Kódování dat, UIDocument

IZA, Martin Hrubý, FIT VUT, 2019

Data v aplikaci

- Uživatelské nastavení — lokálně/Cloud.
- Soubory. App Sandbox.
  - Obecně koncept správy uživatelských souborů.
- Dokumenty.
- Relační databáze.
  - Objektové kontejnery — *CoreData* a CloudKit.

Nakládání s daty

- Data *uložit* lokálně, cloudově.
  - Synchronizace dat mezi zařízeními uživatele.
  - Exportovat. Archivovat. Importovat.
- *Předat* data (síťově) — jiné aplikaci / uživateli.
  - CloudKit.
- Formátování dat.

Podpora správy dat

- Flat soubory — nějaký uživatelský formát dat.
- Dokumenty — soubor spravovaný OS.
  - UNDO, AutoSave, synchronizace, (verzování).
- Core Data — pokud mají data E-R strukturu.
- CloudKit — koncept synchronizace dat.
- UserDefaults.

Dokument

- Strukturovaná informace, se kterou *lze nakládat jako s balíkem dat*.
  - Je to izolovatelný balík objektů tvořící nějaký celek (projekt).
  - Srovnání s CD: záznamy ”všechny pohromadě v DB”.
  - Životní cyklus dokumentu: otevřít, editovat, uzavřít.
- Implementován sadou souborů (a adresářů).
  - (Má binární kódovanou podobu.)
- Metadata, synchronizace.

Formy existence dokumentu

- *Objektová paměť* — síť provázaných objektů.
- *Kódovaná podoba* objektové paměti — NSData.
  - Kódovací/Dekódovací nástroj.
- Technické uložení dokumentu na disku.
- "Ubiquitous" dokument.
  - Dokument pro zrcadlení — tzv. ubiquitous (*všudypřítomný*).

Datový obsah, kódování

- Posloupnost záznamů — pole objektů.
- Obecná objektová paměť.
  - Reference mezi objekty, raději bez cyklů…
  - Stromová struktura objektové paměti.
- Serialization — přepis objektové paměti na ”pásku”.
- Rekonstrukce obj. paměti včetně referencí.

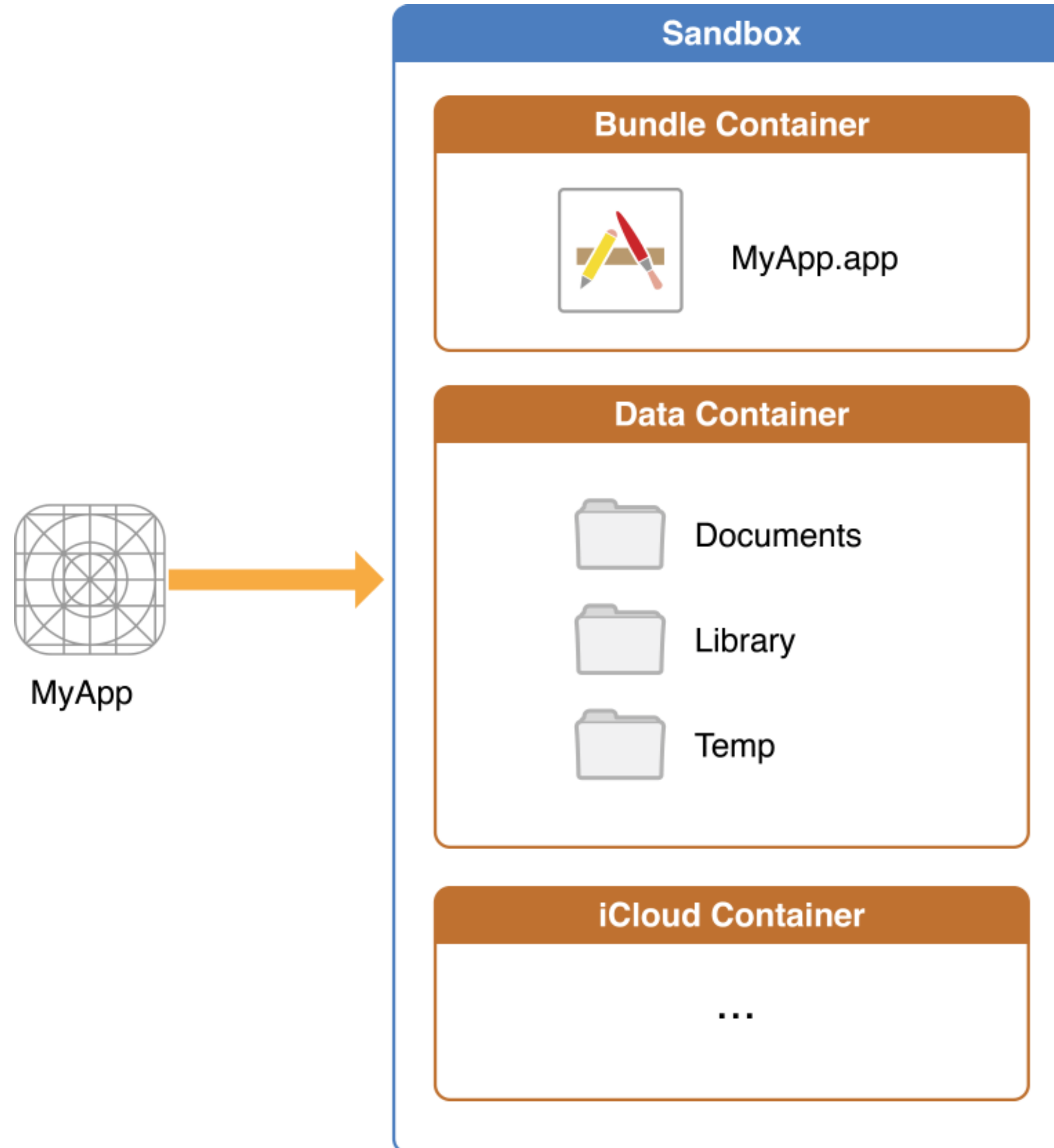
Práce se soubory

- iOS (macOS) aplikace žije v *sandboxu*. Cesty.
  - Uživatelská data aplikace. Data pro synchronizaci iCloudem.
- Datový obsah. Uložení/načtení.
  - Třída String — textový obsah.
  - Třída Data — obecný binární obsah.
  - Převody Data/String a naopak.

Cloudové dokumenty

- Uživatel aplikace *nemusí podporovat iCloud*.
  - Aplikace navíc musí předpokládat, že se toto mění (události od OS).
  - Ověřit nastavení iCloudu, vybrat adresář pro ukládání. Kdykoli soubory přemístit.
  - Události o změnách dokumentů.
  - Řešení konfliktů.
- iCloud uložení CoreData. Dnes CloudKit.

Sandbox aplikace (iOS)



Uložit obsah, String

```
//
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"))
```

do-try-catch

- I/O operace (funkce) jsou často typu "throws".
- Volat je lze s použitím "try".
  - Pokud chceme zachytit/ošetřit výjimku, pak v sekci do-try-catch.
  - "throw" hodnota — náhradní návratová hodnota funkce.
  - … musí to být objekt implementující protokol *Error*.

do-try-catch

```
// DT implementujici Error
enum BadValues: Error {
    case critical
    case justBad(howBad: String)
}

//
func readMyFile(name: String) throws -> String {
    // nejaka situace
    if (...) {
        throw BadValues.justBad(howBad: "Soubor mimo")
    }

    // jina situace
    if (...) {
        throw BadValues.critical
    }
    //
    return "Tady je obsah souboru"
}
```

do-try-catch

```
//
var cont: String? = nil

do {
    // musim volat s "try"
    cont = try readMyFile(name: "soubor")
    //
} catch BadValues.critical {
    print("Uplne kriticke")
} catch BadValues.justBad(let jakMoc) {
    //
    print("Chybne zpusobem \(jakMoc)")
}
```

try, try?, try!

- Funkci typu ″throws″ nutno volat s ″try″.
- Volající funkce má 3+1 možnosti ošetření:
  - Buď je sama ″throws″.
  - Nebo provádí do-try-catch.
  - Nebo volá ″try?″
  - Volá ″try!″ a případně slítne.

```swift
// 1) Interni osetreni chyby, vraci hodnotu
// (pokud hodnota "0" dava nejaky smysl)
func callingIO1() -> Int {
    //
    do {
        return try myIO()
    } catch {
        //
        return 0 // nebo -> Int?, nil
    }
}
// 2) Predava pripadnou vyjimku
func callingIO2() throws -> Int {
    //
    return try myIO()
}
// 3) try? vyraz je Optional, tj. Int?
func callingIO3() -> Int? {
    //
    return try? myIO()
}
// 4) try!, kaslu na chyby
func callingIO4() -> Int {
    //
    return try! myIO()
}
```

Exceptions, stylistická poznámka

- Osobně radím programovat bez Exceptions.
- Držme se funkcionálního pojetí programování:
  - **Každá funkce/procedura/podprogram vrací hodnotu (byť bool).**
  - **Návratová hodnota je struktura: obsah+status. Testovat.**
  - **Vracet výsledky "v parametrech" funkce je zlo.**
- Rozjet si v programu výjimky (C++) je jako upsat se ďáblu ;) Jednou si pro vás přijde.

Souborové I/O s Data

```swift
// 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, Foundation

- Singleton *FileManager.default* pro správu souborů.
  - URL / FilePath.
  - fileExists:, remove:, …
  - Konstrukce filePath.
- Zjišťování uživatelských adresářů:
  - home, Documents, tmp, …

```swift
let _fm = FileManager.default
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

//
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")
```

FileManager, Sandbox

```
// 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

- *Vstup*: objektová paměť.
- *Výstup*: balík data. Instance Data / NSData.
- Uložení do souboru. Transport jako balík data.
- Encoders / Decoders. Protokoly:
  - *NSCoding* — starý Foundation.
  - *Codable* — nový Foundation / Swift.
- Cykly v refencích — kodér se zacyklí.

NSCoding

- class (je pouze třídní protokol, tj. enum / struct NE)
  - init?(coder: NSCoder) — konstrukce objektu z kódu.
  - func encode(with: NSCoder) — export objektu do kodéru.
- Archiv typu Data je BLOB. Lze vložit do DB, do souboru, poslat, …
- Kodování primitivních typů, polí, dictionary…

NSCoding demo

```
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")
    }
}
```

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
let _decoded = NSKeyedUnarchiver.unarchiveObject(with: _encoded!)
// as?

if let _decoded2 = NSKeyedUnarchiver.unarchiveObject(with: _encoded!)
as? [City] {
    //
    print(_decoded2)
}
```

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]
}
```

JSON, kódování

```
//
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"}]}
}
```

JSON, .prettyPrinted

```
// JSON Encoder
let _enco = JSONEncoder()
// :)
_enco.outputFormatting = .prettyPrinted
//
if let _encoded2 = try? _enco.encode(_d1) {
    // vysledek konstrukce je String?
    if let _stringVersion = String(data: _encoded2, encoding: .utf8) {
        //
        print(_stringVersion)
    }
}
```


```
{
  "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)
}
```

```swift
//
protocol MujP {
    //
    init(fromDecoder: String)
}

// obj: T - hodnota typu T
// obj: T.Type - datovy typ hodnoty typu T
func udelej<T:MujP>(_ obj: T.Type) -> T {
    // instanciace 1
    let _val1 = obj.init(fromDecoder: "ahoj")
    // instanciace 2
    let _val2 = T(fromDecoder: "ahoj")

    //
    return _val2
}

//
struct Demo : MujP {
    //
    init(fromDecoder: String) {
        //
        print(fromDecoder)
    }
}

//
var _h = udelej(Demo.self)
```

Uživatelská manipulace kódování

```
//
struct MFile : Codable {
    //
    let name: String
    let size: Int
    let owner: String = "ja"
    var created: Date = Date()

    // definice exportu:
    // 1) exportovat se budou pouze zminene "properties"
    // 2) rawValue pojmenovava prop v kodu
    // 3) dekoder tedy bude inicializovat: name, size
    // ==> owner ma hodnotu
    // ==> zbyva dodat hodnotu "created"
    enum CodingKeys : String, CodingKey {
        case name = "podNazvem"
        case size = "delka"
    }
}
```

Implicitní konstruktor z JSON

```
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
        }
    }
}
// https://medium.com/commencis/swift-4s-codable-one-last-battle-for-serialization-30ceb3ccb051
```

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

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

PropertyListEncoder

```
//
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)!)
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://
www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>delka</key>
    <integer>1000</integer>
    <key>podNazvem</key>
    <string>soubor</string>
</dict>
</plist>
```

UIDocument

- Abstrakce nad ”dokumentem”.
- Dokument — strukturovaná informace:
  - Obsah — vzniklý typicky kódováním (je to Data).
  - Metadata — verzování, Ubiquitous, …
- Asynchronní přístup.
- Autosave.
- FileWrapper.

Návrh vlastního dokumentu

- Odvození od UIDocument.
- Implementace metod:
  - *contents(forType:) -> Data* nebo *FileWrapper*
  - *load(fromContents:ofType)*
- Dokument je datový objekt s funkcemi save/load.
  - Může být např. dataSource pro někoho.

Lifecycle Dokumentu

- Instanciace (URL), uložení ”forCreating”.
- Instanciace (URL), načtení, uložení ”forOverwriting”.
- Autosave.
- Přesun do iCloud adresáře.

Demo Document, formát obsahu

```
//
struct Note : Codable {
    //
    let text: String
    let created: Date
}

//
struct Notebook : Codable {
    //
    let name: String
    var notes = [Note]()
}

//
enum MujDocError : Error { case badCont, encError }
```

Demo, obyč kódování

```
//
class MujDOC: UIDocument {
    //
    var notebooks = [Notebook]()

    // Vystupem je bud Data() nebo FileWrapper()
    override func contents(forType typeName: String) throws -> Any {
        //
        do { return try JSONEncoder().encode(notebooks) }
        catch { throw MujDocError.encError }
    }
```

Demo, dekódování

```
//
class MujDOC: UIDocument {
    // datovy obsah
    var notebooks = [Notebook]()

    // vstupem je Data() nebo FileWrapper()
    override func load(fromContents contents: Any,
                       ofType typeName: String?) throws {
        //
        guard let _dt = contents as? Data else {
            //
            throw MujDocError.badCont
        }

        //
        if let _out = try? JSONDecoder().decode([Notebook].self, from:
_dt) {
            //
            self.notebooks = _out
        }
    }
```

Držení dokumentu aplikací

- AppDelegate, ViewController.
- Operace save/load se typicky provádějí v *MainThread*, tj. i callBacky.
  - Lze spouštět i v GlobalQueue, pokud si to uhlídate.
- UIImage — nejsou přímo Codable:
  - pokusy o konverzi na String,
  - FileWrapper.

```swift
class MujDocVC: UITableViewController {
    // kompletace URL v Documents iOS sandboxing
    let _mujDocURL = MujDOC.docURL(forFP: "tabData.json")
    var _mujDoc : MujDOC!
    //
    override func viewDidLoad() {
        //
        super.viewDidLoad()
        // instancuju objekt (zatim bez obsahu)
        _mujDoc = MujDOC(fileURL: _mujDocURL)
        // pokud existuje ulozeny dokument, zrejme nacist
        if FileManager.default.fileExists(atPath: _mujDocURL.path) {
            // aktivace nacteni, callback
            _mujDoc.open { (res:Bool) in
                // res true/false uspech operace
                if res == true {
                    self.tableView.reloadData()
                }
            }
        } else {
            _mujDoc.save(to: _mujDocURL, for: .forCreating) { (res:Bool)
in
                //
                print("Save dokonceno \(res)")
            }
        }
    }
```

Dokument obsahuje UIImage

```
//
struct Notebook {
    //
    let name: String
    let pic: UIImage?
    var notes = [Note]()

    //
    enum CodingKeys : CodingKey {
        case name
        case pic
        case notes
    }
}
```

Explicitní "Decodable"

```
extension Notebook : Decodable {
    // převzato: Hatena Blog
    init(from decoder: Decoder) throws {
        //
        let values = try decoder.container(keyedBy: CodingKeys.self)
        //
        name = try values.decode(String.self, forKey: .name)
        notes = try values.decode([Note].self, forKey: .notes)
        //
        let imageDataBase64String = try values.decode(String.self,
forKey: .pic)
        //
        if let data = Data(base64Encoded: imageDataBase64String) {
            pic = UIImage(data: data)
        } else {
            pic = nil
        }
    }
}
```

Explicitní "Encodable"

```
extension Notebook : Encodable {
    //
    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(name, forKey: .name)
        try container.encode(notes, forKey: .notes)

        if let _pic = pic, let imageData =
UIImagePNGRepresentation(_pic) {
            let imageDataBase64String = imageData.base64EncodedString()
            try container.encode(imageDataBase64String, forKey: .pic)
        }
    }
}
```

FileWrapper

- Technicky je to adresář, ke kterému se přistupuje jako k souboru.
- UIDocument načte obsah (nyní *FileWrapper*). Nutno dále dekódovat.
- Vhodné pro rozsáhlejší dokumenty, které chceme ukládat po částech (aktualizace).

Metodika FileWrapper

- FW — je buď nad souborem nebo adresářem
- Dokument alokuje adresářový FW,
  - property: fileWrappers: [String: FileWrapper]
  - jednotlivé FW pro elementární soubory
- Elem. FW — JSON kód, přílohy: obrázky apod.

FileWrapper jednoho souboru

```
// 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
}
```

FW adresář, Dokument

```
//
    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
    }
```

FW, Dokument

```
//
    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"] {
            //
            if let _dt = _picFW.regularFileContents {
                //
                pic = UIImage(data: _dt)
            }
        }
    }
```

Dokumenty v iCloud

- Oddělené adresáře pro soubory (dokumenty) ryze lokální a cloudové
  - FileManager: (forUbiquityContainerIdentifier: nil)
- Uživatel musí mít povoleny Cloud-Doc
  - Ke všemu se to ještě může měnit.

iCloud

- Postup:
  - Vytvořte lokální dokument.
  - Konstruujte URL pro jeho umístění v iCloud-adresáři.
  - FileManager: přesun/kopie souboru
  - setUbiquitous: (X:Bool, itemAt:URL, destinationURL: URL)
  - X==true, dokument se stává ubiqiutous
  - false, přestává být…
- setUbiquitous: volat z GlobalQueue

Přístup ke Cloud Dokumentům

```
class ICloud {
    //
    static let shared = ICloud()

    //
    var ubiqDocs : URL? {
        //
        guard let _ubis =
FileManager.default.url(forUbiquityContainerIdentifier: nil) else
{ return nil }

        //
        return _ubis.appendingPathComponent("Documents", isDirectory:
true)
    }

    //
    var cloudON : Bool { return ubiqDocs != nil }
}
```

```
// pripadne kodovat tri navratove hodnoty
    func setCloudable(fileURL: URL) -> Bool {
        //
        guard let _cloud = ubiqDocs else {
            // cloud neni
            return false
        }

        //
        let _fn = fileURL.lastPathComponent
        let _destURL = _cloud.appendingPathComponent(_fn)

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

        // okay
        return true
    }
```

Dynamika práce s cloud-doc

- Dokument byl vytvořen a označen jako UBIQ.
- iOS organizuje jeho synchronizaci.
- Potřebujeme notifikace o aktualizacích.
  - Řešení kolizí.
  - UndoManagement.

Sledování notifikací

- UIDocument je FilePresenter (controller nad dokumentem).
- Sleduje stav cloud-dokumentu. Posílá Notification.
    - ViewController (např) není přímým delegátem Document.

Notifikace o změně UBIQ-doc

- Document.documentState — bitový vektor stavových informací:
  - .normal, .progresAvailable, .inConflict, .editingDisabled
  - Registrace k odběru notifikací
  - Odchycení .progresAvailable
  - Pak odchycení .normal
  - Pak reload

```swift
//
    var _anyProgress = false
    //
    override func viewWillAppear(_ animated: Bool) {
        // pak odregistrovat...!

NotificationCenter.default.addObserver(forName: .UIDocumentStateChanged,
                                               object: mujDoc,
                                               queue: nil) { _  in
            //
            if self.mujDoc.documentState.contains(.progressAvailable) {
                //
                self._anyProgress = true
            }

            //
            if self.mujDoc.documentState == [UIDocumentState.normal] {
                //
                if self._anyProgress {
                    //
                    self._anyProgress = false
                    self.mujDoc.reload()
                }
            }
        }
    }
```

Závěr & Příště

- Ukládání dat: CoreData, Codable, UIDocument.
  - Opomenuto: UNDO management.
- Příště CloudKit, tj. nástroj pro synchronizaci dat:
  - mezi zařízeními uživatele (iOS/macOS, tvOS),
  - mezi uživateli nějaké pracovní skupiny,
  - … pro archivaci dat uživatele.