iPhone Application Programming
Networking

Christian Cherek
Media Computing Group
RWTH Aachen University

WS 2015/16
http://hci.rwth-aachen.de/iphone

Networking on iOS

- There are many ways to network
  - Think about what you need to do first
  - Then select the most appropriate technology
- Networking operates on binary data
- All networking is done asynchronously
  - Delegate
  - Blocks

Converting to/from NSData

- NSString (NSString ↔ NSData)
- NSPropertyListSerialization (NSDictionary/NSArray ↔ Plist)
- NSJSONSerialization (NSDictionary/NSArray ↔ JSON)
- NSCoding (Any Object ↔ NSData)

Networking Technologies

GameKit

Multipeer Connectivity

Air Drop

Push Notifications

URL Loading System

Bonjour

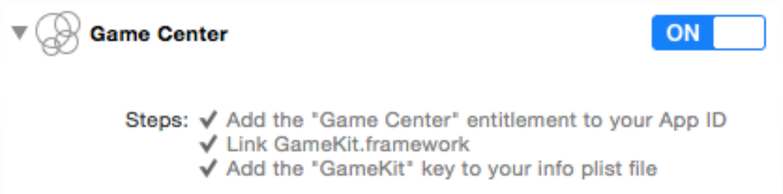
Socket Networking

GameKit

- Leaderboards, Achievements, Challenges, In-game Voice Chat
- Matchmaking
  - Real-time: all players are connected simultaneously and can exchange data
  - Turn-based: players connect sequentially, data is exchanged as needed
  - Self-hosted: Game Center provides the players, you provide the networking

GameKit Checklist

- Set up your app in iTunes Connect
- Enable Game Center in your app's capabilities



- Import GameKit framework

```
import GameKit
```

- Authenticate the local player

GameKit Authentication

```
var localPlayer = GKLocalPlayer.localPlayer()
localPlayer.authenticateHandler = {(viewController : UIViewController?, error : NSError?) -> Void in
    if ((viewController) != nil) {
        self.presentViewController(viewController, animated: true, completion: ^{
            self.didAuthenticate
        })
    }else{
        self.didAuthenticate
    }
}
```

Real-Time Matchmaking

- Create **GKMatchRequest**
- Find Players via **GKMatchmaker** or **GKMatchmakerViewController**
  - Matches can be either **peer-to-peer** or **hosted**
  - To receive invitations, you need to provide an **inviteHandler**
- **GKMatch** to send data
  - sendData to all players

```
func sendData(data: NSData,toPlayers playerIDs: [String],withDataMode mode: GKMatchSendDataMode)
```

Turn-Based Matchmaking

- Create **GKMatchRequest**
- Find players via **GKTurnBasedMatch** or **GKTurnBasedMatchmakerViewController**
  - Or load ongoing matches via **GKTurnBasedMatch**
- Determine current player
- Load and update match data
  - Match data is stored with the match and distributed to all players
- Advance the match to the next player

Documentation

- Game Center Overview
- Game Center Programming Guide
- GameKit Framework Reference

Multipeer Connectivity

- Connect to and exchange data with nearby devices
- Uses Wifi, peer-to-peer Wifi, or Bluetooth personal area network

Multipeer Connectivity Overview

- Create an **MCPeerID** for your device
- Create an **MCSession**
- Create an **MCNearbyServiceAdvertiser** or **MCAdvertiserAssistant** to advertise your device
- Create an **MCNearbyServiceBrowser** or **MCBrowserViewController** to find and connect to advertised devices
- Exchange data via the **MCSession** object

Multipeer Connectivity Example

```
//set up peerID
let myPeerId = MCPeerID(displayName: "uniqueID")

lazy var session : MCSession = {
     let session = MCSession(peer: self.myPeerId, securityIdentity: nil, encryptionPreference:
     MCEncryptionPreference.Required)
     session.delegate = self
    return session
}()

//set up advertising service
var serviceAdvertiser : MCNearbyServiceAdvertiser
func setupAdvertiser() {
    self.serviceAdvertiser = MCNearbyServiceAdvertiser(peer: myPeerId, discoveryInfo: nil, serviceType: myServiceType)
    self.serviceAdvertiser.delegate = self
    self.serviceAdvertiser.startAdvertisingPeer()
}

//set up peer browser
var browserViewController : MCBrowserViewController
func presentBrowser() {
    self.browserViewController = MCBrowserViewController(serviceType: myServiceType, session: session)
    browserViewController.delegate = self
    self.presentBrowser()
}
```

Multipeer Connectivity Example (cont.)

```
func sendData(NSData, toPeers: [MCPeerID], withMode: MCSessionSendDataMode)

func startStreamWithName("streamName", toPeer: MCPeerID)
```

```
func session(session: MCSession, didReceiveData data: NSData, fromPeer peerID: MCPeerID)

func session(session: MCSession, didReceiveStream stream: NSInputStream,
        withName streamName: String, fromPeer peerID: MCPeerID) -> NSOutputStream
```

Documentation

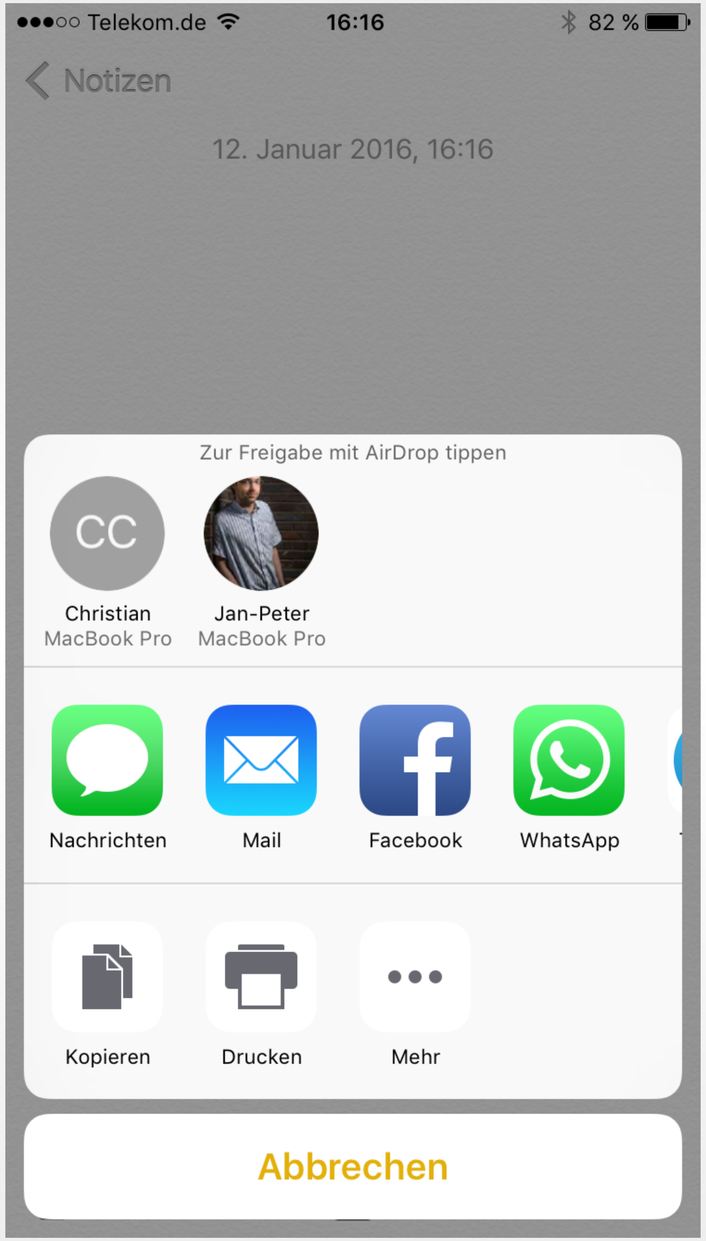
- Multipeer Connectivity Framework Reference

AirDrop

- Exchange data with nearby devices
- Uses Wifi, peer-to-peer, or Bluetooth

AirDrop Checklist

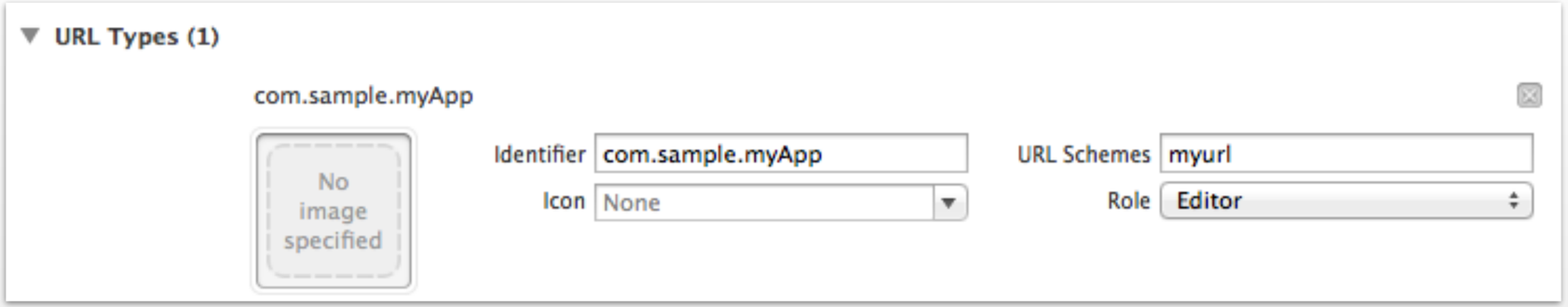
- Transfer files between iOS and Mac devices
  - Must enable AirDrop on the receiving device
- Create and configure **UIActivityViewController**
- Attach your content to be shared
  - Supports **NSString** and **UIImage**
  - Optionally use **UIActivityItemProvider** or **UIActivityItemSource**
- To accept shared content, register a custom UTI or URL scheme



AirDrop Example

```
let activityViewController = UIActivityViewController(activityItems: [image],applicationActivities: [])
presentViewController(vc, animated: true, completion: nil)
```

```
func application(_ app: UIApplication, openURL url: NSURL, options options: [String : AnyObject]) -> Bool
```

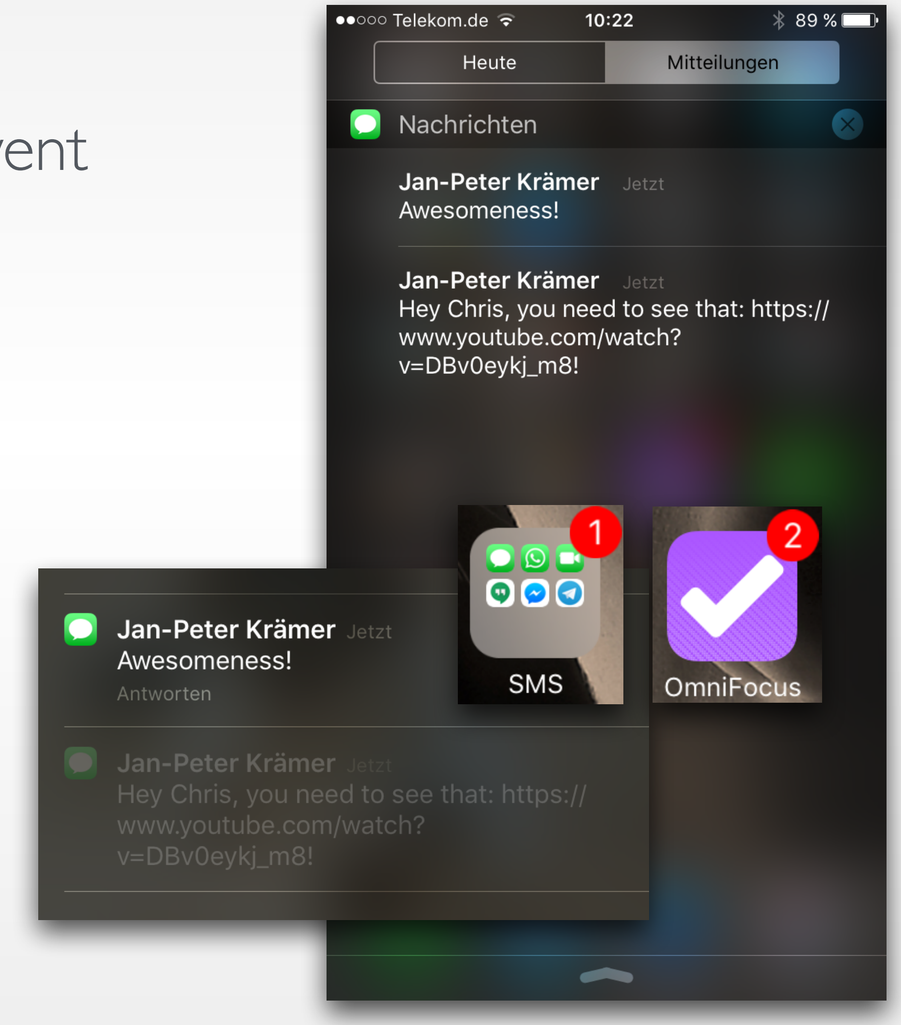


Documentation

- UIActivityViewController Class Reference
- AirDrop Example Code

Local and Push Notifications

- Notify the user (and your app) about an external event
- Notifications are visualized in multiple ways:
  - Alert
  - Sound
  - Notification Center
  - Badge with a number
- Not used to transmit data!



Local Notifications

- Schedule a local notification
  - Create and configure UILocalNotification
  - Tell UIApplication to schedule the local notification
- Receive a local notification
  - Implement UIAppDelegate
  - Application is running:
    application:didReceiveLocalNotification:
  - Application is not running:
    application:didFinishLaunchingWithOptions:

Scheduling Local Notifications Example

```
let notification = UILocalNotification()
notification.fireDate = NSDate(timeIntervalSinceNow: 10.0)
notification.alertBody = "Answer Chris"
notification.soundName = UILocalNotificationDefaultSoundName
notification.applicationIconBadgeNumber = 1
notification.userInfo = self.customInfo;

UIApplication.sharedApplication().scheduleLocalNotifications(notification)
```

Receiving Local Notifications Example

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject : AnyObject]?) -> Bool {

    let notification = launchOptions![UIApplicationLaunchOptionsLocalNotificationKey]
    self.handleNotification()

}
```

```
func application(application: UIApplication, didReceiveLocalNotification notification:
UILocalNotification) {

    self.handleNotification()
}
```

Remote Notifications

- Register for Remote Notifications
  - Tell UIApplication to register for remote notifications
  - Implement UIApplicationDelegate and send device token to your server
- Receive a Remote Notifications
  - Similar to local notifications (replace Local with Remote)
- Send a Remote Notification
  - Implement your own server
  - Request certificate via the Apple iOS Developer Portal

Register for Remote Notifications Example

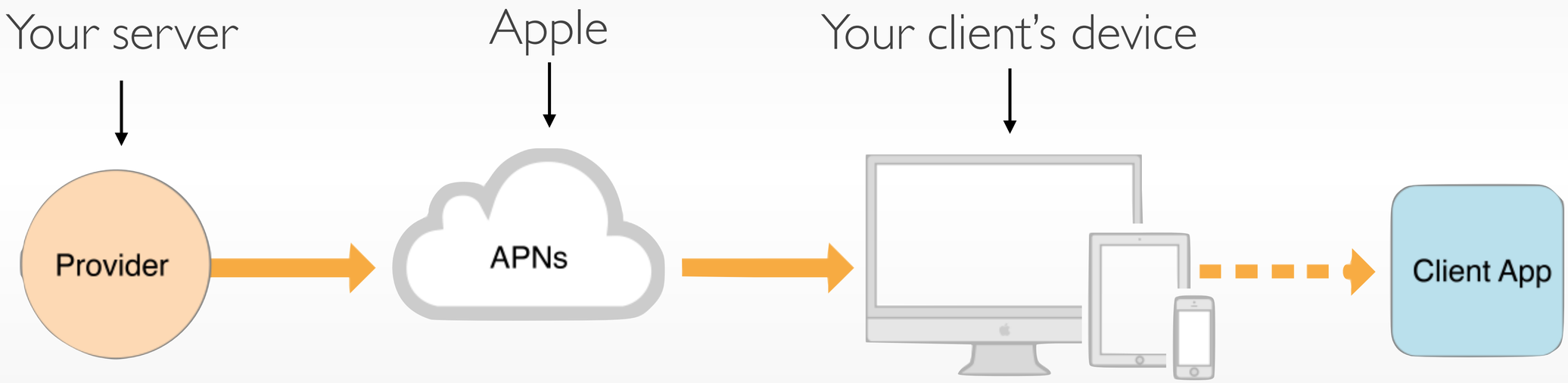
```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject
: AnyObject]?) -> Bool {

    let settings = UIUserNotificationSettings(forTypes: [.Alert, .Badge, .Sound], categories: nil)
    UIApplication.sharedApplication().registerUserNotificationSettings(settings)
    UIApplication.sharedApplication().registerForRemoteNotifications()
    …
}


func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    //send the device token to your server
}
```

Receive Remote Notifications Example

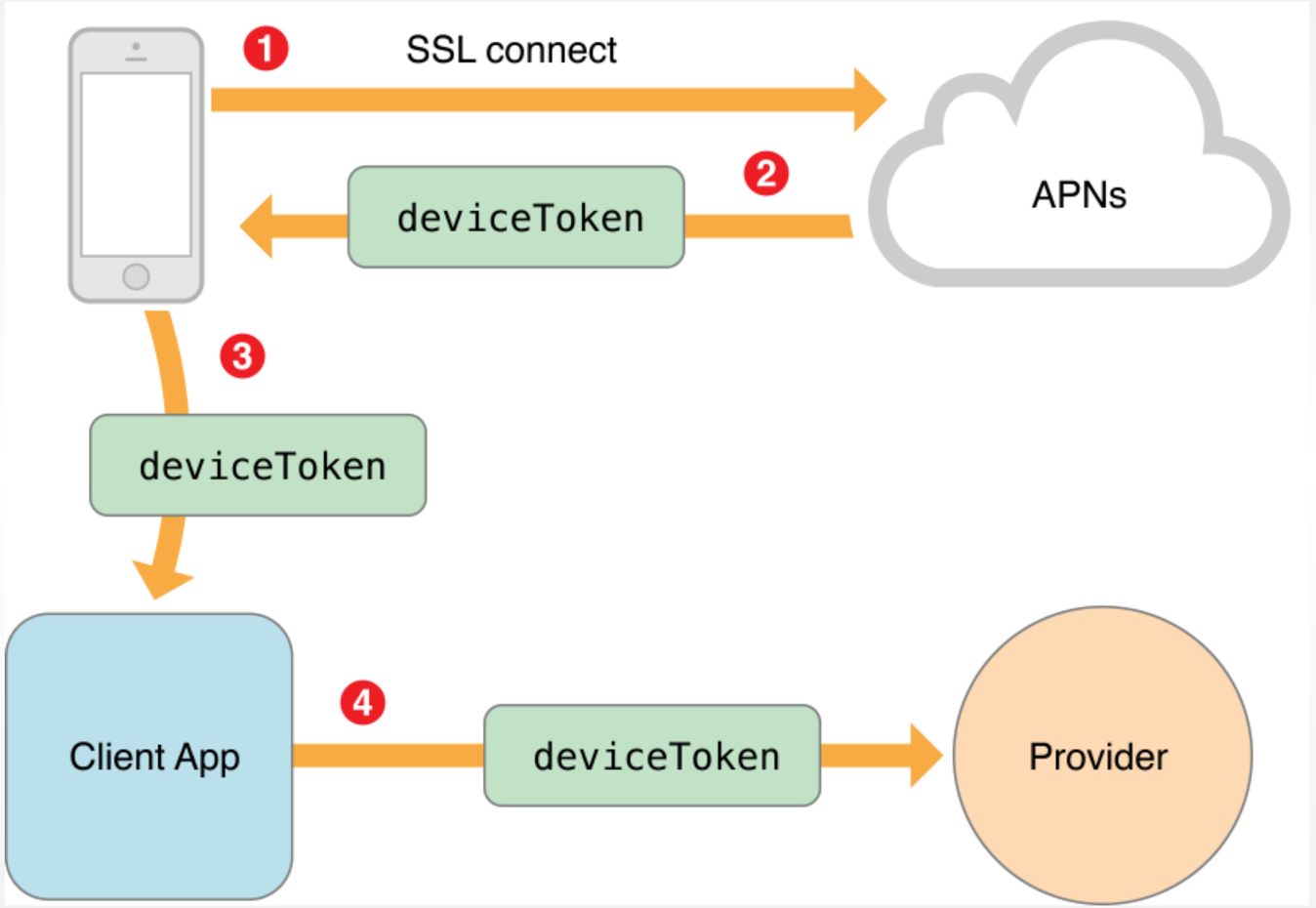
```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject : AnyObject]?) -> Bool {

    let notification = launchOptions![UIApplicationLaunchOptionsRemoteNotificationKey]
    self.handleNotification()

}
```

```
func application(application: UIApplication, didReceiveLocalNotification notification:
UILocalNotification) {

        self.handleNotification()

}
```

Sending Remote Notifications



Device Token



Remote Notification Payload

- JSON (max. 256 bytes)
- **aps** defines notification type
  - **alert message**, **badge count**, **sound**
- Additional keys can be provided as needed
- Payload is available via the **userInfo** dictionary

```
{
    "aps" : {
        "alert" : "You got your emails.",
        "badge" : 9,
        "sound" : "bingbong.aiff"
    },
    "acme1" : "bar",
    "acme2" : 42
}
```

Implementing the Server

- Many packages available
- NodeJS: https://github.com/argon/node-apn
- Ruby: https://github.com/jpoz/APNS
- Python: https://github.com/djacobs/PyAPNs
- PHP: https://code.google.com/p/apns-php
- …

Documentation

- Local and Push Notifications Overview
- Local and Remote Notification Programming Guide

URL Loading System

- Download content from a web server
  - Supports background download
- Communicate with a web service

URL Session

- Access content via HTTP
- Session Types:
  - Default: store content on the disk
  - Ephemeral: store content to memory
  - Background: process task in the background (when app is closed)
- Session Tasks:
  - Data tasks: exchange NSData objects with a web server (not available for background session type)
  - Download/Upload tasks: download or upload large files

Data Task Example

```
func httpGet(imageURL: NSURL!, callback: (UIImage?, String?) -> Void) {
    //create the data task to download
    let session = NSURLSession.sharedSession()
    let task = session.dataTaskWithURL(imageURL){

        (data, response, error) -> Void in
        if error != nil {
            callback(nil, error!.localizedDescription)
        } else {
           let result = UIImage(data: data!)
            callback(result! as UIImage, nil)
        }

    }
    task.resume()
}
```

Background Download Task Example

```
func createSession() -> NSURLSession {
    let identifier = "url_background_session"
    let config = NSURLSessionConfiguration.backgroundSessionConfigurationWithIdentifier(identifier)
    let backgroundSession = NSURLSession(configuration: config, delegate: self.delegate, delegateQueue: nil)
    return backgroundSession
}

func start() {
    let backgroundSession = self.createSession()
    backgroundSession.downloadTaskWithURL(imageURL).resume()
}
func application(application: UIApplication, handleEventsForBackgroundURLSession identifier: String,
completionHandler: () -> Void) {
    self.createSession()
    self.completionHandler = completionHandler;
}

func URLSession(session: NSURLSession, downloadTask:NSURLSessionDownloadTask, didFinishDownloadingToURL
location: NSURL) {
    self.completionHandler();
}
```

Documentation

- URL Loading System Programming Guide

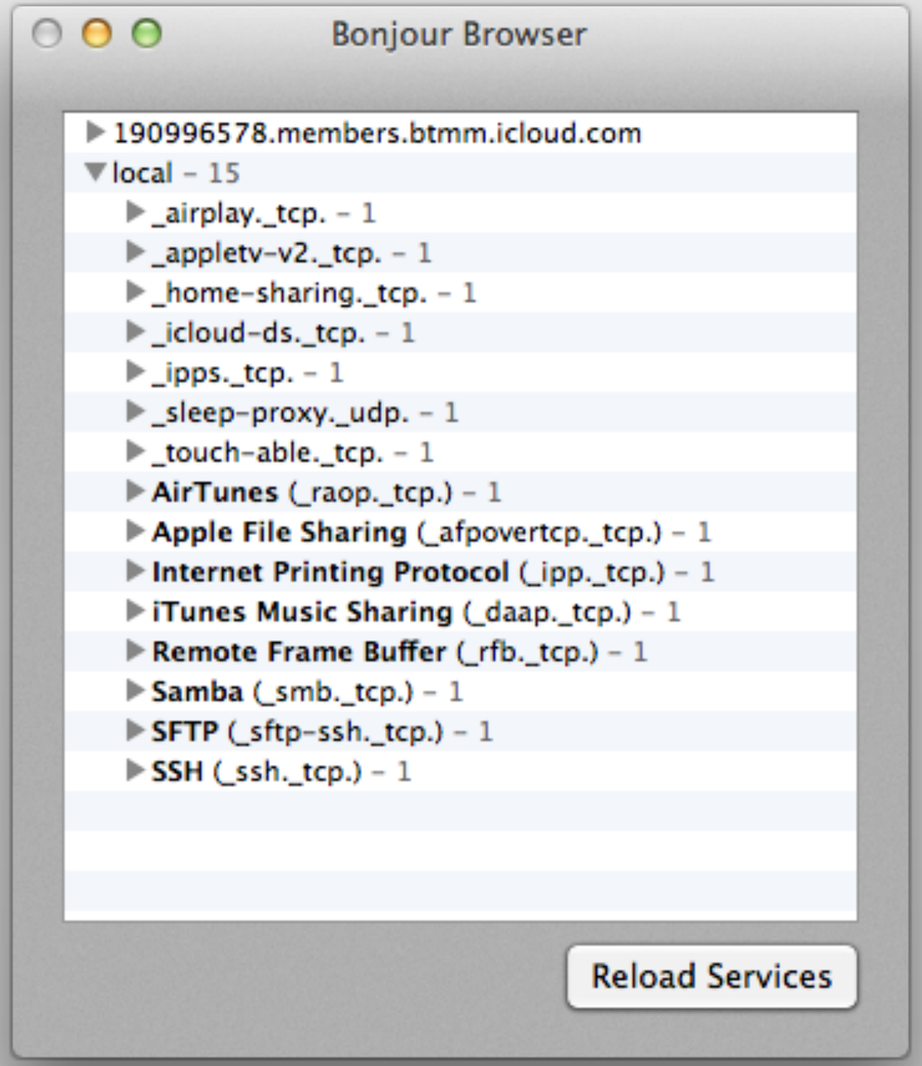
Bonjour

- Service discovery framework
- Does not transfer data
- Also available for Unix/Linux & Windows
- Based on multicast DNS (mDNS)
  - Every client stores own DNS table
  - Service lookup over DNS request broadcast

Bonjour Service

- Publish service using NSNetService
  - Unique Service name
  - Service type and transport layer (“_http._tcp.”)
  - Registration domain (“local.”)
  - Port
- Discover services using NSNetServiceBrowser

Bonjour Browser



Bonjour Example

```
//setup and start net service
let netService = NSNetService(domain: "local", type: "_myservice._tcp", name: "MyService", port: 8000)
netService.publish()
//setup and start net service browser
let netServiceBrowser = NSNetServiceBrowser()
netServiceBrowser.delegate = self
netServiceBrowser.searchForServicesOfType("_myservice._tcp", inDomain: "local")

func netServiceBrowser(browser: NSNetServiceBrowser, didFindService service: NSNetService, moreComing:
Bool) {
        //handle discovered service
}

func netServiceBrowser(browser: NSNetServiceBrowser, didRemoveService service: NSNetService,
moreComing: Bool) {
        // handle the removed service
}
```

Documentation

- Bonjour for Developers
- Bonjour Overview
- NSNetServices Programming Guide
- CocoaEcho Example

Socket Networking

- Sockets are used to establish a connection between devices
  - BSD sockets
  - CFNetwork
- Streams are used to communicate through sockets
  - CFStream
  - NSStream

Socket Networking Checklist

- Include CFNetwork Framework
- Create listening socket on server
- Create read and write streams from client to server
- Create read and write streams from server to client
- Send streaming data and respond to stream events

Socket Networking

```
int socket_connect(const char *host,int port,int timeout){
    struct sockaddr_in sa;
    struct hostent *hp;
    int sockfd = -1;
    hp = gethostbyname(host);
    bcopy((char *)hp->h_addr, (char *)&sa.sin_addr, hp->h_length);
    sa.sin_family = hp->h_addrtype;
    sa.sin_port = htons(port);
    sockfd = socket(hp->h_addrtype, SOCK_STREAM, 0);
    socket_set_block(sockfd,0);
    connect(sockfd, (struct sockaddr *)&sa, sizeof(sa));
    ...
}
```

Socket Networking
CFSocket

```
@asmname("socket_connect") func socket_connect(host:UnsafePointer<Int8>,port:Int32,timeout:Int32) -> Int32
```

```
//create a socket yourself
let socket = CFSocketCreate(kCFAllocatorDefault, 0, SOCK_STREAM, 0, 0, callBack, context)

//attach the socket to your run loop
let source = CFSocketCreateRunLoopSource(kCFAllocatorDefault, socket, 0)
CFRunLoopAddSource(CFRunLoopGetCurrent(), source, kCFRunLoopCommonModes)
source.release()
```

Stream Networking Example

Connection

```
//Create the connection with CFStreamRef
var readStream:  Unmanaged<CFReadStream>?
var writeStream: Unmanaged<CFWriteStream>?
CFStreamCreatePairWithSocketToHost(nil, self.serverAddress, self.serverPort, &readStream,
&writeStream)

//Proceed with NSStream objects
var inputStream: NSInputStream!
var outputStream: NSOutputStream!

self.inputStream = readStream!.takeRetainedValue()
self.outputStream = writeStream!.takeRetainedValue()
self.inputStream.delegate = self
self.outputStream.delegate = self

self.inputStream.scheduleInRunLoop(NSRunLoop.currentRunLoop(), forMode: NSDefaultRunLoopMode)
self.outputStream.scheduleInRunLoop(NSRunLoop.currentRunLoop(), forMode: NSDefaultRunLoopMode)

self.inputStream.open()
self.outputStream.open()
```

Stream Networking Example

Handle Stream Events

```
func stream(stream: NSStream, handleEvent eventCode: NSStreamEvent) {
    switch eventCode {
        case NSStreamEvent.HasBytesAvailable :
        var buffer = u_char(1024)
        var length = 0
        let inputStream = stream as! NSInputStream
        length = inputStream.read(&buffer, maxLength: length)
        if(length > 0) {
            data.appendBytes(&buffer, length: length)
        }
    }
}
```

Data Chunking

- Transferred data might be bigger than the stream buffer
- Data must be sent and arrives in chunks
- Chunking strategies:
  - Transmit a special character **(\0)** at the end of the data
  - Transmit length of the data before the data

Documentation

- Guide to Network Programming
- Stream Programming Guide
- CFSocket Reference