ETSIT UPM

dit UPM

Desarrollo de Apps para iOS

Persistencia

IWEB,LSWC 2013-2014
Santiago Pavón

ver: 2013.11.09

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

```
NSUserDefaults *def =
        [NSUserDefaults standardUserDefaults];
```

- Pueden guardarse combinaciones de:
  - NSData, NSString, NSNumber, NSDate, NSArray o NSDictionary.
- Se usa como un diccionario:
  - para obtener datos:
    - **`objectForKey:`** **`intForKey:`** **`boolForKey:`** …
  - para salvar datos:
    - **`setObject:forKey:`** **`setFloat:forKey:`** …
  - Antes de salir invocar **`synchronize`** para salvar datos de la cache.

Cuando cargar / salvar datos

- Cuando la pantalla va a mostrarse o ocultarse.
  - viewWillAppear: viewWillDisappear:
- Al cargar una pantalla en memoria.
  - viewDidLoad
- Cuando la aplicación vaya a terminar:
  - Apuntarse para recibir una notificación del centro de notificaciones (NSNotificationCenter).
- …

Ejemplo

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

```
- (void)viewDidLoad  {
    [super viewDidLoad];

    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    self.carModel = [defaults objectForKey:@"model"];
    self.carSpeed = [defaults floatForKey:@"speed"];
}
```

- Salvo las preferencias cuando la pantalla desaparece:

```
- (void) viewWillDisappear:(BOOL)animated {
    [super viewWillDisappear:animated];

    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    [defaults setObject:self.carModel forKey:@"model"];
    [defaults setFloat:self.carSpeed forKey:@"speed"];
    [defaults synchronize]; // Forzar la sincronización ahora
}
```

Ejemplo

- Usar notificaciones:
  - Cuando la aplicación va a terminar, se envía a **self** el mensaje **appWillTerminate**.

```
- (void)viewDidLoad
{
  UIApplication *app = [UIApplication sharedApplication];
  [[NSNotificationCenter defaultCenter]
      addObserver:self
         selector:@selector(appWillTerminate:)
             name:UIApplicationWillTerminateNotification
           object:app];
  ...
}
```

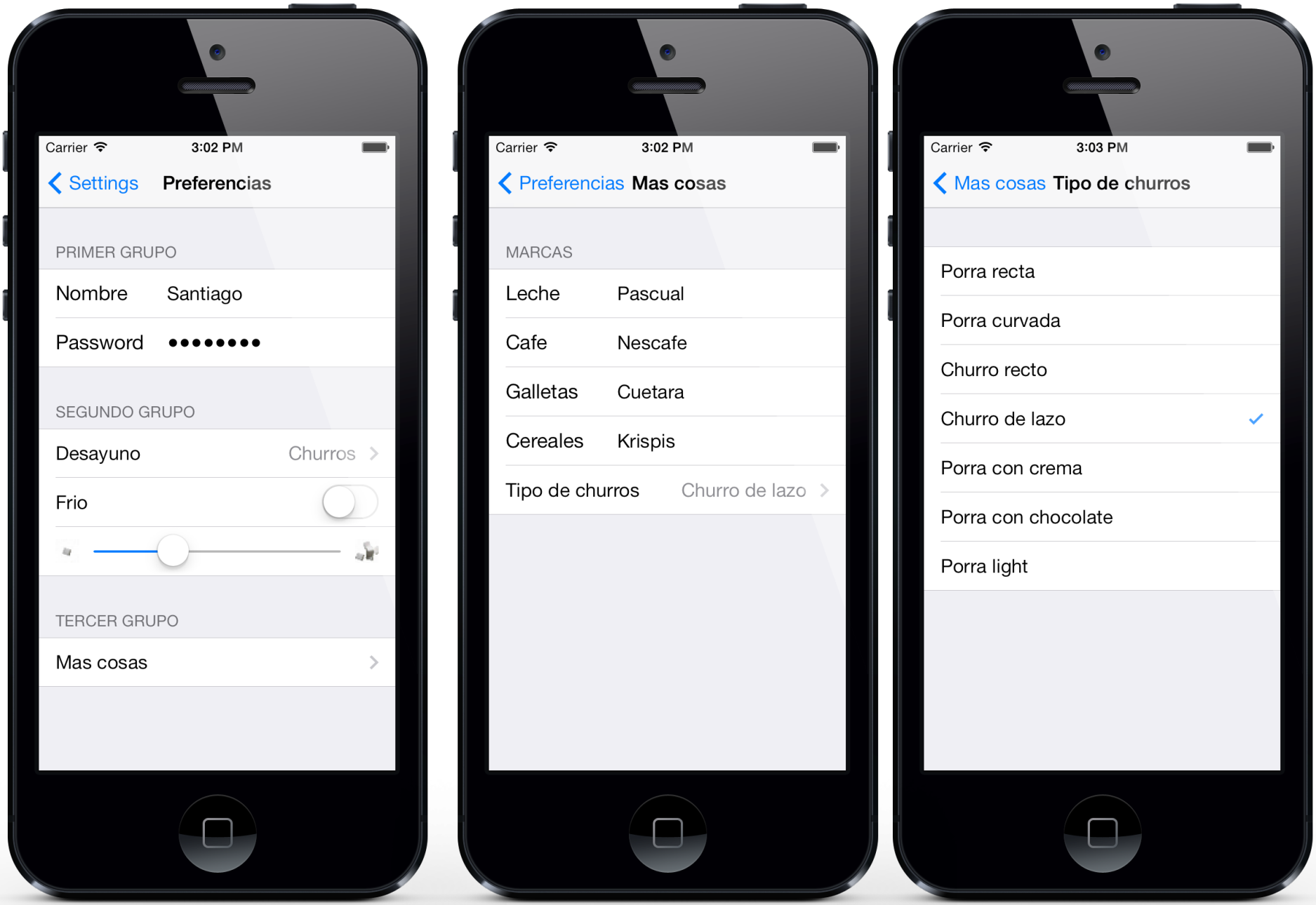
  - El método registrado para la notificación:

```
- (void)appWillTerminate:(NSNotification*)notification {
  // Salvar los datos aqui.
  [[NSUserDefaults standardUserDefaults] synchronize];
}
```

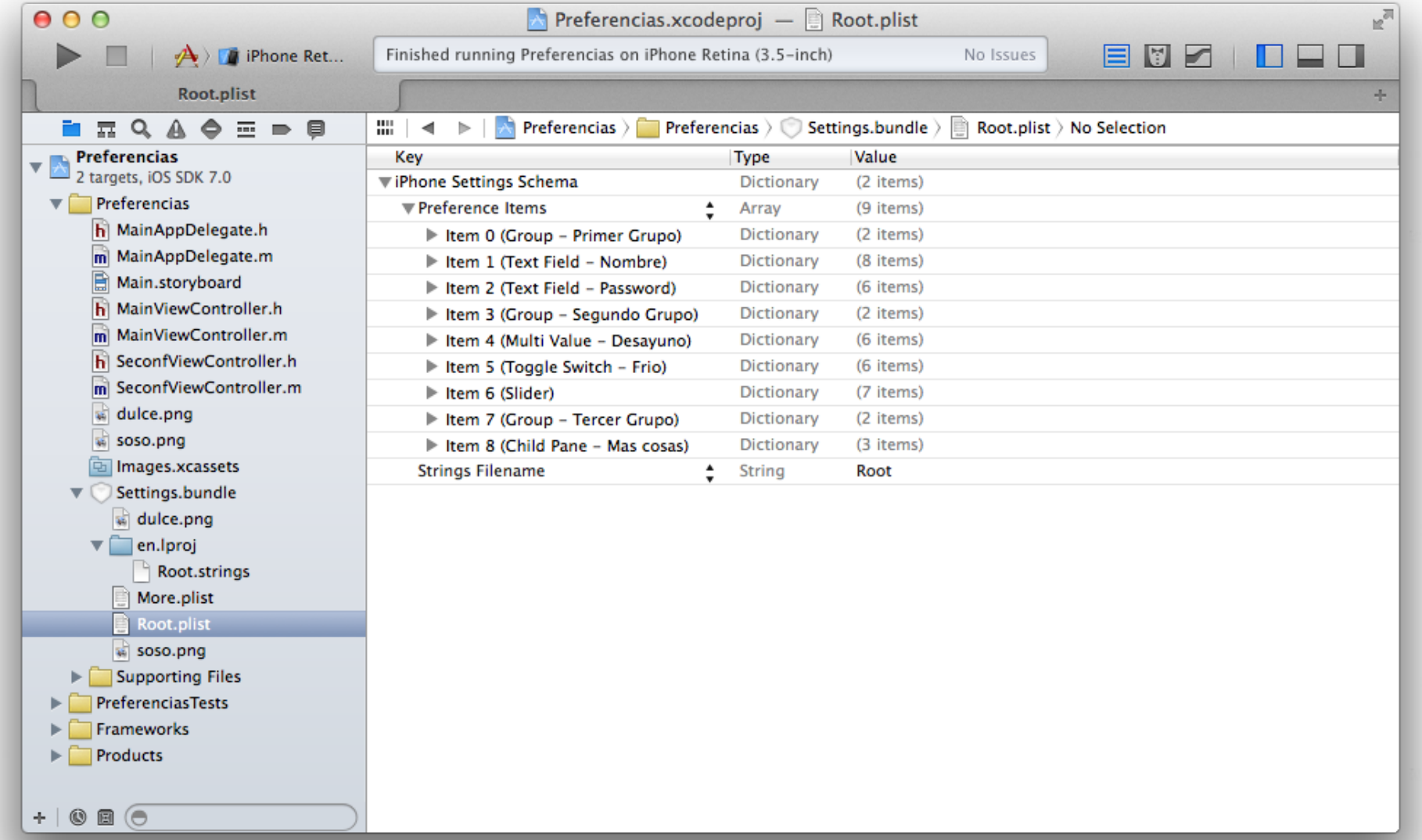
Acceso desde la Aplicación Ajustes

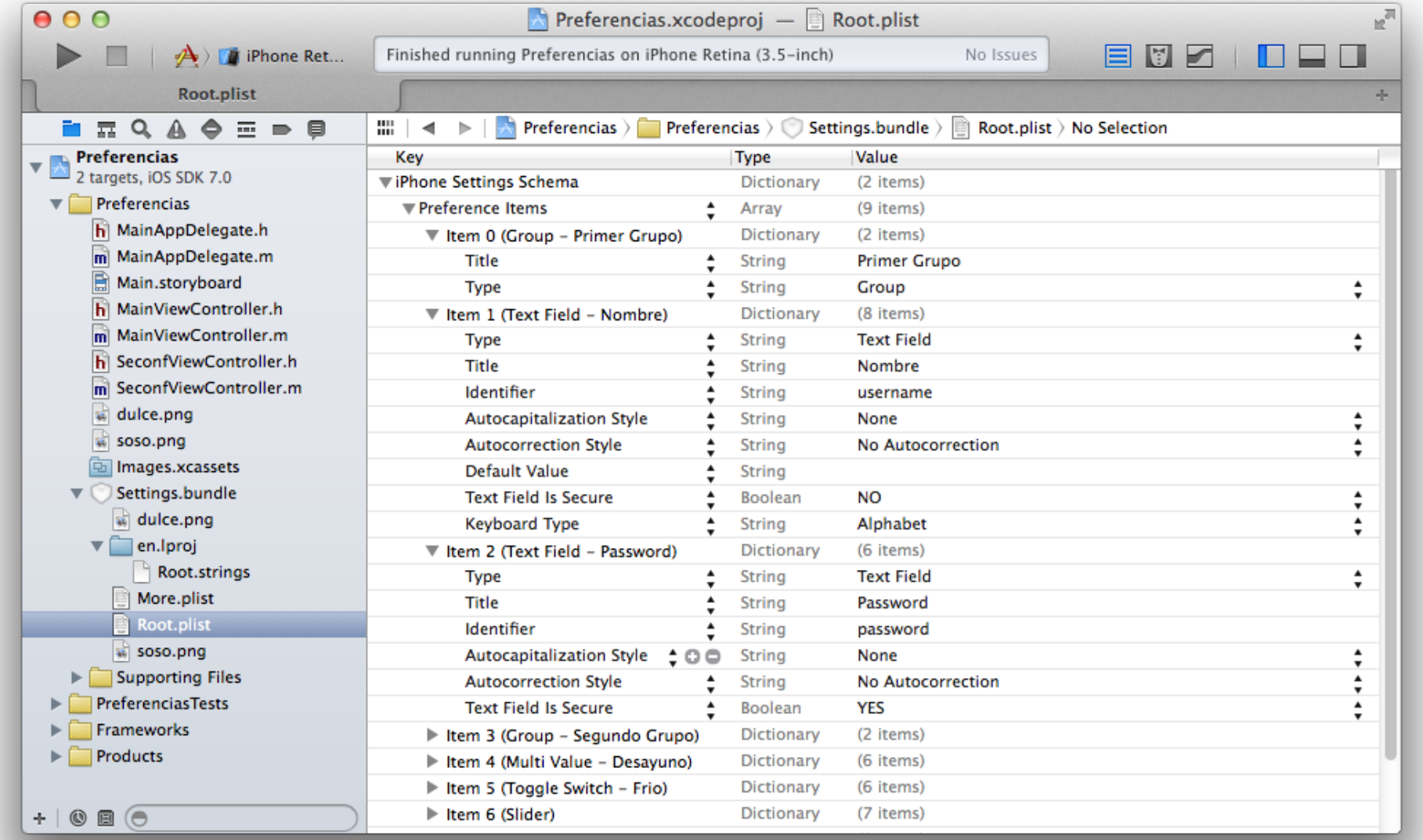
- Las preferencias de usuario pueden editarse desde la aplicación **Ajustes** (**Settings**).
  - Solo algunos tipos de valores.
- Para ello, hay que crear en nuestra aplicación un **Settings Bundle**:

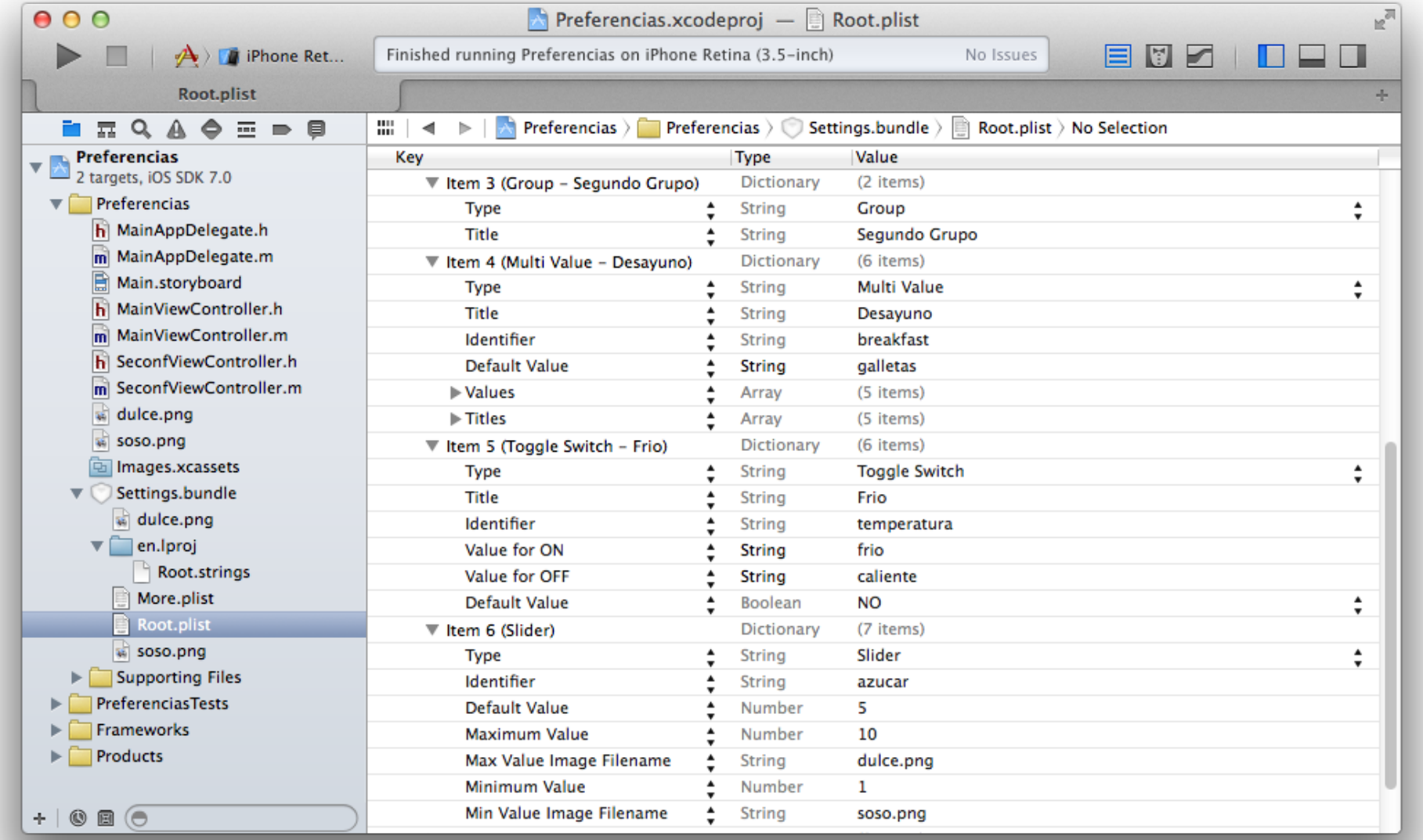
  New File > iOS > Resource > Settings Bundle
  - **Para manipular el contenido de este fichero hay que usar Finder.**
- **`Root.plist`**
  - define la primera vista de las preferencias.
- Para crear subvistas adicionales (nuevas pantallas) hay que crear nuevos ficheros de listas de propiedades.
  - Un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

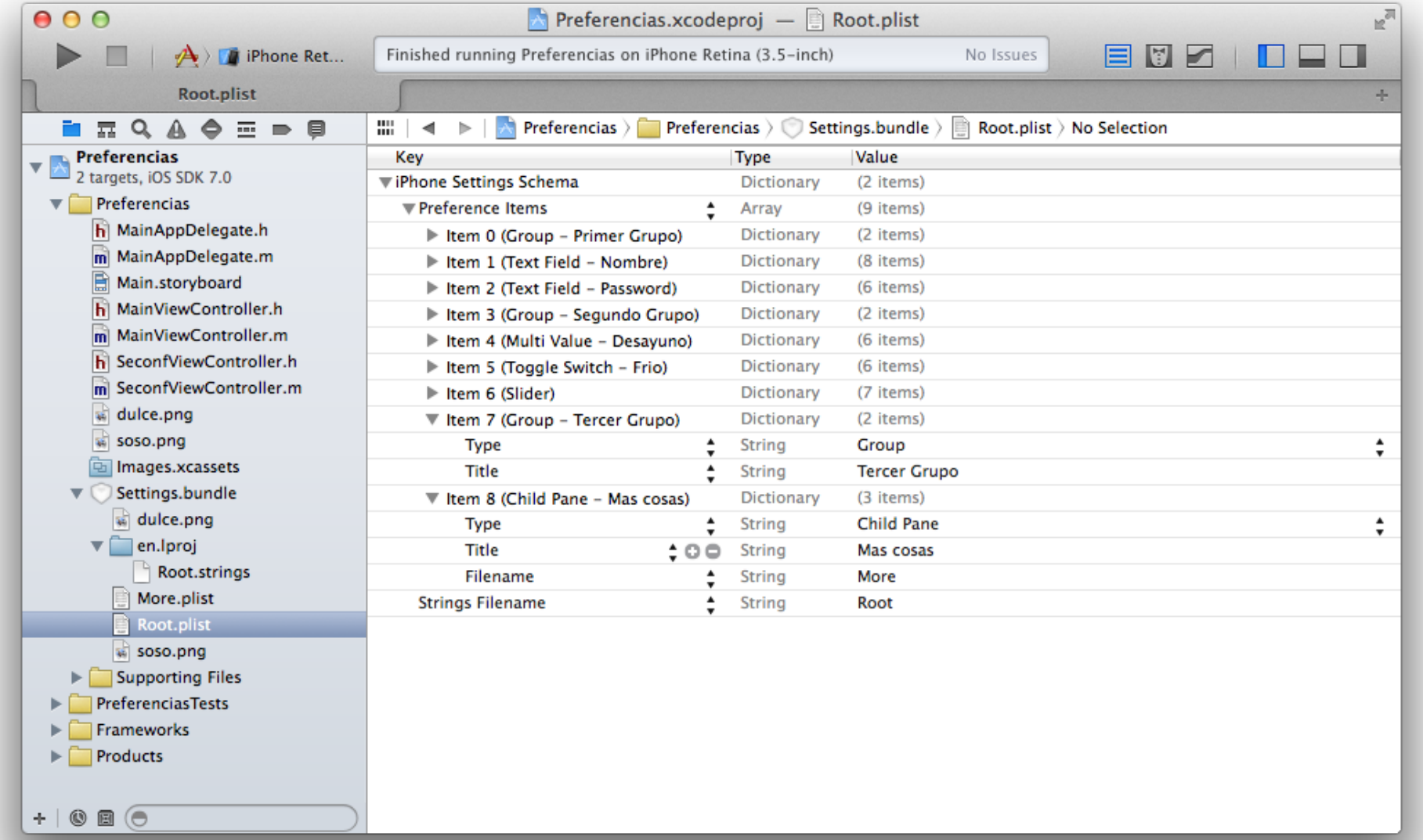


Root.plist

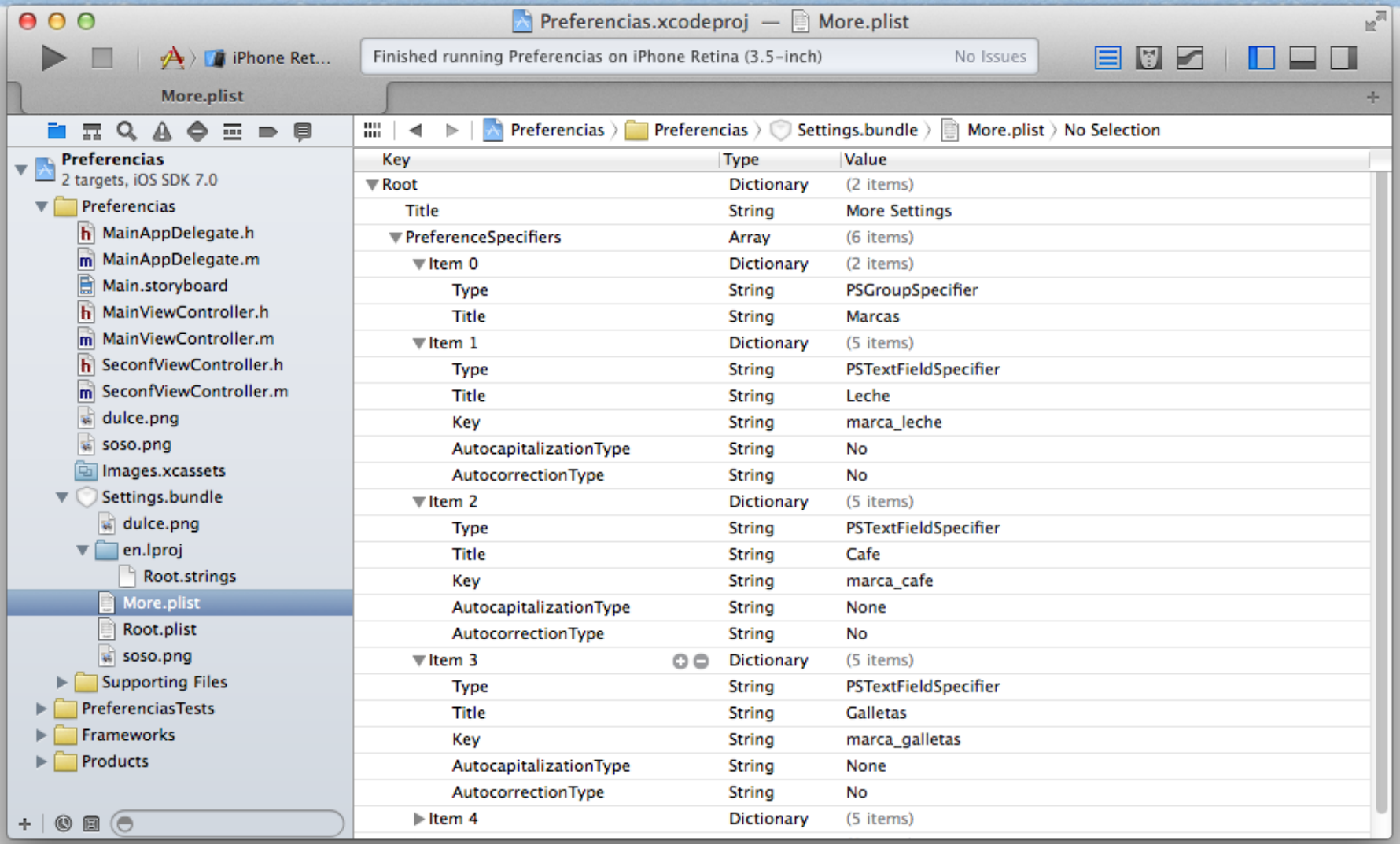


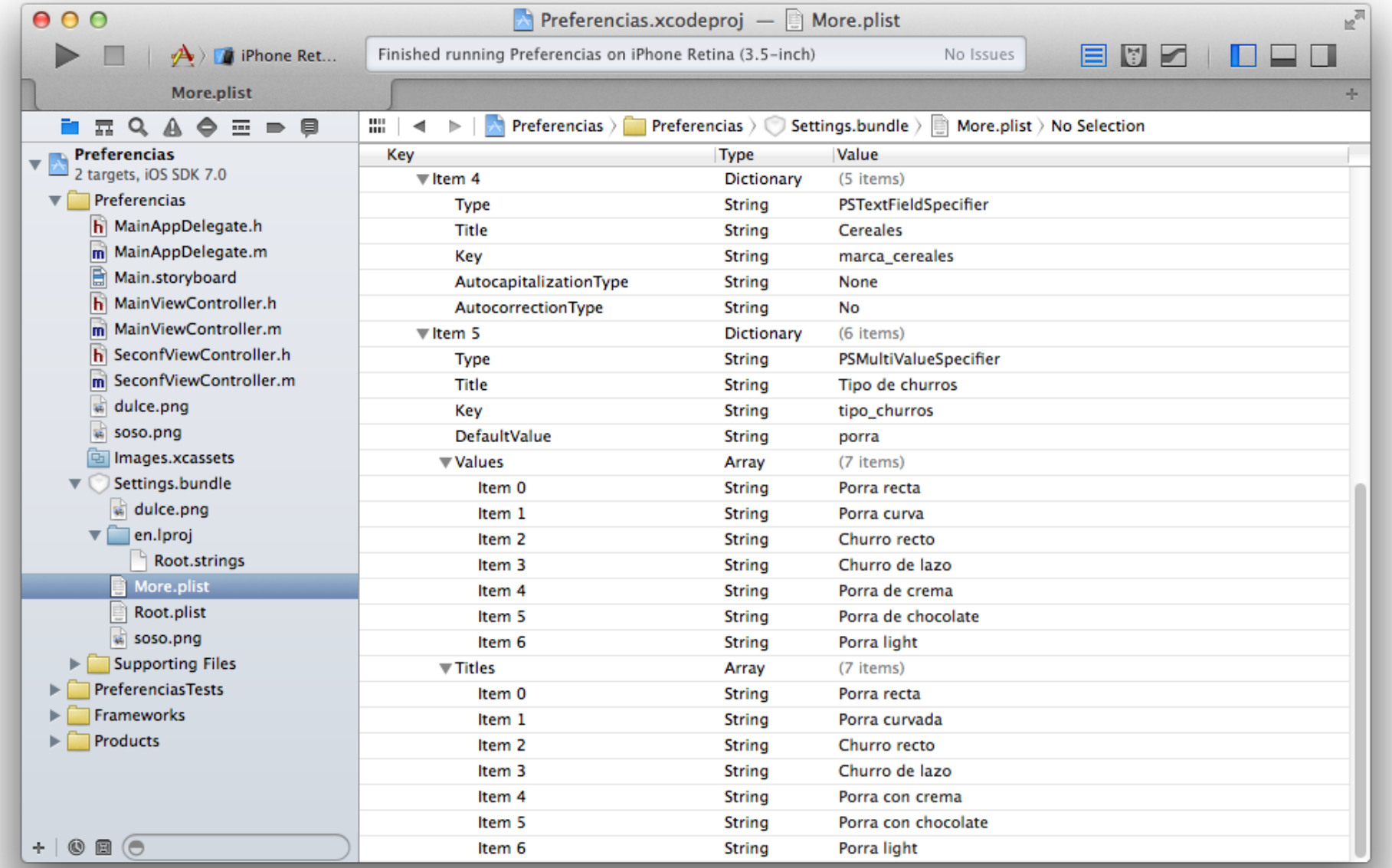






More.plist





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:

```
NSString *home = NSHomeDirectory();
```

- Directorio Temporal:

```
NSString *tmp = NSTemporaryDirectory();
```

- Directorio Documents:

```
NSArray *paths =
        NSSearchPathForDirectoriesInDomains(
                        NSDocumentDirectory,
                        NSUserDomainMask,YES);
NSString *docs = [paths objectAtIndex:0];
```

Manipular Rutas

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

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(
                     NSDocumentDirectory,NSUserDomainMask,YES);
NSString *docsPath = [paths objectAtIndex:0];

NSString *datosPath = [docsPath stringByAppendingPathComponent:@"g.dat"];

NSString *extension = [datosPath pathExtension];
NSString *fileName = [[datosPath lastPathComponent]
                                 stringByDeletingPathExtension];

NSString *basedir = [datosPath stringByDeletingLastPathComponent];

NSURL* docsURL = [NSURL fileURLWithPath:docsPath];

NSURL *datosURL = [docsURL URLByAppendingPathComponent:@"g.dat"];

NSString *absPath = [datosURL absoluteString];
```

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 puede modificarse.

Usar el Main Bundle

```
NSBundle *bundle = [NSBundle mainBundle];

NSString *path =[bundle pathForResource:@"pokemons"
                                 ofType:@“plist"];

NSURL *url =[bundle URLForResource:@"pokemons"
                     withExtension:@"plist"];

NSImage *img =[NSImage imageNamed:@”foto.png”];
```

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
NSFileManager *fm = [[NSFileManager alloc] init];

// fichero origen
NSBundle *bundle = [NSBundle mainBundle];
NSURL *origenURL = [bundle URLForResource:@“pokemons"
                            withExtension:@"plist"];
// destino
NSArray *docsURLs = [fm URLsForDirectory:NSDocumentDirectory
                               inDomains:NSUserDomainMask];
NSURL *docsURL = [docsURLs firstObject];
NSURL *destinoURL = [docsURL URLByAppendingPathComponent:@"pokemons.plist"];

// copiar el fichero
NSError *error;
[fm copyItemAtURL:origenURL
            toURL:destinoURL
            error:&error];

// cargar el fichero copiado
NSMutableDictionary *dic =
              [NSMutableDictionary dictionaryWithContentsOfURL:destinoURL];
```

Lista de Propiedades

- Es un dato formado por cualquier combinación de los tipos:
  - **NSArray**, **NSDictionary**, **NSData**, **NSString**, **NSNumber**, **NSDate**.
- Se pueden guardar y recuperar de ficheros **.plist**.

```
[miArray writeToFile:path1 atomically:YES];

NSDictionary *miDiccionario =
  [NSDictionary dictionaryWithContentsOfFile:path2];
```

NSPropertyListSerialization

- La clase **NSPropertyListSerialization** proporciona métodos para:
  - serializar una lista de propiedades en un **NSData**.

```
+ (NSData*) dataWithPropertyList:(id)plist
            format:(NSPropertyListFormat)format
            options:(NSPropertyListWriteOptions)opt
            error:(NSError**)error
```

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

```
+ (id) propertyListWithData:(NSData*)data
       options:(NSPropertyListReadOptions)opt
       format:(NSPropertyListFormat*)format
       error:(NSError**)error
```

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

```
- (BOOL)writeToURL:(NSURL*)aURL atomically:(BOOL)atomically
- (id)initWithContentsOfURL:(NSURL*)aURL
```

NSCoding

- El protocolo **NSCoding** declara los dos métodos que debe implementar una clase para que sus objetos puedan serializarse (encode) y des-serializarse (decode).
- Una clase debe ser conforme al protocolo **NSCoding**. para poder:
  - Guardar sus objetos en ficheros (encode),
  - Recuperar sus objetos guardados en ficheros (decode).
- Este protocolo se explica en las tranparencias de **Introducción a Objective-C**.

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);
```

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.