What’s New in Foundation

Session 212

Tony Parker, Foundation
Michael LeHew, Foundation
Itai Ferber, Foundation
New API Highlights
Key Paths and Key Value Observation
Encoding and Decoding
New API Highlights

Key Paths and Key Value Observation

Encoding and Decoding
New API Highlights

Key Paths and Key Value Observation

Encoding and Decoding
New API Highlights
Key Paths and Key Value Observation
Encoding and Decoding
New API Highlights
New Foundation Features

File provider communication
New Foundation Features

File provider communication

Improved available storage space API
New Foundation Features

File provider communication

Improved available storage space API

Improved `NSString` ↔ Swift `String` range conversion
New Foundation Features

File provider communication

Improved available storage space API

Improved `NSString <-> Swift String` range conversion

Discrete `NSProgress` support in `NSXPCConnection`
New Foundation Features

File provider communication

Improved available storage space API

Improved `NSString <-> Swift String` range conversion

Discrete `NSProgress` support in `NSXPCConnection`

Thermal notifications on iOS
New Foundation Features

File provider communication

Improved available storage space API

Improved `NSString ↔ Swift String` range conversion

Discrete `NSProgress` support in `NSXPCConnection`

Thermal notifications on iOS
Foundation Performance Improvements

Copy-on-write `NSArray`, `NSDictionary`, `NSSet`
Foundation Performance Improvements

Copy-on-write NSArray, NSDictionary, NSSet

Data inlining
Foundation Performance Improvements

Copy-on-write `NSArray, NSDictionary, NSSet`

Data inlining

Faster calendrical calculations with lower peak memory
Foundation Performance Improvements

Copy-on-write `NSArray`, `NSDictionary`, `NSSet`

Data inlining

Faster calendrical calculations with lower peak memory

Faster bridging of `NSNumber` to and from Swift
Foundation Performance Improvements

Copy-on-write NSArray, NSDictionary, NSSet

Data inlining

Faster calendrical calculations with lower peak memory

Faster bridging of NSNumber to and from Swift

Efficient Interactions with Frameworks

Hall 2

Friday 1:50PM
Key Paths and Key Value Observing

Michael LeHew, Foundation
Key Paths are important
// Swift 3 String Key Paths
// Swift 3 String Key Paths

@objcMembers class Kid : NSObject {
    dynamic var nickname: String = ""
    dynamic var age: Double = 0.0
    dynamic var bestFriend: Kid? = nil
    dynamic var friends: [Kid] = []
}

var ben = Kid(nickname: "Benji", age: 5.5)

let kidsNameKeyPath = #keyPath(Kid.nickname)

let name = ben.valueForKeyPath(kidsNameKeyPath)

ben.setValue("Ben", forKeyPath: kidsNameKeyPath)
// Swift 3 String Key Paths

@objcMembers class Kid : NSObject {
    dynamic var nickname: String = 
    dynamic var age: Double = 0.0
    dynamic var bestFriend: Kid? = nil
    dynamic var friends: [Kid] = []
}

var ben = Kid(nickname: "Benji", age: 5.5)

let kidsNameKeyPath = #keyPath(Kid.nickname)

let name = ben.valueForKeyPath(kidsNameKeyPath)
ben.setValue("Ben", forKeyPath: kidsNameKeyPath)
// Swift 3 String Key Paths

@objcMembers class Kid : NSObject {
    dynamic var nickname: String = ""
    dynamic var age: Double = 0.0
    dynamic var bestFriend: Kid? = nil
    dynamic var friends: [Kid] = []
}

to do

// Swift 3 String Key Paths

@objcMembers class Kid : NSObject {
    dynamic var nickname: String = ""
    dynamic var age: Double = 0.0
    dynamic var bestFriend: Kid? = nil
    dynamic var friends: [Kid] = []
}

var ben = Kid(nickname: "Benji", age: 5.5)

let kidsNameKeyPath = #keyPath(Kid.nickname)

let name = ben.valueForKeyPath(kidsNameKeyPath)
ben.setValue("Ben", forKeyPath: kidsNameKeyPath)
// Swift 3 String Key Paths

@objcMembers class Kid : NSObject {
    dynamic var nickname: String = 
    dynamic var age: Double = 0.0
    dynamic var bestFriend: Kid? = nil
    dynamic var friends: [Kid] = []
}

let ben = Kid(nickname: "Benji", age: 5.5)

let kidsNameKeyPath = #keyPath(Kid.nickname)

let name = ben.valueForKeyPath(kidsNameKeyPath)
ben.setValue("Ben", forKeyPath: kidsNameKeyPath)
// Swift 3 String Key Paths

@objcMembers class Kid : NSObject {
  dynamic var nickname: String = ""
  dynamic var age: Double = 0.0
  dynamic var bestFriend: Kid? = nil
  dynamic var friends: [Kid] = []
}

let ben = Kid(nickname: "Benji", age: 5.5)

let kidsNameKeyPath = #keyPath(Kid.nickname)

let name = ben.valueForKeyPath(kidsNameKeyPath)
ben.setValue("Ben", forKeyPath: kidsNameKeyPath)
// Swift 3 String Key Paths

@objcMembers class Kid : NSObject {
    dynamic var nickname: String = ""
    dynamic var age: Double = 0.0
    dynamic var bestFriend: Kid? = nil
    dynamic var friends: [Kid] = []
}

let ben = Kid(nickname: "Benji", age: 5.5)

let kidsNameKeyPath = #keyPath(Kid.nickname)
// String

let name = ben.valueForKeyPath(kidsNameKeyPath)
ben.setValue("Ben", forKeyPath: kidsNameKeyPath)
// Swift 3 String Key Paths

@objcMembers class Kid : NSObject {
    dynamic var nickname: String = ""
    dynamic var age: Double = 0.0
    dynamic var bestFriend: Kid? = nil
    dynamic var friends: [Kid] = []
}

let ben = Kid(nickname: "Benji", age: 5.5)

let kidsNameKeyPath = #keyPath(Kid.nickname)

let name = ben.valueForKeyPath(kidsNameKeyPath)

let kidsNameKeyPath = #keyPath(Kid.nickname)

let name = ben.valueForKeyPath(kidsNameKeyPath)

ben.setValue("Ben", forKeyPath: kidsNameKeyPath)
// Swift 3 String Key Paths

@objcMembers class Kid : NSObject {
    dynamic var nickname: String = ""
    dynamic var age: Double = 0.0
    dynamic var bestFriend: Kid? = nil
    dynamic var friends: [Kid] = []
}

let ben = Kid(nickname: "Benji", age: 5.5)

let kidsNameKeyPath = #keyPath(Kid.nickname)

let name = ben.valueForKeyPath(kidsNameKeyPath)  // valueForKeyPath(_: String) -> Any
ben.setValue("Ben", forKeyPath: kidsNameKeyPath)  // setValue(_, forKeyPath: String) -> Any
Key Paths
Key Paths

Property traversal
Key Paths

Property traversal

Statically type-safe
Key Paths

Property traversal

Statically type-safe

Fast
Key Paths

Property traversal

Statically type-safe

Fast

Applicable to all values
Key Paths

Property traversal

Statically type-safe

Fast

Applicable to all values

Works on all platforms
Key Paths

Property traversal

Statically type-safe

Fast

Applicable to all values

Works on all platforms
Key Paths

Property traversal

Statically type-safe

Fast

Applicable to all values

Works on all platforms

SE-0161 Smart Key Paths
\Kid.nickname
\Kid.nickname

Base
Type
\Kid\_nickname
\Kid\nickname
\.nickname

Property
Name
<table>
<thead>
<tr>
<th>Base Type</th>
<th>Property Name</th>
<th>Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>\Kid.nickname.characters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base Type</td>
<td>Property Name</td>
<td>Property Name</td>
</tr>
<tr>
<td>----------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>\Kid.bestFriend?.nickname</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Coming Soon
<table>
<thead>
<tr>
<th>Base Type</th>
<th>Property Name</th>
<th>Subscript</th>
</tr>
</thead>
<tbody>
<tr>
<td>\Kid.friends[0]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\Data.[.startIndex]

<table>
<thead>
<tr>
<th>Base Type</th>
<th>Subscript</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

.[.startIndex]

Coming Soon
# Uniform Syntax

<table>
<thead>
<tr>
<th>Types</th>
<th>Properties / Subscripts</th>
</tr>
</thead>
</table>

## Uniform Syntax

<table>
<thead>
<tr>
<th>Types</th>
<th>Properties / Subscripts</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>struct</code></td>
<td></td>
</tr>
<tr>
<td><code>class</code></td>
<td></td>
</tr>
<tr>
<td><code>@objc class</code></td>
<td></td>
</tr>
</tbody>
</table>
## Uniform Syntax

<table>
<thead>
<tr>
<th>Types</th>
<th>Properties / Subscripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td>let/var</td>
</tr>
<tr>
<td>class</td>
<td>get/set</td>
</tr>
<tr>
<td>@objc class</td>
<td>Stored or computed</td>
</tr>
</tbody>
</table>
let age = ben[keyPath: \Kid.age]
let age = ben[keyPath: \Kid.age]
let age = ben[keyPath: \Kid.age]
let age = ben[keyPath: \Kid.age]
let age = ben[keyPath: \Kid.age]
let age = ben[keyPath: \Kid.age]
let age = ben[keyPath: \Kid.age]

ben[keyPath: \Kid.nickname] = "Ben"
// Using Swift 4 KeyPaths

struct BirthdayParty {
    let celebrant: Kid
    var theme: String
    var attending: [Kid]
}
// Using Swift 4 KeyPaths

struct BirthdayParty {
    let celebrant: Kid
    var theme: String
    var attending: [Kid]
}

// Using Swift 4 KeyPaths

struct BirthdayParty {
    let celebrant: Kid
    var theme: String
    var attending: [Kid]
}


let birthdayKid = bensParty[keyPath: \BirthdayParty.celebrant]
// Using Swift 4 KeyPaths

struct BirthdayParty {
    let celebrant: Kid
    var theme: String
    var attending: [Kid]
}


let birthdayKid = bensParty[keyPath: 
BirthdayParty.celebrant]

bensParty[keyPath: 
BirthdayParty.theme] = "Pirate"
// Using Swift 4 KeyPaths

struct BirthdayParty {
    let celebrant: Kid
    var theme: String
    var attending: [Kid]
}


let birthdayKid = bensParty[keyPath: .celebrant]

bensParty[keyPath: .theme] = "Pirate"
// Using Swift 4 KeyPaths

struct BirthdayParty {
    let celebrant: Kid
    var theme: String
    var attending: [Kid]
}


let birthdayKid = bensParty[keyPath: ".celebrant"]

bensParty[keyPath: ".theme"] = "Pirate"
struct BirthdayParty {
    let celebrant: Kid
    var theme: String
    var attending: [Kid]
}


let birthdayKid = bensParty[keyPath: \.celebrant]

bensParty[keyPath: \.theme] = "Ninja"
\Kid.nickname
let nicknameKeyPath = \Kid.nickname
let nicknameKeyPath = \Kid.nickname
let nicknameKeyPath = \Kid.nickname

KeyPath<Kid, String>
let nicknameKeyPath = \Kid.nickname
// Key Paths and Properties

let birthdayKidsAgeKeyPath = \BirthdayParty.celebrant.age
// Key Paths and Properties

let birthdayKidsAgeKeyPath = \BirthdayParty.celebrant.age
// Key Paths and Properties

let birthdayKidsAgeKeyPath = \BirthdayParty.celebrant.age

let birthdayBoysAge = bensParty[keyPath: birthdayKidsAgeKeyPath]
// Key Paths and Properties

let birthdayKidsAgeKeyPath = \BirthdayParty.celebrant.age

let birthdayBoysAge = bensParty[keyPath: birthdayKidsAgeKeyPath]
// Key Paths and Properties

let birthdayKidsAgeKeyPath = 
BirthdayParty.celebrant.age

let birthdayBoysAge = bensParty[keyPath: birthdayKidsAgeKeyPath]

let mia = Kid(nickname: "Mia", age: 4.5)
let miasParty = BirthdayParty(celebrant: mia, theme: "Space", attending: [])
// Key Paths and Properties

let birthdayKidsAgeKeyPath = \BirthdayParty.celebrant.age

let birthdayBoysAge = bensParty[keyPath: birthdayKidsAgeKeyPath]

let mia = Kid(nickname: "Mia", age: 4.5)
let miasParty = BirthdayParty(celebrant: mia, theme: "Space", attending: [])

let birthdayGirlsAge = miasParty[keyPath: birthdayKidsAgeKeyPath]
/ Appending Key Paths
// Appending Key Paths

func partyPersonsAge(party: BirthdayParty,
                     participantPath: KeyPath<BirthdayParty, Kid>) -> Double {

// Appending Key Paths

func partyPersonsAge(party: BirthdayParty,
                     participantPath: KeyPath<BirthdayParty, Kid>) -> Double {
}
// Appending Key Paths

func partyPersonsAge(party: BirthdayParty,
                      participantPath: KeyPath<BirthdayParty, Kid> -> Double {
func partyPersonsAge(party: BirthdayParty,
        participantPath: KeyPath<BirthdayParty, Kid>) -> Double {
    let kidsAgeKeyPath = participantPath.appending(\.age)
}

// Appending Key Paths
// Appending Key Paths

func partyPersonsAge(party: BirthdayParty, 
    participantPath: KeyPath<BirthdayParty, Kid>) -> Double {
    let kidsAgeKeyPath = participantPath.appending(\age)
}

KeyPath<BirthdayParty, Double>
func partyPersonsAge(party: BirthdayParty,
        participantPath: KeyPath<BirthdayParty, Kid>) -> Double {
    let kidsAgeKeyPath = participantPath.appending(\.age)
    return party[keyPath: kidsAgeKeyPath]
}
// Appending Key Paths

func partyPersonsAge(party: BirthdayParty, participantPath: KeyPath<BirthdayParty, Kid>) -> Double {
    let kidsAgeKeyPath = participantPath.appending(".age")
    return party[keyPath: kidsAgeKeyPath]
}

let birthdayBoysAge = partyPersonsAge(bensParty, ".celebrant")
// Appending Key Paths

func partyPersonsAge(party: BirthdayParty,
                      participantPath: KeyPath<BirthdayParty, Kid>) -> Double {
    let kidsAgeKeyPath = participantPath.appending(.age)
    return party[keyPath: kidsAgeKeyPath]
}

let birthdayBoysAge = partyPersonsAge(bensParty, .celebrant)

// Coming Soon
let firstAttendeesAge = partyPersonsAge(bensParty, .attendees[0])
The `.appending` Rule

`\BirthdayParty.celebrant.appending(\Kid.age)`
The `.appending` Rule

`\BirthdayParty.celebrant` .appending `\Kid.age`
The `.appending` Rule

```
\BirthdayParty.celebrant
  .appending \Kid.age
\BirthdayParty.celebrant.age
```
The `.appending` Rule

\[ \text{BirthdayParty.celebrant.} .appending . Kid.age \]

\[ \text{BirthdayParty.celebrant.age} \]

- KeyPath<BirthdayParty, Kid>
- KeyPath<Kid, Double>
- KeyPath<BirthdayParty, Double>
The .appending Rule

KeyPath<\textit{BirthdayParty}, \textit{Kid}>

KeyPath<\textit{Kid}, \textit{Double}>

KeyPath<\textit{BirthdayParty}, \textit{Double}>
The `.appending` Rule

```
KeyPath<BirthdayParty, Kid>
```

```
KeyPath<Kid, Double>
```

```
KeyPath<BirthdayParty, Double>
```
The `.appending` Rule

KeyPath:<BirthdayParty, Kid>

KeyPath:<Kid, Double>

KeyPath:<BirthdayParty, Double>
The .appending Rule

KeyPath<BirthdayParty, Double>
The .appendining Rule

KeyPath<**BirthdayParty**, Double>

Base
Type
The `.appending` Rule

```
KeyPath<BirthdayParty, Double>
  Base       Property
  Type       Type
```
// Type Erased Key Paths
// Type Erased Key Paths

let titles = ["Theme", "Attending", "Birthday Kid"]
let partyPaths = [BirthdayParty.theme, BirthdayParty.attending, BirthdayParty.celebrant]
// Type Erased Key Paths

let titles = 
["Theme", "Attending", "Birthday Kid"]

let partyPaths = 
[\BirthdayParty.theme, \BirthdayParty.attending, \BirthdayParty.celebrant]
// Type Erased Key Paths

let titles = ["Theme", "Attending", "Birthday Kid"]

let partyPaths = [
  \BirthdayParty.theme,
  \BirthdayParty.attending,
  \BirthdayParty.celebrant
]
// Type Erased Key Paths

let titles = ["Theme", "Attending", "Birthday Kid"]
let partyPaths = [
BirthdayParty.theme,
BirthdayParty.attending,
BirthdayParty.celebrant]
// Type Erased Key Paths

let titles = ["Theme", "Attending", "Birthday Kid"]
let partyPaths = [\BirthdayParty.theme, \BirthdayParty.attending, \BirthdayParty.celebrant]
// Type Erased Key Paths

let titles = [
    "Theme",
    "Attending",
    "Birthday Kid"
]

let partyPaths = [
    BirthdayParty.theme,
    BirthdayParty.attending,
    BirthdayParty.celebrant
]

for (title, partyPath) in zip(titles, partyPaths) {
    let partyValue = miasParty[keyPath: partyPath]
    print("\(title)\n\(partyValue)\n")
}
// Type Erased Key Paths

let titles = ["Theme", "Attending", "Birthday Kid"]
let partyPaths = [
    BirthdayParty.theme,
    BirthdayParty.attending,
    BirthdayParty.celebrant
]

for (title, partyPath) in zip(titles, partyPaths) {
    let partyValue = miasParty[keyPath: partyPath]
    print("\(title)\n\(partyValue)\n")
}
/ Type Erased Key Paths

let titles = ["Theme", "Attending", "Birthday Kid"]
let partyPaths = [BirthdayParty.theme, BirthdayParty.attending, BirthdayParty.celebrant]

for (title, partyPath) in zip(titles, partyPaths) {
    let partyValue = miasParty[keyPath: partyPath]
    print("\(title)\n\(partyValue)\n")
}
// Type Erased Key Paths

let titles = ["Theme", "Attending", "Birthday Kid"]
let partyPaths = [
\BirthdayParty.theme, \BirthdayParty.attending, \BirthdayParty.celebrant]

for (title, partyPath) in zip(titles, partyPaths) {
    let partyValue = miasParty[keyPath: partyPath]
    print("\(title)\n\(partyValue)\n")
}
// Type Erased Key Paths

let titles = ["Theme", "Attending", "Birthday Kid"]
let partyPaths = [
    BirthdayParty.theme,
    BirthdayParty.attending,
    BirthdayParty.celebrant
]

for (title, partyPath) in zip(titles, partyPaths) {
    let partyValue = miasParty[keyPath: partyPath]
    print("\(title)\n\(partyValue)\n")
}
// Mutating Key Paths

extension BirthdayParty {

}
// Mutating Key Paths

extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {

    }
}
extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
    
    }
}
// Mutating Key Paths

extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: \.celebrant.age)
extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: \.celebrant.age)
// Mutating Key Paths

extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: ".celebrant.age")
// Mutating Key Paths

extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: \.celebrant.age)

error: Cannot assign to immutable expression of type 'Double'
// Mutating Key Paths

extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: \.celebrant.age)
extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: \.celebrant\.age)
extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: \.celebrant.age)
// Mutating Key Paths

extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: \.celebrant.age)
extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: \.celebrant.age)
// Mutating Key Paths

extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: \.celebrant.age)
extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: \.celebrant.age)
// Mutating Key Paths

extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: ".celebrant.age")
extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: \.celebrant.age)

@objcMembers
class Kid : NSObject {
    dynamic var age: Double
}
// Mutating Key Paths

extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: \.celebrant.age)
// Mutating Key Paths

extension BirthdayParty {
    func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>)
    {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: \.celebrant.age)
// Mutating Key Paths

extension BirthdayParty {
    mutating func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: ".celebrant.age")

@objcMembers
class Kid : NSObject {
    dynamic var age: Double
}

struct BirthdayParty {
    let celebrant: Kid
}
// Mutating Key Paths

extension BirthdayParty {
    mutating func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: \.celebrant.age)

@objcMembers
class Kid : NSObject {
    dynamic var age: Double
}

struct BirthdayParty {
    let celebrant: Kid
}
// Mutating Key Paths

extension BirthdayParty {
    mutating func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath:\.celebrant.age)
// Mutating Key Paths

extension BirthdayParty {
    mutating func blowCandles(ageKeyPath: WritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: ".celebrant.age")

@objcMembers
class Kid : NSObject {
    dynamic var age: Double
}

struct BirthdayParty {
    let celebrant: Kid
}
/** Mutating Key Paths**

extension BirthdayParty {
    func blowCandles(ageKeyPath: ReferenceWritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: \.celebrant.age)
// Mutating Key Paths

extension BirthdayParty {
    func blowCandles(ageKeyPath: ReferenceWritableKeyPath<BirthdayParty, Double>) {
        let age = self[keyPath: ageKeyPath]
        self[keyPath: ageKeyPath] = floor(age) + 1.0
    }
}

bensParty.blowCandles(ageKeyPath: ".celebrant.age")

assert(6.0 == ben.age)
WritableKeyPath

Write directly into value-type base (inout/mutating)
WritableKeyPath
Write directly into value-type base (inout/mutating)

ReferenceWritableKeyPath
Write into a reference-type base
PartialKeyPath<Base>

KeyPath<Base, Property>

WritableKeyPath<Base, Property>

ReferenceWritableKeyPath<Base, Property>
AnyKeyPath

PartialKeyPath<Base>

KeyPath<Base, Property>

WritableKeyPath<Base, Property>

ReferenceWritableKeyPath<Base, Property>
Read-Only Properties
Read-Only Properties

KeyPath
Read-Write Properties
Read-Write Properties
Mutable value type base
Read-Write Properties
Mutable value type base

WritableKeyPath
Read-Write Properties
Immutable value type base
Read-Write Properties
Immutable value type base

KeyPath
Read-Write Properties
Reference type base
Read-Write Properties
Reference type base

ReferenceWritableKeyPath
// Key Paths Capture By Value
// Coming Soon
// Key Paths Capture By Value
// Coming Soon

var index = 0
let whichKidKeyPath = \BirthdayParty.attendees[index]
let firstAttendeesAge = partyPersonsAge(party, whichKidKeyPath)
var index = 0
let whichKidKeyPath = \BirthdayParty.attendees[index]
let firstAttendeesAge = partyPersonsAge(party, whichKidPath)
var index = 0
let whichKidKeyPath = \BirthdayParty.attendees[index]
let firstAttendeesAge = partyPersonsAge(party, whichKidKeyPath)
// Key Paths Capture By Value
// Coming Soon

var index = 0
let whichKidKeyPath = \BirthdayParty.attendees[index]
let firstAttendeesAge = partyPersonsAge(party, whichKidKeyPath)
// Key Paths Capture By Value
// Coming Soon

var index = 0
let whichKidKeyPath = \BirthdayParty.attendees[index]
let firstAttendeesAge = partyPersonsAge(party, whichKidKeyPath)

index = 1
let sameAge = partyPersonsAge(party, whichKidKeyPath)
// Key Paths Capture By Value
// Coming Soon

var index = 0

let whichKidKeyPath = \BirthdayParty.attendees[index]

let firstAttendeesAge = partyPersonsAge(party, whichKidKeyPath)

index = 1

let sameAge = partyPersonsAge(party, whichKidKeyPath)
// Key Paths Capture By Value
// Coming Soon

var index = 0
let whichKidKeyPath = \BirthdayParty.attendees[index]
let firstAttendeesAge = partyPersonsAge(party, whichKidKeyPath)

index = 1
let sameAge = partyPersonsAge(party, whichKidKeyPath)
Key Paths
Key Value Observing
let observation = mia.observe(.age) {
    mia
}
let observation = mia.observe(.age) { ... }
let observation = mia.observe(
.age) {
...
}

Observe using key path
let observation = mia.observe(\.age) { ... }

Observe using
key path
let observation = mia.observe(.age) {
    ...
}
let observation = mia.observe\.age { observed, change in
}

let observation = mia.observe(
  \.age
) {
  observed, change in
}

Kid
let observation = mia.observe(
    \.age
) { observed, change in
}

Kid

NSKeyValueObservedChange<Double>
// Cocoa Adoption of Key Paths: KVO

@objcMembers class KindergartenController : NSObject {
    dynamic var representedKid: Kid

    init(kid: Kid) {
        representedKid = kid
    }
}
// Cocoa Adoption of Key Paths: KVO

@objcMembers class KindergartenController : NSObject {
    dynamic var representedKid: Kid

    var ageObservation: NSKeyValueObservation

    init(kid: Kid) {
        representedKid = kid
    }

}
@objcMembers class KindergartenController: NSObject {
    dynamic var representedKid: Kid
    var ageObservation: NSKeyValueObservation
    init(kid: Kid) {
        representedKid = kid
        ageObservation = observe(\.representedKid.age) { observed, change in } // Missing in the screenshot
// Cocoa Adoption of Key Paths: KVO

@objcMembers class KindergartenController : NSObject {
    dynamic var representedKid: Kid
    var ageObservation: NSKeyValueObservation

    init(kid: Kid) {
        representedKid = kid
        ageObservation = observe(.representedKid.age) { observed, change in
            if observed.kid.age > 5 {
                print("Happy birthday \(observed.kid.nickname)! Time for kindergarten!")
            }
        }
    }
}
@objcMembers class KindergartenController : NSObject {
    dynamic var representedKid: Kid
    var ageObservation: NSKeyValueObservation

    init(kid: Kid) {
        representedKid = kid
        ageObservation = observe(.representedKid.age) { observed, change in
            if observed.kid.age > 5 {
                print("Happy birthday \(observed.kid.nickname)! Time for kindergarten!")
            }
        }
    }
}

let controller = KindergartenController(kid: mia)
miasParty.blowCandles(\.celebrant.age)
// Cocoa Adoption of Key Paths: KVO

@objcMembers class KindergartenController : NSObject {
    dynamic var representedKid: Kid
    var ageObservation: NSKeyValueObservation

    init(kid: Kid) {
        representedKid = kid
        ageObservation = observe(.representedKid.age) { observed, change in
            if observed.kid.age > 5 {
                print("Happy birthday \(observed.kid.nickname)!! Time for kindergarten!")
            }
        }
    }
}

let controller = KindergartenController(kid: mia)
miasParty.blowCandles(.celebrant.age)

Happy birthday Mia! Time for kindergarten!
#keyPath(Kid.nickname)
#keyPath(Kid.nickname)

\Kid.nickname
\Kid.nickname
Encoding and Decoding
Encoding and Decoding

Conversion between Swift data structures and archived formats
Encoding and Decoding

Conversion between Swift data structures and archived formats

Swift and archived formats have strong typing mismatch
Encoding and Decoding

Conversion between Swift data structures and archived formats

Swift and archived formats have strong typing mismatch

Solution is close integration with Swift
{"name": "Monalisa Octocat",
"email": "support@github.com",
"date": "2011-04-14T16:00:49Z"}
{  
  "name": "Monalisa Octocat",  
  "email": "support@github.com",  
  "date": "2011-04-14T16:00:49Z"
}

struct Author {  
  let name: String  
  let email: String  
  let date: Date
}
struct Author: Codable {
    let name: String
    let email: String
    let date: Date
}
let jsonData = ""
{
    "name": "Monalisa Octocat",
    "email": "support@github.com",
    "date": "2011-04-14T16:00:49Z"
}
"".data(using: .utf8)!

struct Author : Codable {
    let name: String
    let email: String
    let date: Date
}
let jsonData = """"{
    "name": "Monalisa Octocat",
    "email": "support@github.com",
    "date": "2011-04-14T16:00:49Z"
}
"""".data(using: .utf8)!

struct Author: Codable {
    let name: String
    let email: String
    let date: Date
}

let decoder = JSONDecoder()
let jsonData = """
{
    "name": "Monalisa Octocat",
    "email": "support@github.com",
    "date": "2011-04-14T16:00:49Z"
}
""".data(using: .utf8)!

struct Author : Codable {
    let name: String
    let email: String
    let date: Date
}

let decoder = JSONDecoder()
let jsonData = 
{
   "name": "Monalisa Octocat",
   "email": "support@github.com",
   "date": "2011-04-14T16:00:49Z"
}

"".data(using: .utf8)!

struct Author: Codable {
    let name: String
    let email: String
    let date: Date
}

let decoder = JSONDecoder()

decoder.dateDecodingStrategy = .iso8601
let jsonData = ""
{
    "name": "Monalisa Octocat",
    "email": "support@github.com",
    "date": "2011-04-14T16:00:49Z"
}
"
"".data(using: .utf8)!

struct Author : Codable {
    let name: String
    let email: String
    let date: Date
}

let decoder = JSONDecoder()
decoder.dateDecodingStrategy = .iso8601
let jsonData = ""
{
    "name": "Monalisa Octocat",
    "email": "support@github.com",
    "date": "2011-04-14T16:00:49Z"
}
"".data(using: .utf8)!

struct Author : Codable {
    let name: String
    let email: String
    let date: Date
}

let decoder = JSONDecoder()
decoder.dateDecodingStrategy = .iso8601
let author = try decoder.decode(Author.self, from: jsonData)
let jsonData = ""
{
    "name": "Monalisa Octocat",
    "email": "support@github.com",
    "date": "2011-04-14T16:00:49Z"
}
"
"".data(using: .utf8)!

struct Author : Codable {
    let name: String
    let email: String
    let date: Date
}

let decoder = JSONDecoder()
decoder.dateDecodingStrategy = .iso8601
let author = try decoder.decode(Author.self, from: jsonData)
{"name": "Monalisa Octocat",
"email": "support@github.com",
"date": "2011-04-14T16:00:49Z"}
{"name": "Monalisa Octocat",
"email": "support@github.com",
"date": "2011-04-14T16:00:49Z"}
{
    "name": "Monalisa Octocat",
    "email": "support@github.com",
    "date": "2011-04-14T16:00:49Z"
}
{
"url": "https://api.github.com/.../6dcb09",
"author": {
"name": "Monalisa Octocat",
"email": "support@github.com",
"date": "2011-04-14T16:00:49Z"
}
},
"message": "Fix all the bugs",
"comment_count": 0,
}


{
  "url": "https://api.github.com/.../6dcb09",
  "author": {
    "name": "Monalisa Octocat",
    "email": "support@github.com",
    "date": "2011-04-14T16:00:49Z"
  },
  "message": "Fix all the bugs",
  "comment_count": 0,
}

struct Author : Codable {
  let name: String
  let email: String
  let date: Date
}
{ "url": "https://api.github.com/.../6dcb09",
"author": {
   "name": "Monalisa Octocat",
   "email": "support@github.com",
   "date": "2011-04-14T16:00:49Z"
},
"message": "Fix all the bugs",
"comment_count": 0,
}

struct Author : Codable {
    let name: String
    let email: String
    let date: Date
}


```
{
   "url": "https://api.github.com/.../6dcb09",
   "author": {
      "name": "Monalisa Octocat",
      "email": "support@github.com",
      "date": "2011-04-14T16:00:49Z"
   },
   "message": "Fix all the bugs",
   "comment_count": 0,
}
```

```swift
struct Commit : Codable {
    let url: URL
    struct Author : Codable {
        let name: String
        let email: String
        let date: Date
    }
    let author: Author
    let message: String
    let comment_count: Int
}
```
let commit = try decoder.decode(Commit.self, from: jsonData)
let commit = try decoder.decode(Commit.self, from: jsonData)
let commit = try decoder.decode(Commit.self, from: jsonData)
let commitDate = commit.author.date
{  
  "url": "https://api.github.com/.../6dcb09",
  "author": {  
    "name": "Monalisa Octocat",
    "email": "support@github.com",
    "date": "2011-04-14T16:00:49Z"
  },
  "message": "Fix all the bugs",
  "comment_count": 0,
}

let commit = try decoder.decode(Commit.self, from: jsonData)

let commitDate = commit.author.date
Coding Protocols

Codable

typealias Codable = Encodable & Decodable
Coding Protocols

Codable

typealias Codable = Encodable & Decodable

Encodable

public protocol Encodable {
    func encode(to encoder: Encoder) throws
}

Coding Protocols

Codable

typealias Codable = Encodable & Decodable

Encodable

public protocol Encodable {
    func encode(to encoder: Encoder) throws
}

Decodable

public protocol Decodable {
    init(from decoder: Decoder) throws
}
Coding Protocols

Use Swift protocol extension behavior
Coding Protocols

Use Swift protocol extension behavior

Write your own implementation to customize
struct Commit : Codable {
    struct Author : Codable {
    } // ... */
    let url: URL
    let message: String
    let author: Author
    let comment_count: Int
struct Commit: Codable {
    struct Author: Codable {
    }
    let url: URL
    let message: String
    let author: Author
    let comment_count: Int
}

// Encodable
public func encode(to encoder: Encoder) throws {
}

// Decodable
init(from decoder: Decoder) throws {
}
struct Commit: Codable {
    struct Author: Codable { /* … */ }
    let url: URL
    let message: String
    let author: Author
    let comment_count: Int
struct Commit : Codable {
    struct Author : Codable { /* ... */ }
    let url: URL
    let message: String
    let author: Author
    let comment_count: Int
struct Commit : Codable {
    struct Author : Codable { /* … */ }
    let url: URL
    let message: String
    let author: Author
    let comment_count: Int
struct Commit: Codable {
    struct Author: Codable { /* ... */ }
    let url: URL
    let message: String
    let author: Author
    let comment_count: Int

    private enum CodingKeys: String, CodingKey {
        case url
        case message
        case author
        case comment_count
    }
}
struct Commit: Codable {
  struct Author: Codable { /* … */ }
  let url: URL
  let message: String
  let author: Author
  let comment_count: Int

private enum CodingKeys: String, CodingKey {
  case url
  case message
  case author
  case comment_count
}
struct Commit: Codable {
    struct Author: Codable { /* … */ }
    let url: URL
    let message: String
    let author: Author
    let comment_count: Int
}

private enum CodingKeys: String, CodingKey {
    case url
    case message
    case author
    case comment_count
}
struct Commit: Codable {
    struct Author: Codable { /* … */ }
    let url: URL
    let message: String
    let author: Author
    let comment_count: Int

    private enum CodingKeys: String, CodingKey {
        case url
        case message
        case author
        case comment_count
    }
}
struct Commit : Codable {
    struct Author : Codable { /* ... */ }
    let url: URL
    let message: String
    let author: Author
    let comment_count: Int
}

private enum CodingKeys : String, CodingKey {
    case url
    case message
    case author
    case comment_count
}
struct Commit: Codable {
    struct Author: Codable {
        /* … */
    }
    let url: URL
    let message: String
    let author: Author
    let commentCount: Int
}

private enum CodingKeys: String, CodingKey {
    case url
    case message
    case author
    case commentCount = "comment_count"
}
struct Commit : Codable {
    struct Author : Codable { /* … */ }
    let url: URL
    let message: String
    let author: Author
    let commentCount: Int

    private enum CodingKeys : String, CodingKey {
        case url
        case message
        case author
        case commentCount = "comment_count"
    }
}
Demo

Encoding and Decoding

Itai Ferber, Foundation
Encoding and Decoding

Tony Parker, Foundation
Codable Philosophy

Error handling built-in

Encapsulate encoding details

Abstract format from types
Codable Philosophy

Error handling built-in
Encapsulate encoding details
Abstract format from types
Error Handling

Unexpected input is not if, but when
Error Handling

Unexpected input is not if, but when

No fatal errors from untrusted data—only for developer mistakes
Error Handling

Unexpected input is not if, but when

No fatal errors from untrusted data—only for developer mistakes

Errors possible on decode and encode
Coder Errors

Encoding
  • Invalid value
Coder Errors

Encoding
• Invalid value

Decoding
• Type mismatch
• Missing key
• Missing value
• Data corrupt
Beyond Basic Error Handling
Beyond Basic Error Handling
Beyond Basic Error Handling

Bytes

Structured bytes
Beyond Basic Error Handling

Bytes

Structured bytes

Typed data
Beyond Basic Error Handling

- Domain-specific validation
- Typed data
- Structured bytes
- Bytes
Beyond Basic Error Handling

- Graph-level validation
- Domain-specific validation
- Typed data
- Structured bytes
- Bytes
struct Commit: Codable {
    struct Author: Codable {
        /* … */
    }
    let url: URL
    let message: String
    let author: Author
    let commentCount: Int

    private enum CodingKeys: String, CodingKey {
        /* … */
    }
}
struct Commit : Codable {
    struct Author : Codable { /* … */ }

    let url: URL
    let message: String
    let author: Author
    let commentCount: Int

    private enum CodingKeys : String, CodingKey { /* … */ }

    public init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        url = try container.decode(URL.self, forKey: .url)
        message = try container.decode(String.self, forKey: .message)
        author = try container.decode(Author.self, forKey: .author)
        commentCount = try container.decode(Int.self, forKey: .commentCount)
    }
}
struct Commit : Codable {
    struct Author : Codable {
    } /* ... */
    let url: URL
    let message: String
    let author: Author
    let commentCount: Int
    private enum CodingKeys : String, CodingKey { /* ... */ }
    public init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        url = try container.decode(URL.self, forKey: .url)
        message = try container.decode(String.self, forKey: .message)
        author = try container.decode(Author.self, forKey: .author)
        commentCount = try container.decode(Int.self, forKey: .commentCount)
    }
}
struct Commit : Codable {
    struct Author : Codable { /* … */ }

    let url: URL
    let message: String
    let author: Author
    let commentCount: Int

    private enum CodingKeys : String, CodingKey { /* … */ }

    public init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)

        url = try container.decode(URL.self, forKey: .url)
        message = try container.decode(String.self, forKey: .message)
        author = try container.decode(Author.self, forKey: .author)
        commentCount = try container.decode(Int.self, forKey: .commentCount)
    }
}
struct Commit : Codable {

    struct Author : Codable {
        /* … */
    }

    let url: URL
    let message: String
    let author: Author
    let commentCount: Int

    private enum CodingKeys : String, CodingKey {
        /* … */
    }

    public init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        url = try container.decode(URL.self, forKey: .url)
        message = try container.decode(String.self, forKey: .message)
        author = try container.decode(Author.self, forKey: .author)
        commentCount = try container.decode(Int.self, forKey: .commentCount)
    }
}
}
struct Commit : Codable {
    struct Author : Codable { /* … */ }
    let url: URL
    let message: String
    let author: Author
    let commentCount: Int
    private enum CodingKeys : String, CodingKey { /* … */ }
    public init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        url = try container.decode(URL.self, forKey: .url)
        message = try container.decode(String.self, forKey: .message)
        author = try container.decode(Author.self, forKey: .author)
        commentCount = try container.decode(Int.self, forKey: .commentCount)
    }
}
struct Commit : Codable {
    struct Author : Codable { /* … */ }
    let url: URL
    let message: String
    let author: Author
    let commentCount: Int

    private enum CodingKeys : String, CodingKey { /* … */ }

    public init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        url = try container.decode(URL.self, forKey: .url)
        message = try container.decode(String.self, forKey: .message)
        author = try container.decode(Author.self, forKey: .author)
        commentCount = try container.decode(Int.self, forKey: .commentCount)

        guard url.scheme == "https" else {
            throw DecodingError.dataCorrupted(DecodingError.Context(
                codingPath: container.codingPath + [CodingKeys.url],
                debugDescription: "URLs require https"))
        }
    }
}
Codable Philosophy

Error handling built-in
Encapsulate encoding details
Abstract format from types
Encapsulate Encoding Details

Keys and values are private

Containers provide storage for values
Keyed Containers
Keyed Containers

{ }

- Key: Value
- Key: Value
- Key: Value
Coding Keys
Strongly-typed replacement for String keys

```swift
public protocol CodingKey {
    var stringValue: String { get }
    var intValue: Int? { get }

    init?(stringValue: String)
    init?(intValue: Int)
}
```
Coding Keys
private enum CodingKeys : String, CodingKey {
    case url
    case author
    case comment_count
}
private enum CodingKeys: String, CodingKey {
    case url
    case author
    case comment_count
}

| Case Name       | stringValue | intValue?
|-----------------|-------------|------------
| url             | url         | nil        |
| author          | author      | nil        |
| comment_count   | comment_count | nil        |
private enum CodingKeys : String, CodingKey {
    case url
    case author
    case commentCount = "comment_count"
}
private enum CodingKeys : Int, CodingKey {
    case url = 42
    case author = 100
    case comment_count
}
private enum CodingKeys : Int, CodingKey {
    case url = 42
    case author = 100
    case comment_count
}

<table>
<thead>
<tr>
<th>Case Name</th>
<th>stringValue</th>
<th>intValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>url</td>
<td>42</td>
</tr>
<tr>
<td>author</td>
<td>author</td>
<td>100</td>
</tr>
<tr>
<td>comment_count</td>
<td>comment_count</td>
<td>101</td>
</tr>
</tbody>
</table>
Unkeyed Containers
Unkeyed Containers

[ Value , Value , Value , Value ]
Single Value Containers
Single Value Containers
struct Commit : Codable {
    struct Author : Codable { /* … */ }
    let url: URL
    let message: String
    let author: Author
    let commentCount: Int

    private enum CodingKeys : String, CodingKey { /* … */ }

    public init(from decoder: Decoder) throws { /* … */ }
}
struct Commit : Codable {
    struct Author : Codable { /* … */ }
    let url: URL
    let message: String
    let author: Author
    let commentCount: Int
    private enum CodingKeys : String, CodingKey { /* … */ }
    public init(from decoder: Decoder) throws { /* … */ }
    public func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(url, forKey: .url)
        try container.encode(message, forKey: .message)
        try container.encode(author, forKey: .author)
        try container.encode(commentCount, forKey: .commentCount)
    }
}
struct Commit : Codable {
    struct Author : Codable { /* … */ }
    let url: URL
    let message: String
    let author: Author
    let commentCount: Int

    private enum CodingKeys : String, CodingKey { /* … */ }
    public init(from decoder: Decoder) throws { /* … */ }
    public func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(url, forKey: .url)
        try container.encode(message, forKey: .message)
        try container.encode(author, forKey: .author)
        try container.encode(commentCount, forKey: .commentCount)
    }
}
struct Commit : Codable {
    struct Author : Codable { /* … */ }
    let url: URL
    let message: String
    let author: Author
    let commentCount: Int

    private enum CodingKeys : String, CodingKey { /* … */ }

    public init(from decoder: Decoder) throws { /* … */ }
    public func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(url, forKey: .url)
        try container.encode(message, forKey: .message)
        try container.encode(author, forKey: .author)
        try container.encode(commentCount, forKey: .commentCount)
    }
}
struct Commit: Codable {
    struct Author: Codable { /* … */ }
    let url: URL
    let message: String
    let author: Author
    let commentCount: Int

    private enum CodingKeys: String, CodingKey { /* … */ }

    public init(from decoder: Decoder) throws { /* … */ }

    public func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(url, forKey: .url)
        try container.encode(message, forKey: .message)
        try container.encode(author, forKey: .author)
        try container.encode(commentCount, forKey: .commentCount)
    }
}
struct Point2D : Encodable {
    var x: Double
    var y: Double
struct Point2D : Encodable {
    var x: Double
    var y: Double
struct Point2D : Encodable {
    var x: Double
    var y: Double

    public func encode(to encoder: Encoder) throws {
        var container = encoder.unkeyedContainer()
        try container.encode(x)
        try container.encode(y)
    }
}
struct Point2D : Encodable {
    var x: Double
    var y: Double

    public func encode(to encoder: Encoder) throws {
        var container = encoder.unkeyedContainer()
        try container.encode(x)
        try container.encode(y)
    }
}
struct Point2D: Encodable {
    var x: Double
    var y: Double

    public func encode(to encoder: Encoder) throws {
        var container = encoder.unkeyedContainer()
        try container.encode(x)
        try container.encode(y)
    }
}

// [ 1.5, 3.9 ]
struct Point2D : Encodable {
    var x: Double
    var y: Double

    public func encode(to encoder: Encoder) throws {
        var container = encoder.unkeyedContainer()
        try container.encode(x)
        try container.encode(y)
    }
}

// [ 1.5, 3.9 ]
Nested Containers

Lightweight encapsulation of additional values

{ 
  Key : Value
  Key : Value
  Key : Value
}
Nested Containers

Lightweight encapsulation of additional values

{ }

Key : Value
Key : Value
Key : Value
Nested Containers

Lightweight encapsulation of additional values

- Key : Value
- Key : [Value, Value, Value]
- Key : Value
Encoding a Class Hierarchy

Use nested container for superclass data

Encapsulates keys and values from superclass
class Animal : Decodable {
    var legCount: Int
    private enum CodingKeys: String, CodingKey { case legCount }
    required init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        legCount = try container.decode(Int.self, forKey: .legCount)
    }
}
class Animal: Decodable {
    var legCount: Int

    private enum CodingKeys: String, CodingKey { case legCount }

    required init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        legCount = try container.decode(Int.self, forKey: .legCount)
    }
}
class Animal : Decodable {
    var legCount: Int

    private enum CodingKeys: String, CodingKey { case legCount }

    required init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        legCount = try container.decode(Int.self, forKey: .legCount)
    }
}

class Dog : Animal {
    var bestFriend: Kid

    private enum CodingKeys : String, CodingKey { case bestFriend }

    required init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        bestFriend = try container.decode(Kid.self, forKey: .bestFriend)
        let superDecoder = try container.superDecoder()
        try super.init(from: superDecoder)
    }
}
class Animal: Decodable {
    var legCount: Int

    private enum CodingKeys: String, CodingKey { case legCount }

    required init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        legCount = try container.decode(Int.self, forKey: .legCount)
    }
}

class Dog: Animal {
    var bestFriend: Kid

    private enum CodingKeys: String, CodingKey { case bestFriend }

    required init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        bestFriend = try container.decode(Kid.self, forKey: .bestFriend)
        let superDecoder = try container.superDecoder()
        try super.init(from: superDecoder)
    }
}
class Animal: Decodable {
    var legCount: Int

    private enum CodingKeys: String, CodingKey { case legCount }

    required init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        legCount = try container.decode(Int.self, forKey: .legCount)
    }
}

class Dog: Animal {
    var bestFriend: Kid

    private enum CodingKeys: String, CodingKey { case bestFriend }

    required init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        bestFriend = try container.decode(Kid.self, forKey: .bestFriend)
        let superDecoder = try container.superDecoder()
        try super.init(from: superDecoder)
    }
}
class Animal: Decodable {
    var legCount: Int

    private enum CodingKeys: String, CodingKey { case legCount }

    required init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        legCount = try container.decode(Int.self, forKey: .legCount)
    }
}

class Dog: Animal {
    var bestFriend: Kid

    private enum CodingKeys: String, CodingKey { case bestFriend }

    required init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        bestFriend = try container.decode(Kid.self, forKey: .bestFriend)
        let superDecoder = try container.superDecoder()
        try super.init(from: superDecoder)
    }
}
Codable Philosophy

Error handling built-in

Encapsulate encoding details

Abstract format from types
Abstract Format from Types

Reuse one implementation of Encodable and Decodable
Abstract Format from Types

Reuse one implementation of `Encodable` and `Decodable`

Allow new formats without library changes
Abstract Format from Types

Reuse one implementation of Encodable and Decodable

Allow new formats without library changes

Formats have different fundamental types and conventions
Encoding Strategies

Encoder-specific customizations for certain types
Encoding Strategies

Encoder-specific customizations for certain types

JSON
Date
Encoding Strategies

Encoder-specific customizations for certain types

JSON

Date

"2017-06-07T18:00:40Z"
Encoding Strategies

Encoder-specific customizations for certain types

JSON

Date

1496858440.0729699
Encoding Strategies

Encoder-specific customizations for certain types

JSON
Date

1496858440072.97
Encoding Strategies

Encoder-specific customizations for certain types

JSON

Date

"Wednesday, June 7, 2017 at 11:00 AM"
Encoding Strategies

Encoder-specific customizations for certain types

JSON

Date

"Wednesday, June 7, 2017 at 11:00 AM"

Data
Encoding Strategies

Encoder-specific customizations for certain types

JSON

Date

"Wednesday, June 7, 2017 at 11:00 AM"

Data

"AAIABAA="
Encoding Strategies

Encoder-specific customizations for certain types

JSON

Date

"Wednesday, June 7, 2017 at 11:00 AM"

Data

[0,2,0,4,0]
Encoding Strategies

Encoder-specific customizations for certain types

**JSON**

- **Date**
  
  "Wednesday, June 7, 2017 at 11:00 AM"

- **Data**
  
  "🐑🐶🐑🐶🐑 "
Encoding Strategies

Encoder-specific customizations for certain types

JSON

Date

"Wednesday, June 7, 2017 at 11:00 AM"

Data

"🐑🐶🐑🐶🐑"
 Codable Foundation Types

CGFloat
AffineTransform
Calendar
CharacterSet
Data
Date
DateComponents
DateInterval
Decimal

IndexPath
IndexSet
Locale
Measurement
NSRange
PersonNameComponents
TimeZone
URL
UUID
Your Type
Your Type
Encodable

Your Type

Decodable
Your Type

Encoder

Decodable

Decoder
Your Type

Encodable

Encoder

Decoder

Decodable

Encoder

Decoder

UnkeyedEncodingContainer

KeyedEncodingContainer

SingleValueEncodingContainer

UnkeyedDecodingContainer

KeyedDecodingContainer

SingleValueDecodingContainer
Summary

New API and improved performance in Foundation

Strongly typed key paths for Swift

New Key-Value Observation API

New **Codable** protocols
More Information

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s New in Cocoa</td>
<td>Grand Ballroom B</td>
<td>Wednesday</td>
<td>9:00AM</td>
</tr>
<tr>
<td>Cocoa Development Tips</td>
<td>Grand Ballroom B</td>
<td>Friday</td>
<td>9:00AM</td>
</tr>
<tr>
<td>Efficient Interactions with Frameworks</td>
<td>Hall 2</td>
<td>Friday</td>
<td>1:50PM</td>
</tr>
<tr>
<td>Labs</td>
<td>Foundation Lab</td>
<td>Technology Lab C</td>
<td>Wed 1:00PM-2:10PM</td>
</tr>
</tbody>
</table>