Mobile Application Development

Lecture 23
Sensors and Multimedia

Lecture Summary

- Core Motion
- Working with Audio and Video: Media Player framework, System Sound Services, AVFoundation framework
- Camera and Photo Library
- DEMO

Core Motion

- Core Motion (`#import <CoreMotion/CoreMotion.h>`) provides a unified framework to receive motion data from device hardware and process that data
- The framework supports accessing both raw and processed accelerometer data using block-based interfaces

- Core Motion provides support for data coming from:
  - accelerometer (CMAccelerometerData)
  - gyroscope (CMGyroData)
  - magnetometer (CMMagnetometerData)

https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CoreMotion_Reference/_index.html

CMMotionManager

- A `CMMotionManager` object is the gateway to the motion services provided by iOS
- Motion services provide accelerometer data, rotation-rate data, magnetometer data, and other device-motion data such as attitude
- A `CMMotionManager` is created with alloc/init
- Only one `CMMotionManager` should be present in an app
- After a `CMMotionManager` has been created it can be configured to set some parameters, such as the update interval for each type of motion

https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CoreMotion_Reference/_index.html

CMMotionManager

- After a `CMMotionManager` has been created and configured it can be asked to handle 4 types of motion:
  - raw accelerometer data
  - raw gyroscope data
  - raw magnetometer data
  - processed device-motion data (which includes accelerometer, rotation-rate, and attitude measurements)
- To receive motion data at specific intervals, the app calls a “start” method that takes an operation queue and a block handler of a specific type for processing those updates
- The motion data is passed into the block handler

```
startAccelerometerUpdatesToQueue:(NSOperationQueue *)
                     withHandler:^(CMAccelerometerData *accelerometerData, NSError *error)
        startGyroUpdatesToQueue:(NSOperationQueue *)
                            withHandler:^(CMGyroData *gyroData, NSError *error)
 startMagnetometerUpdatesToQueue:(NSOperationQueue *)
                      withHandler:^(CMMagnetometerData *magnetometerData, NSError *error)
```

CMMotionManager

- To stop receiving updates the appropriate stop methods of `CMMotionManager` must be called:
  - `stopAccelerometerUpdates`
  - `stopGyroUpdates`
  - `stopMagnetometerUpdates`
  - `stopDeviceMotionUpdates`

Core Motion Cookbook

- **Accelerometer**:
  - Set the `accelerometerUpdateInterval` property to specify an update interval
  - Call the `startAccelerometerUpdatesToQueue:withHandler:` method, passing in a block of type `CMAccelerometerHandler`
  - Accelerometer data is passed into the block as `CMAccelerometerData` objects
- **Gyroscope**:
  - Set the `gyroUpdateInterval` property to specify an update interval
  - Call the `startGyroUpdatesToQueue:withHandler:` method, passing in a block of type `CMGyroHandler`
  - Rotation-rate data is passed into the block as `CMGyroData` objects

Core Motion Cookbook

- **Magnetometer**:
  - Set the `magnetometerUpdateInterval` property to specify an update interval
  - Call the `startMagnetometerUpdatesToQueue:withHandler:` method, passing a block of type `CMMagnetometerHandler`
  - Magnetic-field data is passed into the block as `CMMagnetometerData` objects.
- **Device motion**:
  - Set the `deviceMotionUpdateInterval` property to specify an update interval
  - Call the `startDeviceMotionUpdatesUsingReferenceFrame:` or `startDeviceMotionUpdatesUsingReferenceFrame:toQueue:withHandler:` or `startDeviceMotionUpdatesToQueue:withHandler:` method, passing in a block of type `CMDeviceMotionHandler`
  - Rotation-rate data is passed into the block as `CMDeviceMotion` objects

CMMotionManager example

```
CMMotionManager *manager = [[CMMotionManager alloc] init];

[manager
     startAccelerometerUpdatesToQueue:[[NSOperationQueue alloc] init]
                          withHandler:^(CMAccelerometerData *accelerometerData, NSError *error){

     /* process data block */

}];
```

Multimedia: Audio

- iOS provides several frameworks to be used to deal with audio in applications at different levels:
- Each framework has different features and the use of a framework over another depends on the type of operations that are needed inside an app

Framework	Usage
Media Player	play songs, audio books, or audio podcasts from a user's iPod library **https://developer.apple.com/library/ios/documentation/MediaPlayer/Reference/MediaPlayer_Framework**
AV Foundation	play and record audio using a simple Objective-C interface
Audio Toolbox	play audio with synchronization capabilities, access packets of incoming audio, parse audio streams, convert audio formats, and record audio with access to individual packets
Audio Unit	connect to and use audio processing plug-ins
OpenAL	provide positional audio playback in games and other applications. iOS supports OpenAL 1.1

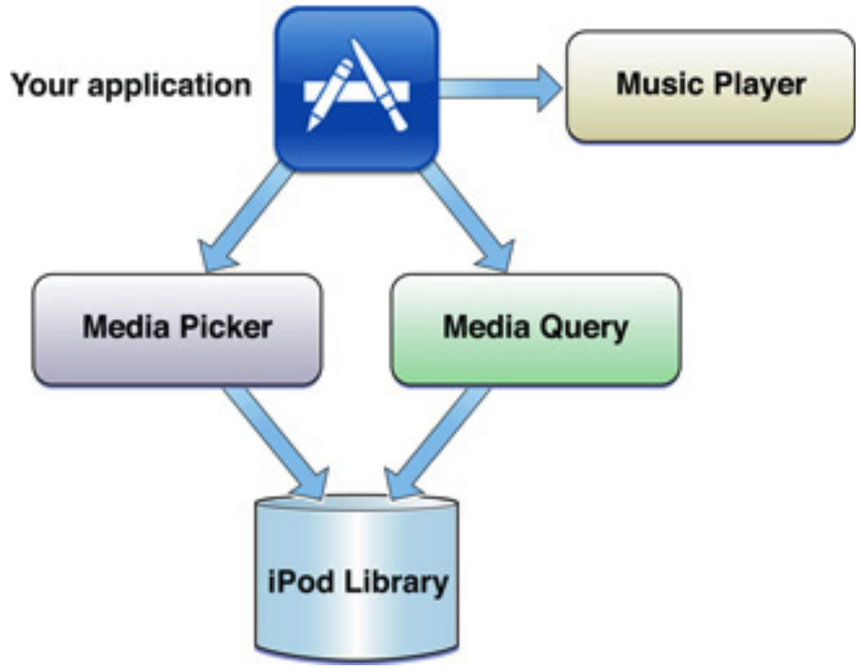
https://developer.apple.com/library/ios/documentation/AudioVideo/Conceptual/MultimediaPG

Accessing the iPod Library

- iPod library access provides mechanisms to play songs, audio books, and audio podcasts
- Basic playback and advanced searching and playback control
- There are two ways to get access to media items:

1. the media picker is a view controller that behaves like the built-in iPod application's music selection interface
2. the media query interface supports predicate-based specification of items from the device iPod library

- The retrieved media items are played using the music player `MPMusicPlayerController`



https://developer.apple.com/library/ios/documentation/Audio/Conceptual/iPodLibraryAccess_Guide

Accessing the iPod Library

- The Media Player framework must be linked to the project and imported in code using the import directive `#import <MediaPlayer/MediaPlayer.h>`
- There are two types of music player:
  - the *application music player* plays music locally within your app
  - the *iPod music player* employs the built-in iPod app
- An app music player can is instantiated with the following line of code

```
MPMusicPlayerController *player = [MPMusicPlayerController applicationMusicPlayer];
```

- After a music player has been created, a playback queue can be set using the method `setQueueWithQuery:` or `setQueueWithItemCollection:`
- The playback queue can then be played with the player's `play` method

Managing playback

- The `MPMusicPlayerController` class conforms to the `MPMediaPlayback` protocol
- This means that the player implements the following methods to change the playback state:
  - `play`
  - `pause`
  - `stop`
  - ... plus other methods
- To control the playback of the queue, the following methods can also be used:
  - `skipToNextItem`
  - `skipToBeginning`
  - `skipToPreviousItem`

MPMediaItem

- A media item (`MPMediaItem`) represents a single media (a song or a video podcast) in the iPod library
- A media item has a unique identifier, which persists across application launches
- Media items can have a wide range of metadata associated, such as the song title, artist, album, ...

- Metadata are accessed using the `valueForProperty:` method
- Some valid properties (constants): `MPMediaItemPropertyTitle`, `MPMediaItemPropertyAlbumTitle`, `MPMediaItemPropertyArtist`, `MPMediaItemPropertyAlbumArtist`, `MPMediaItemPropertyGenre`
- Each property has a different return type: for instance `MPMediaItemPropertyTitle` returns an `NSString`, while `MPMediaItemPropertyArtwork` return an instance of `MPMediaItemArtwork`

MPMediaQuery

- Media items in the iPod library can be queried using the `MPMediaQuery` class
- A media item collection (`MPMediaItemCollection`) is an array of media items
- Collections from the device iPod library can be obtained by accessing a media query's `collections` property

MPMediaPickerController

- An `MPMediaPickerController` is a specialized view controller used to provide a graphical interface for selecting media items
- The `MPMediaPickerController` displays a table view with media items that can be selected
- Tapping on the done button will dismiss the media item picker
- The `MPMediaPickerController` has a delegate (`MPMediaPickerControllerDelegate`) which gets notified when the user selects a media item or cancels the picking, with the methods `mediaPicker:didPickMediaItems:` and `mediaPickerDidCancel:` respectively
- To display a media item picker, it must be presented **modally** on an existing view controller
- To present a view controller modally, it should have its `modalTransitionStyle` property set and then the current view controller should call `presentViewController:animated:completion:`

```
MPMediaPickerController *picker = [[MPMediaPickerController alloc] init];
picker.modalTransitionStyle = UIModalTransitionStyleCoverVertical;
[self presentViewController:picker animated:YES completion:nil];
```

- The media picker delegate should explicitly dismiss the picker in `mediaPickerDidCancel:` with `dismissModalViewControllerAnimated:` (this is automatic when tapping the done button)

System Sound Services

- System Sound Services are used to play user-interface sound effects (such as button clicks), or to invoke vibration on devices that support it (`#import <AudioToolbox/AudioToolbox.h>`)
- Sound files must be:
  - less than 30 seconds long
  - in linear PCM or IMA4 (IMA/ADPCM) format
  - .caf, .aif, or .wav files
- First, first create a sound ID object with the `AudioServicesCreateSystemSoundID()` function
- Then, play the sound with the `AudioServicesPlaySystemSound()` function
- To trigger a vibration, just use the following line of code:

```
AudioServicesPlaySystemSound(kSystemSoundID_Vibrate);
```

https://developer.apple.com/library/ios/documentation/AudioToolbox/Reference/SystemSoundServicesReference

AVAudioPlayer

- The `AVAudioPlayer` class (`#import <AVFoundation/AVFoundation.h>`) provides a simple Objective-C interface for playing sounds
- Apple recommends to use this class for playback of audio that does not require stereo positioning or precise synchronization, and audio was not captured from a network stream
- `AVAudioPlayer` can:
  - Play sounds of any duration
  - Play sounds from files or memory buffers
  - Loop sounds
  - Play multiple sounds simultaneously (although not with precise synchronization)
  - Control relative playback level for each sound you are playing
  - Seek to a particular point in a sound file, which supports application features such as fast forward and rewind
  - Obtain audio power data that you can use for audio level metering
- The `AVAudioPlayer` class has a delegate (`AVAudioPlayerDelegate`) which gets notified of audio interruptions, audio decoding errors, and completion of playback

AVAudioPlayer example

```
/* get the URL of an audio file in the bundle */
NSString *soundFilePath = [[NSBundle mainBundle] pathForResource: @"sound" ofType: @"wav"];
NSURL *fileURL = [[NSURL alloc] initFileURLWithPath: soundFilePath];

/* create an AVAudioPlayer for the given audio file */
AVAudioPlayer *player = [[AVAudioPlayer alloc] initWithContentsOfURL:fileURL error:nil];

/* prepare the audio player for playback by preloading its buffers */
[player prepareToPlay];

/* set the delegate for the player */
[player setDelegate:self];

/* start the playback */
[player play];
```

AVAudioRecorder

- The `AVAudioRecorder` class (`#import <AVFoundation/AVFoundation.h>`) provides a simple Objective-C interface for recording sounds
- `AVAudioRecorder` can:
  - Record until the user stops the recording
  - Record for a specified duration
  - Pause and resume a recording
  - Obtain input audio-level data that you can use to provide level metering
- The `AVAudioPlayer` class has a delegate (`AVAudioPlayerDelegate`) which gets notified of audio interruptions, audio decoding errors, and completion of recording
- The usage of an audio recorder is very similar to that of an audio player, but the recording settings, such as the bitrate and the number of channels, should be configured
- An audio recorder are to be used inside an `AVAudioSession`, which is used to set the audio context for the app

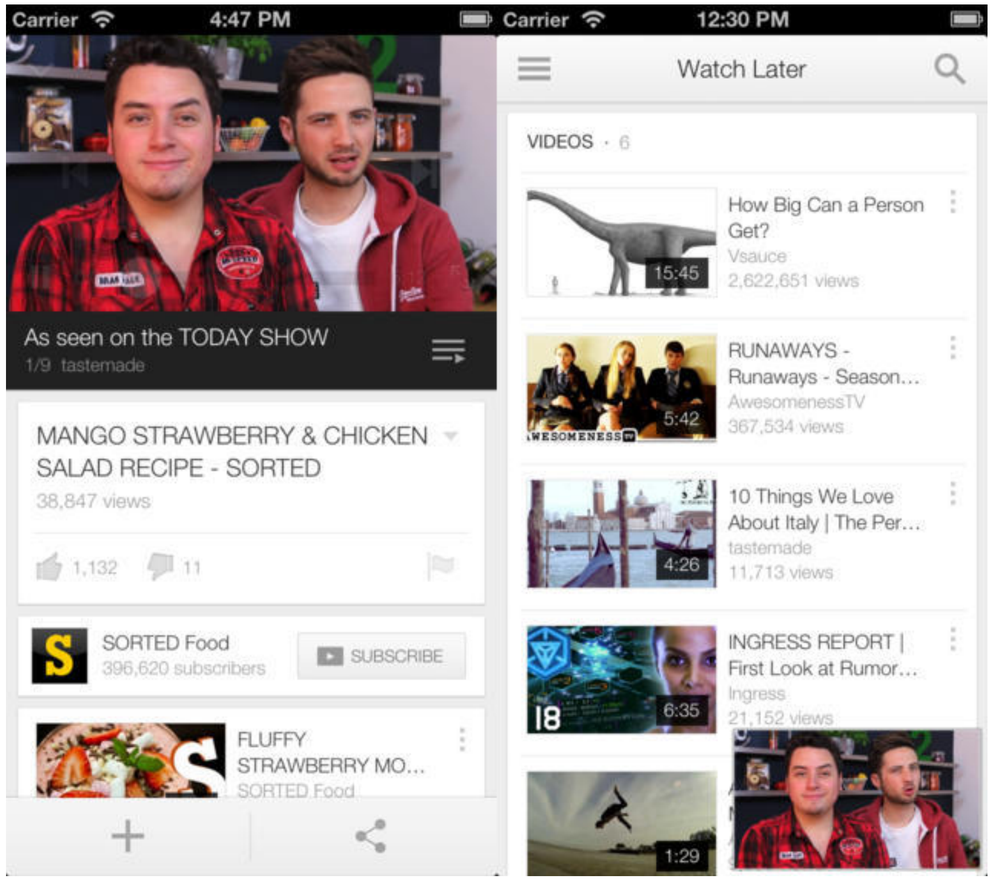
AVAudioRecorder example

```
/* set the AVAudioSession parameters */
[[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryRecord error:nil];
[[AVAudioSession sharedInstance] setMode:AVAudioSessionModeVideoRecording error:nil];
[[AVAudioSession sharedInstance] setActive:YES error:nil];

/* get the URL of an audio file in the bundle */
NSString *soundFilePath = [[NSBundle mainBundle] pathForResource: @"sound" ofType: @"wav"];
NSURL *fileURL = [[NSURL alloc] initFileURLWithPath: soundFilePath];

/* get the URL for the recorded audio file */
NSURL *url = ...;

/* set the audio recorder parameters (format, quality, bitrate, number of channels, sample rate) */
NSDictionary *settings = [NSDictionary dictionaryWithObjectsAndKeys:
                                        @(kAudioFormatMPEG4AAC), AVFormatIDKey,
                                        @(AVAudioQualityMax), AVEncoderAudioQualityKey,
                                        @(16), AVEncoderBitRateKey,
                                        @(1), AVNumberOfChannelsKey,
                                        @(44100), AVSampleRateKey,
                                        nil];

/* create the audio recorder */
AVAudioRecorder *recorder = [[AVAudioRecorder alloc] initWithURL:url settings:settings error:nil];

/* start recording */
[recorder record];  /* or use recordForDuration: */
```

Video Playback with MPMoviePlayerController

- An `MPMoviePlayerController` manages the playback of a movie from a file or a network stream
- Playback occurs in a view owned by the movie player and takes place either fullscreen or inline
- `MPMoviePlayerController` supports wireless movie playback to AirPlay devices like the Apple TV

- An `MPMoviePlayerController` object is initialized with the URL of the movie to be played
- After it has been created, the view of the `MPMoviePlayerController` (accessible via its view property) can be added as a subview of the current view



Video Playback with MPMoviePlayerController

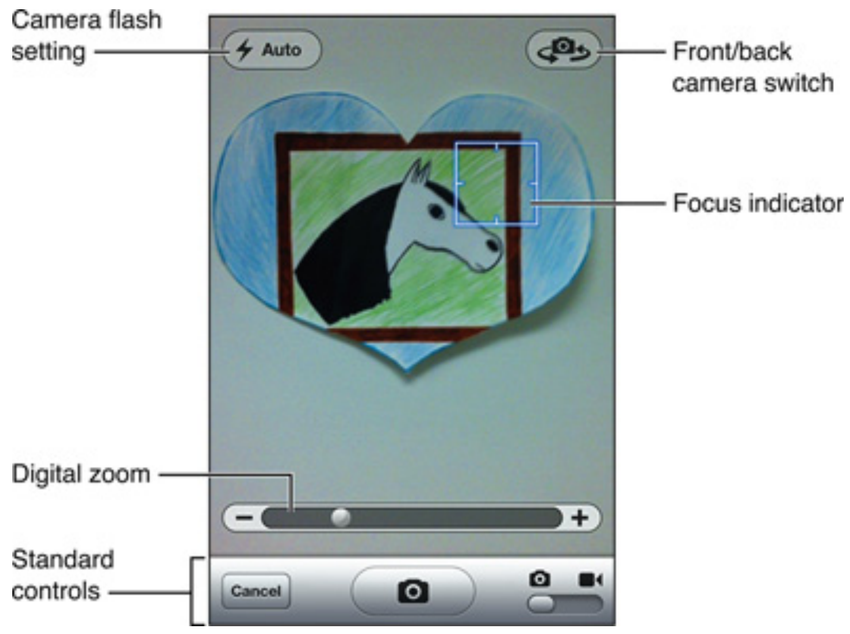
- `MPMoviePlayerController` has many properties that can be used to affect its behavior:
  - `allowsAirPlay` specifies whether the movie player allows AirPlay movie playback
  - `fullscreen` indicates whether the movie player is in full-screen mode (read-only); the value can be changed with the method `setFullscreen:animated:`
  - `controlStyle` specifies the style of the playback controls (no controls, embedded, fullscreen)
  - `initialPlaybackTime` is the time of the video, in seconds, when playback should start
  - `endPlaybackTime` is the time of the video, in seconds, when playback should stop
  - ...
- `MPMoviePlayerController` also generates notifications related to the state of movie playback:
  - `MPMoviePlayerPlaybackStateDidChangeNotification`
  - `MPMoviePlayerContentPreloadDidFinishNotification`
  - `MPMoviePlayerDidEnterFullscreenNotification`/`MPMoviePlayerDidExitFullscreenNotification`
  - ...
- To receive the notifications, it is necessary to register for the appropriate notification

MPMoviePlayerController example

```
/* create a player to play a video at the given URL */
MPMoviePlayerController *player = [[MPMoviePlayerController alloc] initWithContentURL:url];

/* set the player's view frame */
[player.view setFrame:self.view.bounds];

/* add the player's view to the current view controller's view */
[self.view addSubview:player.view];

/* configure player's settings */
player.allowsAirPlay = YES;
player.initialPlaybackTime = 10;
player.endPlaybackTime = 20;

/* start playback */
[player play];
```

Taking pictures and videos

- Taking pictures and videos is a (pretty standard) process that involves three steps:
  - creation and (modal) presentation of the camera interface (`UIImagePickerController`)
  - the user takes a pictures/video or cancels
  - the `UIImagePickerController` notifies its delegate about the result of the operation
- A `UIImagePickerController` displays a camera interface with a number of controls:



Taking pictures and videos

- Before using `UIImagePickerController`, the following conditions are required:
  - the device must have a camera (if using the camera is essential for the app, it must be added to the required device capabilities in the Info.plist of the project; if using the camera is optional, the app should have a different flow if a camera is not available)
  - the device's camera must be available for use with the given source type (photo camera, photo library, camera roll), by checking the return value of the method `isSourceTypeAvailable:`
  - a delegate object (conforming to `UIImagePickerControllerDelegate` and `UINavigationControllerDelegate` protocol) to respond to the user's interaction with the image picker controller must have been implemented
- To specify whether the user can take still images, movies, or both, the `mediaTypes` property must be set; `mediaTypes` is a non-empty array whose elements can be the constant values `kUTTypeImage` and `kUTTypeMovie` (link the Mobile Core Services framework to the project and `#import <MobileCoreServices/MobileCoreServices.h>`)

Showing the image picker controller

```
/* assume the view controller conforms to UIImagePickerControllerDelegate and UINavigationControllerDelegate */
- (BOOL)startCameraController{

    /* first check whether the camera is available for use */
    if (([UIImagePickerController isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera] == NO))
        return NO;

    /* create an image picker controller and set its sourceType property */
    UIImagePickerController *camera = [[UIImagePickerController alloc] init];
    camera.sourceType = UIImagePickerControllerSourceTypeCamera;

    /* Displays a control that allows the user to choose picture or movie capture, if both are available */
    camera.mediaTypes = [UIImagePickerController
                         availableMediaTypesForSourceType:UIImagePickerControllerSourceTypeCamera];

    /* hide the controls for moving and scaling pictures */
    camera.allowsEditing = NO;
    /* set the delegate for the image picker controller */
    camera.delegate = self;

    /* present the image picker controller modally */
    camera.modalTransitionStyle = UIModalTransitionStyleCoverVertical;
    [self presentViewController:camera animated:YES completion:nil];

    return YES;
}
```

Implementing delegate methods for the image picker

```
/*  Responding to the user Cancel (dismiss the image picker controller) */
- (void) imagePickerControllerDidCancel:(UIImagePickerController *)picker{
    /*  Dismiss the image picker controller */
    [picker.parentViewController dismissViewControllerAnimated:YES completion:nil];
}

/*  Responding to the user user accepting a newly-captured picture or movie */
/*  The info argument is a dictionary containing the original image and the edited image, if an
image was picked; or a filesystem URL for the movie, if a movie was picked */
- (void) imagePickerController:(UIImagePickerController *)picker
 didFinishPickingMediaWithInfo:(NSDictionary *)info{

    NSString *mediaType = [info objectForKey:UIImagePickerControllerMediaType];

    /*  Handle still image capture */
    if (CFStringCompare((CFStringRef) mediaType, kUTTypeImage, 0) == kCFCompareEqualTo){
        /*  ... */
    }

    /*  Handle video capture */
    if (CFStringCompare((CFStringRef) mediaType, kUTTypeMovie, 0) == kCFCompareEqualTo){
       /*  ... */
    }

    /*  Dismiss the image picker controller */
    [picker.parentViewController dismissViewControllerAnimated:YES completion:nil];

}
```

Picking pictures and videos from the Photo Library

- The `UIImagePickerController` can be used also to pick items from the device's photo library
- The steps are similar to those for capturing media, with some slight differences:
  - use the Camera Roll album or Saved Photos album, or the entire photo library as the media source
  - the user picks previously-saved media instead of capturing new media
- To configure the picker for browsing saved media the `sourceType` property must be set to:
  - `UIImagePickerControllerSourceTypePhotoLibrary` (all photo albums on the device, including the Camera Roll album)
  - `UIImagePickerControllerSourceTypeSavedPhotosAlbum` (access to the Camera Roll album only)

Mobile Application Development

Lecture 23
Sensors and Multimedia