Parsing JSON into Swift Structs

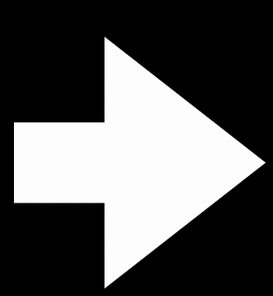
OR

Swift's type system could be great if it worked

Alex Hoppen
@alex_hoppen

Goal

```
[
 {
  "user_id": 67,
  "forename": "Max",
  "surname": "Mustermann"
 },
 {
  "user_id": 1782,
  "forename": "Michael",
  "surname": "Plagge"
 }
]
```

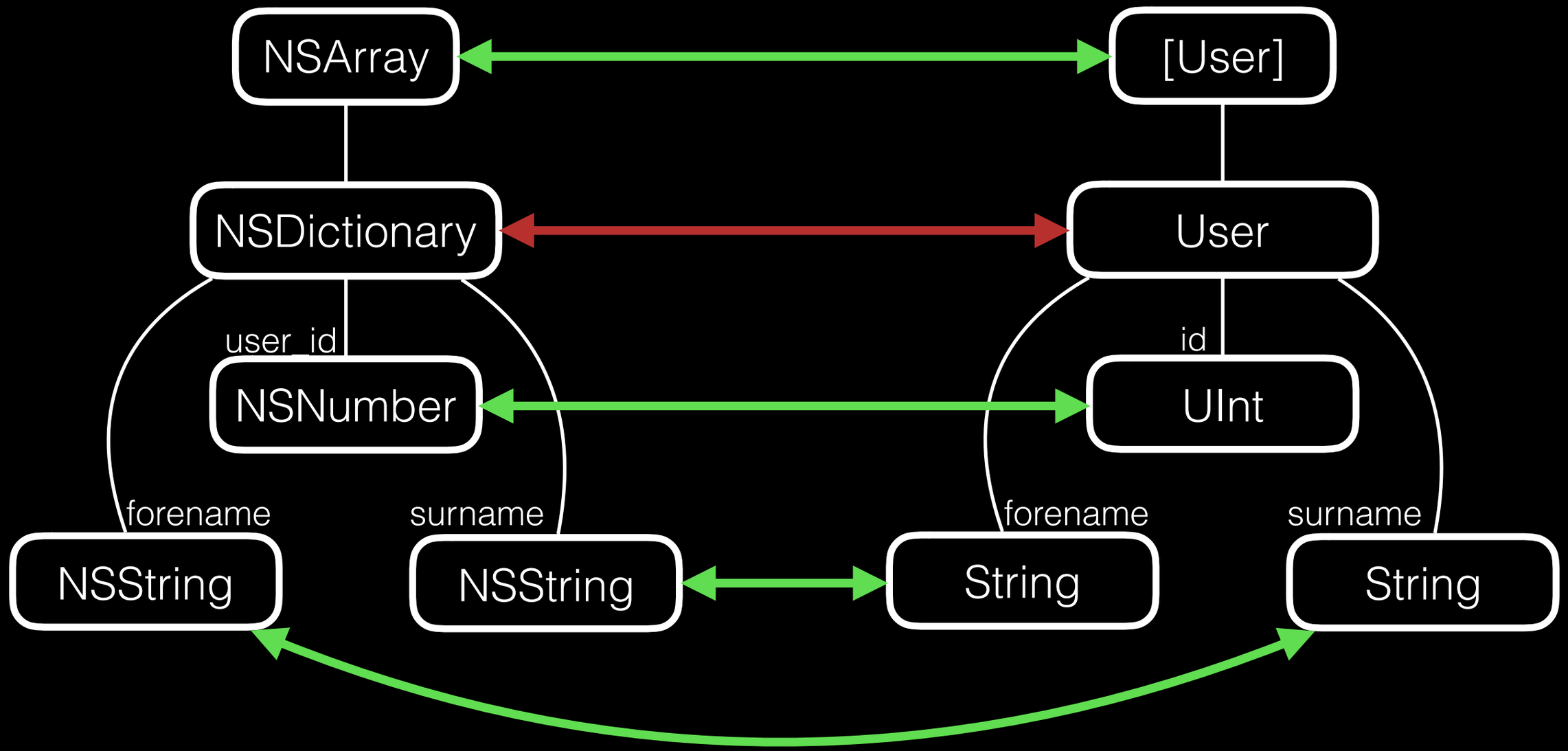


```
struct User {
  let id: UInt
  let forename: String
  let surname: String
}

let users: [Users] =
    try json.decode()
```

Without any code?

- Reflection
  - Read property names and values using Swift's MirrorType
    - Not able to write to instance variables 😥
- Key-Value-Coding
  - Would require ObjC-Classes -> loose struct semantics


NSArray
[User]
NSDictionary
User
user_id
id
NSNumber
UInt
forename
surname
forename
surname
NSString
NSString
String
String

JSONDecodable

- New protocol: `JSONDecodable`
  - All types that can be instantiated from JSON
    - Natively:
      - `String`, `Float`, `Int`, `UInt`, …
      - `[String: JSONDecodable]`, `[JSONDecodable]`
  - Add conformance to custom types
- One method: init yourself with this data or fail !

JSONDecoder

- Wrapper around parts of the JSON data
- Remembers path to the value for error handling
- Provides method `decode()` that decodes its value to the right type (or fails)
  - Heavily overloaded. Right one picked on return type (!)
  - Basically calls the right type's initialiser

```
let json = NSJSONSerialization.(...)
let users: [Users] = JSONDecoder(json).decode()

class JSONDecoder {
  public func decode<T: JSONDecodable>() throws -> [T] {
    return try decode().map { try $0.decode() as T }
  }

  public func decode<T: JSONDecodable>() throws -> T {
    try T.decode(self)
  }
}

struct User: JSONDecodable {
  let id: UInt; let forename: String; let surname: String

  init(_ decoder: JSONDecoder) throws {
    id = try decoder["user_id"].decode()
    forename = try decoder["forename"].decode()
    surname = try decoder["surname"].decode()
  }
}
```

```
extension UInt: JSONDecodable {
   public init(_ decoder: JSONDecoder) throws {
      if let value = decoder.value as? UInt {
         self.init(value)
      } else {
         throw ParsingError.IntegerConversionFailed(
                        atPath: decoder.pathIdentifier)
      }
   }
}
```

So what code was actually needed?

```
struct User: JSONDecodable {
  let id: UInt
  let forename: String
  let surname: String

  init(_ decoder: JSONDecoder) throws {
    id = try decoder["user_id"].decode()
    forename = try decoder["forename"].decode()
    surname = try decoder["surname"].decode()
  }
}

let data: NSData = ...
let parsedData: [User] = try JSONDecoder(data).decode() as [User]
```

Swift isn’t finished yet

- `[String: JSONDecodable]` is JSONDecodable but not `[NSData: NSProxy]`
- Not able to add protocol conformance via extension to generic types with constraints (rdar://23255436)

```
13
14  extension Array: JSONDecodable where Element: JSONEncodable {
15  }
16
```

Extension of type 'Array' with constraints cannot have an inheritance clause

- Let’s not talk about SourceKit crashes

Add custom atomic data types (e.g. NSDate from timestamp)

- All subclasses have to be guaranteed to conform to JSONDecodable
- Required initializer necessary
- Cannot be added using extensions
- Subclassing of NSDate is required

Add custom atomic data types (e.g. NSDate from timestamp)

```
class Date: NSDate, JSONDecodable {
   required convenience init(_ json: JSONDecoder) throws {
      if let timestamp = json.decode() as Double? {
         self.init(timeIntervalSince1970: timestamp)
      } else {
         throw Error
      }
   }
}
```

```
<unknown>:0: error: method 'decode' in non-final class 'Date' must return `Self` to conform to protocol
  'JSONDecodable'
<unknown>:0: error: method 'decode' in non-final class 'Date' must return `Self` to conform to protocol
  'JSONDecodable'
```

What’s the issue?

```
public static func decode(json: JSONDecoder) throws
   -> Self {
   return try self.init(json)
}
```

- self.init may return something else then its own type 😳

Add custom atomic data types (e.g. NSDate from timestamp)

```
final class Date: NSDate, JSONDecodable {
    required convenience init(_ json: JSONDecoder) throws {
        if let timestamp = json.decode() as Double? {
            self.init(timeIntervalSince1970: timestamp)
        } else {
            throw Error
        }
    }
}
```

```
<unknown>:0: error: method 'decode' in non-final class 'Date' must return `Self` to conform to protocol
  'JSONDecodable'
<unknown>:0: error: method 'decode' in non-final class 'Date' must return `Self` to conform to protocol
  'JSONDecodable'
```

rdar://23671426

Thank you

Source code available at:
https://github.com/ahoppen/JSONDecoder