Entwicklung mobiler Anwendungen

WPF in den Informatik-Studiengängen der Technischen Hochschule Köln

Wintersemester 2015/2016

iOS-Version

Über Mich

Dipl.-Inform. (FH) Daniel Klein

Studium in Gummersbach, beendet Mai 2007

Seit 2002 Software-Entwicklung für mobile Plattformen (u.a. Java ME, Android, iOS, Widgets)

3 Jahre Lead Developer der HRS iPhone-App

Derzeit Senior Software-Entwickler iOS bei der Chefkoch GmbH (chefkoch.de)

Gelegentlicher Autor (Fachartikel, Blog, Buch (TBA))

Allgemeine Informationen

Themen der Veranstaltung

- Theorie
  - Geschichte & Plattformen
  - iOS: Grundlagen, Objective-C, Tools
  - User Interfaces
  - Ausgewählte APIs
- Praxis
  - Entwicklung mit Xcode
  - Übungen

Geplanter Ablauf

Blockveranstaltung

- Vorlesung & Übung gemischt

Projekt

Projektpräsentation

Blockveranstaltung

Termin 4. - 6. Januar 2016

Vorlesung und Übungen werden gemischt

Ungefähr Blöcke von je 2 Stunden

Regelmäßige, kurze Pausen, Mittagspause
Erinnern Sie mich bitte! 😴

Anwesenheitspflicht

Übungen

Ähnlich wie Praktika

Aufgabe/Thema wird vorgegeben

Jedoch keine "Hausaufgaben" und Abnahmen

Sie beschäftigen sich mit dem Thema, ich unterstütze Sie

Projekt

Selbst gewähltes Thema unter Absprache mit mir, zur Vermeidung von Dubletten und zur realistischen Aufwandsabschätzung

Erarbeitung

- Zeit bis ungefähr Ende des Semesters
- Mail-Support von mir bei anderweitig nicht lösbaren Fragen

2er Teams (nur) bei entspr. Projekt-Umfang!

Projektpräsentation

Termin: tbd
Ende des Semesters oder
vorlesungsfreie Zeit

Max. 30 Minuten pro Team

Ein paar Folien zur Erläuterung

Live-Demonstration der Anwendung mit
Simulator oder eigenen Endgeräten

Anwesenheitspflicht!

Was sind mobile Anwendungen?

Once upon an Oak...

Once upon an Oak...

1992: Sun stellt als Ergebnis des "Green Project" ein mobiles Endgerät Namens "*7" (Star Seven) vor.

PDA mit Funkanbindung

Programmiersprache "Oak" wurde speziell dafür entwickelt.

Once upon an Oak...

Das Green Project (und damit *7) scheiterte

Aber in Zeiten des gerade beginnenden Internet-Hypes wurde schnell der Nutzen einer modernen Programmiersprache erkannt, die interpretiert auf jeder Art von Gerät laufen konnte, nur eine VM vorausgesetzt. (Ähnlich wie HTML)

Aus Oak wurde Java...

Mobile Java

Erste experimentelle JVM für mobile Geräte: KVM (Kilobyte Virtual Machine) aus dem "Spotless"-Projekt für die Palm PDAs von US Robotics/3COM

Zur JavaOne 1999 wurde die Java 2 Micro Edition (J2ME) vorgestellt

KVM wurde Basis für CLDC 1.0 und damit J2ME

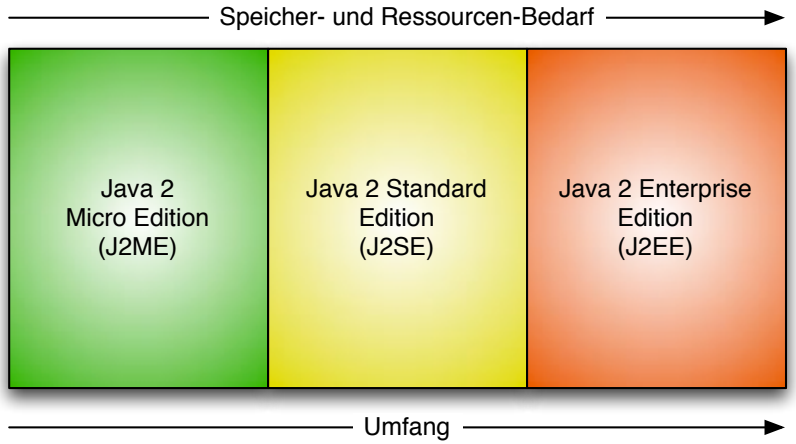
Java Editionen

Neben der Java 2 Micro Edition (J2ME) wurden ebenfalls die anderen, heute bekannten Java Editionen vorgestellt.

Java 2 Standard Edition (J2SE)

Java 2 Enterprise Edition (J2EE)

Alle basierend auf Java 2 v1.2



Später wurde die 2 von Java 2 fallen gelassen -> Aus J2ME wurde JME

Java ME

JME wurde ein phänomenaler Erfolg

Heute gibt es ca. 2,3 Milliarden Java ME-fähige Mobiltelefone die aktiv verwendet werden

Nur leider konnte die Plattform beim Stand der Technik nicht mithalten

Letzte aktuelle Configuration & Profile (feste Basisvorgaben) erschien ca. 2002

Seitdem (teilweise aus politischen Gründen) Stillstand

Massenweise optionale Erweiterungen wurden spezifiziert, doch nie zu einem neuen Standard kombiniert

Die Fragmentierungs-Hölle

Heute gibt es tausende verschiedene JME-fähige Mobiltelefone

Doch alle unterstützen unterschiedliche optionale Erweiterungen (JSRs)

Dazu kommen große Unterschiede bei der verwendeten Hardware (z.B. Art der verwendeten Bildschirme, Eingabemethoden, Geschwindigkeit)

Spiele-Entwicklung
Damals vs. Heute

Damals

Ein Entwickler, ein Grafiker, ein Tester

Unterstützung von zwei, drei Hersteller-Plattformen (und damit ca. ein/zwei Dutzend Geräte)

Heute

Einige Entwickler, einige Grafiker, dutzende Tester

Anpassungen im Code für nahezu jedes Gerät

Indische Testcenter testen auf hunderten von Endgeräten

Unterstützung der Top 10 Geräte der Netzbetreiber

Distributionskanal

Distribution von Anwendungen (hauptsächlich Spiele) findet in der Regel über Netzbetreiber statt

Diese kooperieren nur mit großen, etablierten Anbietern

Alternative: „Abo-Anbieter“ (Abzocke) -> siehe Musiksender

Große Probleme für kleine, innovative Teams sich zu etablieren, daher keine florierende Entwicklerszene

Die Zeit verging...

JME alterte

Sun war nicht in der Lage (oder daran interessiert?) die Probleme zu lösen

JavaFX war ein Versuch zwei Fliegen mit einer Klappe zu schlagen ein Konkurrent für Adobes Flash zu sein und JME eine neue Basis zu bieten

Der Ruf nach einer Fusion von JME und JSE wurde immer lauter...

... aber nichts passierte ...

Währenddessen in Cupertino...

Apple verkaufte als Mobile-Neuling schlagartig Millionen von iPhones und (später) iPads

Integrierte Verkaufsplattform mit günstigen Konditionen für alle

Nach drei Jahren > 700.000 „Apps“, ca. 20 Milliarden Downloads

Trotz „ungewöhnlicher“ technischer Grundlage (Objective-C, Mac-only)

... und Menlo Park

Googles Android entwickelte sich schnell zum potenten Gegenspieler für Apple

„Offene“ Philosophie

Integrierte Verkaufsplattform

Nach drei Jahren > 300.000 Apps, 10 Milliarden Downloads

Multi-Vendor-Strategy

Basierend auf Java-Technologie

Übersicht aktuelle mobile Plattformen

Java ME

Heute spielt JME trotz seiner großen Verbreitung quasi keine Rolle mehr

MIDP 3.0 wurde nie beschlossen

JavaFX floppte (mobile Version kam nie)

November 2012: Neue JSRs für CLDC 8 und Java ME Embedded Profile (EP)

Bisher keine wesentlichen Ergebnisse

Apple/iOS

Klare Hardware-Linie: iPhones, iPads, iPod touch

Kostenlose Firmware-Updates für mindestens zwei Jahre

Derzeit interessanteste Plattform für Entwickler (Monetarisierungschancen, kaufbereite Nutzer)

... und Marketing (Reichweite, Zielgruppe)

Aber: Eine eigene Welt. Objective-C, Cocoa, Xcode, Mac-only-Entwicklung

Android

Chaotisches Hardware-Umfeld: Dutzende Geräte von verschiedensten Herstellern (Qual der Wahl vs. Wahlfreiheit)

Firmware-Updates problematisch (da vom Hersteller abhängig, Interessen)

Motivation für Software zu bezahlen (noch?) deutlich niedriger

Nach Marktanteil vorne, Nutzungszahlen aber niedriger

Für Java-Entwickler einfacher einzusteigen

Microsoft

Alter Haase im Geschäft: Windows CE
Aber keine Konkurrenz für moderne Systeme

MS hat moderne Smartphones verschlafen,
befindet sich mit niemandem im Kampf um Platz 3

Basiert auf Windows CE, .NET, Silverlight, XNA

Klar nicht so weit/erfolgreich wie Apple & Google

Marktanteil im niedrigen einstelligen Bereich

Nokia Smartphone-Sparte übernommen

Nokia

Der einstige Platzhirsch verlor zeitweise Marktanteile im zweistelligen Prozentbereich pro Quartal

JME & Symbian sind nicht mehr Konkurrenzfähig

Symbian fristet Feature-Phone Dasein

MeeGo: Ein Release, dann eingestellt

Zurück zu Gummistiefeln?

Samsung

Samsung hat sich als DER Platzhirsch unter den Android-Anbietern etabliert

Galaxy-Reihe wurde als starke, eigenständige Marke etabliert

Google kann nur hoffen, dass Samsung im Android-Boot bleibt. (Fork?)

Alles nur geklaut?

Weitere Plattformen

BlackBerry: BB10 ist verfügbar, spielt aber keine Rolle.

Palm: Ist bei HP eines grausamen Todes gestorben. webOS verkauft an LG. (Für TVs!)

Verschiedene Open Source-Projekte
Erfolg allgemein Zweifelhaft, da die breite Unterstützung von Hardware-Herstellern fehlt.
Alles ist möglich, aber besser nicht drauf warten...

Grundlagen

Was ist iOS?

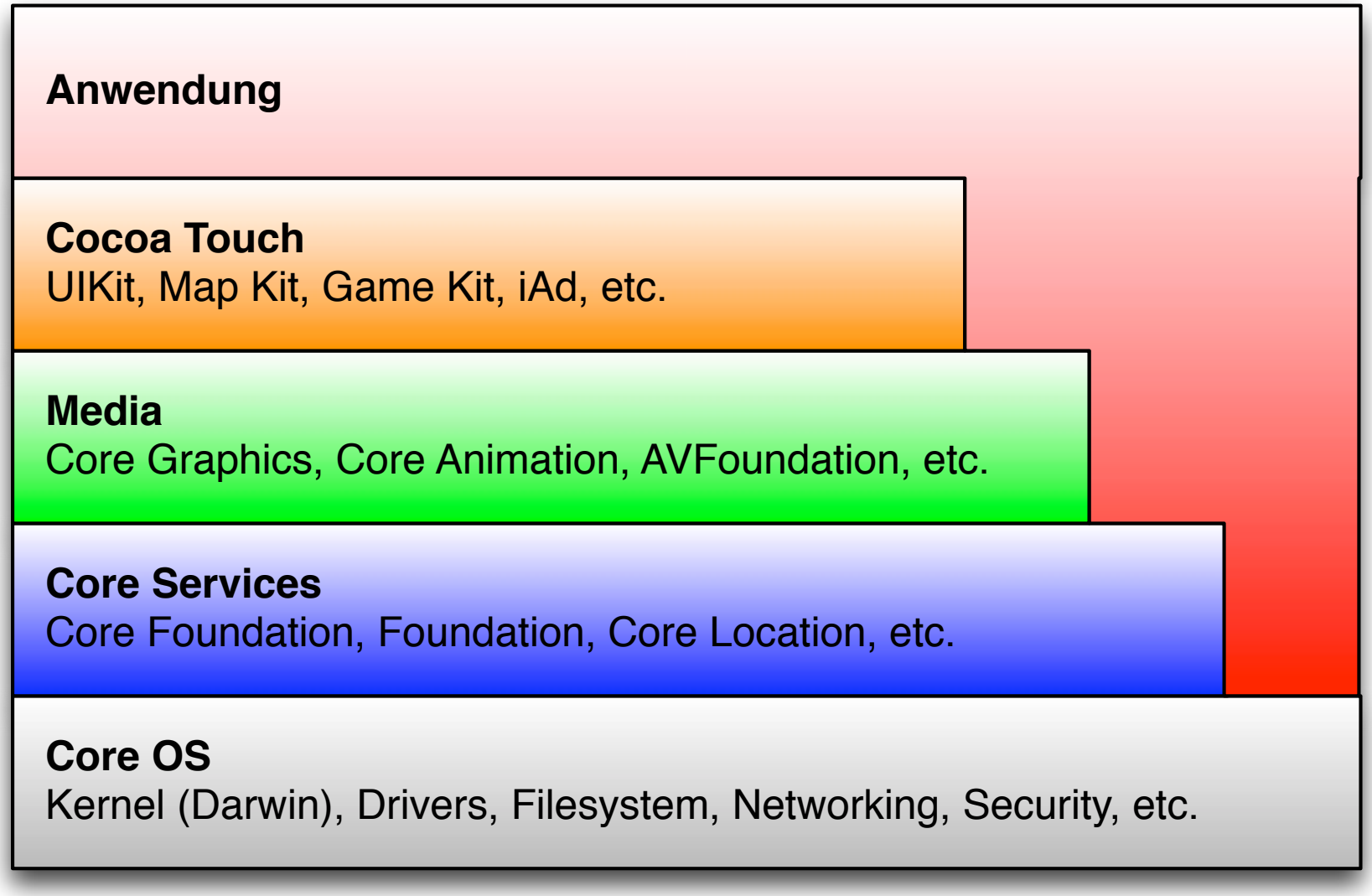
Vollwertiges, modernes OS mit den üblichen Features wie Prozessen, Threads, virtueller Speicher (kein Paging!), Sicherheit, Stromsparfunktionen, etc.

Basiert auf OS X, viele APIs und Konzepte sind identisch. Teilweise aber auch neue APIs: UIKit, Core Location, etc.

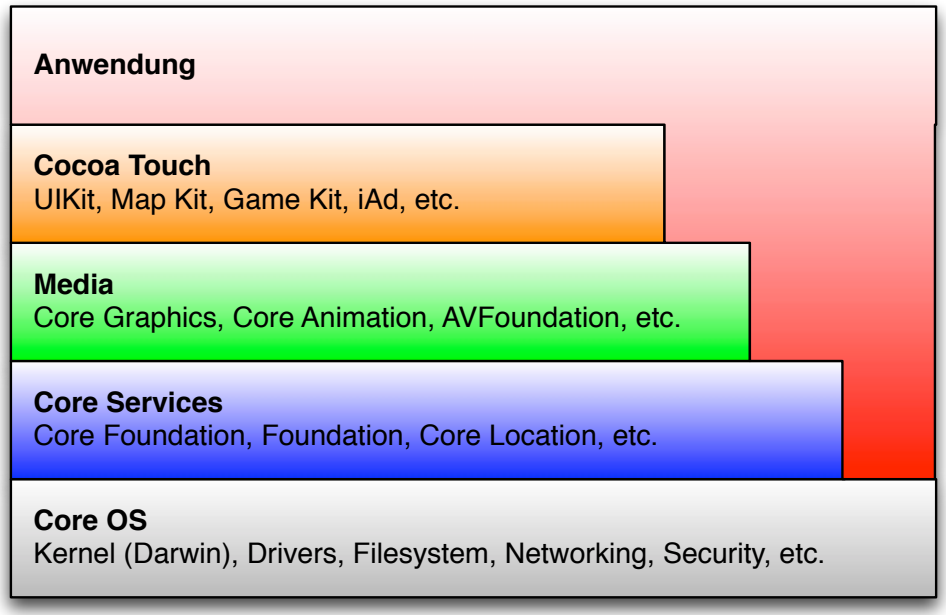
POSIX/BSD UNIX

Abstammung von NeXTStep/OpenStep

iOS Systemarchitektur



Core OS



Darwin Kernel

Treiber

Dateisystem

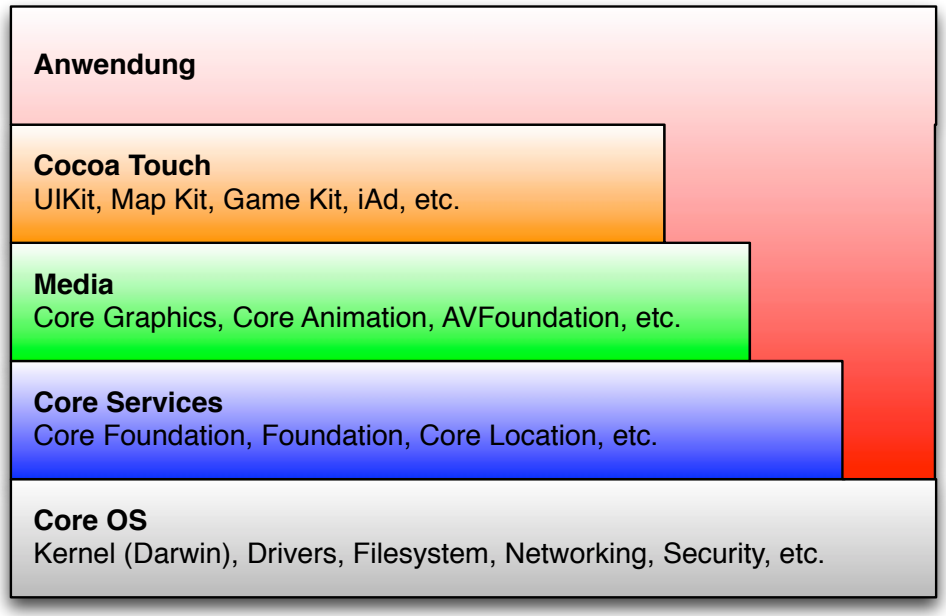
Netzwerk

Sicherheit

Stromsparfunktionen

System-Level Services
(POSIX/BSD 4.4/C99)

Core Services



Core Foundation

Foundation

Adressbuch

Einstellungen

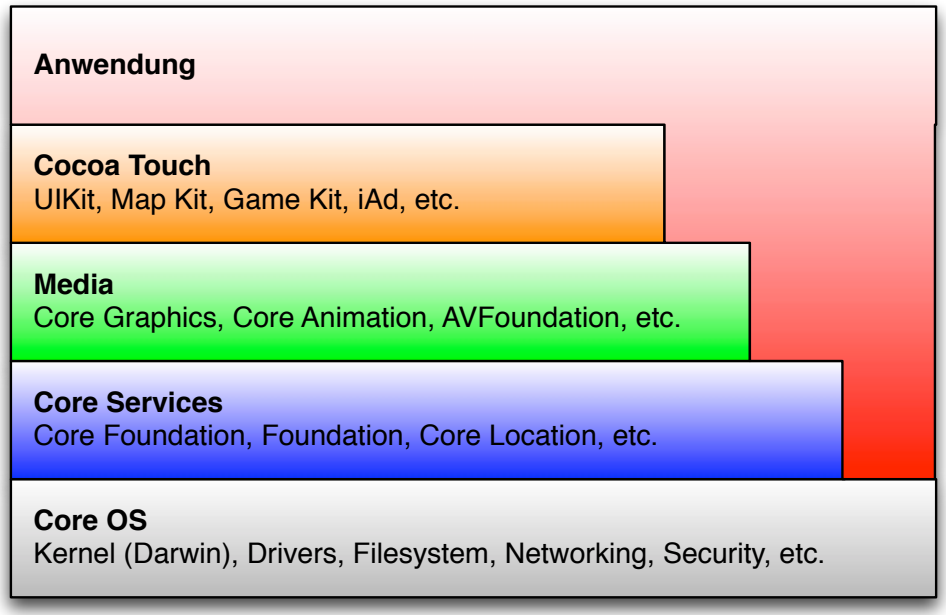
Core Location

Core Motion

Core Data

System Configuration

Media



Core Graphics

Core Animation

AVFoundation

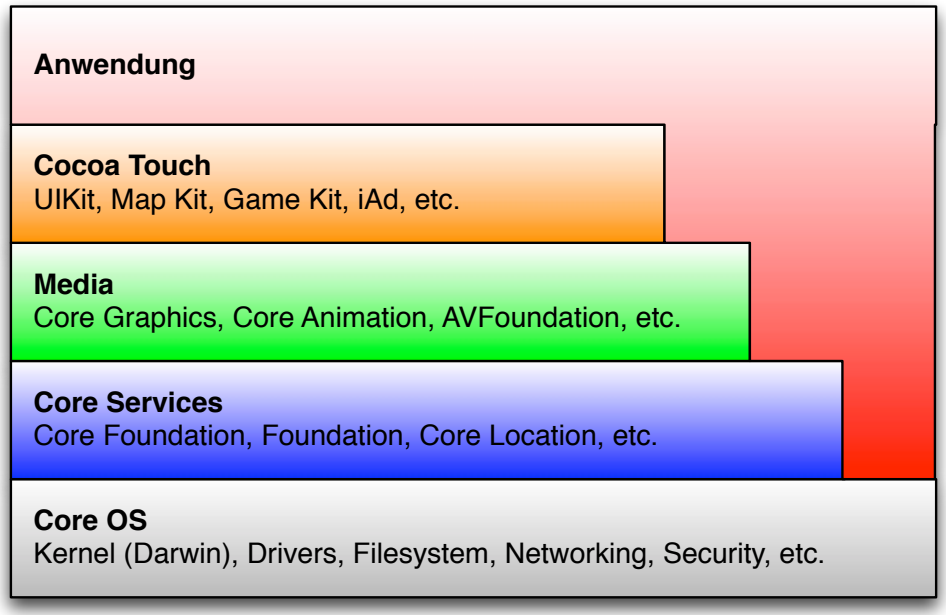
Video-Wiedergabe

Open GL ES

Core Audio

Core Text

Cocoa Touch



UIKit

Map Kit

Game Kit

iAd

Besonders wichtige APIs

Core Foundation (CF)
C-API, implementiert viele Basis-Funktionen wie String-Manipulation, Container, OS-Services, etc.

Foundation (NS)
Objective-C-API, im Wesentlichen die objektorientierte Version von Core Foundation

UIKit (UI)
Definiert die Schnittstelle zur Anwendung und das User Interface

Anwendung

Application-Bundle (.app)
GeZIPtes Verzeichnis mit Binary, Info.plist und weiteren Ressourcen

Mach-O Binary

Mehrere Binaries in einem Container (Fat Binary) möglich. -> Unterstützung für verschiedene Architekturen

Anwendung für iPhone oder iPad, oder "Universal" mit Unterstützung für beide

Erzeugung via Xcode -> Archive -> Export oder mit xcodebuild-Kommandozeilentool

Anwendung

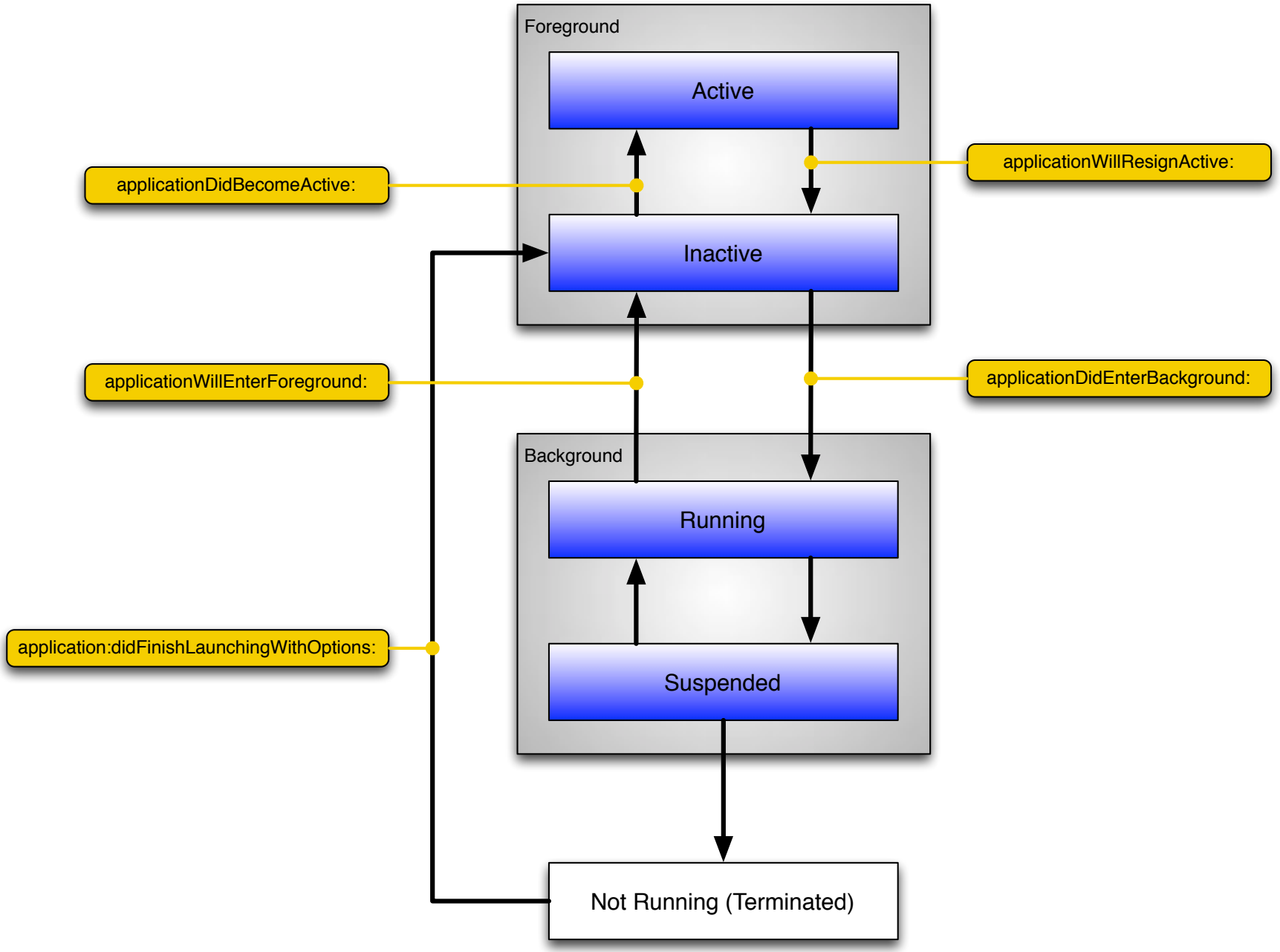
Event-Getrieben

User-Eingaben und andere Ereignisse erzeugen Events

Events werden über die RunLoop auf dem Main-Thread verarbeitet

Ankerpunkt ist eine Instanz der Klasse UIApplication, welche u.a. die Callbacks für die Änderungen des Lebenszyklus erhält.

Lebenszyklus der Anw.



Zustände

Not Running
Die Anwendung wurde noch nicht gestartet oder beendet.

Inactive
Die Anwendung läuft im Vordergrund, empfängt derzeit aber keine Events.

Active
Die Anwendung ist im Vordergrund und empfängt Events.

Running
Die Anwendung ist im Hintergrund, also nicht sichtbar, empfängt keine Events, wird aber ausgeführt.

Suspended
Die Anwendung ist im Hintergrund und wird nicht ausgeführt.
Anwendungen wechseln automatisch von Running hier hin und zurück.

UIApplicationDelegate Callbacks

application:willFinishLaunchingWithOptions:
Die erste Gelegenheit Code auszuführen

application:didFinishLaunchingWithOptions:
Wird ausgeführt bevor die Anwendung sichtbar gemacht wird.

applicationDidBecomeActive:
Die Anwendung ist jetzt aktiv.

applicationWillResignActive:
Die Anwendung wird gleich inaktiv.

applicationDidEnterBackground:
Die Anwendung wird jetzt im Hintergrund ausgeführt.

applicationWillEnterForeground:
Die Anwendung wird gleich in den Vordergrund versetzt werden.

applicationWillTerminate: (Nur in Sonderfällen!)
Die Anwendung wird gleich beendet.
Achtung: Wird nicht aufgerufen wenn der Zustand Suspended ist.

Einstiegspunkt der Anwendung

Die Ausführung einer Anwendung startet, wie bei in C geschriebenen Anwendungen üblich, mit dem Aufruf der Funktion main()

UIApplicationMain() führt RunLoop aus

Die Anwendung endet wenn UIApplicationMain() endet. Ausnahme: Es laufen weitere Threads

Achtung: Das wird NIEMALS durch die Anwendung selber ausgelöst! -> Siehe Lebenszyklus

```
int main(int argc, char *argv[])
{
  @autoreleasepool {
      return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));
  }
}
```

Info.plist

Konfigurations-Datei der Anwendung

Definiert Konstanten: Name, Versionsnummer, Build-Nummer, Name des Application Bundles, Angezeigter Name, Bundle Identifier, Minumum OS-Version, Entwicklungssprache, etc.

Unterstützte Geräte-Typen (iPhone/iPad/iPod touch)

Benötigte Fähigkeiten des Geräts

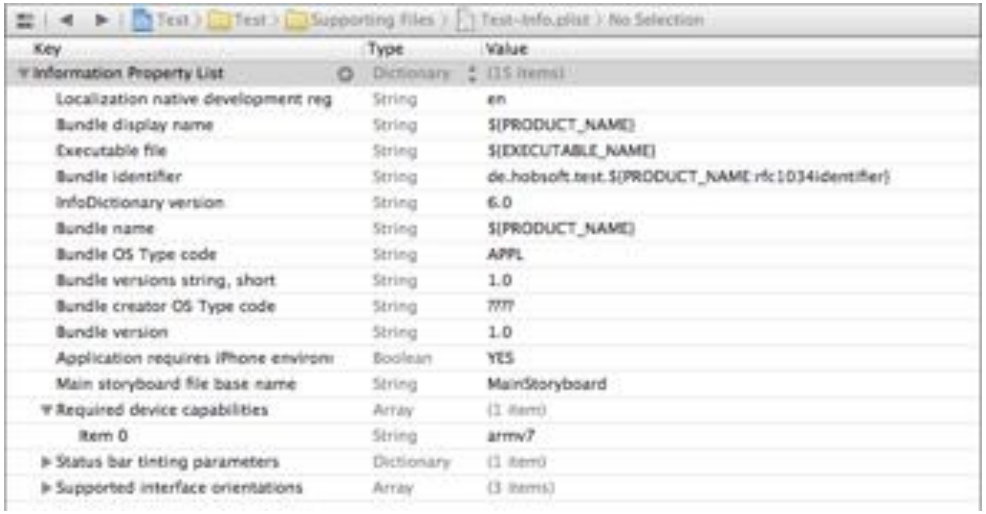
Unterstützte Geräte-Orientierungen

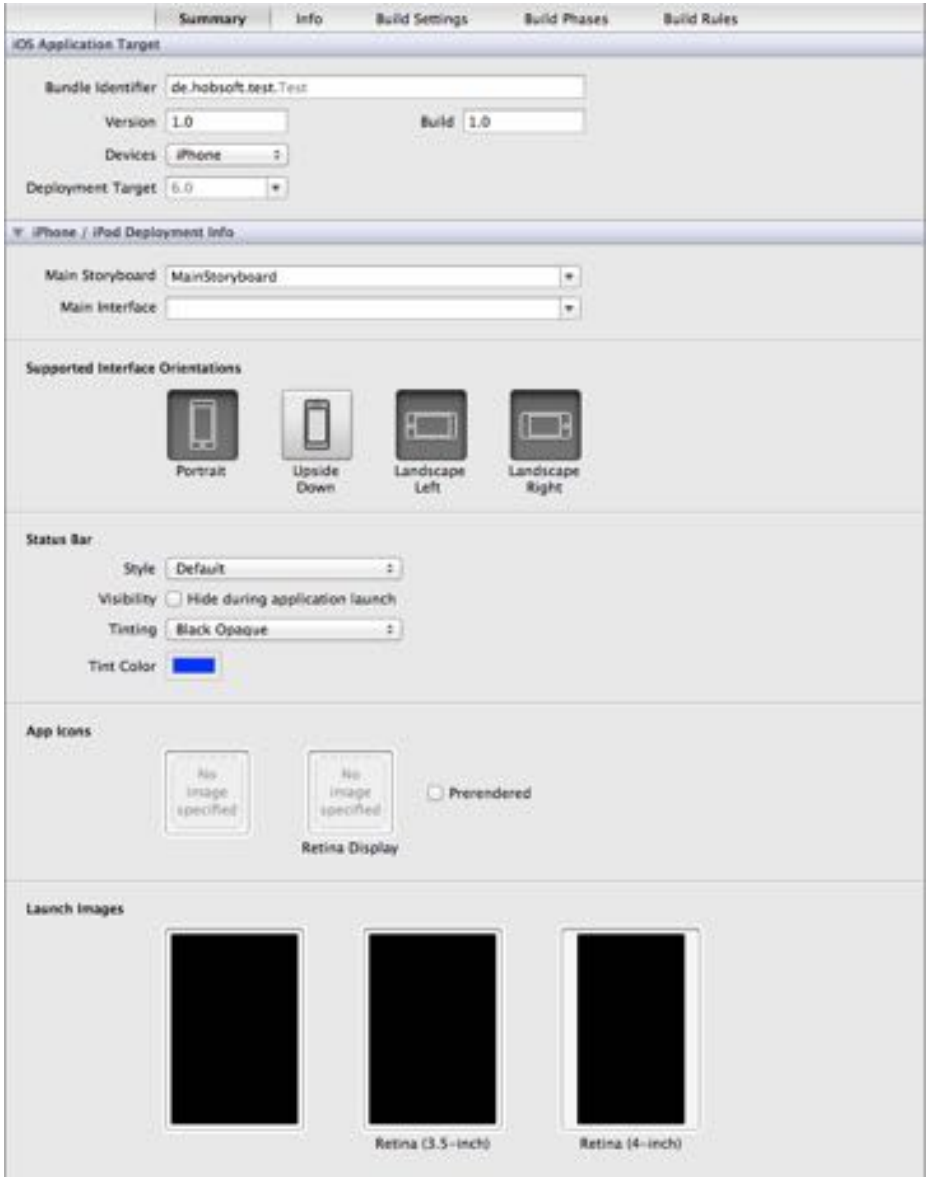
Icon

Start-Bild

Und einiges mehr...

Info.plist


Test › Test › Supporting Files › Test-Info.plist › No Selection
Key
Type
Value
Information Property List
Dictionary
(15 items)
Localization native development reg
String
en
Bundle display name
String
${PRODUCT_NAME}
Executable file
String
${EXECUTABLE_NAME}
Bundle identifier
String
de.hobsoft.test.${PRODUCT_NAME:rfc1034identifier}
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
1.0
Application requires iPhone environ
Boolean
YES
Main storyboard file base name
String
MainStoryboard
Required device capabilities
Array
(1 item)
Item 0
String
armv7
Status bar tinting parameters
Dictionary
(1 item)
Supported interface orientations
Array
(3 items)


Summary
Info
Build Settings
Build Phases
Build Rules
iOS Application Target
Bundle Identifier
de.hobsoft.test.Test
Version
1.0
Build
1.0
Devices
iPhone
Deployment Target
6.0
iPhone / iPod Deployment Info
Main Storyboard
MainStoryboard
Main Interface
Supported Interface Orientations
Portrait
Upside Down
Landscape Left
Landscape Right
Status Bar
Style
Default
Visibility
Hide during application launch
Tinting
Black Opaque
Tint Color
App Icons
No image specified
No image specified
Prerendered
Retina Display
Launch Images
Retina (3.5-inch)
Retina (4-inch)

Bibliotheken

Static Library (.a)
Werden in das Anwendungs-Binary integriert

Dynamic Libray (.dylib)
Separates Binary, wird zur Laufzeit geladen
(Ab iOS 8 auch für Entwickler)

Framework (.framework)
Eigenständiges Bibliotheks-Paket mit Binaries, Ressourcen und Headern

Weak-Linking möglich -> Rückwärtskompatibilität

iOS Developer Program

Varianten

Free
Zugang zu Dokumentation und Anwendungen, Xcode, Test auf eigenen Geräten

Standard (€ 100,- im Jahr)
Zugang zu Beta-Releases, Zertifikate, Test auf registrierten Testgeräten, Release im App Store

Enterprise (€ 299,- im Jahr)
In-House-Distribution, kein App Store

University
Zertifikate und Test auf Geräten für Studenten

iOS-Versionen?

Apple empfiehlt 2 Versionen (Major Releases) zu unterstützen

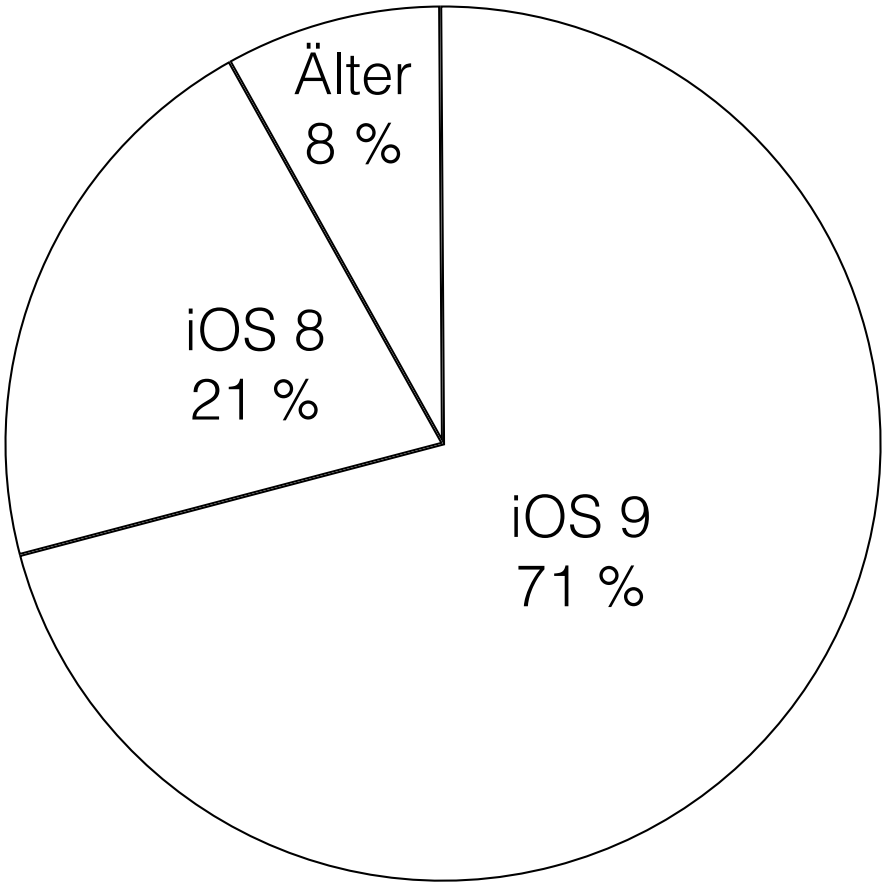
Neuentwicklungen evtl. nur noch für die aktuelle Version (kommt aber auf die Zielgruppe an)

Update-Rate durch On-Device Updates seit iOS 5 sehr hoch

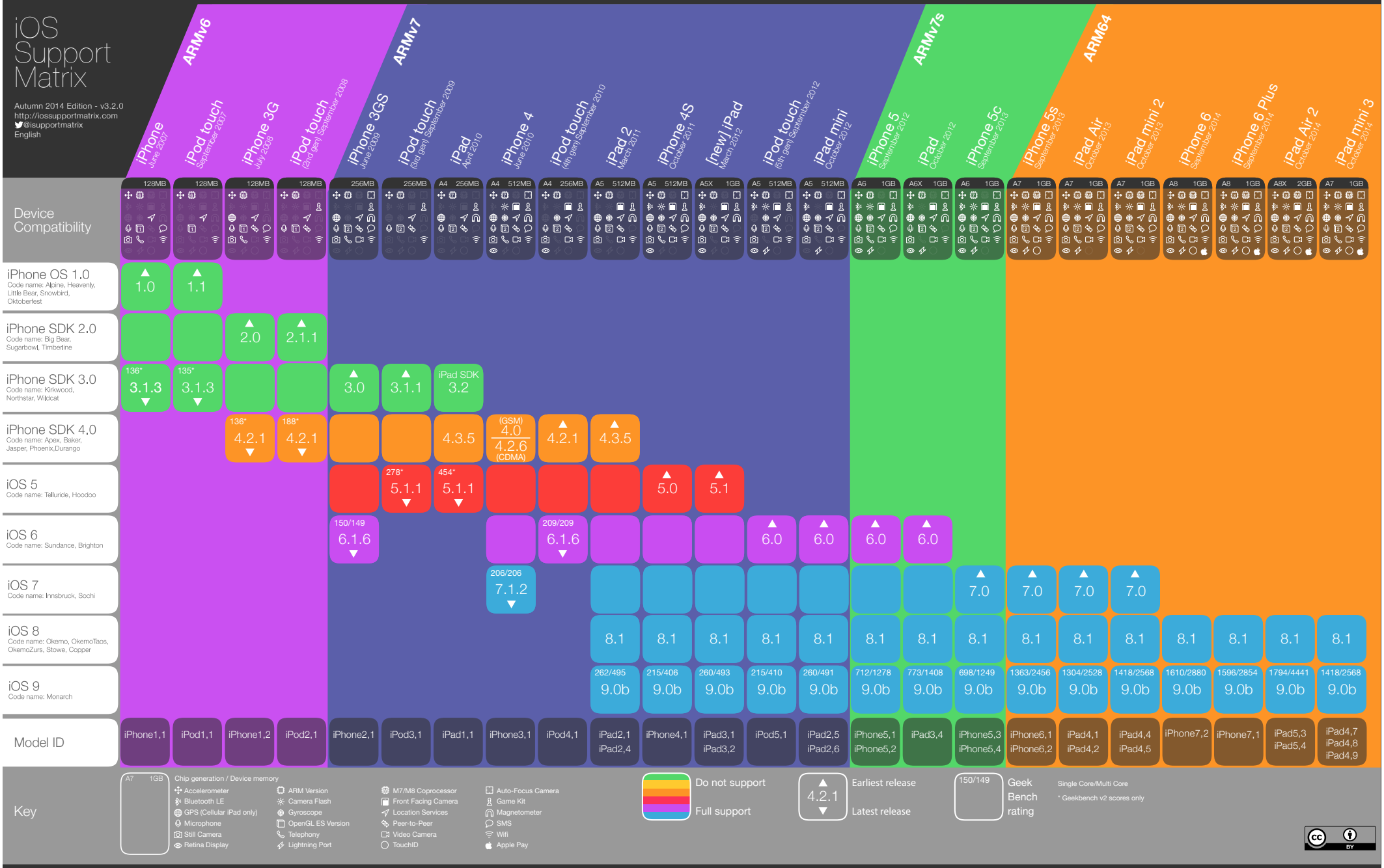
iOS 7-Phänomen

iOS-Versionen

Verteilung der Nutzer des App Stores und damit die relevante Zielgruppe



Quelle: Apple (https://developer.apple.com/support/appstore), Stand 3. Januar 2016


iOS Support Matrix
Autumn 2014 Edition - v3.2.0
http://iossupportmatrix.com
@isupportmatrix
English
ARMv6
ARMv7
ARMv7s
ARM64
iPhone June 2007
iPod touch September 2007
iPhone 3G July 2008
iPod touch (2nd gen) September 2008
iPhone 3GS June 2009
iPod touch (3rd gen) September 2009
iPad April 2010
iPhone 4 June 2010
iPod touch (4th gen) September 2010
iPad 2 March 2011
iPhone 4S October 2011
[new] iPad March 2012
iPod touch (5th gen) September 2012
iPad mini October 2012
iPhone 5 September 2012
iPad October 2012
iPhone 5c September 2013
iPhone 5s September 2013
iPad Air October 2013
iPad mini 2 October 2013
iPhone 6 September 2014
iPhone 6 Plus September 2014
iPad Air 2 October 2014
iPad mini 3 October 2014
Device Compatibility
iPhone OS 1.0
Code name: Alpine, Heavenly, Little Bear, Snowbird, Oktoberfest
iPhone SDK 2.0
Code name: Big Bear, Sugarbowl, Timberline
iPhone SDK 3.0
Code name: Kirkwood, Northstar, Wildcat
iPhone SDK 4.0
Code name: Apex, Baker, Jasper, Phoenix,Durango
iOS 5
Code name: Telluride, Hoodoo
iOS 6
Code name: Sundance, Brighton
iOS 7
Code name: Innsbruck, Sochi
iOS 8
Code name: Okemo, OkemoTaos, OkemoZurs, Stowe, Copper
iOS 9
Code name: Monarch
Model ID
iPhone1,1
iPod1,1
iPhone1,2
iPod2,1
iPhone2,1
iPod3,1
iPad1,1
iPhone3,1
iPod4,1
iPad2,1 iPad2,4
iPhone4,1
iPad3,1 iPad3,2
iPod5,1
iPad2,5 iPad2,6
iPhone5,1 iPhone5,2
iPad3,4
iPhone5,3 iPhone5,4
iPhone6,1 iPhone6,2
iPad4,1 iPad4,2
iPad4,4 iPad4,5
iPhone7,2
iPhone7,1
iPad5,3 iPad5,4
iPad4,7 iPad4,8 iPad4,9
Key
Chip generation / Device memory
Accelerometer
Bluetooth LE
GPS (Cellular iPad only)
Microphone
Still Camera
Retina Display
ARM Version
Camera Flash
Gyroscope
OpenGL ES Version
Telephony
Lightning Port
M7/M8 Coprocessor
Front Facing Camera
Location Services
Peer-to-Peer
Video Camera
TouchID
Auto-Focus Camera
Game Kit
Magnetometer
SMS
Wifi
Apple Pay
Do not support
Full support
Earliest release
Latest release
Geek Bench rating
Single Core/Multi Core
* Geekbench v2 scores only

Tools

LLVM-Toolchain

Seit einigen Jahren neue, auf LLVM basierende Toolchain

LLVM/Clang: C, Objective-C, C++ Compiler

Static Analyzer

LLDB: Debugger

Xcode

Xcode 6: Moderne SDI Entwicklungsumgebung

Übliche Features: Syntax-Highlighting, Fehler/ Warnungen, Codevervollständigung, Dokumentation, Refactoring, etc.

Basierend auf LLVM-Toolchain

Static Analyzer

Interface Builder

LLDB-Integration

Emulator(en)

Emulatoren vs. Simulatoren

Im Sprachgebrauch werden alle oft „Emulatoren“ genannt

Niemals auf Emulatoren verlassen! Immer auf echten Geräten testen!

Der iPhone/iPad Simulator

Simulator

Macht dem Namen volle Ehre: Nicht 100%ig kompatibel

Unterstützt verschiedene Features nicht, schlecht oder anders (Ortsbestimmung, Netzwerk, Bluetooth, Sensoren, etc.)

Laufzeitverhalten hängt vom Host-System ab. In der Regel -viel- schneller!

Instruments

Umfangreiches Werkzeug für Analyse und Performance-Tuning

Das beste Tool am Markt!

Instrumente u.a.: Allocations, Leaks, Time Profiler, Automation, Network, Core Animation, Open GL ES

Praxis

Setup, Xcode-Einführung & Hallo Welt

Apples Programmiersprachen

Objective-C

Apples „Haus- und Hof-Sprache“

> 30 Jahre alt

Stark renoviert aber in die Jahre gekommen

Dynamisch Typisiert

C ergänzender Syntax ist an Smalltalk angelehnt

Swift

Neu!

Neuentwicklung, 2014 vorgestellt

Multiparadigmen-Sprache

Einsetzbar, aber mit Vorsicht

Statisch Typisiert

Syntax grob an C++ angelehnt

Objective-C


I PROGRAMMED IN OBJECTIVE-C
ONCE
IT WAS HORRIBLE

Ursprung

Designer: Brad Cox und Tom Love

Erstes Release: 1983

Basiert auf C mit objektorientierten Erweiterungen die an Smalltalk angelehnt sind

NeXT lizensierte Objective-C, erweiterte GCC, schrieb neue Klassenbibliotheken und Tools (AppKit, Foundation Kit, Interface Builder) und implementierte NeXTStep/OpenStep damit.

Später kaufte NeXT die Rechte an Objective-C

Stark im Wandel

2006 stellte Apple Objective-C 2.0 vor

Seit der Umstellung auf LLVM als Toolchain in 2008 wird nun aktiv an der Sprache gearbeitet

- Reduktion von Boilerplate
- Einführung neuer Features

"Modern Objective-C" kam mit iOS 6

Wir besprechen den aktuellen Stand

Aber Achtung: Rückwärtskompatibilität!
Features sind oft abhängig von der LLVM/Clang-Version (Xcode) und der Runtime (iOS-Version)

Basiert auf ANSI C99

Objective-C ist ein striktes Superset von C (heute ANSI C99), erweitert die Sprache aber im OO-Bereich im Stil und nach Terminologie von Smalltalk

Soll heißen: Alles wie gehabt, außer wenn man konkret Objekte behandelt.

Kompatibles Typsystem: Objekte werden per Zeiger referenziert: `NSObject *obj;`

Trennung in Header (.h) und Implementierung (.c, .m, .cpp, .mm)

Neue Datentypen

`nil` ist das `NULL` der ObjC-Welt

`nil` und `NULL` sind derzeit als 0 definiert, sollten aber in ihren entsprechenden Bereichen benutzt werden.

`BOOL` ist der Boole'sche Datentyp von ObjC.
`YES = 1, NO = 0`

`id` ist der dynamische Datentyp. Er steht für ein beliebiges ObjC-Objekt und wird zur Compile-Zeit nicht weiter geprüft.

`id<Human>` steht für ein ObjC-Objekt welches das Protokoll `Human` implementiert

Prefixes

Keine Namespaces! Als Alternative werden Prefixes verwendet. Zum Beispiel: NSObject, UIApplication

Das Prefix beschreibt das Framework, zu dem Klasse oder Protokoll gehört

Apple reserviert für sich Prefixes mit zwei Buchstaben

Entwickler sollen Prefixes mit drei Buchstaben verwenden

Die Klassen der eigentlichen Anwendung erhalten in der Regel keinen Prefix

Imports

#import ersetzt #include

Funktion wie gehabt, jedoch vermeidet #import Mehrfach-Inklusion, was bei #include die #define/ #ifdef-Orgie notwendig gemacht hat

Neu: Modul-Import mit @import bindet automatisch das entsprechende Framework ein. Aber: Nur bei System-Frameworks


ONE DOES NOT SIMPLY CALL A METHOD
IN OBJECTIVE C

Mitteilung & Selector

Message Passing: Man sendet einem Objekt, dem Empfänger, eine Mitteilung. Die aufzurufende Methode wird erst zur Laufzeit festgelegt.

Ein Selector beschreibt eine Methode symbolisch und wird als Teil der Mitteilung übergeben.

Zur Laufzeit wird der Selector zu einem C Funktions-Zeiger aufgelöst und diese Funktion aufgerufen.

Mitteilungen

```
[obj doSomethingWith:argument];
```

Entspricht dem Java Methodenaufruf:
`obj.doSomething(argument);`

`obj` ist der Empfänger der Mitteilung

`doSomethingWith:` ist der Selector

`argument` ein Parameter

`doSomethingWith:argument` ist die Mitteilung (Selector plus Parameter)

Mitteilungen

```
[obj doSomethingWith:arg1 and:arg2];
```

Entspricht dem Java Methodenaufruf:
`obj.doSomething(arg1, arg2);`

`obj` ist der Empfänger der Mitteilung

`doSomethingWith:and:` ist der Selector

`arg1` und `arg2` sind Parameter

`doSomethingWith:arg1 and:arg2` ist die Mitteilung (Selector und Parameter)

Schreibweise von Mitteilungen

```
[obj doSomethingWith:arg1 and:arg2];
```

Mitteilungen werden mit der Infix-Notation beschrieben. Der entsprechende Selector wird daher von den Parametern unterbrochen.

Nach einem Doppelpunkt folgt jeweils ein Parameter

Schreiben Sie Mitteilungen die wie Sätze klingen!
"Erzeuge das mit diesem und jenem"

Selektoren und Parameter schreibt man in CamelCaseSchreibweise

Selektoren und Parameter am Anfang klein schreiben.
(Nur Klassen groß.)

Indirekte Mitteilungen

```
  SEL sel= @selector(doSomethingWith:);
[obj performSelector:sel withObject:arg];
```

Die Direktive @selector(...) wird vom Compiler in einen "Compiled Selector" überführt. Diese haben den Datentyp "SEL". Mit ihm können Selektoren als Werte übergeben werden.

```
[obj performSelector:@selector(foo)];
```
entspricht
```
[obj foo];
```

Mitteilungen schachteln

```
[[obj foo] bar];
```

Wie von Funktions- oder Methodenaufrufen in C oder Java gewohnt, können auch in ObjC Mitteilungen geschachtelt werden.

Man schickt eine Mitteilung mit dem Selector `foo` an `obj`. Der Rückgabewert ist ein Objekt, dem nun eine Mitteilung mit dem Selector `bar` geschickt wird.

Punkt-Notation

Eingeführt mit Objective-C 2.0

Ermöglicht:

```
int v= obj.value;
obj.value= v;
```

Der Compiler ruft an dieser Stelle einfach die entsprechende Zugriffsmethode (accessor method, get/set) auf.
Also `[obj value]`
oder `[obj setValue:v]`

Methoden

Unterscheidung zwischen Klassenmethoden und Instanzmethoden

Da Klassen in ObjC auch Objekte sind, können diesen ebenfalls Mitteilungen geschickt werden:

```
[NSArray array];
```

Keine Sichtbarkeitsregeln für Methoden!
Der Header dokumentiert die API, der Rest ist mit Hilfe der Runtime-Informationen aufrufbar, wird aber als "Private API" bezeichnet und soll nicht verwendet werden.

Methoden

Deklaration von Klassenmethoden:

```
+ (NSArray*)array;
+ (SpaceShip*)sharedSpaceShip;
```

Deklaration von Instanzmethoden:

```
- (int)length;
- (void)execute;
```

Vor der Implementierung einer Methode wird sie (wie schon in C) wiederholt:

```
- (void)execute {
    // Execute stuff here
}
```

Demo & Praxis

Klassen

```
@interface Person : NSObject
- (void)doSomething;
@end
```

Single Inheritance

Deklaration und Implementierung einer Klasse werden C-typisch getrennt

Der öffentlich sichtbare Teil beschreibt das Interface der Klasse. Es wird von den Compiler-Direktiven @interface und @end eingeschlossen.

Hier wird die Klasse "Person" deklariert, welche von "NSObject" abgeleitet wird.

Klassen

```
@implementation Person
- (void)doSomething { /*...*/ }
@end
```

Die Implementierung einer Klasse befindet sich in einer .m-Datei

Verwendet die Direktiven @implementation und @end

Protokolle

Protokolle in Objective-C sind das, was man in Java als Interface bezeichnet. -> Richtlinien für die Schnittstelle eines Objekts.

Klassen können mehrere Protokolle implementieren

`@optional` erlaubt Teile eines Protokolls als Optional zu deklarieren. Optionale Methoden müssen nicht implementiert werden. Die anderen schon.

Protokolle können von von einem oder mehreren Protokollen abgeleitet werden

Protokolle

```
@protocol Human <Mammal, Speaker>
- (void)think;

@optional
- (void)sing;

@end

@interface Person : NSObject <Human>
- (void)doSomething;
@end
```

Das Protokoll `Human` leitet von `Mammal` und `Speaker` ab und deklariert die Methoden `-think` und `-sing`, wobei `-sing` Optional ist

Die Klasse Person implementiert das Protokoll `Human`

Objekte erzeugen

Objekte werden in zwei Phasen erzeugt:

- Alloc: Reservieren des Speicherbereiches für das Objekt. (Uninitialisiert, noch nicht verwendbar!)
- Init: Initialisierung des Objekts

Beispiel: `[[NSObject alloc] init];`

Alternative: `[NSObject new];`
Ruft intern nach obigem Beispiel `alloc` und `init` auf.

Objekt-Initialisierung

Verschiedene Varianten von init-Methoden werden verwendet um Objekte mit und ohne Parametern zu erzeugen

Designated Initializer: Alle Varianten der init-Methode einer Klasse müssen einen designated initializer der Klasse aufrufen

Ein designated initializer der Klasse muss jeweils einen designated initializer der Oberklasse aufrufen

Der Rückgabewert der designated initializers (evtl. der Oberklasse) muss der Variable `self` zugewiesen werden

Ein Initializer kann `nil` zurückgeben

Objekt-Initialisierung

```
// Designated Initializer
- (instancetype)initWithValue:(int)someValue {
    self = [super init];

    if( self ) {
        value= someValue;
        // Initialize self...
    }

    return self;
}

// Default Initializer
- (id)init {
    return [self initWithValue:0];
}
```

Speicherverwaltung

Instanzen von ObjC-Objekten residieren immer auf dem Heap

ObjC-Objekte nutzen Retain-Count Memory-Management. Hierbei wird pro Objekt ein Zähler (retainCount) verwaltet, der die Lebensdauer des Objekts bestimmt. Erreicht er 0, wird das Objekt sofort gelöscht. (dealloc)

Achtung vor Zyklen!

Manual Reference Counting (MRR)
Manuelles `retain` & `release`

Automatic Reference Counting (ARC)
Manuelles `retain` & `release` verboten, der Compiler kümmert sich darum.

ARC

Seit iOS 5 Standard

Kann pro Compilation-Unit ein- oder ausgeschaltet werden. (-fno-objc-arc)

Alle Objekt-Zeiger (iVars, lokale Variablen) sind standardmäßig "Strong" (__strong) deklariert. Zugewiesene Objekte werden automatisch `retained` bzw. `released`.

Ein Objekt-Zeiger der "Weak" (__weak) deklariert ist, zeigt nur so lange auf ein Objekt, wie es Strong deklarierte Zeiger auf das Objekt gibt. Hat ein Objekt keine Strong-Zeiger mehr, wird es gelöscht und alle Weak-Zeiger die auf das Objekt zeigen auf nil gesetzt. (Nullifying Weak Reference)

In iOS 4 gibt es zwar ARC-Unterstützung, aber noch keinen Weak-Support. Alternative: __unsafe_unretained

Instanzvariablen (iVars)

```
@implementation Person {
@private
  int age;
  Person *spouse;
}
@end
```

Optional können für eine Klasse Instanzvariablen deklariert werden

Sichtbarkeit (Scope)

- @private: Nur in dieser Klasse sichtbar (default)
- @protected: Auch in Ableitungen sichtbar
- @public: Überall sichtbar
- @package: Wie @public innerhalb des Ausführungseinheit (executable image) in dem die Klasse implementiert ist. Außerhalb wie @private.

Properties

Convenience-Notation für die Deklaration und Implementierung von Zugriffsmethoden.

```
@property int value;
@property (copy) NSString *name;
@property (readonly) UIView *view;
```

```
@synthesize value; // Deklariert iVar value
@synthesize name = _surname; // Nutzt iVar surname
```

Default: @synthesize weg lassen erzeugt implizit die iVar _value

@dynamic: Alternative zu @synthesize. Stellt Implementierung zur Laufzeit zur Verfügung.

Properties

Mögliche Optionen

- Schreibrechte: readwrite, readonly
- Referenzierung: strong, weak, copy, assign
- Nebenläufigkeitsschutz: nonatomic, (implizit: atomic)

Default: strong/assign, readwrite, (atomic)

getter/setter kann nach Bedarf ohne Änderungen manuell implementiert werden

Unterschiedliche Option im public/private Interface möglich.

Properties

```
@interface Foo : NSObject

@property int value;
@property (copy) NSString *name;
@property (readonly) UIView *view;

@end
```

```
// Unnamed category of Foo
@interface Foo ()

// Redeclaration of view using different access rights
@property (readwrite) UIView *view;

@end

@implementation Foo {
   NSString *_surname;
}

// value and view iVars are implicitly synthesized
@synthesize name = _surname;

- (instancetype)init {
   self= [super init];

   if( self ) {
      self.value = 0;
      _value = 0; // Alternative

      self.name = @"Miller";
      _surname = @"Miller"; // Does not copy!

      UIView *v1 = self.view; // Calls -view
      UIView *v2 = _view; // Does not call -view!
   }

  return self;
}

// getter-Method for view-property
- (UIView*)view {
   if( !_view )
      _view = [UIView new];

   return _view;
}

@end
```

Demo & Praxis

Blocks

Closures für C

Quasi ein referenzierbarer Code-Block (anonyme Funktion) in dem die Variablen des umliegenden Sichtbarkeitsbereichs (outer scope) sichtbar bleiben

Variablen des umliegenden Sichtbarkeitsbereich stehen standardmäßig nur lesend zur Verfügung

Attribut __block deklariert eine umliegende Variable als Beschreibbar

Readonly Variablen spiegeln den Zustand zum Moment des Capturings dar. (^)

Blocks

Beispiel 1: Block ohne Parameter

Deklaration:

```
void (^block)()=
        ^{ NSLog(@"Hello World!"); };
```

Aufruf:

```
block();
```

Blocks

Beispiel 2: Block mit Parameter

Deklaration:

```
NSString* (^block)(NSString *name)=
    ^{ NSLog(@"Hello %@!", name); };
```

Aufruf:

```
block(@"Daniel");
```

Hypothetischer Anwendungsfall:

```
forEach(arrayOfStrings, block);
```

Alternative: Inline-Deklaration

```
forEach(arrayOfStrings, ^(NSString *name) {
    NSLog(@"Hello %@!", name);
});
```

Blocks

Blocks sind normale ObjC-Objekte, können also Mitteilungen empfangen und haben den üblichen Lebenszyklus

Beispiel asynchrones Ausführen von Code:

```
NSOperationQueue *queue= [NSOperationQueue new];
[queue addOperationWithBlock:^{
   NSLog(@"Look, ma, I'm doing something!");
}];
```

Namensgebung

Dokument: "Coding Guidelines for Cocoa"
Sehr Lesenswert!

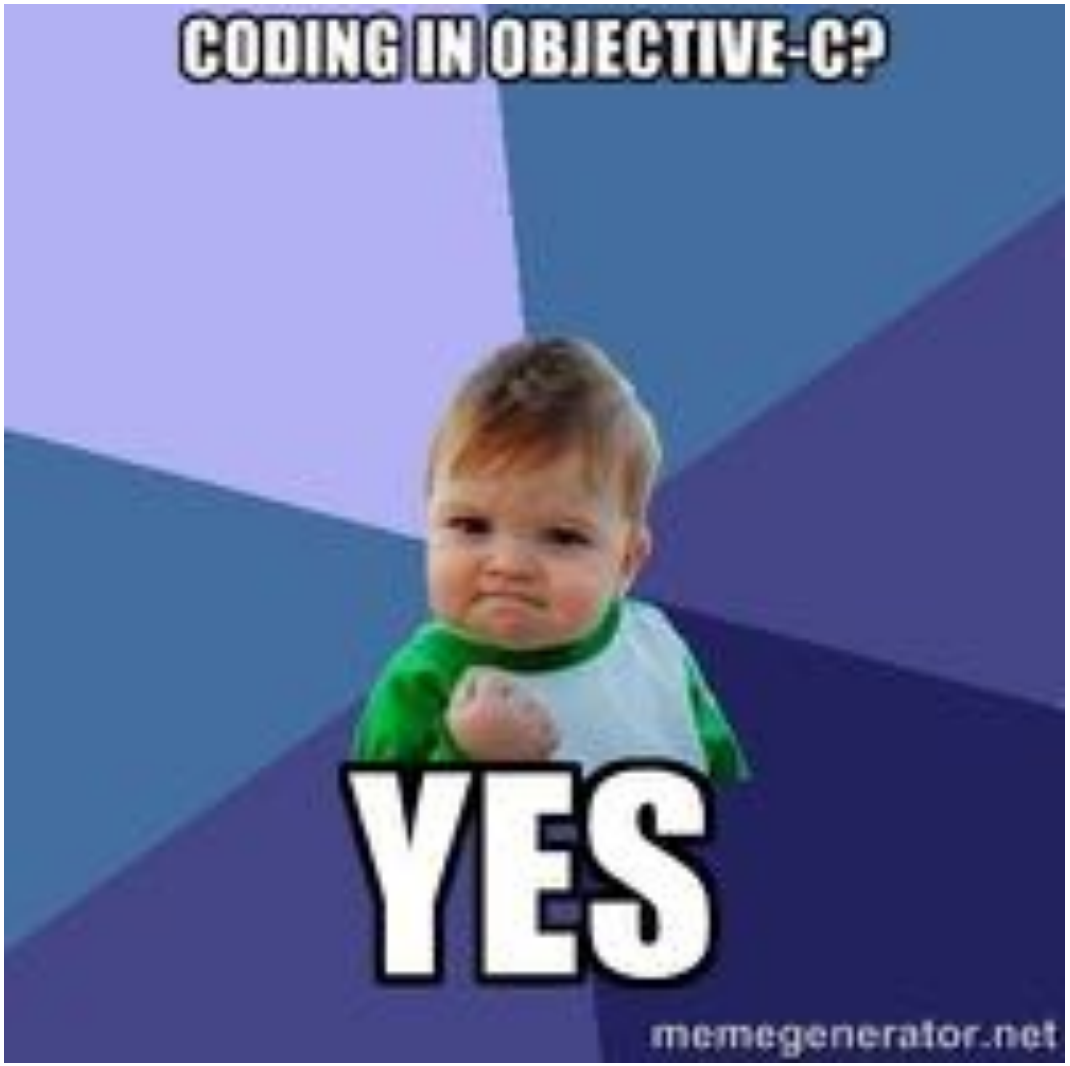
alloc*, init*, new* am Anfang von Methoden haben eine Bedeutung!

Get/Set-Paare: Kein get* benutzen!

Get: - (int)value;
Set: - (void)setValue:(int)value;

Vor allem: Immer eindeutige Bezeichner benutzen!
80/20-Regel: 80% der Zeit wird Code gelesen!

Keine Scheu vor langen Bezeichnern! Die Autovervollständigung ist dein Freund.


CODING IN OBJECTIVE-C?
YES
memegenerator.net

Foundation
Wichtige Basisklassen

Mutable vs. Immutable

Container-Klassen gibt es im Foundation-Framework (fast) immer in zwei Varianten, einer veränderlichen und einer unveränderlichen.

Die veränderliche (mutable) leitet von der unveränderlichen (immutable) Variante ab. Eine veränderliche kann so als eine unveränderliche zurückgegeben werden.

Vom Downcast einer unveränderlichen zu einer veränderlichen Version wird -ausdrücklich- abgeraten!

NSString

NSString und NSMutableString

Constante NSStrings definieren:

```
@"Das ist ein konstanter NSString"
"Das ist ein klassischer nullterminierter C-String"
```

Zum Vergleich -isEqualToString benutzen

Nicht auf die Gleichheit von Objektzeigern verlassen

`@"" == @""` -> SCHLECHT/FALSCH!

`[@"" isEqualToString:@""];` -> OK

Formatierte Strings erzeigen:

```
[NSString stringWithFormat:@"%d, %f, %@", someInt, someFloat, someObject];
```

NSNumber

Wrapper-Klasse für numerische C-Typen

Nur als Immutable-Version!

Beispiele:

```
[NSNumber numberWithInt:0];
[NSNumber numberWithDouble:0.0];
[number boolValue];
[number intValue];
```

Kurzform: `@1, @2.2, @(3+4+value), @YES`

NSArray

NSArray und NSMutableArray

Nur ObjC-Objekte erlaubt, kein nil

Class-Cluster: Unterschiedliche, für unterschiedliche Anwendungsfälle optimierte Implementierungen

Garantierte Laufzeit von O(log n) für
-objectAtIndex:

Konstante NSArrays definieren:
Leeres NSArray: `@[]`
Mit zwei Werten: `@[@1, @2]`

Subscript-Operatoren:
Lesen: `NSNumber *number= @[@1, @2][0];`
Schreiben: `someMutableArray[3]= @1;`

Konstante Arrays und Subscript-Operatoren erst ab Apple LLVM 4.0!

NSDictionary

NSDictionary und NSMutableDictionary

Key/Value Store (aka Hashtabelle)

Nur ObjC-Objekte erlaubt, kein nil

Class-Cluster: Unterschiedliche, für unterschiedliche Anwendungsfälle optimierte Implementierungen

Konstante NSDictionary definieren:
Leeres NSDictionary: `@{}`
Mit zwei Werten:

```
@{@"key1":@"value1", @"key2":@"value2"}
```

Subscript-Operatoren:
Lesen: `NSNumber *number= dict[@"key"];`
Schreiben: `mutableDict[@"key"]= @"value";`

Konstante Dictionaries und Subscript-Operatoren erst ab Apple LLVM 4.0!

NSSet

NSSet und NSMutableSet

Unsortierte Sammlung von eindeutigen Objekten (aka Bag)

Nur ObjC-Objekte erlaubt, kein nil

Class-Cluster: Unterschiedliche, für unterschiedliche Anwendungsfälle optimierte Implementierungen

Keine Kurzform für NSSet... :-(

Fast Enumeration

Alternative zu den klassischen Enumeratoren

Beispiel:

```
NSArray *data= @[@1, @2, @3];

for( NSNumber *value in data )
   NSLog(@"The value: %@", value);
```

Bitte nutzt die Dokumentation!

Demo & Praxis

Das MVC-Pattern

Model: Daten & Businesslogik

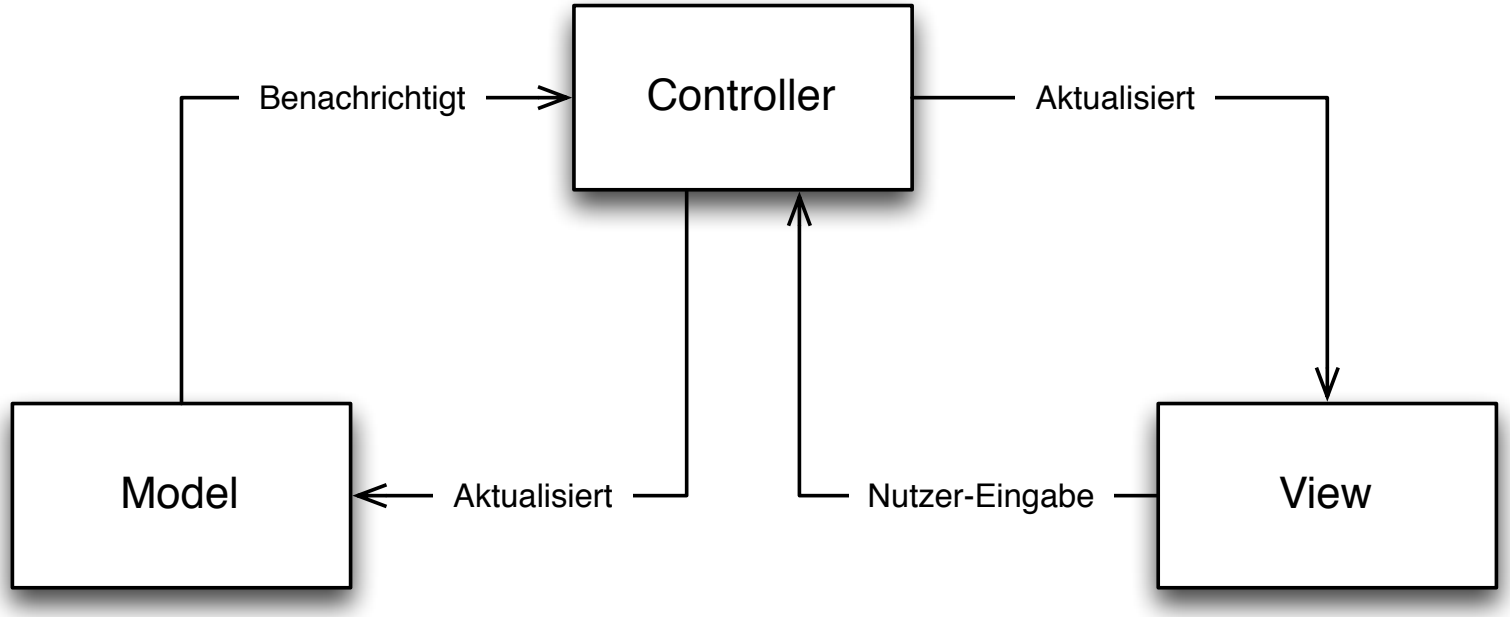
View: Elemente die Dargestellt werden, möglichst passiv

Controller: Hält die Fäden in der Hand, Kontrolliert das Geschehen, Synchronisiert Model und View. (aka "Glue Logic", ViewController)

Kontakt-Punkt von Außen ist i.d.R. nur der Controller

Apple nennt seine Controller ViewController um Verwechselungen mit Kontrollern außerhalb des User Interfaces zu vermeiden

Das MVC-Pattern



User Interfaces

Punkte vs. Pixel

Das von UIKit verwendete Koordinatensystem wird nicht in Pixeln, sondern in Punkten gerechnet

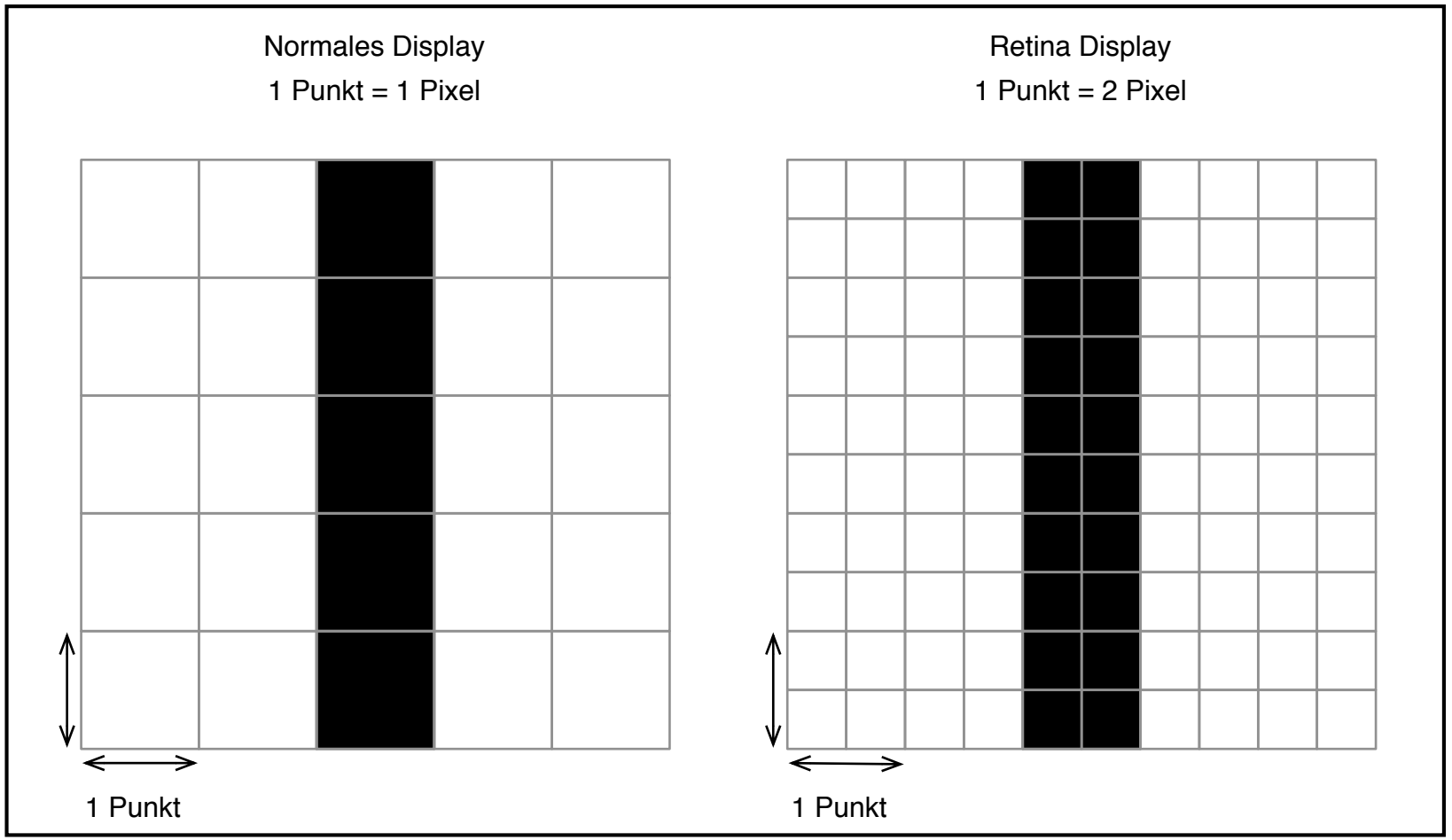
Punkte beschreiben einen Pixel auf einem imaginären Bildschirm mit 160 dpi

160 dpi entspricht der Punktdichte des ursprünglichen iPhone-Displays. Das Verhältnis Punkte zu Pixel war dort also 1:1.

Moderne Retina-iPhone-Displays haben 320 dpi, also ein Verhältnis von 1:2. Ein Punkt wird also von 4 Pixeln dargestellt.

Mit dieser Lösung kann man bei verschiedenen Punktdichten das selbe Layout verwenden, wobei dieses mit den besseren Displays lediglich detaillierter dargestellt wird

Punkte vs. Pixel



Grundidee Mobile UI

Ein Bildschirm, ein Set an Informationen

Ein "Fenster"

Inhalte möglichst ohne notwendiges Scrollen darstellen

"Embrace your framework"

Standard-UI = Bekannte UI

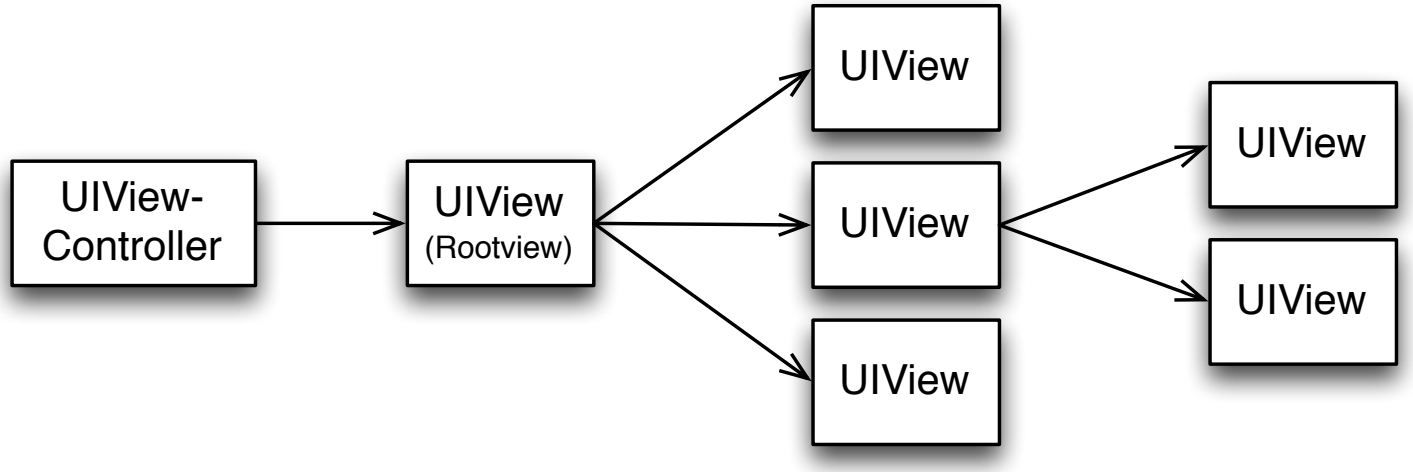
Aufbau eines User Interfaces

Ein View Controller (UIViewController) betreut eine Hierarchie von Views (UIView)

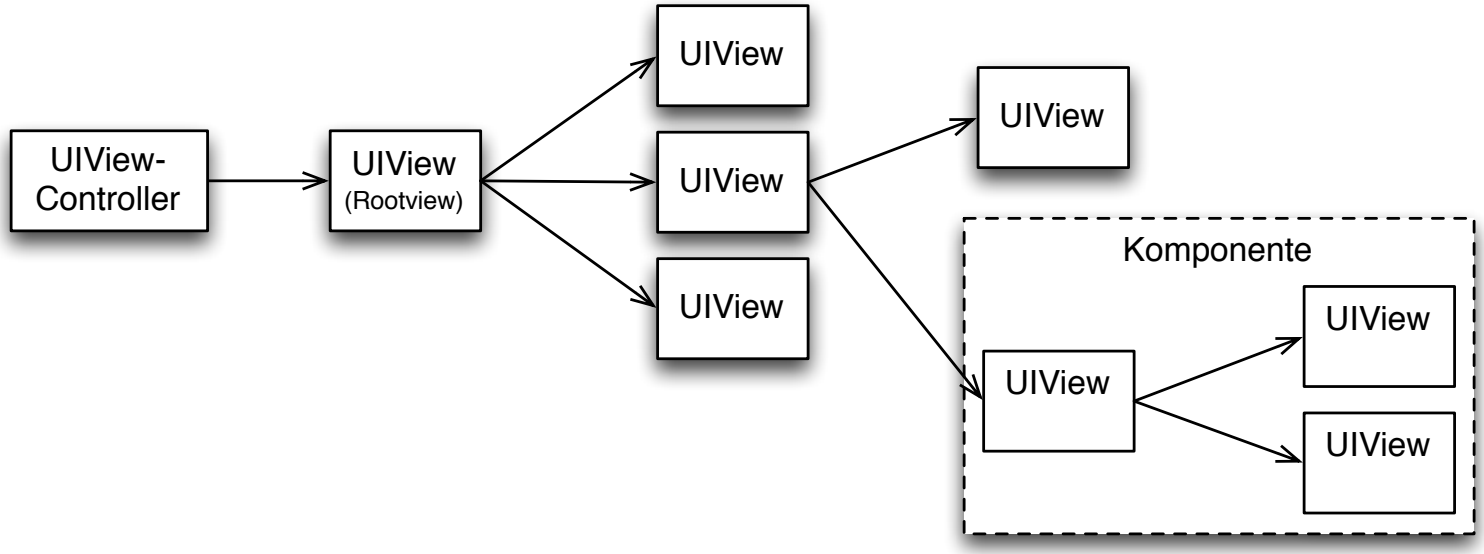
Jede View kann beliebig viele Subviews haben

View-Hierarchie

View-Hierarchie



View-Hierarchie



UIViewController

Steht für einen "Bildschirm"

Schaltstelle für einen "Bildschirm"

Kontaktpunkt nach Außen

Wird i.d.R. abgeleitet

Apple stellt viele View-Controller für die verschiedensten Funktionalitäten zur Verfügung

Unterscheidung zwischen Content-View-Controllern und Container-View-Controllern

Kümmert sich um das Laden der View-Komponenten, sofern diese aus einem NIB kommen

Kann wiederum andere UIViewController anzeigen

UIViewController Relevante Callbacks

-loadView
Kann genutzt werden um Views manuell zu erzeugen

-viewDidLoad
Wird aufgerufen wenn das Laden der Views abgeschlossen ist. Hier können sie noch einmal manipuliert werden bevor sie sichtbar werden

Ab iOS 6: Kein entladen von Views mehr (macht das Handling einfacher)

UIViewController Relevante Callbacks

-viewWillAppear:
Die Views werden gleich sichtbar

-viewDidAppear:
Die Views sind jetzt sichtbar

-viewWillDisappear:
Die Views werden gleich unsichtbar

-viewDidDisappear:
Die Views sind jetzt unsichtbar

UIView

Definiert Ort, Größe und Inhalt eines rechteckigen Bereichs auf dem Bildschirm

Verwaltet eine Liste von Subviews

Kümmert sich um das Layout der Subviews

Empfängt UIResponder-Events (Touches, Motions)

Gesture Recognizer

Basiert auf einem CALayer

UIView
Relevante Properties

backgroundColor (UIColor)
Hintergrundfarbe der View

hidden (BOOL)
View wird nicht angezeigt, wenn YES

alpha (CGFloat)
Transparenz-Wert der gesamten View

bounds (CGFrame)
Ort und Größe der UIView im eigenen Koordinatensystem,
Ort ist i.d.R. 0/0, Größe ist synchron mit der Größe von frame

frame (CGFrame)
Ort und Größe der UIView im Koordinatensystem der Superview,
Größe ist synchron mit bounds, synchron mit center

center (CGPoint)
Das Zentrum der UIView im Koordinatensystem der Superview,
synchron mit frame

Demo & Praxis

UIViewController/UIView, Bindings
UICatalog

Demo & Praxis

Hallo <Name>

Gesten

Gesten

Gesture Recognizer erleichtern das erkennen von Gesten signifikant im Vergleich zu vorher

Von UIGestureRecognizer abgeleitete Klassen implementieren unterschiedliche Gesten: Tap, Pinch, Rotate, Swipe, Pan, LongPress

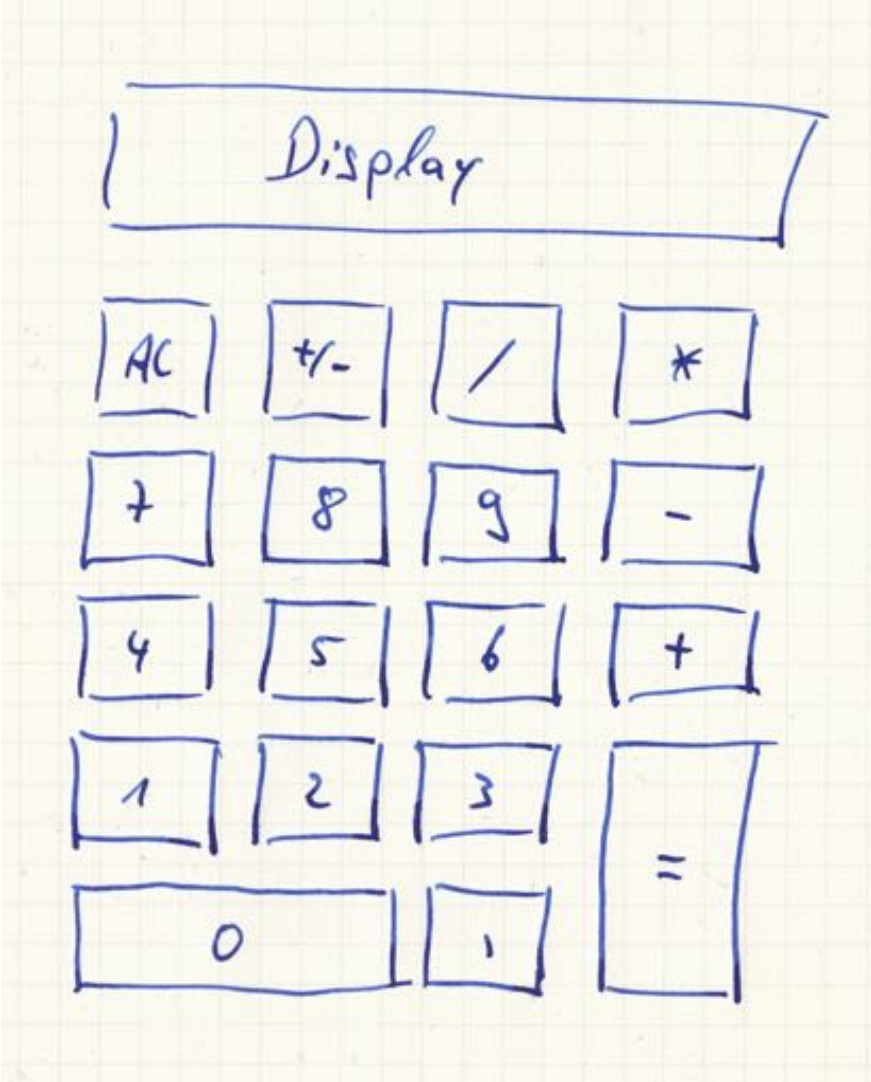
Einfaches Zuweisen zu einer beliebigen UIView per Drag & Drop im IB oder via -addGestureRecognizer:

Target/Action definieren, fertig

Demo & Praxis

Gesten

Praxis: Taschenrechner



Layouts

Traditionell werden Cocoa UIs (im Storyboard oder in Code) pixelpräzise erstellt

Vorteil: SEHR leicht für typische/einfache UIs

Nachteil: Dynamisches Verhalten, zum Beispiel bei wechselnden Auflösungen und Displays, ist aufwändig (deshalb vermeidet Apple das)

Autoresize Masks (Alt!) bringen etwas Flexibilität

Übriges, dynamisches Verhalten musste manuell implementiert werden: -layoutSubviews

Auto-Layout

Neu ab iOS 6, das Constraint-Basierte Auto-Layout

Man definiert Einschränkungen/Anforderungen, die das System versucht einzuhalten

Sehr vielfältige Möglichkeiten des dynamischen Verhaltens

Leider aber ein komplexes Thema. Dauert etwas, bis man die Denkweise verinnerlicht hat.

Demo & Praxis

Layouts

Popups

Popups ergänzen eine Anwendung an den Stellen, wo zusätzliche Navigation zu komplex oder verwirrend wäre

Praktisch für Fehlermeldungen, etc, aber bitte überlegt einsetzen! Übermäßiger Gebrauch kann als sehr störend empfunden werden.

Es gibt zwei Varianten von Popups: UIAlertView und UIActionSheet

UIAlertView

Einfache Möglichkeit zur Darstellung von kurzen Informationen

Kann definiert werden: Titel, Text, Button-Texte

Besondere Rolle: Cancel-Button

Einfache Nutzung:

```
UIAlertView *alert= [[UIAlertView alloc]
initWithTitle:@"Woah!" message:@"Hello
World!" delegate:nil cancelButtonTitle:nil
otherButtonTitles:@"OK", nil];
[alert show];
```

Wird asynchron angezeigt!

UIAlertViewDelegate implementieren: - alertView:clickedButtonAtIndex:

UIActionSheet

Ähnlich wie UIAlertView, hebt jedoch den Prozess der Auswahl besser hervor, integriert sich besser in den Fluss der Anwendung

Beispiel

```
UIActionSheet *actionSheet= [[UIActionSheet
alloc] initWithTitle:@"What is it?" delegate:nil
cancelButtonTitle:nil
destructiveButtonTitle:@"Uncool."
otherButtonTitles:@"Cool!", nil];
[actionSheet showInView:self.view];
```

Asynchron!

UIActionSheetDelegate implementieren:
-actionSheet:clickedButtonAtIndex:

Demo & Praxis

UIAlertView, UIActionSheet
UICatalog

Anwendungs-Navigation

Bei den meisten Anwendungen ist es notwendig, dass zwischen verschiedenen Ansichten hin und her gewechselt wird

Dafür gibt es drei Grundlegende Navigations-Lösungen

- Tab-Bar
- Drilldown-Navigation
- Modale Dialoge

Tab Bar

UITabBarController

UIViewController Container-Klasse

Tableiste zeigt Tabs

Hinter jedem Tab liegt ein UIViewController

Der UIViewController des ausgewählten Tabs wird angezeigt

Drilldown-Navigation

UINavigationController

UIViewController Container-Klasse

Man "bohrt" sich in die Tiefe der Informationen

Logische Hierarchie

"Zurück"-Pfeil erlaubt das zurückkehren zu vorigen Stufe/Ansicht

Rechtspfeil ">" deutet Drilldown-Möglichkeit an

Meistens in Tabellen zu finden

Drilldown-Navigation

UINavigationController

UINavigationController verwaltet einen Navigations-Stack

Der erste UIViewController heißt Root View Controller und ist das unterste Element des Stacks

Der oberste View Controller (Top View Controller) ist gerade sichtbar

Push- und Pop-Operationen verändern den Stack

Modale Dialoge

Jeder UIViewController kann jeweils einen anderen UIViewController modal darstellen

Stacking möglich!

Der Stack existiert hierbei allerdings nur indirekt in der UIViewController-Hierarchie (presentingViewController/ presentedViewController)

Der modal dargestellte UIViewController überdeckt den darstellenden UIViewController vollständig

Der dargestellte UIViewController muss sich darum kümmern, dass er wieder geschlossen werden kann

Eher vermeiden wenn es eine Alternative gibt...

Anwendungs-Navigation

Traditionell wurde Anwendungs-Navigation fest in die View Controller programmiert -> Unflexibel!

Mit iOS 5 hat Apple Storyboards eingeführt. Sie verschieben die hartkodierte Navigations-Logik in ein oder mehrere Storyboards, die außerdem auch noch die View Controller aufnehmen

View Controller werden so flexibler einsetzbar, können aber weiterhin auf besondere Gegebenheiten reagieren

Storyboards

Definiert UIViewController, die UIViews der View Controller und die Übergänge zwischen den View Controllern (Segues) an einem Ort

View Controller und Segues erhalten Identifier, mit denen sie referenziert werden können

Initialisierung der View Controller/Views und die Übergänge sind vollständig automatisiert

Beim bisherigen View Controller (Source VC) wird – `prepareForSegue:sender:` aufgerufen. Das gibt dem VC die Gelegenheit dem neuen VC (Destination VC) Daten mitzugeben

Demo & Praxis

Anwendungs-Navigation

UITableView

Ein Großteil vieler iOS-UIs ist in Form von Tabellen aufgebaut

Einzelne Zellen (UITableViewCell) werden als Komponenten gebaut und mehrfach verwendet

Zeigt einspaltige Tabellen ohne Zellenbegrenzung an

Varianten: Plain & Grouped

Zusätzlich: Header, Footer, Section-Header, Section-Footer

UITableViewDelegate & UITableViewDataSource

UITableViewController (optional aber nützlich) implementiert Convenience-Funktionen

UITableViewCell

Definiert eine einzelne Tabellenzelle

Abgeleitet von UIView

Vordefinierte Cell Stile: Default, Value1, Value2, Subtitle

Custom Cells: Von UITableViewCell abgeleitete Zellen können im IB entworfen werden

Demo & Praxis

UITableView
Adressbuch

Persistenz

User Defaults

NSUserDefaults bietet einen automatisch persistierten Key/Value-Storage (ala NSDictionary)

Wird verwendet um wenige/kurze Werte über die Laufzeitgrenzen der Anwendung hinweg zu speichern

Möglich: C-Datentypen, aber auch NSString, NSArray, NSDictionary, NSURL, NSData

Auch wenn NSUserDefaults Objekte serialisieren und speichern kann, so ist das nur eingeschränkt empfohlen

Wird als P-Liste (XML) im privaten Anwendungsverzeichnis gespeichert

User Defaults

Nutzung

Schreiben

```
[[NSUserDefaults standardUserDefaults]
setInteger:42
forKey:@"That is the Answer"];
```

Lesen

```
[[NSUserDefaults standardUserDefaults]
stringForKey:@"Username"];
```

Achtung: Daten liegen im Klartext auf dem Gerät!
Nicht Sicher! (Alternative für Passwörter: Keychain)

Dateien

P-Listen direkt Lesen und Schreiben mit NSArray, NSDictionary, NSData und NSString

Siehe Beispiel

Demo & Praxis

Persistenz

Ressourcen

Ressourcen Laden

Alle Ressourcen, die ins Projekt eingebunden sind, werden auch in die Anwendung kopiert (Application Bundle)

Pfad für Bundle-Ressource abfragen:

```
[[NSBundle mainBundle]
pathForResource:@"text"ofType:@"txt"];
```

Dann Ressource laden, z.Bsp. mit NSData:

```
[NSData
dataWithContentsOfFile:fileName];
```

UIImage kann direkt aus dem Main-Bundle laden:

```
[UIImage imageNamed:@"img"];
```

Buchempfehlungen


Learn Objective-C on the Mac
iPhone App Development
Cocoa and Objective-C
Test-Driven iOS Development
iPhone Apps Entwicklung für DUMMIES
Objective-C und Cocoa
iOS 5
iPhone 3D
Cocoa Design Patterns
The iPhone Developer's Cookbook
Programming in Objective-C 2.0

Buchempfehlungen

Bücher die ausschließlich Modern Objective-C und Storyboards behandeln (so wie wir es getan haben) sind bisher nicht existent.

Viele alte Bücher sind OK, behandeln aber größtenteils alte Technologien und (heute) unnötig komplizierte Lösungen. Trotzdem können sie natürlich hilfreich sein.

Bitte immer schauen ob eine neue Auflage existiert oder in Kürze kommt!

Buchempfehlungen

Immer darauf achten ob ein Buch Mac (Cocoa) oder iOS (Cocoa Touch) behandelt

iOS/Mac-Hybriden sind oft unnötig kompliziert oder vernachlässigen eine Plattform stark (meist iOS da es i.d.R. alte Mac-Bücher sind, die umgebranded wurden)

Reine iOS-Bücher sind zu bevorzugen

Reine Mac-Bücher sind für unseren Anwendungsfall nicht geeignet!

Buchempfehlungen

Deutsche Bücher

Deutsche Bücher gibt es derzeit keine, die ich für besonders erwähnenswert halte. Viele sind allerdings „OK“.

Viele davon sind auch in der Bibliothek in GM verfügbar.

Buchempfehlungen

Objective-C

Kochan, Steven G.: Programming in Objective-C 2.0
SEHR UMFANGREICH

Buchempfehlungen

Empfehlenswerte A-Press Serie

- Dalrymple; Knaster: Learn Objective-C on the Mac
  Gute Einführung in das alte Objective-C. Die Plattform spielt quasi keine Rolle!
- LaMarche; Mark: Beginning iPhone 3 Development
- LaMarche; Mark: More iPhone 3 Development

Buchempfehlungen

Neuburg, Matt: Programming iOS 5
Gute Referenz

Stanford-Kurs

iTunes U-Kurs von Paul Hegarty, Stanford University

Sehr empfehlenswert und außerdem aktuell (behandelt ebenfalls ausschließlich Modern Objective-C und Storyboards)

Titel: „Coding Together: Developing Apps for iPhone and iPad (Winter 2013)“

Direktlink: https://itunes.apple.com/de/course/coding-together-developing/id593208016

Ergänzend: Tolle iPad-App „iTunes U“ von Apple

Absprache Projekte, Projektpräsentationen

Danke für‘s (zu)hören!