Desarrollo de Apps para iOS
Acceso a Servicios WEB

IWEB 2019-2020

Santiago Pavón

ver: 2019.12.04

© Santiago Pavón - UPM-DIT

Índice

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

© Santiago Pavón - UPM-DIT

¿Qué Soporte Tenemos?

- Disponemos de clases para:
  - Manejar URLs, Peticiones y Respuestas HTTP
    - **`URL`**, **`URLRequest`**, **`HTTPURLResponse`**
    - **`Data(contentsOf URL:)`**
    - **`URLSession`** y clases relacionadas.
    - …
  - Codificaciones
    - Escapado de URL con códigos %, conversión base64, codificación y decodificación a Data, . . .
  - Serialización de datos a formato JSON, e inversa:
    - El protocolo **`Codable`**, la clase **`JSONSerialization`** se usan para serializar y deserializar JSON.
  - XML:
    - Xcode no ofrece soporte para construir documentos XML.
    - Para parsear documentos XML disponemos de la clase **`XMLParser`**.
  - . . .

© Santiago Pavón - UPM-DIT

ATS

App Transport Security

© Santiago Pavón - UPM-DIT

¿Qué es ATS?

- Característica 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, …

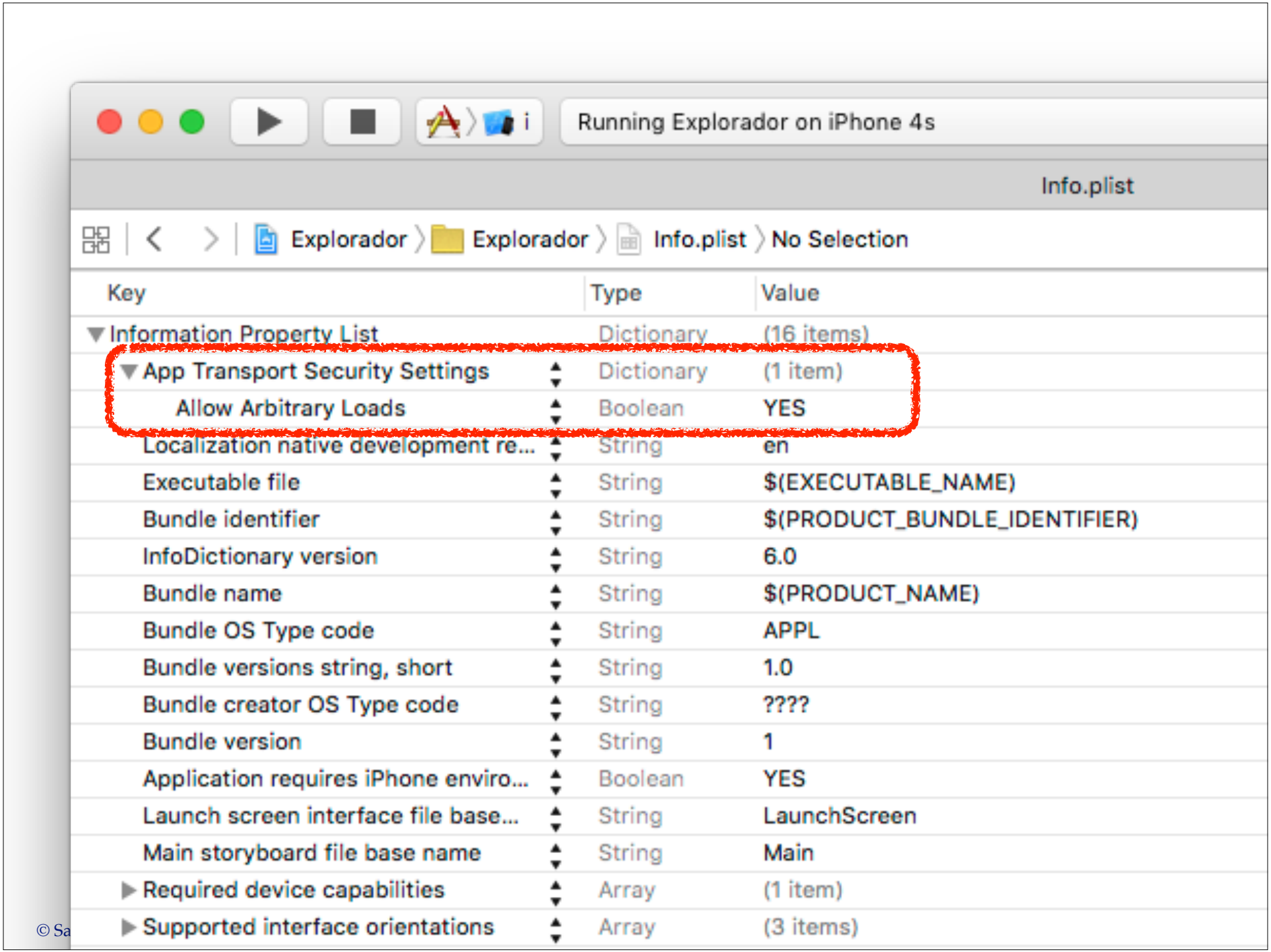
© Santiago Pavón - UPM-DIT

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
```

© Santiago Pavón - UPM-DIT


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)

© Sa

```
<?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>
. . .
```

© Santiago Pavón - UPM-DIT

- **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`

© Santiago Pavón - UPM-DIT

- **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.

© Santiago Pavón - UPM-DIT

Descargas Sencillas de Datos con Peticiones HTTP GET

`Data(contentsOf URL:)`

© Santiago Pavón - UPM-DIT

Data(contentsOf URL:)

- Para descargar un Data con los datos del sitio especificado en una URL puede usarse:

```
let data: Data? = try? Data(contentsOf: url)
```

  - Uso try? para ignorar errores al obtener los datos; devolviendo **nil**.
  - Para conocer los detalles de los posibles errores, usar:

```
do {
    let data = try Data(contentsOf: url)
    // usar data
} catch {
    print(error.localizedDescription)
}
```

- Es una llamada <u>síncrona</u>.
  - 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 **URLSession**.

© Santiago Pavón - UPM-DIT

```swift
struct ContentView: View {

    @State var img: UIImage = UIImage(named: "none")!

    var body: some View {
        VStack {
            Button(action: { self.download() }) {
                Text("Descargar")
            }
            Image(uiImage: img)
        }
    }

    func download() {
        let imgUrl = "http://www.etsit.upm.es/fileadmin/user_upload/banner_portada_escuela.jpg"
        // No hay que escapar caracteres conflictivos de la URL

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

            // Bajar los datos del sitio Web
            if let data = try? Data(contentsOf: url),

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

                // Actualizar el GUI
                self.img = img
            }
        }
    }
}
```

Espera hasta que se han descargado los datos

© Santiago Pavón - UPM-DIT

```swift
struct ContentView: View {

    @State var img: UIImage = UIImage(named: "none")!

    var body: some View {
        VStack {
            Button(action: { self.download2() }) {
                Text("Descargar")
            }
            Image(uiImage: img)
        }
    }

    func download2() {
        let imgUrl = "http://www.etsit.upm.es/fileadmin/user_upload/banner_portada_escuela.jpg"
        // No hay que escapar caracteres conflictivos de la URL

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

            // Envio la tarea a un thread
            let queue = DispatchQueue(label: "Download Queue")
            queue.async {
                // Bajar los datos del sitio Web
                if let data = try? Data(contentsOf: url),

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

                    // Actualizar el GUI
                    // El GUI se actualiza en el Main Thread
                    DispatchQueue.main.async {
                        self.img = img
                    }
                }
            }
        }
    }
}
```

Uso GCD y envío la tarea a un thread

El GUI solo se actualiza en el Main Thread

© Santiago Pavón - UPM-DIT

Escapado URL

© Santiago Pavón - UPM-DIT

Escapado URL

- Por motivos de interoperabilidad, al manejar URLs y el protocolo HTTP, debemos **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 escapado.

© Santiago Pavón - UPM-DIT

- 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 addingPercentEncoding(withAllowedCharacters
                           allowedCharacters: CharacterSet)-> String?
```

    - Los posibles valores de CharacterSet que nos interesan son:

      .urlFragmentAllowed,

      .urlHostAllowed

      .urlPasswordAllowed

      .urlPathAllowed

      .urlQueryAllowed

      .urlUserAllowed

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

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

© Santiago Pavón - UPM-DIT

```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.addingPercentEncoding(
                              withAllowedCharacters: .urlPathAllowed) {

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

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

    let originalUrl = escapedUrl.removingPercentEncoding!


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

}
```

© Santiago Pavón - UPM-DIT

Serialización

© Santiago Pavón - UPM-DIT

String, UIImage

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

```
let str1 = "El Camión"

let data: Data? = str1.data(using: String.Encoding.utf8)
```

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

```
let str2: String? = String(data: data!,
                           encoding: String.Encoding.utf8)
```

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

```
let img = UIImage(data: data)
```

© Santiago Pavón - UPM-DIT

JSONSerialization

- La clase **`JSONSerialization`** 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: Any) -> Bool
```

© Santiago Pavón - UPM-DIT

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

```
class func jsonObject(with data: Data,
                      options opt: JSONSerialization.ReadingOptions = [])
                                   throws -> Any
```

  - Lanza un error si se produce algún fallo interno.
  - El dato JSON debe estar codificado en UTF-8, UTF-16LE, UTF-16BE, UTF-32LE o UTF-32BE.

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

```
class func data(withJSONObject obj: Any,
                options opt: JSONSerialization.WritingOptions = [])
                                  throws -> Data
```

  - Lanza un error si se produce algún fallo interno.
  - El dato JSON devuelto está codificado en UTF-8.

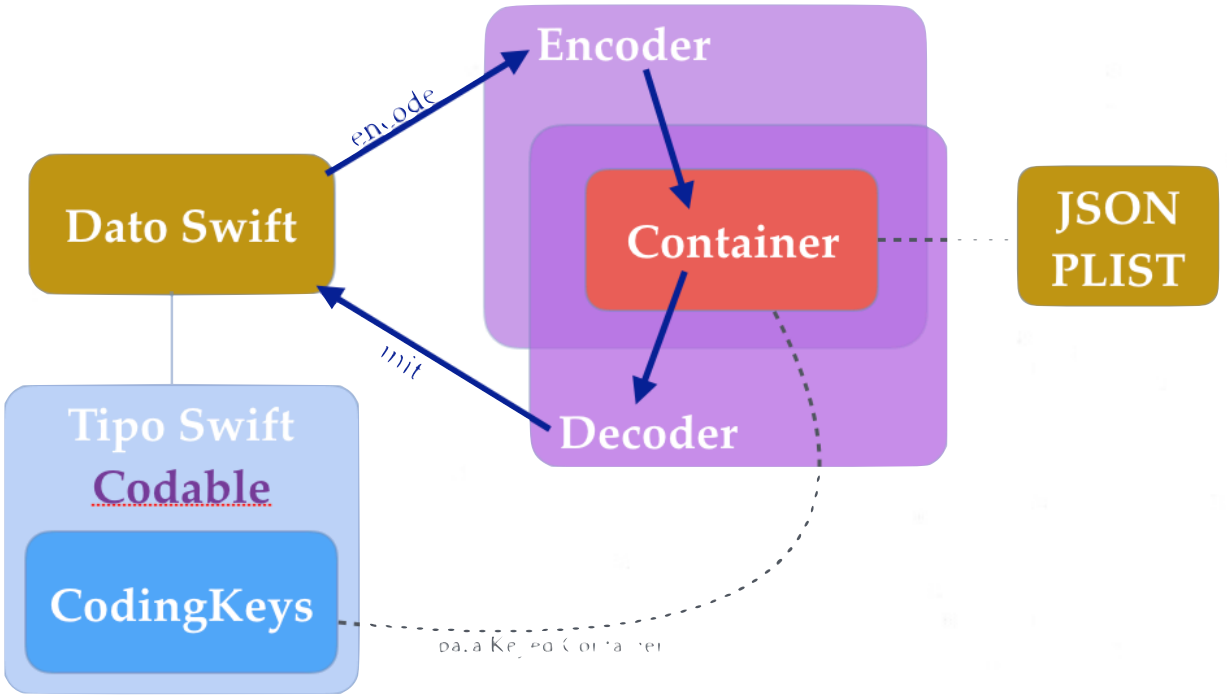
© Santiago Pavón - UPM-DIT

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

// Crear JSON
do {
    let data = try JSONSerialization.data(withJSONObject: person)

    // Ver el buffer de bytes:
    print(data)  // 27 bytes <7b226e6f 6d627265 … 6164223a 32387d>

    // Ver el Data como un String
    let str = String(data: data, encoding: String.Encoding.utf8)
    print(str!) // {"nombre":"Juan","edad":28}

    // Reconstruir el objeto
    let person2 = try JSONSerialization.jsonObject(withData: data)

    // Ver el diccionario
    print(person2) // { nombre: Juan; edad: 28; }
} catch {
    print(error)
}
```

© Santiago Pavón - UPM-DIT

Protocolo **Codable**

- Ver Tema de Persistencia.
- Conversión de los datos entre los tipos del lenguaje Swift y otros formatos externos de representación (ej: JSON, listas de propiedades, XML, ...).



© Santiago Pavón - UPM-DIT

Codificación Base 64

© Santiago Pavón - UPM-DIT

Codificación Base 64

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

```
func base64EncodedString(options: Data.Base64EncodingOptions = default)
                                                    -> String
```

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

```
func base64EncodedData(options: Data.Base64EncodingOptions = default)
                                                    -> Data
```

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

```
init?(base64Encoded base64Data: Data,
      options: Data.Base64DecodingOptions = default)
```

    - desde un String codificado en Base-64:

```
init?(base64Encoded base64String: String,
      options: Data.Base64DecodingOptions = default)
```

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

© Santiago Pavón - UPM-DIT

```swift
// Data con los bytes de una foto
let url = Bundle.main.url(forResource: "perro", withExtension: "jpg")
let data = try? Data(contentsOf: url!)

// String codificado en Base-64 con la foto.
let str64: String = data!.base64EncodedString()



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



// Creo otro Data con los bytes del String en Base-64.
let data2 = Data(base64Encoded: str64)

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

© Santiago Pavón - UPM-DIT

URLSession

© Santiago Pavón - UPM-DIT

URLSession

- Se usa para realizar transferencias de datos usando **https**.
  - También **http**, **ftp**, **file**, **data**, propietarios.
- Características:
  - Soporta HTTP/1.1, SPDY, HTTP/2.
  - App Transport Security (ATS)
  - HTTP Strict Transport Security (HSTS).
  - Caches, cookies, proxies, autenticación.
  - Descargas en background.
  - Configuraciones: version TLS, uso red celular, tipo de calidad de servicio, política caches, política cookies, timeouts.
  - …
- **¿Cómo se usa?**: Una app crea una sesión, la configura, y crea varias tareas de transferencia de datos.
  - Las apps también pueden crear varias sesiones con diferentes configuraciones, y cada sesión coordinará las tareas de transferencia de datos relacionadas.

© Santiago Pavón - UPM-DIT

- Para cada sesión creada se realiza 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.

© Santiago Pavón - UPM-DIT

- 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 URLSession es **Thread-Safe**.
  - Las tareas se ejecutan en **Threads** separados para no bloquear el Main Thread.
    - El thread donde se ejecuta una tarea depende de como se cree.
- Las tareas se 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.
  - Consultar en la documentación la guía **URL Session Programming Guide**.

© Santiago Pavón - UPM-DIT

Clases

- **URLSessionConfiguration**
  - Crear un objeto para configurar inicialmente una sesión.
- **URLSession**
  - Crear los objetos que representan una sesión.
- **URLSessionTask**
  - Clase base de los distintos tipos de tareas.
    - **URLSessionDataTask**
      - Para descargar el contenido de una URL en un Data.
        - **URLSessionUploadTask**
          - Para subir un fichero y recibir los datos de la respuesta en un Data.
    - **URLSessionDownloadTask**
      - Para descargar el contenido de una URL en un fichero temporal.

➡ Otras clases que también se usan son **URL**, **URLRequest**, **URLResponse**, **HTTPURLResponse**, **CachedURLResponse**, …

© Santiago Pavón - UPM-DIT

Protocolos

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

© Santiago Pavón - UPM-DIT

Gestión Personalizada Autenticación y TLS

- URLSession 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.
  - `URLSessionTaskDelegate`

```
func urlSession(URLSession, task: URLSessionTask,
         didReceive: URLAuthenticationChallenge,
         completionHandler: (URLSession.AuthChallengeDisposition,
                             URLCredential?) -> Void)
```

  - `URLSessionDelegate`

```
func urlSession(URLSession, didReceive: URLAuthenticationChallenge,
         completionHandler: (URLSession.AuthChallengeDisposition,
                             URLCredential?) -> Void)
```

- 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.

© Santiago Pavón - UPM-DIT

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 está usando la sintaxis **Trailing Closure**.
        - Si el último parámetro de una función es una closure, se puede sacar fuera de la función.
      - El Completion Handler 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.

© Santiago Pavón - UPM-DIT

```
let session = URLSession.shared  // Crear una sesión


// Construir un URL. No es necesario escapar ningún carácter.
let imgUrl = "http://www.etsit.upm.es/fileadmin/user_upload/banner_portada_escuela.jpg"
let url = URL(string: imgUrl)!


let task = session.dataTask(with: url) { (data: Data?, // Crear Data Task
                                          response: URLResponse?,
                                          error: Error?) in
    if error == nil && (response as! HTTPURLResponse).statusCode == 200 {

       if let img = UIImage(data: data!) { // Construir imagen
          DispatchQueue.main.async  {
              self.image = img  // Actualizar el estado
          }
       } else { print("Error construyendo la imagen") }
    } else { print("Error descargando") }

}

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

© Santiago Pavón - UPM-DIT

Ejemplo 2 - Configurar una Sesión

```
// Siempre hay que crear primero la configuracion:
var config = URLSessionConfiguration.default

// 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 = URLSession(configuration: config)

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

© Santiago Pavón - UPM-DIT

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 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.

© Santiago Pavón - UPM-DIT

```swift
// Crear una sesión con la configuracion default:
let session = URLSession(configuration: URLSessionConfiguration.default)

// Construir un URL. No es necesario escapar ningún carácter.
let imgUrl = "http://www.etsit.upm.es/fileadmin/user_upload/banner_portada_escuela.jpg"
let url = URL(string: imgUrl)!

// Crear un Download Task con un Completion handler
let task = session.downloadTask(with: url) { (location: URL?,
                                              response: URLResponse?,
                                              error: Error?) in
   if error == nil && (response as! HTTPURLResponse).statusCode == 200 {
        // location es la URL donde se ha descargado la imagen.
        // Es un Data.
        if let data = Data(contentsOf: location!),
           let img = UIImage(data: data) {
            DispatchQueue.main.async { // Actualizar el estado
                self.image = img
            }
        }
   }
}

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

© Santiago Pavón - UPM-DIT

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.

© Santiago Pavón - UPM-DIT

```
class ViewController: UIViewController, URLSessionDownloadDelegate {


    // Construir un URL
    let imgUrl = "http://www.etsit.upm.es/fileadmin/user_upload/banner_portada_escuela.jpg"
    let url = URL(string: imgUrl)!

    // Crear la configuracion de la sesion:
    let config = URLSessionConfiguration.default

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

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

    // Arrancar la tarea
    task.resume()

continúa ...
```

© Santiago Pavón - UPM-DIT

```
// Termino la descarga
func urlSession(_ session: URLSession,
     downloadTask: URLSessionDownloadTask,
     didFinishDownloadingTo location: URL) {

    // location es el URL a los datos descargados.
    if let data = try? Data(contentsOf: location),
       let img = UIImage(data: data) {

       DispatchQueue.main.async { // Actualizar el estado
           self.image = img
       }
    }
}

// Trazas de progreso
func urlSession(_ session: URLSession,
     downloadTask: URLSessionDownloadTask,
     didWriteData bytesWritten: Int64,
     totalBytesWritten: Int64,
     totalBytesExpectedToWrite: Int64) {

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

© Santiago Pavón - UPM-DIT

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.

© Santiago Pavón - UPM-DIT

```swift
// Crear la configuracion de la sesion:
let config = URLSessionConfiguration.default

// Crear una session
let session = URLSession(configuration: config)

// URL del sitio de subida
let url = URL(string: "http://localhost:3000/upload")!

// URL de la imagen a subir
let file: URL = Bundle.main.url(forResource: "perro",
                                withExtension:"jpg")!

// La petición HTTP:
var request = URLRequest(url: url)
request.httpMethod = "POST"
request.addValue("image/jpeg", forHTTPHeaderField: "Content-Type")
```

© Santiago Pavón - UPM-DIT

```
// La tarea para subir los datos:
let task = session.uploadTask(with: request, fromFile: file) {
    (data: Data?, res: URLResponse?, error: Error?) in

        if error != nil {
            print(error!.localizedDescription)
            return
        }

        let code = (res as! HTTPURLResponse).statusCode
        if code != 201 {
            print(HTTPURLResponse.localizedString(forStatusCode: code))
            return
        }

        print("Subido")
}

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

© Santiago Pavón - UPM-DIT

© Santiago Pavón - UPM-DIT