Networking with Bonjour & GameKit

Lecture #18 – Chuck Pisula
This Week’s Topics

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

- System for publishing and discovering services on an network
  - An open standard for “zero-configuration” networking

- Makes LANs self configuring
  - Assign addresses without a DHCP server
  - Map names to addresses without a DNS server
  - Find services without a directory server
Example Services

• Resources
  ▪ Printers – Take your laptop to some new location and print

• App Services
  ▪ Music – Find and play streamed music stored on another machine’s iTunes Library
Advertising A Service

- NSNetServices (or CF) used to publish a service via Bonjour
- You provide a service name, type and port
- Type is something like _myservice_.tcp, or _printer_.tcp
- Service name is human readable UTF-8 string (63 char max)
- Example

```objective-c
NSNetService *service;

service = [[NSNetService alloc] initWithDomain:@"" type:@""_myservice_._tcp"
          name:@""Human Readable Name"
          port:4242];
[service setDelegate:self];
[service publish];
```
Service Name Details

- DNS specific-to-general model
- Local domain assumed if not specified
- Leaving service name blank will use the device’s iTunes name
- Some restrictions to allowed characters / string lengths

Canon MP780._ipp._tcp.local.
Service Name Details

- DNS specific-to-general model
- Local domain assumed if not specified
- Leaving service name blank will use the device’s iTunes name
- Some restrictions to allowed characters / string lengths

```c
service = [[NSNetService alloc] initWithDomain:@""
                type:@"_ipp._tcp"
                name:@"Canon MP780"
                port:4721];
```
Publishing

- Status sent to delegate

- (void)netServiceWillPublish:(NSNetService *)sender

- (void)netServiceDidPublish:(NSNetService *)sender

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

- Always remember to unset the delegate in dealloc!

- (void)dealloc {
    [_service setDelegate:nil];
    [_service stop];
    [_service release];
    [super dealloc];
}
Finding A Service

- Search for a service by type and search domain

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

- Delegate informed about services coming and going
- Found services represented as a NSNetService object
- To use a found NSNetService
  - Tell it to resolve (get address / port information)
  - Handle important delegate callbacks
  - Ask it for an NSInputStream or NSOutputStream as needed
NSNetServiceBrowser – Delegate

// 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

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

```objective-c
[netService setDelegate:self];
[netService resolveWithTimeout:5];
```

• Status communicated asynchronously to delegate:

```objective-c
- (void)netService:(NSSNetService *)sender
didNotResolve:(NSDictionary *)errorDict;

- (void)netServiceDidResolveAddress:(NSSNetService *)sender;
```

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

```objective-c
NSOutputStream *outputStream;
[aNetService getInputStream:nil outputStream:&outputStream];
```
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
Demo

Bonjour-Message
Useful Demo / Example Code

- WiTap
  - Simple Bonjour Demo
Useful Demo / Example Code

- TicTacToeNew
  - http://code.google.com/p/iphonebits/source/browse/trunk/src/TicTacToeNew
  - Great example architecture for apps using both GameKit and Bonjour
Are You Connected?

- A question of network reachability...

- Use functions in `SystemConfiguration.framework`
  - Found in `SCNetworkReachability.h`
  - On Mac OS X, this framework provides more features...

- Status of system’s current networking configuration

- Reachability of a target host
  - “reachable” – when a data packet, can leave the local host
  - Reachability cannot tell if you can connect to a particular host, only that an interface is available that might allow a connection
NSStreams
Service Resolution

- **NSNetService** will generate **NSStream** instances for you

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

[netService getInputStream:&inputStream
 outputStream:&outputStream];
```
What’s An NSStream?

• Sort of like sockets, but without select
• State changes are asynchronously sent to the delegate
• Writes / Reads are still synchronous
• You can support multiple streams and still operate on a single thread
• Device agnostic - we’ll use sockets, but could easily be files, memory locations, etc.
NSInputStream

NSOutputStream

NSStream

NSObject
NSStream Class

• Opening a stream

```ObjC
[stream setDelegate:self];
[stream scheduleInRunLoop:[NSRunLoop currentRunLoop]
    forMode:NSRunLoopCommonModes];
[stream open];
```

• Closing a stream

```ObjC
[stream close];
[stream removeFromRunLoop:[NSRunLoop currentRunLoop]
    forMode:NSRunLoopCommonModes];
[stream setDelegate:nil];
```
What’s A Run Loop?

- Easy event processing
  - You’ve been using them, but you don’t even know it!

- Scheduling the NSStream on the NSRunLoop causes it to send its events when that run loop spins.
Okay, What’s A Run Mode?

- Run loops have an unbounded number of run loop modes.
- Events (sources, timers, etc) are scheduled to run only in certain run loop modes.
- This allows you to block events from occurring during high-feedback event loops
  - For instance, `UITrackingRunLoopMode` is used for tracking finger touches. Not servicing other sources here can be a huge responsiveness win.
- `NSRunLoopCommonModes` includes the publicly defined common modes (including tracking). You can also define your own run loop mode to only service your events.
NSStream Delegate Call

- Just a single method

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

- Several different event types
  - Some examples:

```c
NSStreamEventOpenCompleted
NSStreamEventHasSpaceAvailable
NSStreamEventErrorOccurred
NSStreamEventEndEncountered
```
NSOutputStream

• Only one method you’ll really use

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

// 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

- Two useful methods, but we’ll focus on one

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

- For instance:

```c
// 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);
```
Using NSKeyedArchiver

- Create a small object representing the data you want to send
- Convert it to an NSData, with a simple header

```swift
NSData *objectArchive = [NSKeyedArchiver archivedDataWithRootObject:someObject];
int packetLength = [objectArchive length];

NSMutableData *outgoingData = [NSMutableData data]
[outgoingData appendBytes:&packetLength length:sizeof(int)];
[outgoingData appendData:rawPacket];

[outputString write:outgoingData.bytes maxLength:outgoingData.length];
```

- On the receiving side, read using the NSInputStream
- Convert from NSData to your object using NSKeyedUnarchiver
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
    - Example – Bump app (bump devices to exchange contact info)

- In–Game Voice
  - Create a voice chat between iPhones
Game Kit

- Create Sessions
- Find Peers
- Data passing
- In-Game Voice

- Support
  - No support on first generation iPhone and iPod Touch
  - No support on simulator
Peer-to-Peer Connectivity

- GKSession class
  - Encapsulates a *session* object used by *peers* to exchange data
  - Allows discovery and connection to nearby *peers*
  - Connection requests processed by your *delegate*
  - Send / receive data – receive by providing a *data receive handler*
Peer-to-Peer Connectivity

• GKSession class
  - Encapsulates a **session** object used by **peers** to exchange data
  - Allows discovery and connection to nearby **peers**
  - Connection requests processed by your **delegate**
  - Send / receive data – receive by providing a **data receive handler**
  - Peers identified by a **peerID**, used for all communication to other peers
GKSession

```objective-c
GKSession
- (id)initWithSessionID:(NSString *)sessionID
displayName:(NSString *)displayName
sessionMode:(GKSessionMode)mode;
```

- **mode** – server, client, or peer?
  - Peer – acts like a server (advertise) and client (search) simultaneously

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

- **sessionID** – distinguishes your game / service (if *nil* uses bundle ID)
  - If mode is “server” – advertises the sessionID
  - If mode is “client” – searches for the sessionID
GKSession Properties

- **peerID** property – unique identifier for each peer in a session
- **GKSessionDelegate** – allows you to interact with a GKSession
  - All callback methods are optional
  - Connection state changes (connecting, disconnected, connected, ...)
  - Connection request processing (accept, deny)
  - Session errors
- **data receive 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 interact, you supply a delegate conforming to GKPeerPickerControllerDelegate protocol
  - Provide the GKSession to the picker
    - If not implemented, a default peer mode session is created
  - Completed connection callbacks
Connecting To Peers

- Simplest way is to use the GKPeerPickerController
- Possible to create connections without GKPeerPickerController
  - Create GKSessions by hand
  - Server
    - Advertises by setting `available` property to YES
    - Accept, deny connections via GKSession delegate, etc…
  - Client
    - Begin searching by setting `available` property to YES
    - Client can watch for peers and implement its own picker
    - Try to connect using GKSession APIs
Connecting To Peers

- Simplest way is to use the GKPeerPickerController

- Create and display the picker

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

- Peer picker asks for your session object

```objective-c
-(GKSession *)peerPickerController:(GKPeerPickerController *)picker
    sessionForConnectionType:(GKPeerPickerConnectionType)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 Peers

- Connecting to a single peer

```objective-c
- (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:(GKSSession *)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

```objective-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 receive 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

```c
[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
Demo

GameKitSample