eBook Gratuit

APPRENEZ watchkit

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#watchkit

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: watchkit

It is an unofficial and free watchkit ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official watchkit.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec watchkit

Remarques

- Documentation Apple sur WatchKit (via la référence API): https://developer.apple.com/reference/watchkit
- Se familiariser avec Xcode: interface Xcode

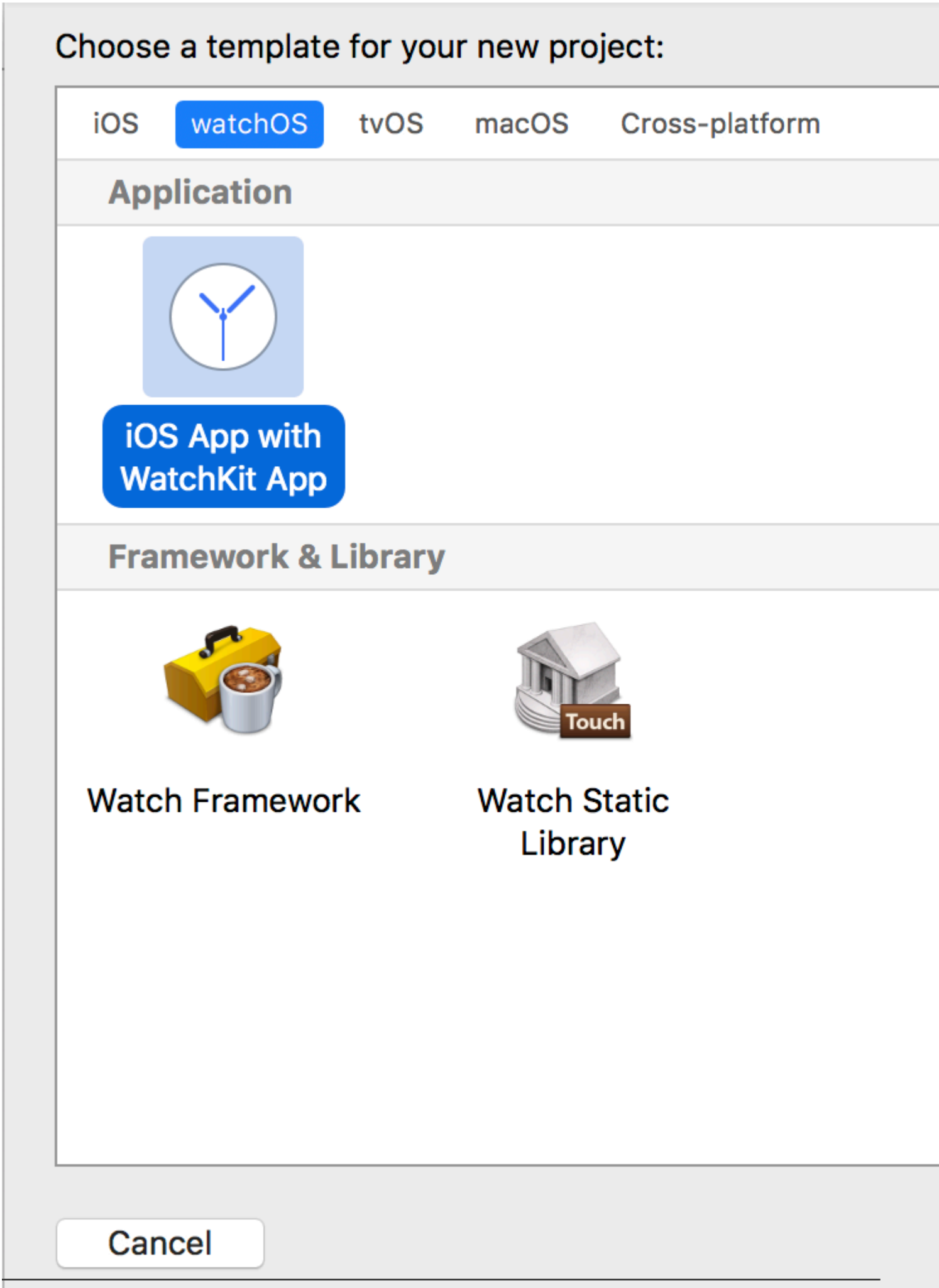
Examples

Créer un nouveau projet watchOS

Pour développer une application pour watchOS, vous devez commencer par Xcode. Xcode ne fonctionne que sur MacOS. Au moment de la rédaction de ce document, la dernière version est Xcode 8.3.

Si vous souhaitez démarrer un nouveau projet à partir de zéro:

1. Démarrez votre Mac et installez Xcode depuis l'App Store s'il n'est pas déjà installé.
2. Choisissez de créer un nouveau projet.
3. Dans les modèles, choisissez watchOS puis "App iOS avec l'application WatchKit".

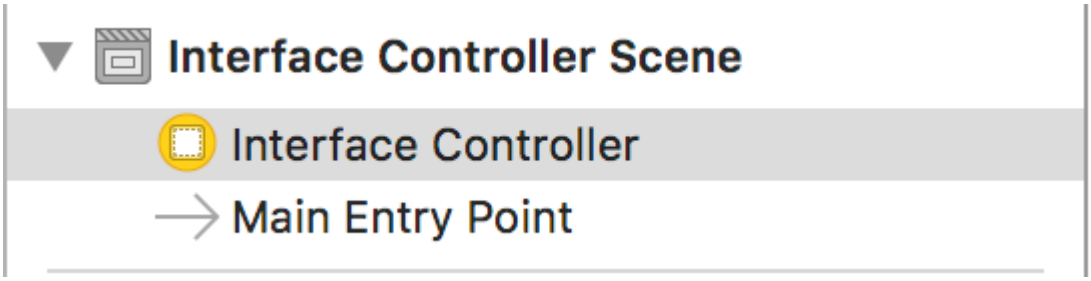

Choose a template for your new project:
iOS
watchOS
tvOS
macOS
Cross-platform
Application
iOS App with WatchKit App
Framework & Library
Touch
Watch Framework
Watch Static Library
Cancel

lequel vous concevez l'application et un fichier `Assets.xcassets` pour y placer vos ressources.

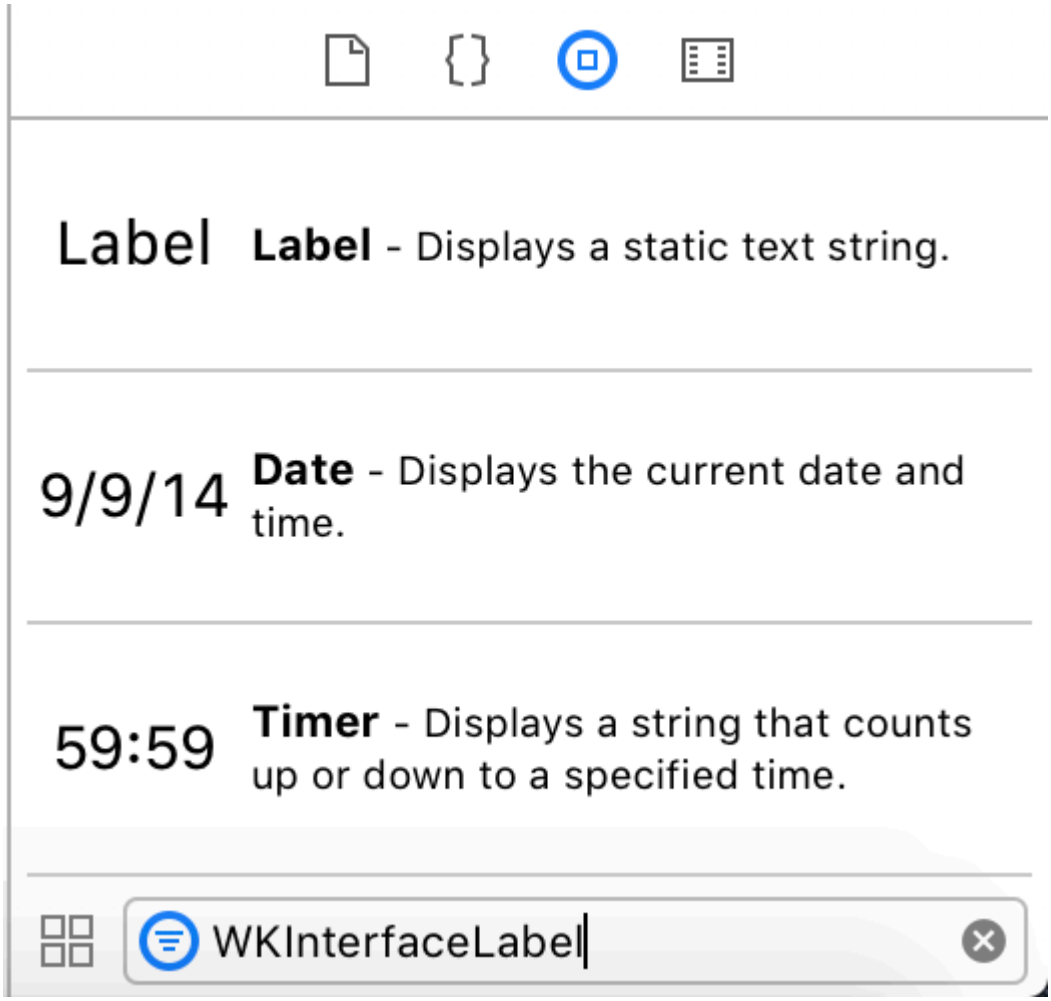
Chaque extension WatchKit a un fichier `InterfaceController.swift` (en fait une sous-classe `WKInterfaceController`) similaire au fichier `ViewController` dans iOS.

Pour créer une application Hello World:

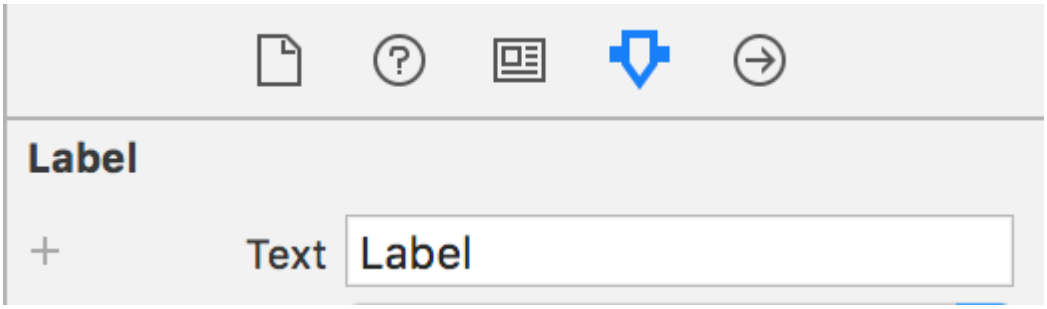
1. Ouvrez l' `Interface.storyboard` .
2. Localisez le principal `InterfaceController` .



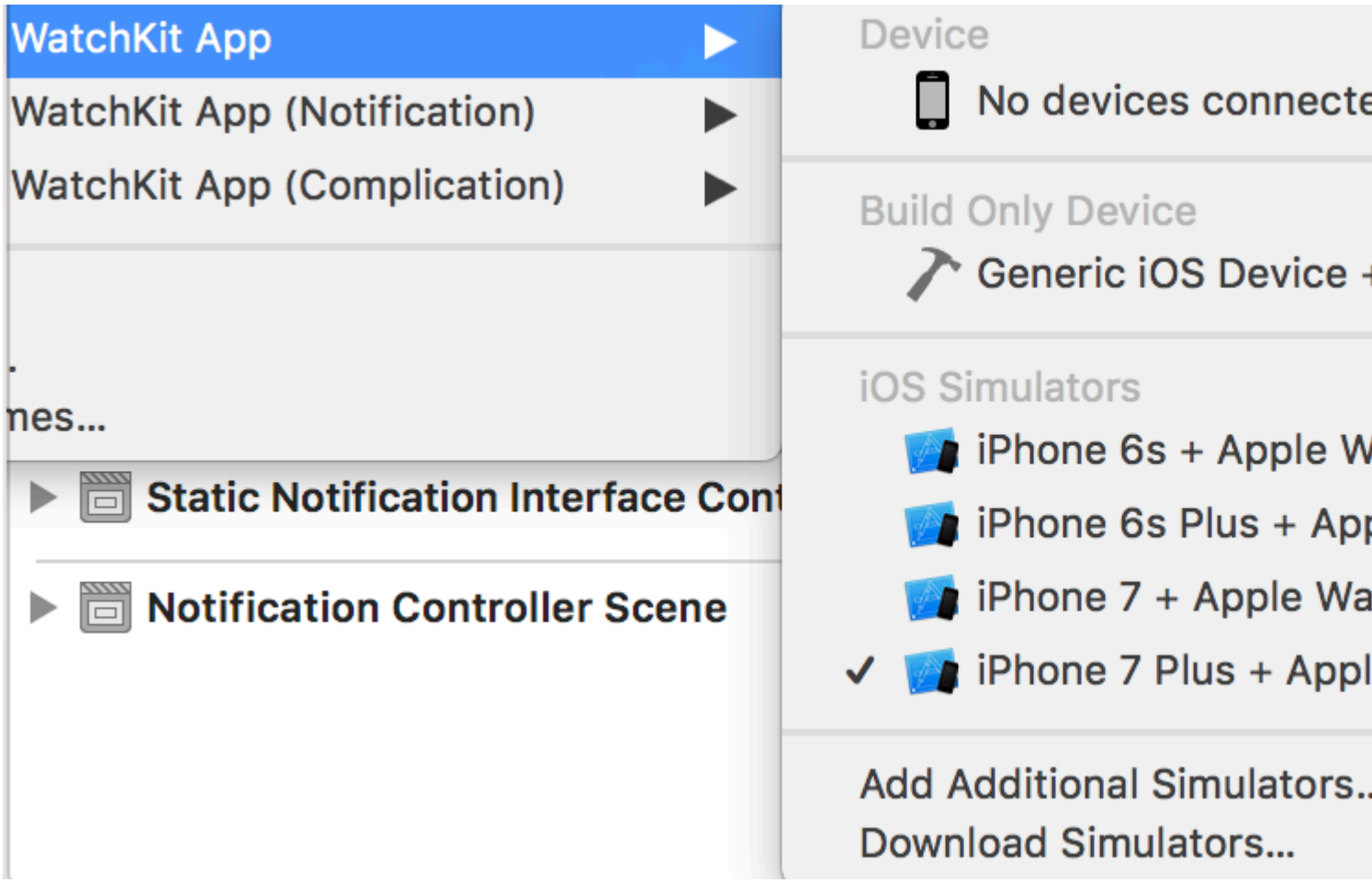
3. Dans la bibliothèque du volet de droite, ajoutez un `WKInterfaceLabel` .

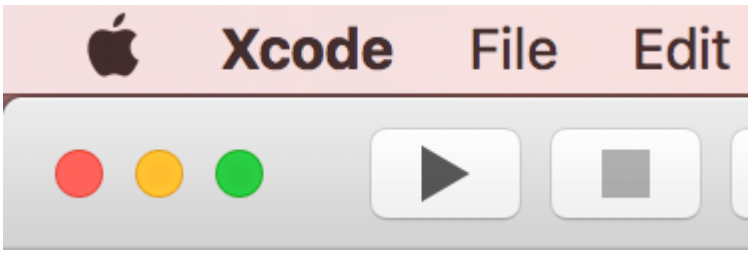


4. Faites glisser l'étiquette et définissez son texte dans le volet de droite sur "Hello, World!".



5. Sélectionnez le bon schéma (selon l'image suivante), puis exécutez le projet en appuyant sur le bouton Exécuter dans la barre supérieure, en utilisant le menu Produit, en appuyant sur Cmd-R ou en appuyant sur Exécuter dans la barre tactile.





Le simulateur Apple Watch sera éventuellement présenté avec votre application en cours d'exécution.

Connecter le code avec l'interface utilisateur

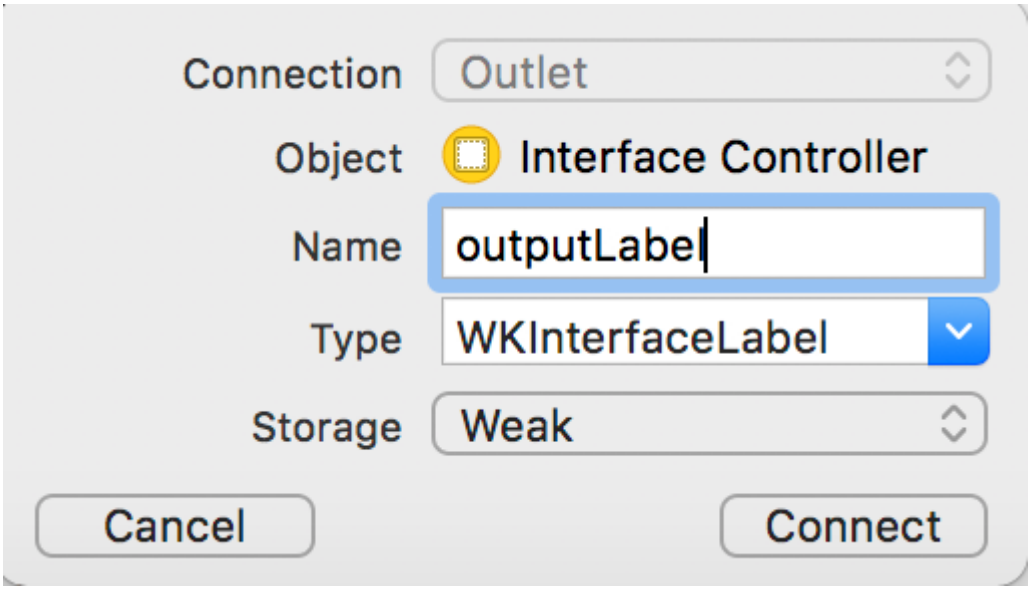
Comme pour iOS, où vous utilisez `@IBOutlet` et `@IBAction` , vous pouvez également les utiliser.

Disons que nous avons un bouton qui, lorsqu'il est cliqué, change le texte de l'étiquette en autre chose.

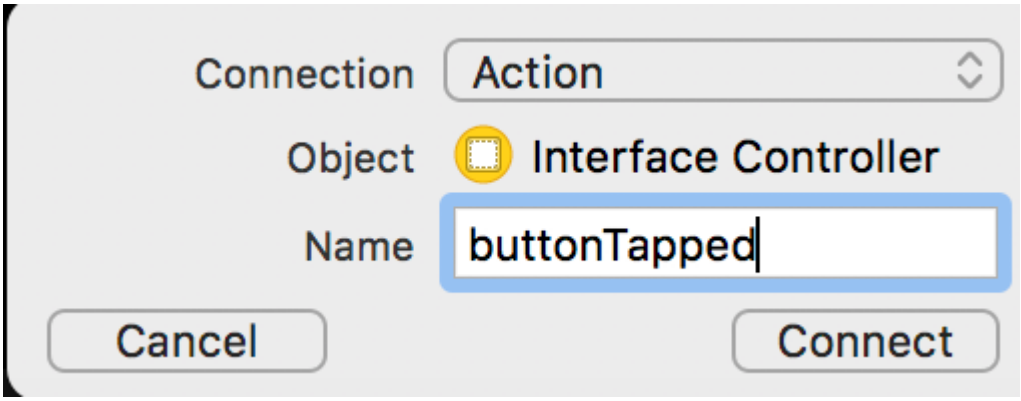
Pour commencer:

1. Ajoutez un `WKInterfaceLabel` et un `WKInterfaceLabel` à `InterfaceController` .

2. Faites glisser le curseur depuis le `WKInterfaceLabel` vers `InterfaceController.swift` et entrez les détails comme indiqué dans l'image suivante pour ajouter une propriété de sortie:



3. Faites glisser le curseur depuis le `WKInterfaceButton` vers `InterfaceController.swift` et entrez les détails, comme illustré ci-dessous, pour ajouter une méthode d'action:



5. Remplissez la méthode d'action:

Rapide

```
outputLabel.setText("Button Tapped!")
```

Objectif c

```
[[self outputLabel] setText:@"Button Tapped!"]
```

6. Exécutez le programme et appuyez sur le bouton pour voir le résultat.

Lire Démarrer avec watchkit en ligne: https://riptutorial.com/fr/watchkit/topic/9568/demarrer-avec-

watchkit

Chapitre 2: La navigation

Remarques

Note importante

> Apple a fortement déconseillé l'utilisation des deux styles de navigation dans un seul contrôleur, ce qui peut entraîner un rejet de l'application.

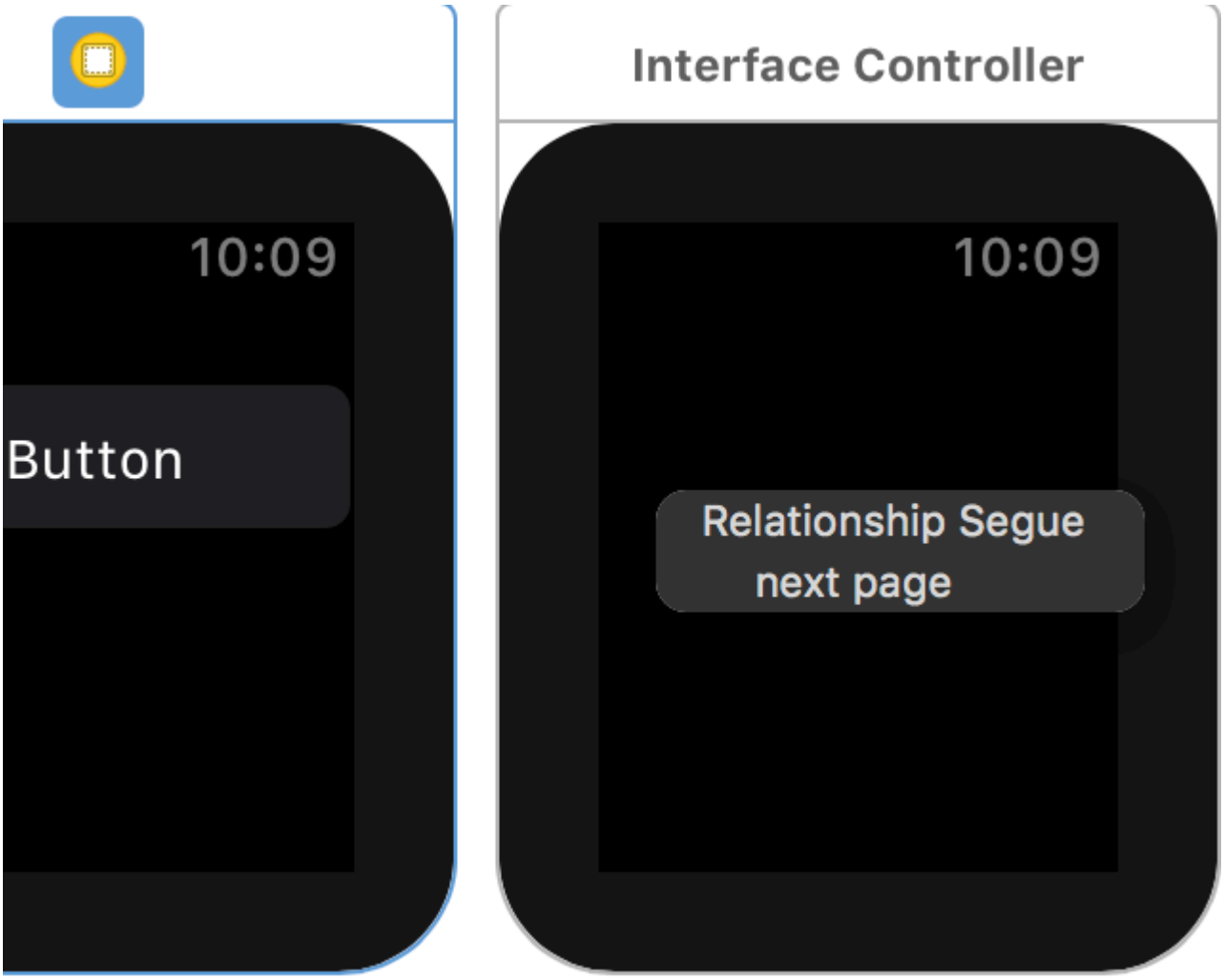
Actuellement, la méthode privilégiée consiste à utiliser un style hiérarchique plutôt que basé sur une page, tel qu'utilisé dans de nombreuses applications Apple.

Examples

Navigation par page

De nombreuses applications watchOS (comme Activity) ont plusieurs pages que vous pouvez simplement faire défiler entre elles, ce qui est un très bon moyen d'utiliser Apple Watch.

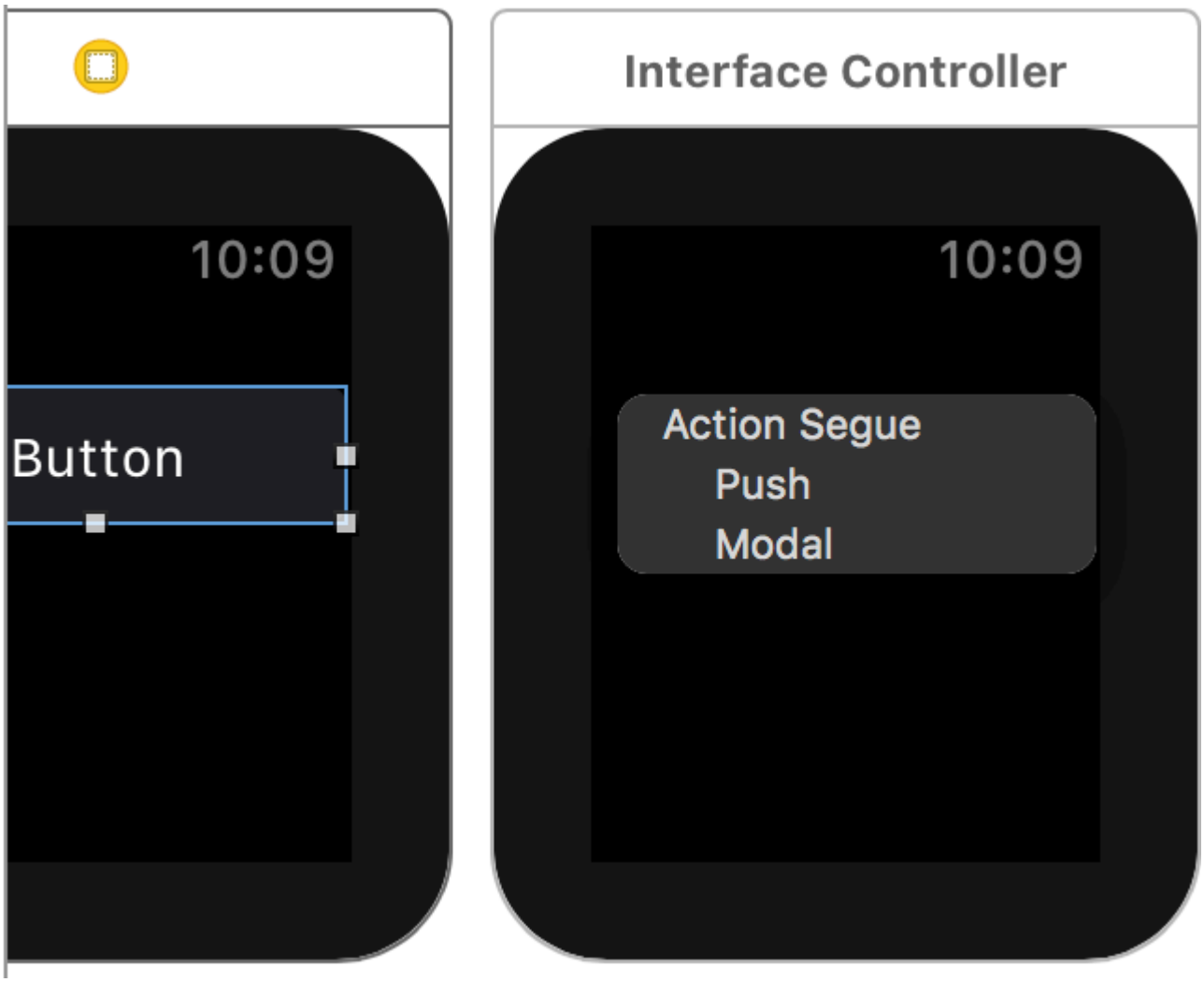
Pour créer une navigation basée sur une page, faites-la glisser d'un contrôleur à un autre et sélectionnez "page suivante", comme illustré ci-dessous:



Structure de vue hiérarchique

De nombreuses applications watchOS (telles que Workout, Weather, Music, etc.) ont une `WKInterfaceTable` principale ou un ensemble de boutons reliés à un autre contrôleur, similaire à la navigation sur iOS. Cela s'appelle la structure de vue hiérarchique.

Pour connecter un bouton, faites-le glisser du bouton vers un contrôleur et sélectionnez "push" ou "modal" en fonction de vos besoins, comme illustré ci-dessous:



Lire La navigation en ligne: https://riptutorial.com/fr/watchkit/topic/9599/la-navigation

Chapitre 3: WatchConnectivity

Introduction

Connecter votre application WatchOS à votre application iOS peut être une tâche à accomplir lorsque vous ne l'avez jamais fait auparavant. Ce tutoriel vous montrera les bases fondamentales pour accomplir cette tâche très importante.

Examples

Configuration iOS

Application iPhone

1. importez WatchConnectivity et conformez-vous à WCSessionDelegate.
2. utilisez le délégué de session statique via `WCSession.default()` .
3. Envoyez des données à l'application Watch en utilisant:

```
WCSession.default().sendMessage(message, replyHandler:_ errorHandler:_)
```

4. L'objet message doit être un dictionnaire de type [String: Any]
5. Si vous recherchez des données à retourner depuis l'application Watch, indiquez la logique d'une fermeture définie dans le paramètre replyHandler. sinon, passer à zéro.
6. Pour répondre aux messages envoyés depuis l'application Watch, vous utiliserez la méthode de rappel WCSessionDelegate

```
func session(_ session: WCSession, didReceiveMessage message: [String : Any],
replyHandler: @escaping ([String : Any]) -> Swift.Void){...}
```

Vous devez importer le framework WatchConnectivity dans votre fichier de *contrôleur de vue* dans l'application iOS. Cela vous donne accès aux objets de classe conçus pour communiquer avec l'application associée Watch. Vous devrez également importer cette fonctionnalité du côté de l'application Watch. La seule vraie différence avec l'application Watch est que vous n'aurez pas de fichier de *contrôleur de vue* , mais un fichier de *contrôleur d'interface* .

```
import WatchConnectivity
```

Ensuite, vous devrez vous assurer que votre application peut prendre en charge une session pour WatchConnectivity. Si c'est le cas, vous devez définir le contrôleur de vue comme délégué et activer la session par défaut. Vous obtiendrez une erreur ici. Vous devez vous conformer à WCSessionDelegate et implémenter quelques méthodes avant que l'EDI ne commence à se calmer.

```
// MARK: - View Life Cycle Callbacks
override func viewDidLoad() {
    super.viewDidLoad()

    automaticallyAdjustsScrollViewInsets = false
    if WCSession.isSupported() {
        WCSession.default().delegate = self
        WCSession.default().activate()
    }
    else {
        print("\nViewController: connectionManager is nil\n")
    }
}
```

Pour se conformer à WCSessionDelegate, ajoutons une extension au bas du contrôleur de vue. Certaines personnes détestent cette approche. J'ai mon chemin, mais pour des raisons didactiques, je vais suivre l'approche *RayWenderlich* . Je suis un grand fan de simplement obtenir le code directement de COMMAND + CLICKING sur les délégués et de récupérer TOUTES LES MÉTHODES hors de la spécification et de commencer à le manipuler et à comprendre comment les choses fonctionnent. Dans cette extension, je vous fournis toutes les méthodes. Ils sont déjà marqués pour que vous puissiez voir quand chacun tirera dans la console. Si vous vous sentez fringant, supprimez certains de ceux qui sont marqués FACULTATIFS pour que votre fichier de code ait l'air épatant.

```
extension ViewController : WCSessionDelegate {

    func session(_ session: WCSession, activationDidCompleteWith activationState:
WCSessionActivationState, error: Error?) {
        print("0. ViewController: ", "activationDidCompleteWith activationState")
    }


    /** ------------------------ iOS App State For Watch ----------------------- */

    func sessionDidBecomeInactive(_ session: WCSession) {
        print("1. ViewController: ", "sessionDidBecomeInactive")
    }


    func sessionDidDeactivate(_ session: WCSession) {
        print("2. ViewController: ", "sessionDidDeactivate")
    }


    func sessionWatchStateDidChange(_ session: WCSession) {
        print("3. ViewController: ", "sessionDidDeactivate")
    }


    /** ------------------------ Interactive Messaging ------------------------ */

    func sessionReachabilityDidChange(_ session: WCSession) {
        print("4. ViewController: ", "sessionReachabilityDidChange")
    }


    func session(_ session: WCSession, didReceiveMessage message: [String : Any]) {
```

```
        print("5. ViewController: ", "didReceiveMessage")
    }



    func session(_ session: WCSession, didReceiveMessage message: [String : Any],
replyHandler: @escaping ([String : Any]) -> Swift.Void) {
        print("6. ViewController: ", "didReceiveMessage")
        // This is where you handle any requests coming from your Watch App
    }

    func session(_ session: WCSession, didReceiveMessageData messageData: Data) {
        print("7. ViewController: ", "didReceiveMessageData")
    }

    func session(_ session: WCSession, didReceiveMessageData messageData: Data, replyHandler:
@escaping (Data) -> Swift.Void) {
        print("8. ViewController: ", "didReceiveMessageData")
    }

    /** ------------------------- Background Transfers ------------------------- */

    func session(_ session: WCSession, didReceiveApplicationContext applicationContext:
[String : Any]) {
        print("9. ViewController: ", "didReceiveApplicationContext")
    }

    func session(_ session: WCSession, didFinish userInfoTransfer: WCSessionUserInfoTransfer,
error: Error?) {
        print("10. ViewController: ", "didFinish userInfoTransfer")
    }

    func session(_ session: WCSession, didReceiveUserInfo userInfo: [String : Any] = [:]) {
        print("11. ViewController: ", "didReceiveUserInfo")
    }

    func session(_ session: WCSession, didFinish fileTransfer: WCSessionFileTransfer, error:
Error?) {
        print("12. ViewController: ", "didFinish fileTransfer")
    }

    func session(_ session: WCSession, didReceive file: WCSessionFile) {
        print("13. ViewController: ", "didReceive file")
    }
}
```

Pour envoyer des données à votre Watch App, une fois que l'application Watch l'a demandée, vous allez gérer cela dans la méthode ...

```
func session(_ session: WCSession, didReceiveMessage message: [String : Any], replyHandler:
@escaping ([String : Any]) -> Swift.Void) {
    print("6. ViewController: ", "didReceiveMessage")
    // build out your response message using a Dictionary
    let returnMessage: [String : Any] = [
       "key1" : value1,
       "key2" : value2,
       "key3" : value3
    ]
    // return your data in this manner
    replyHandler(returnMessage)
```

```
    // WARNING
    // You must call the replyHandler before the method ends, otherwise, your app will crash.
}
```

C'est seulement la moitié de la transaction! Vous devez configurer votre application Watch pour vous connecter à l'application iPhone et gérer les messages renvoyés!

Regarder la configuration de l'extension

App WatchKit

1. importez WatchConnectivity et conformez-vous à WCSessionDelegate.
2. utilisez le délégué de session statique via `WCSession.default()` .
3. Envoyez des données à l'application iPhone en utilisant:

```
WCSession.default().sendMessage(message, replyHandler:_ errorHandler:_)
```

4. L'objet message doit être un dictionnaire de type [String: Any]
5. Si vous recherchez des données à retourner depuis l'application Watch, indiquez la logique d'une fermeture définie dans le paramètre replyHandler. sinon, passer à zéro.
6. Pour répondre aux messages envoyés depuis l'application iPhone, vous utiliserez la méthode de rappel WCSessionDelegate

```
func session(_ session: WCSession, didReceiveMessage message: [String : Any],
replyHandler: @escaping ([String : Any]) -> Swift.Void){...}
```

7. Ces méthodes ne seront pas nécessaires dans votre application Watch pour se conformer correctement à WCSessionDelegate:

```
func sessionDidBecomeInactive(_ session: WCSession)
func sessionDidDeactivate(_ session: WCSession)
func sessionWatchStateDidChange(_ session: WCSession)
```

Enfin, il est généralement recommandé de stocker les images communes, les images et les éléments qui seront utilisés en permanence pour que l'application Watch soit placée dans le dossier xcassets de Watch. Ok, maintenant que vous êtes confus, passons aux détails!

Cela peut être une surprise pour vous, mais vous devez importer à nouveau WatchConnectivity.

```
import WatchConnectivity
```

Ensuite, vous devez vérifier que la session est même possible.

```
// MARK: - View Life Cycle Callbacks
override func awake(withContext context: Any?) {
    super.awake(withContext: context)

    // Configure interface objects here.
```

```
    if WCSession.isSupported() {
        WCSession.default().delegate = self
        WCSession.default().activate()
        print("InterfaceController: Session Activated")

        // Request Data from iPhone App
        let requestMessage = ["message":"get-data"]

        WCSession.default().sendMessage(requestMessage, replyHandler: { (replyMessage) in
            print("Got a reply from the phone: \(replyMessage)")

            // handle reply message here

        }, errorHandler: { (error) in
            print("Got an error sending to the phone: \(error)")
        })
    }
    else {
        print("\nViewController: connectionManager is nil\n")
    }
}
```

Mais tout cela ne fonctionnera pas, sauf si vous implémentez des méthodes requises pour WCSessionDelegate.

```
extension InterfaceController : WCSessionDelegate {

    func session(_ session: WCSession, activationDidCompleteWith activationState:
WCSessionActivationState, error: Error?) {
        print("1. InterfaceController: ", "activationDidCompleteWith activationState") //
first
    }

    /** ------------------------- Interactive Messaging ------------------------- */

    func sessionReachabilityDidChange(_ session: WCSession) {
        print("2. InterfaceController: ", "sessionReachabilityDidChange") // second
    }

    func session(_ session: WCSession, didReceiveMessage message: [String : Any]) {
        print("3. InterfaceController: ", "didReceiveMessage")
    }

    func session(_ session: WCSession, didReceiveMessage message: [String : Any],
replyHandler: @escaping ([String : Any]) -> Swift.Void) { // third
        print("4. InterfaceController: ", "didReceiveMessage")
        //print("Message Contents: ", message["message"]!)
    }

    func session(_ session: WCSession, didReceiveMessageData messageData: Data) {
        print("5. InterfaceController: ", "didReceiveMessageData")
    }

    func session(_ session: WCSession, didReceiveMessageData messageData: Data, replyHandler:
@escaping (Data) -> Swift.Void) {
        print("6. InterfaceController: ", "didReceiveMessageData")
    }

    /** ------------------------- Background Transfers ------------------------- */
```

```
    func session(_ session: WCSession, didReceiveApplicationContext applicationContext:
[String : Any]) {
        print("7. InterfaceController: ", "didReceiveApplicationContext")
    }

    func session(_ session: WCSession, didFinish userInfoTransfer: WCSessionUserInfoTransfer,
error: Error?) {
        print("8. InterfaceController: ", "didFinish userInfoTransfer")
    }

    func session(_ session: WCSession, didReceiveUserInfo userInfo: [String : Any] = [:]) {
        print("9. InterfaceController: ", "didReceiveUserInfo")
    }

    func session(_ session: WCSession, didFinish fileTransfer: WCSessionFileTransfer, error:
Error?) {
        print("10. InterfaceController: ", "didFinish fileTransfer")
    }

    func session(_ session: WCSession, didReceive file: WCSessionFile) {
        print("11. InterfaceController: ", "didReceive file")
    }
}
```

Envoi de données

Envoyer des données à votre application Watch de votre iPhone ou de votre iPhone à votre application Watch après avoir déclenché un événement est très simple, même si le code peut paraître un peu complexe au début.

```
let message = ["key":"value-to-send"]

WCSession.default().sendMessage(message, replyHandler: { (replyMessage) in
    print("Got a reply from the phone: \(replyMessage)")

    if let returnedValues = replyMessage["returned-value"] as? NSArray {
        for val in returnedValues {
            // do something here with the data
            // Dispatch to Main Thread if affecting UI
        }
    }
}, errorHandler: { (error) in
    print("Got an error sending to the phone: \(error)")
})
```

Lire WatchConnectivity en ligne: https://riptutorial.com/fr/watchkit/topic/10733/watchconnectivity

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec watchkit	Community, Seyyed Parsa Neshaei
2	La navigation	Seyyed Parsa Neshaei
3	WatchConnectivity	mrfilter