This Week’s Topics

• Bonjour
• GameKit
• Core Location
• Map Kit
Bonjour
Bonjour

- System for publishing and discovering services on a network without a centralized server
  - An open standard for “zero-configuration” networking

- Example services
  - Resources
    - Printers – Take your laptop to some new location and print
  - App Services
    - Music – Find and play music from another machine’s iTunes Library
Foundation Classes

- NSNetService represents an actual instance of a service
- NSNetServiceBrowser queries the network for available services and collects the results
Asynchronous Results & Delegates

• NSNetService & NSNetServiceBrowser ops are long-running
  • Operations implemented asynchronously

• Need to specify delegate

• Need to implement callback methods
Network Services Operations

• Three fundamental operations
• Publication (advertising a service)
• Discovery (browsing for available services)
• Resolution (translating service names to network addresses)
Service Publication

• You provide a service name, type and port

• Service name: Human readable UTF-8 string (63 char max)
  - e.g., “Canon MP780”

• Type: “_applicationprotocol._transportprotocol”
  - e.g., _printer_.tcp

• Port: Typically in the range 1024 - 49151

```objective-c
NSNetService *service;
service = [[NSNetService alloc] initWithDomain:@"
    type:@"_ipp._tcp"
    name:@"Canon MP780"
    port:4721];
[service setDelegate:delegateObject];
[service publish];
```
Service Publication

• Status sent to delegate

- (void)netServiceWillPublish:(NSNetService *)sender

- (void)netServiceDidPublish:(NSNetService *)sender
didNotPublish:(NSDictionary *)errorDict

• Always remember to unset the delegate in dealloc!

- (void)dealloc {
    [<service setDelegate:nil];
    [<service stop];
    [<service release];

    [super dealloc];
}

Monday, November 15, 2010
Service Discovery

- Create an instance of NSNetServiceBrowser
- Assign delegate object
- Search for a service by type and search domain

```swift
browser = [[NSNetServiceBrowser alloc] init];
[browser setDelegate:self];
[browser searchForServicesOfType:@"_ipp._tcp." inDomain:@""];
```

- Delegate informed about search progress & service lifecycle
// Searching status
- (void)netServiceBrowserWillSearch:(NSNetServiceBrowser *)browser
- (void)netServiceBrowserDidStopSearch:(NSNetServiceBrowser *)browser
- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
didNotSearch:(NSDictionary *)errorInfo

// Service coming / going
- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
didFindService:(NSNetService *)service
    moreComing:(BOOL)more

- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
didRemoveService:(NSNetService *)service
    moreComing:(BOOL)more
Service Resolution

• Found services represented as NSNetService objects

• To use a found NSNetService
  ▪ Tell it to resolve (get address / port information)
  ▪ Handle delegate callbacks
  ▪ Ask it for an NSInputStream or NSOutputStream as needed
Service Resolution

- NSSNetServices found by NSSNetServiceBrowser must have their addresses resolved before use:

  ```
  [netService setDelegate:self];
  [netService resolveWithTimeout:5];
  ```

- Status communicated asynchronously to delegate:

  ```
  - (void)netService:(NSSNetService *)sender
    didNotResolve:(NSDictionary *)errorDict;
  - (void)netServiceDidResolveAddress:(NSSNetService *)sender;
  ```
Service Resolution

• Once a service has been resolved you can use the address information to connect to it

• `NSNetService` will generate `NSStream` instances for you

```c
NSInputStream  *inputStream  = nil;
NSOutputStream *outputStream = nil;

[netService getInputStream:&inputStream
    outputStream:&outputStream];
```
NSStreams
**NSStream**

- Abstract class for device-indep. reading/writing of bitstreams
  - Data can be streamed to/from memory, files, network, etc.

- Using an NSStream

1. Create & configure stream object
2. Schedule stream object on a run loop & open
3. Delegate object handles stream events
   - Writes / Reads are synchronous
   - State changes are asynchronously sent to the delegate
4. When finished, dispose of stream
Using NSStream

• Opening a stream

```objective-c
[stream setDelegate:self];
[stream scheduleInRunLoop:[NSRunLoop currentRunLoop]  
  forMode:NSRunLoopCommonModes];
[stream open];
```

• Closing a stream

```objective-c
[stream close];
[stream removeFromRunLoop:[NSRunLoop currentRunLoop]  
  forMode:NSRunLoopCommonModes];
[stream setDelegate:nil];
```
NSRunLoop

- An event processing loop used to schedule work & handle events
- Receives events
  - Asynchronously from input sources
  - Synchronously from timer sources
- Initiates callbacks to delegate objects
Run Loop Modes

• A collection of sources to be monitored and a collection of run loop observers to be notified
  - Events processing can be restricted to only certain run loop modes
  - Can temporarily block certain events while processing others
    - e.g., NSEventTrackingRunLoopMode used for mouse movements
• NSRunLoopCommonModes represents a group of common modes (including Default, Modal & Event tracking).
• You can also define your own run loop mode to only service your events
NSStream Delegate Protocol

• Handling Events

- (void)stream:(NSStream *)theStream
  handleEvent:(NSStreamEvent)streamEvent

• Several different event types, e.g.,

NSSStreamEventOpenCompleted
NSSStreamEventHasSpaceAvailable
NSSStreamEventHasBytesAvailable
NSSStreamEventErrorOccurred
NSSStreamEventEndEncountered
NSOutputStream

• Writing

- (NSInteger)write:(const uint8_t *)buffer
  maxLength:(NSUInteger)length

• For instance:

```c
// Assume outputStream is an already opened NSOutputStream
// with space available.

const char *buff = “Hello World!”;
NSUInteger buffLen = strlen(buff);
NSInteger writtenLength =
    [outputStream write:(const uint8_t *)buff
        maxLength:strlen(buff)];
if (writtenLength != buffLen) {
    [NSException raise:@”WriteFailure” format:@””];
}
```
**NSInputStream**

- **Reading**

```c
- (NSInteger)read:(uint8_t *)buffer
  maxLength:(NSUInteger)length
```

- **For instance:**

```c
// Assume inputStream is an already opened NSInputStream
// with space available.

uint8_t buff[1024];
bzero(buff, sizeof(buff));
NSInteger readLength =
    [inputStream read:buff
     maxLength:sizeof(buff) - 1];
buff[readLength] = '\0';
NSLog(@"Read: %s", (char *)buff);
```
NSKeyedArchiver For Network Tx

- Create an NSData object containing the data you want to send

```objective-c
NSData *objectArchive = [NSKeyedArchiver archivedDataWithRootObject:someObject];
[outputStream write:outgoingData.bytes maxLength:outgoingData.length];
```

- On the receiving side, read using the NSInputStream

- Convert from NSData to your object using NSKeyedUnarchiver
Demo

Bonjour-Messaging
Demo

• Server
  ▪ Creates socket for incoming connections
  ▪ Publishes service with NSNetServices
  ▪ Accepts socket connections and creates a NSInputStream to read

• Client
  ▪ Browse for services using NSNetServiceBrowser
  ▪ Display found services in UITableView
  ▪ Selected service is resolved and NSOutputStream acquired

• Users can send simple text message from client to server
More Example Code

• WiTap
  ▪ Simple Bonjour Demo
Game Kit
Game Kit

- Designed for game developers, useful to others
- Peer-to-peer connectivity
  - Find and connect to nearby iPhones using over bluetooth
  - Easy NSData exchanging APIs
  - Useful for any kind of data exchange, or collaborative application
    - Games, Chat, Social Networking
- In–Game Voice
  - Create a voice chat between iPhones
- Game Center
  - Social gaming network support
Peer-to-Peer Connectivity

- GKSession class
  - Create & manage an ad-hoc Bluetooth or local wireless network
  - Encapsulates a *session* object used by *peers* to exchange data
  - Allows discovery and connection to nearby *peers*
  - Connection requests processed by *delegate*
  - Send / receive data – receive by providing a *data handler*
Peer-to-Peer Connectivity

- GKSession class
  - Create & manage an ad-hoc Bluetooth or local wireless network
  - Encapsulates a session object used by peers to exchange data
  - Allows discovery and connection to nearby peers
  - Connection requests processed by delegate
  - Send / receive data – receive by providing a data handler
  - Peers identified by a peerID, used for all communication to other peers
GKSession

- (id)initWithSessionID:(NSString *)sessionID
displayName:(NSString *)displayName
sessionMode:(GKSessionMode)mode;

- **mode**
  - server: advertise sessionID
  - client: search for server advertising sessionID
  - peer: both server & client

- **displayName** – human readable text (if *nil* uses device name)

- **sessionId** – distinguishes your game / service (if *nil* uses bundle ID)
GKSession Properties

- **peerID** – unique identifier for each peer in a session

- **GKSessionDelegate**
  - Changes to peer’s state
  - Connection request processing
  - Session errors

- **data handler** – object supplied by you that handles data received from other connected peers
GKPeerPickerController

- Provides a standard peer picking UI
GKPeerPickerController

- Provides a standard peer picking UI
  - Handles all the connection / negotiation logic
  - Bluetooth connections, for internet you need to do your own work
- To use, you need to supply a delegate conforming to GKPeerPickerControllerDelegate protocol
Connecting To Peers Using GKPeerPickerController

• Create and display the picker (Bluetooth connection type shown)

```objective-c
GKPeerPickerController *picker = [[GKPeerPickerController alloc] init];
[picker setDelegate:self];
[picker setConnectionTypesMask: GKPeerPickerControllerConnectionTypeNearby];
[picker show];
```

• Peer picker asks for your session object (uses default if method is unimplemented)

```objective-c
- (GKSession *)peerPickerController:(GKPeerPickerController *)picker
  sessionForConnectionType:(GKPeerPickerControllerConnectionType)type;
```

• Peer picker tells you when a new peer connection is selected

```objective-c
- (void)peerPickerController:(GKPeerPickerController *)picker
  didConnectPeer:(NSString *)peerID toSession:(GKSession *)session;
```
Connecting to a single peer

- (void)showPeerPicker {
    GKPeerPickerController *picker = [[GKPeerPickerController alloc] init];
    [picker setDelegate:self];
    [picker show];
    [picker release]; // picker retains itself while shown
}

- (GKSession *)peerPickerController:(GKPeerPickerController *)picker
    sessionForConnectionType:(GKPeerPickerConnectionType)type
{
    if (!mySession) {
        mySession = [[GKSession alloc] initWithSessionID:com.myCompany.myApp.mService
            displayName:nil
            sessionMode:GKSessionModePeer];
    }
    return mySession;
}

- (void)peerPickerController:(GKPeerPickerController *)picker
    didConnectPeer:(NSString *)peerID
    toSession:(GKSession *)session
{
    // ‘session’ is now connected, do something awesome!
}
Sending Data To Peers

• Pack up as an NSData and send

• All data sending is asynchronous, returns when sending begins

```c
GKSession *session = ...;

[session sendData:packet
toPeers:[NSArray arrayWithObject:gamePeerId
withDataMode:GKSendDataReliable error:&error];
```

• Options for delivery
  ▪ Reliable – Each message acknowledged
    ▪ Slower, but delivery and order guaranteed
  ▪ Unreliable – Fire and forget
    ▪ Faster, but message may be dropped or arrive out of order
Receiving Data From Peers

• Provide a **data handler** for processing incoming data

```objective-c
[self.gameSession setDataReceiveHandler:self withContext:NULL];

-(void)receiveData:(NSData *)data fromPeer:(NSString *)peer
              inSession:(GKSession *)session context:(void *)context
{
    // use the data...
}
```

• Strategies for data exchange
  
  ▪ GKTank
  
  ▪ “Connected iPhone Apps: GameKit vs CFNetwork Prezi” by Peter Bakhirev
    
    ▪ Efficiency vs. Flexibility
    
    ▪ [http://prezi.com/zuxwtxbsiean](http://prezi.com/zuxwtxbsiean)
Watching For Disconnects

- Among other things, the GKSession delegate will be informed of connected peer state changes

```
[self.gameSession setDelegate:self];

- (void)session:(GKSession *)session
    peer:(NSString *)peerID
    didChangeState:(GKPeerConnectionState)state
{
    if(state == GKPeerStateDisconnected) {
        // A peer disconnected for some reason...
    }
}
```
More Other...

• More fine grained control and options we didn’t discuss

• “Game Kit Programming Guide” – Apple online documentation

• Getting peer-to-peer up and running quickly (sample code)
  ▪ **GKTank** – Apple sample source
  ▪ **Beam It!** – Peer-to-peer vcard sharing app, with source!
  ▪ **TicTacToeNew** – Great architecture for apps using both GameKit and Bonjour
In–Game Voice

• Implemented on top of network connections you provide
  ▪ e.g, over a bluetooth connection provided by GKSession

• GKVoiceChatService controls the audio
  ▪ Samples the microphone, asks its client to send the data
  ▪ Plays audio received from the other participant

• You provide a object conforming to GKVoiceChatClient protocol
  ▪ Client object connects participants so that the voice chat service can exchange configuration data
  ▪ Provides callbacks GKVoiceChatService calls when it has data to send over the network
  ▪ Implements property to supply a participantID
In-Game Chat  Using GKSession
In-Game Voice  Using GKSession

• Configure the audio session for playback and recording

```swift
AVAudioSession *audioSession = [AVAudioSession sharedInstance];
[audioSession setCategory:AVAudioSessionCategoryPlayAndRecord error:myErr];
[audioSession setActive: YES error: &error];
```

• Implement the GKVoiceChatClient methods

```swift
- (NSString *)participantID {
    return mySession.peerID;
}
- (void)voiceChatService:(GKVoiceChatService *)vchatService sendData:(NSData *)data
toParticipantID:(NSString *)participantID {
    [mySession sendData:data toPeer:[NSArray arrayWithObject: participantID] withDataMode:GKSendDataReliable error:&error];
}
```
In-Game Voice  Using GKSession

• Attach your client to the chat service

```objective-c
GKVoiceChatService *vchatService = [GKVoiceChatService defaultVoiceChatService];
MyChatClient *myClient = [[MyChatClient alloc] initWithSession: mySession];
[vchatService setClient: myClient];
```

• Connect to the other participant

```objective-c
GKVoiceChatService *vchatService = [GKVoiceChatService defaultVoiceChatService];
[vchatService startVoiceChatWithParticipantID:otherPeerID error: nil];
```
In–Game Voice

Audio Input

GKVoiceChatService

[client voiceChatService:sendData:toParticipantID:]

Your “Client”

Send Data

GKSession, NSStream, CFNetwork, etc…
In–Game Voice

“participant B”

Audio Output

GKVoiceChatService

[vchatService receivedData:fromParticipantID:]

Your “Client”

Receive Data
GKSession receive handler, NSStream, CFNetwork, etc...