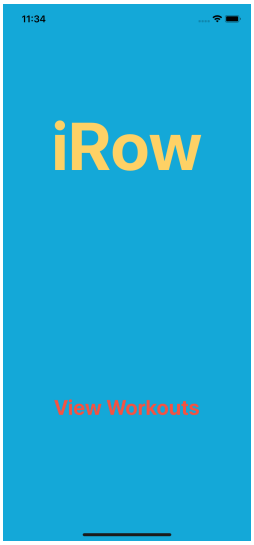
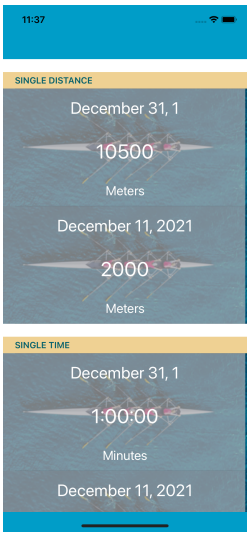
iRow Watch App Tutorial
By: Jacob Stone and Isabel Albaitis

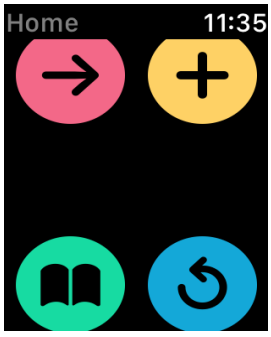
Overview

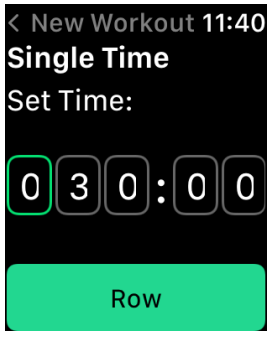
The iRow app was an attempt to study the watchKit focus area for the Xcode platform for making apple apps. WatchKit was designed to allow users to create their own watch apps to be implemented on their apple watch devices.

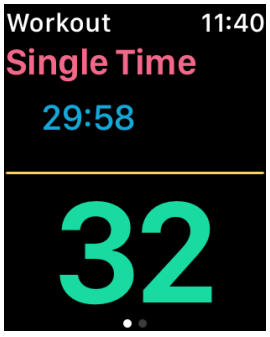
Our app, iRow, uses this feature to implement a watch based program to help track rowing workouts. The user can select a custom workout on their watch, and it will automatically send to the phone app to be displayed for them. The user will no longer worry about remembering specific details about their workouts from prior days as they will now have a catalog of workouts and dates that they have done from the past.











Displayed: The iPhone and Watch App in full use, displaying its capabilities. Example screenshots show a new workout being chosen on the watch and then added to the list on the iPhone app once the workout is stopped.

Getting Started

To start, the user will need the XCode software from Apple. The user will also need a Mac computer of some type to use this software since it is exclusive to Apple products alone. Getting a brief overview of the WatchKit extension may be helpful when coding. Overview can be found here. Another useful link would be for WatchConnectivity. This was a high level interface on the core platform that we studied to better help us make our app. Link is here.

Instructions

iOS App

1. Open XCode and select WatchOS under templates. Then select “iOS app with Watch App”
2. Create project with given name
3. In the Main.storyboard, create a UIView called LogInView. This view will be the first screen the user sees when entering the iphone app.
4. Add the “iRow” Label and “View Workouts” Button to the screen, constrain as needed.
5. Implement a navigation controller to help segue the screen into another scene.
6. The other scene will be the WorkoutView. This is where the user will see the list of all their workouts laid out for them
7. Add a table View and prototype table Cell into the view. Constrain as needed.
8. The Cell will contain an ImageView, a translucent view that is a UIView, and a stack View with three Labels inside. These labels will be what change for each workout. The Date Label, Measured Label, and Units Label will be displayed in the prototype cell.
9. Connect these labels and images to a tableViewCell class called WorkoutCell. This will be used later in code.

```
import UIKit

class WorkoutMainTableViewCell: UITableViewCell {
    @IBOutlet weak var Units: UILabel!
    @IBOutlet weak var Date: UILabel!
    @IBOutlet weak var Measure: UILabel!
    @IBOutlet weak var translucentView: UIView!
    @IBOutlet weak var coverImage: UIImageView!
```

10. Create two Model Level classes called “Workout” and “WorkoutModel”. These will be used to help display the data into individual cells in WorkoutView

Workout.swift

```swift
import Foundation


struct Workout {
    var key : String?
    var type : Int? // single or interval, time or distance
    var date : Date? // MM/DD/YY format
    var meters : Int? // total meters rowed
    var rest : TimeInterval? // avg 500m split
    var totalTime : TimeInterval? //

    init(key: String?, type: Int?, date: Date?, meters: Int?, rest: Double?, totalTime: Double?)
    {
        self.key = key
        self.type = type
        self.date = date
        self.meters = meters
        self.rest = rest
        self.totalTime = totalTime
    }

    init(type: Int?, date: Date?,
         meters: Int?, rest: Double?, totalTime: Double?)
    {
        self.init(key: nil, type: type,  date: date,
                  meters: meters, rest: rest, totalTime: totalTime)
    }

    init() {
        self.init(key: nil, type: nil, date: nil,
                  meters: nil, rest: nil, totalTime: nil)
    }

}
```

WorkoutModel.Swift

```swift
import Foundation
import WatchConnectivity

class WorkoutModel {
    fileprivate var items: [Workout] = [Workout]()

    init()
    {
        createList()
    }

    func getWorkouts() -> [Workout]
    {
        return self.items
    }

    fileprivate func createList() // example list to show how data is displayed
    {                             // OPTIONAL: NOT NEEDED FOR CODE TO COMPILE, SCREEN WILL JUST START A
        items.append(
                Workout( // single distance
                    type: 1,
                    date: Date.distantPast,
                    meters: 10500,
                    rest: 0,
                    totalTime: 2520)
            )
        items.append(
            Workout( // single time
                    type: 0,
                    date: Date.distantPast,
                    meters: 15000,
                    rest: 0,
                    totalTime: 3600)
            )
```

11. Create the LogInViewController and WorkoutViewController. These will be used as custom classes for the Views on the storyboard so they can be connected to the code.
12. Add an IBOutlet for the “View Workouts” button in the LoginViewController class.
13. Add an IBAction for the button so that when pressed, the screen will segue views into the WorkoutScene

```swift
@IBAction func ViewWorkoutsPressed(_ sender: Any) {
    self.performSegue(withIdentifier: "LogInToMain", sender: self)
```

```
}
```

14. The WorkoutViewController class will need a little more instruction
    a. Import WatchConnectivity

```
import WatchConnectivity
```

    b. Add the Following to the class stub

```
class WorkoutViewController: UIViewController, UITableViewDataSource, UITableViewDelegate,
WCSessionDelegate {
```

These will be needed to help us with the TableView and the transfer of data from the Watch to the App. Also create the helper functions that are needed for these helper classes.

    c. Create an IBOutlet for the Workout Table

```
@IBOutlet weak var WorkoutTable: UITableView!
```

    d. Add the following lines of code before and into the ViewDidLoad function. This will be used to establish the View and instantiate the Model and the Connectivity Session for transferring data

```
var workouts: [Workout]?
var session: WCSession?

override func viewDidLoad() {
    super.viewDidLoad()
    let model = WorkoutModel()
    self.workouts = model.getWorkouts()
    if WCSession.isSupported() {
        let session = WCSession.default
        session.delegate = self
        session.activate()
    }
    self.sortIntoSections(workouts: self.workouts!)
}
```

    e. To connect to the Watch that sent a message to us, we need to receive the Message. We need to create the didReceiveMessage function. The userInfo contains the info from the watch. We need to assign it to a workout and add it to our list of workouts. We can then re-sort our data into a compiled list to be viewed

```
34      func session(_ session: WCSession, didReceiveMessage userInfo: [String : Any] = [:]) {
35          self.workouts?.append(
36              Workout( // from user
37                  type: (userInfo["Type"] as! Int),
38                  date: Date.now, // since it is right after user finishes workout. Date will always be today
39                  meters: ((userInfo["Distance"]) as! Int),
40                  rest: (userInfo["Rest Seconds"] as? Double),
41                  totalTime: (userInfo["Work Seconds"] as? Double)
42                  )
43          )
44          self.sortIntoSections(workouts: self.workouts!)
45      }
```

    f. After this, we can move on to the next function we need to create, the sortIntoSections function. This is used to sort the data by type of workout so we can view it more neatly on the screen

```
    func sortIntoSections(workouts: [Workout]) {
        // This function sorts the data into the workout type to better fit the table View
        var SingleDistSection  = [Workout]()
        var SingleTimeSection = [Workout]()
        var IntervalDistSection = [Workout]()
        var IntervalTimeSection = [Workout]()

        for typ in workouts {
            if typ.type == 0 {
                SingleTimeSection.append(typ)
            }
            else if typ.type == 1 {
                SingleDistSection.append(typ)
            }
            else if typ.type == 2{
                IntervalTimeSection.append(typ)
            }
            else{ // for "Distance Intervals" a.k.a 3
                IntervalDistSection.append(typ)
            }
        }
        var tmpData: [(sectionHeader: String, workouts: [Workout])] = []
        if SingleDistSection.count > 0 {
            tmpData.append((sectionHeader: "SINGLE DISTANCE", workouts: SingleDistSection))
        }
        if SingleTimeSection.count > 0 {
            tmpData.append((sectionHeader: "SINGLE TIME", workouts: SingleTimeSection))
        }
        if IntervalTimeSection.count > 0 {
            tmpData.append((sectionHeader: "INTERVAL TIME", workouts: IntervalTimeSection))
        }
        if IntervalDistSection.count > 0 {
            tmpData.append((sectionHeader: "INTERVAL DISTANCE", workouts: IntervalDistSection))
        }
        self.tableViewData = tmpData
    }
```

g. Fill in the appropriate functions for TableViewDelegate to desired heights, headers, and titles

h. Then we can move to filling the cells with the appropriate data from our models.

```
@available(iOS 15.0, *)
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
    -> UITableViewCell
{
    let cell = self.WorkoutTable.dequeueReusableCell(withIdentifier: "ImageCell", for: indexPath) as!
        WorkoutMainTableViewCell

    guard let workoot = tableViewData?[indexPath.section].workouts[indexPath.row] else {
        return cell
    }

    cell.Date?.text = workoot.date?.formatted(date: Date.FormatStyle.DateStyle.long, time:
        Date.FormatStyle.TimeStyle.omitted)
    cell.coverImage?.image = UIImage(named: "Quad")
```

The first part connects the cell to the view using the “ImageView” identifier from our storyboard view. Then we connect the data to a workout

i. Next we connect the cell to the workout data and display depending on the type of workout that was completed. This will assure that the correct units are shown for differing types of workouts (Time Vs. Distance)

```
        cell.Date?.text = workoot.date?.formatted(date: Date.FormatStyle.DateStyle.long, time:
            Date.FormatStyle.TimeStyle.omitted)
        cell.coverImage?.image = UIImage(named: "Quad")

        if(workoot.type == 0){// single time
            cell.Measure?.text = formatter.string(from: (workoot.totalTime ?? 0)!) // prints in MM:SS
                format
            cell.Units?.text = "Minutes"
        }

        else if(workoot.type == 1){ // Single Distance
            cell.Measure?.text = String(workoot.meters!)
            cell.Units?.text = "Meters"
        }

        else if(workoot.type == 2){ // interval Time
            let timeString = formatter.string(from: (workoot.totalTime)!)
            let restString = formatter.string(from: (workoot.rest)!)
            cell.Measure?.text = timeString! + " minutes"
            cell.Units?.text = restString! + " minutes"
        }

        else{ // interval Distance
            let restyString = formatter.string(from: (workoot.rest)!)
            cell.Measure?.text = String(workoot.meters!) + " meters"
            cell.Units?.text = restyString! + " minutes"
        }

        return cell
    }
```

15. Lastly, we need to add an import into our AppDelegate File. import WatchConnectivity.

Watch App

Setting up the data model

1. This app is meant to track workouts, so the first thing that must be done is creating a struct called PracticePlan that will keep track of the workout metrics that we need to track
2. Create a struct in iRow WatchKit Extension called PracticePlan, add to it the following variables:

```
static let SINGLE_TIME: Int = 0
static let SINGLE_DISTANCE: Int = 1
static let TIME_INTERVAL: Int = 2
static let DISTANCE_INTERVAL: Int = 3

let type: Int
var pieceTotalSeconds: TimeInterval?
var pieceDistanceMeters: Int?
var restTotalSeconds: TimeInterval?
```

3. Additionally, add the following init() functions:

```
init(isInterval: Bool, isDistance: Bool) {
    if isInterval == false && isDistance == false   { type = PracticePlan.SINGLE_TIME }
    else if isInterval == false && isDistance == true    { type = PracticePlan.SINGLE_DISTANCE }
    else if isInterval == true && isDistance == false    { type = PracticePlan.TIME_INTERVAL }
```

```
        else if isInterval == true && isDistance == true     { type = PracticePlan.DISTANCE_INTERVAL}
        else    { type = -1 }
        pieceTotalSeconds = 0
        pieceDistanceMeters = 0
        restTotalSeconds = 0
    }

    init(type: Int, pieceSeconds: TimeInterval, pieceMeters: Int, restSeconds: TimeInterval){
        self.type = type
        if type == 1 || type == 3 {
            pieceDistanceMeters = pieceMeters
            pieceTotalSeconds = 0
        }
        else {
            pieceDistanceMeters = 0
            pieceTotalSeconds = pieceSeconds
        }
        if type > 1 { restTotalSeconds = 0 } else { restTotalSeconds = restSeconds }
    }
```

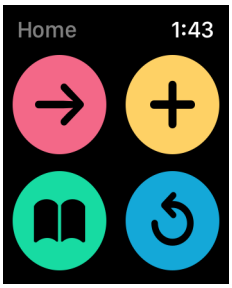
Setting up the HomeController

1. In Interface.storyboard—under the iRow WatchKit App group—add a new Interface Controller and name it HomeController.
2. Add four buttons to this Interface Controller. You will need to embed the top and bottom rows in separate horizontal groups, then embed both groups into a single vertical group. In the element inspector on the right of the screen, adjust the size of each group:
    a. Adjust the width of each horizontal group to be 100% of the width of the screen
    b. Adjust the height to be 50% of the height of the screen
    c. Set the height and width of the vertical group to be 100% of the screen
3. Open the Extension Delegate and add the following lines of code to the top (these will be your app's theme colors):

```
let BRIGHT_GREEN = UIColor.init(red: 0.1333, green: 0.8431, blue: 0.5647, alpha: 1.0)
let SUPER_RED = UIColor.init(red: 0.9294, green: 0.3059, blue: 0.4588, alpha: 1.0)
let PERFECT_YELLOW = UIColor.init(red: 0.9922, green: 0.7882, blue: 0.3255, alpha: 1.0)
let BEST_BLUE = UIColor.init(red: 0.098, green: 0.5922, blue: 0.8118, alpha: 1.0)
```

4. Back in the storyboard editor, open the assistant view.
5. Create outlets for the four buttons we just created and set the names to the following:
    a. Top left button – "Just Row"
    b. Top right button – "New Workout"
    c. Bottom left button – "History"
    d. Bottom right button – "Rerow"
6. Add the icons to these buttons by opening the component library in the interface builder, selecting the image library and setting the background image of each button to the following:
    a. Just Row – arrow.forward.circle.fill

    b. New Workout – plus.circle.fill
    c. History – book.circle.fill
    d. Rerow – arrow.counterclockwise.circle.fill
7. Finally, add our theme colors to the buttons, so the end product will look like this:



Setting up the NewWorkout menu and options:

1. Add a new file under iRow WatchKit Extension called “WOSelectController.swift” and set it to extend WKInterfaceController
2. Add a new InterfaceController in the interface builder and set the custom class to WOSelectController
3. Drag in from the component library a new WKInterfaceTable, set the label to “New Workout Row Controller” and check the box labeled “Selectable”
4. Create a new class in the iRow WatchKit Extension group called “NewWORowController.swift” and set it to extend NSObject
5. Set the custom class of New Workout Row Controller to NewWORowController
6. Drag a label component into the table, set the label to “Type Label”
7. Create an outlet for this label in NewWORowController.swift, this will be the only line of code in this class
8. In WOSelectController:
    a. Create an outlet for the table
    b. Add the follow constant array:

```
let types = ["Single Time", "Single Distance", "Time Intervals", "Distance Intervals"]
```

    c. Add the following function to populate the table:

```
func loadTable(){
    workoutTypeTable.setNumberOfRows(types.count, withRowType: "New Workout Row Controller")

    for (index, labelText) in types.enumerated(){
        let row = workoutTypeTable.rowController(at: index) as! NewWORowController
        row.typeLabel?.setText(labelText)
    }
}
```

    d. Call loadTable() in the awake(withContext:) function
    e. Add the following function to tell the table what to do when an element is pressed:

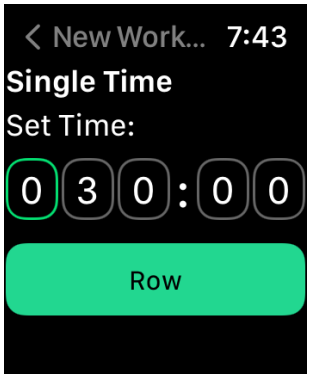
```
override func table(_ table: WKInterfaceTable, didSelectRowAt rowIndex: Int) {

    print(rowIndex)

    var workout: PracticePlan

    if rowIndex == 0 {
        // Single Time
        workout = PracticePlan(isInterval: false, isDistance: false)
        pushController(withName: "Configure Workout", context: workout)
    }
    else if rowIndex == 1 {
        // Single Distance
        workout = PracticePlan(isInterval: false, isDistance: true)
        pushController(withName: "Configure Workout", context: workout)
    }
    else if rowIndex == 2 {
        // Time Intervals
        workout = PracticePlan(isInterval: true, isDistance: false)
        pushController(withName: "Configure Workout", context: workout)
    }
    else {
        // Distance Intervals
        workout = PracticePlan(isInterval: true, isDistance: true)
        pushController(withName: "Configure Workout", context: workout)
    }

}
```

Configuring the Selected Workout

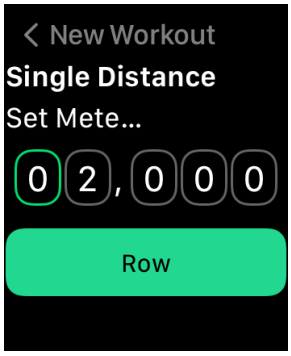
1. Add a new Interface controller to the storyboard and create the corresponding custom class called “WOConfigureController.swift” in the iRow WatchKit Extension group. Make sure to set the controller’s custom class to this class in the storyboard editor.
2. In WOConfigureController, add the following lines of code—these will be some of the workout metrics that are passed on to the iOS app:

```
var NewWorkout: PracticePlan?
var totalTimeInSeconds: Int?
var totalDistanceInMeters: Int?
```

3. In the storyboard editor, add the following components:
    a. Label – Set the text to “Workout Title”, font “System semibold 18.0”
    b. Label – Set the text to “Set Work:”, font “System 18.0”
    c. 5 Pickers (for time), one Label – Embed these in a horizontal group, with the label (whose text shall be set to “:”) in between the third and fourth pickers from the left edge of the screen. Set the width for each picker to be 0.18 and the “:” label to 0.07. Should look like this:



d. 5 Pickers (for meters), one Label – Embed these in a horizontal group, with the label (whose text shall be set to ",") in between the second and third pickers from the left edge of the screen. Set the width for each picker to be 0.17 and the "," label to 0.05. Should look like this:
   i. Set the vertical alignment for both groups of pickers to center
   ii. Set both of these groups to "Hidden"



e. 2 Buttons – One labeled "Row" and one labeled "Set Rest". Set the vertical alignment of both of these to Bottom

4. Create outlets for all of these new components in WOConfigureController.swift, such that

```
@IBOutlet weak var workoutTitle: WKInterfaceLabel!
@IBOutlet weak var setLabel: WKInterfaceLabel!

@IBOutlet weak var TimePickerGroup: WKInterfaceGroup!
@IBOutlet weak var DistPickerGroup: WKInterfaceGroup!

@IBOutlet weak var minutesHundPicker: WKInterfacePicker!
@IBOutlet weak var minutesTensPicker: WKInterfacePicker!
@IBOutlet weak var minutesOnesPicker: WKInterfacePicker!
@IBOutlet weak var secondsTensPicker: WKInterfacePicker!
@IBOutlet weak var secondsOnesPicker: WKInterfacePicker!

@IBOutlet weak var distTenThousPicker: WKInterfacePicker!
@IBOutlet weak var distOneThousPicker: WKInterfacePicker!
@IBOutlet weak var distHundPicker: WKInterfacePicker!
@IBOutlet weak var distTensPicker: WKInterfacePicker!
@IBOutlet weak var distOnesPicker: WKInterfacePicker!
```

```
@IBOutlet weak var setRestButton: WKInterfaceButton!
@IBOutlet weak var rowButton: WKInterfaceButton!
```

5. To this class, add the following global variables to keep track of user input:

```
var MinutesHundDigit: Int = 0
var MinutesTensDigit: Int = 0
var MinutesOnesDigit: Int = 0
var SecondsTensDigit: Int = 0
var SecondsOnesDigit: Int = 0

var DTenThousDigit: Int = 0
var DOneThousDigit: Int = 0
var DHundDigit: Int = 0
var DTensDigit: Int = 0
var DOnesDigit: Int = 0
```

6. To the awake(withContext:context) function add:

```
NewWorkout = (context as! PracticePlan)
let numItems: [WKPickerItem] = numberList.map{
    let pickerItem = WKPickerItem()
    pickerItem.title = "\($0)"
    return pickerItem
}

let tensItems: [WKPickerItem] = secTens.map{
    let pickerItem = WKPickerItem()
    pickerItem.title = "\($0)"
    return pickerItem
}
configForSelection()
```

7. Configure what is displayed on the screen by doing the following:
    a. Add the following function to WOConfigureController, this sets up the various labels and picker groups to be displayed, based on the user's selection.

```
func configForSelection() {
    // Single Time
    if NewWorkout?.type == PracticePlan.SINGLE_TIME {
        workoutTitle.setText("Single Time")
        setLabel.setText("Set Time:")
    }

    // Single Distance
    if NewWorkout?.type == PracticePlan.SINGLE_DISTANCE {
        workoutTitle.setText("Single Distance")
        setLabel.setText("Set Meters:")

        DistPickerGroup.setHidden(false)
        rowButton.setHidden(false)
    }

    // Intervals Time
    if NewWorkout?.type == PracticePlan.TIME_INTERVAL {
        workoutTitle.setText("Timed Intervals")
        setLabel.setText("Set Time:")

        TimePickerGroup.setHidden(false)
        setRestButton.setHidden(false)
    }

    // Intervals Distance
    if NewWorkout?.type == PracticePlan.DISTANCE_INTERVAL {
        workoutTitle.setText("Distance Intervals")
        setLabel.setText("Set Meters:")

        DistPickerGroup.setHidden(false)
        setRestButton.setHidden(false)
    }
}
```

b. Configure the buttons by adding to awake function:

```
if (NewWorkout?.type)! > 1 {
    setRestButton.setBackgroundColor(PERFECT_YELLOW)
    setRestButton.setHidden(false)
}
if (NewWorkout?.type)! < 2 {
    rowButton.setBackgroundColor(BRIGHT_GREEN)
    rowButton.setHidden(false)
}
```

c. Configure for a distance workout by adding to awake function:

```
if (NewWorkout?.type)! % 2 == 1 {
    DistPickerGroup.setHidden(false)

    // Populating Meter Selection Pickers
    distTenThousPicker.setItems(numItems)
    distOneThousPicker.setItems(numItems)
    distHundPicker.setItems(numItems)
    distTensPicker.setItems(numItems)
```

```
    distOnesPicker.setItems(numItems)


    // Setting a Default Selected Distance (2000 Meters)
    // Selecting Default Values for Meter Pickers
    distTenThousPicker.setSelectedItemIndex(0)
    distOneThousPicker.setSelectedItemIndex(2)
    distHundPicker.setSelectedItemIndex(0)
    distTensPicker.setSelectedItemIndex(0)
    distOnesPicker.setSelectedItemIndex(0)

    // Assigning Default Values to Meter Selection Variables
    DTenThousDigit = 0
    DOneThousDigit = 2
    DHundDigit = 0
    DTensDigit = 0
    DOnesDigit = 0

    configureDist()
}
```

And adding the following function to the class:

```
// Configures and displays distance for one interval or single distance
func configureDist(){
    totalDistanceInMeters = (DTenThousDigit * 10000) + (DOneThousDigit * 1000) +
        (DHundDigit * 100) + (DTensDigit * 10) + DOnesDigit

    NewWorkout!.pieceDistanceMeters = totalDistanceInMeters!
    let distometers:Int = NewWorkout!.pieceDistanceMeters!

    workSetLabel.setText(String(format: "%d", distometers))
}
```

Finally add the following IBActions to the class

```
// MARK: - Meter Picker IBActions
@IBAction func distTenThousChanged(_ value: Int) {
    DTenThousDigit = value
    configureDist()
}

@IBAction func distOneThousChanged(_ value: Int) {
    DOneThousDigit = value
    configureDist()
}

@IBAction func distHundChanged(_ value: Int) {
    DHundDigit = value
    configureDist()
}

@IBAction func distTensChanged(_ value: Int) {
    DTensDigit = value
    configureDist()
}

@IBAction func distOnesChanged(_ value: Int) {
    DOnesDigit = value
    configureDist()
}


@IBAction func setRestButtonPushed() {
    pushController(withName: "Set Rest", context: NewWorkout)
}
```

With each IBAction corresponding to one of the pickers

d. Similarly, add the following to the awake function to configure for a time selection:

```
if (NewWorkout?.type)! % 2 == 0 {
        TimePickerGroup.setHidden(false)

        // Populating Time Selection Pickers
        minutesHundPicker.setItems(numItems)
        minutesTensPicker.setItems(numItems)
        minutesOnesPicker.setItems(numItems)
        secondsTensPicker.setItems(tensItems)
        secondsOnesPicker.setItems(numItems)

        // Setting a Defualt Selected Time (30 Minutes)
        // Selecting Default Values for Time Pickers
        minutesHundPicker.setSelectedItemIndex(0)
        minutesTensPicker.setSelectedItemIndex(3)
        minutesOnesPicker.setSelectedItemIndex(0)
        secondsTensPicker.setSelectedItemIndex(0)
        secondsOnesPicker.setSelectedItemIndex(0)
```

```
    // Assigning Default Values to Time Selection Variables
    MinutesHundDigit = 0
    MinutesTensDigit = 3
    MinutesOnesDigit = 0
    SecondsTensDigit = 0
    SecondsOnesDigit = 0

    configureTime()
 }
```

And add the following function to the class:

```
func configureTime(){
    minutes = (MinutesHundDigit * 100) + (MinutesTensDigit * 10) + MinutesOnesDigit
    seconds = (SecondsTensDigit * 10) + SecondsOnesDigit
    totalTimeInSeconds = (minutes! * 60) + seconds!

    print(totalTimeInSeconds!)

    NewWorkout?.pieceTotalSeconds! = Double(totalTimeInSeconds!) as TimeInterval

}
```

Finally, add the following IBActions to the class

```
// MARK: - Time Picker IBActions
@IBAction func minHundrChanged(_ value: Int) {
    MinutesHundDigit = value
    configureTime()
}

@IBAction func minTensChanged(_ value: Int) {
    MinutesTensDigit = value
    configureTime()
}

@IBAction func minOnesChanged(_ value: Int) {
    MinutesOnesDigit = value
    configureTime()
}

@IBAction func secTensChanged(_ value: Int) {
    SecondsTensDigit = value
    configureTime()
}

@IBAction func secOnesChanged(_ value: Int) {
    SecondsOnesDigit = value
    configureTime()
}
```

8. At this point, you need to add to your storyboard two more Interface Controllers:
    a. SetRestController – to set the rest time between interval workouts

b. WOInterfaceController – the interface to be displayed while the watch is tracking a workout

Make sure to set these custom classes to their corresponding interface scenes.

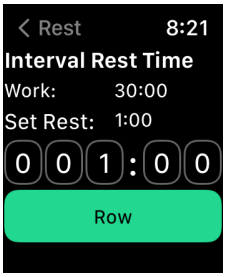
9. Set the context for a segue to WOInterfaceController with the following override:

```
override func contextsForSegue(withIdentifier segueIdentifier: String) -> [Any]? {
    if segueIdentifier == "Config to WO"{
       return [NewWorkout!, NewWorkout!]
    }

    return nil
}
```

Setting up SetRestController

1. Add the following components to the SetRestController:
    a. Label – For the title of the workout
    b. 2 Labels embedded into a horizontal group – to display the distance/time of each interval, set the first label's text to "Work:"
    c. Label – Set the text of this label to "Set rest:"
    d. 5 Pickers, 1 Label – follow the same procedure in adding this group of pickers as you did when adding the pickers for time in the previous scene
    e. 1 Button – Set the text of this button to "Row"

    Final product should look like this



2. Configure this controller the exact way you configured the previous controller for a time workout (omit configurations for any distance workout), with the exception of the contextsForSegue override, set that function to the following:

```
override func contextsForSegue(withIdentifier segueIdentifier: String) -> [Any]? {
    return [workout!, workout!]
}
```

Setting up the Workout display interface

1. Add a segue from both of the "Row" buttons from WOConfigureController and SetRestController with identifiers "Config to WO" and "Rest to WO" respectively. Make sure both of these segues are set to present modally.
2. In the interface builder, add the following components:
    a. Label – for the workout title
    b. Label – for the meters remaining, if applicable
    c. Timer – for the time remaining, if applicable
    d. Separator – Set vertical alignment to center
    e. Label – for the stroke rate, set vertical alignment to bottom

3. Create IBOutlets for the labels and the timer in the WOInterfaceController class
4. Make sure that the timer only displays minutes and seconds
5. In the WOInterfaceController class, add a global variable to keep track of the workout being tracked

```
var CurrentWorkout : PracticePlan?
```

6. Set this workout to the context of the interface controller in the awake function

```
CurrentWorkout = context as! PracticePlan?
```

7. Add the following in the following to the awake function to set up the display for the correct workout:

```
        var seconds: Double = -2
        var dist: Int = -2

        super.awake(withContext: context)

        switch(CurrentWorkout?.type){
        case PracticePlan.SINGLE_TIME:
            workoutTitle.setText("Single Time")
            seconds = (CurrentWorkout?.pieceTotalSeconds)!
            setUpTimedWorkout(seconds: seconds)
            break
        case PracticePlan.SINGLE_DISTANCE:
            workoutTitle.setText("Single Distance")
            dist = (CurrentWorkout?.pieceDistanceMeters)!
            setUpDistanceWorkout(distance: dist)
            break
        case PracticePlan.TIME_INTERVAL:
            workoutTitle.setText("Time Interval")
            seconds = (CurrentWorkout?.pieceTotalSeconds)!
            setUpTimedWorkout(seconds: seconds)
            break
        case PracticePlan.DISTANCE_INTERVAL:
            workoutTitle.setText("Distance Interval")
            dist = (CurrentWorkout?.pieceDistanceMeters)!
            setUpDistanceWorkout(distance: dist)
            break
        default:
            break
        }
```

8. Finally, add the following functions to the class to set up the correct workout:

```
func setUpTimedWorkout(seconds: Double){
    meters.setHidden(true)
    timer.setDate(NSDate(timeIntervalSinceNow: seconds) as Date)
    timer.start()
}

func setUpDistanceWorkout(distance: Int){
    timer.setHidden(true)
    meters.setText(String(format: "%d m", distance))
}
```

Setting up the workout controls page

1. Create a new Interface controller called WOControlsController. Ctrl+click on WOInterfaceController and drag to this controller. Set the relationship to “next page”
2. This page will appear as a “second page” to our workout display and will display a “reset” button and a “stop” button
3. Add outlets to these buttons
4. Add the following global variables:

```
var workout : PracticePlan?
var SentWOData: [String:Any?] = ["Type": nil,
                                 "Work Seconds": nil,
                                 "Distance": nil,
                                 "Rest Seconds": nil]
```

5. Set this class to extend WCSessionDelegate
6. In the awake function add the following code:

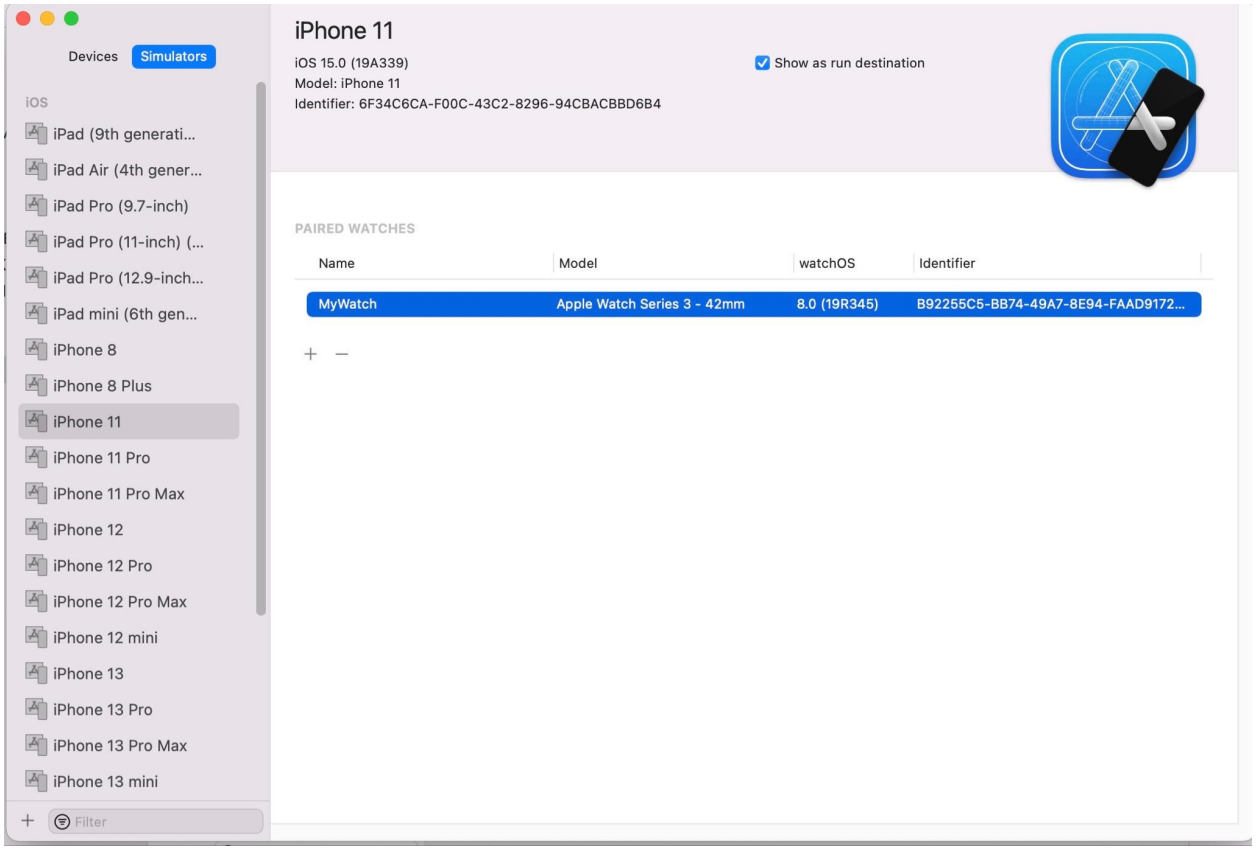
```
session.delegate = self
session.activate()
```

7. Create an IBAction for each of the buttons
    a. The reset button shall call the dismiss() function
    b. The stop button shall send the collected data to the iOS app. Add the following to that action:

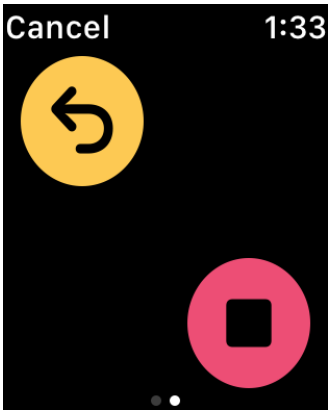
```
SentWOData = ["Type": workout!.type,
              "Work Seconds": workout!.pieceTotalSeconds,
              "Distance": workout!.pieceDistanceMeters,
              "Rest Seconds": workout!.restTotalSeconds]

    WCSession.default.transferUserInfo(SentWOData as [String : Any])
    dismiss()
```

Running The App

To Run the app from XCode, we need two simulators. We can start by adding an additional simulator so that the devices can be paired together when ran. Click on the simulator being ran at the top of the screen then scroll down from the list of available devices to “Add Additional Simulator”



1. Run the simulator for iRow, then click where it says “iRow > iPhone 11” on “iRow” and choose “iRow WatchKit App” from the choices
2. Make sure the simulator is the one you named for the additional simulator (in my case, it was “MyWatch”
3. Next go into iRow WatchKit Extensions info.plist file and scroll down until you see App can run independently of companion iphone app, make sure this says NO for the boolean as it can run into some problems with the WatchConnectivity (NOTE: This is an actual bug in XCode, Swift 5 that has been reported a few times already, until apple fixes this, it cannot send data, believe us, we tried)
4. On the simulation Watch, select the + button, This is for making a new workout.
5. Choose your workout and enter in the time/distance you would like to do.
6. Once finished with your workout, scroll right and click the red stop button. This will send the data to the phone



7. View on your phone simulator

Conclusions

1. Summary of your tutorial, including any other alternative approaches developers might pursue to implement the same functionality (e.g. third party components that can be used in lieu of the native features you have studied) related platform features the reader might want to refer to for further study, as well as a link to a github repo of your complete source code.

In summary, the app can be used to send the data from the user's watch to the user's iPhone. WatchConnectivity must be used as well as WatchKit but other developers may be able to implement HealthKit and Firebase for a larger platform of people to enjoy their workouts.
The GitHub example is here (on master branch)