



# iOS Application Development

## Lecture 6: Save Data and System View Controller

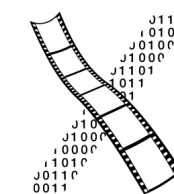
Simon Völker & Philipp Wacker  
Media Computing Group  
RWTH Aachen University

[hci.rwth-aachen.de/ios](https://hci.rwth-aachen.de/ios)



**RWTHAACHEN**  
UNIVERSITY

# Saving Data



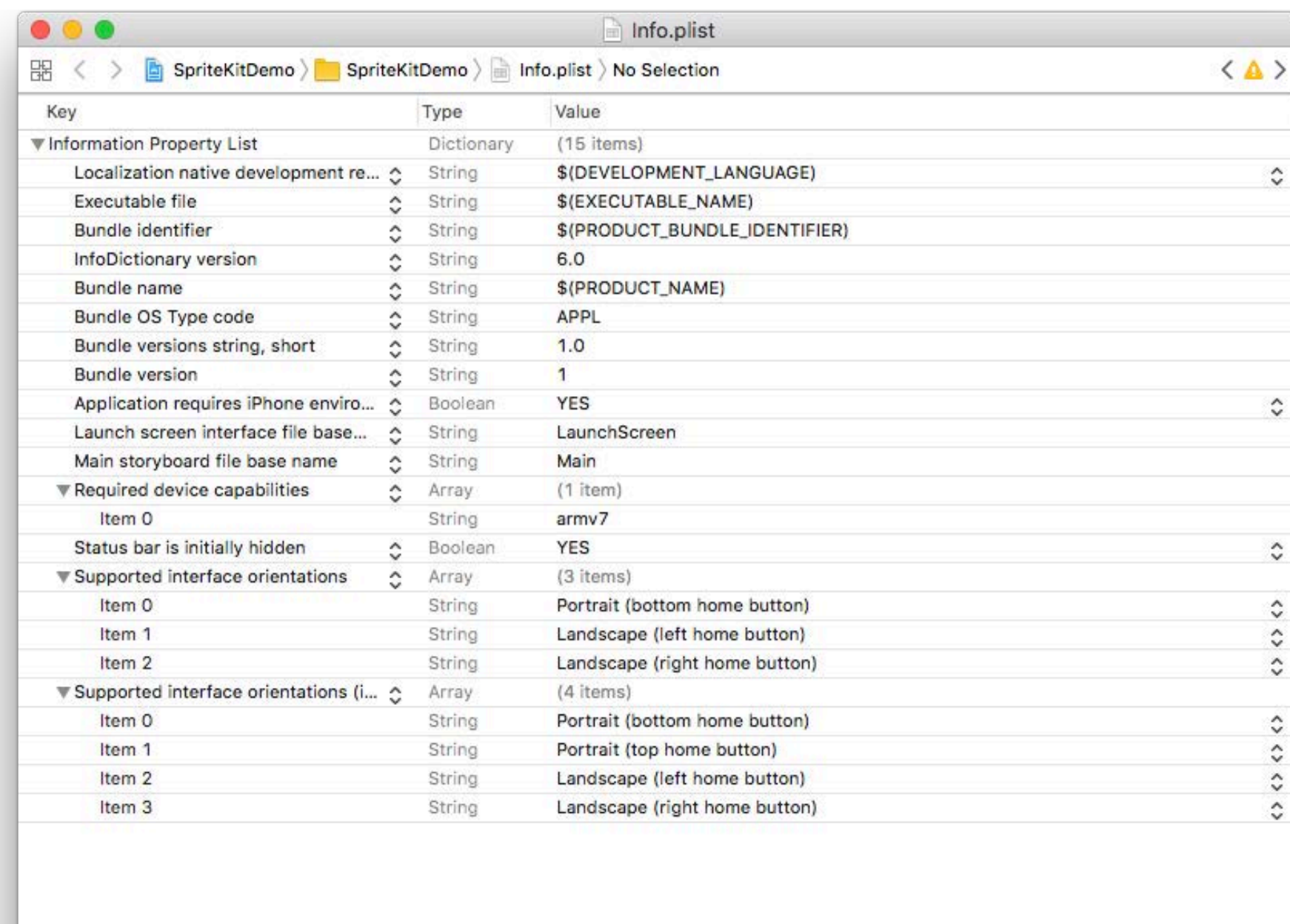
# Saving Data

- Save into a file using encoder (E.g. JSON, or XML pLists)
- Save data using CoreData
- Save data into a Database

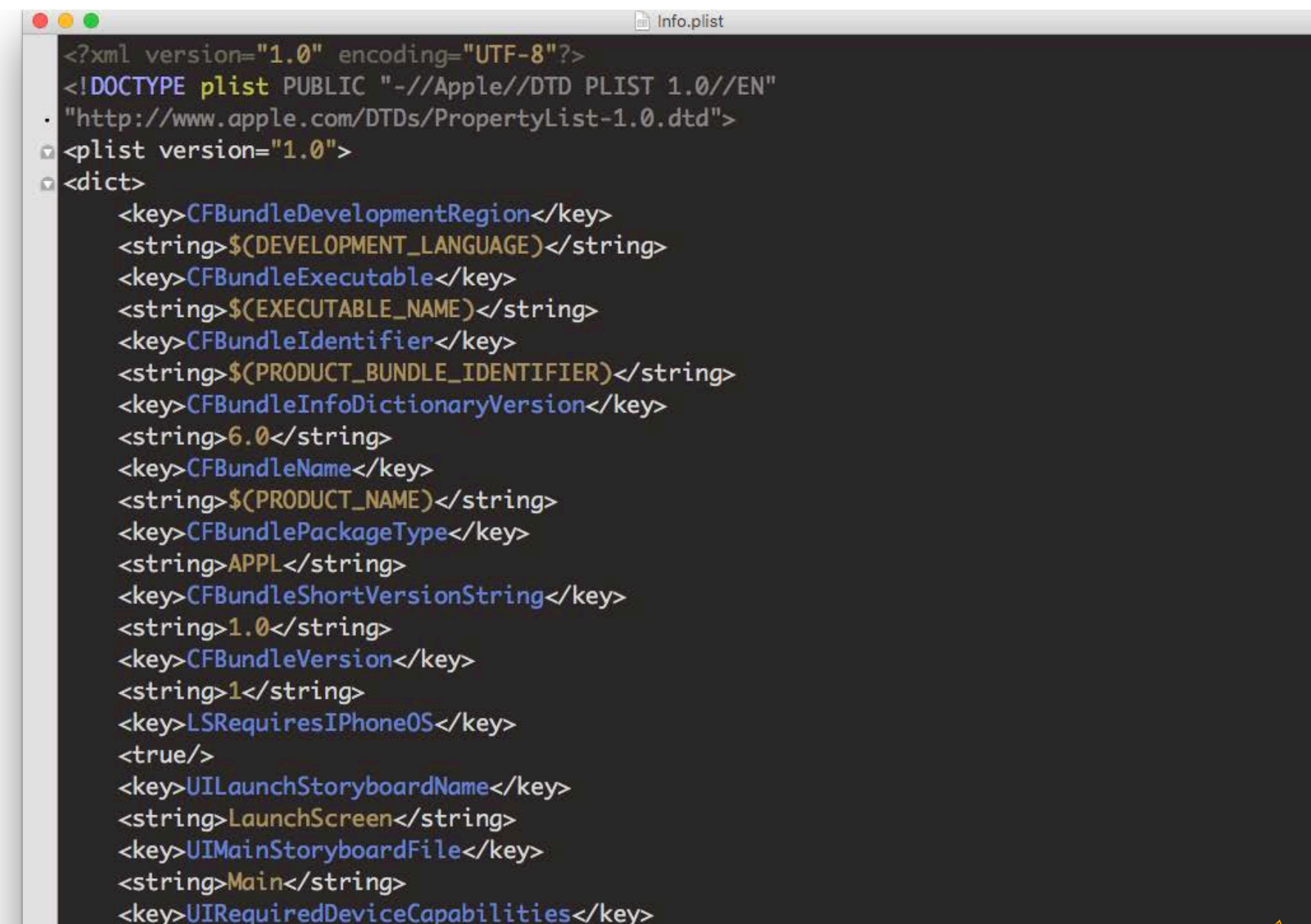


# Saving data into a file

- A common way to store data in iOS is to use propertyList files (.plist)
- pList files are XML files
- To store data in these files they need to be encoded using **PropertyListEncoder()**



Key	Type	Value
Information Property List	Dictionary	(15 items)
Localization native development re...	String	\$(DEVELOPMENT_LANGUAGE)
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone enviro...	Boolean	YES
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main
Required device capabilities	Array	(1 item)
Item 0	String	armv7
Status bar is initially hidden	Boolean	YES
Supported interface orientations	Array	(3 items)
Item 0	String	Portrait (bottom home button)
Item 1	String	Landscape (left home button)
Item 2	String	Landscape (right home button)
Supported interface orientations (i...	Array	(4 items)
Item 0	String	Portrait (bottom home button)
Item 1	String	Portrait (top home button)
Item 2	String	Landscape (left home button)
Item 3	String	Landscape (right home button)



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>CFBundleDevelopmentRegion</key>
  <string>$(DEVELOPMENT_LANGUAGE)</string>
  <key>CFBundleExecutable</key>
  <string>$(EXECUTABLE_NAME)</string>
  <key>CFBundleIdentifier</key>
  <string>$(PRODUCT_BUNDLE_IDENTIFIER)</string>
  <key>CFBundleInfoDictionaryVersion</key>
  <string>6.0</string>
  <key>CFBundleName</key>
  <string>$(PRODUCT_NAME)</string>
  <key>CFBundlePackageType</key>
  <string>APPL</string>
  <key>CFBundleShortVersionString</key>
  <string>1.0</string>
  <key>CFBundleVersion</key>
  <string>1</string>
  <key>LSRequiresIPhoneOS</key>
  <true/>
  <key>UILaunchStoryboardName</key>
  <string>LaunchScreen</string>
  <key>UIMainStoryboardFile</key>
  <string>Main</string>
  <key>UIRequiredDeviceCapabilities</key>
```



# Encoding data

- Classes and Structs that implement the `Codable` Protocol can be encoded
- The `Codable` Protocol requires the functions: `init(from:)` and `encode(to:)`
- If the classes and structs only contain properties that satisfies the `Codable` Protocol the functions: `init(from:)` and `encode(to:)` are not needed
- Almost all basic Swift types such as `String`, `Int`, and `Double` and Foundation types like `Date`, `Data`, and `URL` satisfies the `Codable` Protocol
- Also Collections such as `Arrays` and `Dictionaries` satisfies the `Codable` Protocol



# Encoding data

```
struct Note: Codable {  
    let title: String  
    let text: String  
    let timestamp: Date  
}  
  
let newNote = Note(title: "Grocery run", text: "Pick up mayonnaise, mustard, lettuce, tomato,  
and pickles.", timestamp:  
    Date())
```

# Encoding data

```
struct Note: Codable {
    let title: String
    let text: String
    let timestamp: Date
}

let newNote = Note(title: "Grocery run", text: "Pick up mayonnaise, mustard, lettuce, tomato,
and pickles.", timestamp:
    Date())

let propertyListEncoder = PropertyListEncoder()
if let encodedNote = try? propertyListEncoder.encode(newNote) {
    print(encodedNote)
}
```

# Error Handling

- When trying to encode data or read and write files errors can occur
- Swift handles error with the `do-try-catch` syntax or the keyword `try`?
- • Function, method, or initializer that can throw an error are marked with `throws`

```
func canThrowErrors() throws -> String
```

- Errors are represented by values of types that conform to the `Error` protocol

```
enum ErrorEnum: Error {  
    case error1  
    case error2  
    case error3  
}
```



# Error Handling

- A function that throws an error:

```
func canThrowErrors() throws -> String
{
    guard let unwrapOptional = someOptional else
    {
        throw ErrorEnum.error1
    }

    guard unwrapOptional < 5 else
    {
        throw ErrorEnum.error2
    }

    guard self.numberOfItems > 0 else
    {
        throw ErrorEnum.error3
    }

    return "result"
}
```

# Error Handling

- Use `do-try-catch` syntax to catch an error and handle them differently:

```
do {  
    try expression  
    statements  
} catch pattern 1 {  
    statements  
} catch pattern 2 where condition {  
    statements  
}
```

```
do {  
    try canThrowErrors()  
    // No Error  
} catch ErrorEnum.Error1 {  
    print("Error 1 occurred")  
} catch ErrorEnum.Error2 {  
    print("Error 2 occurred")  
}
```

# Error Handling

- If you want to handle all errors in the same way use `try?`

```
func fetchData() -> Data? {  
    if let data = try? fetchDataFromDisk() {  
        return data  
    }  
  
    if let data = try? fetchDataFromServer() {  
        return data  
    }  
    return nil  
}
```

- You can also disable errors with `try!` if you are sure that they will not be thrown

```
let photo = try! loadImage(atPath: "./Resources/John Appleseed.jpg")
```

# Encoding data

```
struct Note: Codable {
    let title: String
    let text: String
    let timestamp: Date
}

let newNote = Note(title: "Grocery run", text: "Pick up mayonnaise, mustard, lettuce, tomato,
and pickles.", timestamp:
    Date())

let propertyListEncoder = PropertyListEncoder()
if let encodedNote = try? propertyListEncoder.encode(newNote) {
    print(encodedNote)
}
```

# Encoding data

```
struct Note: Codable {
    let title: String
    let text: String
    let timestamp: Date
}

let newNote = Note(title: "Grocery run", text: "Pick up mayonnaise, mustard, lettuce, tomato,
and pickles.", timestamp:
    Date())

let propertyListEncoder = PropertyListEncoder()
if let encodedNote = try? propertyListEncoder.encode(newNote) {
    print(encodedNote)
}

let propertyListDecoder = PropertyListDecoder()
if let decodedNote = try? propertyListDecoder.decode(Note.self, from: encodedNote) {
    print(decodedNote)
}
```

# Writing data to a file

- iOS apps work in the sandbox model
- Your app can access the Documents directory to save data related to your app:

```
let documentsDirectory = FileManager.default.urls(for: .documentDirectory,  
                                                in: .userDomainMask).first!
```

- Create a plist file in in this directory:

```
let archiveURL = documentsDirectory.appendingPathComponent("notes_test")  
                                .appendingPathExtension("plist")
```

# Writing the data

```
// Encode data
let propertyListEncoder = PropertyListEncoder()
if let encodedNote = try? propertyListEncoder.encode(newNote) {
    print(encodedNote)
}
// Create file
let documentsDirectory = FileManager.default.urls(for: .documentDirectory,
                                                in: .userDomainMask).first!

let archiveURL = documentsDirectory.appendingPathComponent("notes_test")
                                .appendingPathExtension("plist")

// Write to file
try? encodedNote?.write(to: archiveURL, options: .noFileProtection)
```

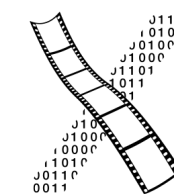
# Read the data from the file

```
let propertyListDecoder = PropertyListDecoder()

if let retrievedNoteData = try? Data(contentsOf: archiveURL),
    let decodedNote = try? propertyListDecoder.decode(Note.self, from: retrievedNoteData) {
    print(decodedNote)
}
```



# Emoji App Demo



# JSONEncoder and JSONDecoder

```
struct Employee: Codable {  
    var name: String  
    var id: Int  
    var favoriteCar: Car  
}
```

```
struct Car: Codable {  
    var name: String  
}
```

```
let car1 = Car(name: "Porsche");  
let employee1 = Employee(name: "John Appleseed", id: 7, favoriteToy:  
car1)
```

```
let jsonEncoder = JSONEncoder()  
let jsonData = try jsonEncoder.encode(employee1)  
// {"name":"John Appleseed","id":7,"favoriteCar":{"name":"Porsche"}}
```

```
let jsonDecoder = JSONDecoder()  
let employee2 = try jsonDecoder.decode(Employee.self, from:  
jsonData)
```

# Coding Keys

```
struct Employee: Codable {  
    var name: String  
    var id: Int  
    var favoriteCar: Car  
  
    enum CodingKeys: String, CodingKey {  
        case id = "employeeId"  
        case name  
        case favoriteCar  
    }  
}
```

# Manual Encoding and Decoding

```
{ "employeeId": 7, "name": "John Appleseed", "favoriteCar": {"name": "Porsche"}}
```

```
{ "employeeId": 7, "name": "John Appleseed", "car": "Porsche" }
```

```
func encode(to encoder: Encoder) throws {
    var container = encoder.container(keyedBy: CodingKeys.self)
    try container.encode(name, forKey: .name)
    try container.encode(id, forKey: .id)
    try container.encode(favoriteCar.name, forKey: .favoriteCar)
}
```

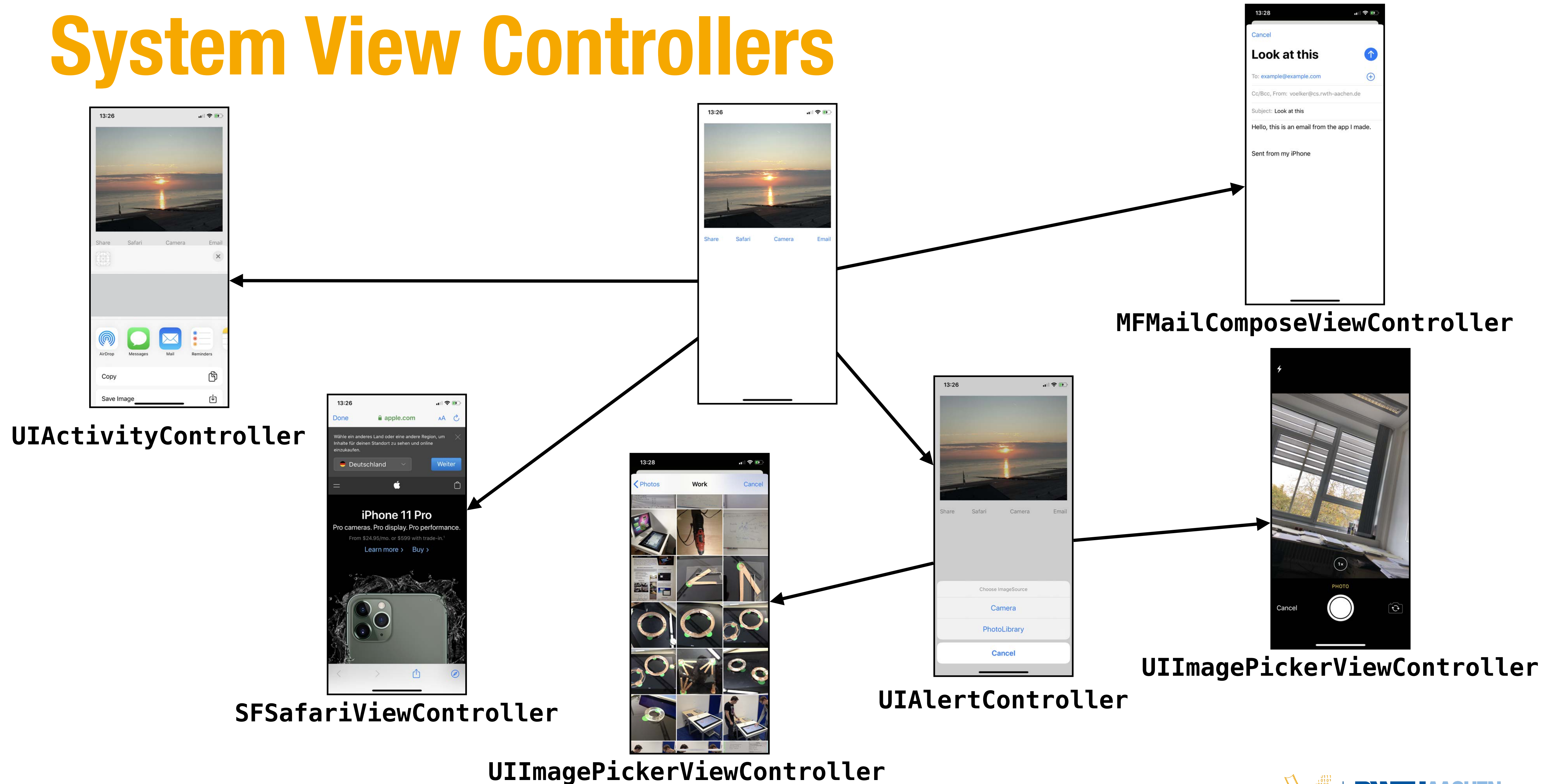
```
init(from decoder: Decoder) throws {
    let values = try decoder.container(keyedBy: CodingKeys.self)
    name = try values.decode(String.self, forKey: .name)
    id = try values.decode(Int.self, forKey: .id)
    let favoriteCar = try values.decode(String.self, forKey: .favoriteCar)
    favoriteCar = Car(name: .favoriteCar)
}
```

# System View Controllers

- System view controllers for displaying alerts, sharing content, sending messages, and accessing the camera and photo library on an iOS device
- The most common system view controllers are:
  - UINavigationController
  - SFSafariViewController
  - UIAlertController
  - UIImagePickerController
  - MFMailComposeViewController



# System View Controllers



# System View Controller Demo

