Dokumenty, synchronizace dat (CloudKit)

Programování zařízení Apple (IZA)

Martin Hrubý, FIT VUT, 2022/23

Úvod

Další způsoby uložení dat (kromě CD):

- User Defaults. Synchronizované UD.
- Soubory v sandboxu.
- Dokumenty. iCloud synchronizace. *Ubiquitous*.
- *CloudKit* objektový kontejner.

UserDefaults

- Singleton v aplikaci `UserDefaults.standard`
- Kódovaný dictionary klíč-hodnota.
  - klíč - String
  - hodnota - String, Int, ..., []
- uložení: `UserDefaults.standard.synchronize()`
- ukládáno do aplikačního sandbox přístupného uživateli.
- Factory defaults.
- iCloud varianta: KeyValueStorage

V macOS aplikacích (prompt-type) UD slouží pro analýzu parametrů příkazové řádky.

UserDefaults: demo

```
// UD je singleton (default/standard/shared)
let UD = UserDefaults.standard

// Zavedme si konstanty pro jednotlive attributy
enum Keys {
    //
    static let cloudIsON = "cloudIsON"
}

// zapis
UD.set(true, forKey: Keys.cloudIsON)

// ulozeni, perzistence, od iOS-12 obsolete UD.synchronize()
// cteni
print(UD.bool(forKey: Keys.cloudIsON))
```

UserDefaults: observable demo

```
// UD, Combine: Observable
class Settings : ObservableObject {
    //
    static let UD = UserDefaults.standard
    static let standard = Settings()

    // pocatecni hodnota, zjistuje se pred kompletaci "self"
    @Published var cloudIsON = UD.bool(forKey: Keys.cloudIsON) {
        //
        didSet {
            // hodnota se propaguje do UD
            Settings.UD.set(cloudIsON, forKey: Keys.cloudIsON)
        }
    }
}
```

Pozor, není zrcadlením fyzického UD.

View

```
// UD, Combine, SwiftUI
struct ContentView: View {
    // View bude dostavat zpravy o zmenach objektu
    @ObservedObject var settings = Settings.standard

    //
    var body: some View {
        //
        List {
            // Binding<Bool>
            Toggle(isOn: $settings.cloudIsON) {
                Text("Jedeme klaud?")
            }
        }
    }
}
```

PropertyWrapper pro UD

```
//
class Settings: ObservableObject {
    //
    @AppStorage("Example") var example: Bool = false {
        //
        willSet {
            // Call objectWillChange manually since
            // @AppStorage is not published
            objectWillChange.send()
        }
    }
}
```

PropertyWrapper pro UD

```
//
struct Settings: View {
    //
    @AppStorage("Example") var example: Bool = false

    //
    var body: some View {
        //
        Form {
            //
            Section(header: Text("cosi")) {
                //
                Toggle("Example", isOn: $example)
            }
        }
    }
}
```

Malá data aplikace v UD

Pokud má aplikace malou datovou potřebu, je UD jako uložiště adekvátní řešení (kromě bezpečnostně citlivých dat).

```
class SmallData: ObservableObject {
    //
    @Published var allMyData: MyDocModel

    //
    init() {
        guard
            let _sd = UserDefaults.standard.object(forKey: "small") as? Data,
            let _deco = try? JSONDecoder().decode(MyDocModel.self, from: _sd)
        else {
            //
            allMyData = MyDocModel(); return
        }

        //
        allMyData = _deco
    }
}
```

Settings v aplikaci

Lze dělat vlastním formulářem a ukládat do UD.

iOS poskytuje navíc:

- v XCode pořídit konfigurační soubor .plist
- pak se formulář generuje v systémové aplikaci Settings
- aplikace tam má příslušnou sekci
- a nastavení se ukládá do UD aplikace

K tomu patří i další vrstvy nastavení podmíněné souhlasem uživatele (notifikace, běh na pozadí, přístupy, ...).

Soubory v sandbox aplikace

- Sandbox aplikace - "home" adresář aplikace (iOS/macOS).
- Bundle aplikace - distribuční adresář aplikace (XCode):
  - CoreData model.
  - NIB/XIB soubory (UIKit/Storyboard). Storyboard.
  - Multi-mediální soubory.
  - další vložené do projektu v XCode.

Připomenutí: filosofie sdružování aplikačně specifických dat s aplikacemi (obrázky, texty, ...). Žádný home adresář.

- iCloud Drive

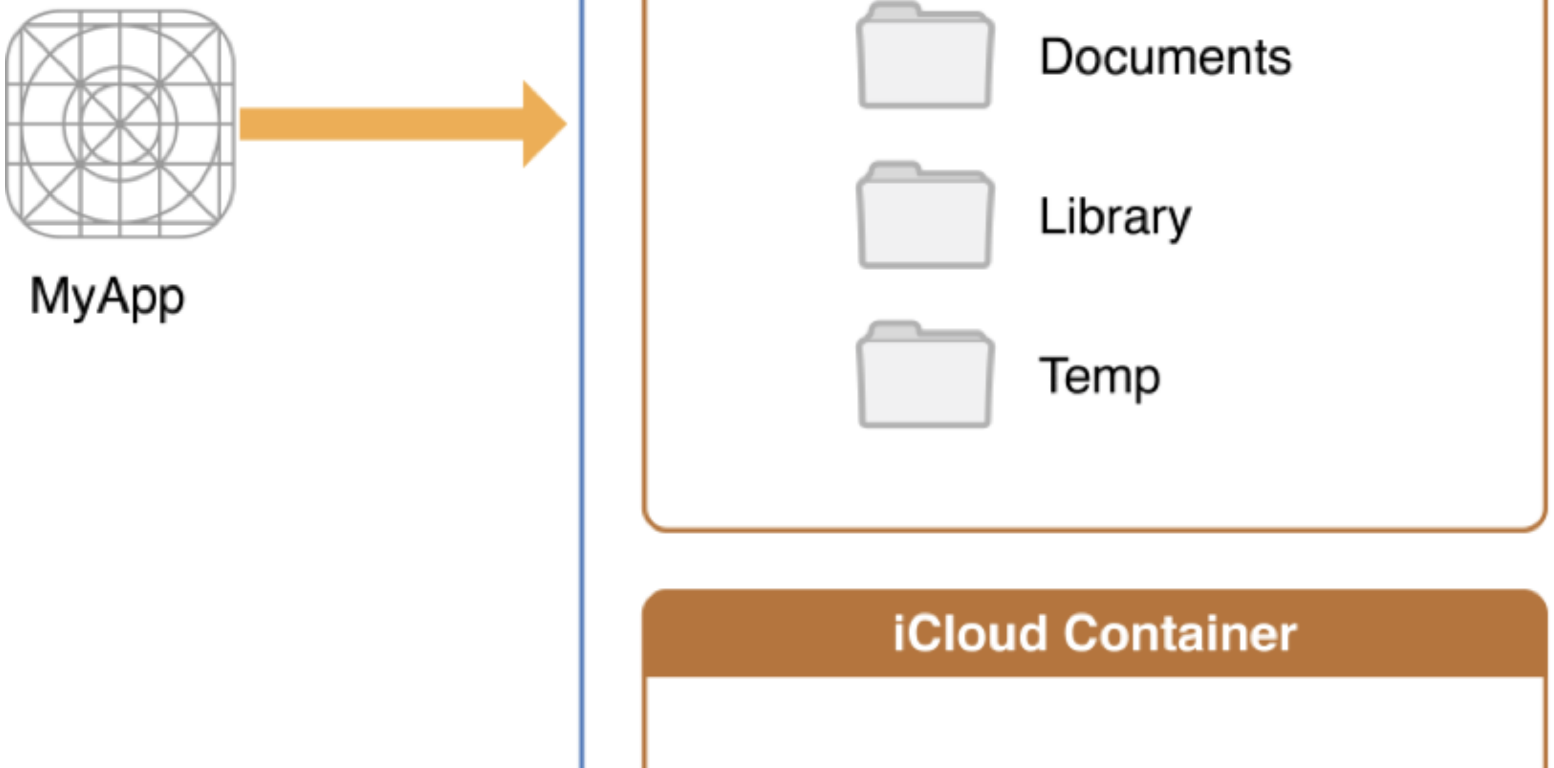
Resource URL/Path.

- Path - cesta k souboru.
- URL - vzniká z Path. URL syntaxe.

Knihovní funkce pracují buď s Path nebo URL.

```
//
let p = Bundle.main.path(forResource: "mujObrazek", ofType: "jpg")

// je optional
Image("mujObrazek", bundle: Bundle.main)
```


Bundle Container
MyApp.app
Data Container
Documents
Library
Temp
iCloud Container
MyApp

Sandbox aplikace

Obecně je sandbox privátní uložiště.

- Bundle.
- Documents - soubory vytvářené uživatelem. Tyto soubory uživatel smí spravovat prostřednictvím iTunes.
- Library. Systém podadresářů. Interní data aplikace mimo dosah uživatele.
- tmp - dočasné soubory. Systém je smí likvidovat.

Prostý přístup na diskový soubor

I/O jsou throws operace.

```
//
let mujString = "Ahoj, prosim ulozit."

//
do {
    try mujString.write(toFile: "/Users/hrubym/tmp/text",
                        atomically: true /* false */,
                        encoding: .utf8)
} catch {
    //
    print("Zrejme error")
}

//
print(try? String(contentsOfFile: "/Users/hrubym/tmp/text"))
```

String versus Data

`Data` je třída pro uložení obecně binárních dat.

Převoditelnost String <-> Data.

```
// Pozn.: predpokladejme do-catch
// konvertuju text do "Data"
if let mojeData = "tady je obsah".data(using: .utf8) {
    // 1) path:
    // mojeData.write(toFile: "dfile", atomically: true)
    // -----
    // 2) URL
    let url = URL(fileURLWithPath: "dfile")

    // zapis, options...
    try mojeData.write(to: url, options: [])
}

// precti obsah souboru na URL
let precti = try Data(contentsOf: URL(fileURLWithPath: "..."))

// konverze Data->String
let stringVerze = String(data: precti, encoding: .utf8)
```

FileManager

Klasika z Foundation. Přenositelná funkcionalita správy souborů a adresářů.

`FileManager.default`

- konstrukce cesty/URL do sandboxu.
- fileExists:, remove: ...,

Paths - macOS

```
let _url = URL(fileURLWithPath: "/Users/hrubym/tmp/soubor.txt")

//
print(_url) // -> file:///Users/hrubym/tmp/soubor.txt
print(_url.path) // -> /Users/hrubym/tmp/soubor.txt
print(_url.pathComponents)
// -> ["/", "Users", "hrubym", "tmp", "soubor.txt"]

print(_url.lastPathComponent) // -> soubor.txt
print(_url.pathExtension) // -> txt
```

Paths - macOS

```
let _fm = FileManager.default
var _urla = _fm.homeDirectoryForCurrentUser

// mutating
_urla.appendPathComponent("Documents", isDirectory: true)
_urla.appendPathComponent("mujDokument.xy")

// -> file:///Users/hrubym/Documents/mujDokument.xy
print(_urla.absoluteURL)

// funkcionalni
let _urlb = _fm.homeDirectoryForCurrentUser
    .appendingPathComponent("Documents", isDirectory: true)
    .appendingPathComponent("mujDokument.xy1")
```

Paths - iOS/Sandboxing

```
// chci URL pro dokumentovy adresar uzivatele
// vraci pole URL, -> .first -> Array.Element?
// hodnota je URL
if let docDir = FileManager.default.urls(for: .documentDirectory,
                                         in: .userDomainMask).first
{
    //
    print(docDir)
    // -> file:///Users/hrubym/Documents/

    //
    let fnURL = docDir.appendingPathComponent("soubor.txt")
    // -> file:///Users/hrubym/Documents/soubor.txt

    //
    print(fnURL)
} /* else ???? */
```

Kódování dat

Foundation.

- Vstup: objektová paměť.
- Výstup: binární archiv ("tar"). Instance `Data`.

Uložení (kam pak s tím):

- Do souboru/dokumentu.
- Do binary/BLOB atributu CD objektu.
- Síťová operace (JSON).

Enkodér. Dekodér.

Protokol NSCoding (Foundation)

```
class City: NSObject, NSCoding
{
    var name: String?
    var id: Int?

    required init?(coder aDecoder: NSCoder)
    {
        self.name = aDecoder.decodeObject(forKey: "name") as? String
        self.id = aDecoder.decodeObject(forKey: "id") as? Int
    }

    func encode(with aCoder: NSCoder)
    {
        aCoder.encode(self.name, forKey: "name")
        aCoder.encode(self.id, forKey: "id")
    }
}
```

required init.

NSCoding: demo

```
// Kodovani
let _mdata = [City("Brno", 1), City("Ostrava", 2)]

// tridni metoda, koduje "_mdata" a vraci Data
let _encoded : Data? =
    NSKeyedArchiver.archivedData(withRootObject: _mdata)

// Dekodovani. Z hlediska prekladace se ted NEVI, CO je vysledny typ
if let _decoded2 =
    NSKeyedUnarchiver.unarchiveObject(with: _encoded!) as? [City]
{
    //
    print(_decoded2)
}
```

Swift: protokol Codable

```
// Codable spojuje dva protokoly
typealias Codable = Decodable & Encodable

// Datovy objet implementujici protokol "Codable"
// vsimnete si: NIC VIC se nechce...
struct MFile : Codable {
    //
    let name: String
    let size: Int
    let owner: String = "ja"
    let created: Date
}

//
struct MFolder : Codable {
    //
    let name: String
    let files: [MFile]
}
```

Codable: demo

```
//
let _f1 = MFile(name: "soubor", size: 1000, created: Date())
let _d1 = MFolder(name: "dic1", files: [_f1])

// try? -- pripadne exception odchyti a vraci nil
// vysledny typ vyrazu je Data?
if let _encoded = try? JSONEncoder().encode(_d1) {
    // vysledek konstrukce je String?
    if let _stringVersion = String(data: _encoded, encoding: .utf8) {
        //
        print("Encoded: \(_stringVersion)")
    }
}
```

Encoded:

```
{"name":"dic1","files":[{"size":1000,"created":543077670.01387095,"owner":"ja","name":"soubor"}]}
```

.prettyPrinted

```
// JSON Encoder
let _enco = JSONEncoder()

// :)
_enco.outputFormatting = .prettyPrinted
```

```
{
  "name" : "dic1",
  "files" : [
    {
      "size" : 1000,
      "created" : 543078066.91694605,
      "owner" : "ja",
      "name" : "soubor"
    }
  ]
}
```

JSON: dekódování

```
//
let _enco = try? JSONEncoder().encode(_d1)

//
// open func decode<T>(_ type: T.Type, from data: Data) throws
//    -> T where T : Decodable

//
if let _back = try? JSONDecoder().decode(MFolder.self, from: _enco!)
{
    // _back je typu MFolder
    print(_back)
}
```

Univerzálně JSON-dekódovatlný protokol

```
protocol JSONable : Decodable {
    //
    // init?(fromJSON: String)
}

//
extension JSONable {
    //
    init?(fromJSON: String) {
        //
        guard let _dt = fromJSON.data(using: .utf8)
            else { return nil }

        //
        do { self = try JSONDecoder().decode(Self.self, from: _dt) }
        catch {
            //
            return nil
        }
    }
}
```

`init()` je něco mezi statickou a instanční metodou.

Vytváří `self`.

... použití

```
//
struct Zaznam : JSONable {
    //
    let jmeno: String
    let vek: Int
}

let _str = "{ \"jmeno\" : \"John\", \"vek\": 123456 }"
let _zaz = Zaznam(fromJSON: _str)

//
print(_zaz)
```

PropertyListEncoder

`.plist` - typické pro metadata aplikace v XCode.

```
//
let _f1 = MFile(name: "soubor", size: 1000, created: Date())
let _d1 = MFolder(name: "dic1", files: [_f1])

//
let _pEnco = PropertyListEncoder()

//
_pEnco.outputFormat = .xml
_pEnco.outputFormat = .binary
_pEnco.outputFormat = .openStep // ??

//
if let _propl = try? _pEnco.encode(_f1) {
    //
    print(String(data: _propl, encoding: .utf8)!)
}
```

UIDocument - document-based app

- aplikace používající koncept dokumentů,
- document-based app (tj specifický druh aplikace).

UIDocument (NSDocument)

Dokument je soubor/svazek (filewrapper) v sandboxu, který je spravován operačním systémem.

- Metadata. Undo-management.
- Životní cyklus.
- Autosave.
- iCloud synchronizace - ubiquitous (všudypřítomný).

Document-Based Application - aplikace je určena pro správu konkrétního (zaregistrovaného) typu dokumentu.

Motivační úvod

Je moje aplikace spíš na CoreData nebo na Documents?

CoreData (obecně SQL/relační DB):

- potřebuju vyhledávat (select, where).
- celý obsah DB je pamětí aplikace (ne po částech).
- správa elementárních záznamů (insert, delete).
- provázanost objektů (relationships) napříč celou DB.

Problém: CoreData nešlo nikdy uspokojivě synchronizovat (zrcadlit). Částečně vyústilo v zavedení CloudKit.

Ekosystém zařízení uživatele.

Motivační úvod

Dokumentní aplikace:

- data uživatele jsou malé celky (dokumenty). Vzájemně nezávislé.
- dokument otevřu, edituju, zavřu.
- potřebuju datovou strukturu, pro kterou nechci tvořit DB schema. Např velká rozmanitost obsahu, přílohy, apod.
- relationships mezi elementy jsou pouze v rámci dokumentu.

Synchronizace: je poměrně funkční a předvídatelná (až na případné konflikty v aktualizaci dat).

Obecné schema DB+synchronizace

Obecně se očekává:

- aplikace má lokální uložiště dat,
- a to je schopna synchronizovat na ostatní zařízení.

Přinejmenším je aplikace schopna

- archivovat data pro případ, že je ze zařízení smazána
- … a následně znovu instalována. iCloud dokumenty přežijí.

Všeobecně archivace/zálohování uživatelských aplikací (kompletní instalace včetně sandboxu).

Ideál: synchronizované CoreData

Vývojář si "nakreslí" komplikované DB a chce to synchronizovat.

Co je předmětem uložení a následně synchronizace:

- datový záznam (Entita CD, objekt CD)
- záznam o relationship (1:1, 1:N, M:N).

Synchronizace:

- prostých záznamů Entit - řešitelná (CKRecord)
- relationships (1:1, 1:N, N:M) - může být peklo.

Ideál: synchronizované CoreData

Obecná dobrá rada: pokud chcete svoje DB schema synchronizovat, pak to nepřehánějte s košatostí E-R relationships.

Nebo implementujte **datový hybrid**:

- Entity s uloženými atributy, kde v "binary" atributu je kódovaný detail záznamu, tj Dokument.
- Relationships implementujte tabulkami (Entitami) explicitně.
    - Tabulka Students, Courses, StCourses (vztah, explicitně klíče jako uložené atributy).

Relationships v CD slouží (snad) jenom pro automatizaci delete operace.

CloudKit - průběžná zmínka

Objektový kontejner (podobnost s CD)

- Entity. `CKRecord` (transportní záznam).
- Uložení relationships CKReference.
- Uložení příloh `CKAsset` (rozsáhlé soubory).

S CloudKit lze úspěšně implementovat synchronizaci dat a hybridní model

- záznamy s binary kódem pro detailní data.

Systémová aplikace *Notes* v iOS/macOS.

Document

- dokumenty z UIKit - UIDocument (příp NSDocument).
- SwiftUI document based app.

UIDocument konvenční

V aplikaci máme:

- struktury tvořící datový model dokumentu,
- třídu odvozenou od UIDocument,
- ViewModel,
- View.

Datový model dokumentu

```
// část dokumentu
struct MyDocModelItem: Codable, Identifiable {
    //
    let id = UUID()
    //
    var created: Date = Date.now
    var note: String = ""
    //
    init(withNote: String) {
        //
        note = withNote
    }
}
// Model dokumentu
struct MyDocModel: Codable {
    //
    var title: String = ""
    var items: [MyDocModelItem] = []
}
// Kódování případných chyb ...
enum MyDocError: Error {
    //
    case cosiJeBlbe
}
```

Třída dokumentu

```
class MyDoc: UIDocument {
    // datový obsah dokumentu
    var model: MyDocModel = MyDocModel()
    // Save operace. Vytvoření archivu.
    override func contents(forType typeName: String) throws -> Any {
        //
        guard let _enco = try? JSONEncoder().encode(model)
        else { throw MyDocError.cosiJeBlbe }

        //
        return _enco
    }
    // Load operace. Rozbalení archivu.
    override func load(fromContents contents: Any,
                       ofType typeName: String?) throws
    {
        //
        guard
            let _code = contents as? Data,
            let _deco = try? JSONDecoder().decode(MyDocModel.self, from: _code)
        else { throw MyDocError.cosiJeBlbe }

        //
        self.model = _deco
    }
}
```

ViewModel dokumentu I.

```
class MyDocVM: ObservableObject {
    // VM
    @Published var title: String = ""
    @Published var items: [MyDocModelItem] = []

    // dokument - reference na otevreny dokument
    var _doc: MyDoc

    // instanciace VM pro dokument
    init(fileName: String) {
        // sestaveni URL
        let _fm = FileManager.default.urls(for: .documentDirectory,
                                           in: .userDomainMask)
        let _fm1 = _fm.first!.appendingPathComponent(fileName)
        // instanciace Doc
        _doc = MyDoc(fileURL: _fm1)
        // ASYNC
        _doc.open(completionHandler: { res in
            //
            if res {
                //
                self.title = self._doc.model.title
                self.items = self._doc.model.items
            }
        })
    }
}
```

ViewModel dokumentu

```
extension MyDocVM {
    //
    func save() {
        //
        _doc.model.items = items
        _doc.model.title = title

        // ASYNC
        _doc.save(to: _doc.fileURL, for: .forOverwriting) { res in
            //
            print("Save", res)
        }
    }
}
```

View

```
struct DocView: View {
    //
    @StateObject var doc: MyDocVM

    //
    init(fileName: String) {
        //
        _doc = StateObject(wrappedValue: MyDocVM(fileName: fileName))
    }

    //
    var body: some View {
        ...
    }
}
```

Takto můžeme budovat dokumentní infrastrukturu nad:

- fyzickými soubory ze sandboxu (následně pak ubiquitous)
- nebo nad BLOB (Data) atributy NSManageObject (CD).

Document FileWrapper

Strukturované dokumenty.

- Implementace: dokument je adresář se soubory pro jednotlivé části dokumentu (tělo, přílohy, multi-media).
- Vnitřní soubor: FileWrapper
- Document: FileWrapper typu dictionary FileWrapperů

Přínosy:

- binární data netřeba kódovat do JSON
- lze ukládat po částech (jednotlivé vnitřní FW)

FileWrapper pro binární data

```
// keyName - ve slovniku FW, fileWrappers [String:FileWrapper]
// preferredFilename: na disku
func wrapImage(ui: UIImage, fileName: String) -> FileWrapper? {
    //
    guard let _png = UIImagePNGRepresentation(ui)
    else { return nil }

    // FW bude mit charakter jednoho souboru (neni adresar)
    // _png je instance Data()
    let _fw = FileWrapper(regularFileWithContents: _png)

    //
    _fw.preferredFilename = fileName

    //
    return _fw
}
```

Zakódování FW

Celý dokument je svazek FileWrapperů, tj zanořených souborů.

OS pracuje s FW dokumentem jako s celkem (přesun, smazání,).

```
override func contents(forType typeName: String) throws -> Any {
    //
    let _fwrap = FileWrapper(directoryWithFileWrappers: [:])

    //
    if let _pic = pic,
        let _picFW = wrapImage(ui: _pic, fileName: "pic.png")
    {
        //
        _fwrap.addFileWrapper(_picFW)
    }

    //
    return _fwrap
}
```

Rozkódování FW

```
// typicky se odehrává ve vedlejším vlákně
override func load(fromContents contents: Any,
                   ofType typeName: String?) throws {
    //
    guard let _fwrap = contents as? FileWrapper else {
        //
        throw MujDocError.encError
    }

    //
    if let _picFW = _fwrap.fileWrappers?["pic.png"] {
        // synchronní operace (ovšem v GLOBAL-thread)
        if let _dt = _picFW.regularFileContents {
            //
            pic = UIImage(data: _dt)
        }
    }
}
```

Dokumenty v iCloud (Ubiquitous)

V sandboxu aplikace je umístění pro soubory

- uživatele (Library nebo Documents).
- pro iCloud synchronizaci.

Označení dokumentu na URL1 jako *ubiquitous*:

- vytvoření URL2 v ubiquitous adresáři sandboxu
- přesun dokumentu

Pokud má uživatel povolen iCloud-doc, pak FM zná URL ubiquitous adresáře.

```
FileManager.default.url(forUbiquityContainerIdentifier: nil)
```

Dokumenty v iCloud (Ubiquitous)

Aplikace musí (by měla) akceptovat proměnlivost prostředí:

- uživatel má povolen iCLoud-doc
- chce ukládat do iCloud-doc a synchronizovat
- Notifikační centrum - hlásí "změny situace"

```
do {
    try FileManager.default.setUbiquitous(true, itemAt: fileURL,
                                          destinationURL: _destURL)
} catch {
    // neco se pokazilo
    return false
}
```

Reverzní operace (z UBIQ adresáře do normálního adresáře).

Dynamika iCloud dokumentu

Když očekáváme aktualizace iCloud dokumentu z jiných zařízení uživatele.

`Document.documentState` — bitový vektor stavových informací:

- `.normal`, `.progresAvailable`, `.inConflict`, `.editingDisabled`
- Registrace k odběru notifikací
- Odchycení .progresAvailable
- Pak odchycení .normal
- Pak reload

SwiftUI - document Based app

Ustálená architektura aplikace. Programuje se pouze dokument a View pro editaci dokumentu.

```
@main
struct MojeDocAppApp: App {
    var body: some Scene {
        // Generuje aplikaci "prohlížeč souborů" + editace
        DocumentGroup(newDocument: MojeDocAppDocument()) { file in
            // View pro editaci souboru
            ContentView(document: file.$document)
        }
    }
}
```

Doctype

```
extension UTType {
    static var exampleText: UTType {
        UTType(importedAs: "com.example.plain-text")
    }
}
```

Document - uložení/načtení

```
struct MojeDocAppDocument: FileDocument {
    // obsah
    var text: String

    init(text: String = "Hello, world!") {
        self.text = text
    }

    // typ dokumentu
    static var readableContentTypes: [UTType] { [.exampleText] }

    init(configuration: ReadConfiguration) throws {
        guard let data = configuration.file.regularFileContents,
              let string = String(data: data, encoding: .utf8)
        else {
            throw CocoaError(.fileReadCorruptFile)
        }
        text = string
    }

    func fileWrapper(configuration: WriteConfiguration) throws -> FileWrapper {
        let data = text.data(using: .utf8)!
        return .init(regularFileWithContents: data)
    }
}
```

Doc - view

```
struct ContentView: View {
    //
    @Binding var document: MojeDocAppDocument

    //
    var body: some View {
        TextEditor(text: $document.text)
    }
}
```

CloudKit velmi stručně

Serverový objektový kontejner.

Terminologie:

- Aplikace - konkrétní iOS/macOS program.
- Uživatel aplikace - vlastnící N zařízení, kde je Aplikace instalována.
- CK Container Aplikace - DB kontejner pro DB objekty, tj každá CK Aplikace má svůj jeden CK kontejner.

CK kontejner se dělí na CK Database (+tzv Shared):

- Private - vidí pouze uživatel iOS/macOS zařízení
- Public - vidí všichni uživatelé dané aplikace

Vývoj CK aplikace

V XCode nebo na vývojářském webu:

- AppID ve vývojářském centru.
- ID pro CK container - je nezávislý na AppID.

Tj, více aplikací JEDNOHO vývojáře/týmu může sdílet CK Container (typické pro iOS/macOS porty aplikace).

CK Dashboard:

- definice DB schematu
- definice subscriptions

Databáze jednoho CK Container

Public:

- všichni můžou všechno, resp. to upravuje daná aplikace.
- disková kvóta registrované aplikace
- subscriptions: trigger detekující konkrétní událost nad DB, push-notifikace do zařízení uživatele.

Private:

- je umístěn v rámci diskové kvóty uživatele.
- umožňuje časové značky a "rozdílový fetch".

CKRecord

Objekt, který nese obsah jednoho záznamu CK Entity.

- má svoje CKRecordID
- je uložen do konkrétní DB (dále pak Zóny)

Operace nad Entitami/DB:

- Query - select,
- Save - změna na záznamu
- DifferentialFetch - seznamy Inserted, Updated, Deleted CKRecords/IDs.

DB operace nad CK jsou Operation objekty nad OperationQueue, tj všechno je značně ASYNC.

CKRecord - metadata záznamu

Update CKRecord. `encodeSystemFields`

```
db.save(ck) { ckrec, error in
//
    if let _ck = ckrec, error == nil {
        let archivedData = NSMutableData()
        let archiver = NSKeyedArchiver(forWritingWith: archivedData)
        archiver.requiresSecureCoding = true

        // !!!
        _ck.encodeSystemFields(with: archiver)

        archiver.finishEncoding()
        // ulozeny atribut v CKDoc
        self.ckencoded = archiver.encodedData
    }
}
```

Hlavní diskomfort CloudKitu. Návaznost na CoreData.

CloudKit+CoreData

Cca 2-3 roky stará úprava CD knihovny.

- Pokus o real-time sync CD objektů do CK (zrcadlení lokální a serverové DB).
- Pro privátní DB.
- Osobně s tím nemám velké zkušenosti.
- … spíš rozpaky.

Závěr

Je více způsobů, jak se "postarat o data" v aplikaci.

- CD - konvence E-R přístupu. Krom toho automatika sledování změn a FetchResultsController.
- Dokumenty - snadná správa. Nelze vyhledávát "skrz dokumenty".

Hybridní DB schema:

- uložené atributy CD jsou pouze ty "vyhledávané" (where ...)
- detaily kódované v BLOB, případně string (JSON).
- minimalizace DB schematu = snadná synchronizace (mirror).