DEBUG LOGS

Debug Logs

- Used for debugging and providing information about intermediate state
  - Trace application flow
  - Intermediate variable values
- iOS
  - NSLog
- Android
  - LogCat

NSLog

- FoundationKit function for printing debug statements to console

```
void NSLog (NSString *format, …);
```

- May use c-style format specifiers or Core Foundation object specifiers

```
NSLog ( @”ClassA : x = %d”, x );
NSLog ( @”ClassB : str = %s”, “mystring” );
NSLog ( @”ClassC : myObject = %@”, myObject );
```

- Be sure specifier matches the arguments

```
int i = 123;
NSLog ( @”i = %@”, i );    // Produces error
```

NSLog format specifiers

%@	Object
%d, %i	signed int
%u	unsigned int
%f	float/double
%x, %X	hexadecimal int
%p	pointer
%e	float/double (in scientific notation)
%s	C string (bytes)
%S	C string (unichar)
%c	character
%C	unichar
%lld	long long
%llu	unsigned long long
%Lf	long double

LogCat

- Android logging system mechanism used to view system debug output
- Can be used to view stack trace of emulator errors
  - Useful for locating line of code were error initiated
- LogCat is viewable in realtime in Debug or DDMS view of Eclipse
- Common logging methods

v	- verbose
d	- debug
i	- information
w	- warning
e	- error

- Usage example
  - Log.i(“MyActivity”, “MyClass.memberfunction – info message”);

IOS PROCESSES AND THREADS

Processes

- From developer's perspective, only one process is active
- iOS 4 places closed applications in suspend state to maintain them in memory
- Small number of accepted background processes allowed in iOS 4

Background tasks

- 3 types supported
  - Audio
  - Location
  - Voip
- Other extensions provided for
  - Task completion
    - beginBackgroundTaskWithExpirationHandler:
    - endBackgroundTask:
  - Local notifications

Concurrency

- Operation objects
  - Define operations which can be pushed onto a queue for asynchronous execution
- Block objects and Grand Central Dispatch (GCD)
  - Supported in iOS 4
  - Define operation blocks inline
- Long operations should not be performed on main thread
  - Blocks UI
- Operations on UI should ONLY be performed on main thread

NSOperationQueue

- Concurrent dispatch queue for Cocoa
- Default execution order is first-in, first-out, but may incorporate other factors
  - Task dependencies
  - Execution priorities
- May define multiple queues in your application
- Automatically retains operations, then releases on completion

NSOperationQueue

- Set concurrency level using
  setMaxConcurrentOperationCount:
- Can achieve locks or synchronization using serial queues or operation object dependencies
- To use a queue, allocate, then add operations

```
NSOperationQueue* aQueue = [[NSOperationQueue alloc] init];
[aQueue addOperation:anOp];

. . .

[aQueue release];
```

NSOperation

- Objective-C operation object which encapsulates work to perform and data and data needed to perform it
- Generate key-value observing notifications
  - Useful for monitoring progress of task
- An abstract class that needs to be subclassed
  - NSInvocationOperation
    - If you already have method that performs needed task

NSInvocationOperation

```
@implementation MyCustomClass

- (NSOperation *) taskWithData:(id)data {
    NSInvocationOperation* theOp = [[[ NSInvocationOperation alloc ]
                                   initWithTarget:self
                                   selector:@selector(myTaskMethod:)
                                   object:data ] autorelease ];
    return theOp;
}


// This is the method that does the actual work of the task.
- (void)myTaskMethod:(id)data {
    // Perform the task.
}
@end
```

NSOperation

- Custom subclass
  - Required implementations
    - Custom init
    - main
  - Additional implementations
    - Custom methods to be called in main
    - Accessor methods for data values
    - dealloc

NSOperation

```
@interface MyOperation : NSOperation {
    id myData;
}
-(id)initWithData:(id)data;
@end

@implementation MyOperation
- (id)initWithData:(id)data {
    if (self = [super init])
        myData = [data retain];
    return self;
}

- (void)dealloc {
    [myData release];
    [super dealloc];
}

-(void)main {
    // Do some work on myData and report the results.
}
@end
```

Modifying UI

- To make modifications to UI from operations on another thread, use UIView method

  `performSelectorOnMainThread:withObject:waitUntilDone:`

Task dependencies

- Set in NSOperation after creation, but before queuing
- Dependency not limited to same queue
- Add dependency using

  (void) addDependency:(NSOperation *) operation
- Avoid circular dependencies!
- Can create custom dependency by overriding *isReady* method

Execution priority

- Priority of operation is within scope of queue
- By default priority is *normal*
- Modify priority using

  - (void) setQueuePriority:(NSOperationQueuePriority) priority

- Valid values
  - NSOperationQueuePriorityVeryLow
  - NSOperationQueuePriorityLow
  - NSOperationQueuePriorityNormal
  - NSOperationQueuePriorityHigh
  - NSOperationQueuePriorityVeryHigh

KVO compliance

- NSOperation is key-value observing compliant for following key paths
  - isCancelled
  - isConcurrent
  - isFinished
  - isReady
  - dependencies
  - queuePriority
  - completionBlock
- If overriding more than main in NSOperation, need to maintain KVO compliance

Dispatch queues

- Grand Central Dispatch queues manage queues of task to be operated
- All dispatch queues are first-in, first-out
- Predefined types
  - Serial
    - Supports multiple self-defined queues
  - Concurrent
    - 3 global predefined queues of differing priority
  - Main dispatch queue

Blocks

- A self contained unit of work
- Typically defined within another function, so it can access variables within that scope
- May be assigned to a variable or passed as an argument

```
typedef double (^my_op_t)(double op);
my_op_t square;
square = ^(double operand) {
  return operand * operand;
}
```

Queues

- Getting the main queue (UI queue)

  dispatch_queue_t dispatch_get_main_queue()
- Creating a serial queue

  dispatch_queue_t dispatch_queue_create (
  const char *label, NULL)
- Releasing a serial queue

  void dispatch_release(dispatch_queue_t)
  - Won’t release queue until it is empty

Queues

- Adding blocks to a queue

```
void dispatch_async(dispatch_queue_t queue,
                    dispatch_block_t block)
```

- Block may be defined inline when adding to queue

Grand central dispatch example

```
- (void) viewWillAppear:(BOOL)animated {
   NSString *url = photo.url;
   dispatch_queue_t downloadQ = dispatch_queue_create
                        ( “picdownload”, NULL );
   dispatch_async( downloadQ, ^{
     NSData *imgData = [ImgFetcher getDataForUrl:url];
     dispatch_async( dispatch_get_main_queue(), ^{
       UIImage *img = [ UIImage imageWithData:imgData ];
       self.imgView.image = img;
     });
   });
   dispatch_release( downloadQ );
}
```