



POLITÉCNICA

ETSIT  
UPM

*dit*  
UPM

# Desarrollo de Apps para iOS

# Persistencia

IWEB,LSWC 2014-2015

Santiago Pavón

ver: 2014.11.25

# Persistencia

- Conservar datos
  - Aunque se pare y relance la aplicación.
  - Aunque apague y encienda el terminal.
- Existen varias formas de guardar datos:
  - User Defaults (preferencias de usuario)
  - Sistema de Ficheros
  - SQLite3
  - Core Data
  - Cloud

# Preferencias de Usuario

# Preferencias de Usuario

- Las preferencias de usuario son valores persistentes usados por la aplicación.
- Pueden modificarse:
  - desde la propia aplicación.
  - desde la aplicación **Ajustes** (**Settings**).
- Para poder modificar las preferencias desde **Ajustes**:
  - La aplicación debe tener un **settings bundle**.
    - conjunto de ficheros describiendo los datos de preferencias.
  - **Ajustes** crea un GUI para editar los datos.
- **NSUserDefaults**
  - Es la clase usada para almacenar / recuperar los valores de las preferencias.
    - Cada valor está asociado a una clave.

# Acceder desde nuestra aplicación

- **NSUserDefaults** implementa un singleton

```
var def: NSUserDefaults = NSUserDefaults.standardUserDefaults()
```

- Pueden guardarse combinaciones de:

- NSData, NSString, String, NSNumber, Int, Double, Float, NSDate, NSArray, Array, NSDictionary, Dictionary, URL, Bool.
- Otros tipos de datos pueden guardarse si se serializan, por ejemplo en un NSData.

- Se usa como un diccionario:

- para obtener datos:

```
func objectForKey(_ defaultName: String) -> AnyObject?  
func integerForKey(_ defaultName: String) -> Int  
func boolForKey(_ defaultName: String) -> Bool  
...
```

- para salvar datos:

```
func setObject(_ value: AnyObject?, forKey defaultName: String)  
func setFloat(_ value: Float, forKey defaultName: String)  
...
```

- Antes de salir invocar **synchronize()** para salvar datos de la cache.

# Cuando cargar / salvar datos

- Cada vez que se cambie o necesite el dato.
- Cuando la pantalla va a mostrarse o ocultarse.
  - `viewWillAppear` `viewWillDisappear`
- Al cargar una pantalla en memoria.
  - `viewDidLoad`
- En los métodos del delegado de la aplicación.
  - Se llaman al pasar a segundo plano, cuando va a terminar la app, ...
- ...

# Ejemplo

- Cuando se carga el VC recupero los valores de las preferencias:

```
override func viewDidLoad() {
    super.viewDidLoad()

    let defaults = UserDefaults.standardUserDefaults()
    if let model = defaults.objectForKey("model") as? String {
        carModel = model
    } else {
        carModel = "Ferrari"
    }
    carSpeed = defaults.doubleForKey("speed") // 0 si no existe
}
```

- Salvo las preferencias cuando la pantalla desaparece:

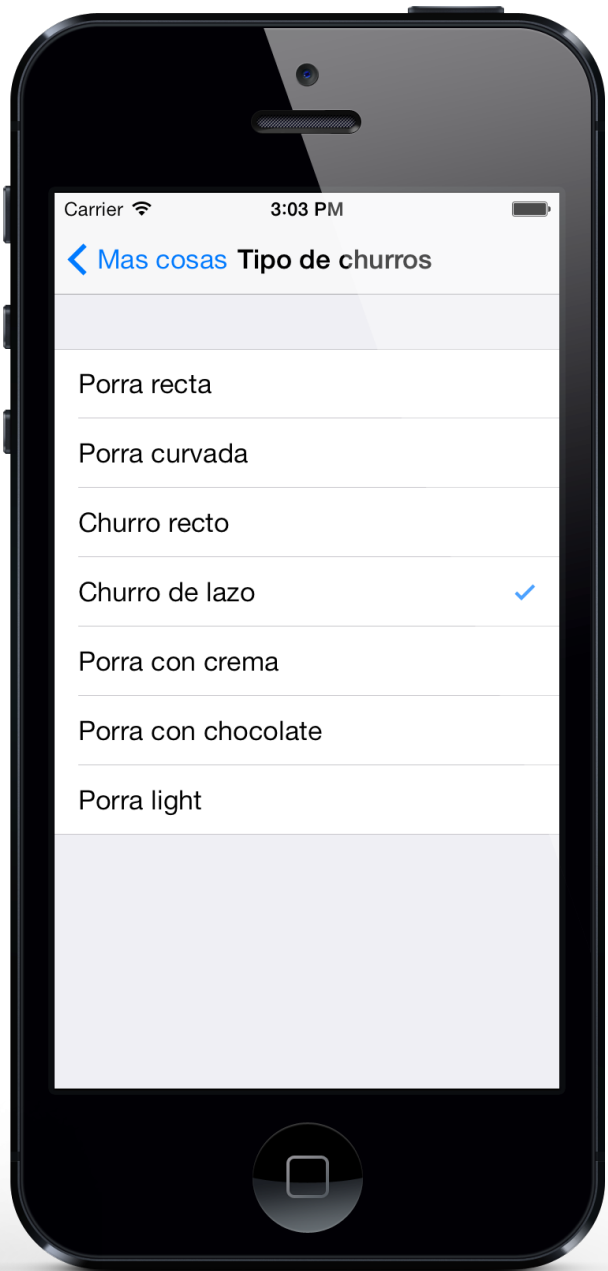
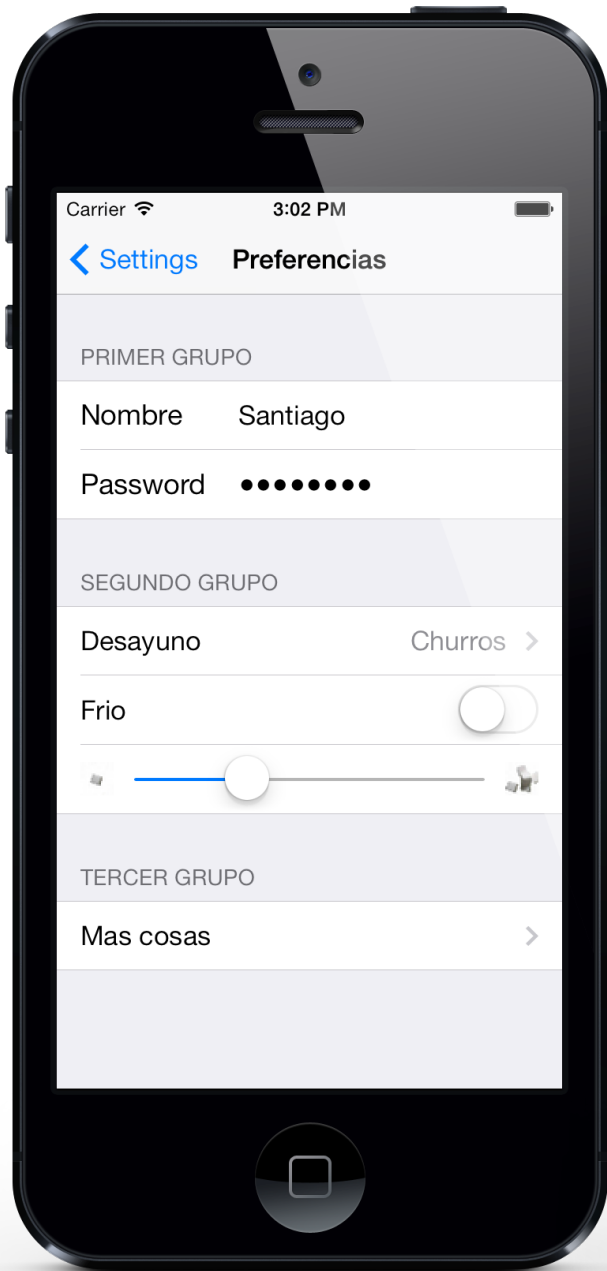
```
override func viewWillDisappear(animated: Bool) {
    super.viewWillDisappear(animated)

    let defaults = UserDefaults.standardUserDefaults()
    defaults.setObject(carModel, forKey: "model")
    defaults.setDouble(carSpeed, forKey: "speed")
    defaults.synchronize() // Forzar la sincronización ahora
}
```

# Acceso desde la Aplicación Ajustes

- Las preferencias de usuario pueden editarse desde la aplicación **Ajustes (Settings)**.
  - Solo pueden editarse algunos tipos de valores.
- Para ello, hay que crear en nuestra aplicación un **Settings Bundle**:
  - New File > iOS > Resource > Settings Bundle
  - Para manipular el contenido de este fichero hay que usar Finder.
    - Por ejemplo, para añadir un imagen.
- **Root.plist**
  - define la primera vista de las preferencias.
- Para crear subvistas adicionales (nuevas pantallas) hay que crear nuevos ficheros de listas de propiedades.
  - Un fichero **plist** para cada pantalla adicional.
- Consultar la guía:
  - **Preferences and Settings Programming Guide: About Preferences and Settings.**





# Root.plist

The screenshot shows the Xcode interface with the 'Root.plist' file selected in the project navigator. The breadcrumb path is 'Preferencias > Preferencias > Settings.bundle > Root.plist > No Selection'. The main editor displays a table with the following data:

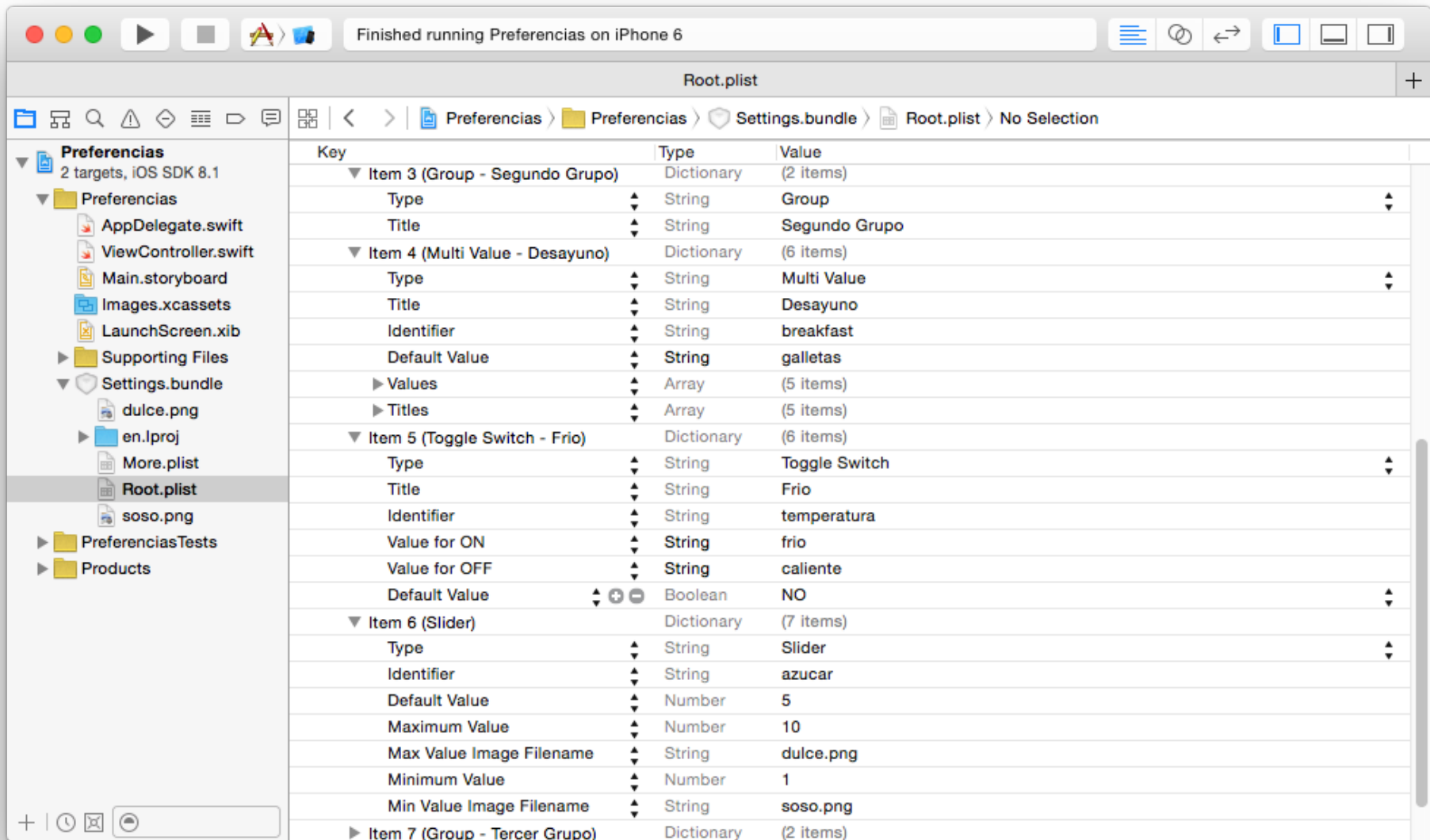
Key	Type	Value
▼ iPhone Settings Schema	Dictionary	(2 items)
▼ Preference Items	Array	(9 items)
▶ Item 0 (Group - Primer Grupo)	Dictionary	(2 items)
▶ Item 1 (Text Field - Nombre)	Dictionary	(8 items)
▶ Item 2 (Text Field - Password)	Dictionary	(6 items)
▶ Item 3 (Group - Segundo Grupo)	Dictionary	(2 items)
▶ Item 4 (Multi Value - Desayuno)	Dictionary	(6 items)
▶ Item 5 (Toggle Switch - Frio)	Dictionary	(6 items)
▶ Item 6 (Slider)	Dictionary	(7 items)
▶ Item 7 (Group - Tercer Grupo)	Dictionary	(2 items)
▶ Item 8 (Child Pane - Mas cosas)	Dictionary	(3 items)
Strings Filename	String	Root

Finished running Preferencias on iPhone 6

Root.plist

Preferencias > Preferencias > Settings.bundle > Root.plist > No Selection

Key	Type	Value
▼ iPhone Settings Schema	Dictionary	(2 items)
▼ Preference Items	Array	(9 items)
▼ Item 0 (Group - Primer Grupo)	Dictionary	(2 items)
Title	String	Primer Grupo
Type	String	Group
▼ Item 1 (Text Field - Nombre)	Dictionary	(8 items)
Type	String	Text Field
Title	String	Nombre
Identifier	String	username
Autocapitalization Style	String	None
Autocorrection Style	String	No Autocorrection
Default Value	String	
Text Field Is Secure	Boolean	NO
Keyboard Type	String	Alphabet
▼ Item 2 (Text Field - Password)	Dictionary	(6 items)
Type	String	Text Field
Title	String	Password
Identifier	String	password
Autocapitalization Style	String	None
Autocorrection Style	String	No Autocorrection
Text Field Is Secure	Boolean	YES
▶ Item 3 (Group - Segundo Grupo)	Dictionary	(2 items)
▶ Item 4 (Multi Value - Desayuno)	Dictionary	(6 items)
▶ Item 5 (Toggle Switch - Frio)	Dictionary	(6 items)
▶ Item 6 (Slider)	Dictionary	(7 items)
▶ Item 7 (Group - Tercer Grupo)	Dictionary	(2 items)



Finished running Preferencias on iPhone 6

Root.plist

Preferencias > Preferencias > Settings.bundle > Root.plist > No Selection

Key	Type	Value
▼ iPhone Settings Schema	Dictionary	(2 items)
▼ Preference Items	Array	(9 items)
▶ Item 0 (Group - Primer Grupo)	Dictionary	(2 items)
▶ Item 1 (Text Field - Nombre)	Dictionary	(8 items)
▶ Item 2 (Text Field - Password)	Dictionary	(6 items)
▶ Item 3 (Group - Segundo Grupo)	Dictionary	(2 items)
▶ Item 4 (Multi Value - Desayuno)	Dictionary	(6 items)
▶ Item 5 (Toggle Switch - Frio)	Dictionary	(6 items)
▶ Item 6 (Slider)	Dictionary	(7 items)
▼ Item 7 (Group - Tercer Grupo)	Dictionary	(2 items)
Type	String	Group
Title	String	Tercer Grupo
▼ Item 8 (Child Pane - Mas cosas)	Dictionary	(3 items)
Type	String	Child Pane
Title	String	Mas cosas
Filename	String	More
Strings Filename	String	Root

# More.plist

Finished running Preferencias on iPhone 6

More.plist

Preferencias > Preferencias > Settings.bundle > More.plist > No Selection

Key	Type	Value
▼ Root	Dictionary	(2 items)
Title	String	More Settings
▼ PreferenceSpecifiers	Array	(6 items)
▼ Item 0	Dictionary	(2 items)
Type	String	PSGroupSpecifier
Title	String	Marcas
▼ Item 1	Dictionary	(5 items)
Type	String	PSTextFieldSpecifier
Title	String	Leche
Key	String	marca_leche
AutocapitalizationType	String	No
AutocorrectionType	String	No
▼ Item 2	Dictionary	(5 items)
Type	String	PSTextFieldSpecifier
Title	String	Cafe
Key	String	marca_cafe
AutocapitalizationType	String	None
AutocorrectionType	String	No
▼ Item 3	Dictionary	(5 items)
Type	String	PSTextFieldSpecifier
Title	String	Galletas
Key	String	marca_galletas
AutocapitalizationType	String	None
AutocorrectionType	String	No
▶ Item 4	Dictionary	(5 items)
▶ Item 5	Dictionary	(6 items)

Finished running Preferencias on iPhone 6

More.plist

Preferencias > Preferencias > Settings.bundle > More.plist > No Selection

Key	Type	Value
▼ Item 4	Dictionary	(5 items)
Type	String	PSTextFieldSpecifier
Title	String	Cereales
Key	String	marca_cereales
AutocapitalizationType	String	None
AutocorrectionType	String	No
▼ Item 5	Dictionary	(6 items)
Type	String	PSMultiValueSpecifier
Title	String	Tipo de churros
Key	String	tipo_churros
DefaultValue	String	porra
▼ Values	Array	(7 items)
Item 0	String	Porra recta
Item 1	String	Porra curva
Item 2	String	Churro recto
Item 3	String	Churro de lazo
Item 4	String	Porra de crema
Item 5	String	Porra de chocolate
Item 6	String	Porra light
▼ Titles	Array	(7 items)
Item 0	String	Porra recta
Item 1	String	Porra curvada
Item 2	String	Churro recto
Item 3	String	Churro de lazo
Item 4	String	Porra con crema
Item 5	String	Porra con chocolate
Item 6	String	Porra light

# Main Bundle



# NSBundle

- Un objeto **NSBundle** representa un lugar del sistema de ficheros.
  - Carpeta donde se guardan recursos, código, etc.
- Las aplicaciones y frameworks son bundles.
- El **main bundle** de una aplicación permite acceder a los recursos que se añadieron en el proyecto.
- Estos objetos están firmados,
  - no pueden modificarse.

# Usar el Main Bundle

```
let bundle: NSBundle = NSBundle mainBundle()
```

```
let path: String? = bundle.pathForResource("pokemons",  
                                           ofType:"plist")
```

```
let url: NSURL? = bundle.URLForResource("pokemons",  
                                        withExtension:"plist")
```

```
let img: UIImage? = UIImage(named:"foto.jpg")
```

# Sistema de Ficheros

# Sistema de Ficheros

- Las aplicaciones ven un sistema de ficheros UNIX.
- Las aplicaciones corren en un Sandbox.
  - La ejecución de un programa no daña a otros.
  - Proteger acceso a los datos de una aplicación.
  - Fácil borrar datos al desinstalar una aplicación.
- Contenido del sandbox:
  - directorio del bundle de aplicación. (sólo lectura).
  - directorio Documents. (donde salvar los datos permanentes).
  - directorio de caches. (temporales sin backup de itunes).
  - ...

# Obtener Rutas a Directorios

- Directorio Home:

```
let home: String = NSHomeDirectory()
```

- Directorio Temporal:

```
let tmp: String = NSTemporaryDirectory()
```

- Directorio Documents:

```
let paths = NSSearchPathForDirectoriesInDomains(  
    NSSearchPathDirectory.DocumentDirectory,  
    NSSearchPathDomainMask.UserDomainMask,  
    true)
```

```
let docsPath = paths[0] as String
```

# Manipular Rutas

- Consultar la documentación de las clases `NSString` y `NSURL`.

```
// Directorio de documentos:
```

```
let paths = NSSearchPathForDirectoriesInDomains(.DocumentDirectory,  
                                                .UserDomainMask, true)
```

```
let docsPath = paths[0] as String
```

```
// Añadir al path:
```

```
let datosPath: String = docsPath.stringByAppendingPathComponent("g.dat")
```

```
// La extension de un fichero:
```

```
let ext: String = datosPath.pathExtension
```

```
// Nombre del fichero sin extension:
```

```
let name: String =  
    datosPath.lastPathComponent.stringByDeletingPathExtension
```

```
// Directorio del fichero:
```

```
let basedir: String = datosPath.stringByDeletingLastPathComponent
```

```
// URL del directorio de Documentos:
let docsURL: NSURL? = NSURL.fileURLWithPath(docsPath)

// Añadir al URL:
let datosURL: NSURL = docsURL!.URLByAppendingPathComponent("g.dat")

// Absolute string:
let absPath: String? = datosURL.absoluteString

// Escapar URL: URL encoding
let esc: String? = "a b c".stringByAddingPercentEscapesUsingEncoding(
    NSUTF8StringEncoding) // a%20b%20c
```

# NSFileManager

- Proporciona métodos para:
  - ver si un fichero existe.
  - crear y examinar directorios.
  - manipular ficheros: copiar, mover, borrar.
  - comparar ficheros.
  - obtener URL de directorios del sistema.
  - ...
- Consultar la documentación para ver todos los métodos disponibles.



# Ejemplo

- Copiar un fichero del Bundle de la aplicación en el directorio de documentos:

```
// Crear un File Manager
let fm = NSFileManager()

// Fichero origen
let bundle: NSBundle = NSBundle.mainBundle()
let origenURL: NSURL? = bundle.URLForResource("pokemons",
                                             withExtension:"plist")

// Fichero destino (en el directorio de documentos)
let docsURLs: [AnyObject] = fm.URLsForDirectory(.DocumentDirectory,
                                              inDomains:.UserDomainMask)

let docsURL: NSURL = docsURLs[0] as NSURL
let destinoURL: NSURL = docsURL.URLByAppendingPathComponent("pokemons.plist")

// Copiar
var error: NSError?
let res: Bool = fm.copyItemAtURL(origenURL!, toURL: destinoURL, error: &error)

// Cargar el fichero copiado
var dic = NSDictionary(contentsOfURL: destinoURL) as? [String:AnyObject]
```

# Usar funciones I/O de C

```
FILE *fp;
if ( (fp = fopen(path,"r")) == NULL) {
    NSLogs("No puedo abrir el fichero de iconos");
    return;
}
char line[1024];
int code;
char name[1024];
while ( fgets(line,1024,fp) != NULL) {
    sscanf(line,"%i %s", &code, name);
    NSString* key = [NSString stringWithFormat:@"%i",code];
    NSString* val = [NSString stringWithFormat:@"%s",name];
    [dictionary setValue:val forKey:key];
}
fclose(fp);
```

Lenguaje C

# Lista de Propiedades

# Lista de Propiedades

- Es un **dato** formado por cualquier combinación de los tipos:
  - **NSArray, NSDictionary, NSData, NSString, NSNumber, NSDate.**
  - y los relacionados de Swift: **Array, Dictionary, String, Int, Double, Float, Bool.**
- Se pueden guardar y recuperar de ficheros **.plist**.

```
var dic = NSDictionary(contentsOfFile:path1) as [String:String]
dic["clave"] = "valor"
(dic as NSDictionary).writeToFile(path1, atomically:true)
```

```
var arr = NSArray(contentsOfFile:path2) as [Int]
arr += 100
(arr as NSArray).writeToFile(path2, atomically:true)
```

# NSPropertyListSerialization

- La clase **NSPropertyListSerialization** proporciona métodos para:

- serializar una lista de propiedades en un **NSData**.

```
class func dataWithPropertyList(_ plist: AnyObject,  
                                format format: NSPropertyListFormat,  
                                options opt: NSPropertyListWriteOptions,  
                                error error: NSErrorPointer) -> NSData?
```

- y crear una lista de propiedades desde un **NSData**.

```
class func propertyListWithData(_ data: NSData,  
                                options opt: NSPropertyListReadOptions,  
                                format format:  
                                UnsafeMutablePointer<NSPropertyListFormat>,  
                                error error: NSErrorPointer) -> AnyObject?
```

- Los objetos **NSData** se pueden leer y escribir en ficheros usando:

```
func writeToURL(_ aURL: NSURL, atomically atomically: Bool) -> Bool
```

```
init?(contentsOfURL aURL: NSURL)
```

```
. . .
```

# Protocolo **NSCoding**

# NSCoding

- El protocolo **NSCoding** declara los dos métodos que debe implementar una clase para que sus objetos puedan serializarse (encode) y des-serializarse (decode).

```
protocol NSCoding {  
    func encodeWithCoder(aCoder: NSCoder)  
    init(coder aDecoder: NSCoder)  
}
```

- Nuestras clases deben adoptar este protocolo si queremos serializarlas para guardarlas en ficheros, guardarlas en las preferencias, transmitir las por un socket, etc.
- **NSCoder** es una clase abstracta.
  - Define métodos para convertir objetos en **NSData**, e inversa.
  - Algunas subclasses:
    - **NSKeyedArchiver**, **NSKeyedUnarchiver**, **NSArchiver**, **NSUnarchiver**, **NSPortCoder**

# Método para Codificar

- Si nuestra clase deriva de `NSObject` o de una superclase que **no es conforme** a `NSCoding`:

```
func encodeWithCoder(aCoder: NSCoder) {  
  
    // Codificar las propiedades de nuestra clase.  
    aCoder.encodeObject(nombre, forKey: "nombre")  
    aCoder.encodeObject(coche, forKey: "vehiculo")  
    aCoder.encodeInt(edad, forKey: "edad")  
    aCoder.encodeFloat(otro, forKey: "mas")  
}
```



- Si nuestra clase deriva de una superclase que es conforme a **NSCoding**:

```
override func encodeWithCoder(aCoder: NSCoder) {  
  
    // La superclase es conforme a NSCoder,  
    // por tanto, codificamos sus propiedades.  
    super.encodeWithCoder(aCoder)  
  
    // Codificar las propiedades de nuestra clase.  
    aCoder.encodeObject(nombre, forKey: "nombre")  
    aCoder.encodeObject(coche, forKey: "vehiculo")  
    aCoder.encodeInt(edad, forKey: "edad")  
    aCoder.encodeFloat(otro, forKey: "mas")  
}
```

# Método para Decodificar

- Si nuestra clase deriva de `NSObject` o de una superclase que **no es conforme** a `NSCoding`:

```
override init(coder aDecoder: NSCoder) {  
  
    super.init()  
  
    // Decodificar las propiedades de nuestra clase.  
    nombre = aDecoder.decodeObjectForKey("nombre") as? String  
    coche  = aDecoder.decodeObjectForKey("vehiculo") as? String  
    edad   = aDecoder.decodeIntForKey("edad")  
    otro   = aDecoder.decodeFloatForKey("mas")  
}
```

- Si nuestra clase deriva de una superclase que es conforme a **NSCoding**:

```
override init(coder aDecoder: NSCoder) {  
  
    // Decodificar las propiedades de nuestra superclase.  
    super.initWithCoder(coder: aDecoder)  
  
    // Decodificar las propiedades de nuestra clase.  
    nombre = aDecoder.decodeObjectForKey("nombre") as? String  
    coche  = aDecoder.decodeObjectForKey("vehiculo") as? String  
    edad   = aDecoder.decodeIntForKey("edad")  
    otro   = aDecoder.decodeFloatForKey("mas")  
  
}
```

# Ejemplo: Codificar 3 Objetos

```
// Crear un codificador que guarda los datos en un buffer:  
let data = NSMutableData()  
let archiver = NSKeyedArchiver(forWritingWithMutableData: data)  
  
// Codificar varios valores asociandolos a varias claves:  
archiver.encodeObject(objeto1, forKey:"clave1")  
archiver.encodeObject(objeto2, forKey:"clave2")  
archiver.encodeObject(objeto3, forKey:"clave3")  
  
// Ya he terminado de codificar:  
archiver.finishEncoding()  
  
// Salvar (el buffer NSData) en un fichero:  
let ok: Bool = data.writeToFile("abc", atomically:true)
```

# Ejemplo: Decodificar 3 Objetos

```
// Leer los datos guardados en un fichero.  
let data = NSData(contentsOfFile: "abc")  
  
// Crear el decodificador que extrae de data.  
let unarchiver = NSKeyedUnarchiver(forReadingWithData: data!)  
  
// Decodificar los objetos asociados a cada clave:  
let objeto1 = unarchiver.decodeObjectForKey("clave1") as? LaClase  
let objeto2 = unarchiver.decodeObjectForKey("clave2") as? LaClase  
let objeto3 = unarchiver.decodeObjectForKey("clave3") as? LaClase  
  
// Ya he terminado de decodificar:  
unarchiver.finishDecoding()
```

# Ejemplo: Agenda

```
class AgendaModel: NSObject, NSCoding {  
  
    var phones: Dictionary<String, String>?  
  
    override init() {  
        super.init()  
    }  
  
    required init(coder aDecoder: NSCoder) {  
        super.init()  
        phones = aDecoder.decodeObjectForKey("phones") as? Dictionary  
    }  
  
    func encodeWithCoder(aCoder: NSCoder) {  
        aCoder.encodeObject(phones, forKey: "phones")  
    }  
  
    . . .  
}
```

```
var agenda: AgendaModel!
```

```
loadAgenda()
```

```
agenda.phones = ["Peter": "123456789", "Bob": "987654321"]
```

```
saveAgenda()
```

```
func loadAgenda() {
```

```
    agenda = nil
```

```
    let def = UserDefaults.standardUserDefaults()
```

```
    if let data = def objectForKey("phonebook") as? NSData {
```

```
        agenda = NSKeyedUnarchiver.unarchiveObjectWithData(data)
                as? AgendaModel
```

```
    }
```

```
    if agenda == nil {
```

```
        agenda = AgendaModel()
```

```
    }
```

```
}
```

```
func saveAgenda() {
```

```
    let data = NSKeyedArchiver.archivedDataWithRootObject(agenda)
```

```
    let def = UserDefaults.standardUserDefaults()
```

```
    def.setObject(data, forKey: "phonebook")
```

```
}
```

# Ejemplo: Directamente a un Fichero

- Guardar una jerarquía de objetos en un fichero:

- Se hace una copia en profundidad

```
let obj = UnaClase()
let path = "un_path/miejemplo.save"
let res: Bool = NSKeyedArchiver.archiveRootObject(obj,
                                                    toFile: path)
```

- Recuperar la jerarquía de objetos desde el fichero:

```
var obj: UnaClase
let path = "un_path/miejemplo.save"
obj = NSKeyedUnarchiver.unarchiveObjectWithFile(path) as UnaClase
```

- Nota: la clase **UnaClase** debe ser conforme a **NSCoding**.



# SQLite3

# SQLite3

- iOS incluye soporte para esta base de datos SQL.
- Es una BD contenida en un único fichero.
- Introducción a API C de SQLite 3  
<http://www.sqlite.org/cintro.html>
- Guía del lenguaje SQL SQLite:  
<http://www.sqlite.org/lang.html>

# Crear una base de datos

```
sqlite *database;  
int res = sqlite3_open("path_a_db",&database);  
  
char *errMsg;  
char *cmd = "CREATE TABLE IF NOT EXISTS PEOPLE  
(ID INTEGER PRIMARY KEY AUTOINCREMENT, NAME TEXT)";  
  
res = sqlite3_exec(database,cmd,NULL,NULL,&errMsg);  
  
sqlite3_close(database);
```

Lenguaje C

# Core Data

# Core Data

- Aplicación para el diseño visual de modelos de datos.
- Los datos se almacenan por defecto usando una base de datos SQLite.
  - Alternativas: ficheros binarios, memoria.
- Manejamos ese almacén de datos usando un contexto.
  - No vemos como se almacena.

- Con el editor creamos entities
  - Son los tipos de datos que creamos.
- En la aplicación creamos “**managed objects**”
  - son las instancias de las entities definidas.
  - Las entities definen propiedades
  - los managed objects usan KVC
    - para acceder al valor de una propiedad se usa su nombre como clave.

