

Codable

Giftbot

Codable

A type that can convert itself into and out of an external representation.

```
public typealias Codable = Decodable & Encodable
```

```
public protocol Encodable {
```

```
    /// Encodes this value into the given encoder.
```

```
    /// - Parameter encoder: The encoder to write data to.
```

```
    public func encode(to encoder: Encoder) throws
```

```
}
```

```
public protocol Decodable {
```

```
    /// Creates a new instance by decoding from the given decoder.
```

```
    /// - Parameter decoder: The decoder to read data from.
```

```
    public init(from decoder: Decoder) throws
```

```
}
```

Encoding & Decoding

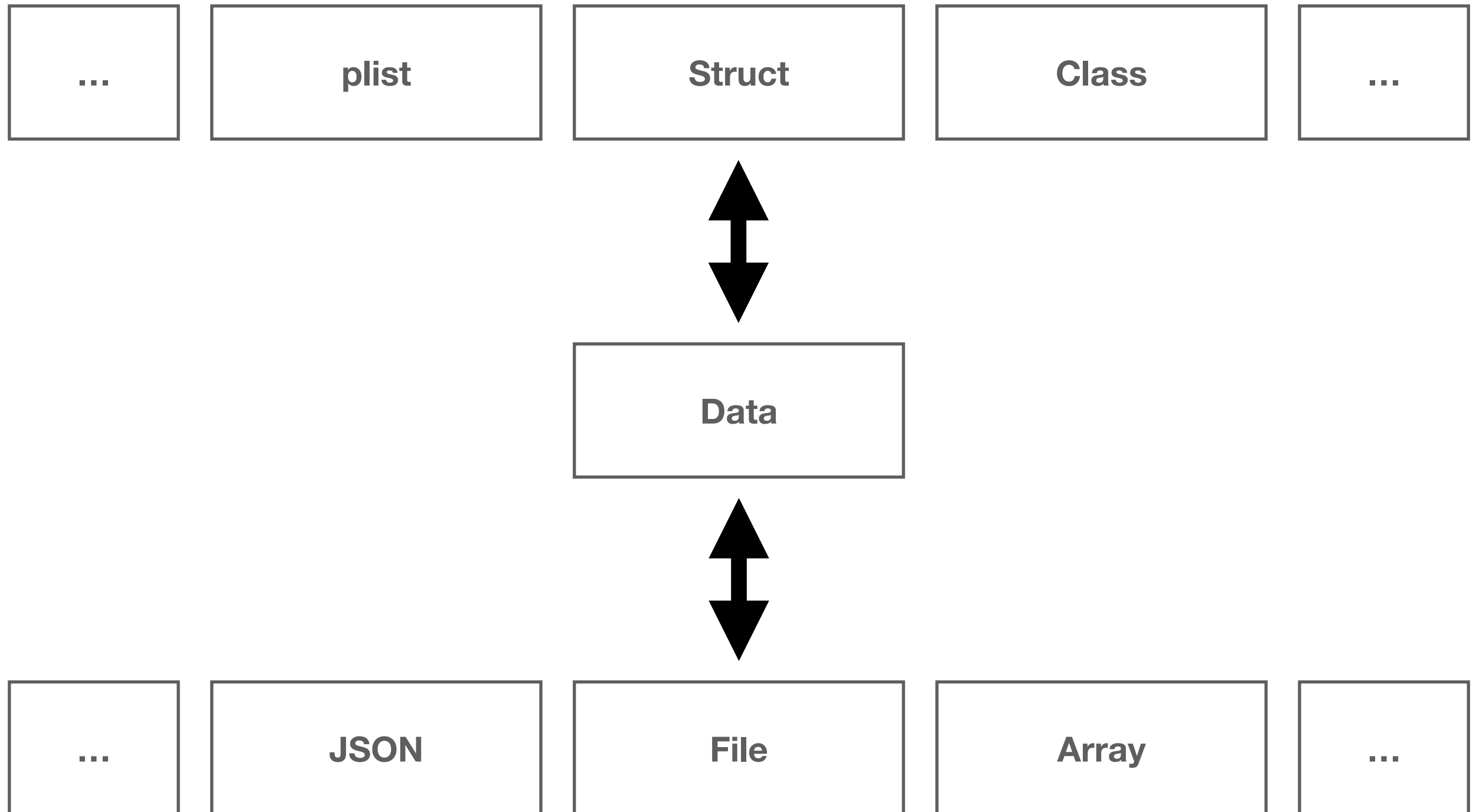
[Encoding, 부호화]

- 정보의 형태나 형식을 표준화, 보안, 처리 속도 향상, 저장 공간 절약 등을 위해서 목적에 맞는 다른 형태나 형식으로 변환하는 처리 혹은 그 처리 방식.
- **Encoder** : 인코딩을 수행하는 장치나 회로, 컴퓨터 소프트웨어, 알고리즘
- **A type that can encode values into a native format for external representation.**

[Decoding, 복호화]

- **Encoding(부호화)된 대상을 원래의 형태로 되돌리는 일**
- 예를 들어, 압축 파일을 다시 풀거나 암호화된 내용을 원래 내용으로 되돌리는 일
- **A type that can decode values from a native format into in-memory representations.**

Encode & Decode



Built-in Decoder / Encoder

/// `PropertyListEncoder` facilitates the encoding of `Encodable` values into property lists.

```
open class PropertyListEncoder { }
```

/// `PropertyListDecoder` facilitates the of property list values into semantic `Decodable` types.

```
open class PropertyListDecoder { }
```

/// `JSONEncoder` facilitates the encoding of `Encodable` values into JSON.

```
open class JSONEncoder { }
```

/// `JSONDecoder` facilitates the decoding of JSON into semantic `Decodable` types.

```
open class JSONDecoder { }
```

Use Encoder

```
struct MacBook: Codable {  
    let model: String  
    let modelYear: Int  
    let display: Int  
}  
  
let macBook = MacBook(  
    model: "MacBook Pro", modelYear: 2020, display: 16  
)  
  
let encoder = JSONEncoder()  
let encodedData = try! encoder.encode(macBook)  
print(type(of: encodedData))    // Data
```

Use Decoder

```
let jsonData = """
{
  "model": "MacBook Pro",
  "modelYear": 2020,
  "display": 16,
}
"""

jsonData.data(using: .utf8)!

let decoder = JSONDecoder()
let decodedData = try! decoder.decode(
    MacBook.self, from: jsonData
)
print(type(of: decodedData))    // MacBook
```

SwiftyJSON

SwiftyJSON / SwiftyJSON

Watch 638

Star 16,196

Fork 2,794

Code

Issues 48

Pull requests 13

Projects 1

Insights

The better way to deal with JSON data in Swift

swiftyjson

json

swift

689 commits

5 branches

23 releases

124 contributors

MIT

Branch: master

New pull request

Create new file

Upload files

Find file

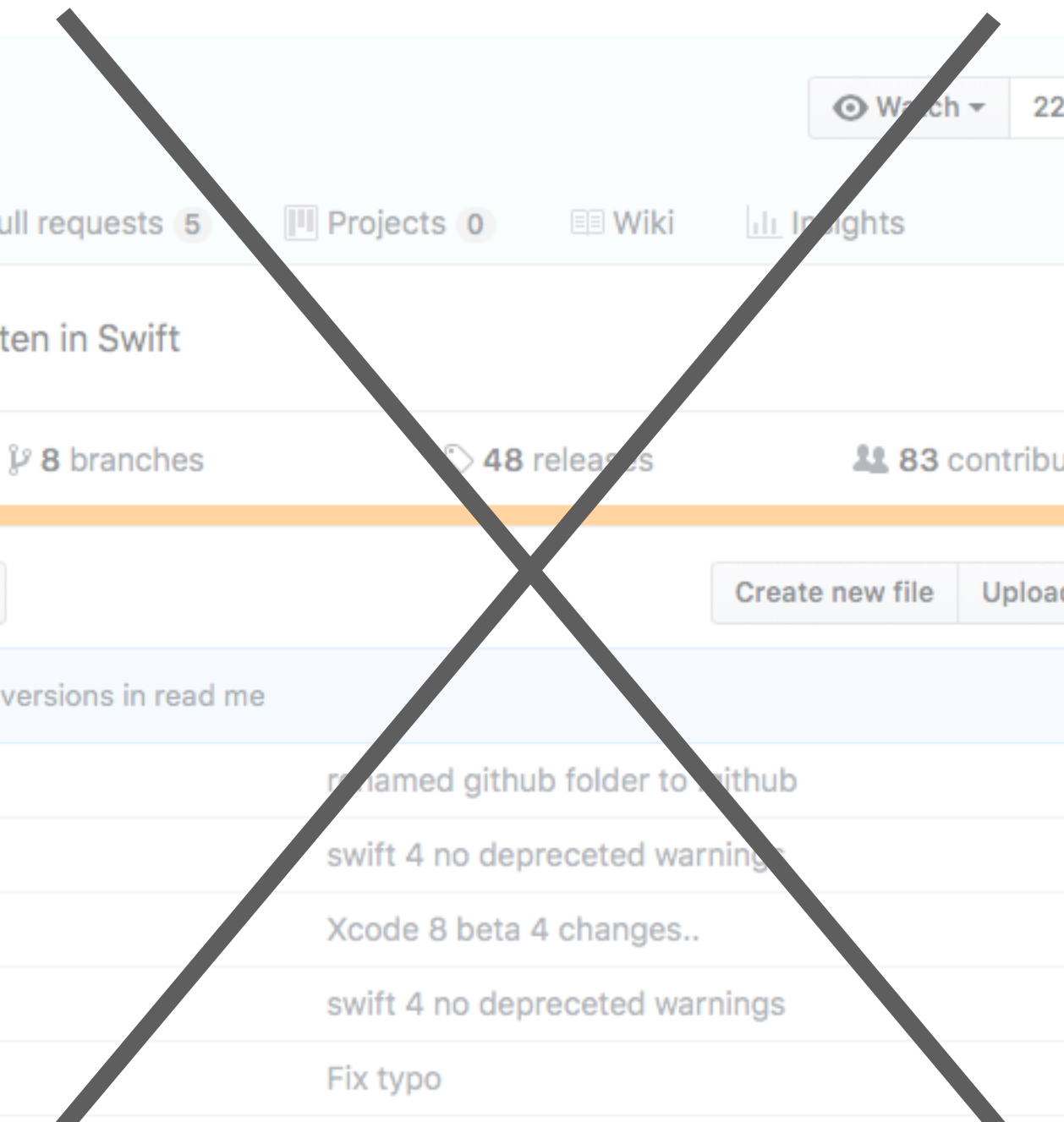
Clone or download

wongzigii Update README.md

Latest commit fb67763 on 8 Nov 2017

.github	Move files into .github directory	11 months ago
Example	Fix violations	3 months ago
Source	Fix SwiftLint violations	2 months ago
SwiftyJSON.xcodeproj	Fix broken path after renaming	4 months ago
SwiftyJSON.xcworkspace	Change example path for carthage. #276	3 years ago
Tests/SwiftyJSONTests	Fix SwiftLint violations	2 months ago

ObjectMapper



Hearst-DD / ObjectMapper

Watch 221 Star 6,740 Fork 737

Code Issues 43 Pull requests 5 Projects 0 Wiki Insights

Simple JSON Object mapping written in Swift

924 commits 8 branches 48 releases 83 contributors MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

tristanhimmelman Updated install versions in read me Latest commit 883a6f1 15 days ago

.github	renamed github folder to .github	11 months ago
ObjectMapper.xcodeproj	swift 4 no deprecated warnings	3 months ago
ObjectMapper.xcworkspace	Xcode 8 beta 4 changes..	2 years ago
Sources	swift 4 no deprecated warnings	3 months ago
Tests	Fix typo	3 months ago

Auto-synthesis example

```
struct User: Codable {  
    var userName: String  
    var score: Int  
}
```

Auto-synthesis by compiler

```
struct User: Codable { // Auto-synthesis example
    var userName: String
    var score: Int

    @derived private enum CodingKeys: String, CodingKey { // @derived = auto-synthesized
        case userName
        case score
    }

    @derived init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        userName = try container.decode(String.self, forKey: .userName)
        score = try container.decode(Int.self, forKey: .score)
    }

    @derived func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(userName, forKey: .userName)
        try container.encode(score, forKey: .score)
    }
}
```

Manual Implementation

```
struct User: Codable { // Manual implementation example
    var userName: String // Let's say, JSON has "user_name" key
    var score: Int        // Let's limit to 0...100

    private enum CodingKeys: String, CodingKey { // manual implementation
        case userName = "user_name" // rename key
        case score
    }

    init(from decoder: Decoder) throws { // manual implementation
        let container = try decoder.container(keyedBy: CodingKeys.self)
        score = try container.decode(Int.self, forKey: .score)
        guard (0...100).contains(score) else { // add validation
            throw DecodingError.dataCorrupted(
                codingPath: container.codingPath + [CodingKeys.score],
                debugDescription: "score is not in range 0...100"
            )
        }
        userName = try container.decode(String.self, forKey: .userName)
    }
}
```

Basic

```
struct Dog: Decodable {  
    let age: Int  
    let name: String  
}
```

Basic

```
let jsonData = """  
{  
  "age": 3,  
  "name": "Tory"  
}  
"""  
jsonData.data(using: .utf8)!
```

```
let dog = try? JSONDecoder().decode(Dog.self, from: jsonData)  
print(dog)
```

Decode Manually

```
struct Dog: Decodable {
    let age: Int
    let name: String

    private enum CodingKeys: String, CodingKey {
        case age
        case name
    }

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

Array

```
let jsonData = """  
[  
  {  
    "age": 3,  
    "name": "Tory"  
  },  
  {  
    "age": 3,  
    "name": "Tory"  
  }  
]  
""".data(using: .utf8)!
```

```
let dogs = try! JSONDecoder().decode([Dog].self, from: jsonData)  
print(dogs)
```


Dictionary

```
let jsonData = """
{
  "first": {
    "age": 3,
    "name": "Tory"
  },
  "second": {
    "age": 3,
    "name": "Tory"
  }
}
""".data(using: .utf8)!
```

```
let decoder = JSONDecoder()
let dogs = try! decoder.decode([String: Dog].self, from: jsonData)
print(dogs)
```

Dictionary

```
let jsonData = """
{
  "latitude": 30.0,
  "longitude": 40.0,
  "additionalInfo": {
    "elevation": 50.0,
  }
}
""".data(using: .utf8)!

let decoder = JSONDecoder()
let coordinate = try! decoder.decode(Coordinate.self, from: jsonData)
print(coordinate)
```

Nested Keys

```
struct Coordinate {  
    var latitude: Double  
    var longitude: Double  
    var elevation: Double  
  
    enum CodingKeys: String, CodingKey {  
        case latitude  
        case longitude  
        case additionalInfo  
    }  
    enum AdditionalInfoKeys: String, CodingKey {  
        case elevation  
    }  
}
```

Nested Keys

```
extension Coordinate: Decodable {
    init(from decoder: Decoder) throws {
        let values = try decoder.container(keyedBy: CodingKeys.self)
        latitude = try values.decode(Double.self, forKey: .latitude)
        longitude = try values.decode(Double.self, forKey: .longitude)

        let additionalInfo = try values.nestedContainer(
            keyedBy: AdditionalInfoKeys.self, forKey: .additionalInfo
        )
        elevation = try additionalInfo.decode(
            Double.self, forKey: .elevation
        )
    }
}
```

Container Protocols

KeyedContainer - 딕셔너리 타입의 데이터에 사용

UnkeyedContainer - 배열 타입의 데이터에 사용

SingleValueContainer - 단일 값을 가진 데이터에 사용

EncodingError

/// An error that occurs during the encoding of a value.

```
public enum EncodingError : Error {  
    /// 주어진 값으로 인코딩을 하지 못할 때  
    case invalidValue(Any, EncodingError.Context)  
}
```

DecodingError

/// An error that occurs during the decoding of a value.

```
public enum DecodingError : Error {  
    /// 프로퍼티 타입 미스매치  
    case typeMismatch(Any.Type, DecodingError.Context)  
    /// 디코딩할 데이터의 키에 해당하는 Value 가 없을 경우  
    case valueNotFound(Any.Type, DecodingError.Context)]  
    /// 디코딩할 데이터에 지정한 키가 없는 경우  
    case keyNotFound(CodingKey, DecodingError.Context)  
    /// 데이터가 망가졌을 경우  
    case dataCorrupted(DecodingError.Context)  
}
```