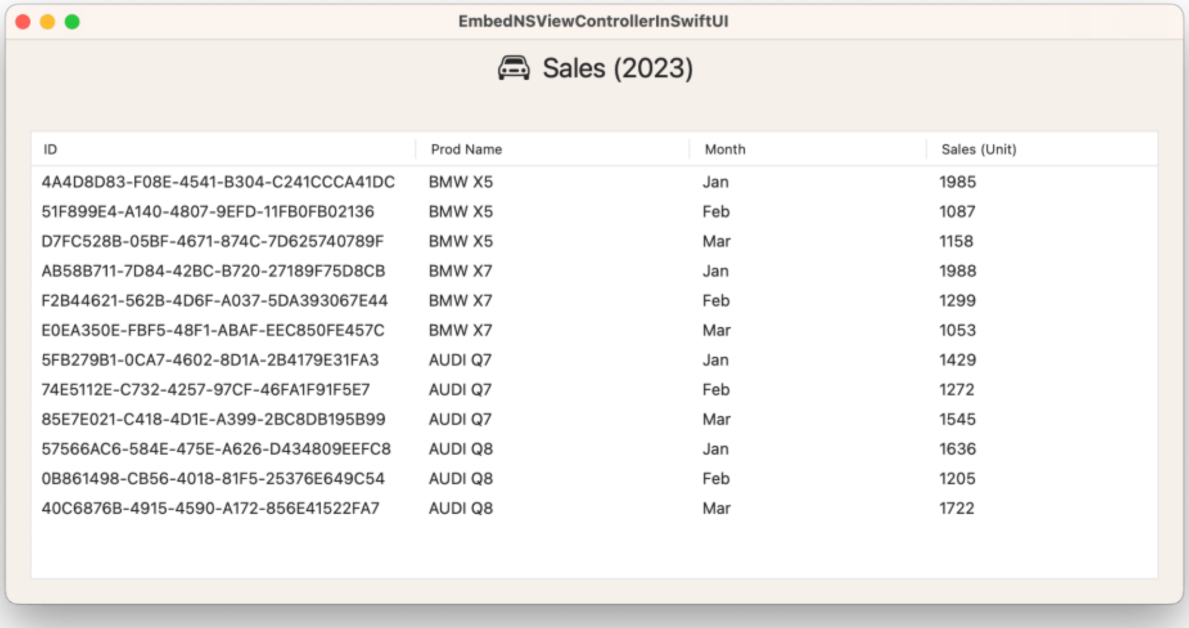
NSViewController in SwiftUI

ddas.tech/nsviewcontroller-in-swiftui/

July 28, 2023

In this post we will embed an NSViewController in SwiftUI using **NSViewControllerRepresentable**.



Sales (2023)

ID	Prod Name	Month	Sales (Unit)
4A4D8D83-F08E-4541-B304-C241CCCA41DC	BMW X5	Jan	1985
51F899E4-A140-4807-9EFD-11FB0FB02136	BMW X5	Feb	1087
D7FC528B-05BF-4671-874C-7D625740789F	BMW X5	Mar	1158
AB58B711-7D84-42BC-B720-27189F75D8CB	BMW X7	Jan	1988
F2B44621-562B-4D6F-A037-5DA393067E44	BMW X7	Feb	1299
E0EA350E-FBF5-48F1-ABAF-EEC850FE457C	BMW X7	Mar	1053
5FB279B1-0CA7-4602-8D1A-2B4179E31FA3	AUDI Q7	Jan	1429
74E5112E-C732-4257-97CF-46FA1F91F5E7	AUDI Q7	Feb	1272
85E7E021-C418-4D1E-A399-2BC8DB195B99	AUDI Q7	Mar	1545
57566AC6-584E-475E-A626-D434809EEFC8	AUDI Q8	Jan	1636
0B861498-CB56-4018-81F5-25376E649C54	AUDI Q8	Feb	1205
40C6876B-4915-4590-A172-856E41522FA7	AUDI Q8	Mar	1722

`NSViewControllerRepresentable` is a protocol in SwiftUI that allows us to integrate macOS AppKit's **`NSViewController`** with SwiftUI's view hierarchy. It is part of the "bridging" functionality provided by SwiftUI to incorporate existing AppKit (macOS) or UIKit (iOS) views and view controllers into SwiftUI-based applications.

When building a SwiftUI app that requires a more complex or specialized view we can use `NSViewControllerRepresentable` to wrap an existing `NSViewController` subclass and use it as a SwiftUI view.

Here's a brief overview of how it works:

1. Create an `NSViewController` subclass: Start by creating a custom `NSViewController` class that encapsulates the behavior and UI we want to display in your SwiftUI app.

2. Adopt **`NSViewControllerRepresentable`**: Make our `NSViewController` subclass conform to the `NSViewControllerRepresentable` protocol. This protocol has two associated types: `NSViewControllerType` and `Context`. we will need to specify the actual `NSViewController` type and a context type that conforms to the `NSViewControllerRepresentableContext` protocol.
3. Implement the required methods: `NSViewControllerRepresentable` requires you to implement two methods:
   - **`makeNSViewController(context:)`**: This method should create and return an instance of your custom `NSViewController`.
   - **`updateNSViewController(_:context:)`**: In this method, we update the `NSViewController` with the latest SwiftUI configuration and data.
4. Use the `NSViewControllerRepresentable` in SwiftUI: Once we have created the `NSViewControllerRepresentable`, we can use it as a SwiftUI view in your app, just like any other SwiftUI view.

Here's an example of how to use `NSViewControllerRepresentable`:

Let's start by creating a TableViewController which will be the subclass of NSViewController. The view controller will have a NSTableView instance and few delegate and datasource methods.

We have also added a Delegate protocol to handle Table View Delegate methods from the SwiftUI Class.

```
//  TableViewController.swift
//  EmbedNSViewControllerInSwiftUI
//  Created by Debasis Das on 7/27/23.

import Cocoa

class TableViewController: NSViewController, NSTableViewDelegate,
NSTableViewDataSource {

    @IBOutlet weak var tableView: NSTableView!
    @objc dynamic var tableContents: [ProductSalesRecord] = []
    weak var additionalDelegate: CustomTableViewControllerDelegate?

    override func viewDidLoad() {
        super.viewDidLoad()
        self.tableView.dataSource = self
        self.tableView.delegate = self
    }

    func reloadUI(){
        self.tableView.reloadData()
    }

    func numberOfRows(in tableView: NSTableView) -> Int {
        return self.tableContents.count
    }

    func tableView(_ tableView: NSTableView, viewFor tableColumn: NSTableColumn?,
row: Int) -> NSView? {
        let record = self.tableContents[row]
        var result:NSTableCellView
        result  = tableView.makeView(withIdentifier: (tableColumn?.identifier)!,
owner: self) as! NSTableCellView
        switch tableColumn?.identifier.rawValue {
        case "rowId":
            result.textField?.stringValue = "\(record.id)"
        case "prodName":
            result.textField?.stringValue = record.prodName
        case "month":
            result.textField?.stringValue = record.month
        case "salesUnit":
            result.textField?.stringValue = "\(record.unitSales)"
        default:
            result.textField?.stringValue = "Default Val"
        }
        return result
    }

    func tableViewSelectionDidChange(_ notification: Notification) {
        guard let tableView = notification.object as? NSTableView else {return}
        let selectionRecord = self.tableContents[tableView.selectedRow]
        self.additionalDelegate?.tableViewSelectionChanged(selectedRecord:
```

```
selectionRecord)
    }
}

protocol CustomTableViewControllerDelegate: AnyObject{
    func tableViewSelectionChanged(selectedRecord: ProductSalesRecord)
}
```

Next we will create the model class for each record in the NSTableView

```
class ProductSalesRecord: NSObject, Identifiable{
    var prodName: String
    var month: String
    var unitSales: Int
    var id =  UUID()

    init(prodName: String, month: String, unitSales: Int) {
        self.prodName = prodName
        self.month = month
        self.unitSales = unitSales
    }
}
```

Next we implement the protocols of the NSViewControllerRepresentable

- **makeNSViewController**
- **updateNSViewController**

```
struct TableVC: NSViewControllerRepresentable {

    @Binding var items: [ProductSalesRecord]
    @Binding var rowSelected: Int
    @Binding var selectedProdName: String

    func makeNSViewController(context: Context) -> NSViewController {
        let vc = TableViewController()
        return vc
    }

    func updateNSViewController(_ nsViewController: NSViewController, context:
Context) {
        guard let vc = nsViewController as? TableViewController else {return}
        vc.tableContents = items
        vc.additionalDelegate = context.coordinator
        vc.reloadUI()
    }

    class Coordinator: NSObject, CustomTableViewControllerDelegate {
        func tableViewSelectionChanged(selectedRecord: ProductSalesRecord) {
            print("tableViewSelectionChanged")
            print(selectedRecord.id)
        }

        var parent: TableVC
        init(_ parent: TableVC) {
            self.parent = parent
        }
    }

    func makeCoordinator() -> Coordinator {
        return Coordinator(self)
    }
}
```

Finally we create the SwiftUI ContentView and create some sample data for ProductSalesRecord and pass it on to the TableViewController, **NSViewControllerRepresentable** Instance

```
struct ContentView: View {
    //@State private var items = [ProductSalesRecord]()
    @State var items: [ProductSalesRecord] = {
        var data:[ProductSalesRecord] = []
        let prodNames = ["BMW X5","BMW X7","AUDI Q7","AUDI Q8"]
        let upperBound = 2000
        let lowerBound = 1000
        for prodName in prodNames {
            for month in ["Jan","Feb","Mar"]{
                let rec = ProductSalesRecord(prodName: prodName, month: month,
unitSales: Int(arc4random_uniform(UInt32(upperBound - lowerBound))) + lowerBound)
                data.append(rec)
            }
        }
        return data
    }()
    @State private var rowSelected = -1
    @State private var selectedProdName = ""
    var body: some View {
        VStack {
            Label("Sales (2023)", systemImage: "car").font(.title).padding([.top,
.bottom], 10)
            TableVC(items: $items, rowSelected: $rowSelected, selectedProdName:
$selectedProdName)
        }

    }
}
```

Download the Sample Code – EmbedNSViewControllerInSwiftUIDownload

You can also read the post where we have added a NSTableView to SwiftUI using NSViewRepresentable

> NSTableView in SwiftUI Sample Code