Understanding iOS Applications

Dr.-Ing. Thomas Springer
M. Sc. Martin Weißbach
Lifecycle of iOS Applications
Application States

- **Not Running**
  - not launched yet or terminated
- **Inactive**
  - running, but not receiving events
- **Active**
  - running and receiving events
Application States

- Background
  - in the background and executing code
- Suspended
  - in the background not running code
  - app remains in memory
• some apps have reasons to perform tasks in the background, hence can register for allowance to perform tasks in this background state
Launch the App in Foreground

- system creates process and main thread for application and calls main-function on main thread
- Xcode project’s default main-function directly hands control over to the UIKit framework

Launch Time
- User taps app icon
- main()
- UIApplicationMain()
- Load main UI file()
- First Initialization
- Restore UI state
- Final Initialization

Application Code
- application(_:willFinishLaunchingWithOptions:)
- application(_:didFinishLaunchingWithOptions:)
- Various Methods
- applicationDidBecomeActive(_:)

Running
- Activate the App
- Event Loop
- Handle Events
- Switch to different App
Launch the App in Background

- Applications can be launched in Background as well
- usually to handle Background events
- Launch cycle differs slightly from Foreground launch
  - app enters Background state after being in the Inactive state
Launch the App in Background

- handles event and may be suspended after the event is handled
- user interface files are loaded, but not displayed in the application window
When an App gets launched into the Background

• Location Apps
  • system receives a location update that meets the apps configured delivery criteria
  • device entered or exited a registered region

• Bluetooth Apps
  • app receives data from a connected peripheral
  • peripheral app receives commands from central device
When an App gets launched into the Background

• Audio Apps
  • audio framework needs the app to process data

• Background Download Apps
  • Push Notification with appropriate payload
  • system wakes app at opportunistic moments
  • NSURLSession background tasks finished
  • download initiated by a Newsstand app finishes
Move the App from Foreground into Background

- application delegate’s `applicationDidEnterBackground(_:)` method gets invoked
  - Prepare for App Snapshot
    - a picture of the application is taken which is used for transition animations
  - Save relevant App state information
  - Free up memory
    - e.g. cached data, image resources etc.

- Apple states that this method is granted approx. 5s of execution time [3]
  → if the time elapsed without a return from this method, the app gets killed and purged from memory
Move the App from Foreground into Background

- call

    `beginBackgroundTask(expirationHandler:)` in `applicationDidEnterBackground()` to get more execution time when transitioning to background.

**Application Code**

- `applicationWillResignActive(_:)`
- `applicationDidEnterBackground(_:)`
Temporary Interruption of the Application

- alert-based interruptions (e.g. phone calls) temporarily cause the app to lose control

- the app is transitioned to inactive state, so that the system can prompt the user on how to proceed

- depending on the user’s decision the app is moved to Background or continues being active in Foreground
Temporary Interruption of the Application

- when interrupted `applicationWillResignActive(_:)` of application delegate invoked
  - save data and state information
  - stop timers, periodic tasks or running queries
  - no initiation of new tasks
  - pause audio / video playback / games

- when application remains in the Foreground
  - `applicationDidBecomeActive(_:)` of application delegate invoked
    - all actions performed in `applicationWillResignActive(_:)` should be reversed
Move the App from Background into Foreground

- `applicationWillEnterForeground`: should undo actions performed in `applicationDidEnterBackground`:

```
applicationWillEnterForeground(_)
```

```
applicationDidEnterBackground()
```

```
applicationDidBecomeActive(_)
```

```
applicationWillEnterForeground("")
```

```
applicationDidEnterBackground()
```

```
applicationDidBecomeActive("")
```

```
Switch to this App
```

```
Background
```

```
Wake Up App
```

```
Foreground
```

```
Enter Foreground
```

```
Event Loop
```

```
Handle Events
```

Event Loop

Switch to this App

Application Code
• `applicationDidBecomeActive`: should perform the same activation tasks as it would at launch time
iOS Applications in Xcode
Xcode Project Structure

- Source Files
- Swift Files
- Cocoa Files
- C/C++ Files + Header
- Metal Files
- Test Files
• User Interface Files
  • Storyboards
  • Nibs
Xcode Project Structure

- Resource Files
  - Asset Catalogs
  - Property List Files
  - Sprite / Scene Kit Files
  - Strings Files (Localization)
# General Application Settings

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bundle versions string, short</td>
<td>String</td>
<td>1.0</td>
</tr>
<tr>
<td>Bundle identifier</td>
<td>String</td>
<td>$(PRODUCT_BUNDLE_IDENTIFIER)</td>
</tr>
<tr>
<td>InfoDictionary version</td>
<td>String</td>
<td>6.0</td>
</tr>
<tr>
<td>Main storyboard file base name</td>
<td>String</td>
<td>Main</td>
</tr>
<tr>
<td>Bundle version</td>
<td>String</td>
<td>1</td>
</tr>
<tr>
<td>Launch screen interface file base name</td>
<td>String</td>
<td>LaunchScreen</td>
</tr>
<tr>
<td>Executable file</td>
<td>String</td>
<td>$(EXECUTABLE_NAME)</td>
</tr>
<tr>
<td>Application requires iPhone environment</td>
<td>Boolean</td>
<td>YES</td>
</tr>
<tr>
<td>Bundle name</td>
<td>String</td>
<td>$(PRODUCT_NAME)</td>
</tr>
<tr>
<td>Supported Interface orientations</td>
<td>Array</td>
<td>(3 items)</td>
</tr>
<tr>
<td>Bundle creator OS Type code</td>
<td>String</td>
<td>???&gt;</td>
</tr>
<tr>
<td>Bundle OS Type code</td>
<td>String</td>
<td>APPL</td>
</tr>
<tr>
<td>Localization native development region</td>
<td>String</td>
<td>en</td>
</tr>
<tr>
<td>Required device capabilities</td>
<td>Array</td>
<td>(1 item)</td>
</tr>
</tbody>
</table>

- **Document Types (0)**
- **Exported UTIs (0)**
- **Imported UTIs (0)**
- **URL Types (0)**
Interface Builder
• Xcode tool to design user interfaces – WYSIWYG
  • graphical editor
  • Nibs and Storyboards
• **Scene**
  • Corresponds to a single view controller and its views
  • may contain arbitrary many other proxy objects

• **Segue**
  • Manages transition between to scenes

• **Dock**
  • Make action and outlet connections between view and controller
Storyboard — Segues

• show — push a new scene on the view stack

• show detail — replace the detail view on a split screen or push a new scene on the view stack otherwise

• modally — display a modal view above all other views

• popover — display a popover view or a modal view in case the screen size is too small
Views
• File’s Owner
  • placeholder object in the Interface Builder document
  • connections to and from the File’s Owner in the document are reestablished when the view is being instantiated

• First Responder
  • target of an action whenever the action is to be sent to the object currently being first responder

• Proxy Object
  • arbitrary object that is loaded when the Scene is loaded
  • e.g., to handle UI inputs or provide additional functionalities
General — Responder Chain

Scene Lifecycle

- awakeFromNib
- viewDidLoad
- viewWillAppear
-DidAppear
- viewWillAppear
- viewWillDisappear
- viewDidDisappear
- deinit
Scene Lifecycle

- awakeFromNib
- viewDidLoad
- viewWillAppear
- viewDidAppear
- viewWillAppear
- viewWillDisappear
- viewDidDisappear
- deinit

only invoked when directly loaded from nib-loading code; not if a File’s Owner, First Responder or any other Proxy object
Scene Lifecycle

- awakeFromNib
- viewDidLoad
- viewWillAppear
- viewDidAppear
- viewWillDisappear
- viewDidDisappear
- deinit

used to perform additional configurations of the view, e.g. {en,dis}abling inputs
A Storyboard Scene Example

Storyboard Scene

```swift
class ViewController: UIViewController {

    private var touchCounter = 0

    @IBOutlet weak var helloWorldLabel: UILabel!

    @IBOutlet weak var touchMeButton: UIButton!

    @IBAction func TouchMeAction(_ sender: Any) {
        touchCounter = touchCounter + 1

        helloWorldLabel.text = (touchCounter == 1)
            ? "I was touched once." : "I was touched \n            (touchCounter) times."
    }
}
```
Implicitly Unwrapped Optionals

- Object graph of scene is archived in the application bundle
- When the scene is needed for display, the object graph is unarchived → objects are instantiated
- When the view controller is instantiated, the concrete UIWidgets have not yet been initialized and have to be `nil`, hence, an optional is required
- When the initialized scene is handed over to the application, all UIWidgets were initialized and references are kept by the system → implicitly unwrapped optional, because UIWidgets can never be `nil` again.
Design Pattern
- target: object
  - action: invokeMethod
  - event: clicked

- button

:object

- Target-Action

- an object stores information on what method to invoke on another object when a given event occurs

- click!
Model-View-Controller

Controller

Model

View
Model-View-Controller

Controller

Model

View
class ExampleViewController: UITableViewController {
    func addItem(_ item: Item) {
        /**Implementation */
    }
}
- **UITableView**
  - Custom behavior for touch-events desired
  - Single-/Multiple selection allowed in different tables
- **Subclass UITableView**
  - Views are actually not reusable anymore
  - You need more than one *.nib* file for a single table view

- **Delegation**
You “tell” the table view “who” has the knowledge on how to react on events

Tasks and responsibilities are forwarded – delegated – to a 3rd class

UITableView is reusable now

One table view can be react differently depending on the set delegate
Delegation

- Height of the View
- Position on the screen
- …

Controller

Delegate

- Is multiple Selection allowed?
- A row was selected
- There was a slide gesture

View
• Often used for asynchronous operations as well
  • Delegate handles behavior when operation finished successfully or unsuccessfully
• Event based mechanisms
• and (many) other use cases