithContentsOfFileメソッドでファイルからのリードができます。バイナリーデータとしてリード/ライトする場合はNSData型を使用します。

NSFileManagerはファイルを管理するためのクラスです。ファイルの新規作成、削除、コピーなどを行うことができます。

NSFileHandleクラスはファイルハンドルを使ったファイル処理を行います。writeToFileメソッド/xxxWithContentsOfFileメソッドによる方法より柔軟な処理が行えます。

アプリケーションを実行するにあたって使用する各種設定値を「Key-Value」形式のファイルにしたものをプリファレンスと呼びます。このファイルはホームデイレクトリ下の「Library/Preferences」デイレクトリに保存されます。

実存するファイルやデイレクトリに対し別名（エイリアス）を付けることができます。これをシンボリックリンクと言いcreateSymbolicLinkAtPathメソッドで作成します。

10-1　デイレクトリとパス

ファイルを扱うにはデイレクトリとパスを把握して置く必要があります。

1.　ホームデイレクトリ

自身のアプリケーションに割り当てられるホームデイレクトリの下に以下のようなサブデイレクトリが作成されます。

ホームデイレクトリ
　　-Documents
　　-Library
　　　　-Preferences
　　-tmp

ホームデイレクトリのパスはNSHomeDirectory()関数で取得できます。Simulatorでは以下のようなパスになります。

/Users/ユーザ名/Library/Application Support/iPhone Simulator/バージョン/Applications/アプリケーションID

実機では以下のようなパスになります。

/var/mobile/Applications/アプリケーションID

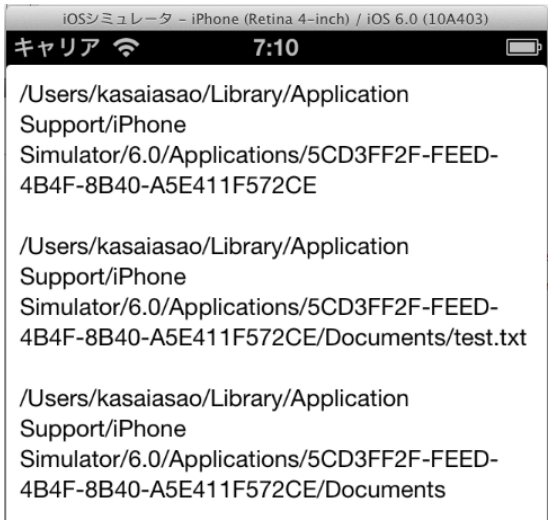
「注」以後のプログラムにおいてtextViewはStoryboard上に配置したTextViewのプロパティ名です。

```
@property (weak, nonatomic) IBOutlet UITextView *textView;
```

2.　パスの追加と削除

stringByAppendingPathComponentメソッドでパスの末尾に新しいパスを追加できます。
stringByDeletingLastPathComponentメソッドでパスの末尾のサブパスを削除できます。

```
NSString *home=NSHomeDirectory();
NSString *test=[home stringByAppendingPathComponent:@"Documents/test.txt"];
NSString *doc=[test stringByDeletingLastPathComponent];
self.textView.text=[NSString stringWithFormat:@"%@¥n¥n%@¥n¥n%@",home,test,doc];
```



10-2　データのリード/ライト

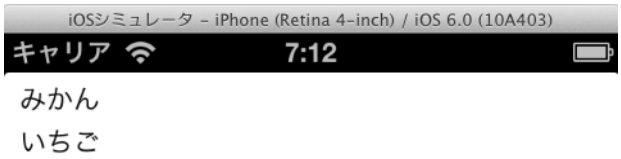
NSxxx型のデータはwriteToFileメソッドでファイルへのライトが、xxxWithContentsOfFileメソッドでファイルからのリードができます。バイナリーデータとしてリード/ライトする場合はNSData型を使用します。

以下の例ではホームデイレクトの下にDocumentsデイレクトがデフォルトで生成されているのでそこにtest.txtというファイル名でリード/ライトします。

1.　NSString型データ

NSString型のデータはstringWithContentsOfFile/writeToFileメソッドを使ってリード/ライトすることができます。

```
    NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
    NSError* err=nil;
    NSString *data=@"みかん¥nいちご";
    [data writeToFile:path atomically:YES encoding:NSUTF8StringEncoding error:&err];
    NSString *readdata=[NSString stringWithContentsOfFile:path
encoding:NSUTF8StringEncoding error:&err];
    self.textView.text=readdata;
```



NSString型データのライトとリードの場合だけ引数にencodingとerrorを指定します。他のデータ型ではこの引数は指定しません。

引数のencodingには以下のエンコード方式を指定します。通常はNSUTF8StringEncodingを指定します。

NSASCIIStringEncoding
NSJapaneseEUCStringEncoding
NSUTF8StringEncoding
NSISOLatin1StringEncoding
NSShiftJISStringEncoding
NSISOLatin2StringEncoding
NSISO2022JPStringEncoding
NSMacOSRomanStringEncoding

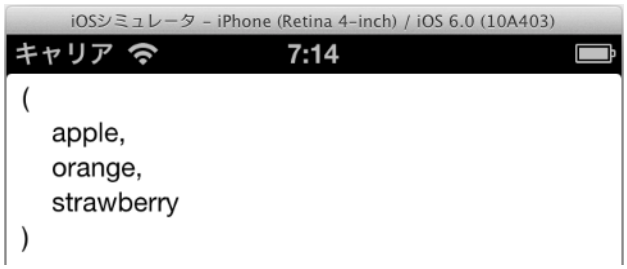
引数のerrorにはNSError型のエラー情報が返されます。エラーがなければnilが返されます。情報を受け取る引数なので「&err」のように&演算子でアドレスを渡します。localizedDescriptionメソッドでエラー内容をNSStringデータとして取得できます。

```
if(err!=nil){
    NSLog(@"%@",[err localizedDescription]);
}
```

2. NSArray型データ

NSArray型のデータはarrayWithContentsOfFile/writeToFileメソッドを使ってリード/ライトすることができます。

```
NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
NSArray *data=[NSArray arrayWithObjects:@"apple",@"orange",@"strawberry",nil];
[data writeToFile:path atomically:YES];
NSArray *readdata=[NSArray arrayWithContentsOfFile:path];
self.textView.text=[NSString stringWithFormat:@"%@",readdata];
```



3.　NSDictionary型データ

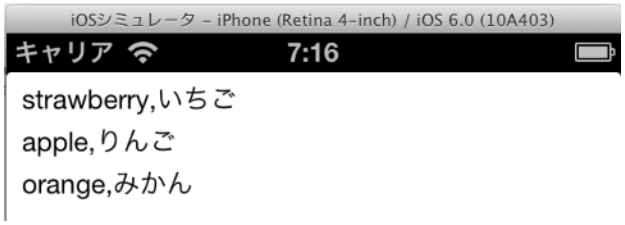
NSDictionary型のデータはdictionaryWithContentsOfFile/writeToFileメソッドを使ってリード/ライトすることができます。

```
    NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
    NSDictionary *data=[NSDictionary dictionaryWithObjectsAndKeys:
                        @"りんご", @"apple",
                        @"みかん", @"orange",
                        @"いちご" , @"strawberry"
                        ,nil];
    [data writeToFile:path atomically:YES];
    NSDictionary *readdata=[NSDictionary dictionaryWithContentsOfFile:path];
    self.textView.text=[NSString stringWithFormat:@"%@",readdata];
```

個々の要素を取り出して表示するには以下のようにします。

```
    NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
    NSDictionary *data=[NSDictionary dictionaryWithObjectsAndKeys:
                        @"りんご", @"apple",
                        @"みかん", @"orange",
                        @"いちご" , @"strawberry"
                        ,nil];
    [data writeToFile:path atomically:YES];
    NSDictionary *readdata=[NSDictionary dictionaryWithContentsOfFile:path];
    NSArray *key=[readdata allKeys];
    int n=[key count];
    NSString *msg=@"";
```

```
    for (int i=0;i<n;i++){
        id obj=[readdata objectForKey:[key objectAtIndex:i]];
        msg=[msg stringByAppendingString:[NSString stringWithFormat:@"%@,%@¥n",[key
objectAtIndex:i],obj]];
    }
    self.textView.text=msg;
```

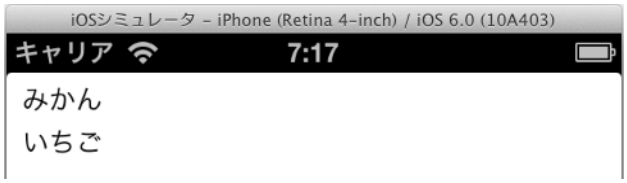


4. NSData型データ

NSData型はバイナリーデータとして扱われます。NSDataクラスのデータはdataWithContentsOfFile/writeToFileメソッドを使ってリード/ライトすることができます。各種データ(NSString,NSDictionary,UIImageなど）はNSData型に変換してファイルへライトします。NSData型のデータをファイルからリードし、各種データ(NSString,NSDictionary,UIImageなど）に変換します。

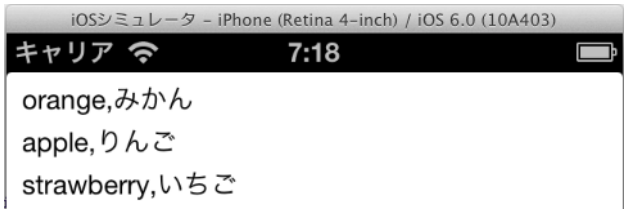
以下はNSString型をNSData型に変換してリード/ライトします。

```
    NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
    NSString *str=@"みかん¥nいちご";
    NSData *data=[str dataUsingEncoding:NSUTF8StringEncoding];
    //ファイルへの書き込み
    [data writeToFile:path atomically:YES];
    // ファイルをバイト列として読み込む
    NSData *readdata=[NSData dataWithContentsOfFile:path];
    // バイト列をテキストに変換する
    NSString *readstr=[[NSString alloc] initWithData:readdata encoding:NSUTF8StringEncoding];
    self.textView.text=readstr;
```



以下はNSDictionary型をNSData型に変換してリード/ライトします。

```
    NSDictionary *dict=[NSDictionary dictionaryWithObjectsAndKeys:
                        @"りんご", @"apple",
                        @"みかん", @"orange",
                        @"いちご" , @"strawberry"
                        ,nil];
    NSData *data=[NSKeyedArchiver archivedDataWithRootObject:dict];
    NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
    [data writeToFile:path atomically:YES];

    NSData *readdata=[NSData dataWithContentsOfFile:path];
    NSDictionary *readdict=[NSKeyedUnarchiver unarchiveObjectWithData:readdata];
    NSArray *key=[readdict allKeys];
    int n=[key count];
    NSString *msg=@"";
    for (int i=0;i<n;i++){
        id obj=[readdict objectForKey:[key objectAtIndex:i]];
        msg=[msg stringByAppendingString:[NSString stringWithFormat:@"%@,%@¥n",[key
objectAtIndex:i],obj]];
    }
    self.textView.text=msg;
```



5.　入力データのライト/リード

テキストフィールドに入力したデータをWriteボタンでライトし、Readボタンでリードしてラベルに表示する例を示します。テキストフィールドとラベルを配置し、アウトレット接続(text1,label1)します。ボタンを２つ配置し、アクション接続(Write,Read)します。

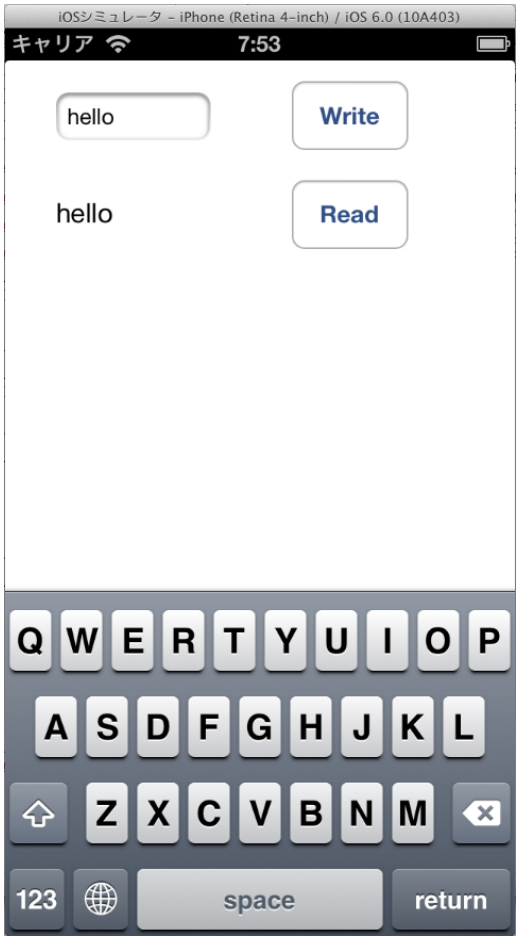
・ViewController.h

```
@property (weak, nonatomic) IBOutlet UITextField *text1;
@property (weak, nonatomic) IBOutlet UILabel *label1;
```

```
- (IBAction)Write:(id)sender;
- (IBAction)Read:(id)sender;
```

・ViewController.m

```
- (IBAction)Write:(id)sender {
    NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
    NSData *data=[self.text1.text dataUsingEncoding:NSUTF8StringEncoding];
    [data writeToFile:path atomically:YES];
}

- (IBAction)Read:(id)sender {
    NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
    if ([[NSFileManager defaultManager] fileExistsAtPath:path]) {
        NSData *data=[NSData dataWithContentsOfFile:path];
        NSString *str=[[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
        self.label1.text=str;
    }
}
```



同様なファイル処理をTextViewに対して行います。TextViewをクリックするとソフトキーが表示されます。

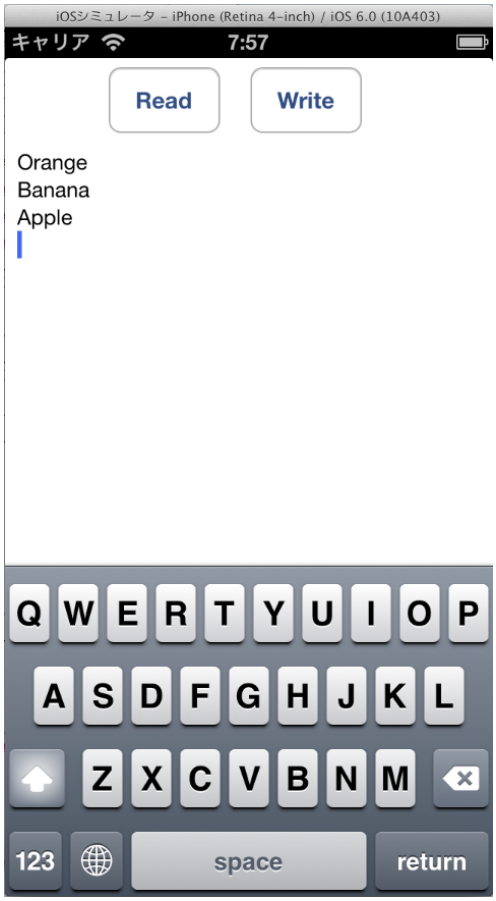
・ViewController.h

```
@property (weak, nonatomic) IBOutlet UITextView *textView;
- (IBAction)Read:(id)sender;
- (IBAction)Write:(id)sender;
```

・ViewController.m

```
- (IBAction)Read:(id)sender {
    NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
    if ([[NSFileManager defaultManager] fileExistsAtPath:path]) {
        NSData *data=[NSData dataWithContentsOfFile:path];
        NSString *str=[[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
        self.textView.text=str;
    }
```

```
}

- (IBAction)Write:(id)sender {
    NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
    NSData *data=[self.textView.text dataUsingEncoding:NSUTF8StringEncoding];
    [data writeToFile:path atomically:YES];
}
```



6. イメージファイルのリード

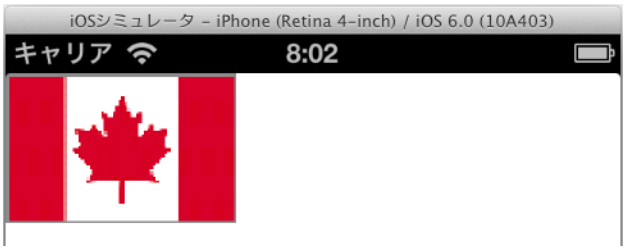
イメージは、

```
    UIImage *img=[UIImage imageNamed:@"star.png"];
```

でキャッシュできますが、ファイルとしてそのつど必要に応じて読み込むこともできます。

```
    NSString *path=[[NSBundle mainBundle] pathForResource:@"canada" ofType:@"png"];
```

```
NSData *data=[NSData dataWithContentsOfFile:path];
UIImage *img=[UIImage imageWithData:data];
UIImageView *iv=[[UIImageView alloc] initWithImage:img];
[self.view addSubview:iv];
```



10-3　NSFileManagerを使ったファイル操作

NSFileManagerはファイルを管理するためのクラスです。利用の際には、まずデフォルトのNSFileManagerインスタンスのfmを取得します。以後fmに対しファイル操作メソッドを適用します。

```
NSFileManager* fm=[NSFileManager defaultManager];
```

NSFileManagerクラスのメソッド	機能
defaultManager	デフォルトマネージャの取得。
contentsOfDirectoryAtPath	ファイル/デイレクトリ一覧の取得。
createFileAtPath	ファイルの新規作成。
fileExistsAtPath	ファイルが存在するかの検査。
copyItemAtPath	ファイルのコピー。
moveItemAtPath	ファイルの移動（リネーム）。
removeItemAtPath	ファイル/デイレクトリの削除。
createDirectoryAtPath	デイレクトリの新規作成。

1.　ファイルの新規作成

createFileAtPathメソッドでファイルを新規作成します。contents引数にライトするデータ、attributes引数にアクセス属性を指定します。NSFileHandleクラスのfileHandleForWritingAtPathメソッドではファイルの新規作成はできません。

```
NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
NSData *data=[@"書き込むデータ" dataUsingEncoding:NSUTF8StringEncoding];
NSFileManager *fm=[NSFileManager defaultManager];
[fm createFileAtPath:path contents:data attributes:nil];
```

2.　ファイルが存在するかの検査

pathで示すファイル（デイレクトリ）が存在するか調べるにはfileExistsAtPathメソッドを使います。存在すればYES,存在しなければNOを返します。

```
NSFileManager *fm= [NSFileManager defaultManager];
NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
BOOL b=[fm fileExistsAtPath:path];
```

さらにそのパスがデイレクトリかを調べるにはisDirectory引数を指定します。引数dirはアドレスを渡すため「&」を付けます。dirに戻される値がYESならデイレクトリです。

```
NSFileManager *fm= [NSFileManager defaultManager];
NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
BOOL dir;
BOOL b=[fm fileExistsAtPath:path isDirectory:&dir];
```

3. アクセス権の検査

pathで示すファイル（デイレクトリ）のアクセス権（たとえば読み取り可能属性）を調べるにはisReadableFileAtPathメソッドを使います。読み取り可能属性ならばYES,そうでないならばNOを返します。

```
NSFileManager *fm= [NSFileManager defaultManager];
NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
BOOL b=[fm isReadableFileAtPath:path];
```

アクセス権を調べるメソッド	調べる属性
isReadableFileAtPath	読み取り可能属性。
isWritableFileAtPath	書き込み可能属性。
isExecutableFileAtPath	実行可能属性。
isDeletableFileAtPath	削除可能属性。

4. ファイルのコピー

copyItemAtPathメソッドでファイルのコピーを行います。

```
NSFileManager* fm=[NSFileManager defaultManager];
NSError* err=nil;
NSString* src=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
NSString* dst=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/copy.txt"];
[fm copyItemAtPath:src toPath:dst error:&err];
if (err==nil) {
    NSLog(@"copyed.");
} else {
```

```
        NSLog(@"%@.",[err localizedDescription]);
    }
```

5. ファイルの移動（リネーム）

moveItemAtPathメソッドでファイルの移動を行います。同じデイレクトリ内での移動を行えばファイルのリネームになります。

```
    NSFileManager* fm=[NSFileManager defaultManager];
    NSError* err=nil;
    NSString* src=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
    NSString* dst=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/move.txt"];
    [fm moveItemAtPath:src toPath:dst error:&err];
    if (err==nil) {
        NSLog(@"moved.");
    } else {
        NSLog(@"%@.",[err localizedDescription]);
    }
```

6. ファイル/デイレクトリの削除

removeItemAtPathメソッドでファイル/デイレクトリの削除を行います。

```
    NSFileManager* fm=[NSFileManager defaultManager];
    NSError* err=nil;
    NSString* src=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
    [fm removeItemAtPath:src error:&err];
    if (err==nil) {
        NSLog(@"Deleted.");
    } else {
        NSLog(@"%@.",[err localizedDescription]);
    }
```

7. デイレクトリの新規作成

createDirectoryAtPathメソッドでデイレクトリの新規作成を行います。

```
    NSFileManager *fm= [NSFileManager defaultManager];
    NSString* path=[NSHomeDirectory() stringByAppendingPathComponent:@"work"];
```

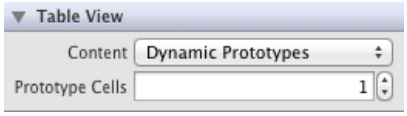
```
if (![fm fileExistsAtPath:path]){
    [fm createDirectoryAtPath:path withIntermediateDirectories:NO attributes:nil error:nil];

}
```

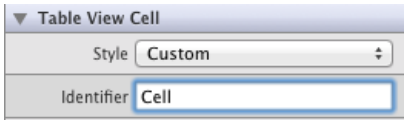
8. デイレクトリ一覧

contentsOfDirectoryAtPathメソッドはpathで示されるデイレクトリ内のサブデイレクトリとファイル一覧を取得します。以下はホームデイレクトリ内のサブデイレクトリとファイル一覧をTextViewに出力するものです。

```
NSString* path=NSHomeDirectory();
NSString *msg=path;
NSArray *fileNames= [[NSFileManager defaultManager] contentsOfDirectoryAtPath:path error:nil];
int n=[fileNames count];
for (int i=0;i<n;i++){
    NSString *str = [fileNames objectAtIndex:i];
    msg=[msg stringByAppendingString:[NSString stringWithFormat:@"%@¥n",str]];
}
self.textView.text=msg;
```

「補足」ファイル一覧をテーブルビューに表示します。テーブルの項目をクリックするとその下のディレクトリの内容を表示します。Backボタンで前の階層に戻ります。View Controller上にTable ViewとtableView Cellを配置します。テーブルビューの属性は以下のように設定します。Prototype Cellsを「1」に設定。Identifierを「Cell」に設定。





contentsOfDirectoryAtPathの返すファイル一覧はNSArrayなので、tablesの型もこれに合わせます。oldPathに１つまえのパスを保存しておきます。テーブルの再表示はreloadDataメソッドを使います。

・ViewController.h

```
@interface ViewController : UIViewController<UITableViewDelegate, UITableViewDataSource>{
    NSArray *tables;
    NSString *path;
    NSString *oldPath[20];
    int N;
}

@property (weak, nonatomic) IBOutlet UITableView *table1;
- (IBAction)Back:(id)sender;
```

・ViewController.m

```
// TableViewのデリゲートメソッド
- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {
    return @"ファイル一覧"; // タイトル
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    return tables.count; // 行数
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    UITableViewCell *cell=[tableView dequeueReusableCellWithIdentifier:@"Cell"];
    NSData *object=[tables objectAtIndex:indexPath.row];
    cell.textLabel.text=[object description];
    return cell; // セルの表示内容
}
-(void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPat
```

```
{
    oldPath[N++]=path;
    path=[path stringByAppendingPathComponent:[tables objectAtIndex:indexPat.row]];
    tables=[[NSFileManager defaultManager] contentsOfDirectoryAtPath:path error:nil];
    [self.table1 reloadData]; // テーブルの再表示
}

- (void)viewDidLoad
{
    [super viewDidLoad];
    path=NSHomeDirectory();
    tables=[[NSFileManager defaultManager] contentsOfDirectoryAtPath:path error:nil];
    N=0;
    self.table1.delegate=self;
    self.table1.dataSource=self;
}

- (IBAction)Back:(id)sender {
    if (N>0){
        path=oldPath[--N];
        tables=[[NSFileManager defaultManager] contentsOfDirectoryAtPath:path error:nil];
        [self.table1 reloadData]; // テーブルの再表示
    }
}
```

「補足」ディレクトリなら「d:」、ファイルなら「f:」の印を項目の先頭につけます。
contentsOfDirectoryAtPathメソッドで取得したファイル一覧を直接tablesに格納するのではなく、各ファイルごとにディレクトリかファイルか調べてtablesに追加します。このためtablesはNSMutableArray型とします。tablesの末尾に追加するには[tables insertObject:item atIndex:[tables count]];とします。

・ViewController.h

```
@interface ViewController : UIViewController<UITableViewDelegate, UITableViewDataSource>{
    NSMutableArray *tables;
    NSString *path;
    NSString *oldPath[20];
    int N;
}

@property (weak, nonatomic) IBOutlet UITableView *table1;
- (IBAction)Back:(id)sender;
```

・ViewContrller.m

```
// TableViewのデリゲートメソッド
- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {
```

```
    return @"ファイル一覧"; // タイトル
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    return tables.count; // 行数
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    UITableViewCell *cell=[tableView dequeueReusableCellWithIdentifier:@"Cell"];
    NSData *object=[tables objectAtIndex:indexPath.row];
    cell.textLabel.text=[object description];
    return cell; // セルの表示内容
}
-(void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPat
{
    oldPath[N]=path;
    NSString *item=[tables objectAtIndex:indexPat.row];
    item=[item substringWithRange:NSMakeRange(2,[item length]-2)];
    path=[path stringByAppendingPathComponent:item];
    BOOL dir;
    [[NSFileManager defaultManager] fileExistsAtPath:path isDirectory:&dir];
    if (dir) {
        [self disp];
        N++;
    }
}
-(void)disp
{
    NSFileManager *fm=[NSFileManager defaultManager];
    tables=[NSMutableArray arrayWithObjects:nil];
    NSArray *fileNames=[fm contentsOfDirectoryAtPath:path error:nil];
    int n=[fileNames count];
    for (int i=0;i<n;i++){
        NSString *file=[fileNames objectAtIndex:i];
        BOOL dir;
```

```
            [fm fileExistsAtPath:[path stringByAppendingPathComponent:file] isDirectory:&dir];
            NSString *item;
            if (dir){// ディレクトリか判定
                item=[NSString stringWithFormat:@"d:%@",file];
            }
            else{
                item=[NSString stringWithFormat:@"f:%@",file];
            }
            [tables insertObject:item atIndex:[tables count]]; // 末尾に追加
        }
        [self.table1 reloadData]; // テーブルの再表示
}
- (void)viewDidLoad
{
        [super viewDidLoad];
        path=NSHomeDirectory();
        [self disp];
        N=0;
        self.table1.delegate=self;
        self.table1.dataSource=self;
}

- (IBAction)Back:(id)sender {
        if (N>0) {
            path=oldPath[--N];
            [self disp];
        }
}
```

9.　ファイル属性

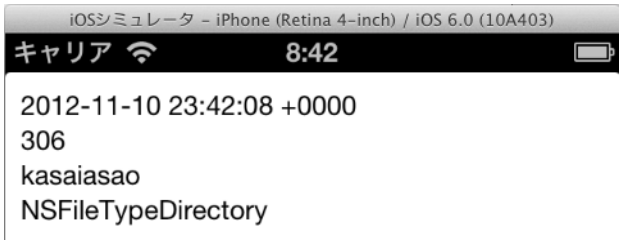
　attributesOfItemAtPathメソッドはファイル属性をNSDictionaryデータとして取得します。取得したデータのキーは以下です。これらのキーに対しobjectForKeyメソッドで値を取得します。

キー	意味
NSFileType	ファイル属性値で以下の値。 NSFileTypeDirectory：ディレクトリ NSFileTypeRegular通常：ファイル NSFileTypeSymbolicLink：シンボリックリンク NSFileTypeSocket：ソケット NSFileTypeCharacterSpecial：キャラクターデバイス(キー入力など) NSFileTypeBlockSpecial：ブロックデバイス(ディスクなど) NSFileTypeUnknown：不明
NSFileSize	ファイルサイズ(バイト)；リソースフォークのサイズは含まない。
NSFileModificationDate	最終変更日。
NSFileReferenceCount	参照回数。
NSFileDeviceIdentifier	ファイルが属する装置のID。
NSFileOwnerAccountName	ファイル所有者の名前。

NSFileGroupOwnerAccountName	ファイル所有者のグループ名。
NSFilePosixPermissions	ファイルのPosixの許可。
NSFileSystemNumber	システム番号。
NSFileSystemFileNumber	ファイル番号。
NSFileExtensionHidden	ファイルの拡張子を隠すかどうか。
NSFileHFSCreatorCode	ファイルのHFSの生成コード。
NSFileHFSTypeCode	ファイルのHFSのタイプコード。
NSFileImmutable	ファイルが可変かどうか。
NSFileAppendOnly	ファイルが読み込み専用かどうか。
NSFileCreationDate	ファイル作成日。
NSFileOwnerAccountID	ファイル所有者のアカウントID。
NSFileGroupOwnerAccountID	ファイルグループのID。
NSFileBusy	ファイルがBUSYかどうか。
NSFileProtectionKey	保護レベル。

以下はFile8というプロジェクト名（プロダクト名）で作成したプロジェクトのホームディレクトリ内にあるFile8.appの各種属性値を表示します。File8.appは一見するとファイルのような名前ですが、実はディレクトリ（NSFileTypeDirectory）であることがわかります。

```
NSFileManager *fm=[NSFileManager defaultManager];
NSString *path=[NSHomeDirectory() stringByAppendingPathComponent:@"File8.app"];
NSError *err=nil;
NSDictionary *attr=[fm attributesOfItemAtPath:path error:&err];
NSString *date=[attr objectForKey:NSFileModificationDate];
NSString *size=[attr objectForKey:NSFileSize];
NSString *name=[attr objectForKey:NSFileOwnerAccountName];
NSString *type=[attr objectForKey:NSFileType];
self.textView.text=[NSString stringWithFormat:@"%@¥n%@¥n%@¥n%@",date,size,name,type];
```



10-4　NSFileHandleを使ったリード/ライト

NSFileHandleクラスはファイルハンドルを使ったファイル処理を行います。「10-2　データのリード/ライト」で示した方法より柔軟な処理が行えます。

メソッド	機能
fileHandleForWritingAtPath	ライトモードでのファイルハンドルのオープン。
fileHandleForReadingAtPath	リードモードでのファイルハンドルのオープン。
writeData	データのライト。
readDataToEndOfFile	ファイルエンドまでのリード。
readDataOfLength	長さを指定したリード。
closeFile	ファイルハンドルのクローズ。
seekToEndOfFile	ファイル現在位置をファイルエンドに移動。
seekToFileOffset	ファイル現在位置を指定オフセット値分だけ移動。
offsetInFile	ファイル現在位置を取得(unsigned long long値)。

1.　新規作成、上書きモード、追加モード

NSFileHandleクラスのfileHandleForWritingAtPathメソッドではファイルの新規作成はできません。新規の場合はcreateFileAtPathメソッドでファイルを新規作成します。すでにファイルがある場合はfileHandleForWritingAtPathメソッドでファイル先頭からの上書きをします。「ABCDEF」というファイルに「123」をwriteDataでライトすると「123DEF」となります。

text1はテキストフィールドのプロパティ名です。

```
    NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
    NSData *data=[text1.text dataUsingEncoding:NSUTF8StringEncoding];
    NSFileManager *fm= [NSFileManager defaultManager];
    if (![fm fileExistsAtPath:path]){
        [fm createFileAtPath:path contents:data attributes:nil]; // 新規作成
    }
    else{
        NSFileHandle *fd= [NSFileHandle fileHandleForWritingAtPath:path]; // ライトモードでフ
ァイルハンドルをオープン
        [fd writeData:data];
        [fd closeFile];
    }
```

以下のように[fd seekToEndOfFile];でファイル現在位置をファイルエンドにしてからwriteDataでライトすると「ABCDEF123」となります。

```
    else{
        NSFileHandle *fd= [NSFileHandle fileHandleForWritingAtPath:path];
        [fd seekToEndOfFile]; // ファイルエンドへ移動（追加モード）
        [fd writeData:data];
        [fd closeFile];
    }
```

2. ファイルのリード

fileHandleForReadingAtPathメソッドでファイルハンドルをリードモードでオープンします。オープンに失敗するとnilを返します。readDataToEndOfFileメソッドでファイルエンドまでリードします。label1はラベルのプロパティ名です。

```
    NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
    NSFileHandle *fd=[NSFileHandle fileHandleForReadingAtPath:path];
    if (fd!=nil){
        NSData *data=[fd readDataToEndOfFile];
        [fd closeFile];
        NSString *str=[[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
        label1.text=str;
    }
```

3. シーケンシャルファイル

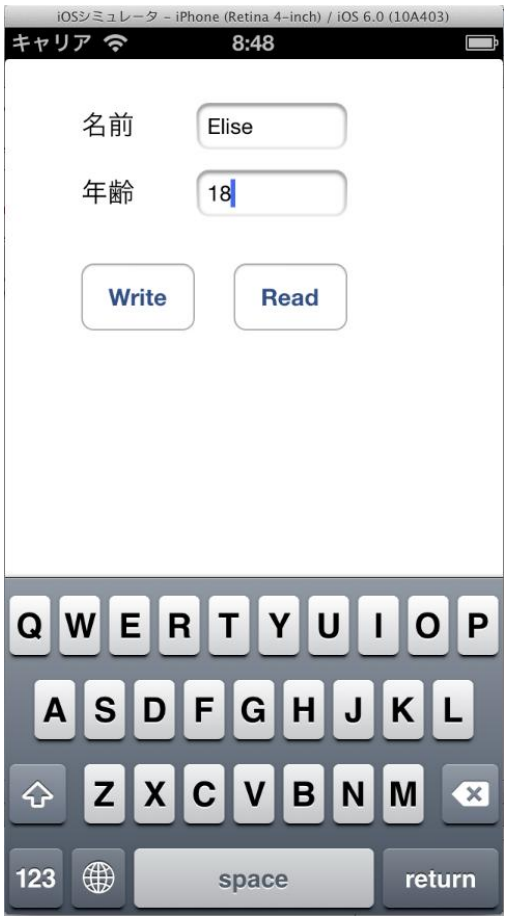
「名前,年齢¥n」を１つのレコードとするシーケンシャルファイルを作ります。Writeボタンでテキストフィールドのデータをファイルに追加書き込します。Readボタンでファイルを一括リードしてログに出力します。

・ViewController.h

```
@property (weak, nonatomic) IBOutlet UITextField *text1;
@property (weak, nonatomic) IBOutlet UITextField *text2;
- (IBAction)Write:(id)sender;
- (IBAction)Read:(id)sender;
```

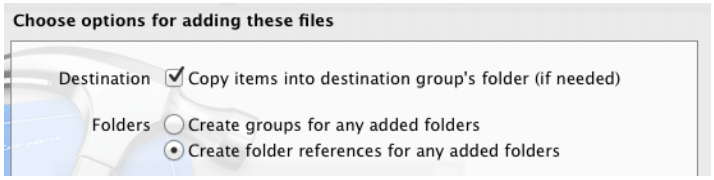
・ViewCpntroller.m

```
- (IBAction)Write:(id)sender {
    NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
    NSString *str=[NSString stringWithFormat:@"%@,%@¥n",self.text1.text,self.text2.text];
    NSData *data=[str dataUsingEncoding:NSUTF8StringEncoding];
    NSFileManager *fm= [NSFileManager defaultManager];
    if (![fm fileExistsAtPath:path]){
        [fm createFileAtPath:path contents:data attributes:nil]; // 新規作成
    }
    else{
        NSFileHandle *fd= [NSFileHandle fileHandleForWritingAtPath:path]; // ライトモードでフ
ァイルハンドルをオープン
        [fd seekToEndOfFile];
        [fd writeData:data];
        [fd closeFile];
    }
}

- (IBAction)Read:(id)sender {
    NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/test.txt"];
    NSFileHandle *fd=[NSFileHandle fileHandleForReadingAtPath:path];
    if (fd!=nil){
        NSData *readdata=[fd readDataToEndOfFile];
        [fd closeFile];
        NSString *readstr=[[NSString alloc] initWithData:readdata
encoding:NSUTF8StringEncoding];
        NSLog(@"%@",readstr);
    }
}
```


iOSシミュレータ - iPhone (Retina 4-inch) / iOS 6.0 (10A403)
キャリア
8:48
名前
Elise
年齢
18
Write
Read
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
123
space
return

10-5　デイレクトリ管理

プロジェクトナビゲータに表示されている黄色のデイレクトリは真のデイレクトリではなく、同じデイレクトリ内の内容をグループ分けしているにすぎません。「File」－「New」－「Group」で作成される黄色のデイレクトリもそうです。一見異なるデイレクトリに見えてもこれらのグループ分けされたファイルは全て同じデイレクトリ内です。真のデイレクトリを作成するには、パソコン上で実際のデイレクトリを作成し、このデイレクトリをプロジェクトナビゲータにドラッグドロップします。その際「Copy items 〜」にチェックを入れ、「Create folder〜」を選択します。



これで以下のように実際のフォルダが作成されます。このflagsデイレクトリは実行ファイルのアプリケーションデイレクトリFolder1.appの下に置かれることになります。実際のデイレクトリは青色で示されます。

アプリケーションデイレクトリ（この例ではFolder1.app）までのパスはNSHomeDirectory()ではなく[NSBundle mainBindle]で取得します。

```
NSBundle *bundle=[NSBundle mainBundle];
NSString *path=[bundle bundlePath];
```

アプリケーションデイレクトリ以外のイメージを参照する場合はimageNamedメソッドではできませんのでfileURLWithPathメソッドでファイルパスを指定して取得します。

```
NSString *imagePath=[path stringByAppendingPathComponent:@"flags/canada.png"];
NSURL *url=[NSURL fileURLWithPath:imagePath];
NSData *data=[NSData dataWithContentsOfURL:url];
UIImage *img=[[UIImage alloc] initWithData:data];
```

以下はflagsデイレクトリにあるcanada.pngとflags.txtをImageViewとtextViewに表示するものです。

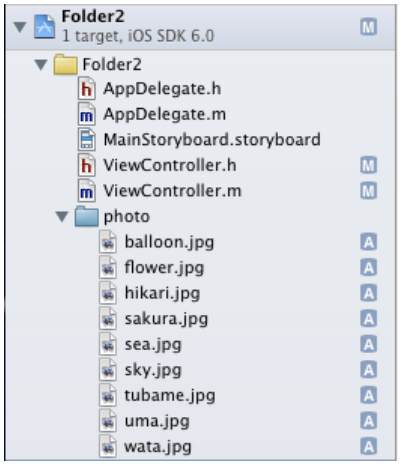
・ViewController.h

```
@property (weak, nonatomic) IBOutlet UIImageView *imageView;
@property (weak, nonatomic) IBOutlet UITextView *textView;
```

・ViewController.m

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    NSBundle *bundle=[NSBundle mainBundle];
    NSString *path=[bundle bundlePath];
    NSString *imagePath=[path stringByAppendingPathComponent:@"flags/canada.png"];
    NSURL *url=[NSURL fileURLWithPath:imagePath];
    self.imageView.image=[[UIImage alloc] initWithData:[NSData dataWithContentsOfURL:url]];

    NSString *txtPath=[path stringByAppendingPathComponent:@"flags/flags.txt"];
    NSError* err=nil;
    NSString *str=[NSString stringWithContentsOfFile:txtPath encoding:NSUTF8StringEncoding
error:&err];
    self.textView.text=str;
}
```

　指定したデイレクトリ内にあるイメージファイル（サフィックスがPNGのもの）をピックアップしてスクロールビューに表示します。スクロールビューへのイメージの表示方法は「5-24　UIScrollView」－「6.　UIScrollViewに複数のイメージを配置」参照。



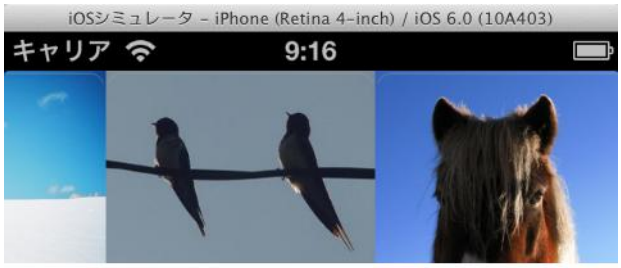
・ViewController.h

```
@interface ViewController：UIViewController<UIScrollViewDelegate>{
    UIImageView *imageView;
    NSMutableArray *photo;
}
```

・ViewController.m

```
- (IBAction)Down:(id)sender { // ボタンのダウンアクションで呼ばれる
    UIButton *bt=(UIButton *)sender;
    int n=bt.tag-1;
    NSURL *url=[NSURL fileURLWithPath:[photo objectAtIndex:n]];
    imageView.image=[[UIImage alloc] initWithData:[NSData dataWithContentsOfURL:url]];
}

- (void)viewDidLoad
{
    [super viewDidLoad];
    int N=0; //写真の枚数
    NSBundle *bundle=[NSBundle mainBundle];
    NSString *path=[bundle bundlePath];
    path=[path stringByAppendingPathComponent:@"photo"];
    NSArray *fileNames=[[NSFileManager defaultManager] contentsOfDirectoryAtPath:path
error:nil];
    photo=[NSMutableArray arrayWithObjects:nil];
    int n=[fileNames count];
    for (int i=0;i<n;i++){
        NSString *item= [fileNames objectAtIndex:i];
        if ([item hasSuffix:@".jpg"]){
            NSString *fullPath=[path stringByAppendingPathComponent:item];
            [photo insertObject:fullPath atIndex:[photo count]]; // 末尾に追加
            N++;
        }
    }
    CGRect rect=[[UIScreen mainScreen] applicationFrame];
    UIScrollView *sview=[[UIScrollView alloc]
initWithFrame:CGRectMake(0,0,rect.size.width,100)];
    [self.view addSubview:sview];
    sview.delegate=self;
    sview.contentSize=CGSizeMake(140*N,100); // スクロールビューのサイズ
    sview.contentOffset=CGPointMake(0,0);
    imageView=[[UIImageView alloc] initWithFrame:CGRectMake(0,0,300,200)];
    imageView.center=CGPointMake(rect.size.width/2,rect.size.height/2);
    [self.view addSubview:imageView];

    UIButton *btn[N];
```

```
    for (int i=0;i<N;i++){
        btn[i]=[UIButton buttonWithType:UIButtonTypeRoundedRect];
        btn[i].tag=i+1;
        btn[i].frame=CGRectMake(140*i,0,140,100); // 配置位置とサイズ
        NSURL *url=[NSURL fileURLWithPath:[photo objectAtIndex:i]]; // ファイルパスから
UIImageを作る
        UIImage *img=[[UIImage alloc] initWithData:[NSData dataWithContentsOfURL:url]];
        [btn[i] setImage:img forState:UIControlStateNormal]; // ボタンにイメージを表示
        [btn[i] addTarget:self action:@selector(Down:)
forControlEvents:UIControlEventTouchDown];
        [sview addSubview:btn[i]]; // スクロールビューに配置
    }
}
```

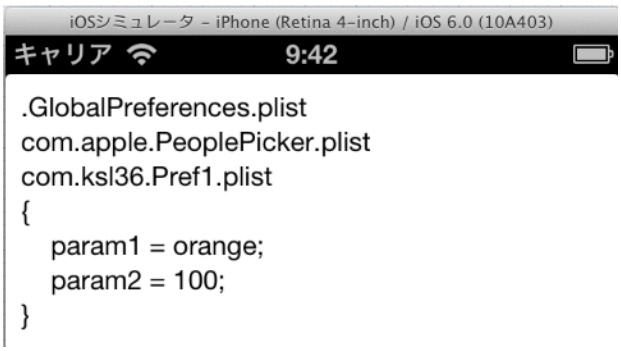


10-6　プリファレンス

　アプリケーションを実行するにあたって使用する各種設定値を「Key-Value」形式のファイルにしたものをプリファレンスと呼びます。このファイルはホームデイレクトリ下の「LIbrary/Preferences」デイレクトリに保存されます。Preferenceの英語の意味は「好み、選択」です。プリファレンスはNSUserDefaultsクラスを使って保存や読み取りが行えます。プリファレンスを使えば同じアプリケーションの異なるクラス（ビュー）でデータを共有することができます。詳細は「4-21　NSUserDefaultクラス（プリファレンス）」、「8-8　プリファレンスを使用したデータの共有」を参照してください。

　ユーザがプリファレンスにキーと値を保存すると「LIbrary/Preferences」デイレクトリに「カンパニーID.プロジェクト名.plist」（たとえばjp.kasai.Pref1.plist）のユーザ用プリファレンスファイルが作成されそこにキーと値が保存されます。デフォルトで「.GlobalPreferences.plist」、「com.apple.PeoplePicker.plist」というシステム用プリファレンスファイルが作成されています。[NSUserDefaults standardUserDefaults]で扱うプリファレンスはシステム用プリファレンスファイルとユーザ用プリファレンスファイルを集めた内容です。プロジェクト名（プロダクト名）を「Pref1」としています。

```
    NSUserDefaults *defaults=[NSUserDefaults standardUserDefaults];
    [defaults setObject:@"orange" forKey:@"param1"];
    [defaults setInteger:100 forKey:@"param2"];

    NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Library/Preferences"];
    NSArray *fileNames= [[NSFileManager defaultManager] contentsOfDirectoryAtPath:path
error:nil];
    int n=[fileNames count];
    NSString *msg=@"";
    for (int i=0;i<n;i++){
        NSString *str=[fileNames objectAtIndex:i];
        msg=[msg stringByAppendingString:[NSString stringWithFormat:@"%@¥n",str]];
    }
    path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Library/Preferences/com.ksl36.Pref1.plist"];
    NSDictionary *dict=[NSDictionary dictionaryWithContentsOfFile:path];
    msg=[msg stringByAppendingString:[NSString stringWithFormat:@"%@",dict]];
    textView.text=msg;
```



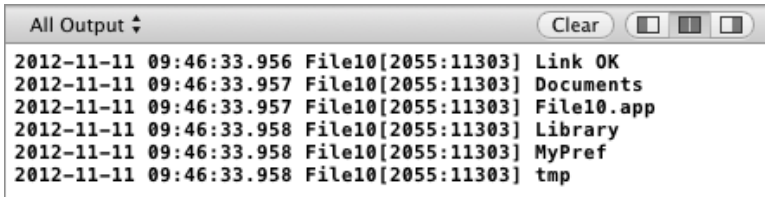
「注」1回目の実行時には書き込んだプリファレンスの内容はTextViewには反映されません。再度実行し直してください。

10-7　シンボリックリンク

実存するファイルやデイレクトリに対し別名（エイリアス）を付けることができます。これをシンボリックリンクと言いcreateSymbolicLinkAtPathメソッドで作成します。シンボリックリンクはホームデイレクトリ上に作成されます。すでにあるシンボリックリンクと同じ名前で再度定義するとエラーとなります。

以下は"Library/Preferences"に対し別名の"MyPref"を設定しています。ホームデイレクトリにシンボリックリンクのMyPrefが作成されたことが確認できます。

```
    NSFileManager *fm=[NSFileManager defaultManager];
    NSString *path=[NSHomeDirectory()
stringByAppendingPathComponent:@"Library/Preferences"];
    NSString *ailias=[NSHomeDirectory() stringByAppendingPathComponent:@"MyPref"];
    NSError* err;
    if (![fm fileExistsAtPath:ailias]){
        if ([fm createSymbolicLinkAtPath:ailias withDestinationPath:path error:&err])
            NSLog(@"Link OK");
        else
            NSLog(@"Link NG");
    }

    NSArray *fileNames= [[NSFileManager defaultManager]
contentsOfDirectoryAtPath:NSHomeDirectory() error:nil];
    int n=[fileNames count];
    for (int i=0;i<n;i++){
        NSString *str = [fileNames objectAtIndex:i];
        NSLog(@"%@",str);
    }
```



シンボリックリンクから元のパスを取得するにはdestinationOfSymbolicLinkAtPathメソッドを使用します。

```
NSString *origin=[fm destinationOfSymbolicLinkAtPath:ailias error:&err];
NSLog(@"%@",origin);
```

著者略歴

河西 朝雄（かさいあさお）
山梨大学工学部電子工学科卒（1974年）。長野県岡谷工業高等学校情報技術科教諭、長野県松本工業高等学校電子工業科教諭を経て、現在は「カサイ．ソフトウエアラボ」代表。

「主な著書」

「入門ソフトウエアシリーズC言語」、「同シリーズJava言語」、「同シリーズC++」、「入門新世代言語シリーズVisualBasic4.0」、「同シリーズDelphi2.0」、「やさしいホームページの作り方シリーズHTML」、「同シリーズJavaScript」、「同シリーズHTML機能引きテクニック編」、「同シリーズホームページのすべてが分かる事典」、「同シリーズiモード対応HTMLとCGI」、「同シリーズiモード対応Javaで作るiアプリ」、「同シリーズVRML2.0」、「チュートリアル式言語入門VisualBasic.NET」、「はじめてのVisualC#．NET」、「C言語用語辞典」ほか（以上ナツメ社）

「構造化BASIC」、「Microsoft LanguageシリーズMicrosoft VISUAL C++初級プログラミング入門上、下」、「同シリーズVisualBasic初級プログラミング入門上、下」、「C言語によるはじめてのアルゴリズム入門」、「Javaによるはじめてのアルゴリズム入門」、「VisualBasicによるはじめてのアルゴリズム入門」、「VisualBasic6.0入門編、中級テクニック編、上級編」、「Internet Language改訂新版シリーズ ホームページの制作」、「同シリーズJavaScript入門」、「同シリーズJava入門」、「New Language シリーズ標準VisualC++プログラミングブック」、「同シリーズ標準Javaプログラミングブック」、「VB.NET基礎学習Bible」、「原理がわかるプログラムの法則」、「プログラムの最初の壁」、「河西メソッド：C言語プログラム学習の方程式」、「基礎から学べるVisualBasic2005標準コースウエア」、「基礎から学べるJavaScript標準コースウエア」、「基礎から学べるC言語標準コースウエア」、「基礎から学べるPHP標準コースウエア」、「なぞりがきC言語学習ドリル」、「C言語　標準ライブラリ関数ポケットリファレンス[ANSI C,ISO C99対応]」、「 C言語　標準文法ポケットリファレンス[ANSI C,ISOC99対応]」ほか（以上技術評論社）

iPhone&iPadプログラミングBible[下]

2014年5月18日　初版　第1刷発行
著者：河西 朝雄
発行者：河西 朝雄
発行所：カサイ．ソフトウエアラボ
長野県茅野市ちの813　TEL.0266-72-4778

デザイン：河西 朝樹

本書の一部または全部を著作権法の定める範囲を超え、無断で複写、複製、転載、あるいはファイルに落とすことを禁じます。
本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた運用は、必ずお客様自身の責任と判断によって行ってください。これらの情報の運用の結果について、発行者および著者はいかなる責任も負いません。

定価＝1,620円（税込）
©2014　河西　朝雄