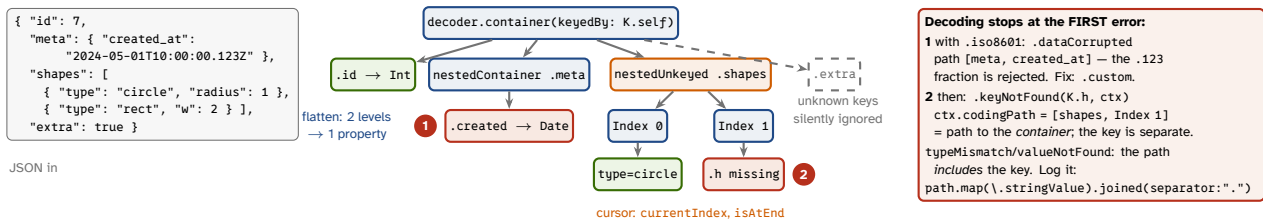


In one line: Codable = Encodable & Decodable. The compiler *synthesises* CodingKeys + init(from:) + encode(to:) from the stored properties; you write them by hand when the JSON's **shape** differs from the type's. A Decoder hands out **containers** (keyed · unkeyed · single-value) and every failure is a DecodingError whose codingPath says *where*.



How it works — the rules

- Synthesis:** every *stored* property Codable (computed ignored). In an **extension** only in the **same file**; a non-final **class** can't get Decodable from one (init(from:) must be required).
- CodingKeys is all-or-nothing:** list a property or give it a default (then never decoded). **Synthesised decoding ignores defaults:** var n = 0 still throws keyNotFound; only optionals tolerate a missing key.
- Enums** (5.5): raw-value = single value; associated values = keyed by case: circle(radius:) → {"circle":{"radius":1}} (unlabeled "_0") — rarely the API's shape.
- Containers:** keyed · unkeyed (a cursor) · single-value; nestedContainer(keyedBy: forKey:) / nestedUnkeyed-Container walk *into* JSON with no nested type. Dynamic keys: struct AnyKey: CodingKey + c.allKeys.
- Strategies:** date* (default .deferredToDate = seconds since **2001**; .iso8601, .secondsSince1970, .millisecondsSince1970, .formatted, .custom) · data* (default .base64) · key* · nonConformingFloat* (default .throw on nan).
- decoder.userInfo:** inject a Core Data context / schema version.

JSON has...	decode	decodeIfPresent	synthesised T?
key absent	keyNotFound	nil	nil
null	valueNotFound	nil	nil
wrong type	typeMismatch	throws too	throws

Example — flatten, default, discriminator

```
enum Shape: Decodable { // "type" picks the case
    case circle(r: Double), rect(w: Double, h: Double)
    enum K: String, CodingKey { case type, radius, w, h }
    init(from d: Decoder) throws {
        let c = try d.container(keyedBy: K.self)
        func num(_ k: K) throws -> Double {
            try c.decode(Double.self, forKey: k)
        }
        switch try c.decode(String.self, forKey: .type) {
            case "circle": self = .circle(r: try num(.radius))
            case "rect": self = .rect(w: try num(.w), h: try num(.h))
            case let t: throw DecodingError.dataCorruptedError(
                forKey: .type, in: c, debugDescription: "type \ \(t)" )
        }
    }
}

// in Post.init(from:) -- flatten + default for a missing key
let m = try c.nestedContainer(keyedBy: M.self, forKey: .meta)
created = try m.decode(Date.self, forKey: .created)
tags = try c.decodeIfPresent([String].self, forKey: .tags) ?? []
```

Example — dates with and without fractions

```
let frac = ISO8601DateFormatter()
frac.formatOptions = [.withInternetDateTime,
    .withFractionalSeconds]
let plain = ISO8601DateFormatter() // rejects ".123"
decoder.dateDecodingStrategy = .custom { d in
    let c = try d.singleValueContainer()
    let s = try c.decode(String.self)
    if let x = frac.date(from: s) { return x }
    if let x = plain.date(from: s) { return x }
    throw DecodingError.dataCorruptedError(in: c,
        debugDescription: "bad date \ \(s)") }
```

Interview traps

- .iso8601 rejects fractional seconds**, and a formatter *with* .withFractionalSeconds rejects strings *without* them — try both. .formatted(DateFormatter) needs locale = en_US_POSIX + an explicit timeZone, or a 12-hour/Buddhist-calendar device breaks it.
- Snake case + acronyms:** user_id is converted to userId *before* matching, so userID never matches → case userID = "userId".
- One bad element fails the whole array.** Lossy decoding: wrap each element in a box whose init does try? — never a bare try? in an unkeyed loop: a failed decode does **not** advance the cursor → infinite loop.
- Dictionary keys:** only String/Int keys encode as a JSON object; [MyEnum: V] (even String-backed) becomes a flat array ["a",1,"b",2] unless the key adopts CodingKeyRepresentable (5.6).
- [any Animal] cannot be decoded** — no static init(from:) on an existential. Decode an enum, then map.
- Subclass of a Codable class:** inherits init(from:)/encode(to:), so new stored properties (with defaults) are **silently skipped**. Override both; call super.init(from: c.superDecoder()) or super.encode(to:).
- Persisted Codable = a schema.** A new non-optional field breaks decoding of old data: make new fields optional or decodeIfPresent + default; version it.

Remember

"Shape differs → containers; kind varies → discriminator enum; failure → read the path."

Likely questions

- Absent vs null? — decodeIfPresent gives nil for both; decode throws keyNotFound / valueNotFound.
- Polymorphic payload? — read "type", switch into an enum.
- Flatten meta.created_at? — nestedContainer(keyedBy: forKey:).
- Pass a context into decoding? — decoder.userInfo.