Desarrollo de Apps para iOS
Acceso a Servicios WEB

IWEB 2015-2016

Santiago Pavón

ver: 2015.11.25

Índice

- Soporte disponible para acceder a servicios Web.
- ATS: App Transport Security.
- Descarga sencilla de datos con peticiones HTTP GET.
  - NSData(contentsOfURL:)
- Codificación y Escapado.
  - NSData, URL Legales, Base-64
- Serialización JSON.
- Manejo de peticiones HTTP con NSURLSession.
  - Bajada y subida de datos, ficheros.
  - Métodos GET, PUT, DELETE, POST, …
  - Cabeceras HTTP.
  - Otros: Seguridad, Caches, …

¿Qué Soporte Tenemos?

- Disponemos de clases para:
  - Manejar URLs, Peticiones y Respuestas HTTP
    - **`NSURL`**, **`NSURLRequest`**, **`NSHTTPURLResponse`**
    - **`NSData(contentsOfURL:)`**
    - **`NSURLSession`** y clases relacionadas.
    - …
  - Codificaciones
    - Escapado de URL con códigos %, conversión base64, codificación y decodificación a NSData, . . .
  - Serialización de datos a formato JSON, e inversa:
    - La clase **`NSJSONSerialization`** proporciona métodos para serializar y des-serializar JSON.
  - XML:
    - Xcode no ofrece soporte para construir documentos XML.
    - Para parsear documentos XML disponemos de la clase **`NSXMLParser`**.
  - . . .

ATS

App Transport Security

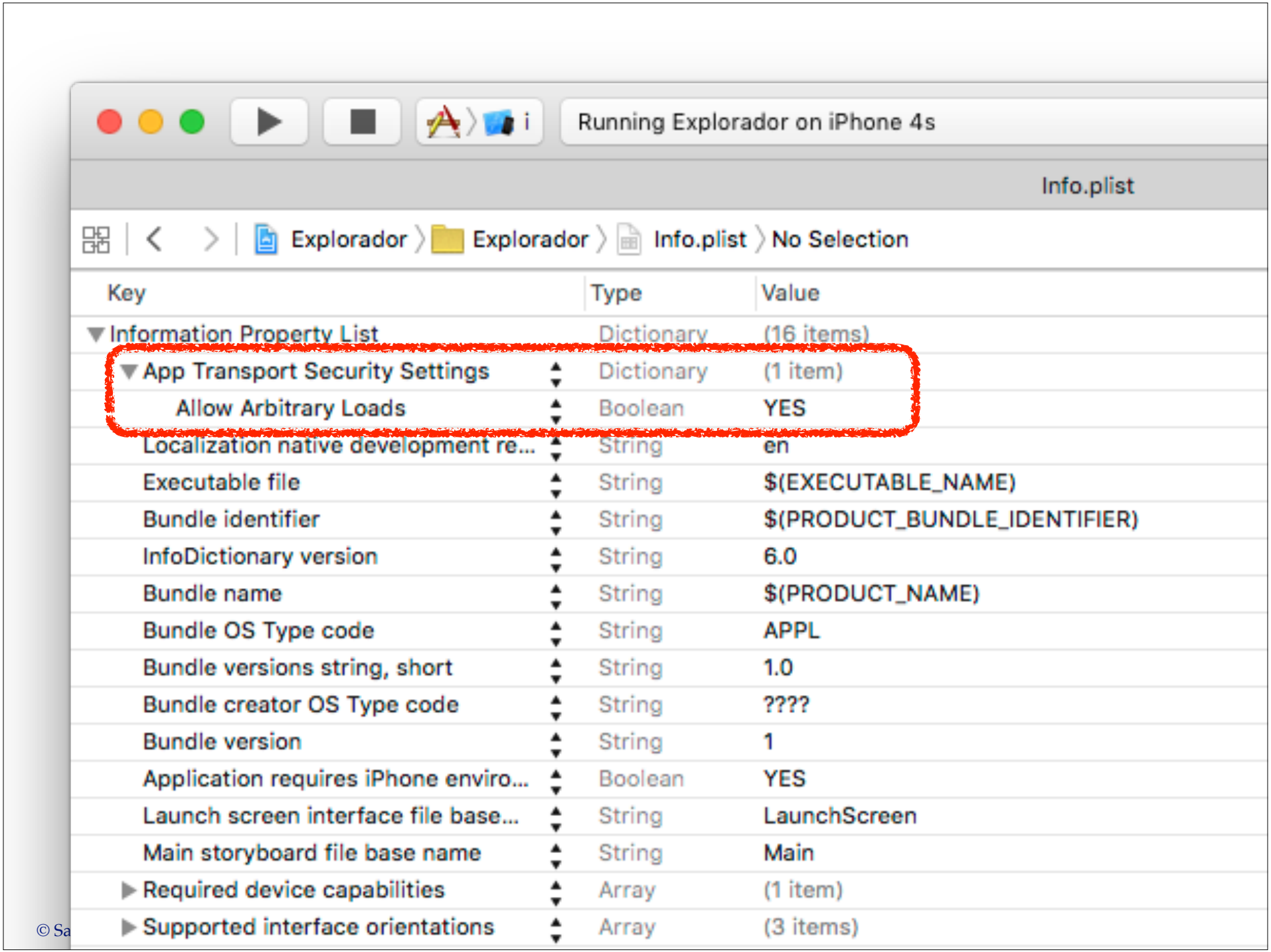
¿Qué es ATS?

- Es una característica nueva introducida por App en iOS 9 *(y OS X El Capitan)* para hacer seguras (privacidad e integridad) las comunicaciones con servicios web usando TLS.
  - Refuerza el nivel de seguridad usado por defecto, fuerza el uso de HTTPS por defecto, sigue las mejores prácticas para uso de versiones de TLS, tipo de cifrado, tamaño de claves, uso de certificados, …

Excepciones ATS

- En el fichero **Info.plist** pueden añadirse excepciones para modificar el funcionamiento de ATS.
- Bajo la clave **`NSAppTransportSecurity`** pueden añadirse:
  - excepciones para las conexiones a cualquier dominio.
  - excepciones para las conexiones a un dominio dado.
- **Ejemplo**: Para deshabilitar ATS para todos los dominios:

```
▼ NSAppTransportSecurity
      NSAllowsArbitraryLoads = YES
```


Running Explorador on iPhone 4s
Info.plist
Explorador
Explorador
Info.plist
No Selection
Key
Type
Value
Information Property List
Dictionary
(16 items)
App Transport Security Settings
Dictionary
(1 item)
Allow Arbitrary Loads
Boolean
YES
Localization native development re...
String
en
Executable file
String
$(EXECUTABLE_NAME)
Bundle identifier
String
$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version
String
6.0
Bundle name
String
$(PRODUCT_NAME)
Bundle OS Type code
String
APPL
Bundle versions string, short
String
1.0
Bundle creator OS Type code
String
????
Bundle version
String
1
Application requires iPhone enviro...
Boolean
YES
Launch screen interface file base...
String
LaunchScreen
Main storyboard file base name
String
Main
Required device capabilities
Array
(1 item)
Supported interface orientations
Array
(3 items)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
      <key>NSAppTransportSecurity</key>
      <dict>
        <key>NSAllowsArbitraryLoads</key>
        <true/>
      </dict>
      <key>CFBundleDevelopmentRegion</key>
      <string>en</string>
      <key>CFBundleExecutable</key>
      <string>$(EXECUTABLE_NAME)</string>
      <key>CFBundleIdentifier</key>
      <string>$(PRODUCT_BUNDLE_IDENTIFIER)</string>
      <key>CFBundleInfoDictionaryVersion</key>
. . .
```

- **Ejemplo**: Para permitir las conexiones HTTP al dominio indicado:
  - `NSAppTransportSecurity`
    - `NSExceptionDomain`
      - `sitio.ejemplo.es`
        - `NSExceptionAllowsInsecureHTTPLoads = YES`

- **Ejemplo**: Para permitir las conexiones HTTP a cualquier dominio, excepto a uno dado:
  - `NSAppTransportSecurity`
    - `NSAllowsArbitraryLoads = YES`
    - `NSExceptionDomain`
      - `sitio.ejemplo.es`
        - `NSExceptionAllowsInsecureHTTPLoads = NO`

- **Ejemplo**: Para permitir las conexiones HTTP a un dominio dado y a todos sus subdominios; y para otro dominio no requerir Forward Secrecy, indicar la versión mínima de TLS a usar :

```
▼ NSAppTransportSecurity
  ▼ NSExceptionDomain
    ▼ ejemplo.es
        NSIncludesSubdomains = YES
        NSExceptionAllowsInsecureHTTPLoads = YES
    ▼ otro.demo.com
        NSExceptionRequiresForwardSecrecy = NO
        NSExceptionMinimumTLSVersion = TLSv1.1
```

- Consultar la documentación **Information Property List Key Reference** para ver todos los detalles de configuración.

Descargas Sencillas de Datos con Peticiones HTTP GET

`NSData(contentsOfURL:)`

NSData(contentsOfURL:)

- Para crear un NSData con los datos del sitio especificado en una URL puede usarse:

```
let data: NSData? = NSData(contentsOfURL: url)
```

  - Si no puede obtener los datos, devuelve **nil**.
  - Si es necesario conocer las razones de los posibles fallos, usar:

```
do {
    let data = try NSData(contentsOfURL:url, options:opts)
    // usar data
} catch let error as NSError {
    print(error.localizedDescription)
}
```

- Es una llamada síncrona.
  - Se bloquea hasta que se han descargado todos los datos.
  - Para evitar bloqueos:
    - Usar **GCD** (u otros) para realizar la descarga en otro thread.
    - Usar las tareas proporcionadas por **NSURLSession**.

```swift
let imgUrl = "http://www.etsit.upm.es/images/portada/logoetsitupm.png"

// No hay que escapar caracteres conflictivos de la URL

// Construir un NSURL
if let url = NSURL(string: imgUrl) {

    // Mostrar indicador de actividad de red (no funciona por bloqueo del UI)
    UIApplication.sharedApplication().networkActivityIndicatorVisible = true
    defer {
       // Ocultar indicador de actividad de red
       UIApplication.sharedApplication().networkActivityIndicatorVisible = false
    }

   // Bajar los datos del sitio Web
   if let data = NSData(contentsOfURL: url),

      // Construir una imagen con los datos bajados
      let img = UIImage(data: data) {

          // Actualizar el GUI
          imageView.image = img
   }
}
```

Espera hasta que se han descargado los datos

```
let imgUrl = "http://www.etsit.upm.es/images/portada/logoetsitupm.png"

// Construir un NSURL
if let url = NSURL(string: imgUrl) {
   // Mostrar indicador de actividad de red
   UIApplication.sharedApplication().networkActivityIndicatorVisible = true

   // Envio la tarea a un thread
   let queue = dispatch_queue_create("Download Queue", DISPATCH_QUEUE_SERIAL)
   dispatch_async(queue, {

      // Bajar los datos del sitio Web
      if let data = NSData(contentsOfURL: url),
         // Construir una imagen con los datos bajados
         let img = UIImage(data: data) {

            // El GUI se actualiza en el Main Thread
            dispatch_async(dispatch_get_main_queue(), {
                  self.imageView.image = img
            })
      }

      // El GUI se actualiza en el Main Thread
      dispatch_async(dispatch_get_main_queue(), {
         // Ocultar indicador de actividad de red
         UIApplication.sharedApplication().networkActivityIndicatorVisible = false
      })
   })
}
```

Uso GCD y envío la tarea a un thread

El GUI solo se actualiza en el Main Thread

Codificación y Escapado

Codificación y Escapado

- Por motivos de interoperabilidad, al manejar URLs y el protocolo HTTP, debemos **codificar** y **escapar** algunos caracteres.
  - Las RFCs 1808, 1738, 2732 y 3986 especifica cómo son las URLs.
    - Hay una serie de reglas para su codificación:
      - Solo pueden usarse unos cuantos caracteres ASCII (letras, números y algunos signos) en una URL, y los caracteres conflictivos debe escaparse.
      - Los caracteres (`$&+,/:;=?@`) que tienen un significado especial en la sintaxis de la URL deben escaparse cuando se usan con otro significado.
      - …
    - Los caracteres de la URL se codifican en UTF-8, y los bytes que no son letras o números ASCII, o tienen un significado conflictivo, se sustituyen por un % seguido de dos dígitos hexadecimales (su código ASCII).
      - Ejemplo: **`El Camión`** -> **`El%20Cami%C3%B3n`**
  - Los valores de algunas cabeceras de las peticiones y respuestas HTTP, también se codifican en UTF-8, Base-64,…

- En Xcode tenemos algunas clases y métodos para realizar estas tareas de codificación y escapado.

String <—> NSData

- Ya sabemos que:
  - El valor de un String es una secuencia de caracteres Unicode.
  - El valor de un NSData es una secuencia de bytes.
    - La secuencia de bytes puede ser un string, una imagen, un audio, etc..
- Para convertir un String en una secuencia de bytes codificada en UTF-8:

```
let str = "El Camión"

let data: NSData? = str.dataUsingEncoding(NSUTF8StringEncoding)
```

- Para reconstruir un String desde un NSData codificado en UTF-8:

```
let str2: String? = NSString(data: data!,
                             encoding: NSUTF8StringEncoding)
                    as? String
```

- Este tipo de conversiones también puede hacerse con otros tipos de datos.

URL Legales Escapando Caracteres

- **Hasta iOS 9:**
  - Obtener un String que representa una URL legal sustituyendo los caracteres inválidos por secuencias de escape %##.

```
func stringByAddingPercentEscapesUsingEncoding(
                    encoding: NSStringEncoding)-> String?
```

  - Reconstruir el string original, sustituyendo las secuencias de escape (%##) por los caracteres que representan.

```
func stringByReplacingPercentEscapesUsingEncoding(
                    encoding: NSStringEncoding) -> String?
```

  - **Estos métodos están deprecados desde iOS 9.**
    - Las reglas de escapado para cada parte de un URL no son iguales, y hay casos en los que estos métodos no funcionan bien.

```
let url = "http://localhost:3000/demo?q=El Camión"

let url2 = url.stringByAddingPercentEscapesUsingEncoding(
                                                NSUTF8StringEncoding)!

println(url2) // http://localhost:3000/demo?q=El%20Cami%C3%B3n


let url3 = url2.stringByReplacingPercentEscapesUsingEncoding(
                                                NSUTF8StringEncoding)!

println(url3) // http://localhost:3000/demo?q=El Camión
```

- **Desde iOS 9:**
  - Obtener un String sustituyendo los caracteres inválidos por secuencias de escape %##.
    - El parámetro indica cuáles son los caracteres válidos que no hay que sustituir.
      - Los caracteres válidos para cada parte de un URL son diferentes.

```
func stringByAddingPercentEncodingWithAllowedCharacters(
        allowedCharacters: NSCharacterSet)-> String?
```

      - Los posibles valores de NSCharacterSet pueden obtenerse llamando a las siguientes funciones estáticas:

        `URLFragmentAllowedCharacterSet()`,
        `URLHostAllowedCharacterSet()`,
        `URLPasswordAllowedCharacterSet()`,
        `URLPathAllowedCharacterSet()`,
        `URLQueryAllowedCharacterSet()` y
        `URLUserAllowedCharacterSet()`.

  - Reconstruir el string original, sustituyendo las secuencias de escape (%##) por los caracteres que representan.

```
var stringByRemovingPercentEncoding: String? { get }
```

```swift
let base = "http://es.pokemon.wikia.com"

// Hay que escapar el path:
let path = "wiki/Nidoran ♂"

// Añadir el path al URL escapando caracteres conflictivos
if let escapedPath = path.stringByAddingPercentEncodingWithAllowedCharacters(
                                                    .URLPathAllowedCharacterSet()) {

    let escapedUrl = "\(base)/\(escapedPath)"

    print(escapedUrl) // http://es.pokemon.wikia.com/wiki/Nidoran%20%E2%99%82

    let originalUrl = escapedUrl.stringByRemovingPercentEncoding!


    print(originalUrl) // http://es.pokemon.wikia.com/wiki/Nidoran ♂

}
```

Base 64

- **Codificar** en Base-64:
  - Crear un String codificado en Base-64 desde un NSData:

```
func base64EncodedStringWithOptions(
                _ options: NSDataBase64EncodingOptions) -> String
```

  - Crear un NSData codificado en Base-64 y UTF-8 desde un NSData:

```
func base64EncodedDataWithOptions(
                _ options: NSDataBase64EncodingOptions) -> NSData
```

- **Decodificar** en Base-64:
  - Construir un NSData desde un NSData codificado en Base-64 y UTF-8:

```
init?(base64EncodedData base64Data: NSData,
                options options: NSDataBase64DecodingOptions)
```

  - Construir un NSData desde un String codificado en Base-64:

```
init?(base64EncodedString base64String: String,
                  options options: NSDataBase64DecodingOptions)
```

- Consultar la documentación para ver las opciones de codificación y descodificación en Base-64.

```swift
// NSData con los bytes de una foto
let f = NSBundle.mainBundle().pathForResource("perro", ofType: "jpg")
let data = NSData(contentsOfFile: f!)!

// String codificado en Base-64 con la foto.
let str64: String = data.base64EncodedStringWithOptions([])



// Ese String se va de vacaciones y cuando vuelve:



// Creo otro NSData con los bytes del String en Base-64.
let data2 = NSData(base64EncodedString: str64, options: [])

// Presento la imagen
let img = UIImage(data: data2!)
imageView.image = img!
```

Serialización de JSON

JSON

- La clase **`NSJSONSerialization`** permite:
  - Convertir JSON en objetos Foundation.
  - Convertir objetos Foundation en JSON.
- Los objetos Foundation deben cumplir estas condiciones:
  - El objeto raíz es un NSArray o un NSDictionary.
  - Todos los objetos son NSString, NSNumber, NSArray, NSDictionary o NSNull.
  - Las claves de los diccionarios son NSString.
  - Los números no pueden se NaN o infinity.
  - y pueden existir más condiciones.

    - O los tipos equivalentes de Swift: Array, Dictionary, String, Int, Float, Double

- Para saber si un objeto puede convertirse en JSON puede usarse el método:

```
class func isValidJSONObject(_ obj: AnyObject) -> Bool
```

- Para crear un objeto (diccionario o array) a partir de un dato JSON (**NSData**) usar el método:

```
class func JSONObjectWithData(_ data: NSData,
                              options opt: NSJSONReadingOptions)
                    throws -> AnyObject
```

  - Devuelve **nil** si encuentra algún error.
  - El dato JSON debe estar codificado en UTF-8, UTF-16LE, UTF-16BE, UTF-32LE o UTF-32BE.

- Para crear un dato JSON (NSData) a partir de un objeto Foundation, usar el método:

```
class func dataWithJSONObject(_ obj: AnyObject,
                              options opt: NSJSONWritingOptions)
                    throws -> NSData
```

  - Devuelve **nil** si encuentra algún error.
  - El dato JSON devuelto esta codificado en UTF-8.

```swift
// Un diccionario:
let person = ["nombre": "Juan", "edad": 28]

// Crear JSON
do {
    let data = try NSJSONSerialization.dataWithJSONObject(person,
                                                           options: []) {

    // Ver el buffer de bytes:
    println(data)  // <7b226e6f 6d627265 223a224a … 6164223a 32387d>

    // Ver el NSData como un String
    let str = NSString(data: data, encoding: NSUTF8StringEncoding)
    println(str!) // {"nombre":"Juan","edad":28}

    // Reconstruir el objeto
    if let person2 = try NSJSONSerialization.JSONObjectWithData(data,
                                options: []) as? [String:AnyObject] {

        // Ver el diccionario
        println(person2) // [nombre: Juan, edad: 28]
    }
} catch let error {
    print(error)
}
```

NSURLSession

NSURLSession

- Se usa para realizar transferencia de datos usando **https**.
  - También **http**, **ftp**, **file**, **data**.
- Las apps pueden crear varias sesiones
  - Cada sesión coordinará varias tareas de transferencia de datos relacionadas.
- En cada sesión se realizará una configuración que se aplicará sobre todas las tareas de la sesión.
  - Existen varios tipos de configuraciones para las sesiones:
    - **shared**: Usa una configuración por defecto compartida. No se pueden configurar delegados, ni objetos de configuración. Para peticiones sencillas. No pueden obtenerse los datos incrementalmente según se reciben, pocas opciones para manejar autenticación, no se pueden realizar tareas en background, …
    - **default**: Usa un objeto de configuración por defecto que puede reconfigurarse, puede usar delegados para realizar descargas incrementales. Usa almacenamiento persistente para caches, credenciales y cookies.
    - **ephemeral**: Similar a default pero no almacenan nada de forma persistente (sesiones privadas).
    - **background**: Usa una configuración que permite que las transferencias de datos HTTP y HTTPS continúen aunque se suspenda, termine o se muera la app.

- Pueden crearse tres tipos de tareas:
  - **Data Task**: Descargar datos en memoria.
  - **Download Task**: Descargar ficheros al disco (al sistema de ficheros).
  - **Upload Task**: Subir ficheros y recibir los datos de la respuesta en memoria.
- Threads:
  - El API de NSURLSession es **Thread-Safe**.
  - Las tareas se ejecutan en **Threads** separados para no bloquear el Main Thread.
    - El thread donde se ejecuta una depende de como se cree.
- Las tareas pueden **cancelar**, **detener**, **pausar** y **reanudar**.
- Hay soporte para tratar de manera personalizada la autenticación y la validación de certificados (TLS).
- Se puede programar usando completion block (closures) o delegados.
  - **Completion block**: se ejecutan cuando ha terminado la transferencia.
  - **Delegados**: se les informa sobre el progreso y finalización de las tareas.

- Este es un tema muy extenso.
  - Se recomienda consultar la guía **URL Session Programming Guide**.

Clases

- **NSURLSessionConfiguration**
  - Crear un objeto para configurar inicialmente una sesión.
- **NSURLSession**
  - Crear los objetos que representan una sesión.
- **NSURLSessionTask**
  - Clase base de los distintos tipos de tareas.
    - **NSURLSessionDataTask**
      - Para descargar el contenido de una URL en un NSData.
        - **NSURLSessionUploadTask**
          - Para subir un fichero y recibir los datos de la respuesta en un NSData.
    - **NSURLSessionDownloadTask**
      - Para descargar el contenido de una URL en un fichero temporal.

➡ Otras clases que también se usan son **NSURL**, **NSURLRequest**, **NSURLResponse**, **NSHTTPURLResponse**, **NSCachedURLResponse**, …

Protocolos

- Hay cuatro protocolos que pueden usarse para tener un control más fino sobre las sesiones y las tareas.
  - **NSURLSessionDelegate**
    - Define métodos para manejar eventos a nivel de sesión.
  - **NSURLSessionTaskDelegate**
    - Define métodos para manejar eventos a nivel de tarea.
      - Se definen los métodos que son comunes para todo tipo de tareas.
  - **NSURLSessionDataDelegate**
    - Define métodos para manejar eventos a nivel de tarea.
      - Se definen los métodos que son específicos de las tareas Data y Upload.
  - **NSURLSessionDownloadDelegate**
    - Define métodos para manejar eventos a nivel de tarea.
      - Se definen los métodos que son específicos de las tareas Download.

Gestión Personalizada Autenticación y TLS

- NSURLSession usa delegados para manejar de forma personalizada las peticiones de autenticación y para validar certificados en la negociación TLS.
- Existen métodos en los delegados para manejar los desafíos enviados por el servidor a nivel de tarea o de sesión.
  - `NSURLSessionTaskDelegate`
    `URLSession:task:didReceiveChallenge:completionHandler:`
  - `NSURLSessionDelegate`
    `URLSession:didReceiveChallenge:completionHandler:`
- Si estos métodos no se implementan, se intenta resolver la autenticación mirando la información proporcionada en la URL Request, o buscando passwords y certificado en el llavero de claves.
  - Finalmente si no se consiguen las credenciales o si se rechazan, las conexiones fallarán.

Ejemplo 1: Bajar una Imagen

- Descargar una imagen usando una **Shared Session** y un **Data Task** con un **Completion Handler**.
- Detalles de la implementación:
  - Creamos una sesión de tipo SharedSession.
    - Esta sesión es un **singleton** que usa una configuración por defecto.
      - Usa las caches, cookies y credenciales globales.
  - Creamos una tarea de tipo DataTask para bajar el contenido de una URL que apunta a una imagen.
    - La tarea que creamos usa un Completion Handler para actualizar el GUI.
      - Se ejecuta cuando ha terminado la descarga.
        - Como no estamos en el Main Thread, usamos GCD para actualizar el GUI enviando closures por la Main Queue.
  - La tarea inicialmente está suspendida. Hay que arrancar su ejecución.

```swift
// Crear sesion
let session = NSURLSession.sharedSession()

// Mostrar indicador de actividad de red
UIApplication.sharedApplication().networkActivityIndicatorVisible = true

// Construir un NSURL
let imgUrl = "http://www.etsit.upm.es/images/portada/logoetsitupm.png"
let url = NSURL(string: imgUrl)!

// Crear la Data Task
let task = session.dataTaskWithURL(url, completionHandler: { (data: NSData?,
                                        res: NSURLResponse?, error: NSError?) in
    if error == nil && (res as! NSHTTPURLResponse).statusCode == 200 {
      // Construir una imagen con los datos bajados
       if let img = UIImage(data: data!) {
         dispatch_async(dispatch_get_main_queue(), {
              self.imageView.image = img
          })
       } else { print("Error construyendo la imagen") }
    } else { print("Error descargando") }
    dispatch_async(dispatch_get_main_queue(), {
        // Ocultar indicador de actividad de red
        UIApplication.sharedApplication().networkActivityIndicatorVisible = false
    })
})
// Arrancar la tarea
task.resume()
```

Ejemplo 2 - Configurar una Sesión

```
// Siempre hay que crear primero la configuracion:
var config = NSURLSessionConfiguration.defaultSessionConfiguration()

// Restringir las operaciones de red a la WIFI:
config.allowsCellularAccess = false

// Configurar cabeceras HTTP:
// Prefiero español.
config.HTTPAdditionalHeaders = ["Accept-Language": "es-es"]

// Configurar opciones:
// Timeout para cada peticion y para el recurso completo.
// Limitar a una conexion con el servidor.
config.timeoutIntervalForRequest = 30.0
config.timeoutIntervalForResource = 60.0
config.HTTPMaximumConnectionsPerHost = 1

//----

// Crear la sesion con la configuracion anterior
let session = NSURLSession(configuration: config)

// Etc:  Crear tareas, . . .
```

Ejemplo 3 - Bajar una Imagen

- Descargar una imagen usando una **Default Session** y un **Download Task** con un **Completion Handler**.

- Detalles de la implementación:
  - Creamos una sesión usando la configuración por defecto.
  - Creamos una tarea de tipo DownloadTask para bajar a nuestro sistema de ficheros el contenido de una URL, que en este caso es una imagen.
    - La tarea creada usa un Completion Handler.
      - Se está usando con la sintaxis **Trailing Closure**.
        - Si el último parámetro de una función es una closure, se puede sacar fuera de la función.
      - La closure se ejecuta cuando ha terminado la descarga.
      - La closure toma como primer parámetro la URL donde se han descargado los datos en nuestro sistema de ficheros.
      - Como no estamos en el Main Thread, usamos GCD para actualizar el GUI enviando closures por la Main Queue.
      - Nota: Se están ignorando los errores.
  - La tarea inicialmente está suspendida. Hay que arrancar su ejecución.

```swift
// Crear la configuracion de la sesion:
let config = NSURLSessionConfiguration.defaultSessionConfiguration()
// Crear una session
let session = NSURLSession(configuration: config)

UIApplication.sharedApplication().networkActivityIndicatorVisible = true

// Construir un NSURL
let imgUrl = "http://www.etsit.upm.es/images/portada/logoetsitupm.png"
let url = NSURL(string: imgUrl)!

// crear un Download Task con un Completion handler
let task = session.downloadTaskWithURL(url) { (location: NSURL?,
                                        res: NSURLResponse?, error: NSError?) in
   if error == nil && (res as! NSHTTPURLResponse).statusCode == 200 {
        // location es la URL donde se ha descargado la imagen. Es un NSData.
        if let data = NSData(contentsOfURL: location!) {
            if let img = UIImage(data: data) {
               dispatch_async(dispatch_get_main_queue(), { // Actualizar GUI
                    self.imageView.image = img
               })
    }   }   }
    // El GUI se actualiza en el Main Thread
    dispatch_async(dispatch_get_main_queue(), {
      UIApplication.sharedApplication().networkActivityIndicatorVisible = false
    })
}
task.resume() // Arrancar la tarea
```

Ejemplo 4 - Bajar una Imagen

- Descargar una imagen usando una **Default Session** y un **Download Task** con un **Download Delegate**.
- Detalles de la implementación:
  - El delegado atiende la finalización y el progreso de la descargas.
    - Hacer que el View Controller adopte este protocolo.
      - Implementamos los métodos encargados de esas labores.
  - Creamos una sesión usando la configuración por defecto y el Download Delegate creado.
  - Crear la Download Task.
  - La tarea inicialmente está suspendida. Hay que arrancar su ejecución.

```
class ViewController: UIViewController, NSURLSessionDownloadDelegate {

    // Termino la descarga
    func URLSession(session: NSURLSession,
         downloadTask: NSURLSessionDownloadTask,
         didFinishDownloadingToURL location: NSURL) {

        // location es el URL a los datos descargados.
        let img = UIImage(data: NSData(contentsOfURL: location)!)

        // Actualizar el GUI
        dispatch_async(dispatch_get_main_queue(), {
            self.imageView.image = img
        })
    }

    // Trazas de progreso
    func URLSession(session: NSURLSession,
         downloadTask: NSURLSessionDownloadTask,
         didWriteData bytesWritten: Int64,
         totalBytesWritten: Int64,
         totalBytesExpectedToWrite: Int64) {

        print("\(totalBytesWritten) / \(totalBytesExpectedToWrite)")
    }
```

```
// Construir un NSURL
let imgUrl = "http://www.etsit.upm.es/images/portada/logoetsitupm.png"
let url = NSURL(string: imgUrl)!

// Crear la configuracion de la sesion:
let config = NSURLSessionConfiguration.defaultSessionConfiguration()

// Crear la session
let session = NSURLSession(configuration: config,
                           delegate: self,
                           delegateQueue: nil)

// Crear Download Task con la URL a descargar
let task = session.downloadTaskWithURL(url)

// Arrancar la tarea
task.resume()
```

Ejemplo 5 - Subir una Imagen

- Subir una imagen usando una **Default Session** y un **Upload Task** con un **Completion Handler**.
- Detalles de la implementación:
  - Creamos una sesión usando la configuración por defecto.
  - Los datos a subir se indican con la URL del fichero a subir.
  - Creamos la petición HTTP con el método POST y la cabecera Content-Type.
  - Creamos una tarea de tipo UploadTask pasando como parámetros:
    - La petición HTTP.
    - La URL del fichero a subir
      - Los datos a subir se cogen de esta URL, ignorando el body de la petición HTTP.
    - Un Completion Handler.
      - Se está usando con la sintaxis **Trailing Closure**.
        - Si el último parámetro de una función es una closure, se puede sacar fuera de la función.
      - La closure se ejecuta cuando ha terminado la subida.
        - La closure recibe los datos descargados, que en este ejemplo ignoramos.
  - La tarea inicialmente está suspendida. Hay que arrancar su ejecución.

```
// Crear la configuracion de la sesion, y la sesion
let config = NSURLSessionConfiguration.defaultSessionConfiguration()
let session = NSURLSession(configuration: config)

UIApplication.sharedApplication().networkActivityIndicatorVisible = true

// URL del sitio de subida
let url = NSURL(string: "http://localhost/upload")!

// URL de la imagen a subir
let file: NSURL = NSBundle.mainBundle().URLForResource("perro", withExtension:"jpg")!

//--- La peticion HTTP:
let request = NSMutableURLRequest(URL: url)
request.HTTPMethod = "POST"
request.addValue("image/jpeg", forHTTPHeaderField: "Content-Type")

//--- La tarea para subir los datos:
// Nota: el tercer parametro es el completion Handler - usado como Trailing Closure.
let task = session.uploadTaskWithRequest(request, fromFile: file) {
    (data: NSData?, res: NSURLResponse?, error: NSError!?) in

    if error != nil { print(error!.localizedDescription)
    } else if (res as! NSHTTPURLResponse).statusCode != 201 {
        print(NSHTTPURLResponse.localizedStringForStatusCode(
                                            (res as NSHTTPURLResponse).statusCode))
    } else { print("Subido") }
    dispatch_async(dispatch_get_main_queue(), {
        UIApplication.sharedApplication().networkActivityIndicatorVisible = false
    })
}
task.resume() // Reanudar la tarea
```