```
import UIKit
import ArcGIS

class MapViewController: UIViewController {

private var allLayers: [AGSLayer] = []

    private var removeLayers: [AGSLayer] {
        if let addLayers = mapView?.map?.operationalLayers as?
[AGSLayer] {
            return allLayers.filter { !addLayers.contains($0) }
        }
        return []
    }


    var map: AGSMap!


 override func viewDidLoad() {
        super.viewDidLoad()

        self.map = AGSMap(basemap: basemapType.getBasemap())

        let municipalityLayer =
AGSArcGISMapImageLayer(url: .Municipality_AT!)
        let zoneLayer = AGSArcGISMapImageLayer(url: .Zone_AT!)
        let districtLayer =
AGSArcGISMapImageLayer(url: .District_AT!)
        let cityLayer = AGSArcGISMapImageLayer(url: .City_AT!)
        let parcelLayer = AGSArcGISMapImageLayer(url: .Parcel_AT!)
        let blockLayer = AGSArcGISMapImageLayer(url: .Block_AT!)
        let landmarkLayer =
AGSArcGISMapImageLayer(url: .LandmarkAT)


}

}
```

```
import UIKit
import ArcGIS


class MMLLayersViewController: UITableViewController {

    /// The map for which to manage the operational layers.
    weak var map: AGSMap?
```

```
    /// Every layer on the map or that could be added to the map.
    var allLayers: [AGSLayer] = []

    /// The layers present in `allLayers` but not in the map's
`operationalLayers`.
    private var removeLayers: [AGSLayer] {
        if let operationalLayers = map?.operationalLayers as?
[AGSLayer] {
            return allLayers.filter { !
operationalLayers.contains($0) }
        }
        return []
    }

    var delegate: LayerName!

    override func viewDidLoad() {
        super.viewDidLoad()

        // enable the editing UI
        tableView.isEditing = true

    }

    /// A convenience type for the table view sections.
    private enum Section: CaseIterable {
        case add, remove

        var label: String {
            switch self {
            case .add:
                return "Remove Layers"
            case .remove:
                return "Add Layers"
            }
        }
    }

    // MARK: - UITableViewDataSource

    override func numberOfSections(in tableView: UITableView) ->
Int {
        return Section.allCases.count
    }

    override func tableView(_ tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
        switch Section.allCases[section] {
        case .add:
            return map?.operationalLayers.count ?? 0
        case .remove:
```

```
            return removeLayers.count
        }
    }

    override func tableView(_ tableView: UITableView,
titleForHeaderInSection section: Int) -> String? {
        return Section.allCases[section].label
    }

    override func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
"LayerCell", for: indexPath)

        let layerForIndexPath: AGSLayer? = {
            switch Section.allCases[indexPath.section] {
            case .add:
                return map?.operationalLayers.object(at:
indexPath.row) as? AGSLayer
            case .remove:
                return removeLayers[indexPath.row]
            }
        }()

        let layerName = layerForIndexPath?.name

        if layerName == "MunicipalityE" {
            cell.textLabel?.text = "Municipality"
        }
        else if layerName == "Zones AT" {
            cell.textLabel?.text = "Zone"
        }
        else if layerName == "District AT" {
            cell.textLabel?.text = "District"
        }
        else if layerName == "City AT" {
            cell.textLabel?.text = "City"
        }
        else if layerName == "Block AT" {
            cell.textLabel?.text = "Block"
        }
        else if layerName == "CadastrePlots AT" {
            cell.textLabel?.text = "Parcel"
        }
        else if layerName == "LandmarksAT" {
            cell.textLabel?.text = "Landmark"
        }

       return cell
    }
```

```
    // MARK: - UITableViewDelegate

    override func tableView(_ tableView: UITableView,
editingStyleForRowAt indexPath: IndexPath) ->
UITableViewCell.EditingStyle {
        switch Section.allCases[indexPath.section] {
        case .add:
            return .delete
        case .remove:
            return .insert
        }
    }

    override func tableView(_ tableView: UITableView, canMoveRowAt
indexPath: IndexPath) -> Bool {
        return Section.allCases[indexPath.section] == .add
    }

    override func tableView(_ tableView: UITableView,
targetIndexPathForMoveFromRowAt sourceIndexPath: IndexPath,
toProposedIndexPath proposedDestinationIndexPath: IndexPath) ->
IndexPath {
        if Section.allCases[sourceIndexPath.section] == .add,
            Section.allCases[proposedDestinationIndexPath.section]
== .add {
            // only allow reordering within the operational layers
section
            return proposedDestinationIndexPath
        }
        return sourceIndexPath
    }

    override func tableView(_ tableView: UITableView, moveRowAt
sourceIndexPath: IndexPath, to destinationIndexPath: IndexPath) {
        // update the order of layers in the array

        if destinationIndexPath != sourceIndexPath {
            map?.operationalLayers.exchangeObject(at:
sourceIndexPath.row, withObjectAt: destinationIndexPath.row)
        }
    }

    override func tableView(_ tableView: UITableView, commit
editingStyle: UITableViewCell.EditingStyle, forRowAt indexPath:
IndexPath) {
        switch editingStyle {
        case .delete:
            // move the layer from the operational layers to the
removed layers
            guard let layerToRemove =
map?.operationalLayers.object(at: indexPath.row) as? AGSLayer else
{
```

```
                return
            }
            map?.operationalLayers.removeObject(at: indexPath.row)

            // update the table
            tableView.performBatchUpdates({
                // delete the row
                tableView.deleteRows(at: [indexPath],
with: .automatic)

                let newIndexPath = IndexPath(row:
removeLayers.firstIndex(of: layerToRemove)!, section: 1)
                // insert the new row
                tableView.insertRows(at: [newIndexPath],
with: .fade)
            })
        case .insert:
            // move the layer from the removed layers to the
operational layers
            let layer = removeLayers[indexPath.row]
            map?.operationalLayers.insert(layer, at: 0)

            print("1:", layer.name)

            delegate.LayerNamePass(lyname: layer.name)

            // update the table
            tableView.performBatchUpdates({
                // delete the row
                tableView.deleteRows(at: [indexPath], with: .fade)
                let newIndexPath = IndexPath(row: 0, section: 0)
                // insert the new row
                tableView.insertRows(at: [newIndexPath],
with: .fade)
            })
        case .none:
            break
        @unknown default:
            break
        }
    }

}
```