```
import UIKit
import ConnectSDK
class ConnectingDeviceViewController: UIViewController {

    @IBOutlet weak var tblData: UITableView?
    @IBOutlet weak var heightConstantTblData: NSLayoutConstraint?
    @IBOutlet weak var activityIndicator: UIActivityIndicatorView?

    var discoveryManager = DiscoveryManager.shared()
    private var arrDevices: [ConnectableDevice] = []
    private var connectedDevice: ConnectableDevice?

    override func viewDidLoad() {
        super.viewDidLoad()
        InitConfig()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        self.configureNavigationBar()
        addTableviewOberver()
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
        NotificationCenter.default.removeObserver(self)
        removeTableviewObserver()
        discoveryManager?.stopDiscovery()
    }

    override func viewDidLayoutSubviews() {
        super.viewDidLayoutSubviews()
    }
}
extension ConnectingDeviceViewController {
    private func InitConfig() {

        let videoCapabilities = [
            kMediaPlayerPlayVideo,
            kMediaControlAny,
            kVolumeControlVolumeUpDown
        ]
        let imageCapabilities = [kMediaPlayerDisplayImage]
        let videoFilter = CapabilityFilter(capabilities: videoCapabilities)
        let imageFilter = CapabilityFilter(capabilities: imageCapabilities)
        DiscoveryManager.shared().capabilityFilters = [videoFilter!, imageFilter!]
        DiscoveryManager.shared().delegate = self
```

```
        DiscoveryManager.shared().registerDeviceService(AirPlayService.self, withDiscovery:
ZeroConfDiscoveryProvider.self)
        DiscoveryManager.shared().registerDeviceService(CastService.self, withDiscovery:
CastDiscoveryProvider.self)
        DiscoveryManager.shared().registerDeviceService(DIALService.self, withDiscovery:
SSDPDiscoveryProvider.self)
        DiscoveryManager.shared().registerDeviceService(RokuService.self, withDiscovery:
SSDPDiscoveryProvider.self)
        DiscoveryManager.shared().registerDeviceService(DLNAService.self, withDiscovery:
SSDPDiscoveryProvider.self)
        DiscoveryManager.shared().registerDeviceService(WebOSTVService.self,
withDiscovery: SSDPDiscoveryProvider.self)

        DiscoveryManager.shared().startDiscovery()
    }
}
extension ConnectingDeviceViewController: DiscoveryManagerDelegate {
    func discoveryManager(_ manager: DiscoveryManager!, didFind device:
ConnectableDevice!) {
        // A ConnectableDevice was found
        print(device.friendlyName ?? "")
        setDeviceInArray(device: device)
    }

    func discoveryManager(_ manager: DiscoveryManager!, didLose device:
ConnectableDevice!) {
        // A ConnectableDevice was lost
        print(device.friendlyName ?? "")
        self.arrDevices.removeAll(where: {$0.id == device.id})
        DispatchQueue.main.async {
            self.tblData?.reloadData()
        }
    }
    func discoveryManagerDidFailWithError(_ error: Error!) {
        print("Discovery error: \(error.localizedDescription)")
    }

    func discoveryManager(_ manager: DiscoveryManager!, didUpdate device:
ConnectableDevice!) {
       // print(device.friendlyName ?? "")
    }

    private func setDeviceInArray(device: ConnectableDevice) {
        if !arrDevices.contains(where: {$0.id == device.id}) {
            self.arrDevices.append(device)
            self.tblData?.reloadData()
        }
    }
```

```
}
//MARK: - UITableView Delegate & DataSource
extension ConnectingDeviceViewController : UITableViewDataSource, UITableViewDelegate
{

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return self.arrDevices.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
        let cell = tableView.dequeueReusableCell(for: indexPath, with: ConnectingTVCell.self)
        if !self.arrDevices.isEmpty {
            let obj = self.arrDevices[indexPath.row]
            cell.lblHeader?.text = obj.lastKnownIPAddress ?? ""
            cell.lblSubHeader?.text = obj.friendlyName ?? ""
        }
        return cell
    }
    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        DispatchQueue.main.async {
            if !self.arrDevices.isEmpty {
                let obj = self.arrDevices[indexPath.row]

                self.connectedDevice = obj
                self.connectedDevice?.delegate = self
                self.connectedDevice?.setPairingType(DeviceServicePairingTypePinCode)
                self.connectedDevice?.connect()
            }
        }
    }
}
extension ConnectingDeviceViewController: DeviceServiceDelegate {
    func deviceServicePairingSuccess(_ service: DeviceService!) {
        print("success")
    }
    func deviceService(_ service: DeviceService!, pairingFailedWithError error: Error!) {
        print(error)
    }

    func deviceServiceConnectionSuccess(_ service: DeviceService!) {
        print(service.serviceName)
    }

    func deviceService(_ service: DeviceService!, didFailConnectWithError error: Error!) {
        print(error)
    }
```

```
    func deviceService(_ service: DeviceService!, pairingRequiredOf pairingType:
DeviceServicePairingType, withData pairingData: Any!) {
        print(service.serviceName)
        print(service.pairingType)
        present((pairingData as? UIAlertController)!, animated: true)
    }
    func deviceServiceConnectionRequired(_ service: DeviceService!) {
        print(service.serviceName)
    }
    func deviceService(_ service: DeviceService!, disconnectedWithError error: Error!) {
        print(error)
    }
}
extension ConnectingDeviceViewController: ConnectableDeviceDelegate {

    func connectableDeviceReady(_ device: ConnectableDevice!) {
        print(device.connected)
    }

    func connectableDeviceDisconnected(_ device: ConnectableDevice!, withError error:
Error!) {
        print(error)
    }

    func connectableDevicePairingSuccess(_ device: ConnectableDevice!, service:
DeviceService!) {
        print(device.friendlyName)
    }

    func connectableDevice(_ device: ConnectableDevice!, service: DeviceService!,
pairingRequiredOfType pairingType: Int32, withData pairingData: Any!) {
        print(service.serviceName)
        print(service.pairingType.rawValue)
        present((pairingData as? UIAlertController)!, animated: true)

        //if pairingType == DeviceServicePairingTypePinCode.rawValue {
            // Present a UI for the user to input the PIN code
            let alertController = UIAlertController(title: "Enter PIN Code", message: "Please enter
the PIN code displayed on your TV", preferredStyle: .alert)

            alertController.addTextField { textField in
                textField.placeholder = "PIN Code"
                textField.keyboardType = .numberPad
            }

            alertController.addAction(UIAlertAction(title: "Connect", style: .default, handler: { _ in
                // Get the PIN code input from the text field
                if let pinCode = alertController.textFields?.first?.text {
```

```
                // Provide the PIN code to the ConnectableDevice
                //self.connectableDevice.sendPairingKey(pinCode)
                //device.pairWithPairingData(["pin": pinCode])
                //service.pair(withData: <#T##Any!#>)
                //for service in services {
                    // Check if the service supports PIN code pairing
                    //if service.pairingType == .pinCode {
                        // Initiate pairing with the PIN code
                    service.pair(withData: ["pin": pinCode])
                        // break // Exit loop after initiating pairing
                  //  }
                // }

            }
        }))

        present(alertController, animated: true, completion: nil)
    // }
  }
}
```