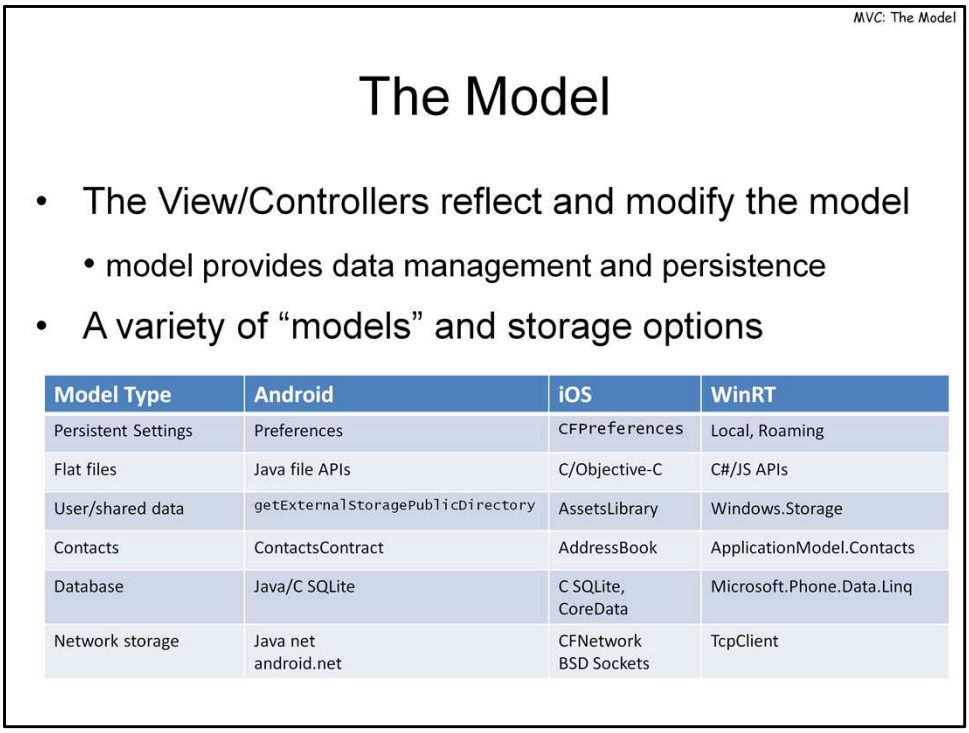
Mobile Frameworks and APIs

Part I: File and Storage Access

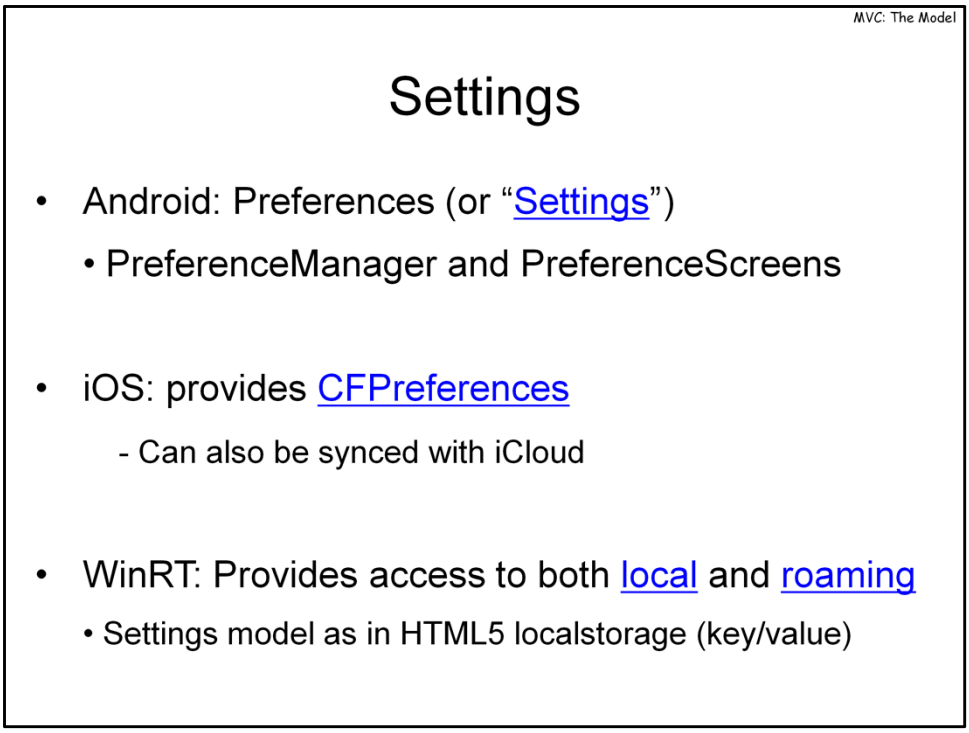
CSCI E-65: Lecture IX – 11/08/13

Table of Contents

MVC: The Model



So far, we've given a lot of attention to the "VC" part of the MVC paradigm – The View/Controllers of Android, iOS, and Windows Mobile. In practice, however, the View/Controllers present and control the data model, and no app can be complete without a robust data model.



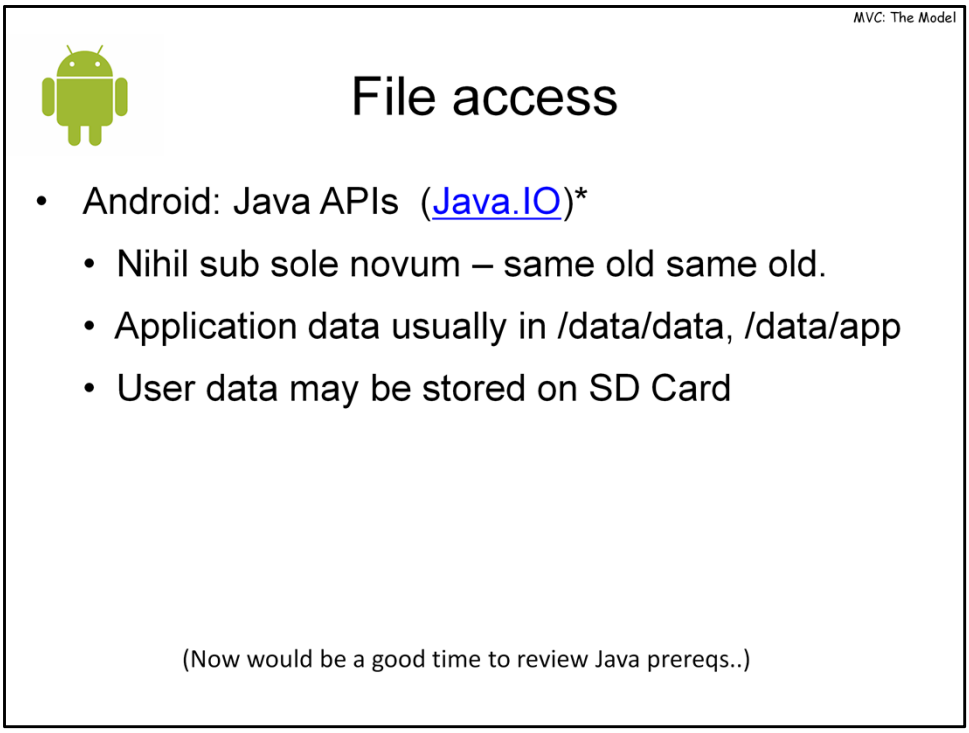
WinRT is the most innovative in its separation of settings (and, in fact, all application data) to "Local" (on-device) and "Roaming" (across device) data. Roaming is a concept Windows had for generations in the enterprise environment, when users logging on to a Windows Domain had their "policy" applied, and their desktops synced, no matter which physical computer they logged on from.

Note, roaming storage is subject to a (really tiny) RoamingStorageQuota (http://msdn.microsoft.com/en-us/library/windows/apps/windows.storage.applicationdata.roamingstoragequota.aspx).

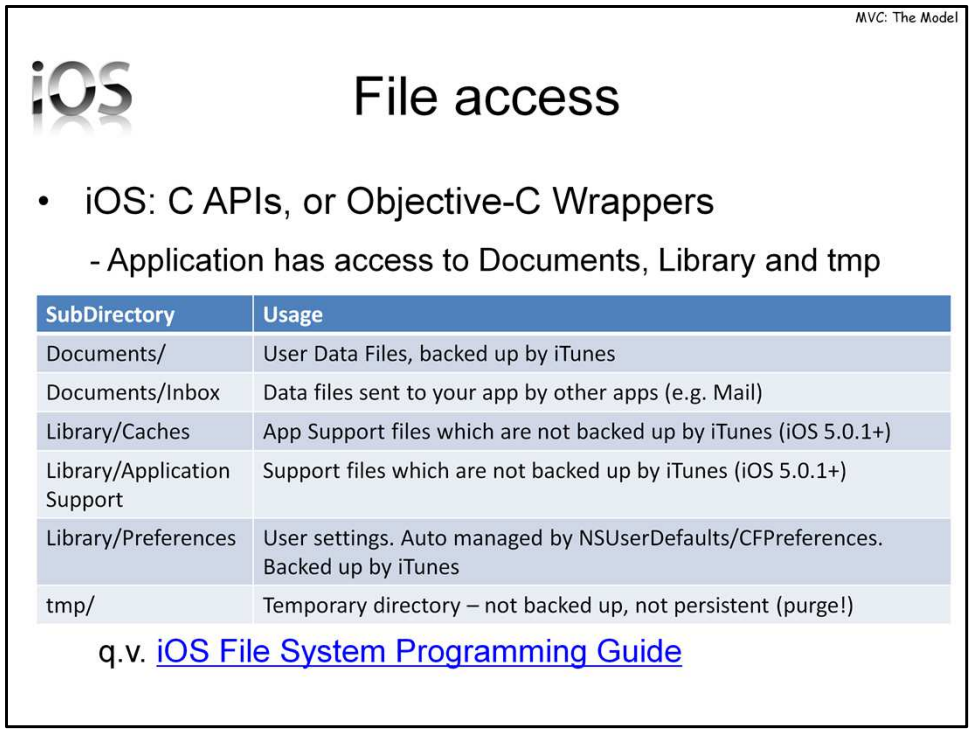
File access

- Android: Java APIs (Java.IO)
- iOS: C APIs, or Objective-C Wrappers
  - Application has access to Documents, Library and tmp
- WinRT: Provides access to Data and *Settings*:
  - Local/Roaming Settings and Folder
  - TemporaryFolder

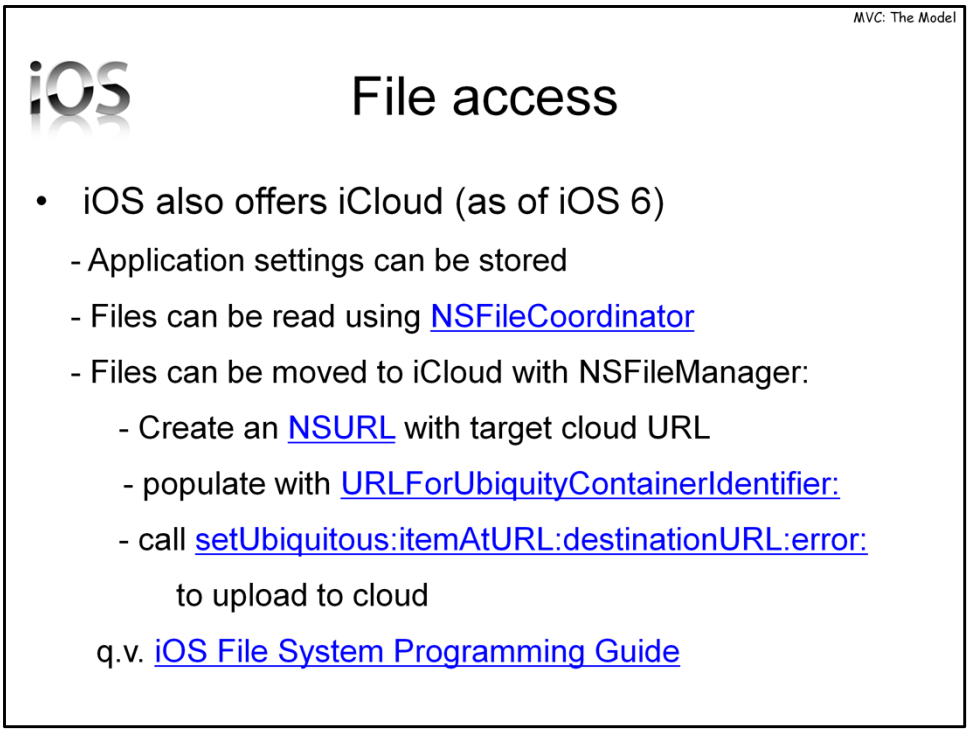
Settings are an often very limited form of storage for applications – they're great for keys and values, but are not suitable for larger amounts of data, or specific types of data, such as images – for this, we need files.



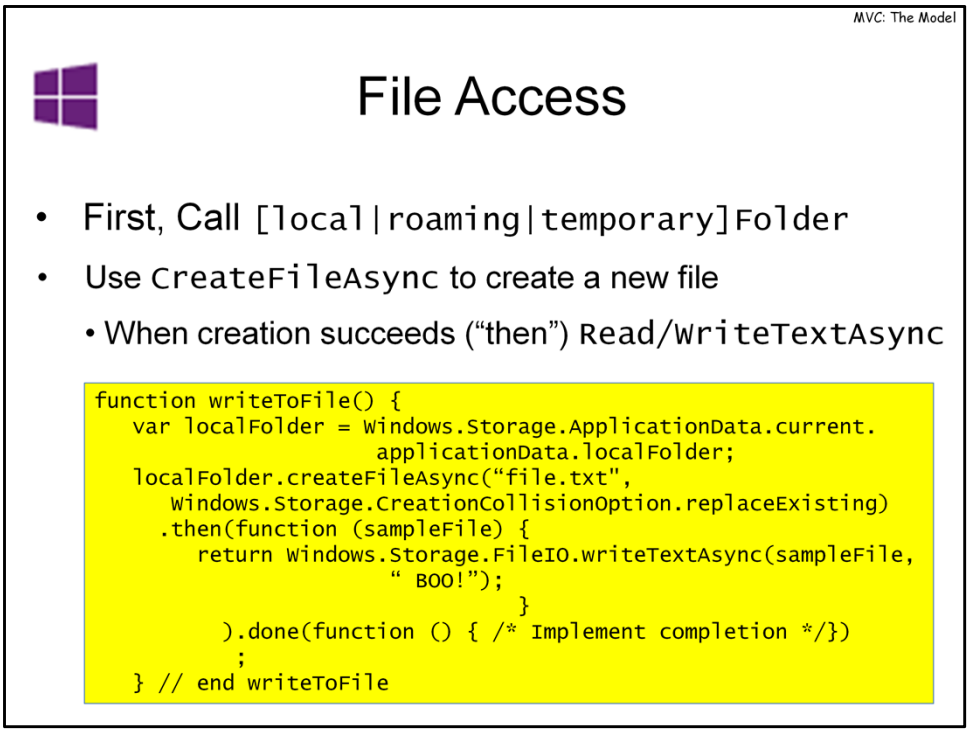
Android relies exclusively on Java's IO package for file access. It is (for the most part) compliant with the Java specification, and there is nothing new here.



In iOS you can use underlying C calls to open files, or you can use Objective-C wrappers like `NSDocument` and friends.



Another feature of iOS is using the cloud (not just any cloud, Apple's own iCloud), to store files ("Documents") and settings in a way which enables syncing across iOS devices – with the same Apple ID.



Windows allows you to read and write data to files in one of three folders – the Local folder (default choice), the temporary folder (Which is not guaranteed persistency between reboots or app restarts) and the Roaming folder (Which may be synced with other WinRT or Win8 devices).

You can listen on changes in the roaming folder (i.e. changes to data shared between devices) using addEventListener("datachanged", datachangeHandler); on Windows.Storage.ApplicationData.

The preferred method of file creation in WinRT is asynchronous. This involves two things:

1) Using "Async" suffixed functions to create, read or write from a file
2) Implementing callback functions and passing them as an argument to the "then" method
3) Implementing completion functions (if you need them) and passing them as an argument to the "done" method.

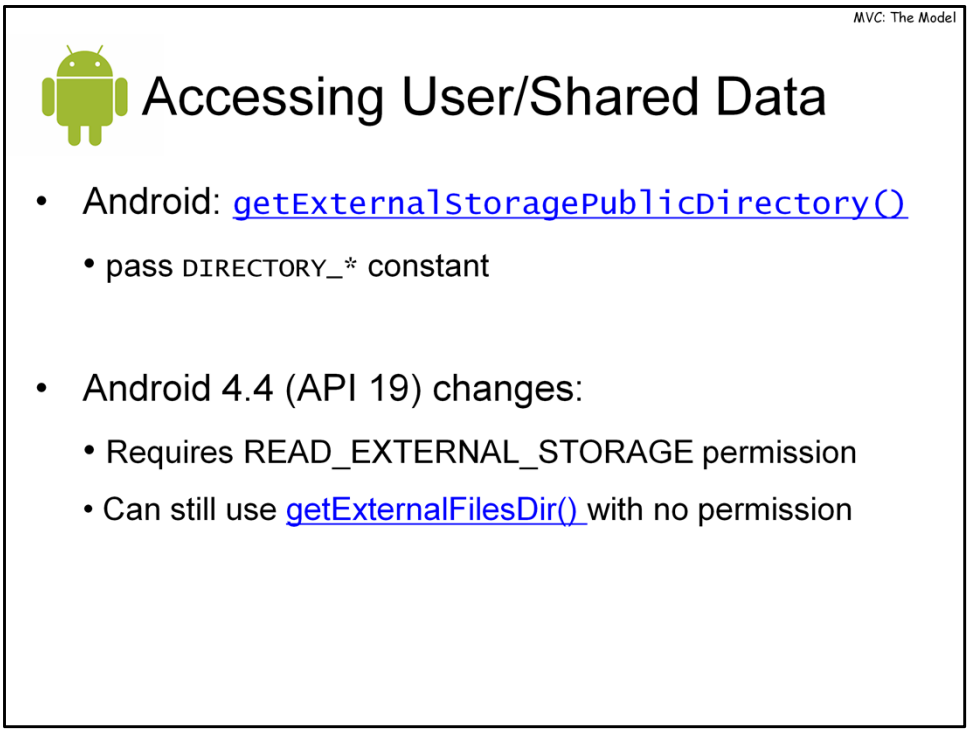
Note this is often done in a fluent manner (remember we discussed the fluency pattern as shown in Android code in a previous lecture). The result is similar to the code shown above.

Accessing User/Shared Data

- Android: `getExternalStoragePublicDirectory()`
  - pass `DIRECTORY_*` constant
- iOS: uses the Assets framework
  - Can only access photos/media, with permission
- WinRT: Provides the windows.storage namespace

Android and WinRT both allow an app to access shared directories and repositories, such as the user's documents, photos, and media files. iOS provides similar functionality with the Assets.framework, but does not support the notion of any external storage.

Let's explore each of these in detail.



In Android, a handle to an external directory call to `Environment.getExternalStoragePublicDirectory()`, with one of the directory constants:

```
field public static java.lang.String DIRECTORY_ALARMS;
field public static java.lang.String DIRECTORY_DCIM;
field public static java.lang.String DIRECTORY_DOWNLOADS;
field public static java.lang.String DIRECTORY_MOVIES;
field public static java.lang.String DIRECTORY_MUSIC;
field public static java.lang.String DIRECTORY_NOTIFICATIONS;
field public static java.lang.String DIRECTORY_PICTURES;
field public static java.lang.String DIRECTORY_PODCASTS;
field public static java.lang.String DIRECTORY_RINGTONES;
```

Note that KitKat now requires permissions for the particular directory constants. If you do not have permissions, you will have to handle a runtime exception. A safer API is to use getExternalFilesDir(), which provides a handle to your application's private storage directory (usually on the SD Card) but does not tell you where it is. You won't need permissions – but you'll have less control over the location of your files.

MVC: The Model

iOS Accessing User/Shared Data

- iOS uses the Assets framework for photo/videos
  - An instance of ALAssetsLibrary provides access
  - Call enumerateGroupsWithTypes:usingBlock:failureBlock:
    - Specify enumerator callback block in usingBlock
    - Specify failure callbackblock in failureBlock
      - (may fail because user denies permission)
- Tip: Use ALAssetLibrary's authorizationStatus (iOS 6.0)

  (use to detect whether or not you are allowed access)

In iOS, sharing is far more restricted than Android. Each application has its own document set (in its sandbox), and there is no direct API to share documents between applications or use a common shared storage like in Android. Instead, you have limited sharing of the user's photos and/or videos, using the AssetsLibrary. Basically, this limited form of sharing allows you to see whichever documents the Photos application can see – provided you've the user's permission – like in the illustration:

The code to enable this functionality is a fairly straightforward sequence of steps:

1) Instantiate an ALAssetsLibrary

```
ALAssetsLibrary *library = [[ALAssetsLibrary alloc] init];
```

2) Call the enumerateGroupsWithTypes library enumerator function to iterate through the assets, or call assetForURL in order to find a particular asset by its URL. The enumeration function is asynchronous, and requires two blocks – one called on enumeration, and one called on failure (which usually implies the user has denied permission – but note you can query [ALAssetsLibrary authorizationStatus] to preempt failure).

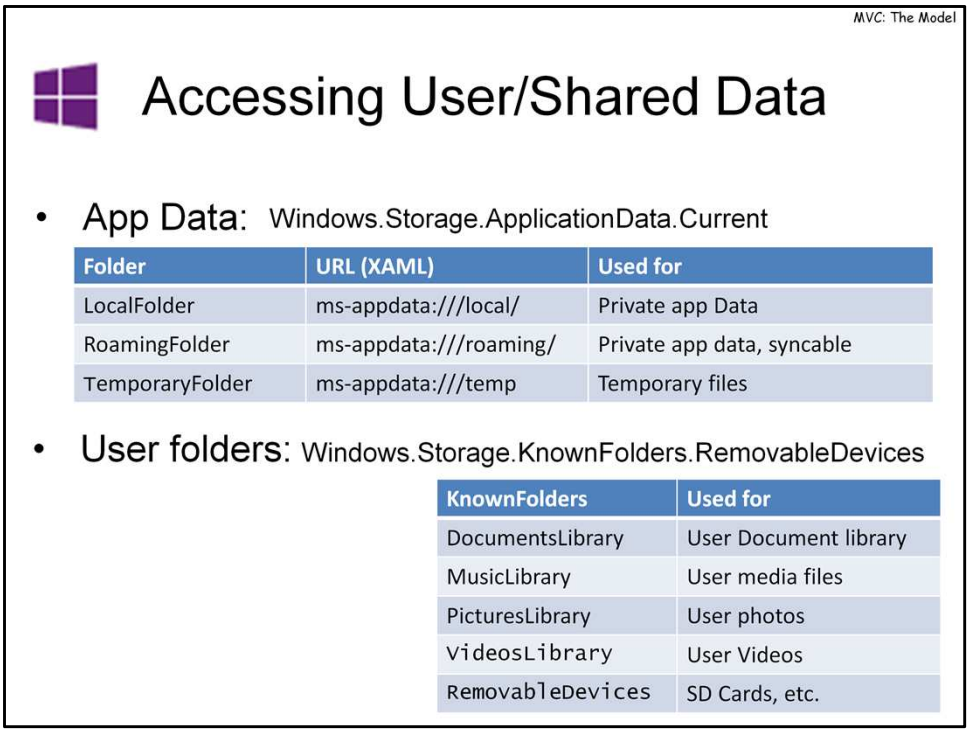
Code to enumerate would look something like this:

```
// Instantiate the assets library
ALAssetsLibrary *library = [[ALAssetsLibrary alloc] init];

// Assets will be added to an array
NSMutableArray  *assets  = [[NSMutableArray alloc]  init];

// Internal enumerator which we will call for each asset in a
// given group
ALAssetsGroupEnumerationResultsBlock assetEnumerator =
   ^(ALAsset *result, NSUInteger index, BOOL *stop) {
    if (result != NULL) {
        // NSLog(@"Asset result: %@", result);
        [assets addObject:result];

    }
};

// This is the enumerator called from the ALAssetsLibrary
ALAssetsLibraryGroupsEnumerationResultsBlock
assetGroupEnumerator = ^(ALAssetsGroup *group, BOOL *stop) {
    if(group != nil) { // NSLog(@"Asset class: ");
        // Call internal enumerator
        [group enumerateAssetsUsingBlock:assetEnumerator];
    }
};

[library enumerateGroupsWithTypes:ALAssetsGroupAlbum
                       usingBlock:assetGroupEnumerator
                       failureBlock: ^(NSError *error) {
                                       NSLog(@"Dang!");
                                                    }
                       ];
```

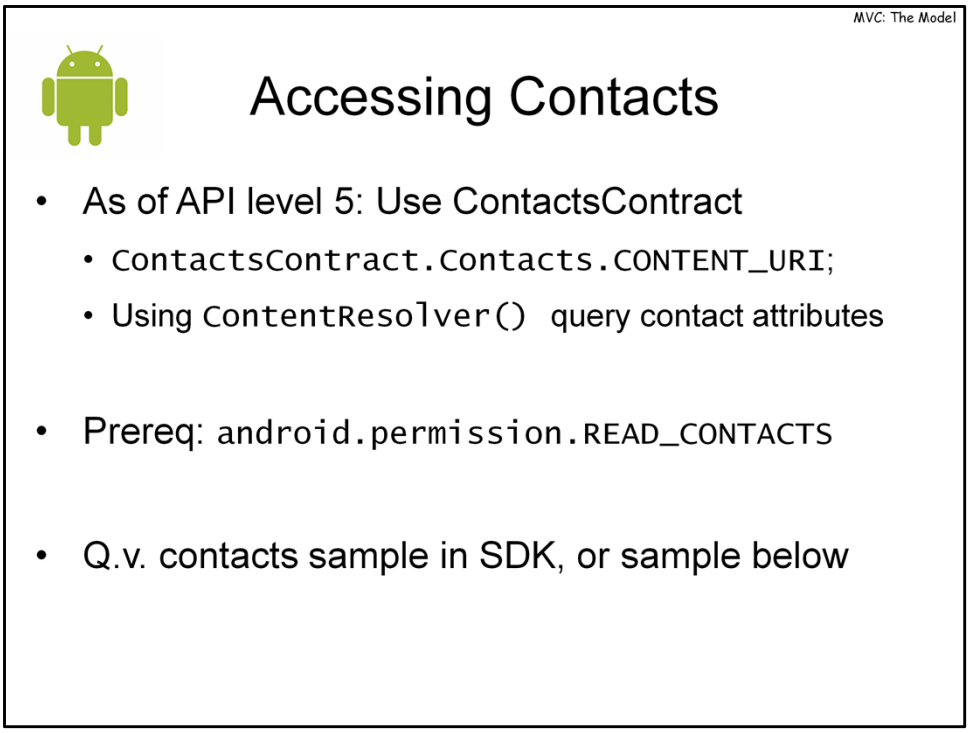
Note that we use two blocks (and an anonymous error block) here:

1) Called from `enumerateGroupsWithTypes:usingBlock`: This block is an `ALAssetsLibraryGroupsEnumerationResultsBlock,` and will be called once per group. The parameters of the block:
   `ALAssetsGroup *group`: A pointer to the group object being enumerated
   `BOOL *stop`: Enabling you to block further enumeration.

2) Called from within the first, will enumerate each asset within the group. It is an `ALAssetsGroupEnumerationResultsBlock` (Note, Group, not Groups). The parameters of the block are:
   `ALAsset *result:` A pointer to the result
   `NSUInteger index:` Index of the result inside the result set
   `BOOL *stop:` Enabling you to block further enumeration



http://lunarfrog.com/blog/2012/05/21/winrt-folders-access/



The following sample code can be used to get the user's contact list, using the ContactsContract API:

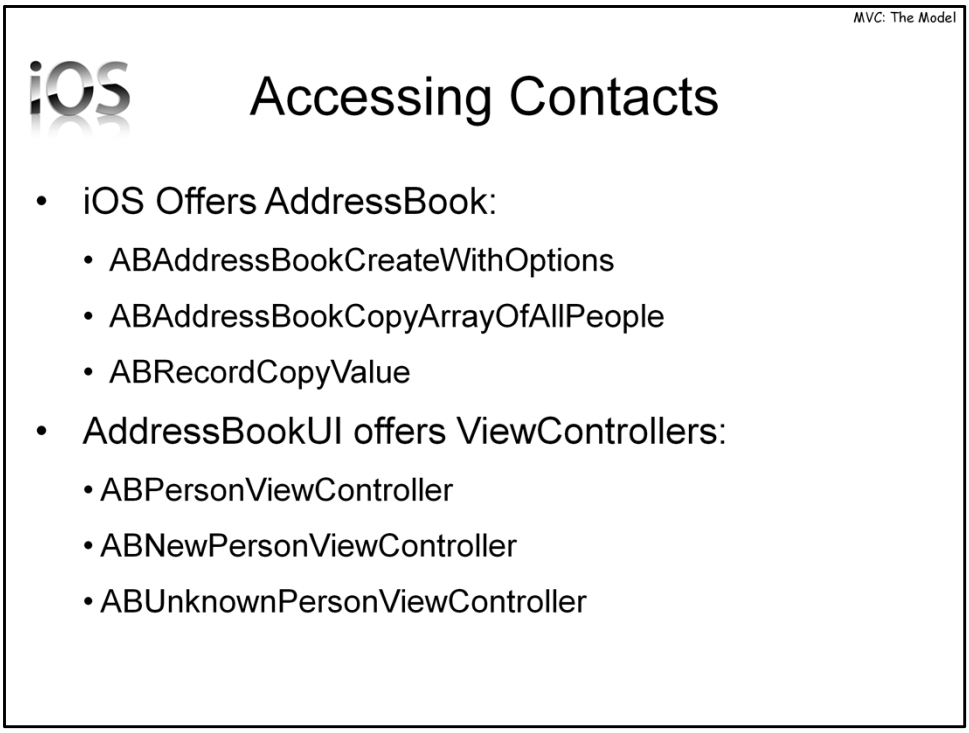
```
public StringBuffer dumpContacts() {
  StringBuffer output = new StringBuffer();
  ContentResolver contentResolver = getContentResolver();
  Cursor cursor=contentResolver.query(ContactsContract.Contacts.CONTENT_URI, // URI
                                      null,                                  // projection
                                      null,                                  // selection
                                      null,                                  // selectionArgs
                                      null);                                 // sortOrder

  // The query returns a "cursor", and you may query it for the count of records returned
  if (cursor.getCount() == 0) {  /* No records… Do something? Return null? You decide.. */ }

  while (cursor.moveToNext()) {
    // Save contact unique ID for later use
    String contact_id = cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts._ID));

     if (cursor.getInt (cursor.getColumnIndex(ContactsContract.Contacts.HAS_PHONE_NUMBER))> 0)
        {
           output.append("\n First Name:" +
              cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME)));

            output.append (doPhoneNumbers(contact_id));
        }
   output.append(doEmailAddresses(contact_id));
   return (output);
} // end fetchContacts
```

What about the phone numbers and email addresses? You might want to think about this before turning to the next page..

```
StringBuffer doPhoneNumbers (String ContactID)
{
       StringBuffer Output = new StringBuffer();
       // Sub query for every phone number this contact may have. Note new cursor..
       Cursor phoneCursor = contentResolver.query(
                 ContactsContract.CommonDataKinds.Phone.CONTENT_URI,         // URI
                 null,                                                       // projection
                 ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = ?", // selection
                 new String[] { ContactID },                                 // selectionArgs
                 null);
       // Sub loop to iterate over phone numbers, in similar manner ..
       while (phoneCursor.moveToNext()) {
             phoneNumber = phoneCursor.getString
              (phoneCursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER));

            output.append("\n Phone number:" + phoneNumber);
            }
       phoneCursor.close();
       return (output);
}
```

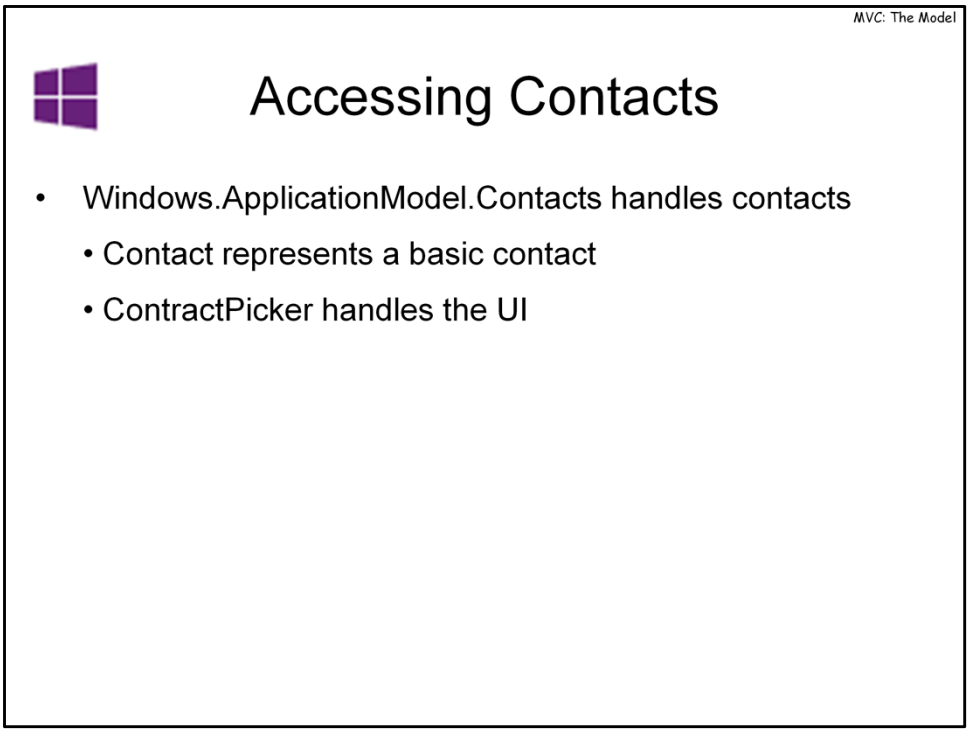
The same cursor idea can be used for phone numbers, as you can see above (note the `selection` and `selectionArgs` are populated and tied to `ContactID`, the argument), and – below – for emails (nearly identical code)

```
StringBuffer doEmailAddresses (String ContactID)
{
        StringBuffer Output = new StringBuffer();
       // Note only parameters modified here are Email.CONTENT_URI and EMAIL_CONTACT_ID.
       Cursor emailCursor = contentResolver.query(
             ContactsContract.CommonDataKinds.Email.CONTENT_URI,
             null,
             ContactsContract.CommonDataKinds.Email.CONTACT_ID + " = ?",
             new String[] { contact_id },
             null);

       while (emailCursor.moveToNext()) {
             output.append ("\nEmail" + emailCursor.getString(emailCursor.getColumnIndex
                                    (ContactsContract.CommonDataKinds.Email.DATA));
       }
       emailCursor.close();
       return(output);
}
```



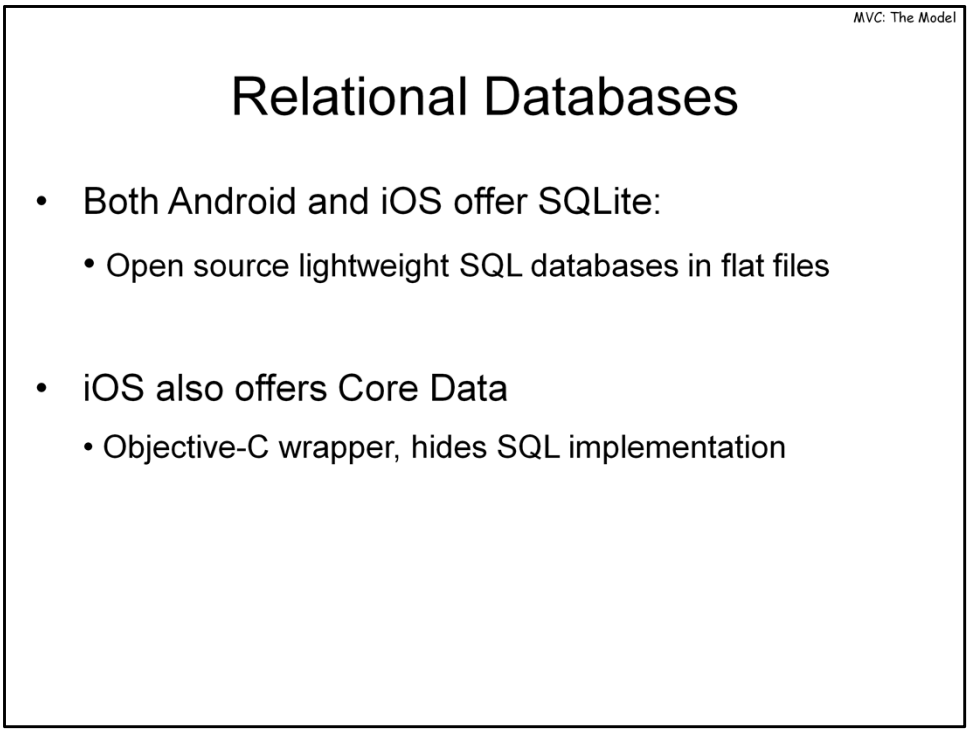
In iOS, There are two separate frameworks dealing with contacts: AddressBook – which provides the programmatic access to the data, and AddressBookUI, which provides custom view controllers for displaying that data. The sample code to use AddressBook (AB) is shown below:

```
- (void)getAddressBook
{
    CFErrorRef error = NULL;
    ABAddressBookRef addressBook = ABAddressBookCreateWithOptions(NULL, &error);

    if (addressBook != nil)
    {
         NSArray *allContacts = (__bridge_transfer NSArray *)ABAddressBookCopyArrayOfAllPeople(addressBook);
         NSUInteger i = 0;
         for (i = 0; i < [allContacts count]; i++)
         {
            ABRecordRef contactPerson = (__bridge ABRecordRef)allContacts[i];

            NSString *firstName = (__bridge_transfer NSString*)
                     ABRecordCopyValue(contactPerson, kABPersonFirstNameProperty);

            NSString *lastName =  (__bridge_transfer NSString *)
                     ABRecordCopyValue(contactPerson, kABPersonLastNameProperty);

           ABMultiValueRef emails = ABRecordCopyValue(contactPerson, kABPersonEmailProperty);

           NSUInteger j = 0;
          for (j = 0; j < ABMultiValueGetCount(emails); j++)
          {
            NSString *email = (__bridge_transfer NSString *)ABMultiValueCopyValueAtIndex(emails, j);
          }
     }
     CFRelease(addressBook);
}
```



http://msdn.microsoft.com/en-us/library/windows/apps/windows.applicationmodel.contacts.aspx

Sample @ http://msdn.microsoft.com/en-US/library/windows/apps/jj153343



Flat files and XML will only get you so far, and do not scale well for large amounts of data. In those caes, a relational, table-based database is preferred. When it comes to relational database support both major OSes utilitze SQLite3, a free open source library. This library is ubiquitous in desktops as well, and is used by browsers as well (for example, to provide the "Web Database" APIs, as well as store their own data, such as cookies and sites visited).

iOS also offers a proprietary wrapper called ***Core Data*.** This is a wrapper that binds directly to SQLite, but does a good job at hiding the underlying implementation, and SQL in general.

SQLite

- Popular, portable solution for relational databases
- Simple, but powerful command prompt
  - Drops into a SQL-command shell
  - maintenance commands prefixed by a "."
- Experiment: Explore MobileSync files with sqlite3
  - Try: iTunes MobileSync directory or on i-Device

SQLite wouldn't be used by both rival OSes unless it were exceptionally simple and straightforward To create and maintain a database, for example, it takes only a few commands :

```
morpheus@Erudite (~)$ sqlite3 /tmp/test
SQLite version 3.7.12 2012-04-03 19:43:07
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> CREATE TABLE foo ( id  INTEGER PRIMARY KEY,
   ...>                    name VARCHAR(20),
   ...>                    desc TEXT);
sqlite> INSERT INTO foo (name, description) VALUES ('Me', 'My description');
sqlite> select * from foo;
1|Me|My description
sqlite> .quit
morpheus@Erudite (~)$ file /tmp/test
/tmp/test: SQLite 3.x database
```

Opening an existing database is just as simple. You can use ".dump" to get the commands used to create it:

```
morpheus@Erudite (~)$ sqlite3 /tmp/test
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE foo ( id INTEGER PRIMARY KEY, name varchar(20), desc TEXT);
INSERT INTO "xx" VALUES(1,'Me','My description');
COMMIT;
sqlite>
```

But just creating your own databases is boring an uninspiring. If you have a i-device, you can easily investigate more interesting databases on it. In fact, you don't need the device to be jailbroken – its data files are backed up anyway. You can inspect the files in `~/Library/Application Support/MobileSync/Backup/`: You should be able to locate a hash directory for every device you've synced. What you find in the directory will be a mix of property list and SQLite databases, belonging to both your installed apps and the built-in apps. The names are also hashed (SHA-1, apparently), though some (like `ca3bc056d4da0bbf88b5fb3be254f`… , a.k.a `notes.db`, for the user notes) are well known.

```
root@erudite (~)# cd ~/Library/Application\ Support/MobileSync/Backup/
root@erudite (..up)# ls
745c1b9302fc096b91b8a173a99ae1f0846dcf41
C87fa7d4895cabdacbe7b5e6996b4e6c8b0111d3
root@erudite (..up)# cd 745c1b9302fc096b91b8a173a99ae1f0846dcf41
root@erudite (..41)# file * | grep SQL |more
0980c97ed02272065fe0462ef9c1c63dd59c7ed2: SQLite 3.x database
0a97fa1d8a196bc77a0dff0e35f374e8fbbdf704: SQLite 3.x database
 ..
af0a461cff85322d0c029fedc42e7841ecbd5b9f: SQLite 3.x database, user version 31
..

root@erudite (..41)# for f in `file * | grep SQL | cut -d':' -f1`; do echo
".dump" | sqlite3 $f; done | grep _something_of_interest_
```

Using "`sqlite3`" on any of the files, followed by "`.dump`", as shown above (in a script snippet on all files) will dump both the database schema and the values. Using "grep" can then isolate interesting tidbits of information. This technique is used very often by forensics analysts and law enforcement officials (incidentally, without a need for a warrant) to get phone call records, chats, texts, places where the phone has been , wi-fi you have connected to, music you've been listening to, websites you've perused, and so much more: today's mobile phones keep a plethora of information. If the phone isn't passcode locked, all they need to do is connect it to iTunes. MobileSync will do the rest.

On the device itself, most of the SQLite database carry the "`.sqlite3db`" extension. If your device is jailbroken, you can try "`find . –name "*.sqlite3db`" on the device itself.

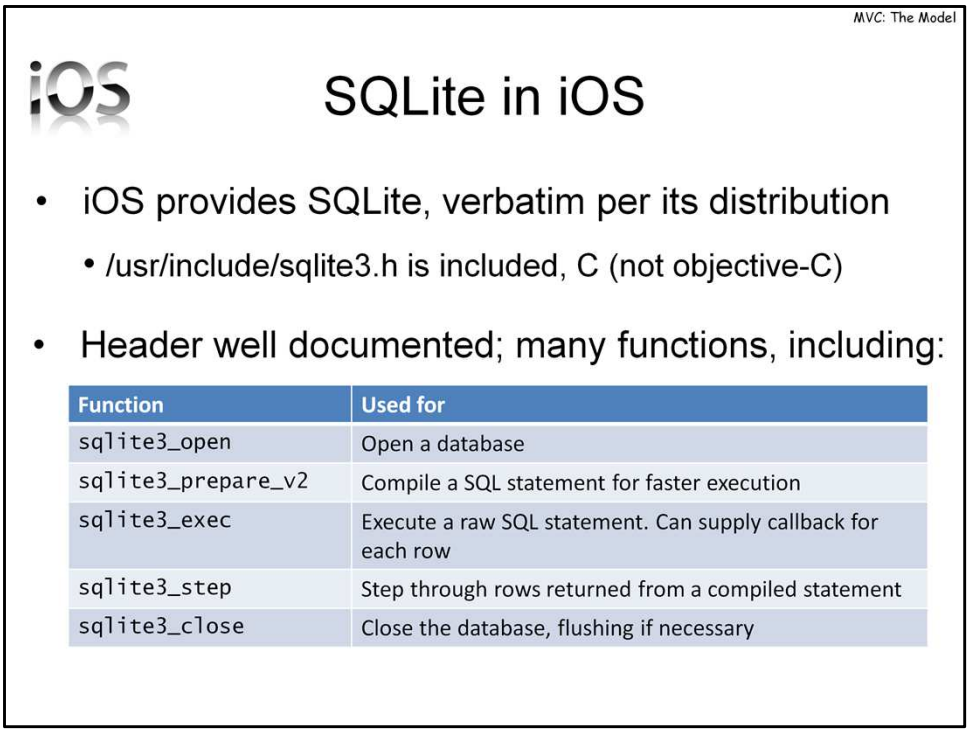
MVC: The Model

SQLite in Android

- Android provides `SQLiteOpenHelper`
  - subclass, pass through constructor, extend OnCreate
- Methods operate on `SQLiteDatabase`
- Methods return read/write "`Cursor`" object
  - Developer needs to implement CRUD operations
- Implementation tied to `libsqlite3.so` via JNI.
- Tutorial: http://www.androidhive.info/2011/11/android-sqlite-database-tutorial/

Android wraps the SQLite APIs with Java objects.

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
                "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
                KEY_WORD + " TEXT, " +
                KEY_DEFINITION + " TEXT);";

    DictionaryOpenHelper(Context context) {
        super(context,
              DATABASE_NAME,
              null,
                 DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```



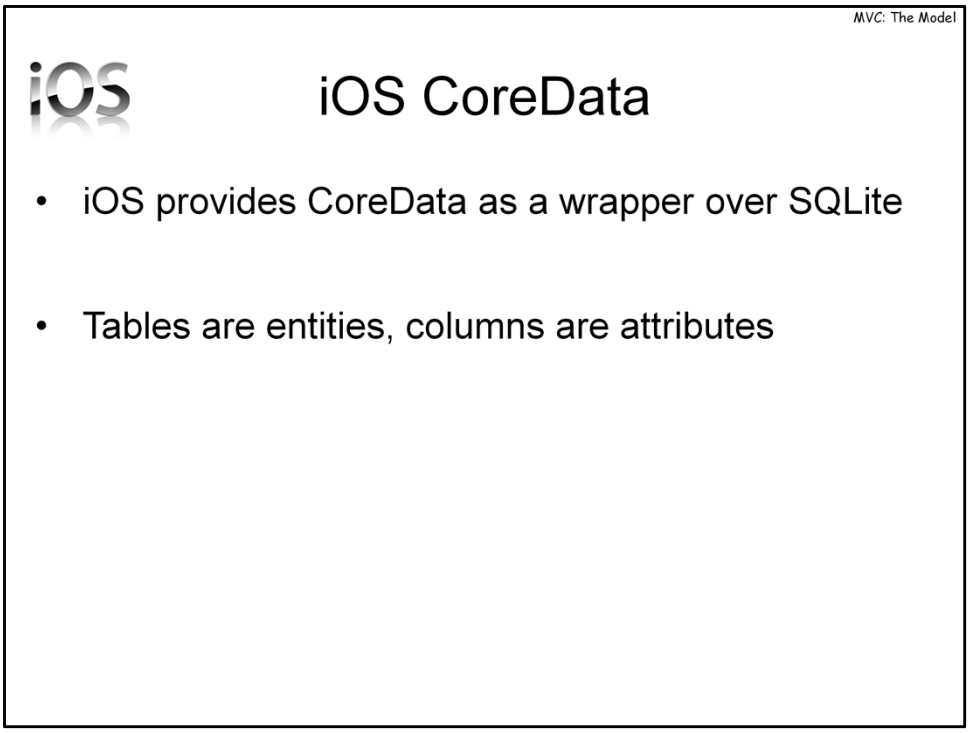
In iOS access to SQLite from objective-C is actually performed through C bindings. You will need to #import <sqlite3.h> , which is the standard open source header. The header is exceedingly well documented, and the reader is encouraged to check it (it is included in virtually every Linux or OS X with gcc installed). Its useful methods are shown below:

```
SQLITE_API int sqlite3_open(
  const char *filename,   /* Database filename (UTF-8) */
  sqlite3 **ppDb);         /* OUT: SQLite db handle */

SQLITE_API int sqlite3_prepare_v2(
  sqlite3 *db,            /* Database handle */
  const char *zSql,       /* SQL statement, UTF-8 encoded */
  int nByte,              /* Maximum length of zSql in bytes. */
  sqlite3_stmt **ppStmt,  /* OUT: Statement handle */
  const char **pzTail);     /* OUT: Pointer to unused portion of zSql */

SQLITE_API int sqlite3_step(sqlite3_stmt*);

SQLITE_API int sqlite3_column_int(sqlite3_stmt*, int iCol);

SQLITE_API const unsigned char *sqlite3_column_text(sqlite3_stmt*, int iCol);

SQLITE_API int sqlite3_finalize(sqlite3_stmt *pStmt);

SQLITE_API int sqlite3_close(sqlite3 *);
```

The following simple example demonstrates its usage to read rows from some database (specified by dbPath), and some table ("myTable"). Note the example is in Objective-C, but the bulk of the work is carried out by C statements:

```
-(int) readDatabase:NSString dbPath
{
  // Setup the database object
  sqlite3 *database;

  // Open the database from the users filesystem
  if(sqlite3_open([dbPath UTF8String], &database) == SQLITE_OK)
  {
   // Compiling SQL isn't strictly mandatory, but helps performance
   const char *sql = "select * from myTable";
   sqlite3_stmt *compiledStatement;
   if (sqlite3_prepare_v2(database,
                          sql,
                          -1,
                         &compiledStatement,
                          NULL) != SQLITE_OK) { /* error */ }

   // iterate through results
   while(sqlite3_step(compiledStatement) == SQLITE_ROW) {
         // Read the data from the result rows
         char *col1 = sqlite3_column_text(compiledStatement, 1);
         char *col2 = sqlite3_column_text(compiledStatement, 2);

         // The char * can be converted to NSString, for example:
         NSString *col1NSstr = [NSString stringWithUTF8String:col1];

        } // end while

   // Release the compiled statement from memory
   sqlite3_finalize(compiledStatement);
   sqlite3_close(database);
   return (0);
}
```

Aside from the minor overhead of wrapping the C datatypes in the objective-C wrappers, working with SQLite directly is simple – and in fact portable to Android, if native code is used in the latter.



For those developers not wishing to use C or be too closely coupled to the database implementation, iOS provides a powerful abstraction layer called **Core Data**.

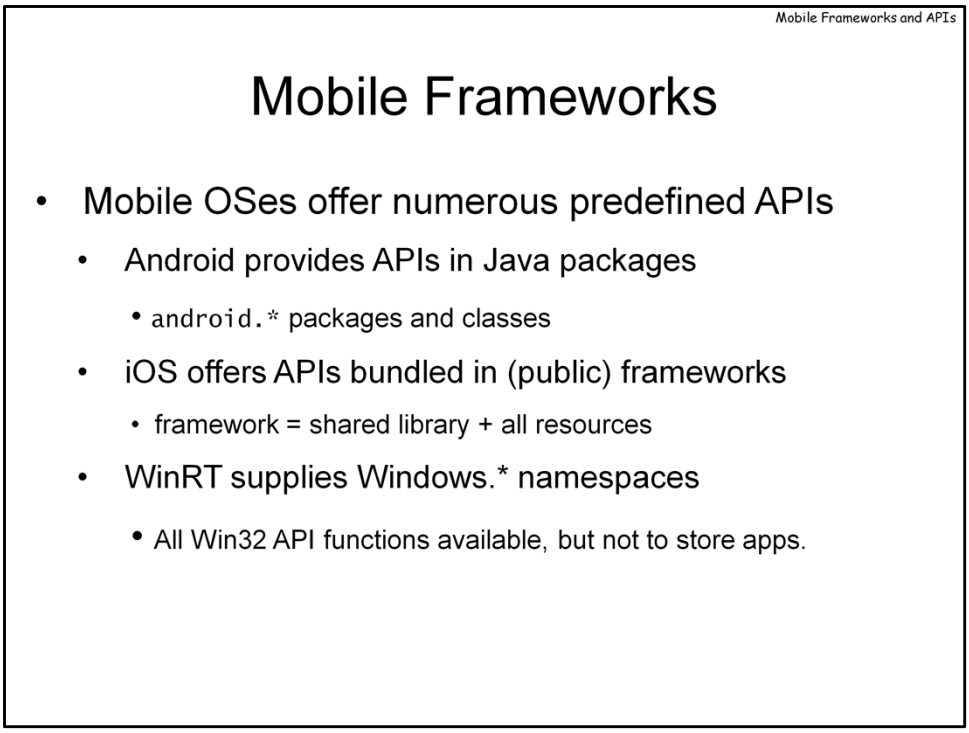
Good references on iOS CoreData can be found in:

- Apple's Core Data Tutorial
- Dr. Dobbs' Journal article on Core Data (http://www.drdobbs.com/database/understanding-core-data-on-ios/240004648),
- Stackmob tutorial (https://blog.stackmob.com/2012/11/iphone-database-tutorial-part-1-learning-core-data/).

Mobile Frameworks and APIs

Part II: Frameworks – A tour

CSCI E-65: Lecture X – 11/15/13



All modern operating systems promote rapid application development by providing useful APIs for common programming tasks, and mobile OSes are no exception. In fact, since mobile devices possess even more features than desktops (such as location, touch support and sensors), mobile APIs are often far richer than those of their counterpart OSes.

Support for APIs in all MOSes comes in the form of packages or libraries, which may be linked with the application code to expose various classes and objects.

Android Packages

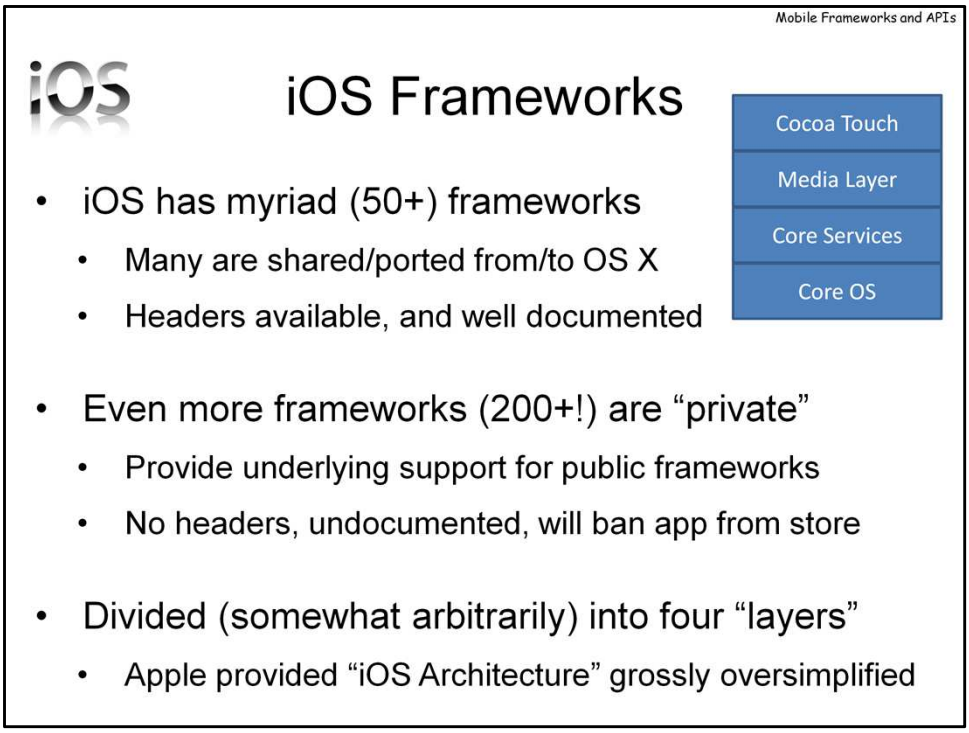
android.* Package	Usage
accessibilityservice	Accessibility options
accounts	Authentication support
animation	Animation effects (via XML)
app	Admin, backup
appwidget	Application widget support
bluetooth	Bluetooth support
content	Device content. Include pm and res sub-packages
database	Database support. Includes sqlite sub-package
drm	Digital Rights-Management
gesture	Gesture and multi-touch support
graphics	Drawable
hardware	Sensor support. Also includes display, input, usb sub-packages

Android Packages

android.* Package	Usage
inputmethodservice	Keyboard input methods (rarely used)
location	Location services (LocationManager)
media	audiofx,effect
mtp	Media Transfer Protocol and Picture Transfer Protocol support
net	Wrapper over Java network sockets, and higher level protocols. Sub-packages include http, nsd, rtp, sip, and wifi.
nfc	Near Field Communication
opengl	Wrapper over OpenGLES.
os	Android system. Includes storage sub-package
preference	System and user-defined preferences
provider	System content providers (MediaStore, Contacts, Calendar..)
renderscript	Renderscript support (3D Graphics rendering)
sax	SAX XML parsing

Android Packages

android.* Package	Usage
security	Security support
service	Dreams, textservice, wallpaper
speech	Speech support. Include tts (Text-To-Speech) sub-package
support	Packages for backward compatibility
telephony	Telephony support. Sub packages include gsm, cdma
test	Test Suites
text	Sub-packages include format, method, style, util
util	XML, date, integer and string manipulation , base64, etc
view	The View/Controller collection. Sub-packages include accessibility, animation, inputmethod and textservice
webkit	WebKit (browser) interfaces
widget	Display widgets

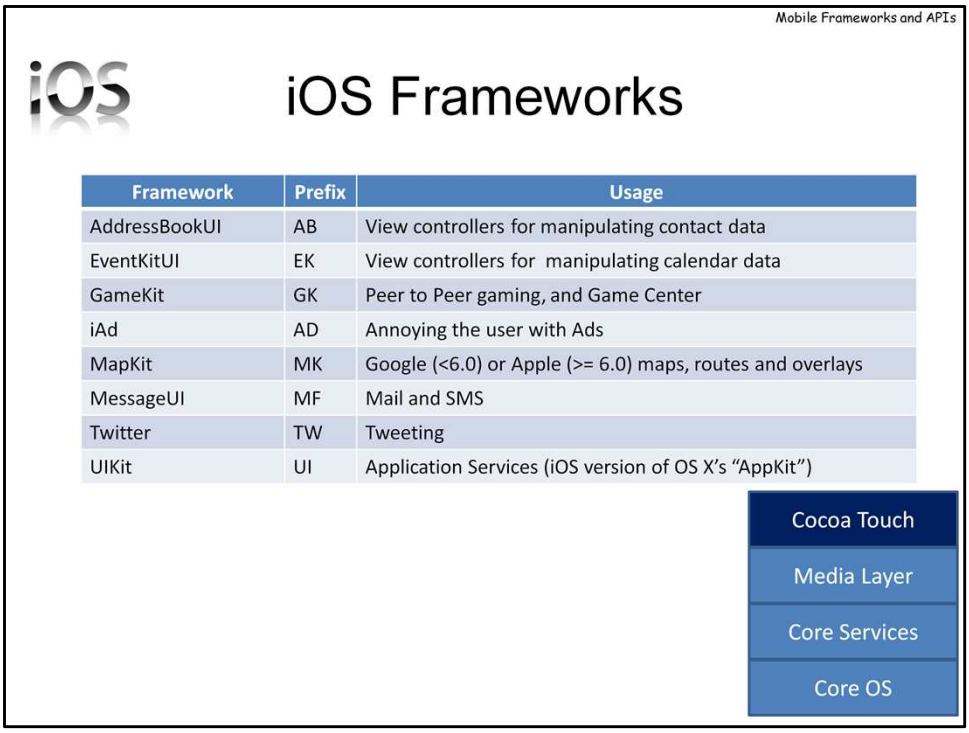


iOS has myriad frameworks sporting a wide variety of features. Apple makes the distinction between "Public" frameworks (in `/System/Library/Frameworks`), and "Private" frameworks (`/System/Library/PrivateFrameworks`). You can see the supported frameworks on any device, or – for lack of one – you can inspect the iPhone SDK. Doing so will reveal the following:

```
morpheus@Erudite (..)# pwd
/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk/System/Library
bash-3.2# ls Frameworks
AVFoundation.framework      CoreMIDI.framework          MediaToolbox.framework
Accelerate.framework        CoreMedia.framework         MessageUI.framework
Accounts.framework          CoreMotion.framework        MobileCoreServices.framework
AdSupport.framework         CoreTelephony.framework     NewsstandKit.framework
AddressBook.framework       CoreText.framework          OpenAL.framework
AddressBookUI.framework     CoreVideo.framework         OpenGLES.framework
AssetsLibrary.framework     EventKit.framework          PassKit.framework
AudioToolbox.framework      EventKitUI.framework        QuartzCore.framework
AudioUnit.framework         ExternalAccessory.framework QuickLook.framework
CFNetwork.framework         Foundation.framework        Security.framework
CoreAudio.framework         GLKit.framework             Social.framework
CoreBluetooth.framework     GSS.framework               StoreKit.framework
CoreData.framework          GameKit.framework           SystemConfiguration.framework
CoreFoundation.framework    IOKit.framework             Twitter.framework
CoreGraphics.framework      ImageIO.framework           UIKit.framework
CoreImage.framework         MapKit.framework            VideoToolbox.framework
CoreLocation.framework      MediaPlayer.framework       iAd.framework
```

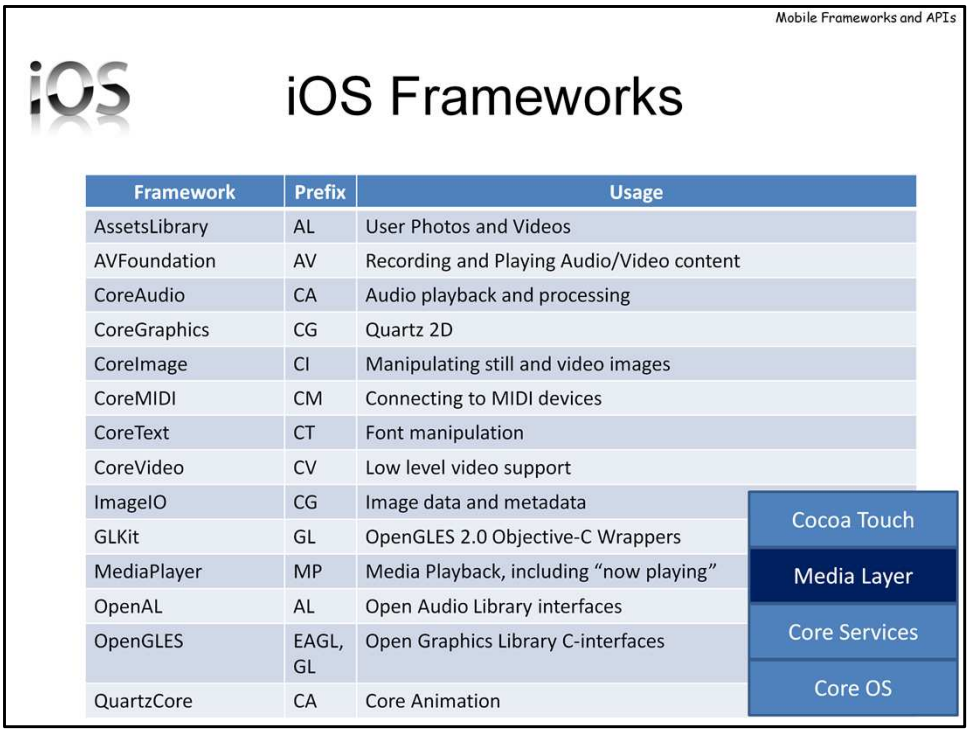
Using otool(1) or jtool with the –L switch will reveal the many dependencies the frameworks have on private frameworks. One such framework, in particular, is `UIKit`:

```
morpheus@Erudite (…)$ jtool -L AVFoundation.framework/UIKit  |  grep Priv
/System/Library/PrivateFrameworks/UIFoundation.framework/UIFoundation
/System/Library/PrivateFrameworks/IOSurface.framework/IOSurface
/System/Library/PrivateFrameworks/MobileKeyBag.framework/MobileKeyBag
/System/Library/PrivateFrameworks/MobileAsset.framework/MobileAsset
/System/Library/PrivateFrameworks/TelephonyUtilities.framework/TelephonyUtilities
/System/Library/PrivateFrameworks/WebBookmarks.framework/WebBookmarks
/System/Library/PrivateFrameworks/BackBoardServices.framework/BackBoardServices
/System/Library/PrivateFrameworks/XPCObjects.framework/XPCObjects
/System/Library/PrivateFrameworks/DictionaryServices.framework/DictionaryServices
/System/Library/PrivateFrameworks/GraphicsServices.framework/GraphicsServices
/System/Library/PrivateFrameworks/SpringBoardServices.framework/SpringBoardServices
/System/Library/PrivateFrameworks/AppSupport.framework/AppSupport
/System/Library/PrivateFrameworks/WebKit.framework/WebKit
/System/Library/PrivateFrameworks/WebCore.framework/WebCore
/System/Library/PrivateFrameworks/ProofReader.framework/ProofReader
/System/Library/PrivateFrameworks/PrintKit.framework/PrintKit
/System/Library/PrivateFrameworks/UIFoundation.framework/UIFoundation
/System/Library/PrivateFrameworks/IOSurface.framework/IOSurface
/System/Library/PrivateFrameworks/MobileKeyBag.framework/MobileKeyBag
/System/Library/PrivateFrameworks/MobileAsset.framework/MobileAsset
/System/Library/PrivateFrameworks/TelephonyUtilities.framework/TelephonyUtilities
/System/Library/PrivateFrameworks/WebBookmarks.framework/WebBookmarks
/System/Library/PrivateFrameworks/BackBoardServices.framework/BackBoardServices
/System/Library/PrivateFrameworks/XPCObjects.framework/XPCObjects
/System/Library/PrivateFrameworks/DictionaryServices.framework/DictionaryServices
/System/Library/PrivateFrameworks/GraphicsServices.framework/GraphicsServices
/System/Library/PrivateFrameworks/SpringBoardServices.framework/SpringBoardServices
/System/Library/PrivateFrameworks/AppSupport.framework/AppSupport
/System/Library/PrivateFrameworks/WebKit.framework/WebKit
/System/Library/PrivateFrameworks/WebCore.framework/WebCore
/System/Library/PrivateFrameworks/ProofReader.framework/ProofReader
/System/Library/PrivateFrameworks/PrintKit.framework/PrintKit
```

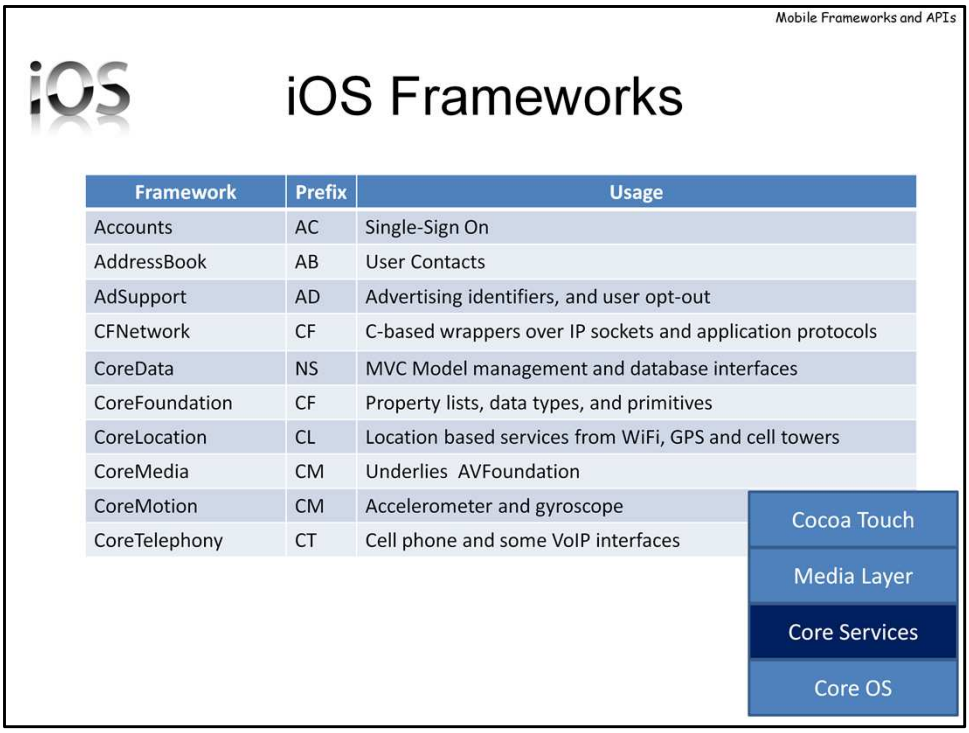


While on disk they are very neatly packaged and largely self-contained, from an API perspective the iOS frameworks are not as neatly defined as Android's packages. Because Objective-C uses a flat namespace, framework functions are identified by a two letter uppercased prefix.

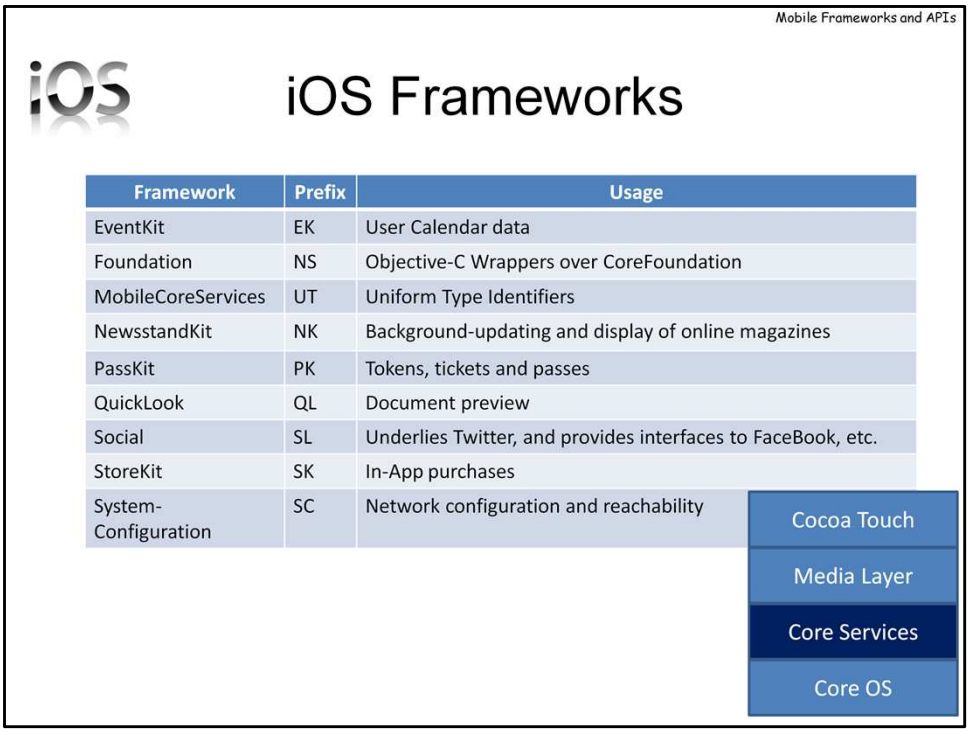
The "Cocoa Touch" frameworks are primarily involved with View/Controllers, and as was previously discussed most of those are pacakged in UIKit (The parallel of Mac OS X's AppKit).

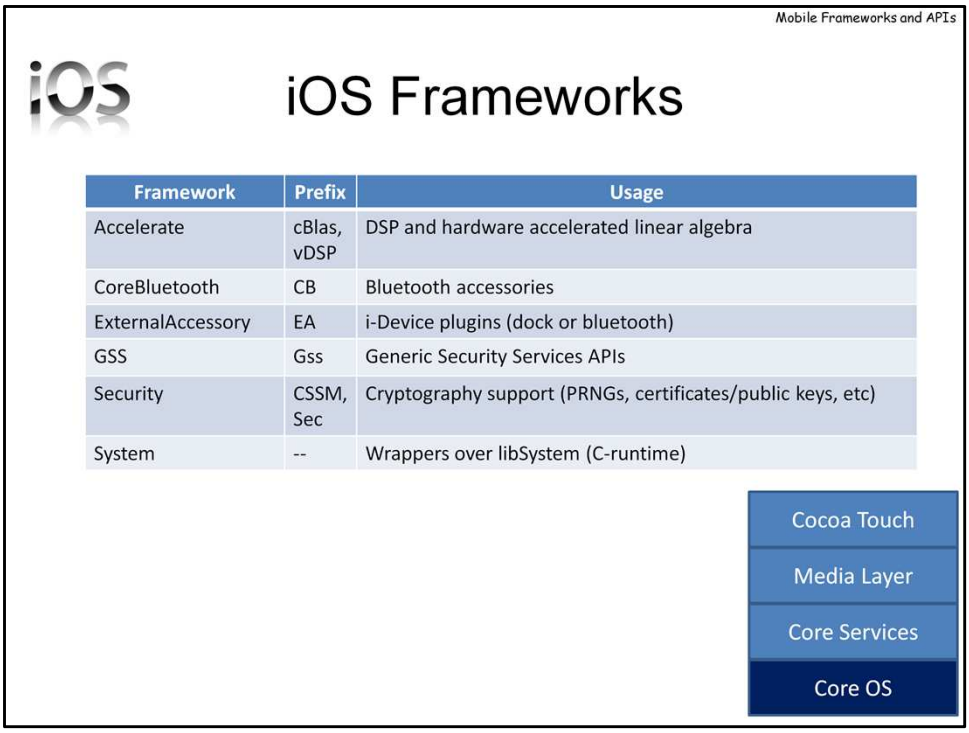


The "Media Layer" frameworks deal with graphics, audio, and video.



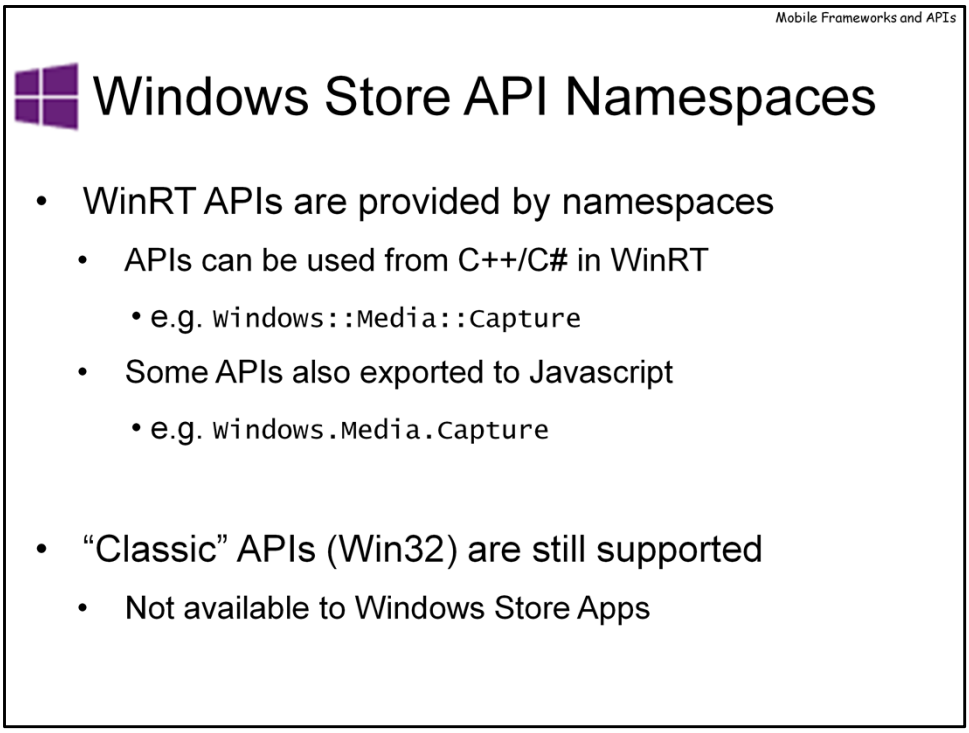
The so called "Core Services" provide access to various support features, but do not provide much UI (for the most part – `AddressBook`, `EventKit` and several others do have specific UI component, as does `QuickLook`, though Apple separates those into "`AddressBookUI`", "`EventKitUI`" , etc, presenting them (somewhat inaccurately) as part of Cocoa Touch, even though they have the same package prefix).


Mobile Frameworks and APIs
iOS
iOS Frameworks
Framework
Prefix
Usage
EventKit
EK
User Calendar data
Foundation
NS
Objective-C Wrappers over CoreFoundation
MobileCoreServices
UT
Uniform Type Identifiers
NewsstandKit
NK
Background-updating and display of online magazines
PassKit
PK
Tokens, tickets and passes
QuickLook
QL
Document preview
Social
SL
Underlies Twitter, and provides interfaces to FaceBook, etc.
StoreKit
SK
In-App purchases
System-Configuration
SC
Network configuration and reachability
Cocoa Touch
Media Layer
Core Services
Core OS



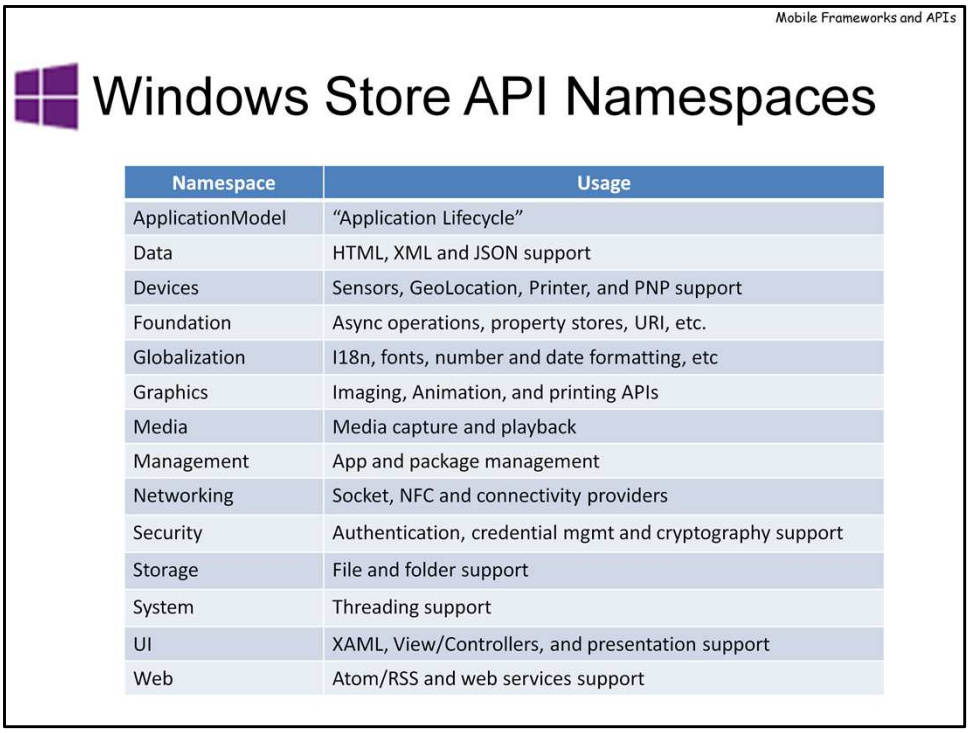
The "Core OS" frameworks are, per Apple, the "lowest level" of APIs provided by the frameworks, though in practice it's only one library (System) which holds this title rightfully. The libSystem.dylib is, in addition to the C-runtime library, a wrapper over much additional functionality:

```
(…)$ cd /Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk
(…)$ cd  ls usr/lib/system/
libcache.dylib          libkeymgr.dylib         libsystem_kernel.dylib
libcommonCrypto.dylib   liblaunch.dylib         libsystem_m.dylib
libcompiler_rt.dylib    libmacho.dylib          libsystem_network.dylib
libcopyfile.dylib       libremovefile.dylib     libsystem_notify.dylib
libcorecrypto.dylib     libsystem_blocks.dylib  libsystem_sandbox.dylib
libdispatch.dylib       libsystem_c.dylib       libunwind.dylib
libdnsinfo.dylib        libsystem_dnssd.dylib   libxpc.dylib
libdyld.dylib           libsystem_info.dylib
```



WinRT provides access to all of the runtime abilities by means of namespaces – somewhat similar to Android. These namespaces are accessed through COM objects, which make them available across several languages and development platforms – though Microsoft is promoting the use of Javascript and .Net (i.e. C#) for WinRT development.

It should be noted that these namespaces are merely wrappers over the native Win32 (or WoW64) APIs. As per the requirements of the Windows Store, however, only WinRT APIs are allowed, meaning that using Win32/WoW64 will disqualify the app from the Windows Store.

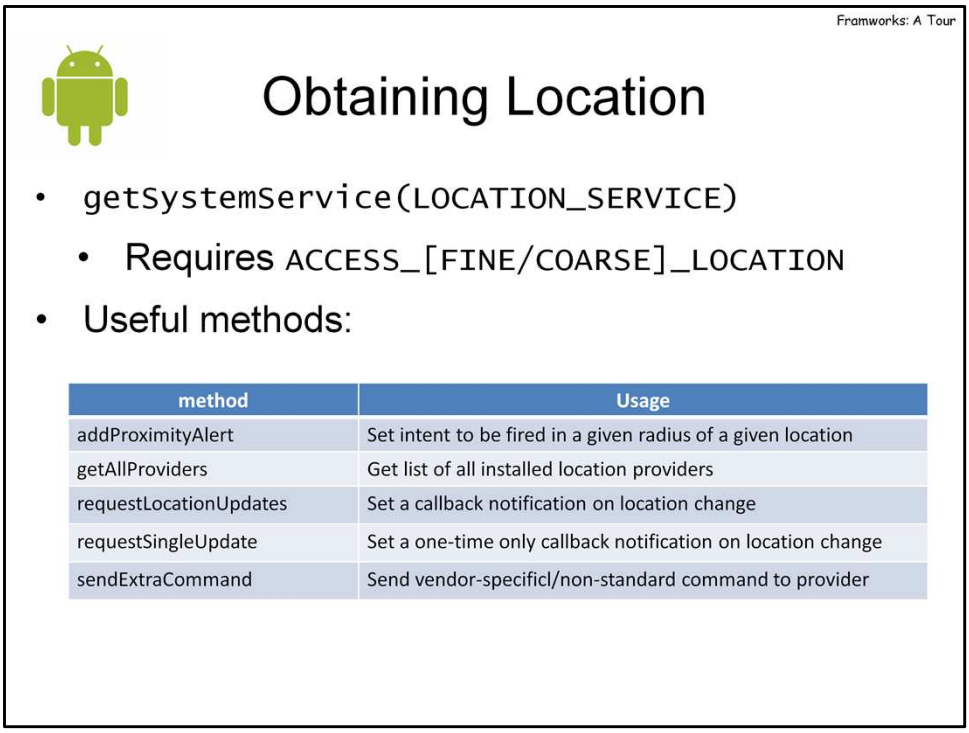


Namespace	Usage
ApplicationModel	"Application Lifecycle"
Data	HTML, XML and JSON support
Devices	Sensors, GeoLocation, Printer, and PNP support
Foundation	Async operations, property stores, URI, etc.
Globalization	I18n, fonts, number and date formatting, etc
Graphics	Imaging, Animation, and printing APIs
Media	Media capture and playback
Management	App and package management
Networking	Socket, NFC and connectivity providers
Security	Authentication, credential mgmt and cryptography support
Storage	File and folder support
System	Threading support
UI	XAML, View/Controllers, and presentation support
Web	Atom/RSS and web services support

Listing of Windows namespaces can be found at **http://msdn.microsoft.com/en-us/library/windows/apps/br211377.aspx**

Mobile Frameworks: A Tour

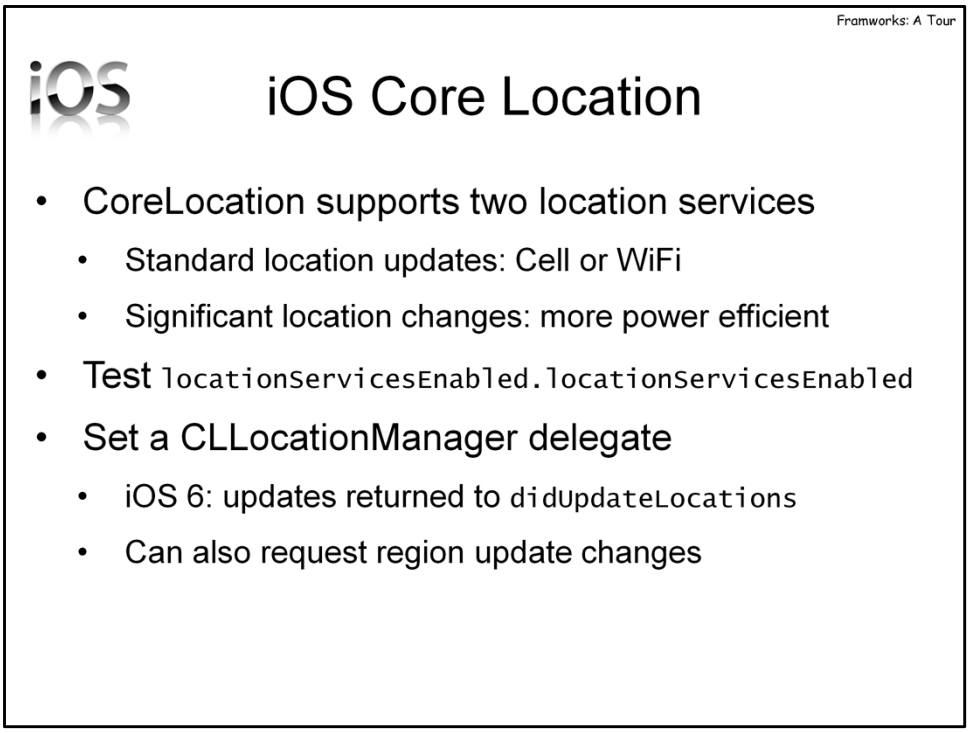
Common tasks: Location, Maps and Sensors



http://developer.android.com/reference/android/location/LocationManager.html

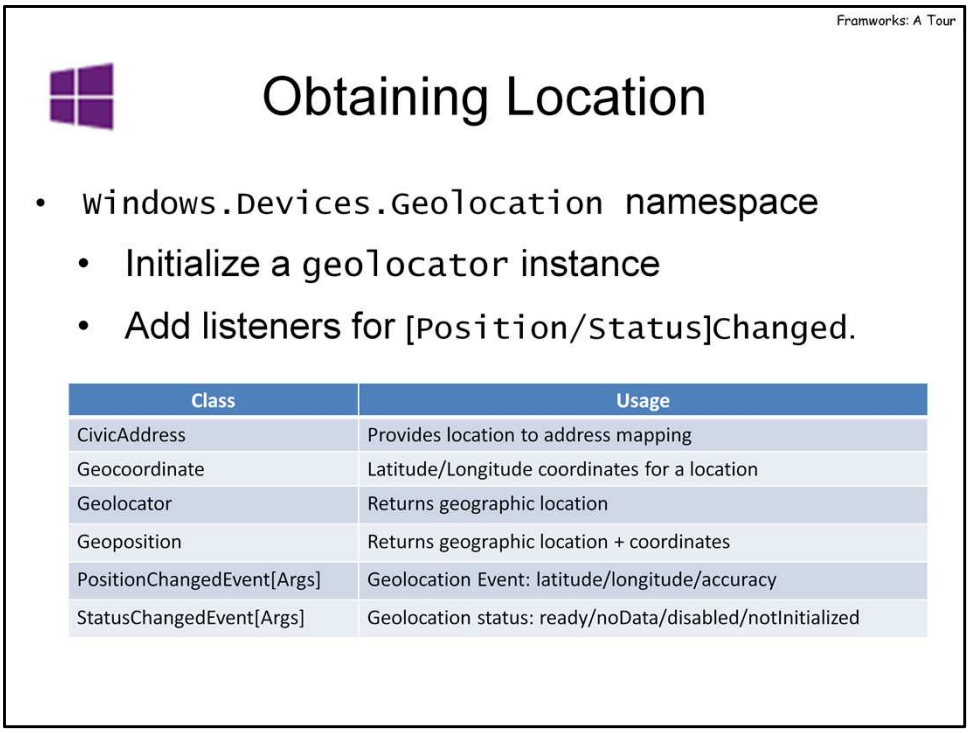
Instantiated through system server: Context.getSystemService(Context.LOCATION_SERVICE).

Permissions: ACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION



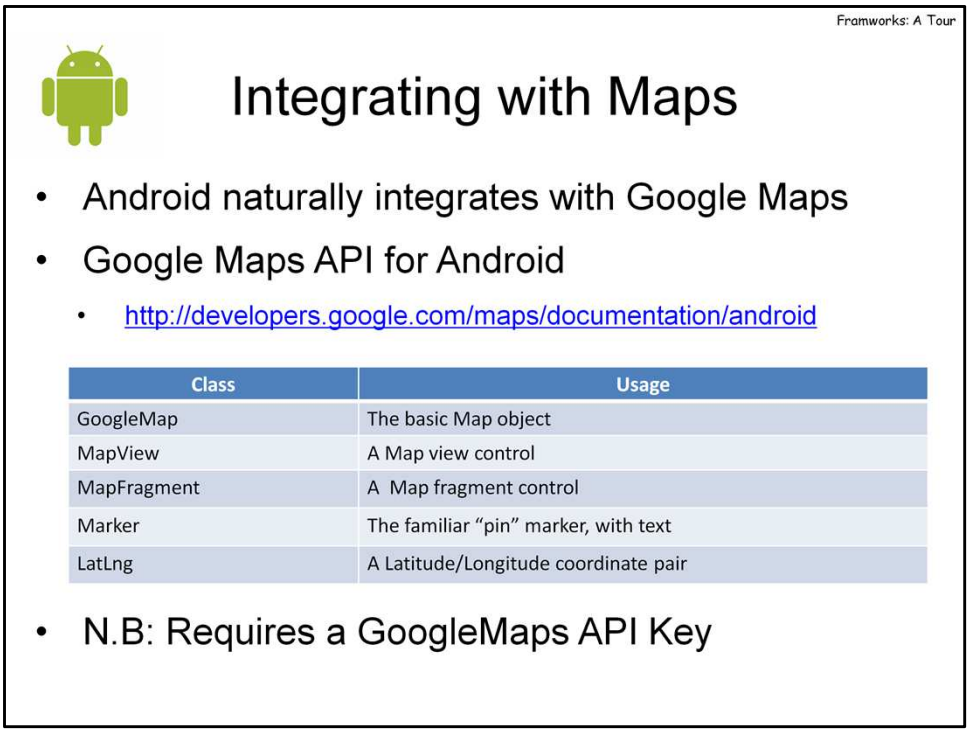
The Apple Developer "Location Awareness Programming Guide" contains many more details on location services.

```
- (void) startLocationServices
{

    if (nil == locationManager)
        locationManager = [[CLLocationManager alloc] init];

    locationManager.delegate = self;

    if (wantSignificant)
    [locationManager startMonitoringSignificantLocationChanges];

    else // want Standard:
      {
        // Set accuracy and a filter
        locationManager.desiredAccuracy = kCLLocationAccuracyKilometer;
        locationManager.distanceFilter = 500;
        [locationManager startUpdatingLocation];
      }
}
```



Using the geolocation services in Windows is fairly straightforward. The example below is in Javascript, but can be ported to C++ or C# easily:

```
geolocator = new Windows.Devices.Geolocation.Geolocator();

geolocator.addEventListener("positionchanged", onPositionChanged);
geolocator.addEventListener("statuschanged", onStatusChanged);

function onPositionChanged(e) {
  var coord = e.position.coordinate;
  // coord.latitude,
  // coord.longitude
  // coord.accuracy;
}

Function onStatusChanged(e) {

switch (e.status) {
  case Windows.Devices.Geolocation.PositionStatus.ready:        /* … */
  case Windows.Devices.Geolocation.PositionStatus.initializing: /* … */
  case Windows.Devices.Geolocation.PositionStatus.noData:       /* … */
  case Windows.Devices.Geolocation.PositionStatus.disabled:     /* … */

  }
}
```



Once we have a user's location, the next common task is to display it graphically, on a map. Each mobile OS is tied to a particular mapping service – and with Android, it's only natural that it be Google Maps.

Google constantly refines and updates its mapping APIs, and the current version (at the time of writing, that is) is APIv2.

NOTE: To use maps, you will need to download the Google Play Services SDK, and to acquire a Google Maps API key. You will also need to get a developer certificate from Google, so your app is recognized and served. You then have to add the key to your application's manifest, that is add a meta-data element:

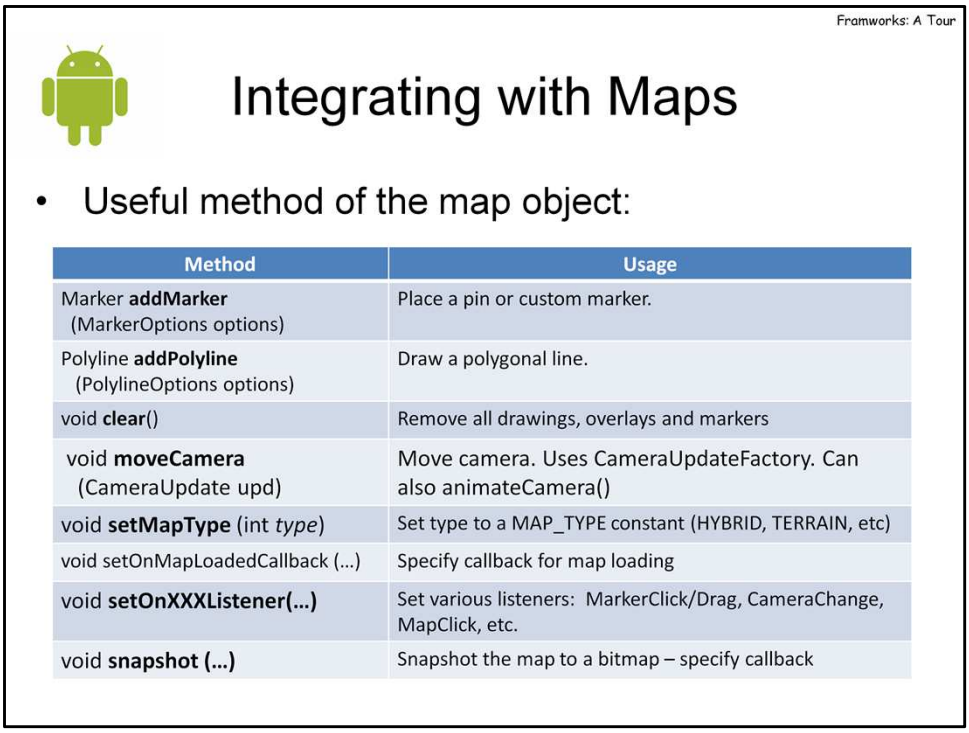
```
<application>
 ...
      <meta-data  android:name="com.google.android.maps.v2.API_KEY"
          android:value="API_KEY"/>
</application>
```

The basic usage of a map is straightforward, and shown in this example (Which is a variant of the "Hello Map" shown in the API documentation:

```
public class MapPane extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState)
   {
         super.onCreate(savedInstanceState);
         setContentView(R.layout.map_activity);
         // Get a handle to the Map Fragment. This assumes you
         // specified the map fragment in your resources, of course
         GoogleMap map = ((MapFragment) getFragmentManager()
                        .findFragmentById(R.id.map)).getMap();
         // Set type: can do NORMAL, TERRAIN, HYBRID, SATELLITE or NONE
         map.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
         LatLng harvard = new LatLng(42.3744     //42.3744°N,
                                     -71.1169);  //71.1169°W
         map.setMyLocationEnabled(true);
         map.moveCamera(CameraUpdateFactory.newLatLngZoom(harvard, 12));
         map.addMarker(new MarkerOptions().title("Harvard").
   .icon(BitmapDescriptorFactory.fromResource(R.drawable.harvard_crest))
                           .snippet("Where this course is delivered")
                           .position(harvard));
 }  // end onCreate
} // end Activity
```

This assumes the XML layout has been specified like this:

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@+id/map"
          android:name="com.google.android.gms.maps.MapFragment"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
/>
```



A full reference for the map object can be found at the Google Map API for Android site - https://developers.google.com/maps/documentation/android/reference/com/google/android/gms/maps/GoogleMap, but the above makes for a quick reference of the methods you're likely to use.

Integrating with Maps

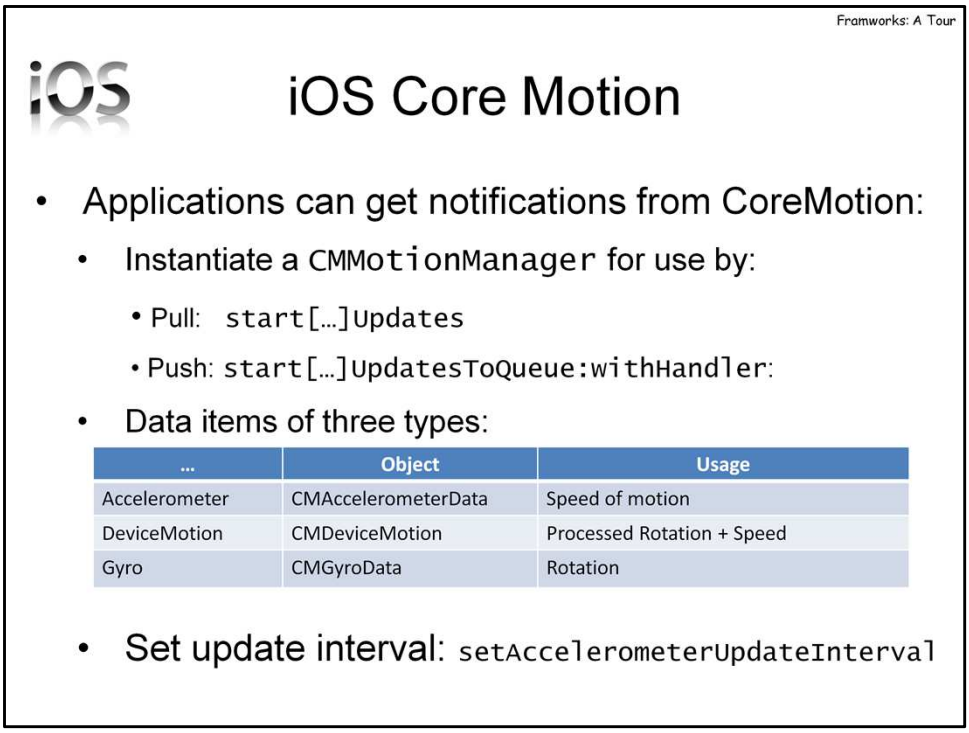
- iOS (and OSX) use the MapKit.framework
- API specified in Location/Maps guide:
  - https://developer.apple.com/library/ios/documentation/userexperience/conceptual/LocationAwarenessPG

Class	Usage
MKMapView	A Map view control
MKMapItem	Launch Maps app programmatically
MKAnnotationView	The familiar "pin" marker, with text
MKPolyline, MKCircle, MKShape	Overlays on map (with ..View objects)
MKCoordinateRegion (struct)	CLLocationCoordinate2D center; MKCoordinateSpan span : longitude and latitudeDelta

Integrating with Maps

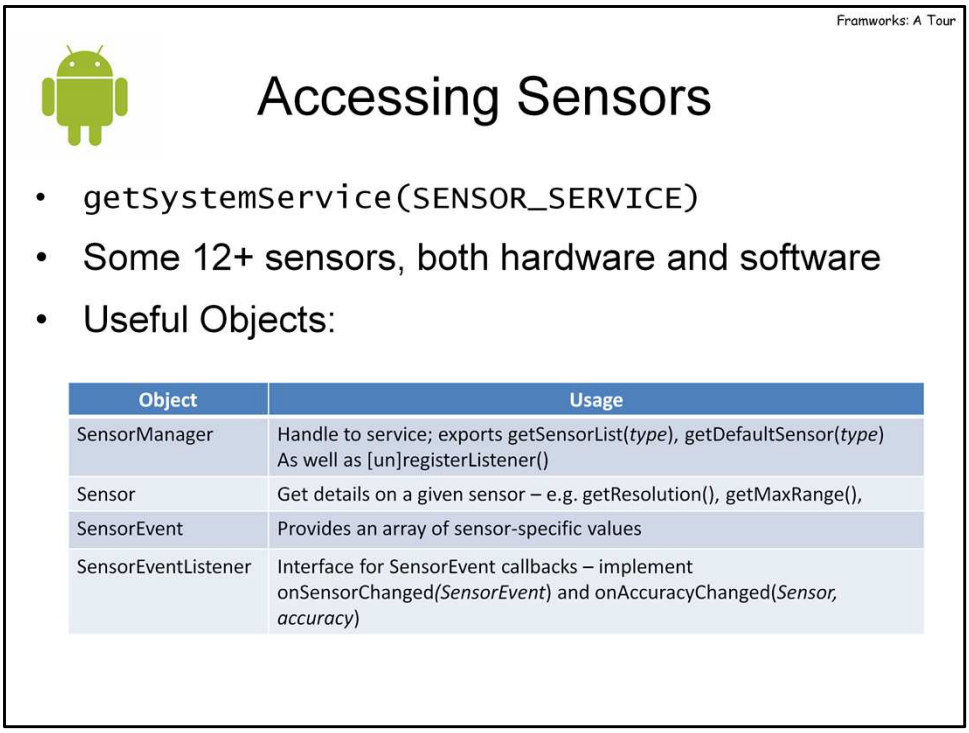
- Windows still pushes BING Maps
  - http://msdn.microsoft.com/en-us/library/dd877180.aspx
- You'll need a BING Maps API key

Class	Usage
Map	The basic map object
Pushpin	The familiar "pin" marker, with text
MapLayer	Overlays on map
MapPolygon, MapPolyline	Overlaid objects on map
TrafficManager	Displays traffic information



"CoreMotion", one of the aptly titled "Core Frameworks" of iOS, can be used to convey motion notifications, in an alternative manner to using UIEvents.

A good reference:
http://developer.apple.com/library/ios/#documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/motion_event_basics/motion_event_basics.html#//apple_ref/doc/uid/TP40009541-CH6-SW14



Android is suited not just for mobile devices, but also for embedded ones. The OS supports a wide array of sensors, which are especially useful when Android is used in climate control devices, or other embedded platform, like Arduino based ones. Not all devices support all sensors, and some sensors (for example, GRAVITY and RORATION_VECTOR) may be implemented in software (i.e. by resampling another sensor, such as (in this case) ACCELEROMETER). Sensors are either binary (e.g. proximity) or continuous

```
public static final int TYPE_ACCELEROMETER = 1;
public static final int TYPE_MAGNETIC_FIELD = 2;
public static final int TYPE_ORIENTATION = 3;
public static final int TYPE_GYROSCOPE = 4;
public static final int TYPE_LIGHT = 5;
public static final int TYPE_PRESSURE = 6;
// TYPE_TEMPERATURE is deprecated in favor of AMBIENT
public static final int TYPE_PROXIMITY = 8;
public static final int TYPE_GRAVITY = 9;
public static final int TYPE_LINEAR_ACCELERATION = 10;
public static final int TYPE_ROTATION_VECTOR = 11;
public static final int TYPE_RELATIVE_HUMIDITY = 12;
public static final int TYPE_AMBIENT_TEMPERATURE = 13;
public static final int TYPE_ALL = -1;
```

http://developer.android.com/guide/topics/sensors/sensors_overview.html contains detailed information about sesnors and how to use them. The following provides a quick cheat sheet:

I) In AndroidManifest.xml:

declare the need for the sensor, by specifying

```
<uses-feature android:name="android.hardware.sensor.type"
              android:required="true/false" />
```

Specifying one of the type constants, and optionally setting `android:required` to `true` if you want your application to only be installable if the sensor is present.

II) **In Activity or service:**

- Declare the component as `implements SensorEventListener`, specifically:
  - `public final void onAccuracyChanged(Sensor s, int accuracy);`
  - `public final void onSensorChanged(SensorEvent event);`
- When implementing `onSensorChanged()` remember not to block.
- Declare private fields (names are, of course, only suggestions):
  - `private SensorManager mSensorMgr;`
  - `private Sensor mSesnsor;`
- In `OnCreate`, initialize those fields:
  - `mSensorMgr = getSystemService(Context.SENSOR_SERVICE);`
  - Either set `mSensor = getDefaultSensor(SENSOR.TYPE_type)` or iterate through `getSensorList(SENSOR.TYPE_type);`

You can query the sensor properties from the Sensor object, using the `getResolution(), getMaximumRange()` and similar methods.

- In `OnResume` (for activities)

    - `mSensorMgr.registerListener(this, mSensor, SensorManager.SENSOR_DELAY_[FASTEST|GAME|UI|NORMAL]);`

    Setting `SENSOR_DELAY` will control the flux of messages, and impact battery life.

- In `OnPause` (for activities)

    - `mSensorMgr.unRegisterListener (this);`

    It's important to unregister listeners in order to conserve battery life, since paused activities can't handle any callback processing anyway.

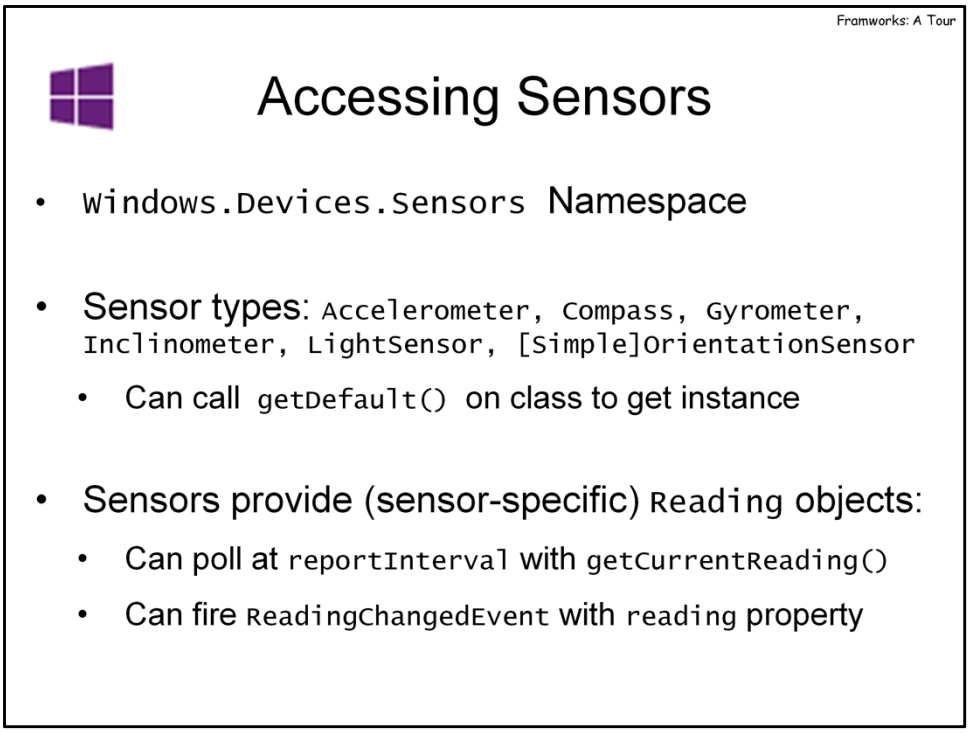
- In `onSensorChanged (SensorEvent e)`:

    - `e.accuracy` returns an accuracy constant: `SENSOR_STATUS_ACCURACY_[HIGH|MEDIUM|LOW|UNRELIABLE].` Binary sensors (or continuous sensors in need of calibration) return UNRELIABLE.

    - `e.sensor` gives you an instance of the sensor which generated the event. This is needed if you register for more than one sensor notification in the same activity or service.

    - `e.timestamp` is specified in milliseconds, and tells you when the event happened

    - `e.values` is an array of e.value.length values – depending on the sensor type. Accelerometer, for example, using three values (the x, y and z axis).



Windows 8 is closer to Android than to iOS in its implementation of sensors. Sensors are grouped in the `Windows.Devices.Sensors` namespace. Windows 8 exports 7 sensors as classes, and a call to `getDefault()` will obtain an instance. That is, to get an instance of an accelerometer, you would use:

```
accelerometer = Windows.Devices.Sensors.Accelerometer.getDefault();
   if (accelerometer != null) { /* Habemus accelerometer */ }
```

Sensors can be used in either polling mode, or asynchronous notification mode. In polling mode, the app needs to specify the `reportInterval` property, taking care not to be smaller than the sensor's (read-only) `minimumReportInterval`. The application then needs to set an interval polling function, (for example, by a dedicated thread or Javascript `setInterval`) and call `getCurrentReading()`, to return a reading object. The reading itself is sensor dependent – continuing the example of the accelerometer, it would provide the `acceleration[X|Y|Z]` properties.

Alternatively, an application may opt for asynchronous notifications, by installing a readingChanged handler. For the accelerometer, an "shaken" handler can be installed, though the shake event delivers no meaningful information past the act of shaking itself.

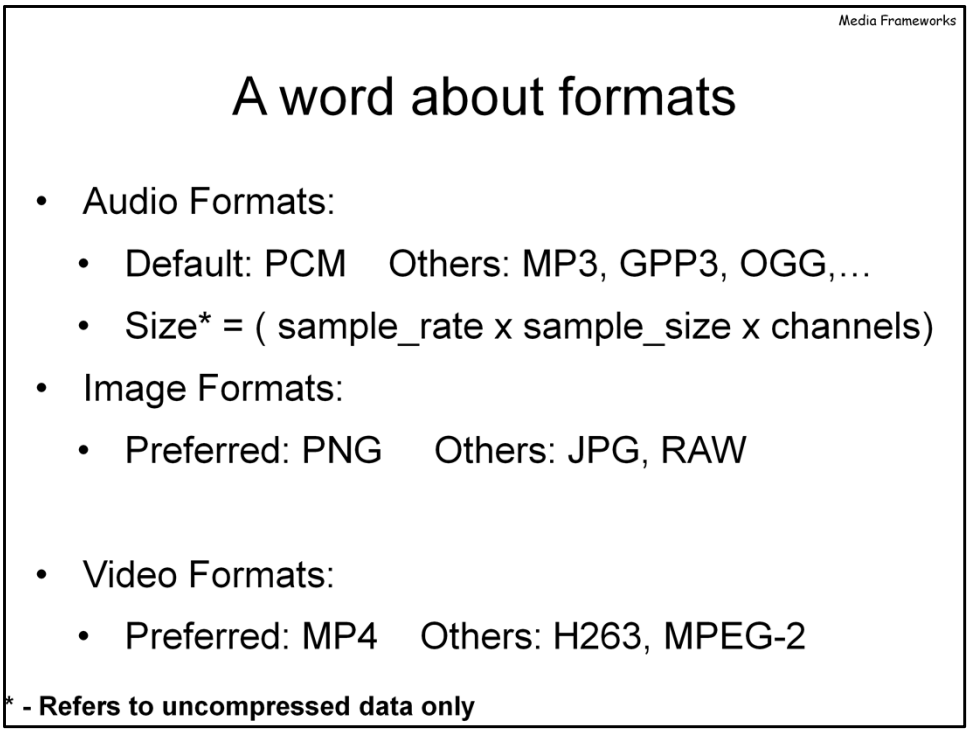
Mobile Frameworks and APIs

Part II: Frameworks – A tour

CSCI E-65: Lecture XI – 11/22/13

Mobile Frameworks: A Tour

Media Playback & Recording



Tips:

- Channels: Remember mono = 1 Stereo = 2
- Sample size: 16 bit is default per channel
- Sample rate: 16khz: FM quality 44.1: CD quality

http://developer.android.com/guide/appendix/media-formats.html covers supported formats in Android.

Media Frameworks

Android AudioFocus

- `AudioFocus` transfers ownership of audio
  - Specify focus change listener
  - Declare audio type (STREAM_MUSIC)
  - Request Focus type
- Implement focus change listener
  - Start audio on focus gain, pause on loss
- Focus gain may allow “ducking” (lower volume)

Sample code:

```
AudioManager am = mContext.getSystemService(Context.AUDIO_SERVICE);
...

// Focus types: GAIN_TRANSIENT (temporary request)
//              GAIN_TRANSIENT_EXCLUSIVE (no other sounds)
//              GAIN_TRANSIENT_MAY_DUCK (previous owner can “duck”
//              GAIN (permanent)
int result = am.requestAudioFocus(afChangeListener,
                                  AudioManager.STREAM_MUSIC,
                                  AudioManager.AUDIOFOCUS_GAIN);

if (result == AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
    am.unregisterMediaButtonEventReceiver(RemoteControlReceiver);
    // Start playback.
}

// When done
Am.abandonAudioFocus(afChangeListener)
```

```
OnAudioFocusChangeListener afChangeListener = new
OnAudioFocusChangeListener() {
    public void onAudioFocusChange(int focusChange) {
        switch (focusChange)
        {
            case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT:
            // Pause playback

            case AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK:
            // keep on playing, but lower volume
            case AudioManager.AUDIOFOCUS_GAIN:
            // We have focus - Resume playback
            case AudioManager.AUDIOFOCUS_LOSS:
            // abandon Audio Focus
         } // switch
    }  // onAudioFocusChange
}; // OnAudioFocusChangeListener
```

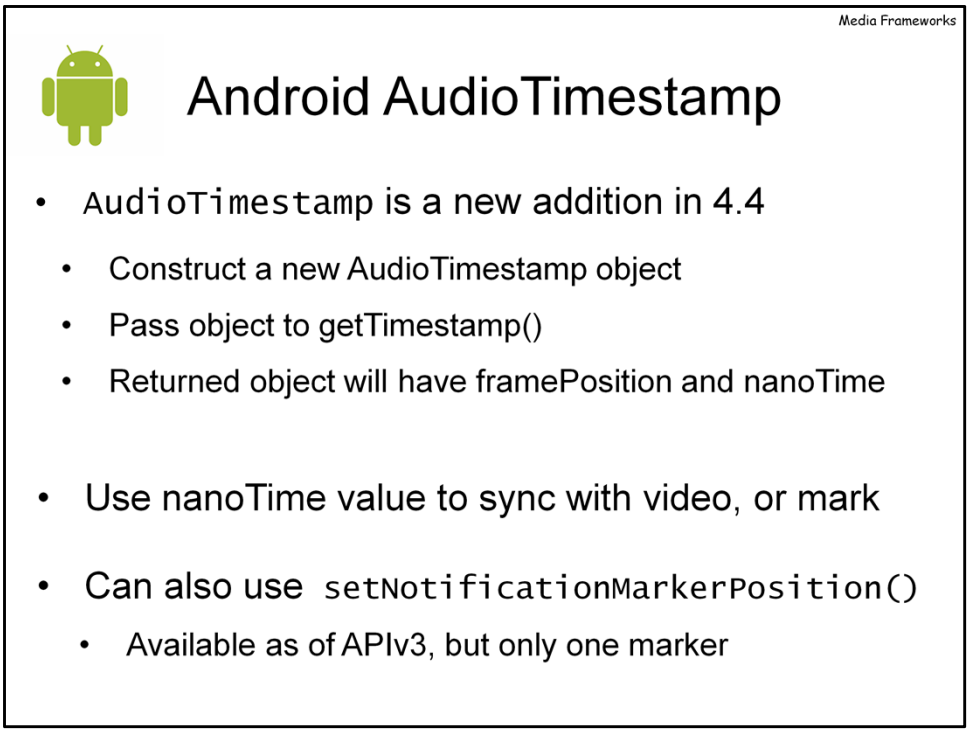
Media Frameworks

Android AudioTrack

- `AudioTrack` allows you to play raw PCM data
  - `MODE_STREAM`: Continuous streaming mode
  - `MODE_STATIC`: Data written once
- `Write()` data samples to track object and `play()`

Sample code:

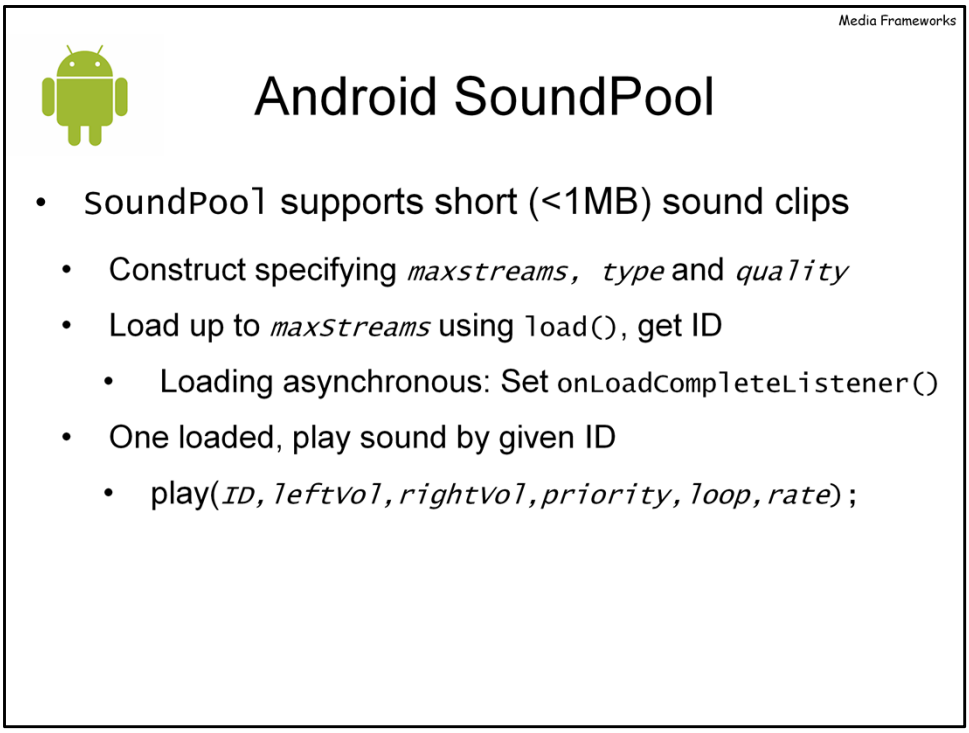
```
byte[] audioData = // get data from some FileInputStream
int size = android.media.AudioTrack.getMinBufferSize(8000,
         AudioFormat.CHANNEL_CONFIGURATION_MONO,
         AudioFormat.ENCODING_PCM_8BIT);

AudioTrack at = new AudioTrack(AudioManager.STREAM_MUSIC,
            8000,
            AudioFormat.CHANNEL_CONFIGURATION_MONO,
            AudioFormat.ENCODING_PCM_8BIT,
            size,
            AudioTrack.MODE_STATIC);
if (at!=null) {
// Write data to track
 at.write(audioData, 0, audioData.length);
 at.play();
 at.release();
}
```



AudioTimestamp is a new addition in Android 4.4 (KitKat), though it has been present for a while in iOS. The idea is to allow application creators to better sync audio and video, or to provide support for “markers” in audio files.

The usage is fairly simply – you create a new AudioTimestamp object, and then call getTimestamp with it, to populate the object with data from a playing audiotrack. The object is a simple structure containing two fields – the index of the playing audio frame (framePosition) and the time in nanoseconds (nanoTime).

Note that, while this is fairly accurate, Android makes no guarantees as to accuracy, as there may be latency between submitting the PCM data to the driver and the actual playback of the data.

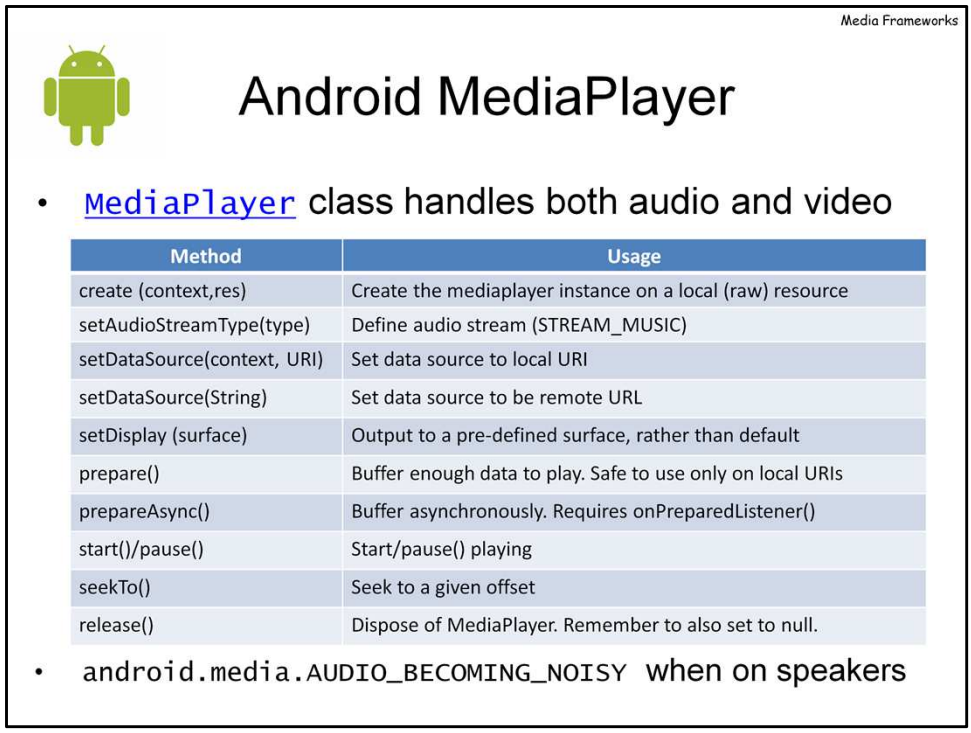


The `android.media.SoundPool` class supports the playing of short sound files by preloading them into the "pool" , and then playing them by ID. When constructing the pool, the user should specify three parameters – how many streams are to be loaded, their type (usually `AudioManager.STREAM_MUSIC`), and their quality (usually left as 0). Following that, the streams are loaded, and can be played by ID. This is shown in the following simple example:

```
private sp = new SoundPool(2,         // maxStreams
                           type,      // AudioManager.STREAM_MUSIC,
                           0);        // srcQuality
soundPool.setOnLoadCompleteListener(new OnLoadCompleteListener() {
@Override
 public void onLoadComplete(SoundPool sp, int sample, int status)
  {        this.loaded = true;      }
  }
  );

soundID = soundPool.load(this, R.raw.sound1, 1);

// To play sound
if (this.loaded) {
      sp.play(soundID, volume, volume, 1, 0, 1f);
  }
```

Android VideoView

- `VideoView` provides a simple way to show videos
  - Get view from app's layout (or create programmatically)
  - Call setVideoPath() to load video
  - Call start() – and you're done
  - Use a MediaController if you want controls
- Tutorial: here



Method	Usage
create (context,res)	Create the mediaplayer instance on a local (raw) resource
setAudioStreamType(type)	Define audio stream (STREAM_MUSIC)
setDataSource(context, URI)	Set data source to local URI
setDataSource(String)	Set data source to be remote URL
setDisplay (surface)	Output to a pre-defined surface, rather than default
prepare()	Buffer enough data to play. Safe to use only on local URIs
prepareAsync()	Buffer asynchronously. Requires onPreparedListener()
start()/pause()	Start/pause() playing
seekTo()	Seek to a given offset
release()	Dispose of MediaPlayer. Remember to also set to null.

The `VideoView` is a wrapper over another class - `MediaPlayer` – which can handle both audio and video. It is also fairly simple to use – Either construct it (or use the `create()` factory method), set the `DataSource`, call `prepare()` for external resources, and `start()`.

```
private SurfaceView surfaceView;
private SurfaceHolder surfaceHolder;
@Override public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    getWindow().setFormat(PixelFormat.UNKNOWN);
    surfaceView = (SurfaceView) findViewById(R.id.surfaceview);
    surfaceHolder = surfaceView.getHolder();
    surfaceHolder.addCallback(this);
    surfaceHolder.setFixedSize(x, y);
    SurfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    mediaPlayer = new MediaPlayer();

    if (mediaPlayer.isPlaying()){        mediaPlayer.reset();    }
    mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
    mediaPlayer.setDisplay(surfaceHolder);
    try   {
           mediaPlayer.setDataSource("/path/to/media");
           mediaPlayer.prepare();
           } catch (Exception e)}
    mediaPlayer.start();
 }
```

Note:

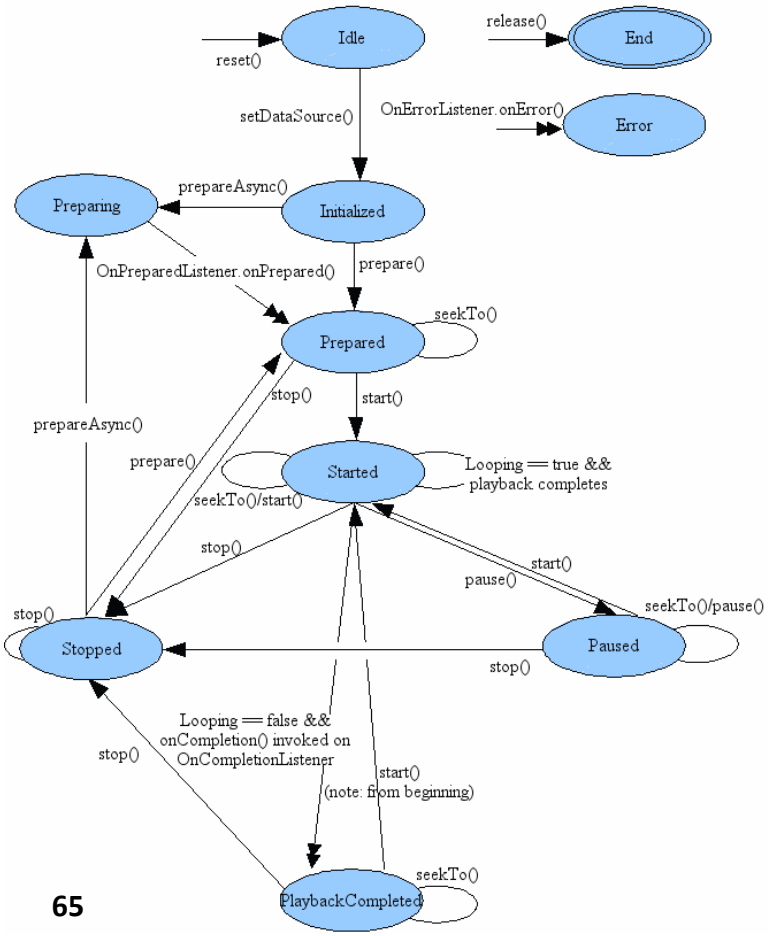
- Media preparation can take a significant amount of time, and risks blocking your UI thread, especially for off-device (read: Internet) resources. Calling `prepare()` on these resources is a bad idea, as the application must remain responsive. For this, `MediaPlayer` provides `prepareAsync()`, but this requires you to call `setOnPreparedListener()`, and provide a listener which implements `onPrepared()`.

- `MediaPlayer` instances you create in your activity are constrained by your activity lifecycle. This means that if your activity loses visibility, media playback will be interrupted. Likewise, if the device orientation changes (which forces a call to `onStop()`). It's important to release and re-create the `MediaPlayer` instances.

- Most applications opt to perform the media playback through a service. This makes more sense, because it enables the media playback (specifically, audio) to continue in the background, even if the application is not visible. The service can and should run as a foreground service, and should implement a wake lock while it is active, to avoid the device shutting off the screen and/or wi-fi to conserve power while the stream is active. If using a service, remember to release the `MediaPlayer` instance in the `onDestroy()`.

Also, see http://developer.android.com/guide/topics/media/mediaplayer.html for more tips.

The reference page on the MediaPlayer class has a comprehensive state diagram which shows the media player class state transitions:



Media Frameworks

iOS MPMoviePlayerController

- Same basic idea as Android Media Player
- Allocate an instance, initWithContentURL
- Provides view to play movies part/full screen
- Can notify on movie finish

```
(void) moviePlayBackDidFinish:(NSNotification*)
```

Sample code:

```
NSURL *url = [NSURL fileURLWithPath:[[NSBundle mainBundle]
             pathForResource:introVideoFileName ofType:@""]];
MPMoviePlayerController *moviePlayer =
     [[MPMoviePlayerController alloc] initWithContentURL:url];

// Can set notification on MPMoviePlayerPlaybackDidFinishNotification
[[NSNotificationCenter defaultCenter] addObserver:self
           selector:@selector(moviePlayBackDidFinish:)
           name:MPMoviePlayerPlaybackDidFinishNotification
           object:moviePlayer];    // can call [moviePlayer release]

moviePlayer.controlStyle = MPMovieControlStyleNone;
moviePlayer.shouldAutoplay = YES;
[self.view addSubview:moviePlayer.view];
[moviePlayer setFullscreen:YES animated:YES];
```

Media Frameworks

iOS

AVFoundation

- iOS 4+ provides the AVFoundation framework
  - AVAudioPlayer[Delegate]
  - AVAudioRecorder[Delegate]
- When instantiating, specify delegate

```
-(void)audioPlayerDidFinishPlaying:successfully:(BOOL)
-(void)audioPlayerDecodeErrorDidOccur:error:(NSError *)
-(void)audioPlayerBeginInterruption:(AVAudioPlayer *)
-(void)audioPlayerEndInterruption:(AVAudioPlayer *)
```

```
- (void)viewDidLoad {
    [super viewDidLoad];
    NSURL *url = [NSURL fileURLWithPath:[[NSBundle mainBundle]
            pathForResource:@“URL PATH HERE"
            ofType:@"mp3"] // Or whatever codec
           ];
    NSError *error;
    audioPlayer = [[AVAudioPlayer alloc]
                   initWithContentsOfURL:url  error:&error];
        if (error)
        {
                NSLog(@"Error in audioPlayer: %@",
                        [error localizedDescription]);
        } else {
                audioPlayer.delegate = self;
                [audioPlayer prepareToPlay];
        }
}
-(void)playAudio
{    [audioPlayer play]; }
-(void)stopAudio
{    [audioPlayer stop]; }
-(void)adjustVolume
{
    if (audioPlayer != nil)
    {
         audioPlayer.volume = // Adjust volume value here
    }
}
```

Media Frameworks

Screen Capture

- `ImageReader` is a new addition in 4.4
  - Create `ImageReader` with `ImageReader.getInstance()`
  - Implement an onImageAvailableListener()
  - Call `ImageReader.getSurface()` for a surface view
  - Pass surface to `MediaPlayer/MediaCodec`, use normally
  - `onImageAvailable()` will be called back
    - Use `acquire[Latest/Next]Image()` to capture Image

Nested Classes		
interface	ImageReader.OnImageAvailableListener	Callback interface for being notified that a new image is available.

Public Methods	
Image	acquireLatestImage() Acquire the latest `Image` from the ImageReader's queue, dropping older `images`.
Image	acquireNextImage() Acquire the next Image from the ImageReader's queue.
void	close() Free up all the resources associated with this ImageReader.
int	getHeight() The height of each `Image`, in pixels.
int	getImageFormat() The `image format` of each Image.
int	getMaxImages() Maximum number of images that can be acquired from the ImageReader by any time (for example, with `acquireNextImage()`).
Surface	getSurface() Get a `Surface` that can be used to produce `Images` for this `ImageReader`.
int	getWidth() The width of each `Image`, in pixels.
static ImageReader	newInstance(int width, int height, int format, int maxImages) Create a new reader for images of the desired size and format.
void	setOnImageAvailableListener(ImageReader.OnImageAvailableListener listener, Handler handler) Register a listener to be invoked when a new image becomes available from the ImageReader.

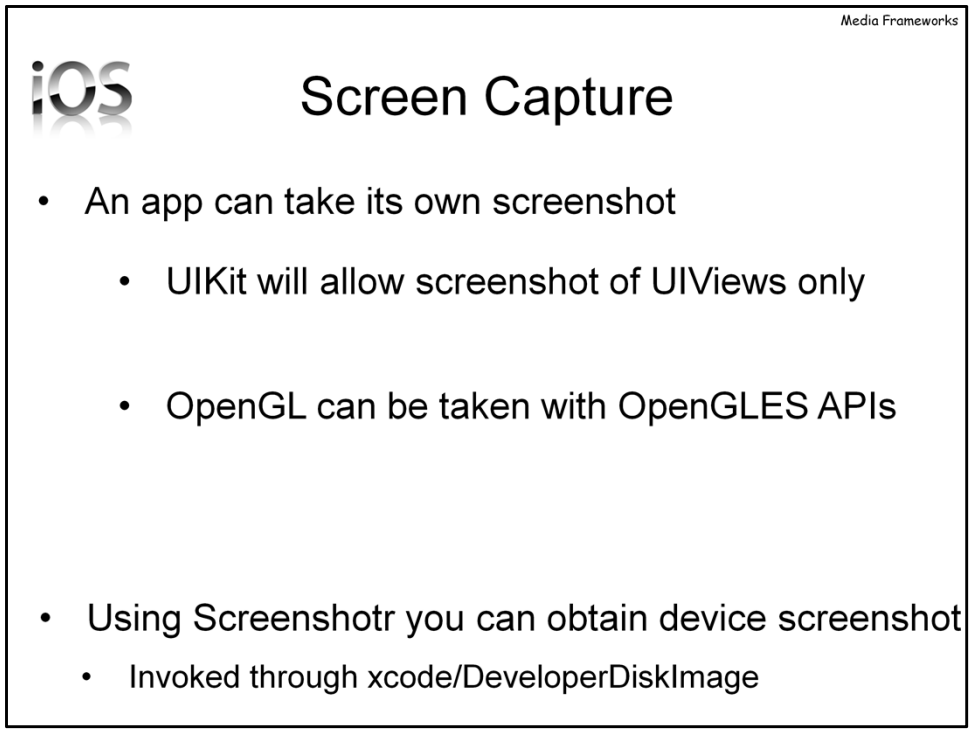
Screen Capture

- From the command line (adb shell)
  - `screencap:` Get capture from surfaceflinger
  - `screenshot:` Get shot directly from fb
    - Optional sound clip on success
  - `screenrecord:` record .mp4 of screen
- On Device (4.1 and later): Power + Volume down
  - Can be prevented with code:

```
getWindow().setFlags(WindowManager.LayoutParams.FLAG_SECURE,
WindowManager.LayoutParams.FLAG_SECURE);
```

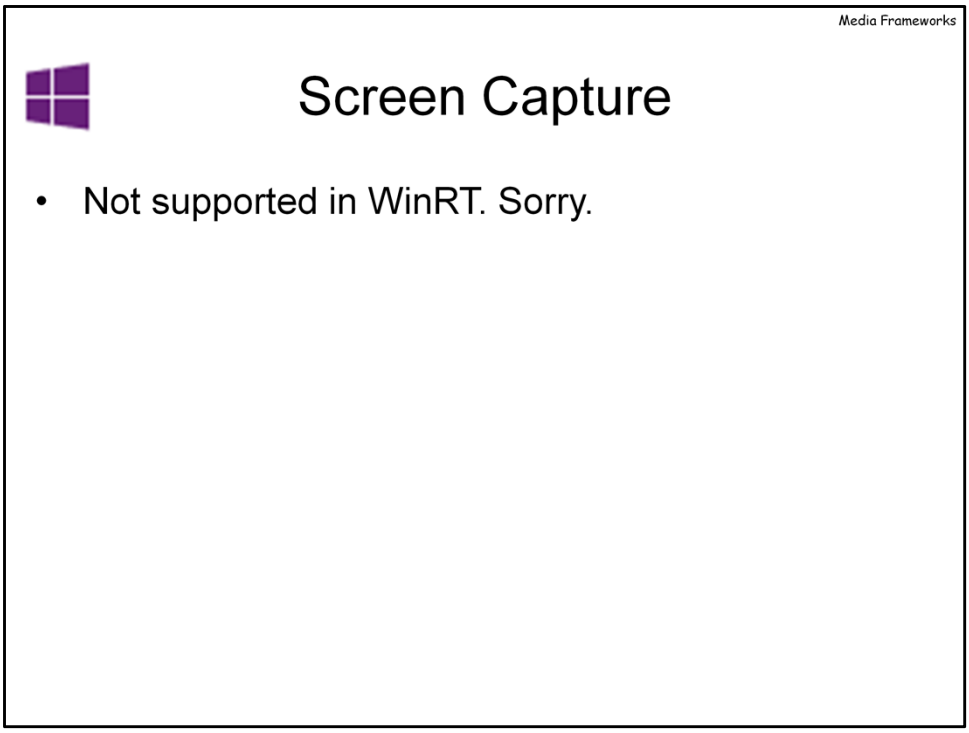
```
root@generic:/system/bin # screencap -h
usage: screencap [-hp] [-d display-id] [FILENAME]
   -h: this message
   -p: save the file as a png.
   -d: specify the display id to capture, default 0.
If FILENAME ends with .png it will be saved as a png.
If FILENAME is not given, the results will be printed to stdout.
```

```
root@generic:/system/bin # screenshot
usage: screenshot [-s soundfile] filename.png
   -s: play a sound effect to signal success
   -i: autoincrement to avoid overwriting filename.png

root@generic:/system/bin # screenrecord  --help
Usage: screenrecord [options] <filename>

Records the device's display to a .mp4 file.

Options:
--size WIDTHxHEIGHT
    Set the video size, e.g. "1280x720".  Default is the device's main
    display resolution (if supported), 1280x720 if not.  For best results,
    use a size supported by the AVC encoder.
--bit-rate RATE
    Set the video bit rate, in megabits per second.  Default 4Mbps.
--time-limit TIME
    Set the maximum recording time, in seconds.  Default / maximum is 180.
--rotate
    Rotate the output 90 degrees.
--verbose
```

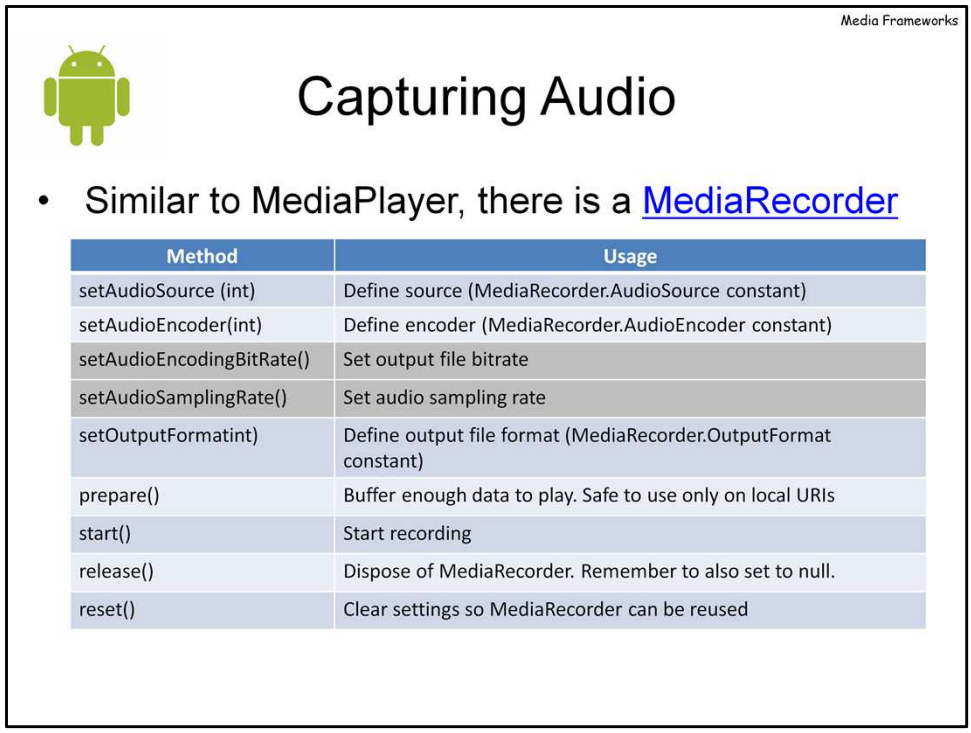


Sample code to generate a screenshot:

```
- (void) getScreenShot:(NSString *Output)
{
  CGFloat scale = 1.0f;
  if ([[UIScreen mainScreen] respondsToSelector:@selector(scale)])
      scale = [UIScreen mainScreen].scale;

  UIGraphicsBeginImageContextWithOptions
        (self.window.bounds.size, NO, scale);

  UIGraphicsBeginImageContext(self.window.bounds.size);
  [self.window.layerrenderInContext:UIGraphicsGetCurrentContext()];
  UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
  UIGraphicsEndImageContext();

  NSData * data = UIImagePNGRepresentation(image);
  // PNG is generally better, but if you want JPG:
  // NSData * data = UIImageJPEGRepresentation(image, quality);
  [data writeToFile:Output atomically:YES];
}
```

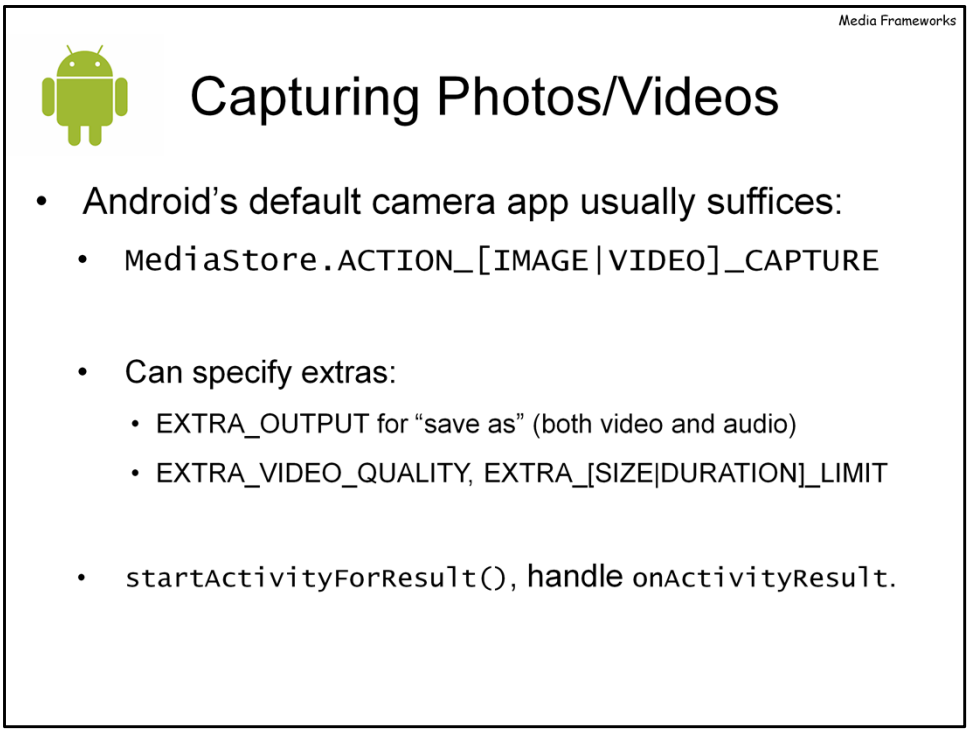


http://social.msdn.microsoft.com/Forums/windowsapps/en-US/63dd9596-bf94-440b-847a-961cbf036e7b/how-to-capture-screen-in-metro-app?forum=winappswithcsharp



Usage example: Note the order, as well as all the method calls, are mandatory.

```
MediaRecorder recorder = new MediaRecorder();
// Select source (required) DEFAULT, MIC, VOICE_*, CAMCORDER, etc
recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
// Select recording format (required) AMR_NB/WB, AAC_ADTS, THREE_GPP.
recorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);
// Select encoder (required) AAC, AAC_ELD, HE_AAC, AMR_NB/WB,..
recorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
// Select output file (required)
recorder.setOutputFile(PATH_NAME);

recorder.prepare();
// and record..
recorder.start();
...
recorder.stop();
```



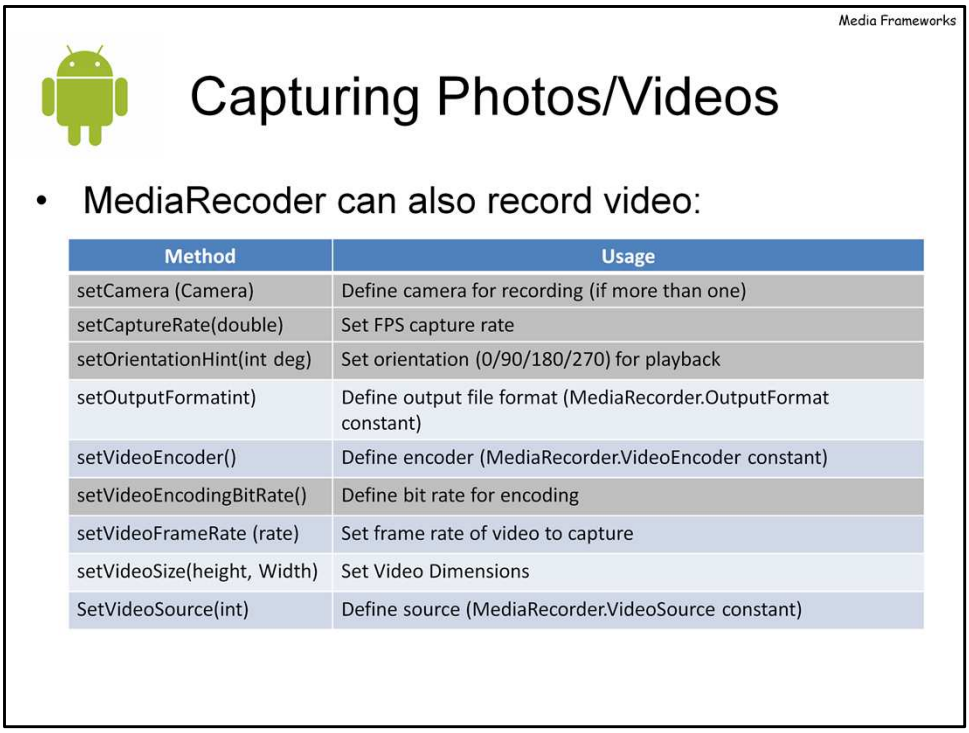
For most applications, capturing a photo on video is better off delegated to the default camera application. Android's camera app (as well as user-installable camera apps) support the `ACTION_IMAGE_CAPTURE` and `ACTION_VIDEO_CAPTURE` intents, for stills and videos, respectively.

To pass more information with the intent, you can use `MediaStore.EXTRA_OUTPUT`. If set, Android will save the capture data to the URI specified in it. Otherwise, if it is not specified and the data is small enough, it will be passed with the resulting intent. For video, you can also specify `EXTRA_VIDEO_QUALITY`, and `EXTRA_SIZE_LIMIT/EXTRA_DURATION_LIMIT`, to indicate the maximum size of the capture data.

Once you have the intent ready, it's a simple matter of calling `startActivityForResult`, with some request code you define in your activity. Then, as is always the case, you need to implement `onActivityResult()`, and look for the `requestCode` to match the one you have specified.

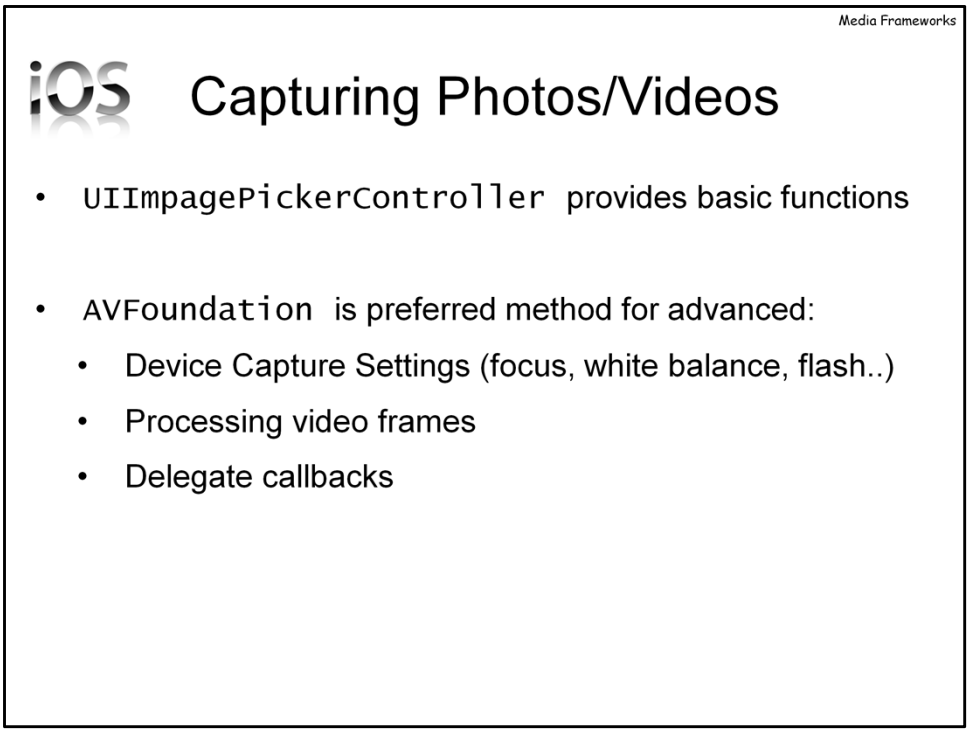
This is shown in the following sample code

```
import android.provider.MediaStore;

private final int MY_ACTIVITY_CODE = 1;
private final String FILENAME = "...";
private File mFile;

private void launchCamera(){

  // "android.media.action.IMAGE_CAPTURE"

  Intent intent = new  Intent(ACTION_IMAGE_CAPTURE);

   mFile = new File(FILENAME);
   Uri outputFileUri = Uri.fromFile(mFile);
   intent.putExtra(MediaStore.EXTRA_OUTPUT, outputFileUri);
   startActivityForResult(intent, PICTURE_ACTIVITY_CODE);

}

protected void onActivityResult(int requestCode,
                                int resultCode,
                                Intent data)
{
   if (requestCode == MY_ACTIVITY_CODE)
       {
           if (resultCode != RESULT_OK) { /* Error */ return; }
           // Otherwise, we have the output file in mFile

       }

} // end onActivityResult
```

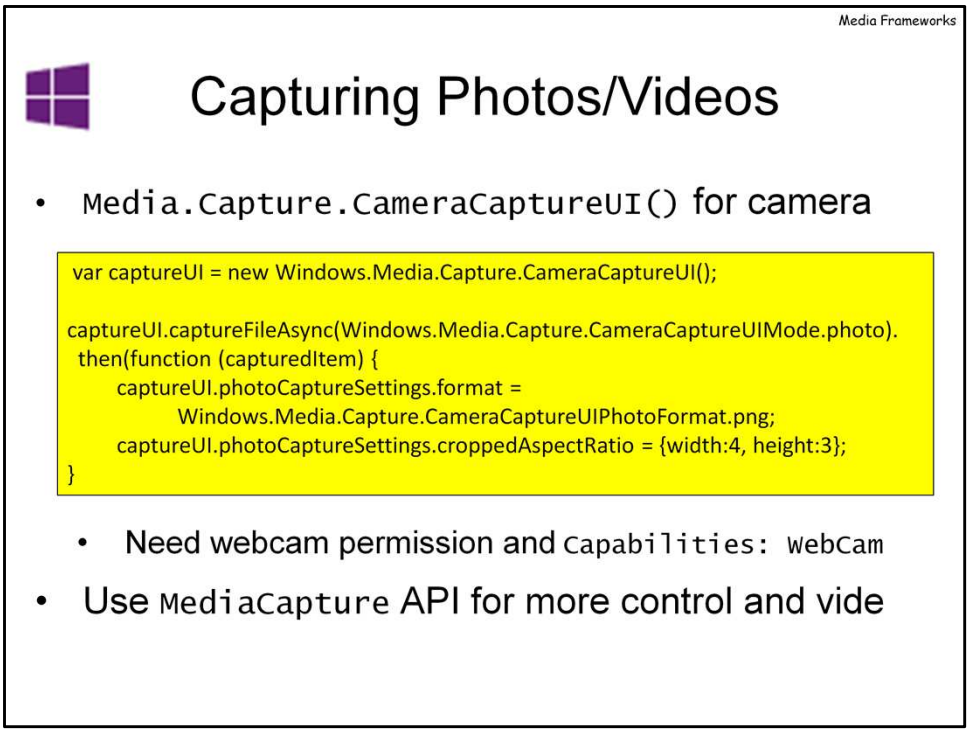


As with MediaPlayer – which supports both audio and video – so does MediaRecorder. Usage for video is just like in the audio case. Once again, order of invocation is pretty rigid.

```
MediaRecorder recorder = new MediaRecorder();
// Select recording format – MPEG_4
recorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);
// Select source   (required) CAMERA or DEFAULT
recorder.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
// Select encoder  (required)  H263, H264, MPEG_4_SP or DEFAULT
recorder.setVideoEncoder(MediaRecorder.VideoEncoder.DEFAULT);
// Optionally Set Size
recorder.setVideoSize(height,width);
// Select output file (required)
recorder.setOutputFile(PATH_NAME);
// Initialize, with all selected parameters..
recorder.prepare();
// and record..
recorder.start();
...
recorder.stop();
```



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



http://code.msdn.microsoft.com/Media-Capture-Sample-adf87622 – Sample capture

http://msdn.microsoft.com/en-us/library/windows/apps/hh465152.aspx - capturing a photo or video using the camera dialog. Requires Webcam and microphone (for video)

```
function initCaptureSettings() {
    captureInitSettings = null;
    captureInitSettings = new
Windows.Media.Capture.MediaCaptureInitializationSettings();
    captureInitSettings.audioDeviceId = "";
    captureInitSettings.videoDeviceId = "";
    captureInitSettings.streamingCaptureMode =
Windows.Media.Capture.StreamingCaptureMode.audioAndVideo;
    captureInitSettings.photoCaptureSource =
Windows.Media.Capture.PhotoCaptureSource.videoPreview;
    captureInitSettings.realTimeModeEnabled = true;
    if (deviceList.length > 0)
        captureInitSettings.videoDeviceId = deviceList[0].id;
}
```

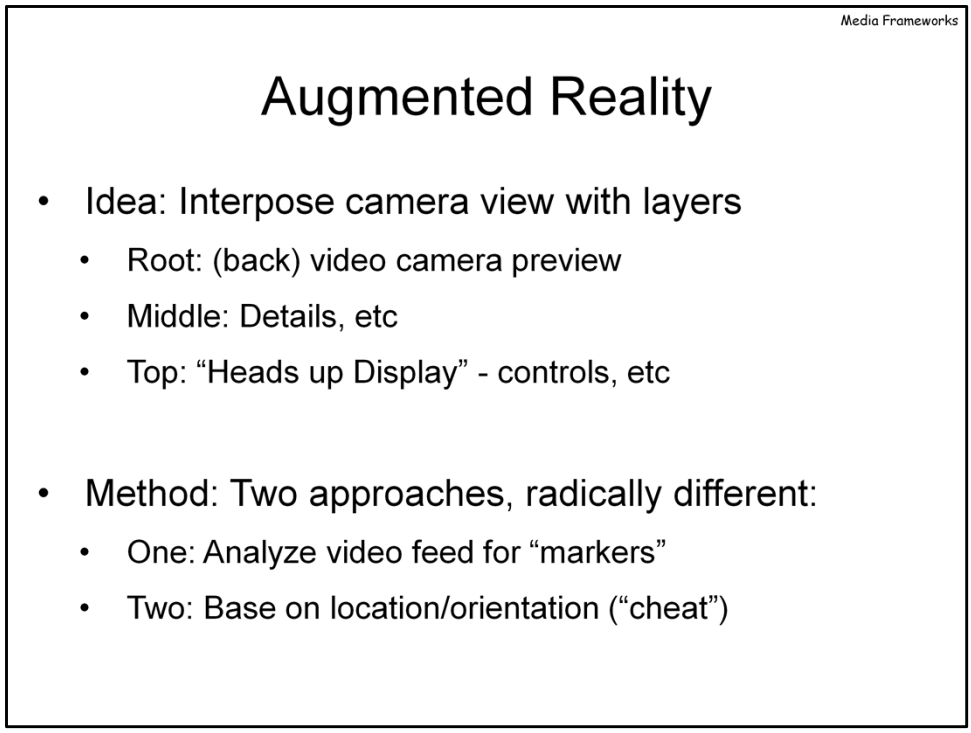
Media Frameworks

iOS

Full Screen Video

- Start with `UIImagePickerController`:
  - Set source:`ip.sourceType = UIImagePickerSourceTypeCamera`
  - Hide Camera Controls: `ip.showCameraControls = NO;`
  - Hide navigation/toolbars
    - `ip.toolbarHidden = ip.navigationBarHidden = NO`
  - Set Full Screen `ip.wantsFullScreenLayout = YES`
- Implement delegate protocol
  - `imagePickerController:didFinishPickingMediaWithInfo`
  - Need to `dismissModalViewControllerAnimated`

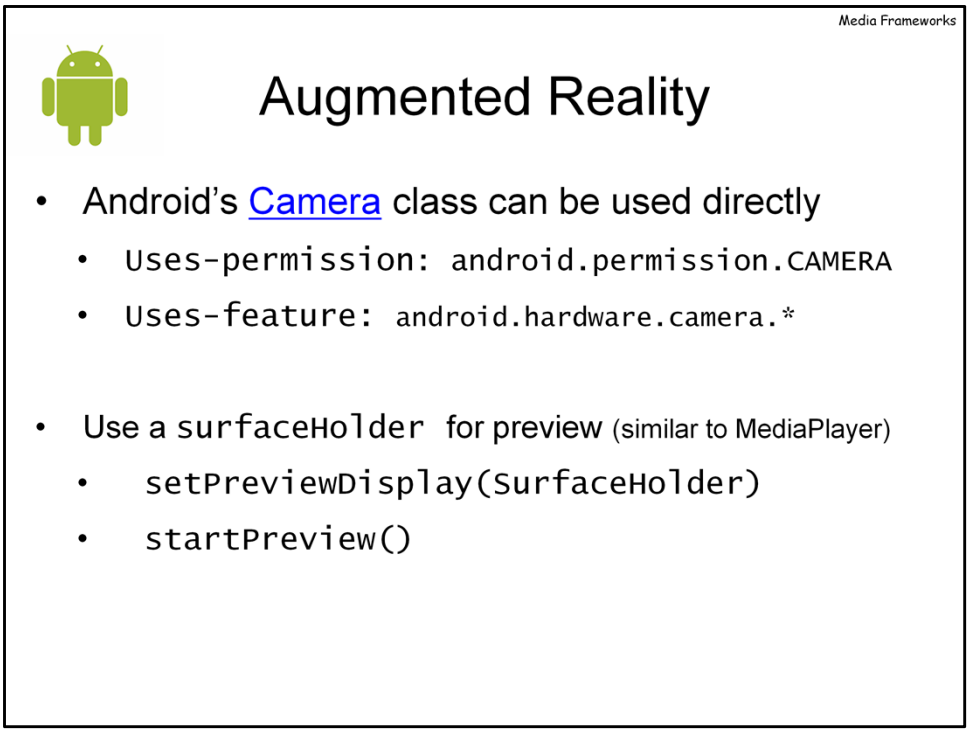
```
UIImagePickerController *ip =
   [[UIImagePickerController alloc] init];
      cameraUI.sourceType = UIImagePickerControllerSourceTypeCamera;

// Filter media type capture to video only
ip.mediaTypes =
  [[NSArray alloc] initWithObjects: (NSString *) kUTTypeMovie, nil];

ip.showCameraControls    =
  ip.toolbarHidden       =
  ip.navigationBarHidden = NO;

ip.wantsFullScreenLayout = YES;

// can also use this:
ip.allowsEditing = NO;

ip.delegate = self; // Must implement delegate interface

// Might need to tweak cameraViewTransform using CGAffineTransformScale
```



Augmented Reality (AR) is an exciting and popular realm of applications which combine video input with real-time details on items in the feed. This can be accomplished by getting the video camera preview directly in the application/activity's view, then overlaying additional layers on it. Usually, one or two more layers are added, containing the "augmentation" (i.e. details, images, etc), and optionally the application's controls. Google has openly declared its interest in AR with its "Google Glass" – Android-based glasses, which will let their wearer see the world from the glasses' camera, along with details overlaid by the system.

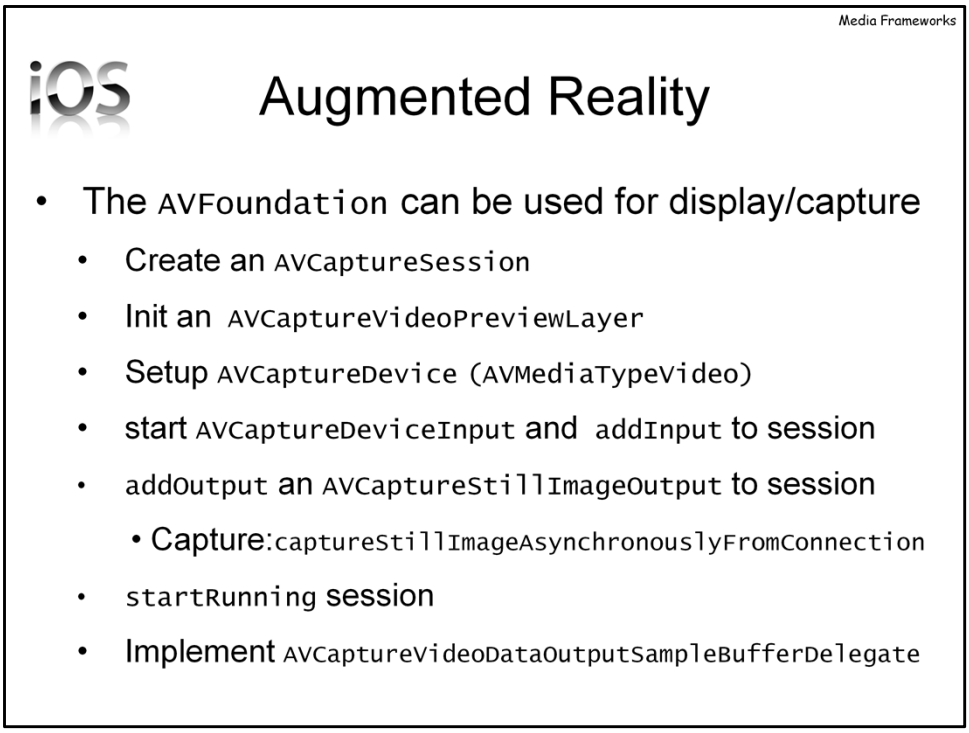
There are two different approaches to treating AR in applications: The first requires processing each video frame, analyzing it for "markers" – patterns which are recognizable by the app: e.g. faces, landmarks, fonts and letters, etc. Once those are detected, the second layer "kicks in" and overlays information on or by the markers, potentially overwriting them with other pixels (effectively "erasing" them from existence). This approach provides "true" AR in that it is sensitive to the images the user would have seen through the camera.

The second approach is "cheating" in the sense that the camera view doesn't even matter. Instead, the device location and orientation (if accurate enough) provide the precise coordinates to determine what the camera would have been presenting to the user. The overlaid information can be stored well ahead of time and displayed without consideration for the images (i.e. with no image processing) – in the hope that nothing substantial has changed in the location since the details were input.



You can use the Android Camera class to capture photos and videos directly from the camera, though in many cases MediaRecorder provides a simpler API. There is an advantage, however, to using the camera directly – and that is for augmented reality. You can create a Surface to display what the camera is viewing, and layer additional views on top of it.

Remember that using the camera directly, as any type of recoding, requires the appropriate permissions, and possibly feature sets (that is, aside from android.hardware.camera you might want autofocus, flash, and other specific features).



An iOS full screen video (as discussed previously) will be suitable for the "cheating" method, as it doesn't offer image processing capabilities. In order to also process the image input stream, the `AVFoundation` can be used. While somewhat more complicated than the `UIImagePicker` method described previously, it offers the main advantage of asynchronous image capture and retention in a buffer (i.e. no filesystem access required) – which makes it perfect for further processing.

Sample code to do so would look thus – assuming your controller is called `MyVideoViewController`, this is the .h file:

```
#import <AVFoundation/AVFoundation.h> // for AV* stuff

@interface MyVideoViewController : UIViewController {

}
@property  (strong, nonatomic) IBOutlet UIView *vpView;
@property (nonatomic, retain) AVCaptureStillImageOutput *siOutput;

@end
```

The implementation would initialize the `AVCaptureSession`, normally in the `viewDidLoad` callback handler:

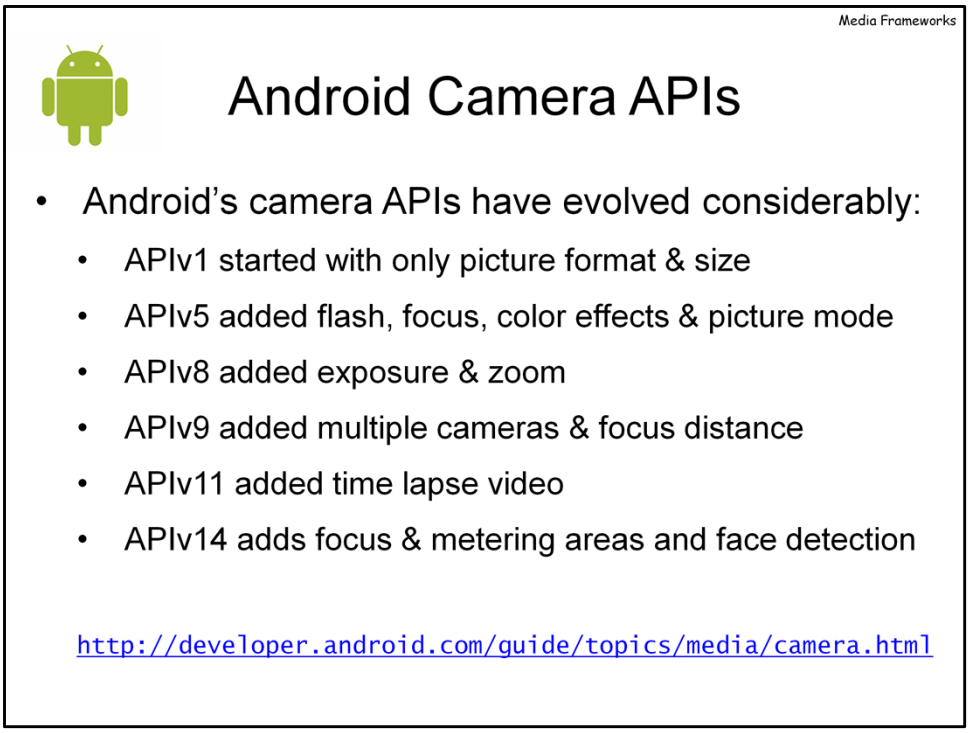
```
AVCaptureSession *avcSession = [[AVCaptureSession alloc] init];

avcSession.sessionPreset = AVCaptureSessionPresetHigh; // Medium, Low..

AVCaptureVideoPreviewLayer *vpLayer =
  [[AVCaptureVideoPreviewLayer alloc] initWithSession:avcSession];

// Maximize preview layer over our view (i.e. match bounds)
vpLayer.frame = self.vpView.bounds;

// Add preview layer as sub-view
[self.vpView.layer addSublayer:vpLayer];

AVCaptureDevice *device =
  [AVCaptureDevice defaultDeviceWithMediaType:AVMediaTypeVideo];

// Set up an AVCaptureDeviceInput. Warning - This may fail.
AVCaptureDeviceInput *input =
   [AVCaptureDeviceInput deviceInputWithDevice:device error:nil];
if (!input) {  /* some error .. Probably want to abort */ }

// Otherwise, assuming input is not nil
[avcSession addInput:input];

// Now add output: Sample stillImageOutput:
siOutput = [[AVCaptureStillImageOutput alloc] init];
NSDictionary *outputSettings = [[NSDictionary alloc]
  initWithObjectsAndKeys: AVVideoCodecJPEG, AVVideoCodecKey, nil];

[siOutput setOutputSettings:outputSettings];
[avcSession addOutput:siOutput];

// Could also add VideoDataOutput: This will require a dispatch queue
// and would require a delegate implementing SampleBufferDelegate
AVCaptureVideoDataOutput *vOutput =
  [[AVCaptureVideoDataOutput alloc] init];
dispatch_queue_t queue = dispatch_queue_create("..", NULL);
[vOutput setSampleBufferDelegate:self queue:queue];
dispatch_release(queue);

// Can set vOutput.minFrameDuration and videoSettings Here.
// [avcSession setSessionPreset:AVCaptureSessionPreset640x480];

[avcSession addOutput:vOutput];

// Start session
[avcSession startRunning];
```

Implementing the `OutputSampleDataBufferDelegate` involves a single method – `didOutputSampleBuffer:`

```
- (void) captureOutput:(AVCaptureOutpu *)captureOutput
   didOutputSampleBuffer:(CMSampleBufferRef) sampleBuffer
   fromConnection:(AVCaptureConnection *) connection
{

   CVImageBufferRef img = CMSampleBufferGetImageBuffer(sampleBuffer);

   if (CVPixelBufferLockBaseAddress(img, 0) == kCVReturnSuccess)
    {
                // Do image processing

       CVPixelBufferUnlockBaseAddress(img, 0);
    } // end if CVPixelBufferLockBaseAddress..
} // end didOutputSampleBuffer
```

MUCH more detail on this can be found in:

a) @jrpowers, “Computer Vision and Augmented Reality on iOS” - VTM Iphone Dev Con 2011
b) “Pro iOS 5 Augmented Reality” - APress

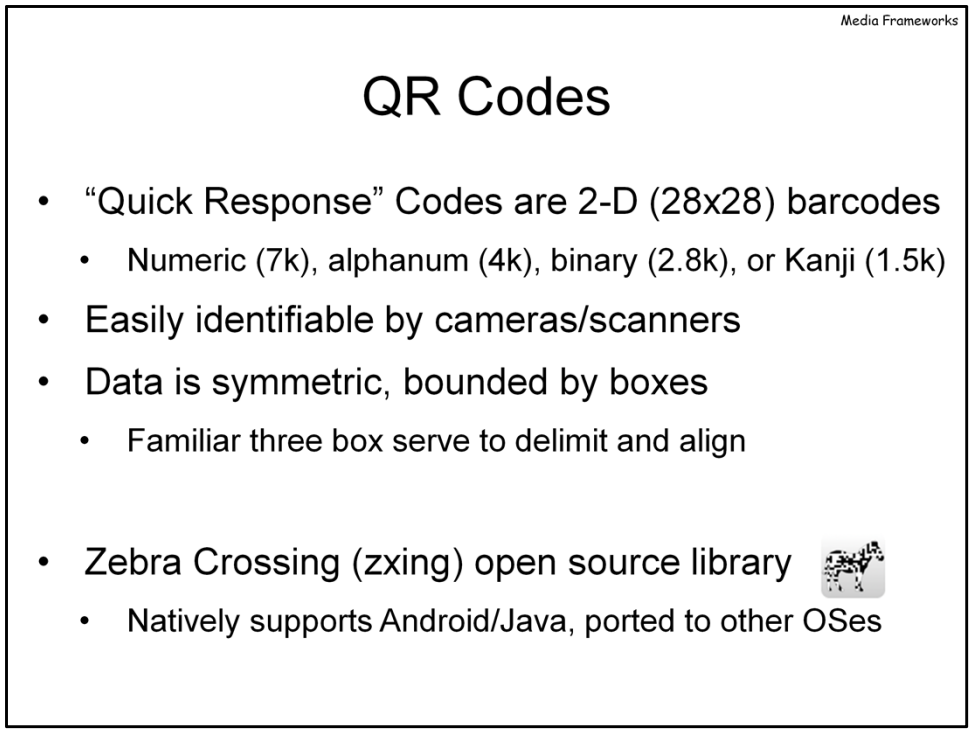


Android's Camera APIs are quite powerful, and constantly evolve. They are a combination of the Android API version, and the vendor's camera feature support. In order to figure out which camera parameters are supported, the following code can be used:

```
// Create an instance of Camera
mCamera = getCameraInstance();

// get Camera parameters
Camera.Parameters params = mCamera.getParameters();
```

Where the "Parameters" are an object which can be queried using one of its myriad getters. The setters allow the enablement of a particular feature, and the actual usage varies with the feature in question. For example., face detection:

```
mCamera.setFaceDetectionListener(new MyFaceDetectionListener());

class MyFaceDetectionListener implements
Camera.FaceDetectionListener {
  @Override
  public void onFaceDetection(Face[] faces, Camera camera) {
       // Handle array of faces.length faces, each found at
       // faces[i].rect.centerX(), faces[i].rect.centerY()
    }
}
```

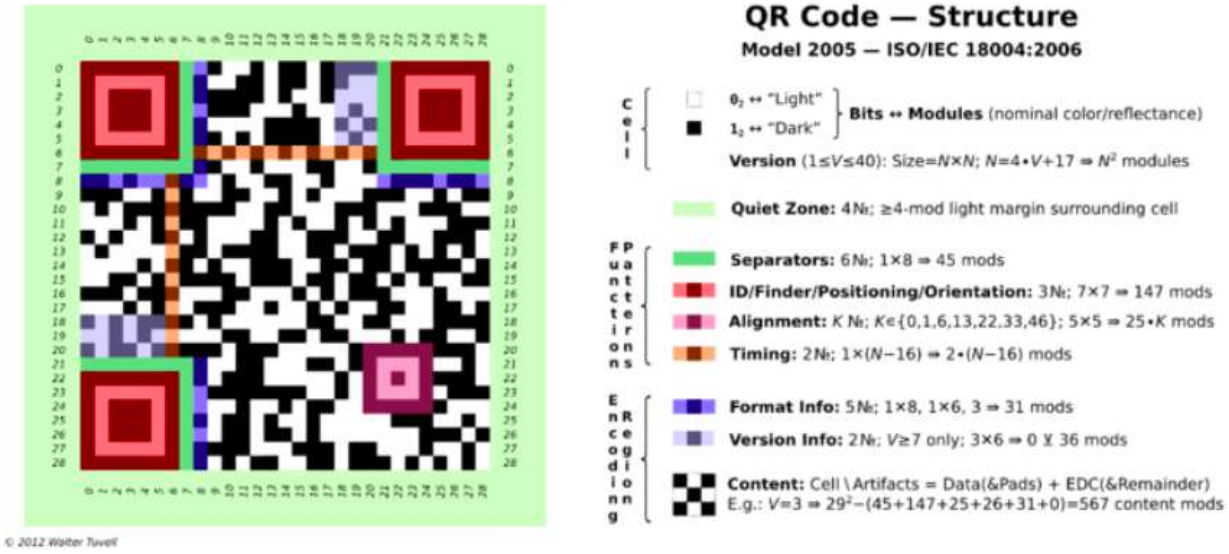


"Quick Response" (or simply QR) codes have exploded in popularity over the past several years, and now appear in ads, prompting the passer-by to "scan this code with your mobile phone" to be redirected to a web-page or obtain other information, such as a vCard. QR Codes are, in effect, two dimensional bar-codes, and operate in the same way: the data is encoded in a clearly visible pattern, with is delimited by fixed markers. These markers (three of them) are recognizable by any scanner due to their fairly large size, and enable software recognizers to "home in" on the tag. Because there are three such markers, there is only one way to align them – so the QR code can be read when the mobile scanner is at any orientation, portrait or landscape.

The amount of data in a QR code is fairly limited – up to about 8k, depending on the data type. The most common datatype (alphanumeric) only allows up to 2.8k, which is why it is commonly used as a link or (in some cases) calendar or contact data. QR Codes also have support for Kanji, since they were developed in Japan, though the limit there is even smaller, about 1.5k or so.

Image data can often be blurry, which is why QR codes use built-in error correction with Reed-Solomon error correcting codes. Error correcting codes imply a certain necessary redundancy, which is part of the reason the amount of data is so limited. Nonetheless, using error correction makes the codes very efficient, because it allows the correct decoding of the QR code by low resolution cameras as well.

The following illustration (from WikiPedia, under Creative Commons), demonstrates the format of a QR code:



The open source “Zebra Crossing” library (http://code.google.com/p/zxing/) is one of several implementations of QR code parsing, and has become the de facto standard for developers who want to plug-in a fairly simple API that hides all the rather complicated encoding details.