

CS193P - Lecture 17

iPhone Application Development

Bonjour
NSStream
GameKit

Announcements

- All Paparazzi assignments should be in!
- Work on your final projects

- Final exam is Thursday, 3/18
 - 12:15 - 3:15pm
 - Hewlett 201
 - We will work on schedule for demos
 - If you have any special requests, let us know

Topics

- Bonjour
 - Automatic Configuration
- NSStream
 - Asynchronous communication



Bonjour

Bonjour

- Three main functions:
 - Automate address distribution and name mapping
 - Publish availability of a service
 - Discover available services
- Open protocol Apple submitted to IETF
 - www.zeroconf.org

Bonjour

- Makes LANs self configuring
 - Requires no administration
 - Assign addresses without a DHCP server
 - Map names to addresses without a DNS server
 - Find services without a directory server

Automatic Addressing

- Bonjour will pick a random address, see if it is in use
 - If it is not in use, it's yours
 - If it is in use, try again
- Uses ".local." as a virtual top-level domain
 - For example: iPhone3G.local.

Advertising Services

- Applications provide a service name and port
- Follows same DNS specific-to-general model
- **ServiceName._ServiceType._TransportProtocolName.Domain**
 - **Service Name** is a human readable descriptive name
 - Maximum of 63 octets of UTF-8
 - All characters are allowed

Advertising Services

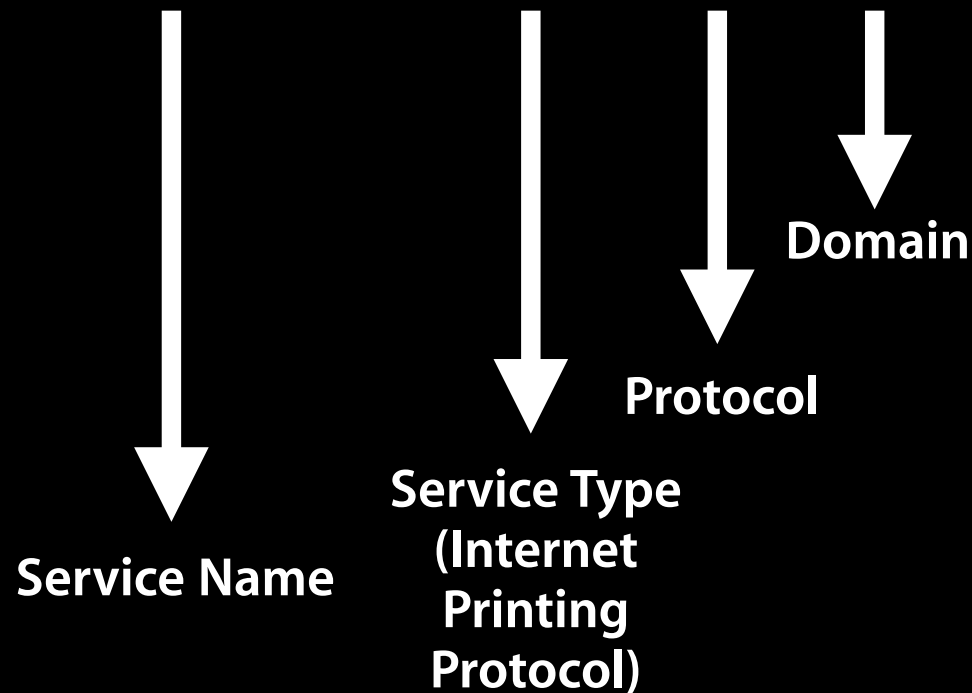
- Applications provide a service name and port
- Follows same DNS specific-to-general model
- **ServiceName**._ServiceType._TransportProtocolName.Domain
 - Service Type is an IANA registered protocol name
 - Maximum of fourteen characters
 - Format of [a-z0-9]([a-z0-9\-*][a-z0-9])?

Advertising Services

- Applications provide a service name and port
- Follows same DNS specific-to-general model
- **ServiceName._ServiceType._TransportProtocolName.Domain**
 - Transport Protocol Name is either TCP or UDP
 - Your own awesomely inventive protocol is not supported...

Service Naming

Canon MP780._ipp._tcp.local.



Publishing a Service

- `NSNetService` is used to publish services via Bonjour

```
NSNetService *_service;
```

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

- Leaving domain blank implies ".local."
- Leaving `name` blank will use the device's iTunes name

Publishing a Service

- `NSNetService` is entirely asynchronous

```
// Set up delegate to receive callbacks
[_service setDelegate:self];

[_service publish];
```

- Always remember to unset the delegate in dealloc!

```
- (void)dealloc {
    [_service setDelegate:nil];
    [_service stop];
    [_service release];

    [super dealloc];
}
```

NSNetService Delegate Methods

- Conflict resolution handled automatically
- Status is communicated to the delegate
 - (void)`netServiceWillPublish:(NSNetService *)sender`
 - (void)`netServiceDidPublish:(NSNetService *)sender`
 - (void)`netService:(NSNetService *)sender`
`didNotPublish:(NSDictionary *)errorDict`
 - `errorDict` is like an NSError - has two keys, one for error domain and one for error code.

Finding a Service

- Applications register service names with local daemon which handles responding to lookup queries
- Service discovery is completely independent of service implementation
- Resolving a service gives you an address and a port
 - Can also get NSStreams pointing to that location

Finding a Service

- NSNetServiceBrowser is used to search for services on the network.

```
NSNetServiceBrowser *_browser;

_browser = [[NSNetServiceBrowser alloc] init];

[_browser setDelegate:self];
[_browser searchForServicesOfType:@"_ipp._tcp."
    inDomain:@""];
```


NSNetServiceBrowser Delegate Methods

- NSNetServiceBrowser browsing is also asynchronous
 - Delegate methods called as services come and go
- ```
- (void)netServiceBrowserWillSearch:(NSNetServiceBrowser *)browser
- (void)netServiceBrowserDidStopSearch:(NSNetServiceBrowser *)browser

- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
 didNotSearch:(NSDictionary *)errorInfo

- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
 didFindService:(NSNetService *)service
 moreComing:(BOOL)more

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

# Service Resolution

- NSNetServices found by NSNetServiceBrowser must have their addresses resolved before use:

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

- Status communicated asynchronously to delegate:

```
- (void)netService:(NSNetService *)sender
 didNotResolve:(NSDictionary *)errorDict;
 • Same errorDict as before.
- (void)netServiceDidResolveAddress:(NSNetService *)sender;
```

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

# Bonjour Service Publishing and Searching Demo

# NSStreams

# Service Resolution

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

# Service Resolution

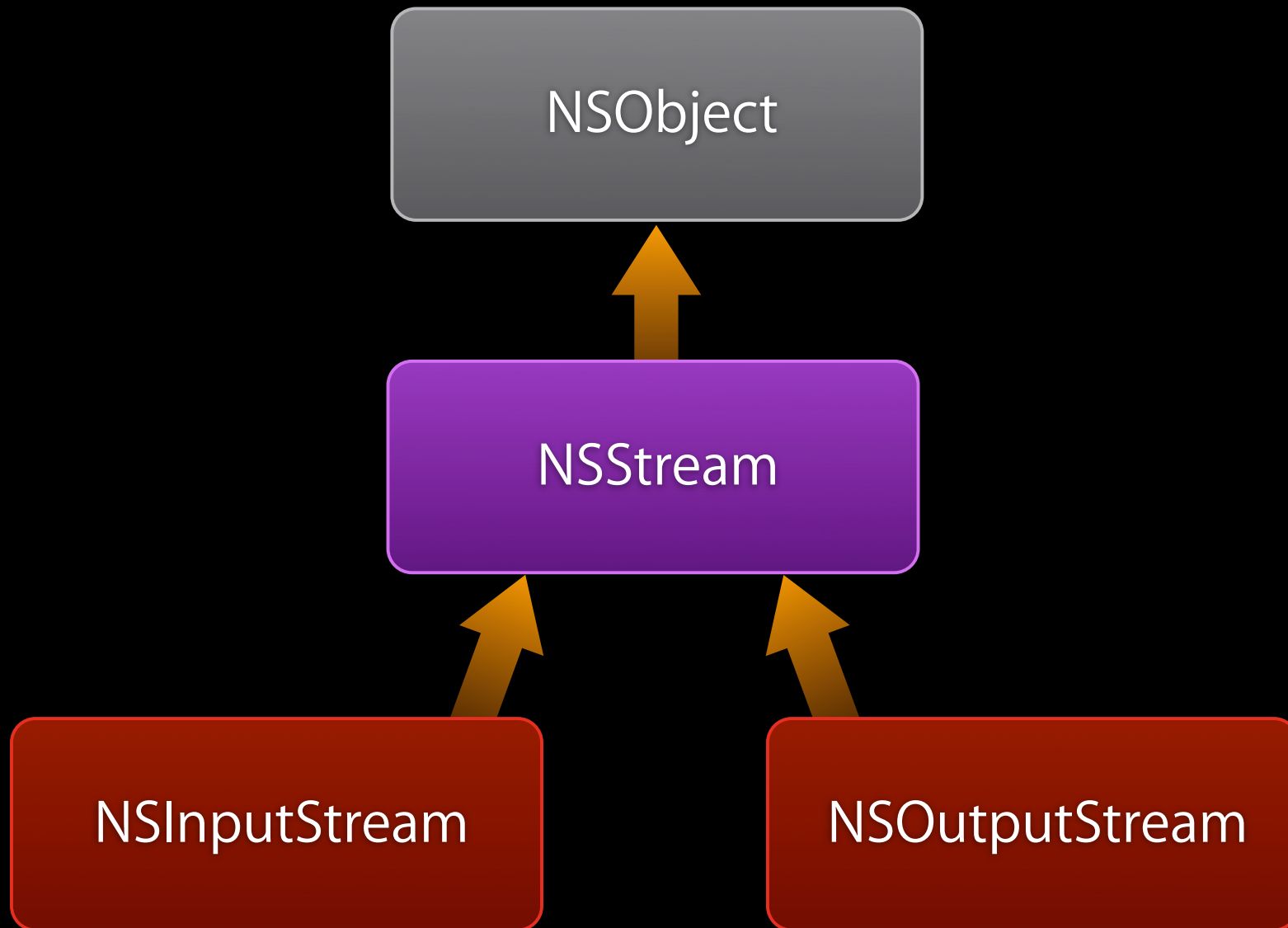
- `NSNetService` will generate `NSStream` instances for you

```
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.





# NSStream Class

- Opening a stream

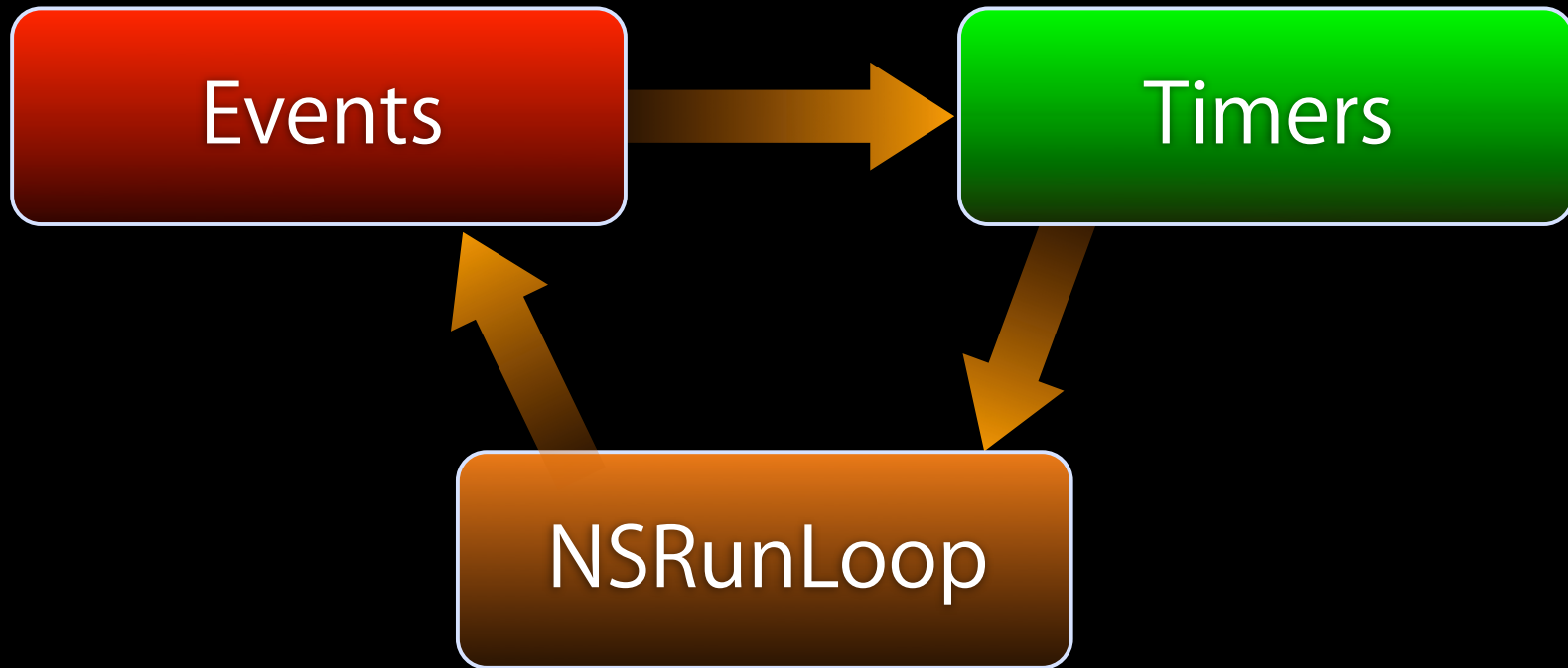
```
[stream setDelegate:self];
[stream scheduleInRunLoop:[NSRunLoop currentRunLoop]
 forMode:NSRunLoopCommonModes];
[stream open];
```

- Closing a stream

```
[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

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

- Several different event types
  - Some examples:

```
NSStreamEventOpenCompleted
```

```
NSStreamEventHasSpaceAvailable
```

```
NSStreamEventErrorOccurred
```

```
NSStreamEventEndEncountered
```

# NSOutputStream

- Only one method you'll really use

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

- For instance:

```
// 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:

```
// 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);
```

# Messaging with NSStream

# GameKit

Discovery and Connectivity



# GameKit

- Peer-to-Peer connectivity
  - Abstracts away Bonjour and Stream creation
  - Nearby (Bluetooth) & Online (Wifi) support
- In-Game Voice
  - Facilitates voice communication between two devices

# GameKit Classes

- GKPeerPickerController
  - Presents UI prompting user to search for peers
  - Facilitates creation of GKSessions
- GKSession
  - Manages streams of data between peers
  - Allows sending to single peer, or broadcast to all
- GKVoiceChatService
  - Manages audio between peers
  - Controls volume, detecting whether peer or local user is speaking

# GKSession

- Manages discovery of peers
  - Abstracts streaming and Bonjour code
- ```
- (id)initWithSessionID:(NSString *)sessionID  
    displayName:(NSString *)name  
    sessionMode:(GKSessionMode)mode
```

```
typedef enum {  
    GKSessionModeServer,  
    GKSessionModeClient,  
    GKSessionModePeer  
} GKSessionMode;
```

GKSession

- Properties include:
 - **displayName** - Your name, as seen by your peers
 - **peerID** - unique ID, identifying your Session to your peers
 - **sessionID** - String used by your Application to filter peers
- Delegate methods:
 - Notifies of state change of a peer's session
 - Notifies of connection requests
 - Notifies of errors with sessions or connections

GKSession - Sending & Receiving Data

- Send data to specific peers

```
- (BOOL)sendData:(NSData *)data toPeers:(NSArray *)peers  
withDataMode:(GKSendDataMode)mode error:(NSError **)error;
```

- Send data to ALL connected peers

```
- (BOOL)sendDataToAllPeers:(NSData *)data withDataMode:  
(GKSendDataMode)mode error:(NSError **)error;
```

- Delegate method to receive data:

```
- (void)receiveData:(NSData *)data fromPeer:(NSString *)peer  
inSession:(GKSession *)session context:(void *)context;
```

GKPeerPickerController

- Provides UI to connect to peer
- Allows user to pick between “Nearby” and “Online”
- Nearby
 - Bluetooth communication
 - Most of the work is handled for you under-the-hood
- Online
 - Hands off responsibility to the application
 - App builds or connects to server
 - Associates users in server
 - App is responsible for handling communication

Initiating a connection

```
// Allocate the PeerPickerController
GKPeerPickerController *peerPicker;
peerPicker = [[GKPeerPickerController alloc] init];

// Set up delegate to receive callbacks
peerPicker.delegate = self;
peerPicker.connectionMask =
    GKPeerPickerControllerConnectionTypeOnline |
    GKPeerPickerControllerConnectionTypeNearby;

// Display the Peer Picker
[peerPicker show];
```

Receiving a connection

- Return a session for the requested type

```
- (GKSession *)peerPickerController:  
(GKPeerPickerController *)picker sessionForConnectionType:  
(GKPeerPickerConnectionType)streamEvent {  
  
    return [[[GKSession alloc] initWithSessionID:nil  
            displayName:localName sessionMode:GKSessionModePeer]  
            autorelease];  
  
}
```

- Accept connection from peer

```
- (GKSession *)session:(GKSession *)session  
didReceiveConnectionRequestFromPeer:(NSString *)peerID {  
  
    [session acceptConnectionFromPeer:peerID error:nil];  
  
}
```


GameKit demo