ETSIT UPM

dit UPM

Desarrollo de Apps para iOS

Persistencia

IWEB 2016-2017
Santiago Pavón

ver: 2016.11.24

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.
- **UserDefaults**
  - Es la clase usada para almacenar/recuperar los valores de las preferencias.
    - Cada valor está asociado a una clave.

Acceder desde nuestra aplicación

- **UserDefaults** implementa un singleton

```
var def: UserDefaults = UserDefaults.standard
```

- Pueden guardarse combinaciones de:
  - Data, String, Int, Double, Float, Date, Array, Dictionary, URL, Bool.
  - NSData, NSString, NSNumber, NSDate, NSArray, NSDictionary, NSURL.
  - Otros tipos de datos pueden guardarse si se serializan, por ejemplo en un Data.
- Se usa como un diccionario:
  - para obtener datos:

```
func object(forKey defaultName: String) -> Any?
func integer(forKey defaultName: String) -> Int
func bool(forKey defaultName: String) -> Bool
...
```

  - para salvar datos:

```
func set(_ value: Any?, forKey defaultName: String)
func set(_ value: Int, forKey defaultName: String)
func set(_ value: Bool, forKey defaultName: String)
...
```

  - Invocar `synchronize() -> Bool` para salvar los datos de la cache que no se hayan salvado aun.
    - Este método se llama automáticamente periódicamente

Cuando cargar / salvar datos

- Cada vez que se cambie o necesite el dato.
- Cuando la pantalla va a mostrarse o ocultarse.
  - viewWillAppear viewWillDisappear
    - *Recordad que viewWillDisappear y viewDidDisappear no se llaman cuando termina una aplicación.*
- 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.standard
    if let model = defaults.object(forKey: "model") as? String {
        carModel = model
    } else {
        carModel = "Ferrari"
    }
    carSpeed = defaults.doubleForKey("speed")  // devuelve 0 si no existe
}
```

- Salvo las preferencias cuando la pantalla desaparece:

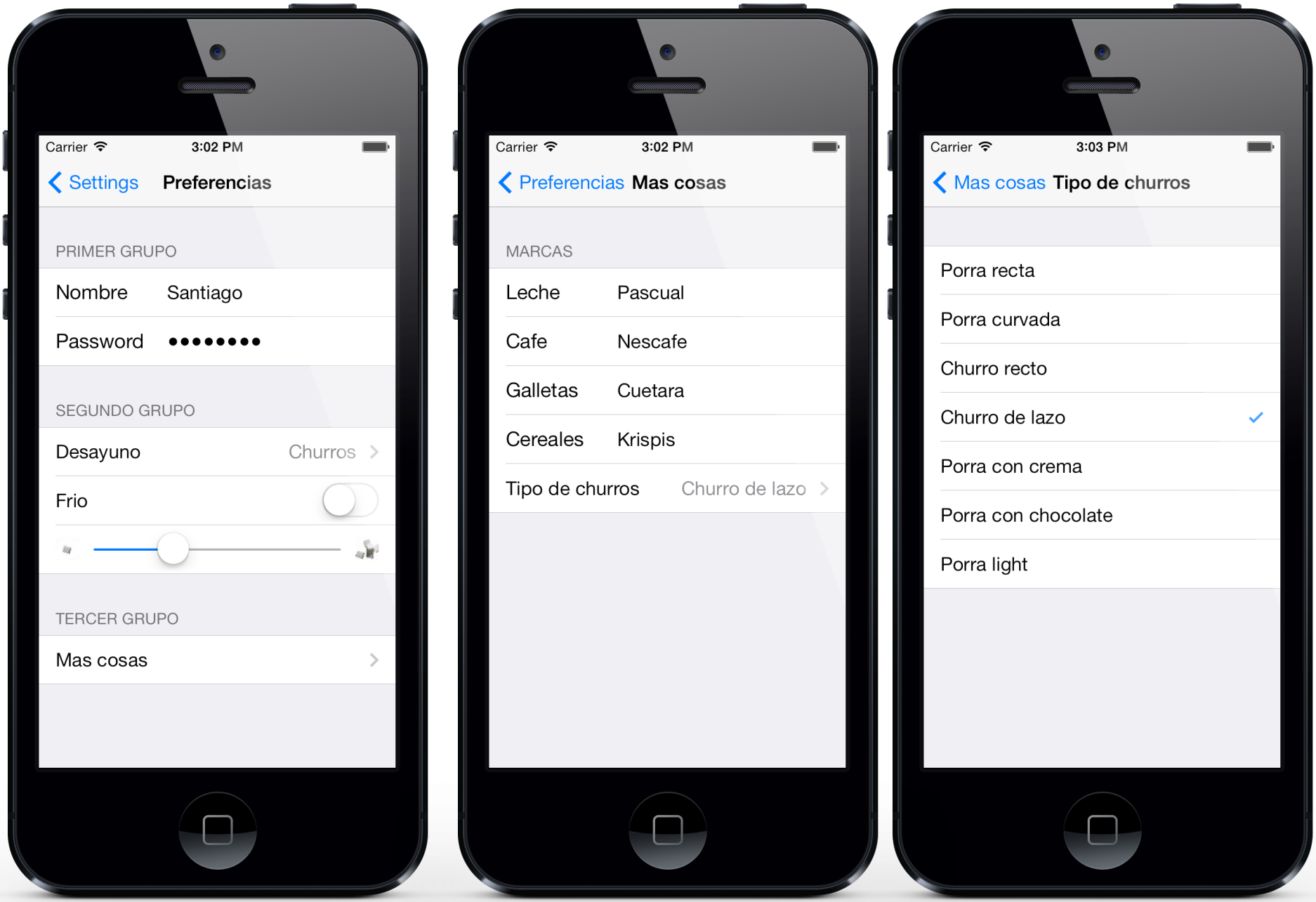
```
override func viewWillDisappear(animated: Bool) {
    super.viewWillDisappear(animated)

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

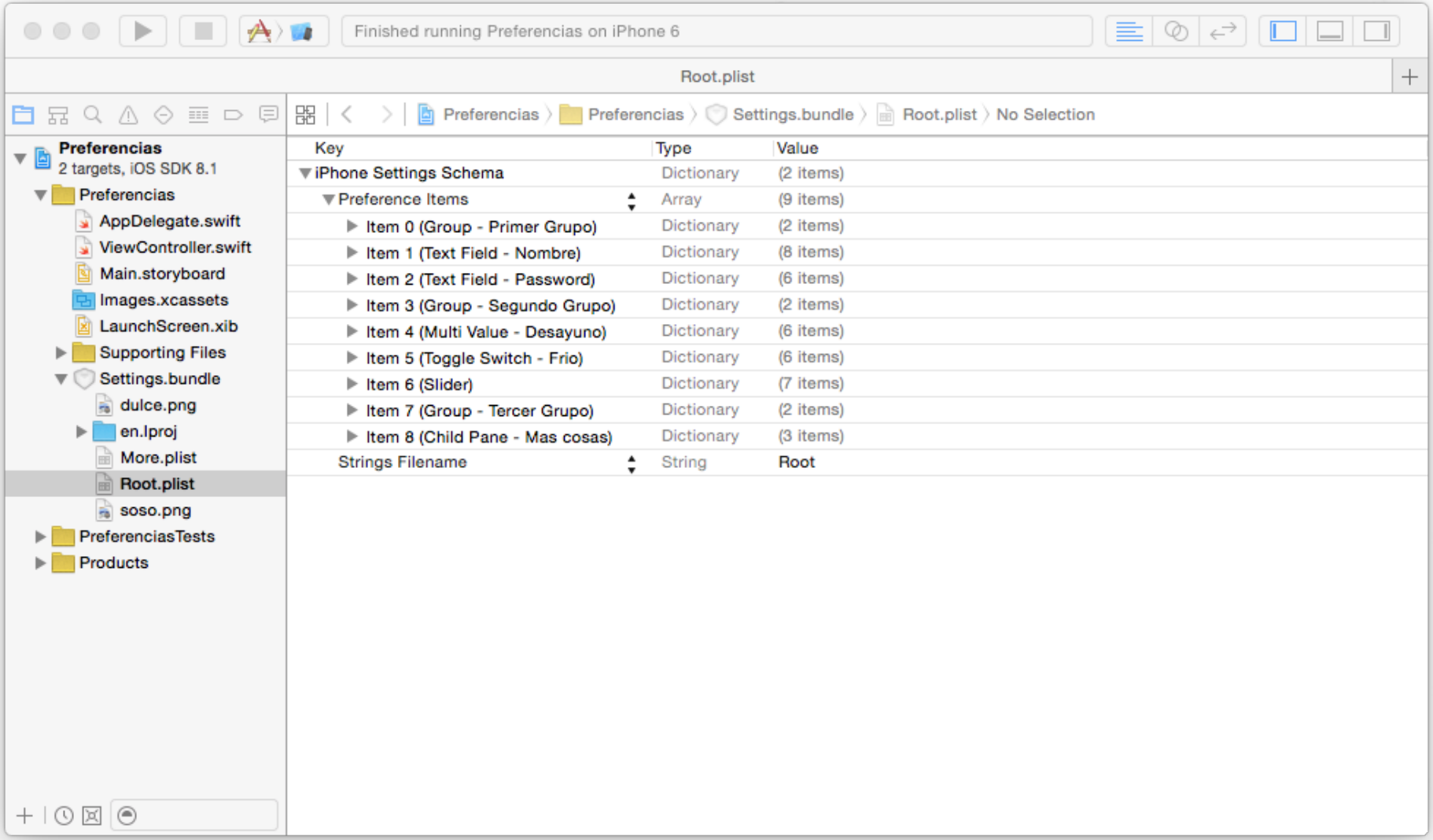
Acceso desde la Aplicación Ajustes

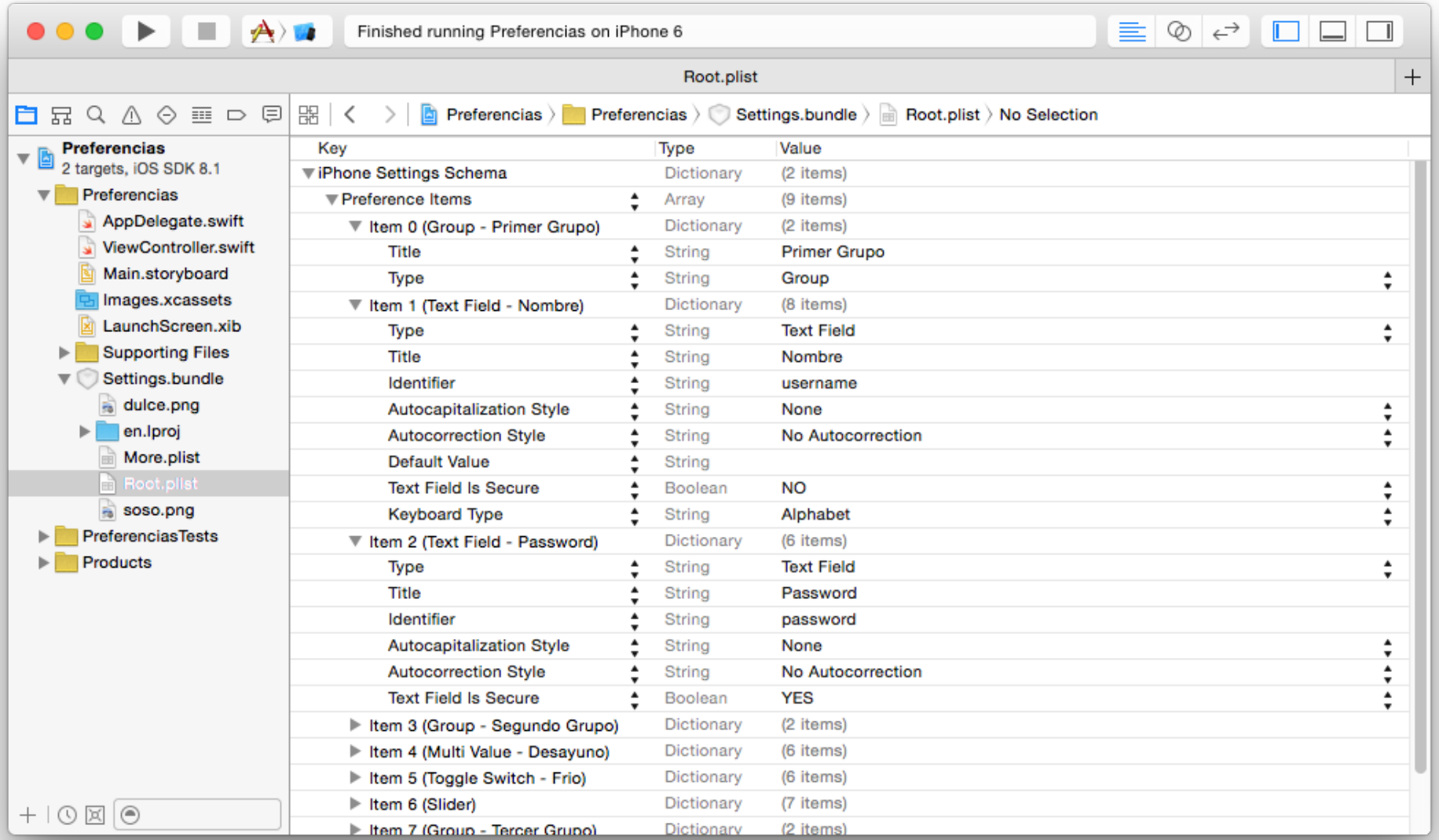
- Las preferencias de usuario pueden editarse desde la aplicación **Ajustes** (**Settings**).
  - Solo pueden editarse algunos tipos de valores.
- Para ello, hay que crear en nuestra aplicación un **Settings Bundle**:

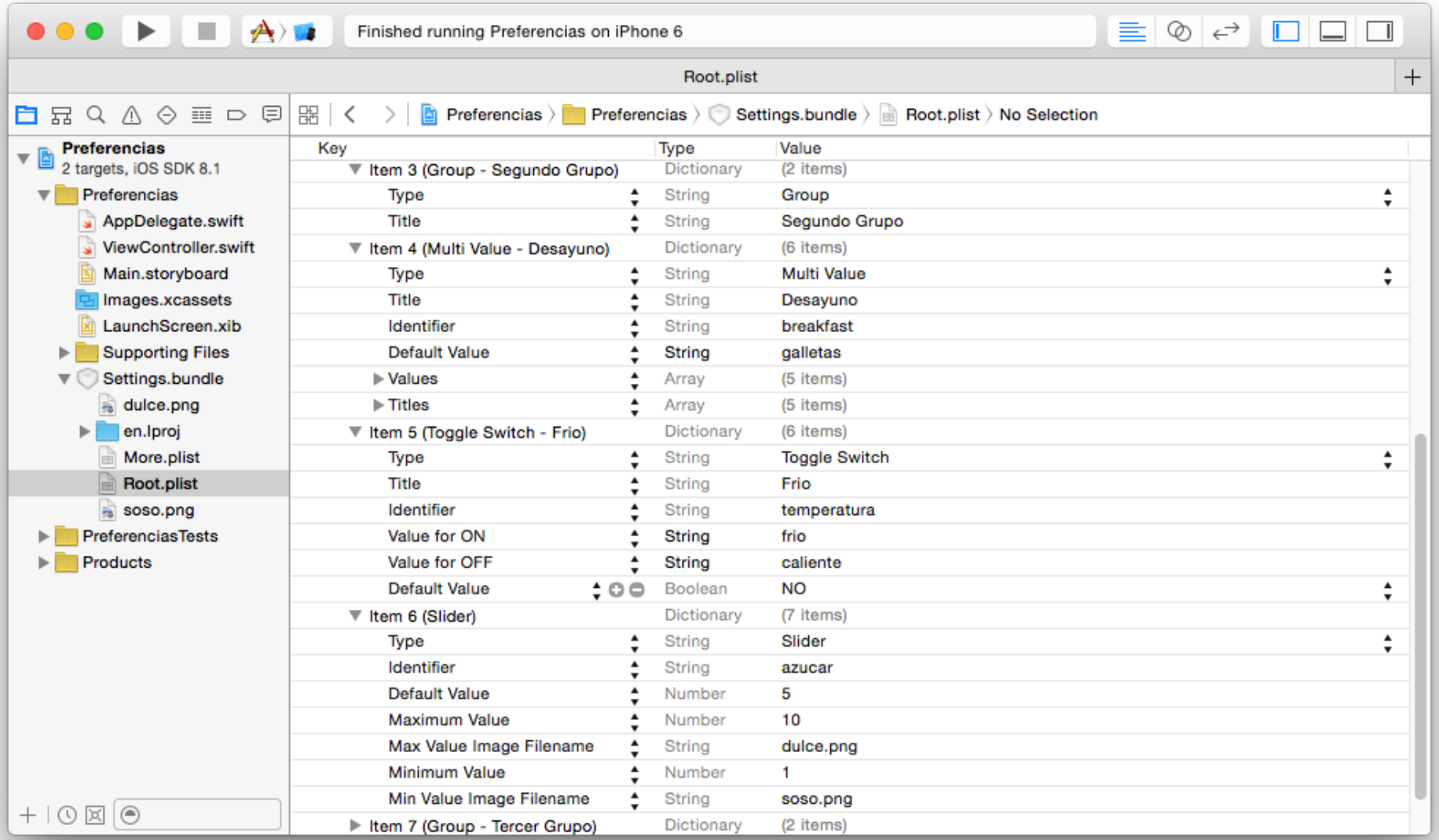
  New File > iOS > Resource > Settings Bundle
  - **Para manipular el contenido de este fichero hay que usar Finder.**
    - **Por ejemplo, para añadir un imagen.**
- **`Root.plist`**
  - define la primera vista de las preferencias.
- Para crear subvistas adicionales (nuevas pantallas) hay que crear nuevos ficheros de listas de propiedades.
  - Un fichero **plist** para cada pantalla adicional.
- Consultar la guía:
  - **Preferences and Settings Programming Guide: About Preferences and Settings**.

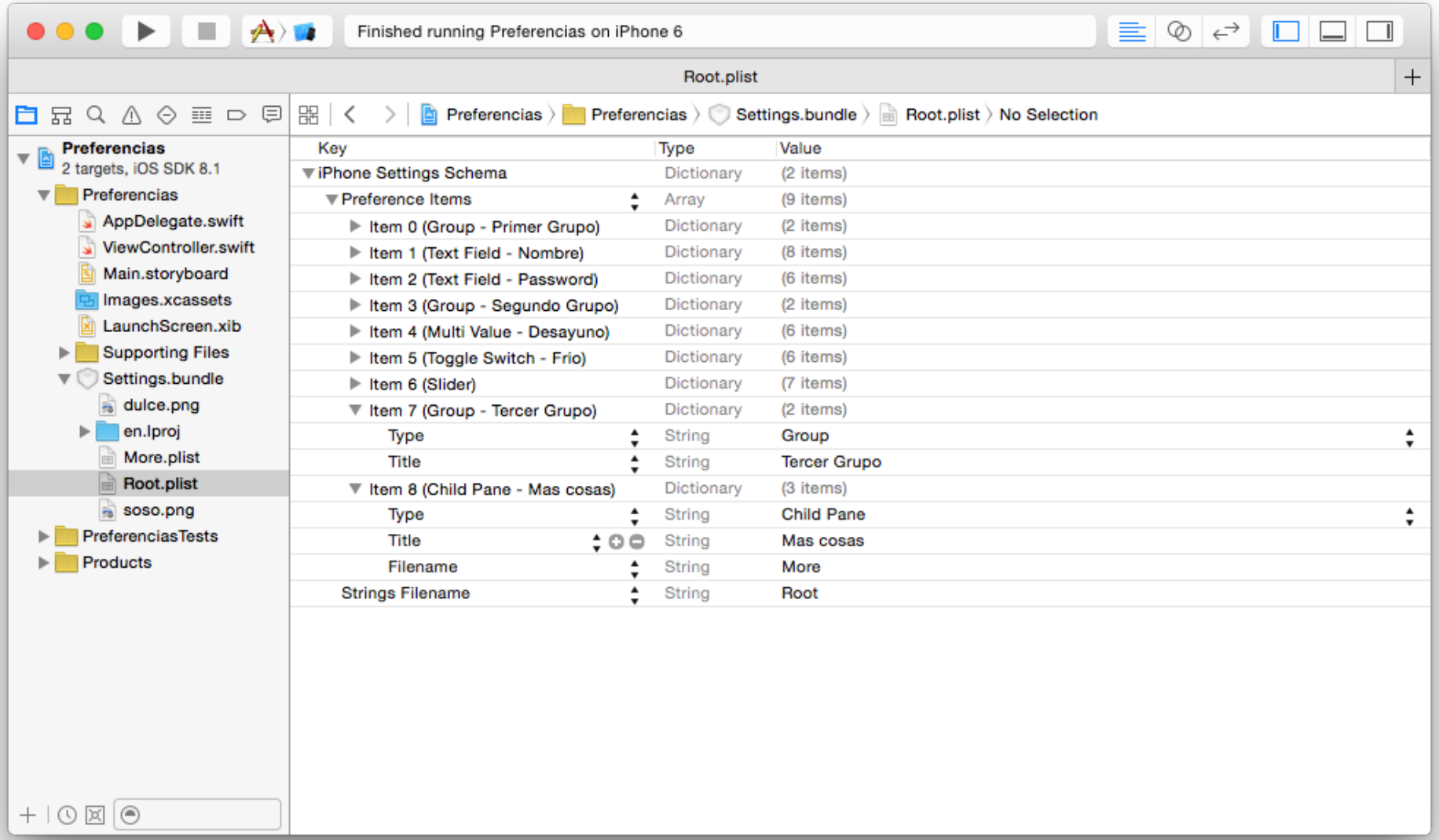


Root.plist

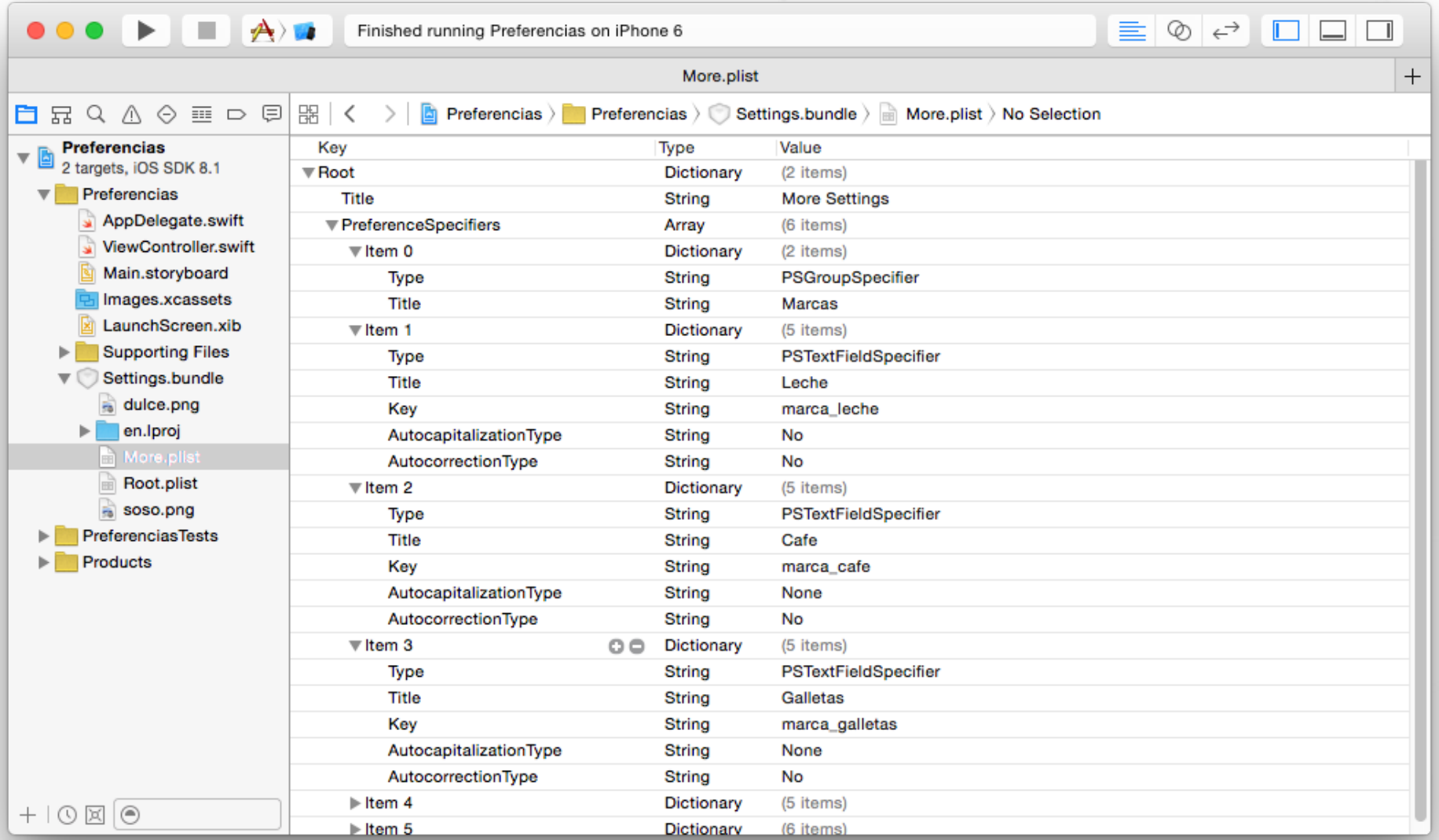


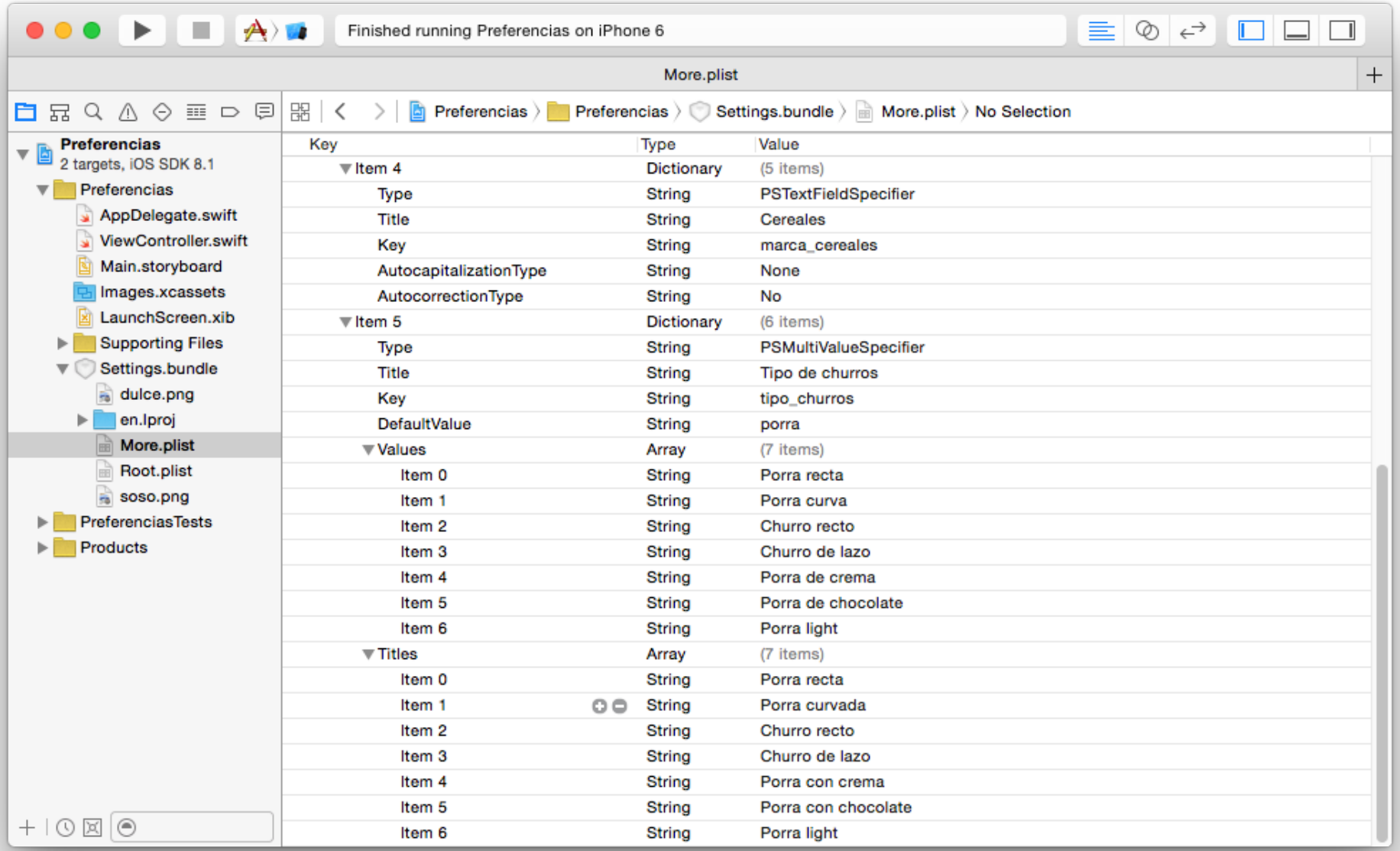






More.plist





Main Bundle

Bundle

- Un objeto **Bundle** 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: Bundle = Bundle.main

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

let url: URL? = bundle.url(forResource: "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(
                    .documentDirectory,
                    .userDomainMask,
                    true)
let docsPath: String = paths[0]
```

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 NSString

// Añadir al path:
let datosPath: NSString = docsPath.appendingPathComponent("g.dat") as NSString

// La extension de un fichero:
let ext = datosPath.pathExtension

// Nombre del fichero:
let fileName: NSString = datosPath.lastPathComponent as NSString

// Nombre del fichero sin extension:
let fileBasename = fileName.deletingPathExtension

// Directorio del fichero:
let basedir = datosPath.deletingLastPathComponent
```

```swift
// Directorio de documentos:
let paths = NSSearchPathForDirectoriesInDomains(.DocumentDirectory,
                                                .UserDomainMask, true)
let docsPath = paths[0]

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

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

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

// Escapar URL: URL encoding
let esc: String? = "a b c".addingPercentEscapes(
                              using: String.Encoding.utf8) // a%20b%20c
```

FileManager

- 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 desde el Bundle de la aplicación al directorio de documentos:

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

// Fichero origen
let bundle = Bundle.main
let origenURL = bundle.url(forResource: "pokemons", withExtension:"plist")

// Fichero destino (en el directorio de documentos)
let docsURLs = fm.urls(for: .DocumentDirectory, in: .UserDomainMask)
let docsURL = docsURLs[0]
let destinoURL = docsURL.appendingPathComponent("pokemons.plist")

// Copiar
do {
   try fm.copyItem(at: origenURL!, to: destinoURL)

   // Cargar el fichero copiado
   let dic = NSDictionary(contentsOf: destinoURL)
} catch let error {
   print(error)
}
```

Falta comprobar si los Optional son nil

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**, **Data**, **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)
```

PropertyListSerialization

- La clase **PropertyListSerialization** proporciona métodos de tipo para:
  - serializar una lista de propiedades en un **Data**.

```
class func data(fromPropertyList plist: Any,
                format: PropertyListSerialization PropertyListFormat,
                options opt: PropertyListSerialization.WriteOptions)
                                    throws -> Data
```

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

```
class func propertyList(from data: Data,
                options opt: PropertyListSerialization.WriteOptions = [],
                format: UnsafeMutablePointer<
                        PropertyListSerialization.PropertyListFormat>?)
                                    throws -> Any
```

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

```
func write(to url: URL, options: Data.WritingOptions = default) throws
init(contentsOf url: URL, options: Data.ReadingOptions = default) throws
. . .
```

- La clase **NSData** tiene más métodos para leer y escribir en ficheros.

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 encode(with aCoder: NSCoder)
    init?(coder aDecoder: NSCoder)
}
```

  - Nuestras clases deben adoptar este protocolo si queremos serializarlas para guardarlas en ficheros, guardarlas en las preferencias, transmitirlas por un socket, etc.

  - **NSCoder** es una clase abstracta.
    - Define métodos para convertir objetos en **Data**, e inversa.
    - Subclases:
      - **`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 encode(with aCoder: NSCoder) {

    // Codificar las propiedades de nuestra clase.
    aCoder.encode(nombre, forKey:"nombre")
    aCoder.encode(coche,  forKey:"vehiculo")
    aCoder.encode(edad,   forKey:"edad")
    aCoder.encode(otro,   forKey:"mas")
}
```

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

```
override func encode(with aCoder: NSCoder) {

    // La superclase es conforme a NSCoding,
    // por tanto, codificamos sus propiedades.
    super.encode(with: aCoder)

    // Codificar las propiedades de nuestra clase.
    aCoder.encode(nombre, forKey:"nombre")
    aCoder.encode(coche,  forKey:"vehiculo")
    aCoder.encode(edad,   forKey:"edad")
    aCoder.encode(otro,   forKey:"mas")
}
```

Método para Decodificar

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

```
required init?(coder aDecoder: NSCoder) {

  super.init()

  // Decodificar las propiedades de nuestra clase.
  nombre = aDecoder.decodeObject(forKey: "nombre") as? String
  coche  = aDecoder.decodeObject(forKey: "vehiculo") as? String
  edad   = aDecoder.decodeInteger(forKey: "edad")
  otro   = aDecoder.decodeFloat(forKey: "mas")
}
```

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

```
required init?(coder aDecoder: NSCoder) {

    // Decodificar las propiedades de nuestra superclase.
    super.init(coder: aDecoder)

    // Decodificar las propiedades de nuestra clase.
    nombre = aDecoder.decodeObject(forKey: "nombre") as? String
    coche  = aDecoder.decodeObject(forKey: "vehiculo") as? String
    edad   = aDecoder.decodeInteger(forKey: "edad")
    otro   = aDecoder.decodeFloat(forKey: "mas")
}
```

Ejemplo: Codificar y Salvar 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.encode(objeto1, forKey:"clave1")
archiver.encode(objeto2, forKey:"clave2")
archiver.encode(objeto3, forKey:"clave3")

// Ya he terminado de codificar:
archiver.finishEncoding()

// Salvar (el buffer NSMutableData) en un fichero:
let ok: Bool = data.write(toFile: "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.decodeObject(forKey: "clave1") as? LaClase
let objeto2 = unarchiver.decodeObject(forKey: "clave2") as? LaClase
let objeto3 = unarchiver.decodeObject(forKey: "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()
        if let phones = aDecoder.decodeObject(forKey: "phones")
                                        as? Dictionary<String, String> {
            self.phones = phones
        }
    }

    func encode(with aCoder: NSCoder) {
      aCoder.encode(phones, forKey: "phones")
    }
}
```

```
var agenda: AgendaModel!

loadAgenda()
agenda.phones["Peter"] = "123456789"
agenda.phones["Bob"] = "987654321"
saveAgenda()

func loadAgenda() {
    agenda = nil
    let def = UserDefaults.standard
    if let data = def.object(forKey: "phonebook") as? Data {
        agenda = NSKeyedUnarchiver.unarchiveObject(with: data)
                 as? AgendaModel
    }
    if agenda == nil {
        agenda = AgendaModel()
  }
}

func saveAgenda() {
    let data = NSKeyedArchiver.archivedData(withRootObject: agenda)
    let def = UserDefaults.standard
    def.set(data, forKey: "phonebook")
}
```

Saco un Data del UserDefaults y lo transformo en una Agenda

Transformo la Agenda en un Data, y el Data lo meto en UserDefaults.

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.unarchiveObject(withFile: path) as! UnaClase
```

- Nota: Estamos suponiendo que la clase **UnaClase** es conforme con **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);
```

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.